FITPED | AI

AI Introduction

Jozef Kapusta
Oldřich Trenz
Ján Skalka
Eugenia Smyrnova-Trybulska
Vladimiras Dolgopolovas

www.fitped.eu

2024

Funded by the European Union

Erasmus+ FITPED-AI
Future IT Professionals Education in Artificial Intelligence
(Project 2021-1-SK01-KA220-HED-000032095)

FITPED | AI

AI Introduction

Published on

November 2024

Authors

Jozef Kapusta | Teacher.sk, Slovakia

Oldřich Trenz | Mendel University in Brno, Czech Republic

Ján Skalka | Constantine the Philosopher University in Nitra, Slovakia

Eugenia Smyrnova-Trybulska | University of Silesia in Katowice, Poland

Vladimiras Dolgopolovas | Vilnius University, Lithuania

Reviewers

Piet Kommers | Helix5, Netherland

Peter Švec | Teacher.sk, Slovakia

Vaida Masiulionytė-Dagienė | Vilnius University, Lithuania

Erasmus+ FITPED-AI

Future IT Professionals Education in Artificial Intelligence

Project 2021-1-SK01-KA220-HED-000032095

Funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Slovak Academic Association for International Cooperation. Neither the European Union nor the granting authority can be held responsible for them.

ISBN 978-80-558-2224-2

TABLE OF CONTENTS

Foundations of AI

Chapter 1

1.1 Introduction

🕮 1.1.1

Artificial Intelligence, or AI, is a fascinating technology that enables machines to perform tasks that typically require human intelligence. These tasks range from understanding speech and recognizing images to making decisions and learning from experience. AI is becoming a vital part of our daily lives and is used in many important areas. For example, in healthcare, AI helps doctors analyze medical data to diagnose illnesses more quickly and accurately. In transportation, AI powers self-driving cars, making it possible to travel safely without a human driver.

Have you ever interacted with Siri, Alexa, or Google Assistant? These virtual assistants rely on AI to understand your voice and respond to your questions in a meaningful way. They can set reminders, answer questions, and even tell jokes! AI is also at work when Netflix or YouTube recommends shows, movies, or videos based on what you've already watched. This happens because AI analyzes your viewing history to guess what you might enjoy next. These examples show how AI simplifies and improves our daily routines, often without us even realizing it.

AI isn't magic - it's a tool that humans have created to solve complex problems. It works by analyzing data and recognizing patterns, which allows it to make predictions or decisions in various fields. For instance, in agriculture, AI helps farmers monitor soil conditions and predict the best times to plant or harvest crops, leading to better yields. In finance, AI is used to detect suspicious transactions, helping to prevent fraud. By using AI wisely, people can solve problems faster and more efficiently in industries like education, entertainment, and even environmental conservation.

📝 1.1.2

Which of the following is an example of Artificial Intelligence in everyday life?

- Siri understanding your voice commands.
- Writing a letter by hand.
- Watching a movie in a theater.
- Planting crops manually.

🕮 1.1.3

How does AI work?

Artificial intelligence, or AI, works by using data to learn and make decisions. Think of it like studying for a test: just as you learn by reviewing notes and examples, AI learns by analyzing data. For example, if you show a computer thousands of pictures of dogs, it can "learn" to recognize dogs in new pictures. This process of learning from data is called "machine learning", which is a major part of how AI functions.

It’s important to know that AI doesn’t think or understand like a human does. Instead, it uses special mathematical formulas called algorithms to process and analyze data. These algorithms help AI identify patterns, make predictions, and provide solutions. The more data AI has to work with, the better it can learn and improve. For instance, an AI system that is trained with thousands of medical scans can help doctors detect diseases, such as cancer, more quickly and accurately.

Despite its amazing capabilities, AI is not perfect. If AI is trained with too little data, outdated information, or incorrect examples, it can make mistakes. Sometimes, it might even give answers that don’t make sense, a phenomenon called "hallucination“. This happens when AI predicts something completely wrong or unrelated to the problem it is solving. These limitations show that while AI is powerful, it still relies on humans to guide its development and provide accurate data.

1.1.4

Which of the following are examples of how AI works?

- Learning from data to recognize patterns.
- Using algorithms to analyze information.
- Thinking like a human to solve problems.
- Making decisions without any data.

1.1.5

What does AI need to improve its accuracy?

- A large amount of data.
- Human emotions.
- Random guesses.
- Unlimited computing power.

1.1.6

AI in everyday life

AI is all around us, even if you don’t realize it. Have you ever used Google to search for something? AI is what helps organize and rank the search results to show you the most helpful and relevant websites. If you’ve ever used a language translator like Google Translate or DeepL, AI is working behind the scenes to convert text from one language to another, making it easier for people to communicate across the globe.

AI also powers chatbots, which you might have come across on websites. These chatbots can answer your questions instantly without the need for a human customer service representative. They work by analyzing your messages and providing responses based on programmed patterns and data. Similarly, social media platforms like Instagram, TikTok, or YouTube use AI to suggest posts, videos, or accounts that match your interests, based on your activity and preferences.

One of the most exciting applications of AI is in self-driving cars. These vehicles rely on AI to "see" the road, recognize obstacles, and make decisions in real time to ensure a safe journey. AI analyzes data from cameras, sensors, and maps to understand the environment and navigate the car to its destination. With its ability to solve problems and adapt to different situations, AI is changing the way we interact with technology and shaping the future in ways we could only dream of.

📝 1.1.7

Which of the following are examples of AI in everyday life?

- AI ranking Google search results.
- A chatbot answering customer questions.
- Self-driving cars navigating the road.
- Writing a book by hand.

📝 1.1.8

What does AI do on social media platforms?

- Suggests content based on your activity.
- Deletes unwanted posts automatically.
- Writes posts for influencers.
- Blocks all advertisements.

🕮 1.1.9

AI is already a significant part of our lives, and its role is rapidly expanding. In the future, AI is expected to play an even bigger role in solving some of the world's biggest challenges. For instance, in medicine, AI can help doctors discover new drugs more quickly and accurately by analyzing vast amounts of medical data. In space exploration, AI can assist astronauts in studying planets far beyond our Solar System, helping humanity understand the universe better. These advancements show how AI has the potential to improve many areas of our lives.

AI is not just about machines - it's also about creating new opportunities for people. As AI becomes more integrated into our lives, there will be a growing need for professionals who understand how AI works. Many careers, including those in science, engineering, and even the arts, will benefit from knowledge about AI. For example, artists can use AI to create new types of digital artwork, and engineers can design smarter systems for transportation and energy. By learning about AI, you can prepare yourself for exciting and innovative career opportunities in the future.

However, with the power of AI comes the responsibility to use it wisely and ethically. AI must be developed and used in ways that avoid harm, such as bias or unfair outcomes. This is why it's essential to understand not just the technology behind AI but also the ethical questions it raises. AI should be a tool for making the world a better place, ensuring fairness and equality for everyone. By learning how AI works

and thinking about its impact, you can help shape a future where technology is used responsibly.

📝 1.1.10

Which of the following are future uses of AI mentioned in the text?

- Helping doctors discover new medicines.
- Studying planets beyond our Solar System.
- Writing novels without human input.
- Creating smarter transportation systems.

📝 1.1.11

Why is it important to learn about AI?

- To prepare for new job opportunities.
- To avoid using technology.
- To replace human workers.
- To control all machines.

🕮 1.1.12

How AI learns

The way AI learns is similar to how we learn new things - by studying information. This process is called "training". For example, imagine you are learning to identify different types of animals by looking at thousands of pictures. Similarly, if you give an AI model thousands of pictures of animals, it will analyze the images to recognize patterns, shapes, and features. Over time, the AI learns to identify different types of animals, even in pictures it has never seen before.

Once an AI system has been trained, it uses what it learned to make predictions or decisions. For example, a trained AI system can predict tomorrow’s weather by analyzing patterns in past weather data. It can also recommend movies or TV shows based on your viewing history, or even write an essay by piecing together information it has learned. These systems, designed to learn from data and make decisions, are called "machine learning models".

One specific type of machine learning model is called a "large language model", or LLM. These models, such as GPT, are trained on enormous amounts of text data. Because they have studied so much language information, they can answer questions, write creative stories, and have conversations that sound natural and human-like. This is why AI tools can help you with homework, generate creative ideas, or even explain complex topics in an easy-to-understand way.

📝 1.1.13

Which of the following are examples of what AI can do after it has been trained?

- Predict the weather.
- Identify animals in pictures.
- Write an essay or answer questions.
- Play video games by itself.

📝 1.1.14

What is a "large language model" (LLM)?

- A machine learning model trained on large amounts of text.
- A system that learns from pictures.
- A robot that builds houses.
- A tool that stores all human knowledge.

🕮 1.1.15

Limitations of AI

Even though AI is a powerful tool, it has some important challenges and limitations. One major challenge is that AI needs a large amount of high-quality data to work well. If the data is incomplete, inaccurate, or biased, the AI system may not function as intended. For instance, if an AI is trained with poor-quality data, it might mistakenly identify a dog as a cat, or fail to recognize certain objects altogether. The quality and quantity of data directly affect how accurate and reliable AI predictions or decisions will be.

Another limitation of AI is that it cannot think or understand like a human. Unlike humans, AI doesn't have emotions, creativity, or the ability to understand the context in a nuanced way. This means that AI might misinterpret situations, especially in complex or emotional scenarios, where human understanding is crucial. For example, an AI might give a literal answer to a sarcastic comment, showing it doesn't grasp the deeper meaning.

AI can also reflect biases that exist in the data it was trained on. If the training data includes unfair patterns or prejudices, the AI might make biased or unfair decisions. For instance, an AI used in hiring might favor one group of people over another if its training data is not balanced or representative. This is why it's essential for developers and researchers to carefully design AI systems, test them rigorously, and ensure they work accurately and fairly for everyone. Addressing these challenges is a critical step in making AI a tool that benefits society in a responsible way.

📝 1.1.16

Which of the following are limitations of AI?

- AI requires large amounts of data to work accurately.
- AI might make biased decisions if trained with unfair data.
- AI can think and understand emotions like a human.
- AI can solve every problem without human intervention.

📝 1.1.17

Why is it important to test AI systems for fairness?

- To prevent biased or unfair decisions.
- To ensure the AI understands emotions.
- To make AI work faster.
- To reduce the amount of data needed for training.

1.2 What AI is and what is not

🕮 1.2.1

AI is not always easy to define because the term can be used to describe many different technologies. Almost anything can be labeled as AI, from simple tools like business analytics to advanced systems like neural networks. For example, some programs use manually programmed "if-then" rules to make decisions. Even though these systems don’t involve learning or adapting, they are sometimes considered AI because they mimic intelligent behavior in specific situations.

Interestingly, our understanding of AI has changed a lot over time. Technologies that were once considered advanced AI are now seen as standard tools. For instance, years ago, search algorithms used to find the shortest path between two cities were considered groundbreaking examples of AI. Today, these algorithms are so common that they are taught in basic computer science courses. This evolution shows that as technology becomes more familiar, we often stop calling it AI and view it as standard technology instead.

There’s even a popular joke about AI: it’s defined as “cool things that computers can’t do yet”. This highlights how challenging it is to clearly define AI. The definition keeps shifting as computers become capable of doing more and more tasks that were once thought to require human intelligence. This flexibility in the definition of AI makes it an exciting and ever-changing field to study.

📝 1.2.2

What does the joke about AI suggest?

- AI refers to tasks that computers haven’t mastered yet.
- AI is only used for entertainment.
- AI includes everything computers can do today.
- AI is a field with a fixed definition.

📝 1.2.3

Which of the following are reasons why AI is hard to define?

- AI can include simple tools like "if-then" rules.
- AI technologies become standard over time.
- The definition of AI changes as technology advances.
- AI is only about robots replacing humans.

🕮 1.2.4

Defining what AI is and isn’t can be tricky because the term AI covers a wide variety of technologies. It includes simple systems, like "if-then" rules that follow basic instructions, and complex ones, like machine learning, where computers learn from data to make decisions. AI is also a constantly changing and evolving field. What we consider advanced AI today might become a routine technology tomorrow.

For example, optical character recognition (OCR), a technology that allows computers to read printed text and turn it into digital data, was once thought to be an incredible achievement in AI. Today, OCR is so common in apps and devices that we no longer see it as “true AI”. This demonstrates how the definition of AI shifts as technology improves and becomes more familiar.

AI also overlaps with other areas, such as robotics and data science, which can make it harder to define its boundaries. For instance, consider a program that uses advanced statistics to predict stock prices. Is this AI, or is it simply data analytics? Similarly, if a robot uses sensors to navigate a room, is it robotics, AI, or both? The lines between AI and other technologies are often blurry, making it a challenge to pinpoint exactly what qualifies as AI.

📝 1.2.5

Why is it difficult to define AI?

- AI includes both simple and complex systems.
- AI boundaries overlap with other fields like robotics and data science.
- What is considered AI today might not be in the future.
- AI technologies never evolve or change.

📖 1.2.6

Types of AI

AI can be challenging to define because it comes in different forms, each with unique capabilities. One type is **narrow AI**, also called "weak AI". This kind of AI is designed to do specific tasks, such as recognizing faces in photos, recommending songs on Spotify, or helping virtual assistants like Siri or Alexa understand your voice commands. Narrow AI doesn't "think" or act like a human. Instead, it focuses on one thing and does it very well, often better than humans in that particular task.

On the other hand, there is **strong AI**, which would have general intelligence similar to a person's. Strong AI could solve problems, understand emotions, and adapt to completely new and unfamiliar situations, much like humans can. However, strong AI doesn't exist yet - it's still an idea researchers are working toward in the future. Developing strong AI is a complex challenge, as it would require computers to understand the world the way humans do.

The difference between narrow and strong AI highlights how the term "AI" can mean very different things depending on the context. Most of the AI we use today, such as virtual assistants, recommendation systems, and chatbots, falls into the category of narrow AI. These systems are incredibly useful for specific tasks but are far from the general intelligence that defines strong AI.

📝 1.2.7

What is a key difference between narrow AI and strong AI?

- Strong AI doesn't exist yet, while narrow AI is commonly used.
- Narrow AI can adapt to new situations like a human.
- Strong AI is less intelligent than narrow AI.
- Narrow AI understands emotions like humans do.

📝 1.2.8

Which of the following are examples of narrow AI?

- Recognizing faces in photos.
- A virtual assistant like Siri.
- Recommending songs on Spotify.
- Solving completely new problems without human input.

📖 1.2.9

Another reason why defining AI is tricky is its overlap with other technologies. For example, automation and robotics often use AI, but they are not the same thing. Automation involves machines following a set of pre-programmed steps to complete a task. A robot in a factory might assemble cars without any AI - it simply follows instructions. On the other hand, a self-driving car uses AI to navigate roads, avoid obstacles, and make decisions in real time. The difference lies in AI's ability to "think" and adapt, while automation does not require this capability.

AI is also closely connected with data science, which makes the boundaries between the two fields hard to define. Both involve analyzing large amounts of data, but they have different goals. Data science focuses on discovering patterns and insights from data, while AI takes it a step further by using that data to make predictions or decisions. For example, a data scientist might analyze past weather data to identify trends, while an AI system could use that data to predict tomorrow's weather. This overlap makes it challenging to decide where data science ends and AI begins.

As AI becomes more integrated into everyday technology, the line between AI and traditional programming also becomes blurry. Consider a smartphone's face recognition feature: is it AI, or is it simply a clever program that matches patterns? The answer depends on how the technology is designed, showing how interconnected AI is with other fields. These overlaps make defining AI a complex and fascinating challenge.

📝 1.2.10

Which of the following highlight the overlap between AI and other technologies?

- AI is used to help self-driving cars make decisions.
- Data science analyzes large datasets, like AI does.
- A smartphone's face recognition feature could involve AI.
- Automation always requires AI to function.

📝 1.2.11

How is AI different from automation?

- AI can adapt and make decisions, while automation cannot.
- AI always follows pre-programmed steps.
- Automation analyzes data to make predictions.
- Automation is smarter than AI.

📖 1.2.12

The definition of AI isn't fixed - it changes as technology advances. What seems intelligent and groundbreaking today might become ordinary tomorrow. For example, technologies like Optical character recognition (OCR), which allows computers to read printed text, and search engines, which help us find information online, were once considered cutting-edge AI. Now, they're seen as everyday tools that we use without a second thought.

This shift happens because AI evolves alongside advancements in technology. As more people understand how a technology works, it stops feeling magical or futuristic and is no longer labeled as AI. Instead, it becomes a standard tool in our daily lives. This constant change makes it challenging to define exactly what AI is, since its definition keeps moving forward with each technological breakthrough.

In summary, AI is a broad and ever-changing field. It covers many different technologies, from simple to complex, and often overlaps with other areas like data science and robotics. As technology continues to improve, the distinction between what is and isn't AI will remain blurry. This evolution makes AI an exciting and dynamic area of study and innovation.

1.2.13

Why does the definition of AI change over time?

- AI evolves with advancements in technology.
- Everyday tools like OCR were once considered AI.
- People no longer see common tools as "intelligent".
- AI stops being useful as technology advances.

1.2.14

What happens when more people understand how a technology works?

- It is no longer seen as AI.
- It becomes more futuristic.
- It stops being used.
- It becomes less efficient.

1.3 Easy or difficult

1.3.1

One surprising thing about AI is that tasks that seem easy for humans can be incredibly difficult for AI. For example, consider picking up an object, something you probably do multiple times a day without even thinking about it. This seemingly simple action is actually a highly complex process when broken down into individual steps.

First, you use your eyes to scan your surroundings and figure out where the objects are. Then, you decide which object to pick up, plan how to move your hand toward it, and adjust the force of your grip to hold the object securely. If the object is heavier or lighter than you expected, you quickly adapt without dropping it or losing balance. This entire process feels natural because your brain and body work together seamlessly.

For robots, however, performing these actions is an enormous challenge. Robots must use cameras and sensors to detect objects, complex algorithms to plan their movements, and highly precise motors to adjust their grip. They also struggle to adapt to unexpected situations, such as when the weight of an object changes or the environment around them shifts. Grasping objects remains an active area of research in robotics and AI because replicating this "simple" human skill is anything but simple for machines.

1.3.2

Why is it hard for robots to pick up objects?

- Robots need to detect objects using cameras and sensors.
- Robots struggle to adapt to unexpected changes.
- Robots cannot plan movements as naturally as humans.
- Robots don’t have human-like fingers.

1.3.3

Another surprising aspect of AI is that tasks that seem difficult for humans can be relatively easy for AI. For instance, solving complex mathematical problems, analyzing large datasets, or playing chess at a high level might take humans years of training and practice. For AI, these tasks can often be performed quickly and accurately with the right programming and resources.

Take chess as an example. A human player might rely on memory, experience, and strategy to decide their next move. AI, on the other hand, can evaluate millions of possible moves in seconds, using algorithms to determine the best outcome. Similarly, analyzing a massive dataset might seem overwhelming to a person, but AI can sift through the information, find patterns, and make predictions in a fraction of the time.

This ability comes from the fact that AI is designed to handle repetitive, data-driven tasks efficiently. While it struggles with things humans find intuitive, like recognizing emotions or picking up objects, it excels at systematic tasks that require speed,

precision, and the ability to process vast amounts of information. This unique balance of strengths and weaknesses helps us better understand where AI can be most useful.

📝 1.3.4

What is the reason AI excels at tasks like playing chess or analyzing data?

- AI can evaluate many possibilities very quickly.
- AI can understand emotions and adapt naturally.
- AI uses human intuition to solve problems.
- AI is designed for physical tasks like picking up objects.

🕮 1.3.5

One way to understand AI is by looking at its key properties and characteristics. These properties make AI systems unique and different from traditional software or algorithms. While traditional programs follow fixed instructions, AI systems can change, improve, and operate more independently. Let's explore two important properties of AI:

- **Adaptivity** is the ability of an AI system to adjust its behavior based on new data or experiences. For example, when you use a music streaming service like Spotify, the AI learns your preferences over time. If you listen to more pop songs, it adapts by recommending similar tracks. This flexibility makes AI powerful because it doesn't rely on the same fixed rules - it gets better as it processes more data.
- **Autonomy** is another key property of AI. This means that an AI system can perform tasks in complex environments without needing constant instructions or guidance from a human. A self-driving car, for instance, can navigate roads, avoid obstacles, and make decisions without someone actively controlling it. This ability to work independently is what makes AI so useful in tasks that require quick decision-making or operation in unpredictable environments.

These properties allow AI to handle a wide range of tasks and solve problems in ways that traditional software cannot.

📝 1.3.6

What is the characteristic that distinguishes AI from traditional software?

- AI can adapt based on new data.
- AI follows fixed instructions.
- AI requires constant human input.

- AI only works in simple environments.

1.4 AI structure

🕮 1.4.1

AI is a broad field that includes many different technologies, each designed to solve specific types of problems. To better understand AI, we can group its technologies into categories based on what they do and how they work. Each of these categories serves a unique purpose:

- Machine learning focuses on teaching computers to learn from data and make predictions.
- Natural language processing helps computers understand and use human language
- Speech recognition enables machines to process spoken words.
- Expert systems mimic the decision-making of a human expert.
- Planning, scheduling, and optimization make processes more efficient by finding the best solutions.
- Robotics combines AI with machines to perform physical tasks
- Computer vision allows computers to interpret visual information like images and videos.

Together, these technologies form the building blocks of AI applications we see today. From self-driving cars and virtual assistants to smart manufacturing and personalized recommendations, AI is changing the way we live and work. Understanding these categories helps us appreciate how AI is shaping our world and preparing us for its future advancements.

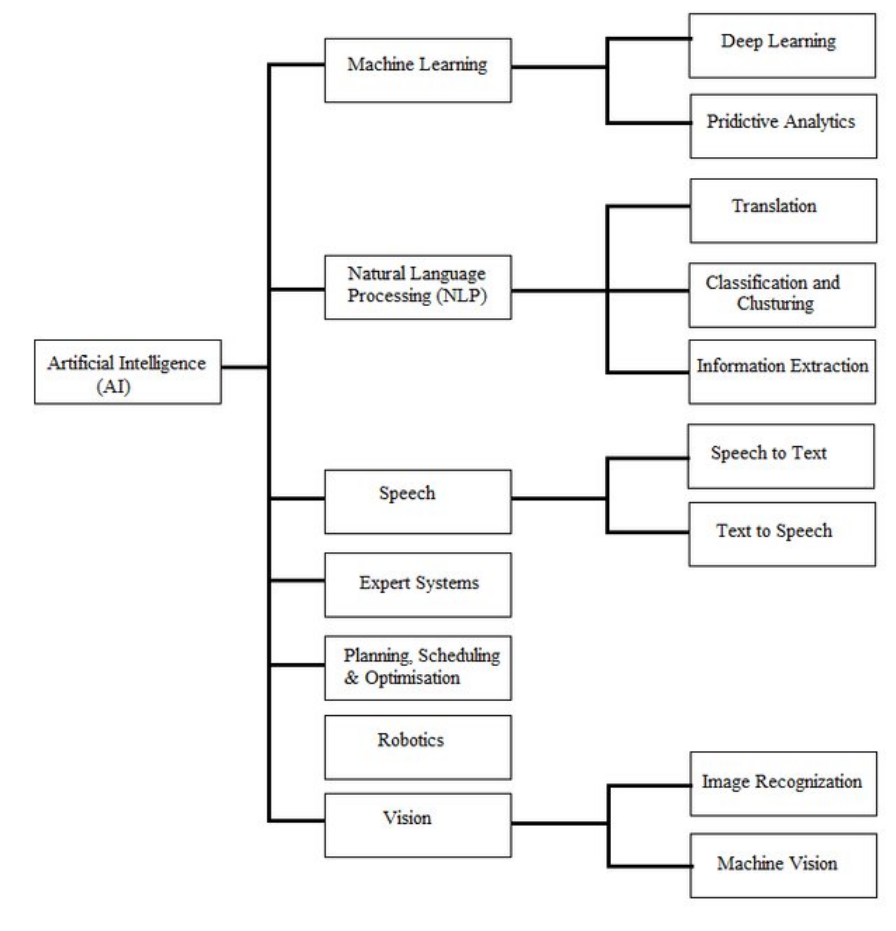


Figure: AI technologies

1.4.2

Which of the following are categories of AI technologies?

- Machine Learning.
- Speech Recognition.
- Web Development.
- Robotics.
- Computer Vision.

1.4.3

Machine learning

Machine learning is a branch of AI that enables computers to learn from data instead of being explicitly programmed. Imagine teaching a computer by showing it examples instead of writing a detailed set of instructions. For instance, if you give a computer thousands of images labeled as "cats" and "dogs", it can learn to identify whether a new picture shows a cat or a dog. This process of learning from data is called "training".

Machine learning systems use mathematical models, called algorithms, to find patterns in the data they are given. These patterns allow the system to make predictions or decisions based on new information. For example, when Netflix recommends movies or Spotify suggests songs, it's using machine learning to understand your preferences based on what you've watched or listened to before.

There are different types of machine learning: supervised learning, where the system learns from labeled data; unsupervised learning, where it identifies patterns without labels; and reinforcement learning, where the system learns by trial and error to achieve a goal. Each type is used for different tasks, like identifying spam emails, predicting weather, or teaching robots to walk.

1.4.4

What is the purpose of machine learning?

- To teach computers by showing examples.
- To write explicit instructions for tasks.
- To fix hardware issues in computers.
- To only store large amounts of data.

1.4.5

Natural language processing

Natural language processing (NLP) is a technology that helps computers understand and interact with human language. Whether you're chatting with Siri, using Google Translate, or getting suggestions for your emails, NLP is at work. Its goal is to bridge the gap between human language and computer understanding.

One key task of NLP is understanding the meaning of words and sentences. For example, when you type "restaurants near me", a search engine uses NLP to recognize that you're asking for nearby places to eat. Another task is language generation, like when AI writes text or answers questions in a chatbot.

NLP relies on large datasets of text and sophisticated algorithms to learn how people use language. These systems can even detect emotions, analyze sentiment in social media posts, and summarize long articles. NLP makes it easier for humans and machines to communicate in ways that feel natural and intuitive.

1.4.6

What are some examples of NLP?

- Chatting with Siri or Alexa.
- Using Google Translate.
- Summarizing long articles.
- Calculating the shortest driving route.

🕮 1.4.7

Speech recognition

Speech recognition is an AI technology that enables machines to understand spoken language. For example, when you say, “What’s the weather like?” to a virtual assistant like Alexa or Google Assistant, it uses speech recognition to convert your voice into text that the system can process.

The technology works by analyzing the sounds in your voice and matching them to patterns in its database. It then uses algorithms to predict what words you said. Speech recognition is incredibly useful, allowing hands-free control of devices, helping people with disabilities, and even aiding language learning.

Despite its usefulness, speech recognition faces challenges. Different accents, background noise, or unclear pronunciation can make it harder for the system to understand correctly. However, as AI improves, speech recognition systems are becoming more accurate and adaptable to various languages and speaking styles.

1.4.8

What is the main purpose of speech recognition?

- To understand and process spoken language.
- To translate written language into spoken words.
- To write programs for voice devices.
- To fix pronunciation errors in human speech.

🕮 1.4.9

Expert systems

Expert systems are AI programs designed to solve problems in a specific field using expert knowledge. They mimic the decision-making ability of a human expert. For example, an expert system in medicine can help doctors diagnose diseases based on symptoms entered into the system.

These systems consist of two main components: a knowledge base, which stores facts and rules, and an inference engine, which applies those rules to analyze new situations. For example, if a medical expert system "knows" that a fever and cough may indicate the flu, it can suggest flu as a possible diagnosis when a patient has these symptoms.

Expert systems are used in many areas, including finance, engineering, and customer support. However, they rely heavily on the quality of their knowledge base. If the data or rules are incomplete, the system may give incorrect advice.

📝 1.4.10

Where can expert systems be applied?

- Diagnosing diseases in medicine.
- Supporting customer service.
- Recommending financial strategies.
- Creating computer hardware.

🕮 1.4.11

Planning, scheduling, and optimization

Planning, scheduling, and optimization are AI technologies that help organize and improve processes. They are used to decide the best way to achieve a goal, such as scheduling airline flights, managing delivery routes, or optimizing production in a factory.

These systems analyze large amounts of data to find the most efficient solutions. For example, in delivery services, AI can calculate the fastest routes to deliver packages while saving fuel. In factories, optimization systems decide how to use machines and resources effectively to meet production deadlines.

These AI tools are critical in industries like logistics, manufacturing, and even healthcare, where planning and scheduling are essential for efficiency. By reducing time and costs, they make businesses more competitive and reliable.

📝 1.4.12

What is the purpose of optimization in AI?

- To find the most efficient solution for a task.
- To reduce errors in written language.
- To design new AI algorithms.
- To create smarter chatbots.

🕮 1.4.13

Robotics

Robotics is a field where AI helps machines perform physical tasks. Robots equipped with AI can learn, adapt, and interact with their environment. For example, a robot vacuum cleaner can navigate a room, avoid obstacles, and clean efficiently without human control.

Robots use sensors, cameras, and algorithms to understand their surroundings. This allows them to perform tasks like assembling cars, delivering packages, or even assisting in surgeries. AI gives robots the ability to make decisions and react to changes in real time.

However, robotics faces challenges like ensuring safety, especially when robots work near humans. Engineers must design systems that prevent accidents and allow robots to function reliably in different environments.

📝 1.4.14

Which of these are tasks performed by AI in robotics?

- Navigating a room to clean.
- Assembling cars in factories.
- Assisting in surgeries.
- Writing software programs.

🕮 1.4.15

Computer vision

Computer vision is an AI technology that allows computers to "see" and interpret visual information, such as images and videos. For example, AI can analyze photos to identify objects, like recognizing a cat in a picture or detecting a face in a crowd.

This technology is used in many applications, like self-driving cars, which use cameras and computer vision to identify road signs, other vehicles, and pedestrians. It’s also used in medical imaging to help doctors detect diseases in X-rays or MRIs.

Computer vision works by analyzing patterns in visual data, using algorithms trained on large datasets of images. As it improves, it’s becoming more accurate and finding use in industries like security, entertainment, and retail.

📝 1.4.16

What is the main goal of computer vision?

- To interpret visual information like images and videos.
- To teach robots to walk.
- To improve text translation accuracy.
- To analyze human speech patterns.

Perspectives on AI

Chapter 2

2.1 Turing test

📖 2.1.1

Can machines think like humans?

The Turing test, proposed by mathematician and computing pioneer Alan Turing in 1950, is one of the most famous methods for evaluating whether a machine can exhibit human-like intelligence. Turing was fascinated by the concept of intelligence and whether machines could replicate human thinking. He proposed a simple but powerful idea to test this: the **imitation game**.

The imitation game was originally about distinguishing people based on gender. Turing adapted this concept into a test for machines. The goal of the Turing test is to determine whether a computer can imitate human behavior so convincingly that a person cannot reliably tell if they are interacting with a machine or a human.

Here's a simple version of the Turing test:

1. You enter a room with a computer terminal.
2. For a fixed period, you can type questions or statements into the terminal and observe the replies.
3. On the other end of the terminal is either a human or a computer system.
4. If the responses are so convincing that you cannot reliably tell whether they come from a human or a computer, the system is considered "intelligent".

The Turing test challenges AI systems to use language, context, and logic like humans. While some systems can mimic conversation effectively, no program has yet fully passed the test under strict conditions. This test remains an important benchmark in AI research, driving efforts to create systems capable of understanding and interacting naturally with humans.

📝 2.1.2

What is the main goal of the Turing test?

- To test whether a machine can imitate human intelligence convincingly.
- To determine whether a machine can perform calculations faster than humans.
- To evaluate whether a computer can identify objects.
- To check if machines can solve mathematical puzzles.

📖 2.1.3

One variation of the Turing test takes the imitation game a step further by introducing two players - one human and one computer - and a human interrogator. The interrogator communicates with both players through written messages, such as in a chat interface, and their task is to figure out which player is human and which is the computer.

The computer's goal is to convince the interrogator that it is the human, while the human participant simply tries to be themselves. If the interrogator cannot reliably distinguish the computer from the human, then the computer is said to have passed the test.

The idea behind this variation is that human-level intelligence would allow a computer to engage in a general natural language conversation, just as a person would. This means understanding context, providing relevant answers, and even displaying creativity or humor. Passing this test would indicate that a computer has reached a level of intelligence comparable to that of a human being.

While modern AI systems have made impressive progress in mimicking human conversation, no system has yet consistently passed this variation of the Turing test under strict conditions. However, it remains a significant goal in the field of AI development, pushing researchers to create systems that better understand and respond to human communication.

📝 2.1.4

What are key elements of the Turing test variation?

- A human interrogator communicates with two players.
- One player is a human, and the other is a computer.
- The goal is for the computer to be indistinguishable from the human.
- Both must solve mathematical problems.

📖 2.1.5

Criticism of the Turing test

The Turing test is a well-known method for evaluating whether a machine can exhibit human-like intelligence, but it has faced criticism. One major criticism is that the test may measure how much a computer behaves like a human rather than how intelligent it actually is. This distinction is important because behaving like a human doesn’t always mean demonstrating true intelligence or understanding.

For example, some programs have "passed" the Turing test by relying on strategies like changing the subject, making spelling mistakes, or refusing to answer certain questions. These tactics make the program appear human-like but don’t necessarily reflect real intelligence. One famous case is **Eugene Goostman**, a program that

pretends to be a 13-year-old Ukrainian boy. Eugene avoids answering questions directly, frequently makes jokes, and changes the topic to his pet guinea pig. These behaviors can confuse the human interrogator and make it difficult to determine if Eugene is a machine, but they don't demonstrate deep understanding or logical reasoning.

This criticism suggests that the Turing Test might not be the best measure of intelligence. Instead, it evaluates how convincingly a computer can simulate human conversation, which is a different goal. Researchers continue to explore other methods for testing intelligence that go beyond simply imitating human behavior.

📝 2.1.6

What is the criticism of the Turing test?

- It focuses on human-like behavior rather than true intelligence.
- It doesn't test how well a computer solves math problems.
- It only works with computers programmed to avoid questions.
- It requires a human and computer to work together.

🕮 2.1.7

Chinese room experiment

Is intelligence the same as intelligent behavior? This question has been debated by many thinkers, and one of the most famous challenges to this idea is **John Searle's Chinese room experiment**. This thought experiment highlights the difference between appearing intelligent and truly understanding.

In the Chinese room experiment, imagine a person who doesn't know Chinese locked inside a room. Outside the room, someone who speaks Chinese writes messages and slips them through a mail slot. Inside the room, the person has a large manual with detailed instructions for responding to the Chinese messages. By following the instructions step by step, the person inside the room writes replies in Chinese and sends them back out.

To the person outside the room, it might seem like they are having a conversation with someone who understands Chinese. But in reality, the person inside the room is just following instructions and doesn't understand the language at all.

Searle's argument is that even if a machine behaves in an intelligent way - such as passing the Turing test - it doesn't mean the machine is truly intelligent. It simply processes information according to programmed rules, without understanding or having a "mind" like a human. This thought experiment challenges the idea that intelligent behavior alone is enough to define intelligence.

📝 2.1.8

What does the Chinese room experiment argue about machines?

- Machines can behave intelligently without true understanding.
- Machines follow programmed instructions like a manual.
- Passing the Turing Test doesn't prove true intelligence.
- Machines can have a "mind" just like humans.

🕮 2.1.9

What does passing the Turing test imply?

Passing the Turing test means that a computer can produce responses so human-like that it convinces a person they are interacting with another human. However, this does not necessarily prove that the machine is truly "intelligent" in the way humans are. It simply shows that the machine can imitate human-like responses under specific conditions defined by the test.

Some critics argue that the Turing test focuses too much on mimicking human conversational behaviors rather than measuring true intelligence. For example, it doesn't evaluate a machine's ability to think independently, understand emotions, or solve problems outside of a conversation. Others believe the test overlooks other forms of intelligence that machines might demonstrate, such as the ability to process vast amounts of data quickly or solve complex mathematical problems.

The legacy of the Turing test

Despite its limitations and criticisms, the Turing test remains a landmark concept in the study of artificial intelligence. It raises profound questions about the nature of intelligence, the mind, and whether machines can ever be considered equivalent to humans. The Turing test continues to inspire research and debates in fields like cognitive science, computer science, and philosophy.

Rather than focusing solely on whether machines are "intelligent", researchers today are more interested in creating AI systems that solve practical problems. Whether these systems are genuinely intelligent or simply act as if they are doesn't always matter as long as they are helpful in real-world applications.

📝 2.1.10

What does passing the Turing test show about a machine?

- The machine can imitate human-like responses under specific conditions.
- The machine is truly intelligent like a human.
- The machine can solve mathematical problems independently.
- The machine can process emotions like a human.

2.2 Types of systems

📖 2.2.1

Artificial intelligence can be understood from different perspectives, each focusing on specific aspects of intelligence and how to simulate or replicate it. These perspectives guide how researchers design AI systems and what goals they aim to achieve. The study of AI is often categorized into four main viewpoints:

1. **Systems that think like humans** focus on replicating human thought processes, such as problem-solving and decision-making.
2. **Systems that act like humans** aim to mimic human behavior, such as holding conversations or performing tasks like walking or cooking.
3. **Systems that think rationally** emphasizes logical reasoning, where AI uses mathematical models to make the best possible decisions.
4. **Systems that act rationally** are designed to take rational actions to achieve specific goals, such as navigating a self-driving car or delivering packages.

Each approach highlights a unique aspect of AI, from understanding how humans think to finding the most efficient solutions. Together, these perspectives help us build AI systems that can solve complex problems, interact naturally with people, and improve our daily lives.

📝 2.2.2

Which perspective focuses on AI systems that mimic human behavior?

- Systems that act like humans.
- Systems that think rationally.
- Systems that think like humans.
- Systems that act rationally.

📖 2.2.3

Systems that think like humans

One approach to studying AI focuses on creating systems that **think like humans**. This means designing AI that tries to mimic the way people think and solve problems. For example, when you solve a puzzle, your brain goes through steps like analyzing the problem, identifying possible solutions, and picking the best one. AI systems that think like humans attempt to recreate these mental processes.

Researchers studying this approach often look to psychology and neuroscience for inspiration. They study how the human brain works and try to program computers to imitate these thought patterns. For instance, they might design an AI to learn like a person by observing examples and practicing tasks.

An example of this type of AI is an expert system that mimics how a doctor thinks when diagnosing an illness. The AI collects symptoms, analyzes them, and reaches a conclusion, much like a human doctor would. This focus on replicating human

thinking helps AI systems handle tasks requiring problem-solving and decision-making.

📝 2.2.4

What do AI systems that think like humans try to mimic?

- The way humans think and solve problems.
- The way humans speak.
- The way humans move.
- The way humans see objects.

🕮 2.2.5

Systems that act like humans

Another approach to AI is designing systems that **act like humans**. Instead of focusing on thought processes, this approach aims to replicate human behavior. For example, a chatbot that can hold a conversation or a robot that can perform tasks like walking, talking, or even cooking is designed to act like a human.

These systems are often evaluated based on how well their behavior matches human actions. For instance, the famous "Turing Test" measures whether an AI can respond in a way that makes it indistinguishable from a human during a conversation. If you can't tell whether you're talking to a machine or a person, the AI has succeeded in acting like a human.

This approach combines elements of AI, robotics, and linguistics to create systems that can interact naturally with people. By simulating human behavior, these AI systems make technology more accessible and relatable, like when virtual assistants like Siri or Alexa help you with tasks.

📝 2.2.6

Which of the following are examples of AI systems that act like humans?

- A chatbot that holds a conversation.
- A robot that cooks or walks.
- A self-driving car.
- A program that analyzes data patterns.

🕮 2.2.7

Systems that think rationally

Some AI systems are designed to **think rationally**, meaning they use logical reasoning to make decisions. These systems focus on finding the most logical and efficient solutions to problems. For example, a navigation app that calculates the fastest route to your destination is designed to think rationally - it uses data about traffic, distance, and speed limits to make the best decision.

This approach often involves mathematical models and algorithms. Researchers study principles of logic to program AI systems that can analyze complex problems and break them into smaller, solvable parts. Rational thinking doesn't try to imitate how humans think but focuses on achieving the most accurate results.

One example is a chess-playing AI, which evaluates millions of potential moves to choose the best one. These systems are often used in fields like engineering, science, and finance, where precision and efficiency are essential.

📝 2.2.8

What do AI systems that think rationally focus on?

- Using logical reasoning to solve problems.
- Mimicking human emotions.
- Replicating human speech patterns.
- Performing physical tasks like humans.

🕮 2.2.9

Systems that act rationally

The fourth approach to AI focuses on designing systems that **act rationally**. These systems aim to achieve the best possible outcome in any situation by taking rational actions. For example, a self-driving car acts rationally when it avoids obstacles, follows traffic rules, and gets passengers to their destination safely and efficiently.

This approach is often associated with intelligent agents - AI systems designed to perceive their environment, make decisions, and take actions to achieve specific goals. These systems don't necessarily mimic humans but aim to perform tasks in the best possible way based on logic and data.

Rationally acting AI is commonly used in areas like robotics, logistics, and autonomous systems. For instance, a delivery robot that navigates crowded sidewalks to deliver a package uses rational actions to achieve its goal while avoiding accidents or delays.

📝 2.2.10

Which of the following describe AI systems that act rationally?

- They aim to achieve the best possible outcome.

- They follow logical actions based on data.
- They mimic human thought processes.
- They focus on recreating human speech.

2.2.11

Add a correct example to the approach of AI:

- Systems that Act Rationally - _____

- Systems that Think Rationally - _____

- Systems that Act Like Humans - _____

- Systems that Think Like Humans - _____

- Neuroscience-based AI systems
- Expert systems
- AI systems in games like chess or go
- Chatbots and virtual assistants

2.2.12

Differences between human and artificial intelligence

Now that we understand the various types of artificial intelligence (AI) systems, let's explore the key differences between human intelligence and artificial intelligence. Human intelligence is represented by individuals like doctors or engineers, while artificial intelligence is represented by systems like expert programs or machine learning models.

Artificial intelligence:

- Stability - AI doesn’t forget or leave an organization like humans might. Once programmed, its knowledge remains stable.
- Scalability - expanding an AI system is easier because it doesn’t require the long learning process humans need to acquire new skills. For example, updating an AI involves programming new rules or adding data, while training a human takes years of education and practice.
- Cost - using AI can often be cheaper in the long run compared to human labor, particularly for repetitive tasks.
- Consistency - AI provides constant performance and doesn’t face issues like mood swings, fatigue, or unreliability.
- Documentation - AI's decision-making processes are often easier to document and analyze, providing transparency that human intuition sometimes lacks.

Human intelligence:

- Creativity - humans can think outside the box, coming up with new ideas or solutions that AI cannot generate without guidance.
- Sensory perception - humans can directly use their senses—sight, hearing, touch, etc.—to understand their environment without relying on sensors and interpretation.
- Contextual experience - humans draw from a broad context of personal and cultural experiences, which helps them make nuanced decisions.
- Intuition - humans often rely on intuition to make quick judgments in situations where there isn't enough time to analyze all the data, something AI struggles with.

Each type of intelligence excels in different areas. AI is better suited for repetitive, data-driven, and scalable tasks, while human intelligence shines in creative, intuitive, and context-dependent situations.

2.2.13

Choose which features of human intelligence are better than artificial intelligence

- Creativity
- Using senses directly
- Actions based on intuition
- Documentability

2.2.14

Which of the following are advantages of artificial intelligence over human intelligence?

- AI is more stable and doesn't forget.
- AI is easier to extend with new skills.
- AI is better at creativity and intuition.
- AI provides consistent performance without fatigue.

Machine Learning

Chapter 3

3.1 Introduction

📖 3.1.1

Machine learning is a branch of artificial intelligence that focuses on teaching computers to learn from data and improve their performance without being explicitly programmed for every task. In traditional programming, humans write detailed instructions for computers to follow. However, machine learning takes a different approach - computers use algorithms and statistical models to figure things out on their own.

For example, in traditional programming, if you want a computer to recognize a cat in an image, you'd need to write a detailed program describing the specific features of a cat. In machine learning, the system learns to recognize cats by analyzing a large dataset of labeled images of cats and non-cats. Over time, it becomes better at recognizing cats, even in pictures it hasn't seen before.

The main idea behind machine learning is to use data to identify patterns, make predictions, and solve problems efficiently. This makes machine learning one of the most important and rapidly growing areas in technology today.

📝 3.1.2

How is machine learning different from traditional programming?

- Machine learning writes detailed instructions for every task.
- Machine learning requires humans to manually program each step.
- Machine learning teaches computers to learn from data and improve.
- Machine learning only works for mathematical calculations.

📖 3.1.3

Machine learning is especially useful in situations where creating explicit programs would be too complex or time-consuming. For example, programming the behavior of an autonomous robot involves many factors - navigation, obstacle avoidance, and decision-making. Writing instructions for every possible scenario would be nearly impossible. Instead, machine learning enables the robot to learn from its environment and adapt its behavior based on what it has learned.

By analyzing data, machine learning helps solve practical problems that are too complicated to solve manually. It can identify patterns, optimize processes, and make decisions that would take humans much longer. This makes machine learning essential for tasks like self-driving cars, medical diagnoses, and even recommending what shows to watch on Netflix.

Machine learning is valuable because it simplifies complex tasks, reduces the workload for programmers, and creates systems that can adapt and improve over time.

📝 3.1.4

Why is machine learning useful?

- It simplifies programming complex behaviors like those of autonomous robots.
- It enables computers to learn from their environment and adapt.
- It solves problems that are too complex for manual solutions.
- It eliminates the need for any programming altogether.
- It only works for creating video games.

🕮 3.1.5

The process of machine learning can be broken into two main steps:

1. Gathering a dataset - the first step is to collect a large amount of data that is related to the problem you want to solve. For example, if you want to teach a system to predict house prices, you would gather data about house sizes, locations, and prices.
2. Building a statistical model - the second step involves using algorithms to create a model that analyzes the dataset and identifies patterns. This model can then make predictions or decisions.

Once the statistical model is built, it is used to solve real-world problems. For instance, a weather prediction system might analyze past weather data to predict tomorrow's forecast. Machine learning models continue to improve as they are exposed to more data, making them more accurate and reliable over time.

📝 3.1.6

What is the first step in the machine learning process?

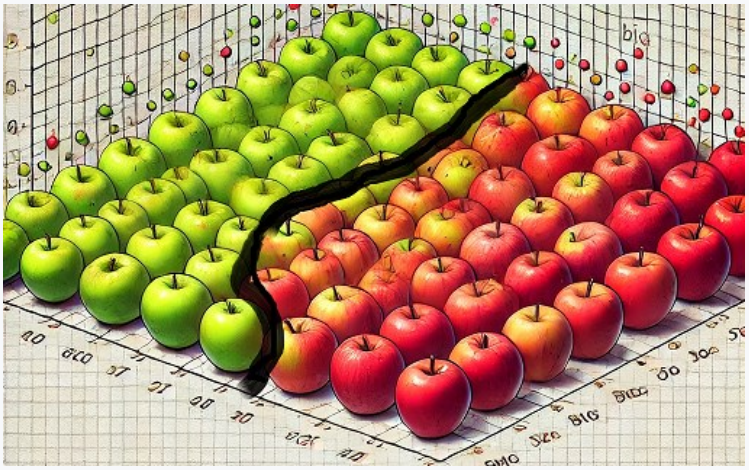
- Gathering a dataset related to the problem.
- Building a statistical model.
- Writing detailed programming instructions.
- Testing the model on unrelated data.

🕮 3.1.7

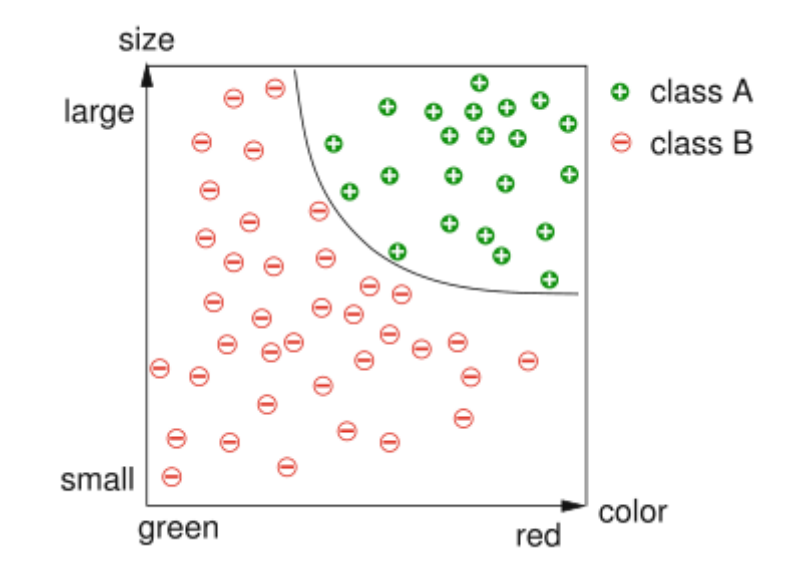
What does "learning" mean

In machine learning, the term **"learning"** refers to the process by which a computer system improves its ability to perform a task by using data or experience. Instead of being explicitly programmed for every task, the system analyzes data to find patterns and make predictions.

Imagine you want to sort apples into 2 merchandise categories based on two **features** size and color. Apples are hand-picked by a specialist and classified by him. This is the dataset that can be used to find out the border between these two classes. The task in machine learning consists of generating a function from the collected, classified data which calculates the class value (A or B) for a new apple from the two features' size and colour.



We can represent the situation with positive (green) and negative (red) elements.



This example is still very simple to find such a dividing line for the two classes. It is a more difficult, and above all much less visualizable task, when the objects to be classified are described by not just two, but many features.

📝 3.1.8

Which of the following describe tasks in machine learning?

- Finding a dividing line (curve) between two classes using features.
- Generating a function from training data to predict new outputs.
- Minimizing errors in classifying objects based on their features.
- Sorting objects by hand using predefined rules.
- Writing explicit instructions to handle every possible scenario.

🕮 3.1.9

Imagine you own a business that sells apples, and you want to divide them into two categories: **Premium** and **Regular**. Premium apples might be larger and have a bright red color, while Regular apples could be smaller or green or have a mix of colors. This division is important because customers are willing to pay more for Premium apples, and you want to ensure consistency in your sorting process.

Traditionally, you might hire a specialist to manually inspect **each** apple and sort them into the two categories based on their size and color. However, this process can be slow, subjective, and prone to errors, especially when sorting hundreds or thousands of apples. That's where machine learning can help!

How can machine learning solve the problem:

1. Training the system - at first you collect a dataset by asking the specialist to sort a batch of apples and record their features (size and color) along with their categories (Premium or Regular). This dataset is used to train a machine learning model to recognize patterns in the features of Premium and Regular apples.
2. Making predictions - once the model is trained, it can analyze new apples and classify them as Premium or Regular based on their features. For example, if an apple is large and bright red, the model might classify it as Premium. If it's smaller and less colorful, it would classify it as Regular.

Benefits:

- Speed - the system can sort apples much faster than a human.
- Consistency - it applies the same rules every time, reducing errors caused by fatigue or bias.
- Scalability - the system can handle large volumes of apples without additional effort.

By using machine learning, you can streamline the sorting process, save time, reduce costs, and improve customer satisfaction by ensuring consistent quality in your product categories.

📝 3.1.10

What is one way machine learning helps in sorting apples?

- By identifying patterns in features like size and color to classify apples.
- By memorizing each apple's features individually.
- By replacing the need for collecting data.
- By manually inspecting each apple for quality.

3.2 Classification

🕮 3.2.1

Imagine you own a business selling apples, and you need to decide whether each apple belongs in the **Premium** or **Regular** category. This decision is based on specific features, such as the size and color of the apples. A **classifier** is a tool in machine learning that helps you make this decision automatically by learning from examples.

A classifier is like a digital expert trained to spot patterns in data. During training, the classifier examines a dataset of apples, where each apple is labeled as either Premium or Regular, along with its size and color information. The classifier learns the relationship between the features (size and color) and the labels (Premium or Regular).

Red_Percentage	Weight	Category
85	210	Premium
70	190	Premium
50	160	Standard
75	130	Standard
90	250	Premium
65	180	Premium

Once the classifier is trained, it can make predictions about new apples. For instance, if you give it an apple's size and color, it will decide whether the apple is Premium or Regular. The more accurately the classifier understands the patterns, the better it will perform.

Why use a classifier? Because it's faster, more consistent, and scalable compared to human sorting. It can handle hundreds or even thousands of apples without getting tired or making mistakes.

📝 3.2.2

What is the purpose of a classifier in machine learning?

- To automatically categorize apples based on features
- To manually inspect apples
- To clean the dataset
- To measure the size of the apples

📖 3.2.3

Now that you understand what a classifier is, let's talk about how it learns and how we ensure it works properly. This is where **training** and **testing datasets** come in.

When you train a classifier, you need to give it examples to learn from. This is the **training dataset**, which contains labeled examples of apples. For instance, the training dataset might include information about 30 apples: their sizes, colors, and whether they are Premium or Regular. The classifier uses this data to learn patterns, such as "larger and redder apples are more likely to be Premium".

But how do you know the classifier is actually doing a good job? That's where the **testing dataset** comes in. The testing dataset is a separate group of apples (e.g., 10 apples) that the classifier hasn't seen before. You use it to check whether the classifier can correctly categorize these apples. If it does well on the testing data, it's likely to work well on new apples in real life.

Why do we separate the data? If we used the same apples for training and testing, the classifier might just memorize those examples instead of learning general patterns. This would make it perform poorly on new apples. Separating the data ensures the classifier is genuinely learning and not just memorizing.

📝 3.2.4

Why do we use a testing dataset in machine learning?

- To evaluate how well the classifier works on new data
- To ensure the classifier doesn't just memorize the training data
- To measure the size of the apples
- To train the classifier to recognize patterns

📖 3.2.5

How to choose the right classifier

Now that we understand what a classifier is and why we need training and testing datasets, let's talk about an important question: **How do you choose the correct classifier?**

A classifier is like a tool in a toolbox. Just as you wouldn't use a screwdriver to hammer a nail or screwdriver and screws, you need to choose the right classifier for the problem you're trying to solve. In our apple-sorting example, the classifier's job is to categorize apples as Premium or Regular based on their size and color. So we need to consider a few factors:

- Type of data - some classifiers work better with numerical data (like apple size), while others handle categorical data (like Premium or Regular labels). For apple sorting, we have numerical data (size and percentage of red color) and a categorical label (Premium or Regular). A classifier like Decision Tree or Logistic Regression works well in such cases.
- Complexity of the problem - if the patterns in the data are simple and clear, we can use a simple classifier like Logistic Regression. For more complex patterns, such as multiple overlapping features, advanced classifiers like Support Vector Machines (SVM) or Neural Networks might be needed.
- Amount of data - if we have a small dataset (e.g., 40 apples), simpler classifiers like Decision Trees or k-Nearest Neighbors (k-NN) are a good choice because they don't need a lot of data to work well. For larger datasets, more complex classifiers like Neural Networks or Gradient Boosted Trees can leverage the extra information.
- Speed and resources - If we need quick results and don't have powerful computers, stick with simple models like Decision Trees. If we have more time and resources, advanced classifiers like Neural Networks can give better accuracy but may take longer to train.
- Interpretable results - some classifiers, like Decision Trees, are easy to understand and explain. For example, we can clearly see why an apple was classified as Premium or Regular. Others, like Neural Networks, act more like a "black box", meaning they're harder to interpret.

We don't create the classifier ourselves, it is available in extensive Python libraries, we just need to know which one to choose. Choosing the right classifier depends on your data, the complexity of the problem, the resources you have, and how important it is to understand the model's decisions.

3.2.6

Which factors are important when choosing a classifier?

- The type of data
- The complexity of the problem
- The color of the classifier
- The price of the apples

3.2.7

Project: Apple sorting by color and size

An apple orchard wants to automate the sorting process of their apples into two categories: **Premium** and **Standard**. Premium apples are those that have a high percentage of red color and are heavier in weight. Standard apples have less red color and weigh less. The orchard has provided data for 40 apples, each categorized by experts. Your task is to analyze this data, build a classification model, and use it to classify new apples into the correct category.

1. Prepare dataset

```
import pandas as pd

# Apple dataset provided by the orchard
data = {
    'Apple_ID': range(1, 41),
    'Red_Percentage': [85, 70, 50, 90, 65, 30, 95, 55, 40, 80,
                       75, 60, 35, 85, 45, 20, 88, 53, 67, 78,
                       69, 49, 90, 77, 63, 31, 72, 68, 51, 83,
                       88, 54, 47, 92, 39, 70, 79, 66, 58,
62],
    'Weight': [210, 190, 160, 250, 180, 130, 270, 150, 120,
220,
               200, 175, 140, 240, 160, 110, 260, 155, 185,
230,
               170, 150, 245, 210, 180, 125, 195, 185, 165,
225,
               250, 155, 145, 265, 135, 200, 215, 190, 175,
180],
    'Category': ['Premium', 'Premium', 'Standard', 'Premium',
'Premium',
                 'Standard', 'Premium', 'Standard',
'Standard', 'Premium',
                 'Premium', 'Premium', 'Standard', 'Premium',
'Standard',
                 'Standard', 'Premium', 'Standard', 'Premium',
'Premium',
```

```
                  'Premium', 'Standard', 'Premium', 'Premium',
'Premium',
                  'Standard', 'Premium', 'Premium', 'Standard',
'Premium',
                  'Premium', 'Standard', 'Standard', 'Premium',
'Standard',
                  'Premium', 'Premium', 'Premium', 'Premium',
'Premium']
}

# Create a DataFrame
df = pd.DataFrame(data)
print(df.head())
```

Program output:

```
   Apple_ID  Red_Percentage  Weight  Category
0         1              85     210   Premium
1         2              70     190   Premium
2         3              50     160  Standard
3         4              90     250   Premium
4         5              65     180   Premium
```

2. Understanding the dataset

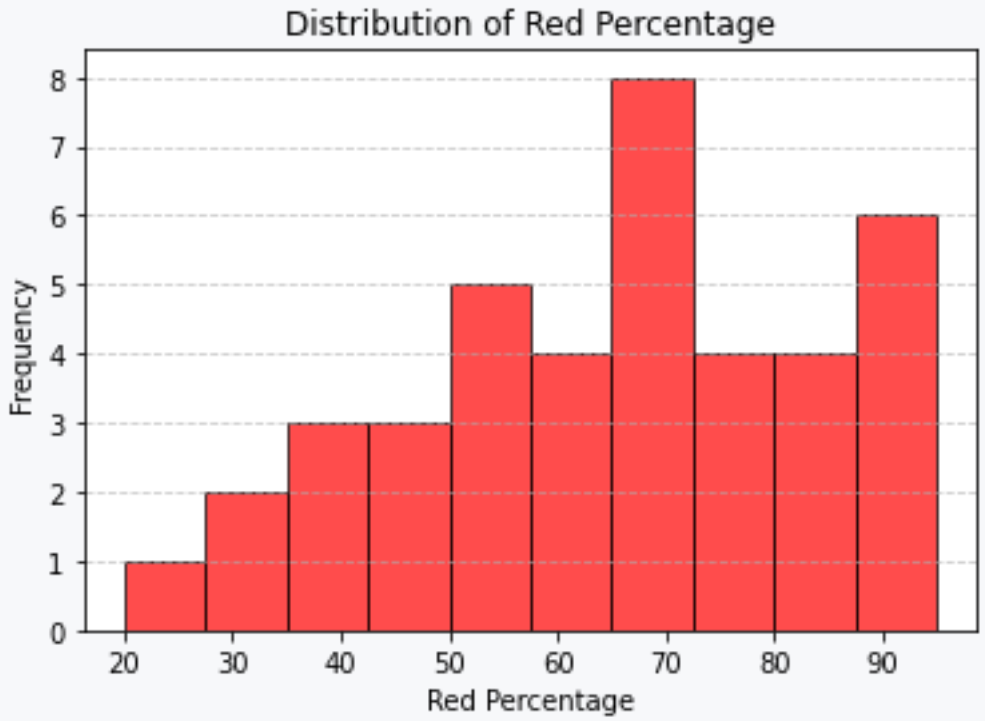
- To understand the data we need to explore the data to gain insights into the variables and their relationships. This process is named (Exploratory data analysis - EDA)

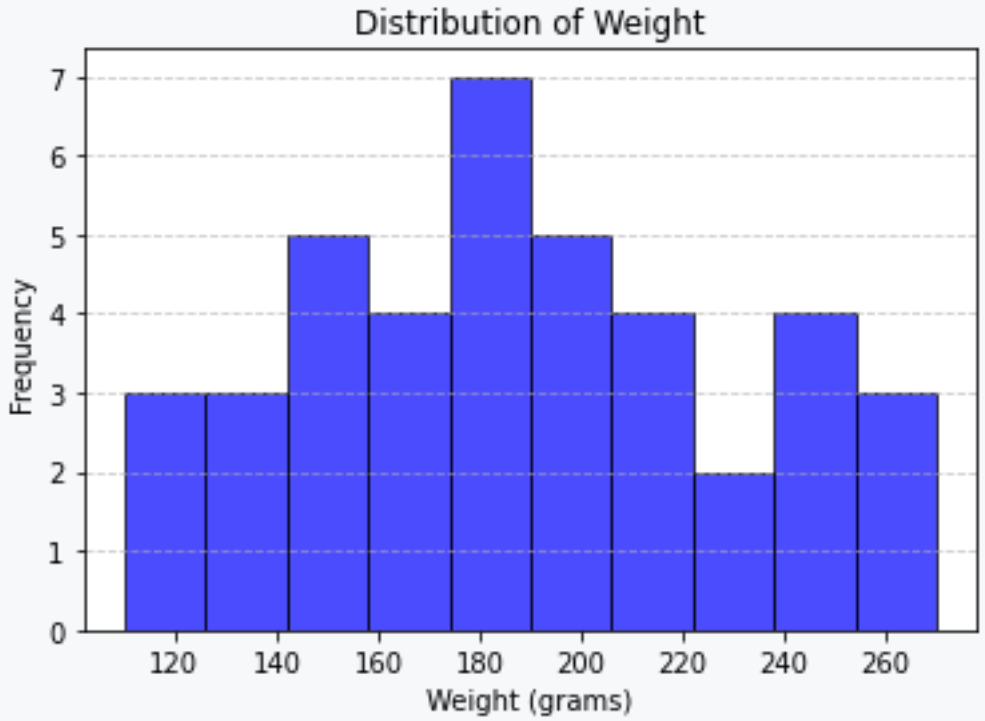
```
import matplotlib.pyplot as plt

# Check basic information
print(df.info())
print(df.describe())

# Visualize the distribution of red percentage
plt.hist(df['Red_Percentage'], bins=10, color='red',
alpha=0.7, edgecolor='black')
plt.title('Distribution of Red Percentage')
plt.xlabel('Red Percentage')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Visualize the distribution of weight
```

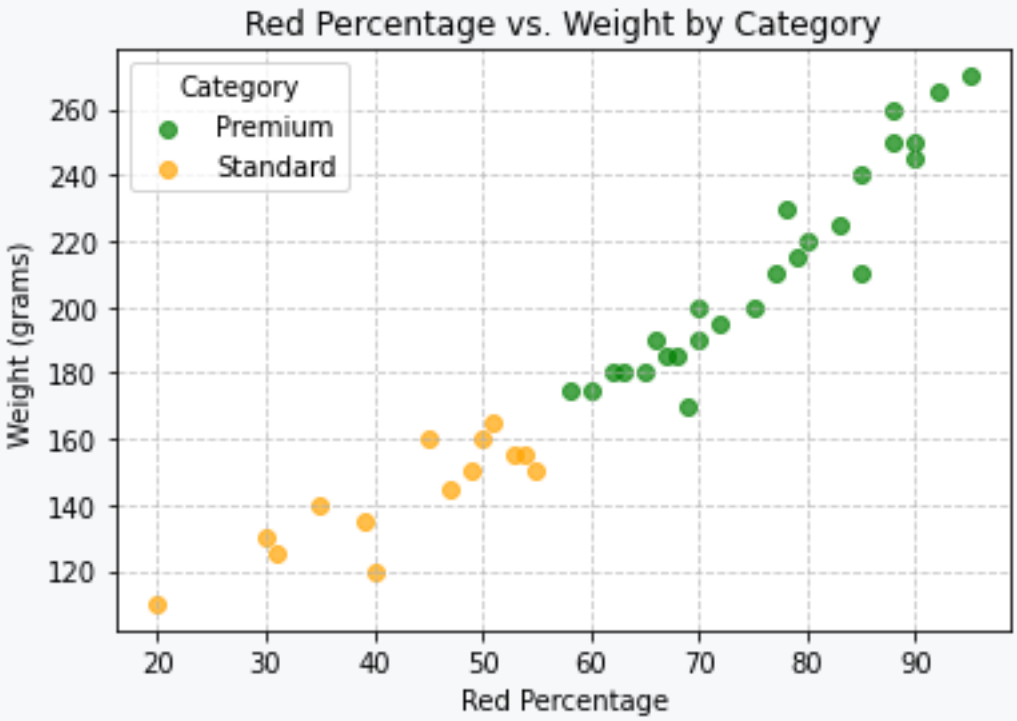
```
plt.hist(df['Weight'], bins=10, color='blue', alpha=0.7,
edgecolor='black')
plt.title('Distribution of Weight')
plt.xlabel('Weight (grams)')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Relationship between red percentage and weight by category
colors = {'Premium': 'green', 'Standard': 'orange'}
for category in df['Category'].unique():
    subset = df[df['Category'] == category]
    plt.scatter(subset['Red_Percentage'], subset['Weight'],
label=category, color=colors[category], alpha=0.7)

plt.title('Red Percentage vs. Weight by Category')
plt.xlabel('Red Percentage')
plt.ylabel('Weight (grams)')
plt.legend(title='Category')
plt.grid(linestyle='--', alpha=0.7)
plt.show()
```

Program output:

```
RangeIndex: 40 entries, 0 to 39
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Apple_ID        40 non-null     int64
 1   Red_Percentage  40 non-null     int64
 2   Weight          40 non-null     int64
 3   Category        40 non-null     object
dtypes: int64(3), object(1)
memory usage: 1.4+ KB
None
        Apple_ID  Red_Percentage      Weight
count  40.000000       40.000000   40.000000
mean   20.500000       64.350000  187.375000
std    11.690452       19.260096   42.577042
min     1.000000       20.000000  110.000000
25%    10.750000       50.750000  155.000000
50%    20.500000       66.500000  182.500000
75%    30.250000       79.250000  216.250000
max    40.000000       95.000000  270.000000
```


Distribution of Red Percentage
Frequency
Red Percentage
0
1
2
3
4
5
6
7
8
20
30
40
50
60
70
80
90


Distribution of Weight
Frequency
Weight (grams)
0
1
2
3
4
5
6
7
120
140
160
180
200
220
240
260



3. Preparing the classification model

When building a machine learning model, it's important to evaluate how well it can make predictions on unseen data, not just on the data it was trained on. To do this, we divide the dataset into two parts:

- **Training set** is used to "train" the model. The model learns patterns and relationships in the data during training.
- **Testing set** is kept separate and is used to test how well the trained model performs on data it hasn't seen before.

By splitting the dataset, we simulate real-world scenarios where the model will encounter new data. This helps us:

- **Avoid overfitting** what happens when a model learns the training data too well but fails to generalize to new data. Splitting helps us detect this problem.
- **Evaluate accuracy** which lets us measure how reliable the model is for practical use.

We need a Model classifier?

A **classifier** is a type of machine learning model designed to assign labels or categories to data points based on input features. For example, in this project:

- The **input features** are the percentage of red color and the weight of the apple.

- The **output labels** are the categories: **Premium** or **Standard**.

The model analyzes patterns in the training data and learns how these features correspond to each label. Once trained, the model can classify new, unseen apples into one of the two categories.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report

# Prepare the data
X = df[['Red_Percentage', 'Weight']]
y = df['Category']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train the model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Program output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

     Premium       1.00      1.00      1.00         6
    Standard       1.00      1.00      1.00         2

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8
```

4. Classifying new apples

- And now we can classify new apple data using the trained model:

```
# New apple data
new_apples = pd.DataFrame({
    'Red_Percentage': [85, 50, 75],
    'Weight': [240, 160, 200]
})

# Predict the category
new_apples['Category'] = model.predict(new_apples)
print(new_apples)
```

Program output:

```
   Red_Percentage  Weight  Category
0              85     240   Premium
1              50     160  Standard
2              75     200   Premium
```

3.3 Regression

3.3.1

Imagine you are running a fruit juice business, and you want to predict how much juice you can get from an orange. You know that bigger oranges usually produce more juice, but you don't have time to measure their size. Instead, you decide to use the **weight** of the oranges to make an approximate prediction.

Regression is a method that helps us predict a **number** based on input data. In this case, you will use the weight of the orange to predict how much juice it will produce. For example:

- If an orange weighs 120 grams, it might produce around 160 ml of juice.
- If it weighs 180 grams, it might produce more, say 210 ml.

The model doesn't give an exact answer because the amount of juice also depends on other factors, like size and how juicy the orange is. But weight is a good enough feature to make a reasonable prediction.

We can draw information about collected and experienced pieces.

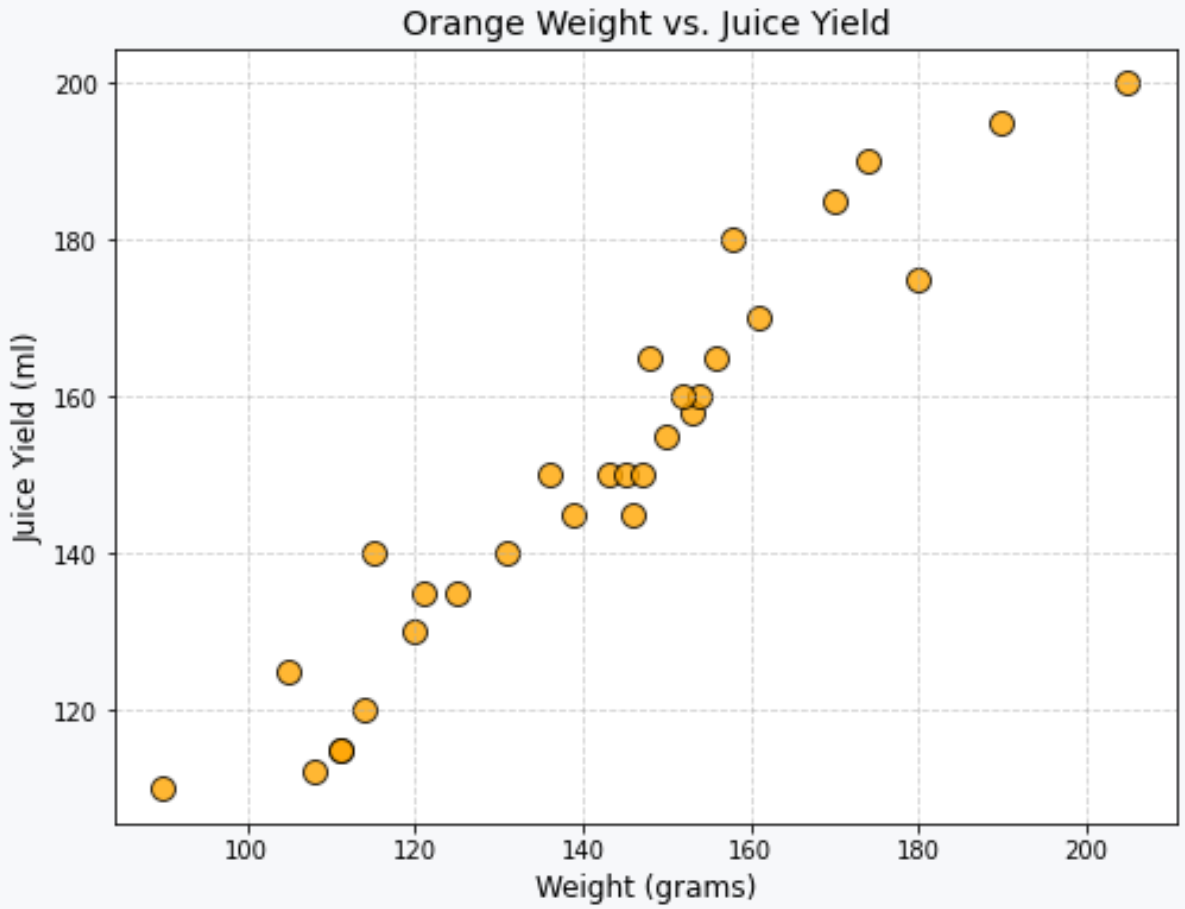
A regression model looks at the relationship between weight and juice output in the data you provide. It tries to find a pattern - like a line that roughly matches the data

points. Once the model learns this pattern, you can use it to predict the amount of juice for any new orange based on its weight.

Why use regression? It helps you predict numerical values, like the amount of juice. Even if it's not perfect, it saves time by using only the most important feature

```
import matplotlib.pyplot as plt
import random

# Define the dataset of weights and juice yields
weights = [153, 143, 120, 105, 115, 90, 121, 146, 180, 114,
           111, 154, 108, 111, 150, 145, 125, 156, 152, 170,
           147, 161, 139, 131, 190, 158, 148, 136, 174, 205]
juice_yield = [158, 150, 130, 125, 140, 110, 135, 145, 175,
120,
               115, 160, 112, 115, 155, 150, 135, 165, 160,
185,
               150, 170, 145, 140, 195, 180, 165, 150, 190,
200]

# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(weights, juice_yield, s=100, color='orange',
edgecolor='black', alpha=0.8)
plt.title("Orange Weight vs. Juice Yield", fontsize=14)
plt.xlabel("Weight (grams)", fontsize=12)
plt.ylabel("Juice Yield (ml)", fontsize=12)
plt.grid(linestyle='--', alpha=0.6)
plt.show()
```

Program output:



3.3.2

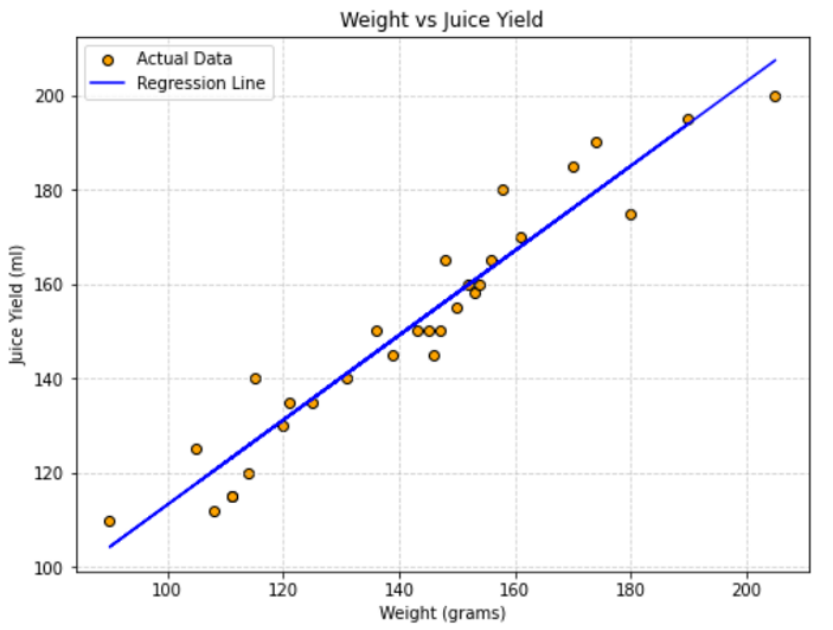
Why would you use regression to predict juice output from orange weight?

- To estimate a numerical value like juice amount
- To measure the weight of oranges
- To predict a category
- To clean the dataset

3.3.3

Linear regression

Linear regression is a simple method used in machine learning to predict a **numerical value** (like the amount of juice from an orange) based on one or more input features (like the weight of the orange). It finds the **best straight line** that represents the relationship between the inputs and outputs.



If you have a scatter plot showing orange weights on the x-axis and the amount of juice on the y-axis and the points might not form a perfect line, we can find there a trend - heavier oranges generally produce more juice. Linear regression finds the line that best fits this trend.

The line can be described using a mathematical equation:

y = m*x + b

Where:

- **y** is the output we want to predict (juice yield).
- **x** is the input feature (weight of the orange).
- **m** is the slope of the line, showing how much y changes for every 1-unit increase in x.
- **b** is the y-intercept, the value of y when x = 0

For example, the equation might be:

Juice Yield = 0.9 × Weight + 20

This means that for every gram increase in weight, the juice yield increases by 0.75 ml, starting from 10 ml when the weight is 0 (hypothetically).

Linear regression is useful because:

1. **It summarizes data** - shows the overall trend in a way that's easy to understand.
2. **It makes predictions** - you can use the line to predict juice yield for new oranges based on their weight.

For example, if an orange weighs 200 grams, you can calculate:

Juice Yield = 0.9 × 200 + 20 = 200 ml

3.3.4

What does linear regression do?

- It finds a line that represents the relationship between input and output
- It divides data into categories
- It measures the weight of oranges
- It removes outliers from the data

3.3.5

Project: Predicting juice yield from orange weight

Apply linear regression to find the best-fit line describing the relationship between the weight of oranges and the amount of juice they produce.

1. Understanding the dataset

- The dataset contains 30 examples of oranges with their weights (in grams) and the amount of juice produced (in milliliters).
- Input feature: Weight
- Output: Juice Yield

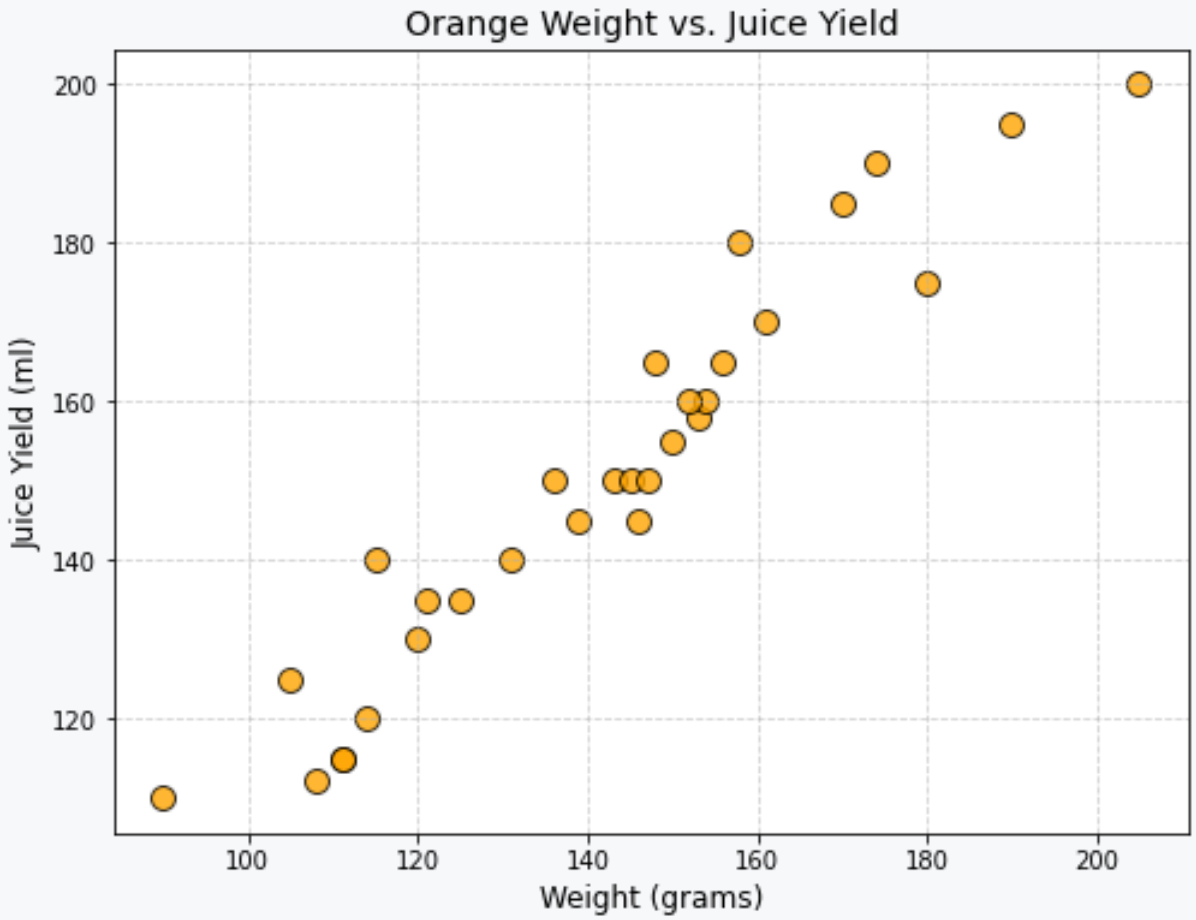
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import numpy as np

weights = [153, 143, 120, 105, 115, 90, 121, 146, 180, 114,
           111, 154, 108, 111, 150, 145, 125, 156, 152, 170,
           147, 161, 139, 131, 190, 158, 148, 136, 174, 205]

juice_yield = [158, 150, 130, 125, 140, 110, 135, 145, 175,
120,
               115, 160, 112, 115, 155, 150, 135, 165, 160,
185,
               150, 170, 145, 140, 195, 180, 165, 150, 190,
200]
```

2. Exploratory data analysis

- Plot the data points on a scatter plot to observe the relationship between weight and juice yield.

```
# Create the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(weights, juice_yield, s=100, color='orange',
edgecolor='black', alpha=0.8)
plt.title("Orange Weight vs. Juice Yield", fontsize=14)
plt.xlabel("Weight (grams)", fontsize=12)
plt.ylabel("Juice Yield (ml)", fontsize=12)
plt.grid(linestyle='--', alpha=0.6)
plt.show()
```

Program output:



3. Apply linear regression

- Use Python libraries to perform linear regression Import LinearRegression from sklearn.linear_model and fit the model to the data (weights and juice yield).
- Extract the equation of the line (y = mx + b).

In Python, libraries like scikit-learn require the input data to have a specific shape. The LinearRegression model expects the input features (x) to be in a 2D array format. For linear regression, even if there's only one feature (like weight), it still needs to be structured as a 2D array of shape (30, 1) - 30 rows (one for each orange), 1 column (weight as the feature).

```
weights = np.array(weights).reshape(-1, 1)

# Step 1: Fit Linear Regression Model
model = LinearRegression()
model.fit(weights, juice_yield)

# Step 2: Get the line equation
slope = model.coef_[0]
intercept = model.intercept_
print(f"Line Equation: Juice Yield = {slope:.2f} * Weight +
{intercept:.2f}")
```

Program output:

```
Line Equation: Juice Yield = 0.90 * Weight + 23.68
```

4. Make predictions

- Use the model to predict juice yield for weights in the dataset.
- Add new data (e.g., an orange weighing 200 g) and predict its juice yield.

```
# Step 3: Make Predictions
predicted_juice = model.predict(weights)
```

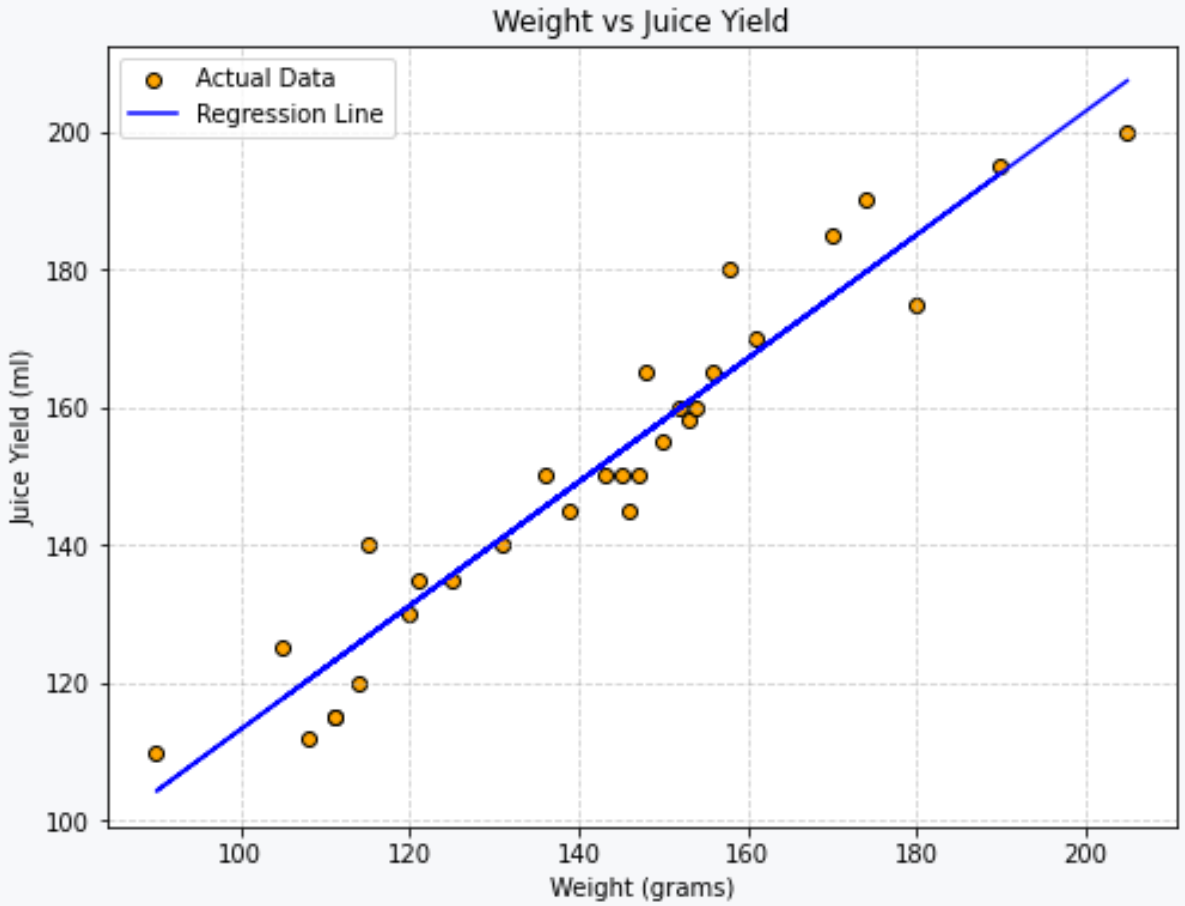
5. Visualize the results

- Plot the original data points along with the regression line.
- Evaluate how well the line fits the data by observing its closeness to the points.

```
# Step 4: Visualize the Data and the Line
plt.figure(figsize=(8, 6))
```

```
plt.scatter(weights, juice_yield, color='orange',
label='Actual Data', edgecolor='black')
plt.plot(weights, predicted_juice, color='blue',
label='Regression Line')
plt.title("Weight vs Juice Yield")
plt.xlabel("Weight (grams)")
plt.ylabel("Juice Yield (ml)")
plt.legend()
plt.grid(alpha=0.6, linestyle='--')
plt.show()
```

Program output:



3.3.6

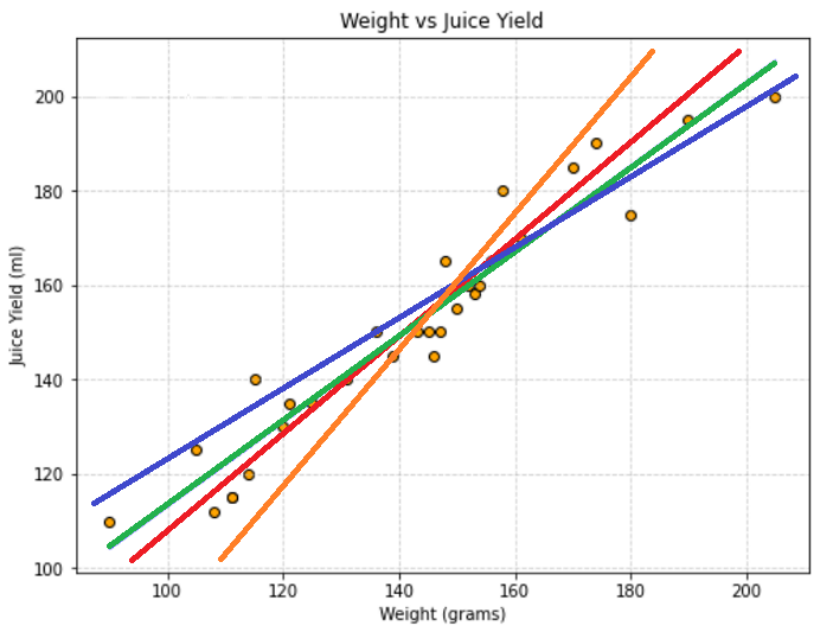
Why do we reshape input data for linear regression in scikit-learn?

- To meet scikit-learn's requirement for a 2D input format
- To improve the accuracy of the model
- To ensure all data points are the same size
- To normalize the input data

🕮 3.3.7

Best possible line

If we use linear regression tools, they will find the best line for us. But how do they go about determining which of the many solutions is the best? What does "best" mean? Imagine there are many lines we could draw through the data points. Some lines might be close to most points, while others might miss the mark entirely. The goal is to find the **one line** that best fits the data.

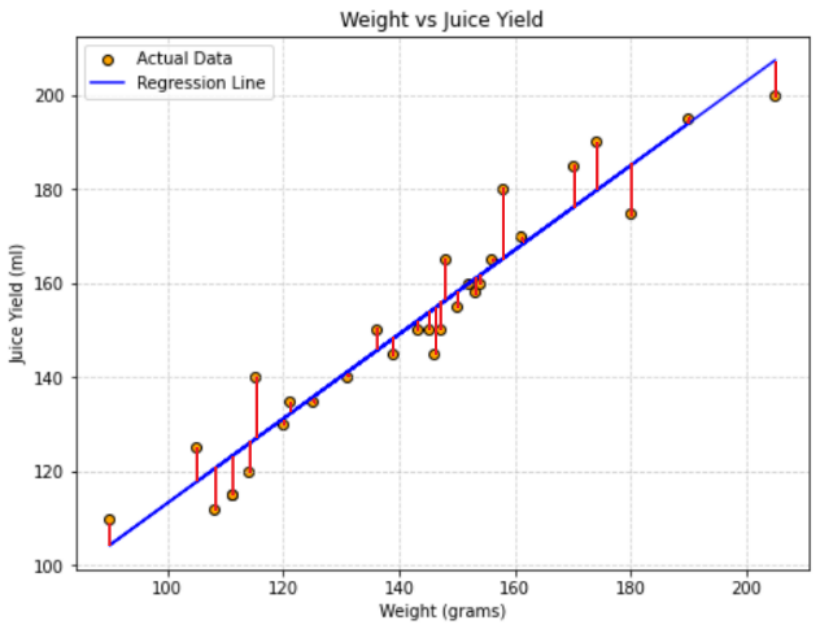


Each line has a slope (m) defines how steep the line is and an intercept defines where the line crosses the y-axis. Changing the slope or intercept gives us a new line. For example:

- One line might go through two points but miss others.
- Another line might pass near most points but not directly through any of them.

There are infinite possibilities for how we can draw a line through the data.

To find the best line, we focus on minimizing the **error**. Error is the difference between the actual value (e.g., juice yield) and the predicted value (what the line says the juice yield should be). For each line, we calculate how far off it is from the actual data points. The best line is the one with the **smallest total error**.



Errors are drawn by red line and is calculated as the sum of all differences between the actual value (point, real value) and the predicted value (point for the same x-value at line)

Error = $y_{actual} - y_{predicted}$

- Square the error to ensure all differences are positive: **Squared Error=$(y_{actual} - y_{predicted})^2$**
- Add up the squared errors for all points: **Total Error = ∑ (Squared Errors)**

This total error tells us how well the line fits the data. The smaller the total error, the better the line.

- Squaring makes all errors positive so they don't cancel each other out.
- It gives more weight to larger errors, which ensures the line fits as closely as possible to all points.

The best line is the one with the **smallest total squared error**. Out of all the possible lines, this line will be the closest overall to the data points.

3.3.8

Why is the best line in linear regression the best :)?

- It minimizes the total squared error
- It has the steepest slope.
- It passes through every data point.
- It maximizes the distance between points and the line.

🕮 3.3.9

Classification and regression

In machine learning, tasks can be broadly divided into two types: **classification** and **regression**. While both involve learning from data, they serve very different purposes. Let’s explore how they differ.

Classification is about assigning **categories** to data. It answers questions like:

- "Is this apple Premium or Regular?"
- "Does this email belong to the 'spam' or 'not spam' category?"

The output of a classification model is **discrete**, meaning it gives specific labels or classes. For example:

- Binary classification for two possible categories (e.g., spam or not spam).
- Multi-class classification for more than two categories (e.g., sorting apples into Premium, Regular, and Reject).

Classification uses models like Decision Trees, Logistic Regression, or Neural Networks to group data into categories based on input features.

Regression is about predicting **numerical values**. It answers questions like:

- "How much juice will this orange produce based on its weight?"
- "What will the temperature be tomorrow?"

The output of a regression model is **continuous**, meaning it can be any number within a range. For example:

- Predicting juice yield (in milliliters).
- Predicting house prices (in dollars).

Regression uses methods like Linear Regression or Polynomial Regression to fit a line (or curve) to the data, allowing predictions of numerical outputs.

When to use which

- Use classification when your goal is to categorize data into distinct groups.
- Use regression when you need to predict a number or continuous value.

For example:

- Sorting apples into Premium and Regular? - classification.
- Predicting how much juice an orange will produce? - regression.

📝 3.3.10

Which of the following are differences between classification and regression?

- Classification predicts categories; regression predicts numerical values.
- Classification answers "what type?"; regression answers "how much?"
- Both always produce the same type of output.
- Classification outputs continuous values; regression outputs categories.

3.4 Learning process

🕮 3.4.1

Learning process

The process of teaching a machine learning model to make predictions is called the learning process. It involves several key steps and components, which work together to help the model understand the data and make accurate predictions.

1. **Data** - every machine learning model starts with data. Data is like the teacher for the model - it provides examples of what to learn. The more data you have, and the better its quality, the more the model can learn. For example in our orange juice example, the data might include the weights of oranges and the amount of juice they produce. The quality and quantity of the training data significantly impact the performance of the model.
2. **Model** is a mathematical or computational representation that captures patterns and relationships in the data. The model is trained on a labelled dataset, where the input data and corresponding desired outputs (labels) are provided. Think of it as the "brain" of the system. For example a linear regression model learns the relationship between weight and juice yield. The model is trained using **labeled data**, where each input (e.g., orange weight) has a corresponding output (e.g., juice yield).
3. **Training** - during the training phase, the model is shown the labeled data. It tries to make predictions and then compares its predictions to the actual outputs. If the prediction is wrong, the model adjusts itself to improve. The learning algorithm tunes the model based on the feedback provided by the training data. For example if the model predicts 150 ml of juice for a 200g orange but the actual yield is 160 ml, it learns from this mistake and updates its internal calculations.
4. **Learning algorithm** is a set of rules and procedures that guide the model's adjustment process during training. It determines how the model updates its parameters to improve its performance. It’s like the instructions a student uses to learn better. For example linear regression uses an algorithm to adjust the slope and intercept of the line to reduce prediction errors.The algorithm ensures the model learns effectively and improves over time.

5. **Testing and evaluation** - once the model is trained, it's important to see how well it works on **new, unseen data**. This is called **testing**. The goal is to check whether the model can make good predictions for data it hasn't encountered before. **Evaluation metrics** help measure the model's performance. For example in regression, we might calculate the average error in predictions to see how close they are to the real values. If the model doesn't perform well, adjustments can be made, such as collecting more data or tweaking the learning algorithm.

The learning process ensures that the model doesn't just memorize the training data but also understands the general patterns.

3.4.2

What are important steps in the machine learning process?

- Training the model with labeled data
- Testing the model on new data to evaluate its performance
- Ignoring unseen data
- Manually adjusting outputs after predictions

3.4.3

What is the role of data in the machine learning process?

- To provide examples for the model to learn patterns from
- To directly calculate the model's outputs.
- To test the model's predictions.
- To reduce the number of features in the dataset.

3.4.4

Which of the following are part of the training phase in machine learning?

- Adjusting the model based on errors
- Using labeled data to teach the model
- Calculating the evaluation metrics.
- Testing the model on unseen data.

3.4.5

Why do we test a model on unseen data after training?

- To check if the model can generalize to new data
- To improve the quality of training data.
- To identify outliers in the training data.
- To reduce the number of input features.

3.4.6

Project: Predicting students' test scores

The goal of this project is to predict students' test scores based on the number of hours they study. This involves using linear regression, a machine learning method that finds the best-fit line to describe the relationship between study hours (input) and test scores (output). The steps include exploring the data, building a regression model, and interpreting the results.

1. Dataset - we create a simple dataset with two columns:

- Hours_Studied - the number of hours a student studied.
- Test_Score - the corresponding test score out of 100.

```
import pandas as pd

# Creating the dataset
data = {
    "Hours_Studied": [4, 5, 3, 8, 9, 12, 10, 11, 14, 12],
    "Test_Score": [30, 50, 35, 58, 56, 54, 65, 62, 65, 60]
}
```

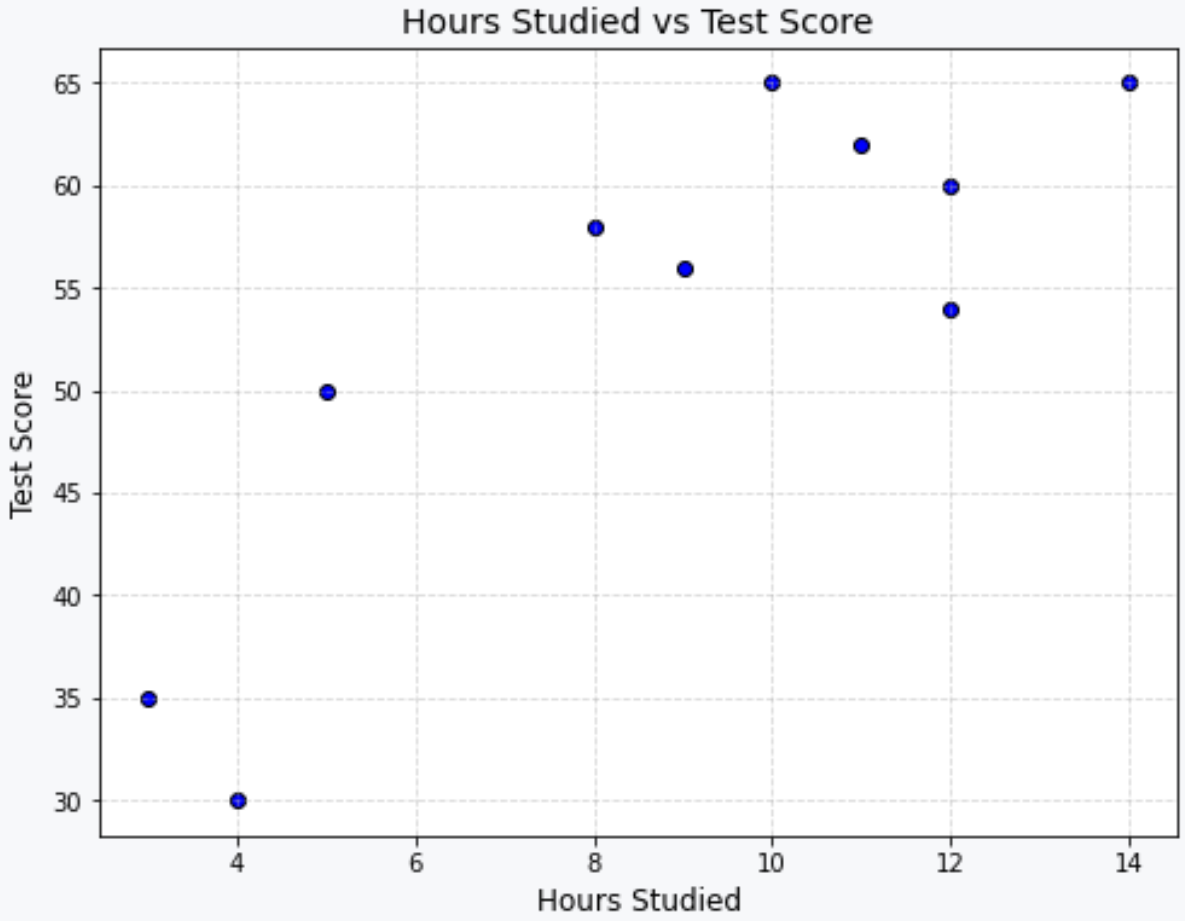
2. Exploratory data analysis

- Visualize the relationship between Hours_Studied and Test_Score using a scatter plot.
- Check for trends, such as whether scores increase as study hours increase.

```
import matplotlib.pyplot as plt

# Convert to DataFrame
df = pd.DataFrame(data)
print(df)

# Scatter plot to visualize the relationship
plt.figure(figsize=(8, 6))
plt.scatter(df["Hours_Studied"], df["Test_Score"],
color="blue", edgecolor="black")
plt.title("Hours Studied vs Test Score", fontsize=14)
plt.xlabel("Hours Studied", fontsize=12)
plt.ylabel("Test Score", fontsize=12)
plt.grid(alpha=0.5, linestyle="--")
plt.show()
```

Program output:

	Hours_Studied	Test_Score
0	4	30
1	5	50
2	3	35
3	8	58
4	9	56
5	12	54
6	10	65
7	11	62
8	14	65
9	12	60



3. Apply linear regression

- Train the model using the dataset.
- Use the model to predict test scores for specific study hours.

- Extract the equation of the best-fit line.

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Preparing the data
X = df["Hours_Studied"].values.reshape(-1, 1)  # Input feature
y = df["Test_Score"].values  # Target output

# Initialize and train the model
model = LinearRegression()
model.fit(X, y)

# Extracting the line equation
slope = model.coef_[0]
intercept = model.intercept_
print(f"Line Equation: Test_Score = {slope:.2f} *
Hours_Studied + {intercept:.2f}")
```

Program output:

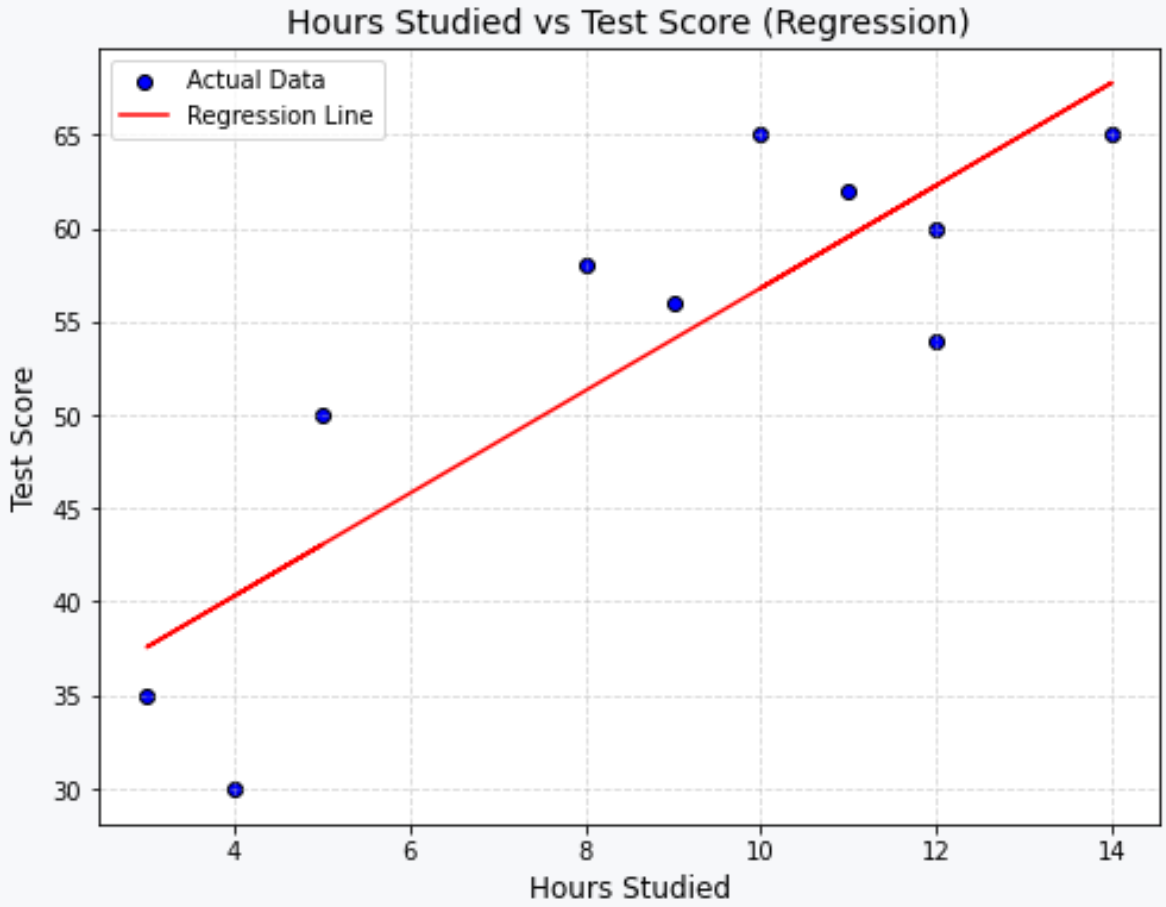
```
Line Equation: Test_Score = 2.75 * Hours_Studied + 29.33
```

4. Visualize the results

- Plot the original data points and overlay the regression line to show the model's fit.

```
# Predicting test scores for given hours
predicted_scores = model.predict(X)

# Plotting the data points and regression line
plt.figure(figsize=(8, 6))
plt.scatter(df["Hours_Studied"], df["Test_Score"],
color="blue", edgecolor="black", label="Actual Data")
plt.plot(df["Hours_Studied"], predicted_scores, color="red",
label="Regression Line")
plt.title("Hours Studied vs Test Score (Regression)",
fontsize=14)
plt.xlabel("Hours Studied", fontsize=12)
plt.ylabel("Test Score", fontsize=12)
plt.legend()
plt.grid(alpha=0.5, linestyle="--")
plt.show()
```

Program output:



3.4.7

Project: Predicting monthly electricity bills

The goal of this project is to predict a household's monthly electricity bill based on two input features:

1. Number of electrical gadgets - the number of electrical devices in the household.
2. Average monthly temperature (°C) - the average daily temperature during the month to cover needs of heating or air conditioning.

We will use multiple linear regression to find the relationship between these features and the electricity bill.

1. Dataset

- Num_Gadgets - number of electrical devices in the household.

- Avg_Temperature - average temperature in degrees Celsius for the month.
- Bill_Amount - the total monthly electricity bill in dollars.

```
import pandas as pd

# Creating the dataset
data = {
  "Num_Gadgets": [5, 8, 12, 15, 20, 25, 30, 35, 40, 50, 10,
18, 22, 28, 32, 38, 45, 12, 25, 50],
  "Avg_Temperature": [15, 20, 25, 30, 35, 10, 5, 25, 20, 15,
18, 22, 24, 27, 33, 40, 5, 14, 20, 28],
  "Bill_Amount":  [123.48, 236.06, 316.91, 374.21, 405.33,
448.45, 502.17, 859.85, 873.97, 1115.73, 190.74, 479.94,
552.89, 543.35, 616.15, 733.62, 838.74, 270.65, 525.30,
967.82]
}
```

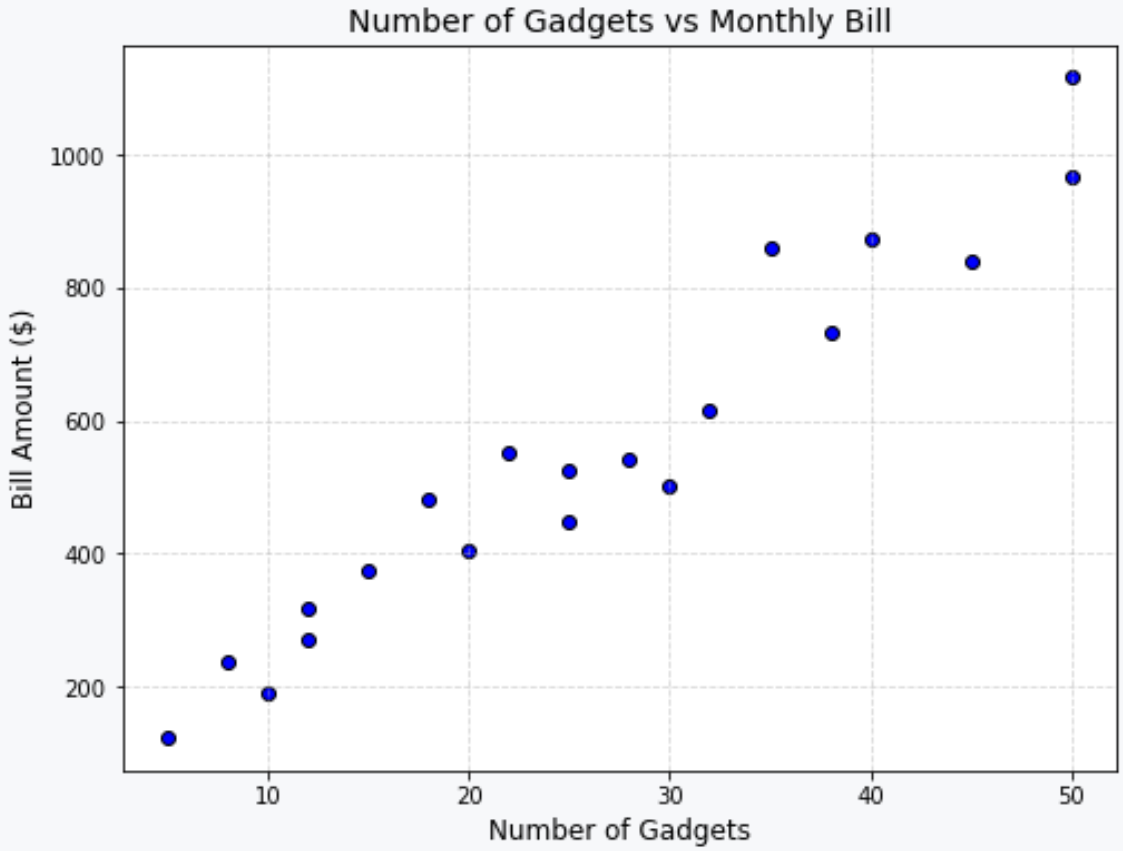
2. Exploratory data analysis

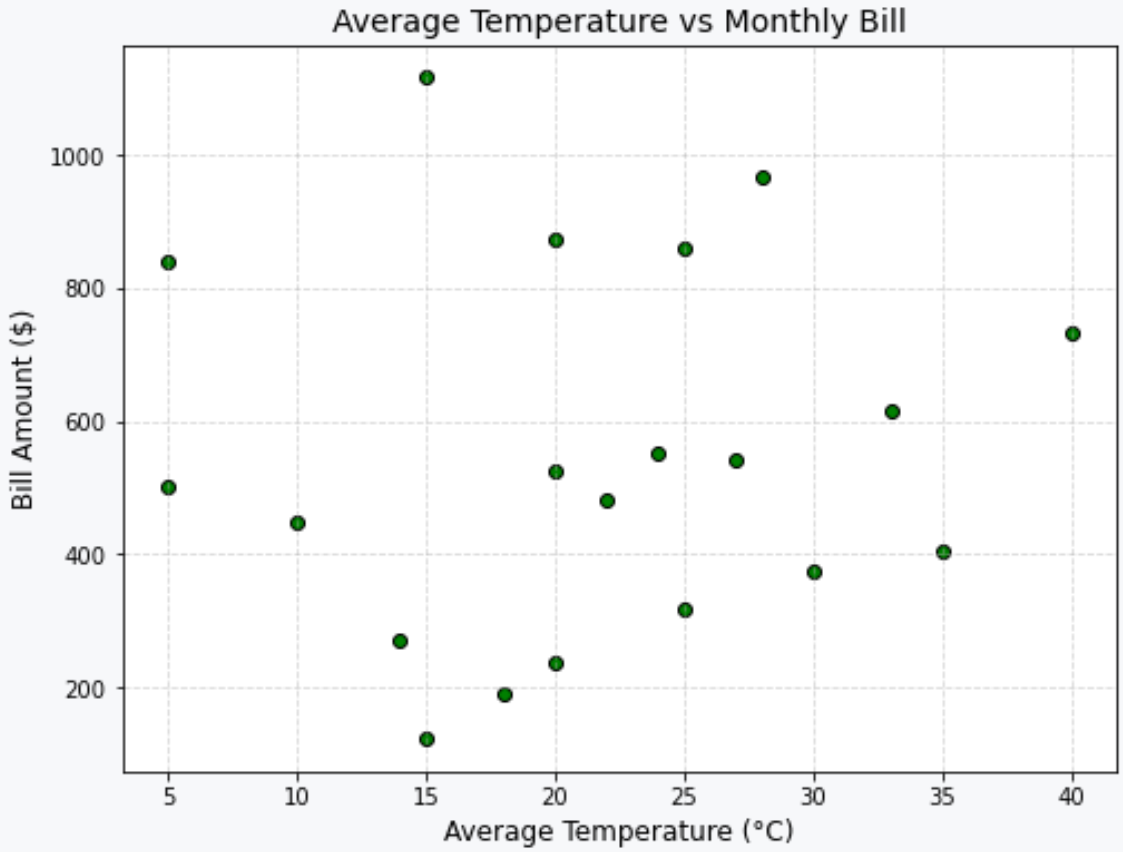
- Visualize the relationship between Hours_Studied and Test_Score using a scatter plot.
- Check for trends, such as whether scores increase as study hours increase.

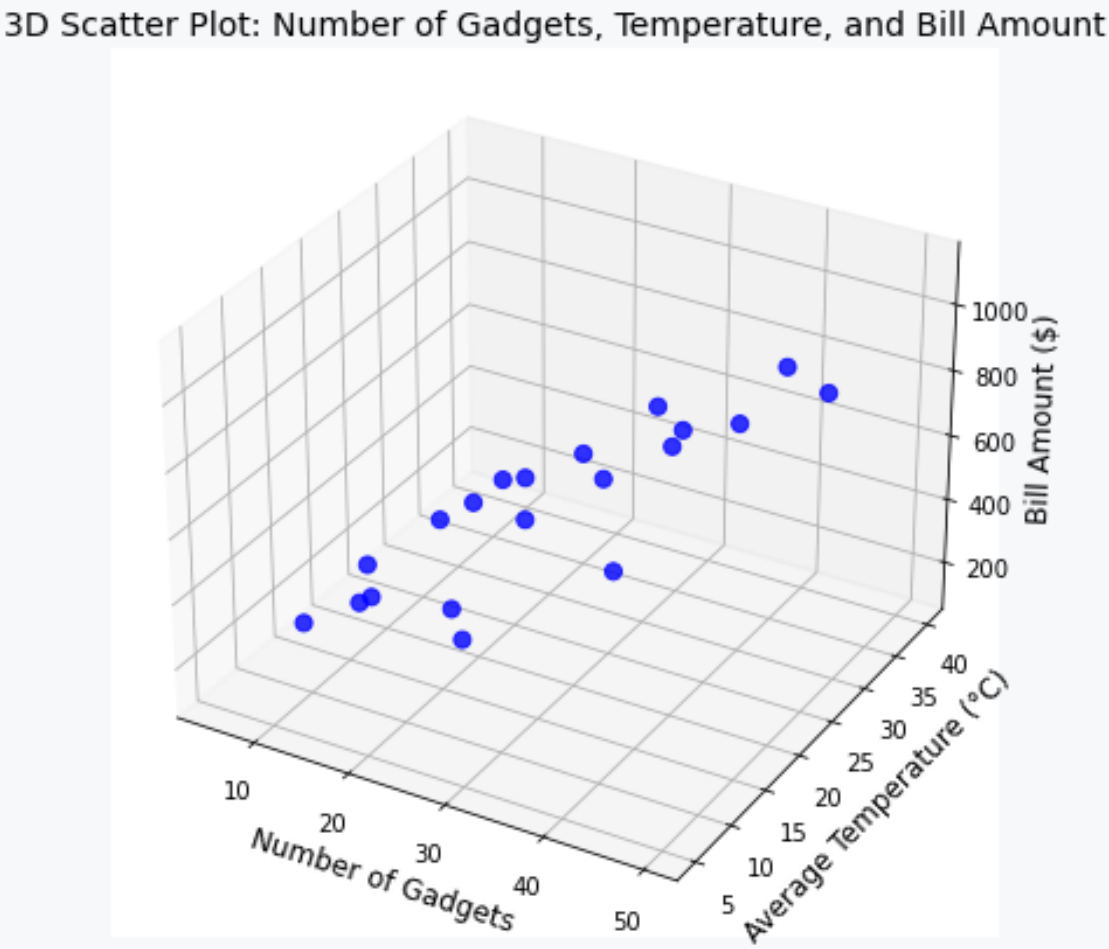
```
import matplotlib.pyplot as plt

# Convert to DataFrame
df = pd.DataFrame(data)
# print(df)

# Scatter plot for Num_Gadgets vs Bill_Amount
plt.figure(figsize=(8, 6))
plt.scatter(df["Num_Gadgets"], df["Bill_Amount"],
color="blue", edgecolor="black")
plt.title("Number of Gadgets vs Monthly Bill", fontsize=14)
plt.xlabel("Number of Gadgets", fontsize=12)
plt.ylabel("Bill Amount ($)", fontsize=12)
plt.grid(alpha=0.5, linestyle="--")
plt.show()

# Scatter plot for Avg_Temperature vs Bill_Amount
plt.figure(figsize=(8, 6))
plt.scatter(df["Avg_Temperature"], df["Bill_Amount"],
color="green", edgecolor="black")
```

```
plt.title("Average Temperature vs Monthly Bill", fontsize=14)
plt.xlabel("Average Temperature (°C)", fontsize=12)
plt.ylabel("Bill Amount ($)", fontsize=12)
plt.grid(alpha=0.5, linestyle="--")
plt.show()

from mpl_toolkits.mplot3d import Axes3D

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
ax.scatter(data["Num_Gadgets"], data["Avg_Temperature"],
data["Bill_Amount"], c='blue', s=50, alpha=0.8)

# Set labels and title
ax.set_title("3D Scatter Plot: Number of Gadgets, Temperature,
and Bill Amount", fontsize=14)
ax.set_xlabel("Number of Gadgets", fontsize=12)
ax.set_ylabel("Average Temperature (°C)", fontsize=12)
ax.set_zlabel("Bill Amount ($)", fontsize=12)

# Show the plot
plt.show()
```

Program output:


Number of Gadgets vs Monthly Bill
Bill Amount ($)
Number of Gadgets
200
400
600
800
1000
10
20
30
40
50


Average Temperature vs Monthly Bill
Bill Amount ($)
Average Temperature (°C)
200
400
600
800
1000
5
10
15
20
25
30
35
40



3. Apply linear regression

- Train the model using the dataset.
- Use the model to predict test scores for specific study hours.
- Extract the equation of the best-fit line.

```
from sklearn.linear_model import LinearRegression

# Preparing the data
X = df[["Num_Gadgets", "Avg_Temperature"]].values  # Input
features
y = df["Bill_Amount"].values  # Target output

# Initialize and train the model
model = LinearRegression()
model.fit(X, y)

# Extracting the regression equation
slope_gadgets = model.coef_[0]
```

```
slope_temperature = model.coef_[1]
intercept = model.intercept_
print(f"Regression Equation: Bill_Amount = {slope_gadgets:.2f}
* Num_Gadgets + "
      f"{slope_temperature:.2f} * Avg_Temperature +
{intercept:.2f}")
```

Program output:

```
Regression Equation: Bill_Amount = 19.16 * Num_Gadgets + 1.60
* Avg_Temperature + 16.34
```

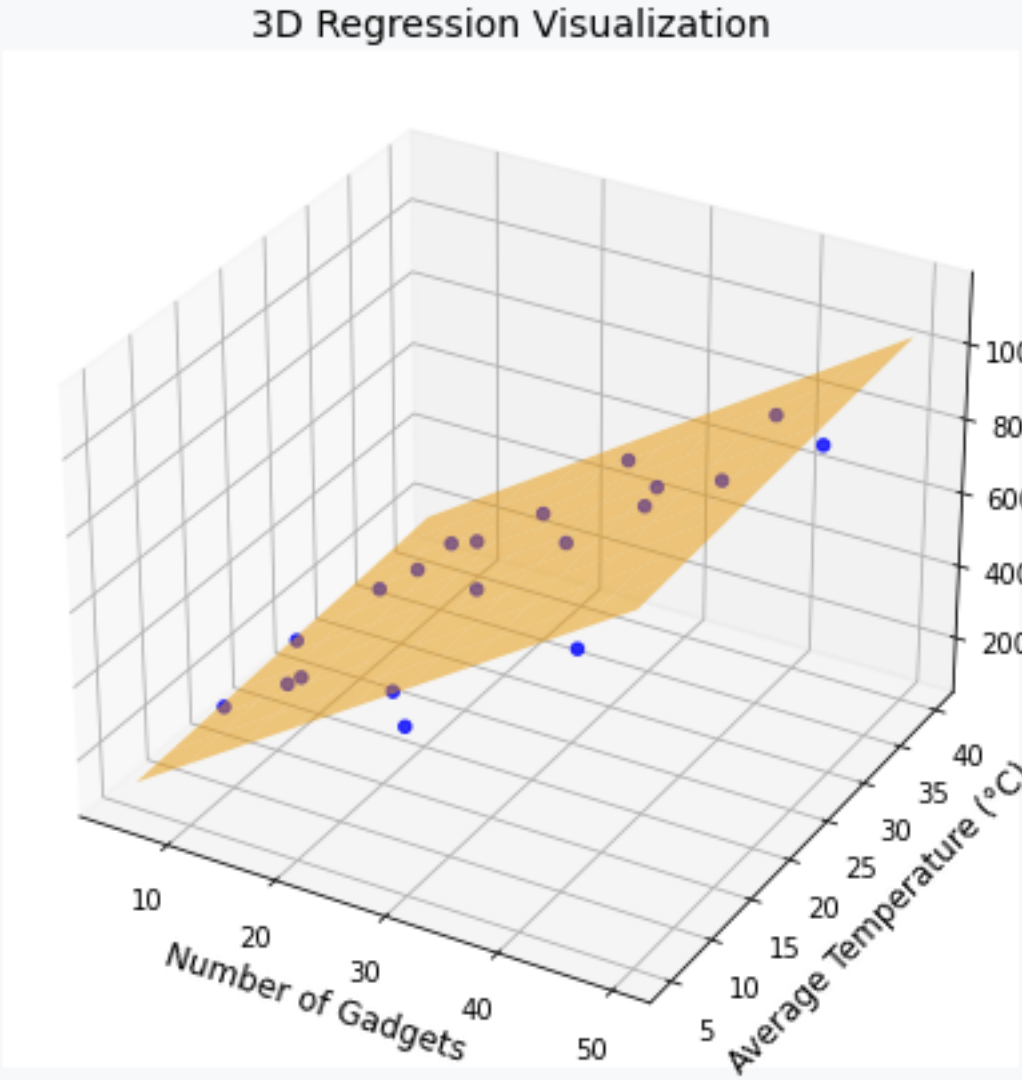
4. Visualize the results

- Plot the original data points and overlay the regression line to show the model's fit.

```
import numpy as np

# Predicting bill amounts for the dataset
predicted_bills = model.predict(X)

# Create grid for surface plot
num_gadgets_range = np.linspace(df["Num_Gadgets"].min(),
df["Num_Gadgets"].max(), 10)
temp_range = np.linspace(df["Avg_Temperature"].min(),
df["Avg_Temperature"].max(), 10)
num_gadgets_grid, temp_grid = np.meshgrid(num_gadgets_range,
temp_range)
bill_pred = model.predict(np.c_[num_gadgets_grid.ravel(),
temp_grid.ravel()]).reshape(num_gadgets_grid.shape)

# 3D Scatter and Surface Plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df["Num_Gadgets"], df["Avg_Temperature"],
df["Bill_Amount"], color='blue', label='Actual Data',
alpha=0.8)
ax.plot_surface(num_gadgets_grid, temp_grid, bill_pred,
color='orange', alpha=0.5)

# Labels and title
ax.set_title("3D Regression Visualization", fontsize=14)
ax.set_xlabel("Number of Gadgets", fontsize=12)
```

```
ax.set_ylabel("Average Temperature (°C)", fontsize=12)
ax.set_zlabel("Bill Amount ($)", fontsize=12)

# Show plot
plt.show()
```

Program output:



3.4.8

Project: Classifying weather as rainy or sunny

Classify weather conditions as either "Rainy" or "Sunny" based on three features:

1. Temperature (°C) - the average temperature of the day.
2. Humidity (%) - the level of moisture in the air.
3. Cloud cover (%) - the percentage of sky covered by clouds.

We have classification task - we will classify result based on 3 features into one of two groups.

1. Dataset - Below is a snippet defining the dataset directly as code:

```
import pandas as pd

# Weather dataset
data = {
    "Temperature": [22, 30, 25, 15, 18, 35, 28, 17, 20, 33,
10, 16, 29, 19, 21, 32, 24, 27, 13, 14],
    "Humidity": [85, 40, 70, 90, 80, 35, 50, 95, 88, 30, 98,
92, 45, 89, 87, 25, 75, 60, 93, 90],
    "Cloud_Cover": [90, 10, 50, 95, 85, 5, 20, 98, 92, 10, 99,
95, 15, 90, 88, 8, 55, 35, 97, 93],
    "Weather": ["Rainy", "Sunny", "Rainy", "Rainy", "Rainy",
"Sunny", "Sunny", "Rainy", "Rainy", "Sunny",,Rainy", "Rainy",
"Sunny", "Rainy", "Rainy", "Sunny", "Rainy", "Sunny", "Rainy",
"Rainy"]
}
```

2. Exploratory data analysis

- Visualize the relationship between Hours_Studied and Test_Score using a scatter plot.
- Check for trends, such as whether scores increase as study hours increase.

```
# Convert to DataFrame
df = pd.DataFrame(data)

# Display the dataset
print(df.info())
```

Program output:

```
RangeIndex: 20 entries, 0 to 19
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Temperature  20 non-null     int64
 1   Humidity     20 non-null     int64
 2   Cloud_Cover  20 non-null     int64
```

```
 3   Weather      20 non-null     object
dtypes: int64(3), object(1)
memory usage: 768.0+ bytes
None
```

3. Data processing

- Split data into features and labels
- Train the model using the dataset.

```
# Step 2: Split data into features (X) and labels (y)
X = df[["Temperature", "Humidity", "Cloud_Cover"]]
y = df["Weather"]

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 4: Train a Decision Tree Classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

4. Make prediction

- and check the model accuracy

```
# Step 5: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 6: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Output results
print("Accuracy:", accuracy)
print("\nClassification Report:\n", report)
```

Program output:

```
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

       Rainy       1.00      1.00      1.00         2
       Sunny       1.00      1.00      1.00         4

    accuracy                           1.00         6
   macro avg       1.00      1.00      1.00         6
weighted avg       1.00      1.00      1.00         6
```

3.4.9

Project: Classifying students' performance

Classify students' academic performance into two categories: "**Pass**" or "**Fail**" based on their study habits:

1. Hours of study per week - the number of hours a student spends studying in a week.
2. Attendance percentage - the percentage of classes a student attends.
3. Homework completion percentage - the percentage of homework completed.

1. Dataset creation

- We have a dataset with 30 students and the above features.

```
import pandas as pd
import numpy as np

# Dataset
data = {
    "Hours_Study": np.random.randint(5, 20, size=30),
    "Attendance": np.random.randint(50, 100, size=30),
    "Homework_Completion": np.random.randint(50, 100,
size=30),
    "Performance": ["Pass" if np.random.random() > 0.3 else
"Fail" for _ in range(30)]
}
```

```
# Create DataFrame
df = pd.DataFrame(data)
print(df.head())
```

Program output:

```
   Hours_Study  Attendance  Homework_Completion Performance
0            5          87                   99        Pass
1           12          98                   86        Pass
2           11          77                   69        Pass
3           10          74                   64        Fail
4           17          52                   93        Pass
```

2. Split data

- Divide the dataset into training and testing sets.

```
from sklearn.model_selection import train_test_split

# Features and Labels
X = df[["Hours_Study", "Attendance", "Homework_Completion"]]
y = df["Performance"]

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

3. Train the model

- Use a classification algorithm (e.g., Decision Tree).

```
from sklearn.tree import DecisionTreeClassifier

# Train a Decision Tree Classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

4. Evaluate the model

```
from sklearn.metrics import accuracy_score,
classification_report

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nClassification Report:\n", report)
```

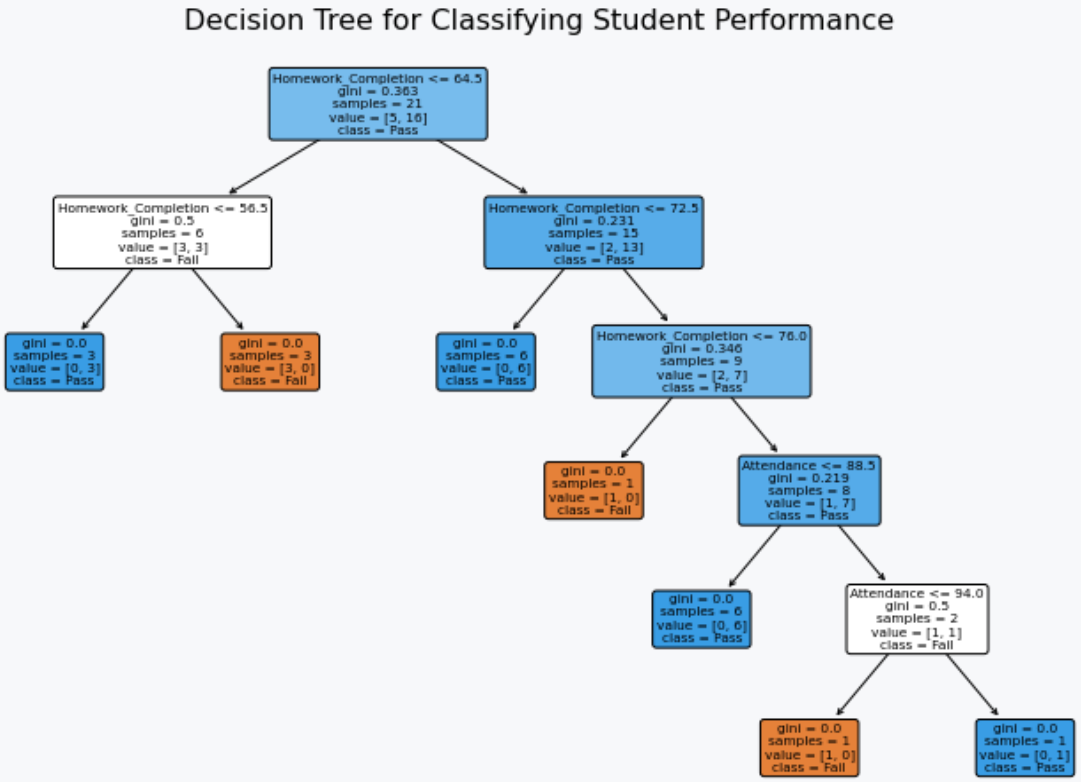
Program output:

```
Accuracy: 0.5555555555555556

Classification Report:
              precision    recall  f1-score   support

        Fail       0.00      0.00      0.00         1
        Pass       0.83      0.62      0.71         8

    accuracy                           0.56         9
   macro avg       0.42      0.31      0.36         9
weighted avg       0.74      0.56      0.63         9
```

5. Visualization

```
import matplotlib.pyplot as plt
from sklearn import tree

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
tree.plot_tree(model, feature_names=["Hours_Study",
"Attendance", "Homework_Completion"],
               class_names=["Fail", "Pass"], filled=True,
rounded=True)
plt.title("Decision Tree for Classifying Student Performance",
fontsize=16)
plt.show()
```

Program output:



3.5 Metrics in machine learning

🕮 3.5.1

Model performance assessment

When we create a machine learning model, it's important to evaluate how well it performs. For regression models, this process is straightforward. A good regression model will predict values that are close to the actual data. If we had no useful information to predict with, we would rely on the **mean model**. This model simply predicts the average of the labels from the training data. A good regression model should perform better than this simple mean model.

One way to check how well a regression model works is by calculating the **mean squared error (MSE)**. The MSE is a number that tells us how far the predicted values are from the actual values. We calculate the MSE for the training data and the test data. If the MSE for the test data is much higher than for the training data, it suggests that the model is **overfitting** - it learned the training data too well and can't generalize to new data. To fix this, we might use techniques like **regularization** or adjust the model's settings, known as hyperparameter tuning.

For classification models, the evaluation methods are a bit different. Classification models are judged using metrics like the **confusion matrix**, **accuracy**, **precision and recall**, and the **area under the ROC curve (AUC)**. These metrics help us understand how often the model is correct and how well it distinguishes between different classes. Each metric has its strengths and is chosen based on the specific problem we're trying to solve.

📝 3.5.2

What does a substantially higher MSE on test data compared to training data indicate?

- The model is overfitting.
- The model is underfitting.
- The test data is perfect.
- The mean model is better.

📝 3.5.3

Which metrics are commonly used to evaluate classification models?

- Precision/Recall
- Confusion matrix
- Mean squared error
- Standard deviation

🕮 3.5.4

Confusion matrix

A confusion matrix is a table used in classification to assess the performance of a machine-learning model. It provides a summary of the model's predictions compared to the actual outcomes for each class in a classification problem. The confusion matrix has four main components:

- True positive (TP) instances where the model predicted the positive class correctly.
- True negative (TN) instances where the model predicted the negative class correctly.
- False positive (FP) instances where the model predicted the positive class, but the actual class is negative.
- False negative (FN) instances where the model predicted the negative class, but the actual class is positive.

Example of a model results for predicting classes spam and not_spam:

```
                          spam (predicted)  not_spam
(predicted)
spam (actual)                  23 (TP)             1 (FN)
not_spam (actual)              12 (FP)             556 (TN)
```

The above confusion matrix shows that of the 24 examples that actually were spam, the model correctly classified 23 as spam. In this case, we say that we have 23 true positives or TP = 23. The model incorrectly classified 1 example as not_spam. In this case, we have 1 false negative, or FN = 1. Similarly, of 568 examples that actually were not spam, 556 were correctly classified (556 true negatives or TN = 556), and 12 were incorrectly classified (12 false positives, FP = 12). The confusion matrix for multiclass classification has as many rows and columns as there are different classes.

The confusion matrix is used to calculate two other performance metrics: precision and recall.

3.5.5

The confusion matrix is used for:

- to assess the performance of a machine learning model. It provides a summary of the model's predictions compared to the actual outcomes for each class in a classification problem.
- to assess the performance of a machine learning model but suitable only for regression problems
- assign class names to classification results

3.5.6

Precision and Recall

When evaluating a classification model, two important metrics are **precision** and **recall**. These metrics help us understand how well the model identifies positive cases without making too many mistakes.

Precision is calculated using the formula:

- **Precision = TP / (TP + FP)**

Precision tells us the ratio of correctly predicted positive cases to the total number of cases predicted as positive. A high precision means fewer false positives.

Recall is calculated using the formula:

- **Recall = TP / (TP + FN)**

Recall shows the ratio of correctly predicted positive cases to all actual positive cases in the dataset. A high recall means fewer false negatives.

For example, in a **spam detection system**, precision is critical to ensure legitimate emails aren't mistakenly labeled as spam (reducing false positives). Recall is less critical in this context since it's okay to allow some spam messages in the inbox rather than risk losing important emails.

In practice, it's hard to maximize both precision and recall at the same time, so a balance is often struck based on the problem's needs. Even for **multiclass classification**, precision and recall can be computed. To do this, treat one class as the "positive" class and all others as "negative", then calculate precision and recall as if it were a binary classification problem.

📝 3.5.7

What does precision measure in a classification model?

- The proportion of correct positive predictions out of all positive predictions.
- The proportion of correct positive predictions out of all true positive examples.
- The proportion of correct predictions out of all predictions.
- The ratio of true negatives to false positives.

📝 3.5.8

Why is precision important in a spam detection system?

- To avoid misclassifying legitimate messages as spam.
- Because it minimizes false positives.
- To minimize false negatives.
- To ensure every spam message is caught.

🕮 3.5.9

Accuracy

Accuracy is a simple and widely used metric to evaluate classification models. It measures the proportion of correctly classified examples out of all the examples. The formula for accuracy is:

Accuracy = (TP + TN) / (TP + TN + FP + FN), or we can use number of all examples

In terms of the **confusion matrix**, accuracy takes into account all correctly classified examples (both true positives and true negatives) and divides them by the total number of examples.

While accuracy is helpful when all classification errors are equally important, it may not always be the best metric. For example, in a **spam detection system**, errors have different impacts. A **false positive** occurs when a legitimate email from your friend is marked as spam. This is a serious issue because you might never see the message. On the other hand, a **false negative** happens when a spam email is mistakenly identified as legitimate. While annoying, this is usually less problematic than losing an important email.

For such scenarios, accuracy alone might not give a complete picture, and metrics like precision and recall are often more useful.

📝 3.5.10

Why might accuracy not always be the best metric for spam detection?

- Because false positives are more problematic than false negatives.
- Because losing important emails is worse than receiving spam.
- Because accuracy ignores the size of the dataset.
- Because accuracy focuses only on true positives.

🕮 3.5.11

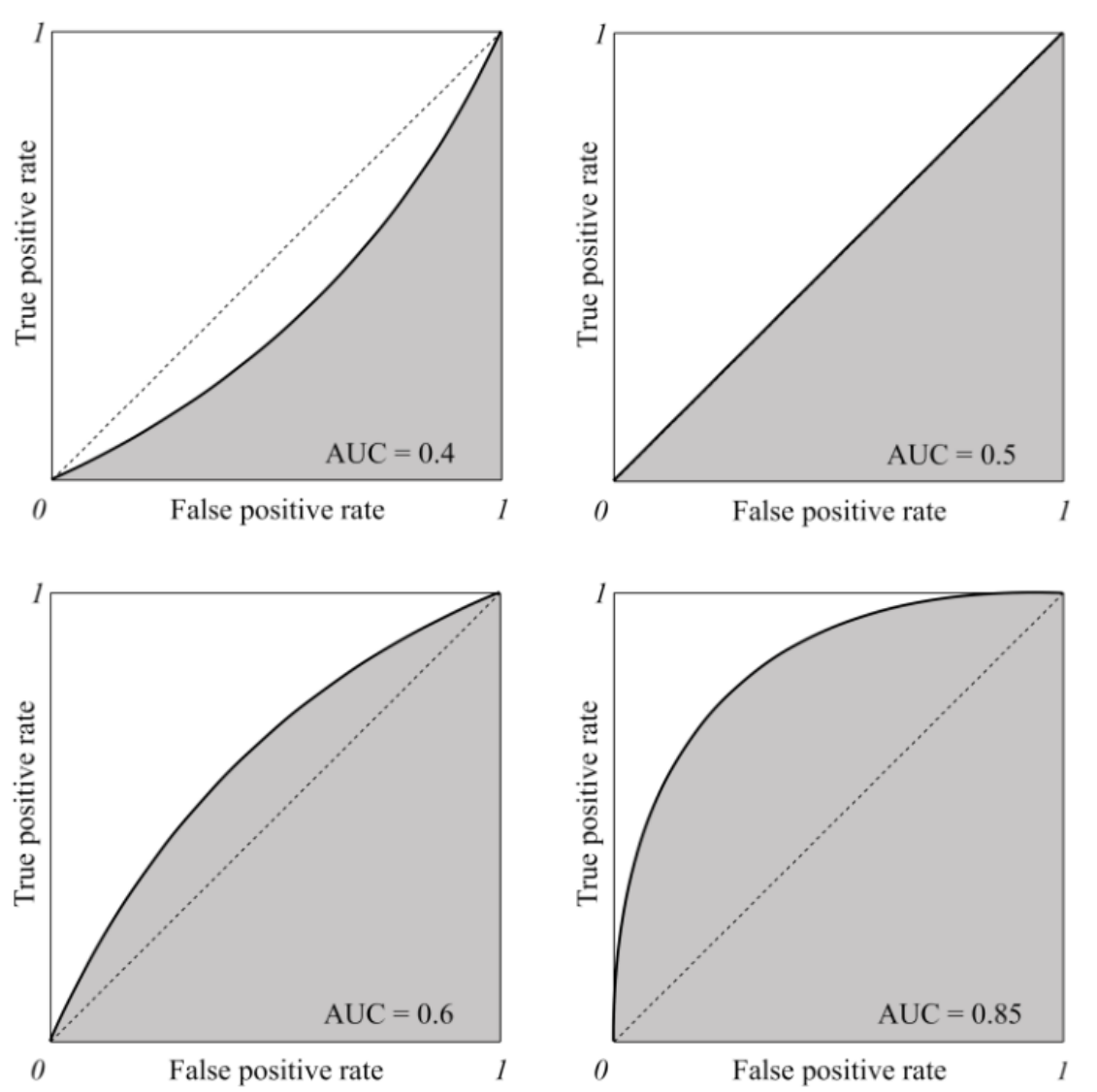
Area under the ROC curve (AUC)

The ROC curve (stands for “receiver operating characteristic;” the term comes from radar engineering) is a commonly used method to assess the performance of classification models. ROC curves use a combination of the true positive rate (defined exactly as recall) and the false positive rate (the proportion of negative examples predicted incorrectly) to build up a summary picture of the classification performance. The true positive rate (TPR) and the false positive rate (FPR) are respectively defined as,

TPR = TP / (TP + FN)

FPR = FP / (FP + TN)

ROC curves can only be used to assess classifiers that return some confidence score (or a probability) of prediction. For example, logistic regression, neural networks, and decision trees (and ensemble models based on decision trees) can be assessed using ROC curves.



If you want to compute TPR and FPR for the threshold equal to 0.7, you apply the model to each example, get the score, and, if the score is higher than or equal to 0.7, you predict the positive class otherwise, you predict the negative class.

The higher the area under the ROC curve (AUC), the better the classifier. A classifier with an AUC higher than 0.5 is better than a random classifier. If AUC is lower than 0.5, then something is wrong with your model. A perfect classifier would have an AUC of 1. Usually, if your model behaves well, you obtain a good classifier by selecting the value of the threshold that gives TPR close to 1 while keeping FPR near 0.

ROC curves are popular because they are relatively simple to understand, they capture more than one aspect of the classification (by taking both false positives and negatives into account) and allow visually and with low effort comparing the performance of different models.

3.5.12

What does a perfect classifier's AUC value equal?

- 1
- 0.5
- 0.7
- 0.9

3.5.13

What are advantages of using ROC curves?

- They help compare the performance of different models.
- They consider both false positives and false negatives.
- They calculate the accuracy of the model.
- They work with any kind of classifier, even without confidence scores.

Fundamental ML Algorithms

Chapter 4

4.1 Fundamental algorithms

📖 4.1.1

Fundamental algorithms

Machine learning algorithms are like tools that help computers learn patterns from data. Once a computer learns these patterns, it can make decisions or predictions based on new information. Let's explore some of the most common and fundamental algorithms with unique strengths and suited for different types of tasks.

- **Linear regression** is one of the simplest machine learning algorithms. It is used for problems where we want to predict a value, like house prices or temperatures. The algorithm finds a straight line that best fits the data points, helping us understand and predict relationships between variables. For example, if you know how study hours affect test scores, linear regression can predict the score based on the hours studied.
- **K-Nearest neighbors (KNN)** works like a good friend giving advice. If you're unsure about something, you look at similar situations around you. KNN uses this idea: it looks at the closest data points (its "neighbors") to decide how to classify new data or make predictions. For example, if you're trying to identify if an image is a cat or a dog, KNN will compare it to images of cats and dogs it has seen before.
- The **Bayesian classifier** is based on probability. It predicts the most likely category for new data by calculating the chances for each possible outcome. This method is great for tasks like spam detection, where it looks at features (like specific words in an email) and calculates the likelihood of the email being spam or not.

📝 4.1.2

Which of the following algorithms is based on probability?

- Bayesian Classifier
- K-Nearest Neighbors
- Linear Regression
- Decision Trees

📝 4.1.3

Which algorithms can be used for classification tasks?

- K-Nearest Neighbors
- Bayesian Classifier
- Decision Trees
- Linear Regression

🕮 4.1.4

Linear regression

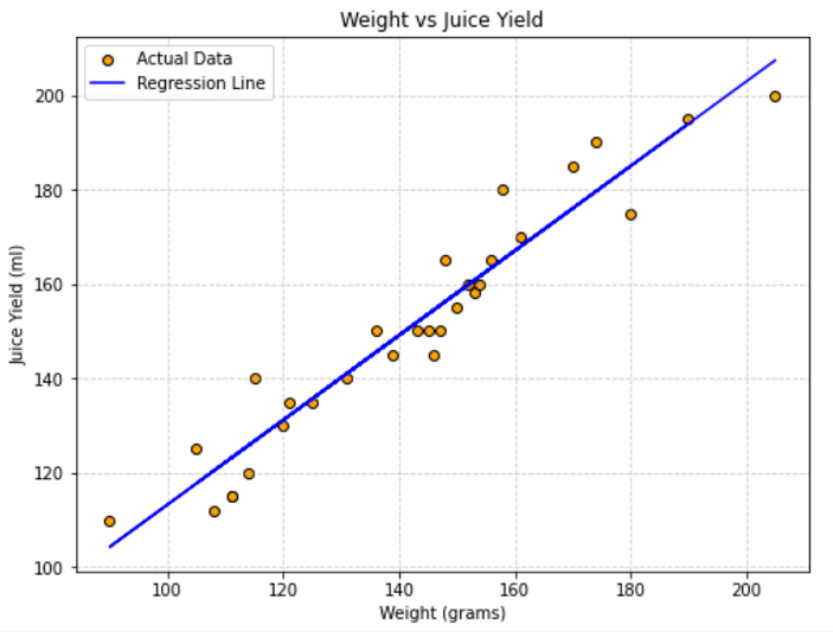
Linear regression is a supervised machine learning algorithm used for predicting a continuous outcome variable (also called the dependent variable) based on one or more predictor variables (independent variables). The relationship between the input variables and the output variable is modelled as a linear equation, which is a straight line in the case of simple linear regression (one predictor variable) and a hyperplane in the case of multiple linear regression (more than one predictor variable).

The general form of a linear regression equation for simple linear regression is:

$y = m * x + b$

where:

- y is the dependent variable (the variable we want to predict),
- x is the independent variable (the input feature),
- m is the slope of the line (how much y changes for a unit change in x),
- b is the y-intercept (the value of y when x is 0).



For multiple linear regression with **n** predictor variables, the equation becomes:

$y = b_0 + b_1*x_1 + b_2*x_2 + b_3*x_3 + ... + b_n*x_n$

where:

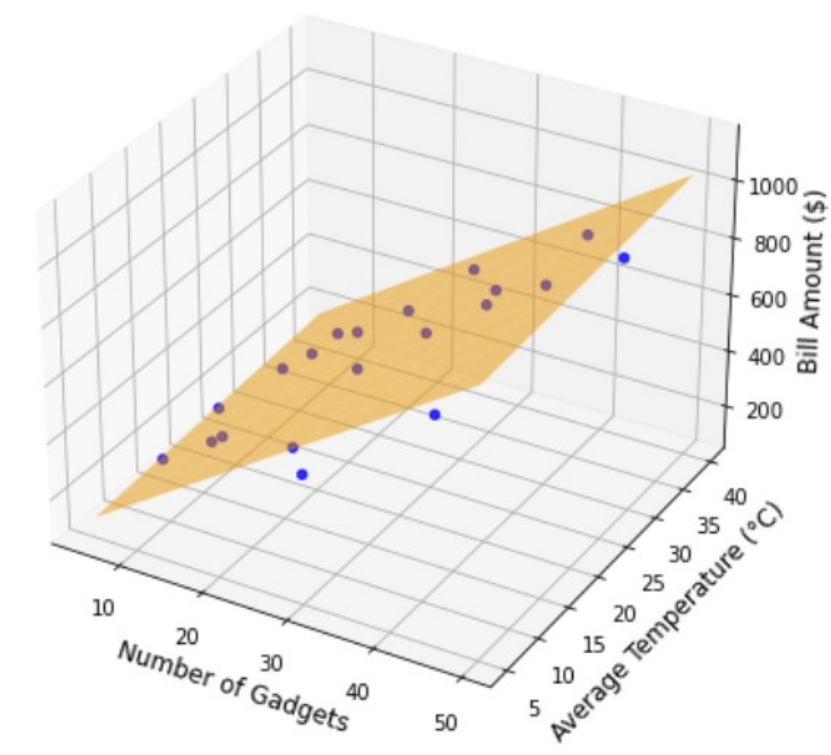
- y is the dependent variable,
- x_1, x_2, ...,x_n are the independent variables,
- b_0 is the y-intercept,

- b_1, b_2, …, b_n are the coefficients representing the contribution of each predictor variable to the outcome.

The goal of linear regression is to find the values of the coefficients (m and b in simple linear regression, and b_0, b_1, …, b_n

In multiple linear regression) that minimizes the sum of the squared differences between predicted and actual values.

The example for 2 variables is in a following picture.



4.1.5

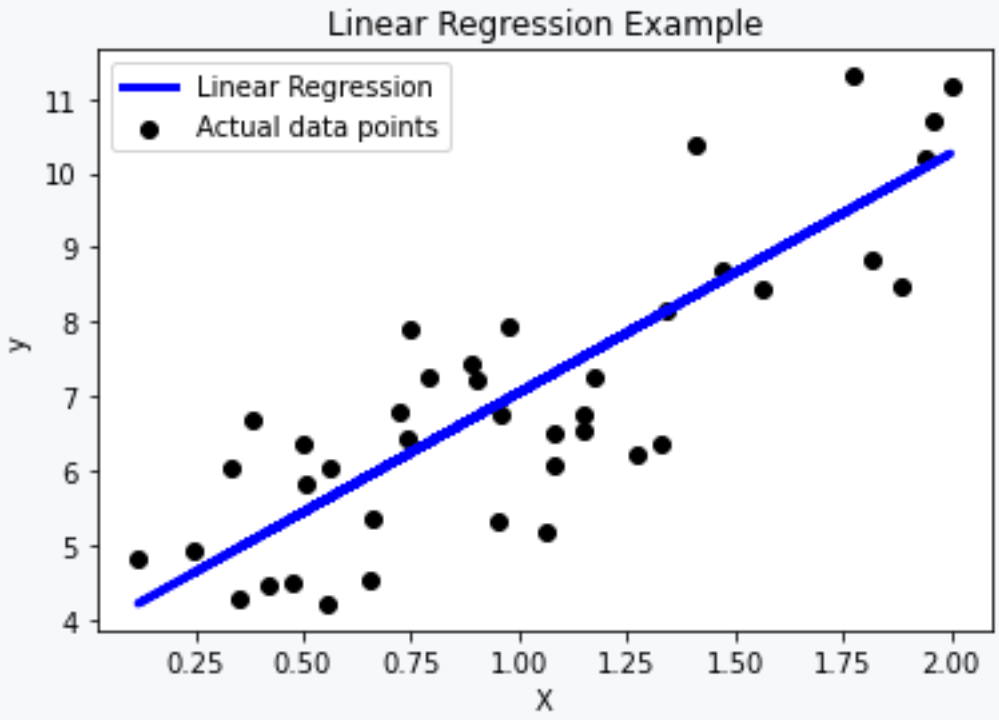
Linear regression example

This is a template of simple Linear regression example using sklearn library in python with 100 randomly generated values.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate some synthetic data for demonstration
# np.random.seed(42)
X = 2 * np.random.rand(200, 1)
y = 4 + 3 * X + np.random.randn(200, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Linear Regression model
lin_reg = LinearRegression()

# Train the model on the training data
lin_reg.fit(X_train, y_train)

# Make predictions on the test data
y_pred = lin_reg.predict(X_test)

# Plot the data points and the regression line
plt.scatter(X_test, y_test, color='black', label='Actual data
points')
plt.plot(X_test, y_pred, color='blue', linewidth=3,
label='Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Example')
plt.legend()
plt.show()

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
```

Program output:



Mean Squared Error: 1.13

4.1.6

Regression is used for

- prediction
- classification
- clustering

4.2 The k-nearest neighbors

4.2.1

The k-nearest neighbors

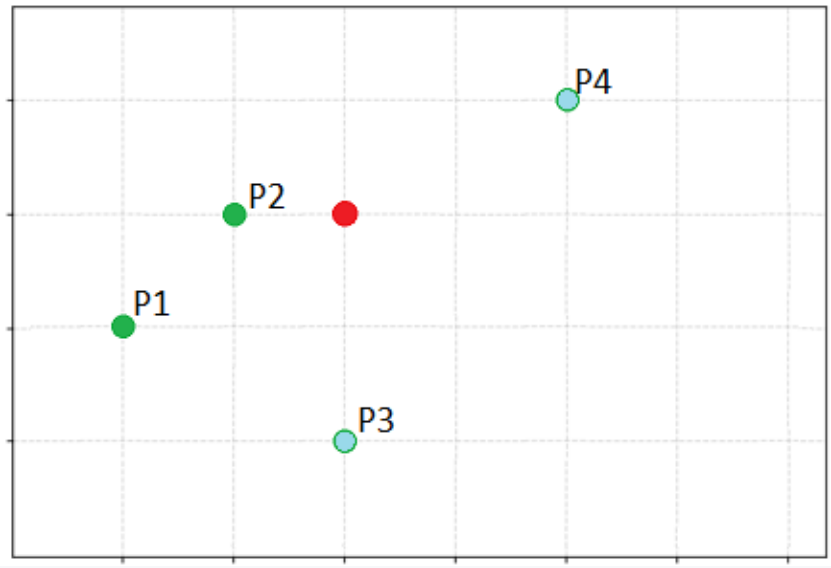
The k-nearest neighbors (KNN) is simple but powerful algorithm. The basic idea of k-NN is to compare something new (a new "data point") to things it already knows (its "neighbors"). It asks: "Who are my closest neighbors?" If most neighbors belong to one group, the new data point is likely to belong to that group too. This process is often used for classification (deciding the category of something) or regression (predicting a number, like a house price).

Key idea is based on measuring "distance" between data points, often using something like Euclidean distance (like a ruler in 2D or 3D space).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Euclidean distance is often used measure how close the points are to each other. Now, consider the following data points with two features (e.g., size and weight) and their classes:

- P1(1,2) labeled "Class A"
- P2(2,3) labeled "Class A"
- P3(3,1) labeled "Class B"
- P4(5,4) labeled "Class B"



We want to classify a new point **P(3,3)**. Let's calculate the distance between PPP and each known point:

1. Distance to P1(1,2) = sqrt((3 − 1)^2 + (3 − 2)^2) = sqrt (4 + 1)= sqrt (5) ≈ 2.24
2. Distance to P2(2,3) = sqrt((3 − 2)^2 + (3 − 3)^2) = sqrt (1 + 0)= sqrt (1) = 1
3. Distance to P3(3,1) = sqrt((3 − 3)^2 + (3 − 1)^2) = sqrt (0 + 4)= sqrt (4) = 2
4. Distance to P4(5,4) = sqrt((3 − 5)^2 + (3 − 4)^2) = sqrt (4 + 1)= sqrt (5) ≈ 2.24

If we set k=3 (the 3 nearest neighbors), we pick P2, P3, and either P1 or P4 (since they tie).

- P2: Class A
- P3: Class B
- P1 or P4: Class A or B.

The majority vote is **Class A**, so the new point P(3,3) is classified as **Class A**.

4.2.2

What is the primary idea behind the k-Nearest Neighbors algorithm?

- Comparing new data points to its closest neighbors.
- Generating random numbers to decide classifications.
- Sorting data points into alphabetical order.
- Using complex math equations to group data points.

📖 4.2.3

Steps for classification with KNN

Classification with KNN follows these steps:

1. Prepare the data - organize the data into two groups – the "training data" (what the computer learns from) and the "test data" (what the computer makes predictions on). Each data point has features (like color, size, or weight) and a label (its category, like "apple" or "orange").
2. Choose the number of neighbors (k) - select how many neighbors to check. A smaller k focuses on the closest data points, while a larger k looks at a wider group.
3. Measure distance for a new data point, calculate how far it is from every point in the training data.
4. Vote on neighbors - identify the k closest neighbors and see which group is the most common.
5. Classify the new point - assign the new point to the most common group.

For example, if a fruit is described by its size and color, and its 5 nearest neighbors are all labeled as "apple“, we classify it as an apple too.

📝 4.2.4

What are steps in using k-nearest neighbors for classification?

- Preparing the data.
- Measuring distance between points.
- Assigning labels randomly.
- Sorting all data alphabetically.

📝 4.2.5

Pass or not using k-nearest neighbors

Let's use k-Nearest Neighbors (k-NN) to predict whether a student will **pass** or **fail** based on their study hours and attendance percentage. We'll use a small dataset for simplicity.

The dataset - here's the data of known students:

Student	Study hours	Attendance(%)	Outcome (Pass/Fail)
A	10	90	Pass
B	8	85	Pass
C	5	70	Fail
D	4	60	Fail

Now, we want to predict the outcome (Pass/Fail) for a new student S who studies for **6 hours** and has **75% attendance**.

We'll calculate the Euclidean distance between the new student and each known student using their study hours and attendance as features.

Distance for students:

d for Student A (10,90) = sqrt{(10 - 6)^2 + (90 - 75)^2}≈_____

for B (8,85) = sqrt{(8 - 6)^2 + (85 - 75)^2}≈_____

for C (5,70) = sqrt{(5 - 6)^2 + (70 - 75)^2}≈_____

for D (4,60) = sqrt{(4 - 6)^2 + (60 - 75)^2}≈_____

Select nearest neighbors

If k=3, the 3 nearest neighbors are:

C with distance 5.1 is Fail

_____ with distance 10.2 is _____

_____ with distance 15.13 is _____

With two "_____" votes and one "_____" vote, the majority prediction is _____.

So, the new student S is predicted to _____.

- B
- D
- 5.1
- Pass
- 15.13
- Fail
- Pass
- 10.2
- Fail
- 15.52
- Fail
- Fail
- Pass
- Pass

4.2.6

Project: Iris dataset

The Iris dataset includes 150 samples of iris flowers with 4 features: sepal length, sepal width, petal length, and petal width. The goal is to classify the samples into one of three species: Iris-setosa, Iris-versicolor, or Iris-virginica.

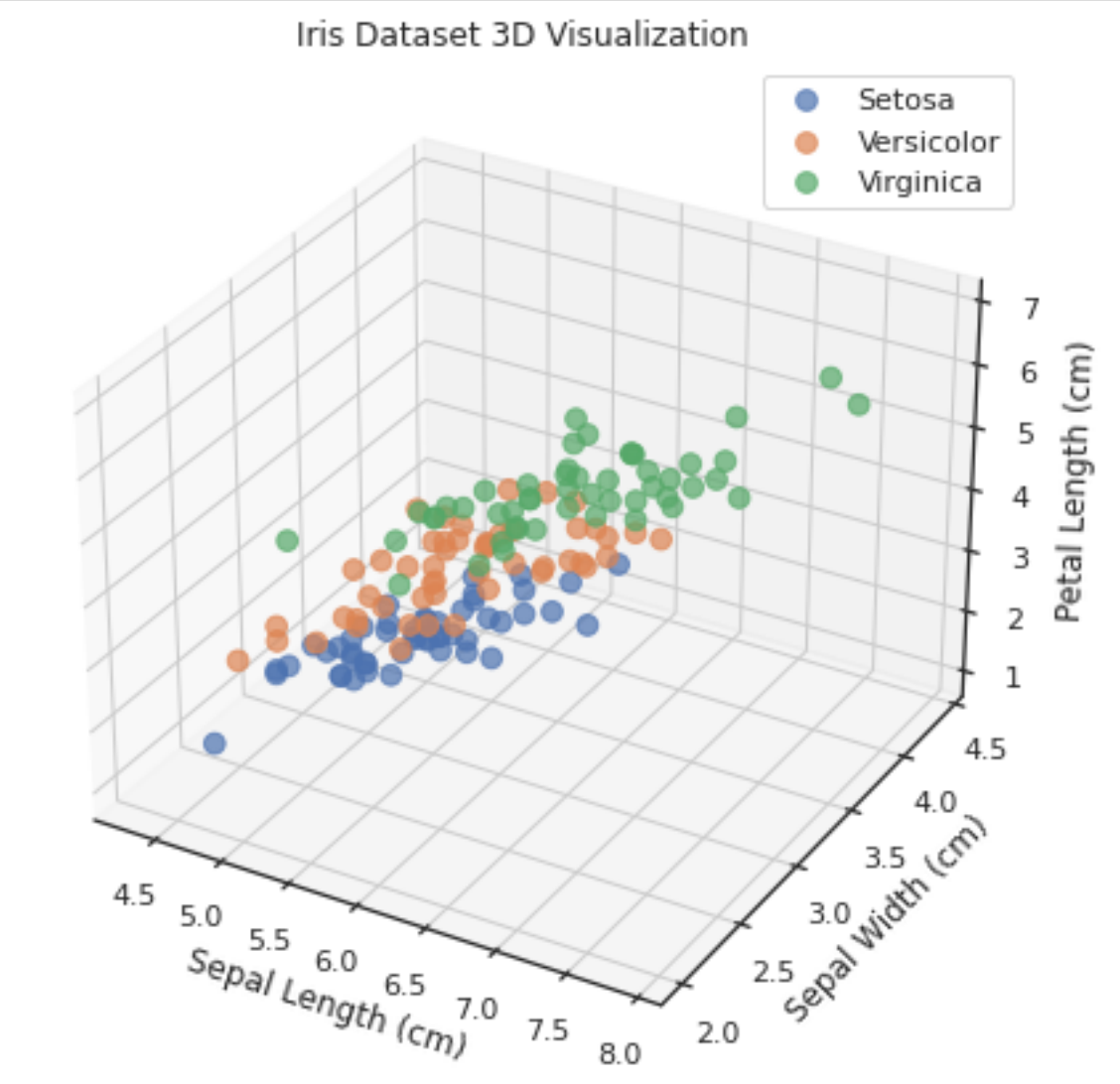
```
# Import necessary libraries
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
import pandas as pd

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features: Sepal length, Sepal width, Petal
length, Petal width
y = iris.target  # Target: Species (0: Setosa, 1: Versicolor,
2: Virginica)

# Convert data into a DataFrame for easier visualization
iris_df = pd.DataFrame(X, columns=iris.feature_names)
iris_df['species'] = y
```

```
# Split the dataset into training and testing sets (80%
training, 20% testing)
# This ensures we can evaluate the model on unseen data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the KNN classifier (using k=3 neighbors)
# The number of neighbors determines how many data points are
considered
# when classifying a new sample.
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Train the classifier on the training data
# The model learns the patterns in the training data to
classify unseen data
knn_classifier.fit(X_train, y_train)

# Make predictions on the test data
# The model uses the patterns it learned to predict species of
flowers in the test set
y_pred = knn_classifier.predict(X_test)

# Evaluate the accuracy of the model
# The accuracy score measures the proportion of correct
predictions
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Visualize the dataset using Matplotlib
# Create a 3D scatter plot to show the relationships among
features
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Scatter points for each species
for species, color in zip(['Setosa', 'Versicolor',
'Virginica'], ['r', 'g', 'b']):
    subset = iris_df[iris_df['species'] == species]
    ax.scatter(
        subset[iris.feature_names[0]],  # Sepal length
        subset[iris.feature_names[1]],  # Sepal width
        subset[iris.feature_names[2]],  # Petal length
        label=species,
        alpha=0.7,
```

```
        s=60
    )

# Add labels and title
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')
ax.set_title('Iris Dataset 3D Visualization')
ax.legend()
plt.show()
```

Program output:

```
Accuracy: 100.00%
```



🕮 4.2.7

Regression with KNN

While KNN is popular for classification, it can also be used for **regression**, which means predicting a number instead of a category. For example, predicting the price of a house based on its size, number of rooms, and location.

Steps for regression with k-NN are similar to classification but with one key difference:

1. Follow the first three steps of classification - prepare the data, choose k, and measure distances.
2. Average the neighbors but instead of voting on the most common category, calculate the average value of the k nearest neighbors.
3. Predict the number as the average value that becomes the predicted number for the new data point.

4.2.8

Project: KNN in regression to find house price

Let's predict the price of a house using KNN regression based on three features:

- Size measured in square meters.
- Number of rooms.
- Distance to city center measured in kilometers

We have the following training data (known houses):

House	Size(m²)	Rooms	Distance CC (km)	Price (in $1000)
A	120	3	5	300
B	150	4	3	400
C	100	2	10	200
D	180	5	2	500
E	140	4	4	350

Now, we want to predict the price of a new house H with the following characteristics:

- Size: 130 m²
- Rooms: 3
- Distance to city center: 6 km

Calculate distances for each house:

1. Distance to House A = sqrt((130 - 120)^2 + (3 - 3)^2 + (6 - 5)^2) ≈10.05
2. B≈20.25
3. C≈30.29
4. D≈50.2

5. E≈10.25

Select neighbors - if k=3, we choose the three closest houses:

- House A (d = 10.05, Price = 300)
- House E (d = 10.25, Price = 350)
- House B (d = 20.25, Price = 400)

Calculate the predicted price - in regression, we take the average price of the nearest neighbors:

Predicted price = (Price of A + Price of E + Price of B) /3 = (300 + 350 + 400) / 3 = 350

So, the predicted price for the new house is **$350,000**.

The predicted price of the house is calculated by averaging the prices of the 3 nearest neighbors. KNN regression works well when the relationship between features and target values can be inferred from nearby examples.

📝 4.2.9

What is a key difference between classification and regression in KNN?

- Regression predicts a number, not a category.
- Regression uses the average value of neighbors.
- Classification involves guessing random numbers.
- Regression sorts data alphabetically.

🕮 4.2.10

Suitability of KNN

KNN is a good choice when the data is simple and easy to understand. It works well for small datasets and doesn't need a lot of training time. However, it can become slow when the dataset is large because it must calculate distances for every new point.

KNN is suitable for problems where the relationships between data points can be represented by distances. However, it is not ideal if the data has a lot of noise (irrelevant information) or if the features are very different in scale. For example, comparing height (in centimeters) with age (in years) might give unfair results unless the data is scaled.

📝 4.2.11

When is k-nearest neighbors most suitable?

- When the dataset is small and relationships can be measured by distance.
- When the dataset is large and requires very fast predictions.
- When the data points are highly noisy.
- When the data involves no numeric values.

4.3 Naive Bayes

📖 4.3.1

Naive Bayes classifier

The Naive Bayes classifier is a type of machine learning algorithm used for classification tasks. This means it helps predict the group (or "class") something belongs to, based on its characteristics (or "features"). For example, it can predict if an email is spam or not spam by looking at the words in it.

This algorithm is based on **Bayes' theorem**, a mathematical formula that calculates how likely something is, given some data. In the case of Naive Bayes, it figures out the probability that a data point (like an email) belongs to a class (like "spam" or "not spam") based on the features it has.

The "naive" part of the name comes from a strong assumption: it assumes that all the features are **independent** of each other. For example, it assumes that the presence of the word "win" in an email is unrelated to the presence of the word "prize“, even though they might actually occur together often in spam emails. Despite this assumption, Naive Bayes still works surprisingly well in many situations.

📝 4.3.2

What does the Naive Bayes classifier predict?

- The probability that a data point belongs to a specific class.
- The exact relationship between all features.
- The speed of the classification process.
- The time it takes to train a machine learning model.

📖 4.3.3

Bayes' theorem

Naive Bayes uses Bayes' theorem, which looks like this:

P(y|X) = P(X|y) * P(y) / P(X)

In this formula:

- P(y|X) is the posterior probability - the probability of the class y given the data X.
- P(X|y) is the likelihood - the probability of the data X given the class y.
- P(y) is the prior probability - how likely the class y is overall.
- P(X)is the evidence - the probability of the data X happening in general.

During the training phase, Naive Bayes calculates these probabilities using the training data. When making predictions, it chooses the class with the highest posterior probability.

For example, in spam filtering, the algorithm might calculate the likelihood of the word "win" appearing in spam emails and compare it to the likelihood of "win" appearing in regular emails. Based on the probabilities, it decides if the email is spam or not.

4.3.4

What components are used in Bayes' theorem?

- Posterior probability P(y|X)
- Likelihood P(X|y)
- Training speed of the algorithm v(x|X)
- Number of features in the dataset

4.3.5

Project: Spam filtering using Naive Bayes

Let’s classify an email as **Spam** or **Not Spam** using the Naive Bayes algorithm. This example calculates the probability that an email is spam or not based on the presence of specific words.

At the beginning we have a dataset of 10 emails with used words transformed into matrix with occurrences of individual words:

Email	Win	Prize	Offer	Label
1	Yes	Yes	Yes	Spam
2	Yes	Yes	No	Spam
3	Yes	No	Yes	Spam
4	No	Yes	Yes	Spam
5	Yes	No	No	Spam
6	No	No	Yes	Not Spam

7	No	Yes	No	Not Spam
8	No	No	No	Not Spam
9	Yes	No	No	Not Spam
10	No	No	Yes	Not Spam

A new email contains the words "Win" and "Offer“. Is it Spam or Not Spam?

We calculate:

```
P(Spam|Win, Offer) = P(Win, Offer|Spam) * P(Spam) /
P(Win, Offer)
```

The formula for **Not Spam** is similar:

```
P(Not Spam|Win, Offer) = P(Win, Offer|Not Spam) * P(Not Spam) /
P(Win, Offer)
```

We will focus on the numerator for each calculation, as the denominator P(Win, Offer) is the same for both and cancels out when comparing probabilities.

Prior Probabilities

- P(Spam) = Number of Spam Emails / Total Emails = 5 / 10 = 0.5
- P(Not Spam) = Number of Not Spam Emails / Total Emails = 5 / 10 = 0.5

Likelihoods P(Win, Offer|Spam)

- P(Win|Spam) = Spam Emails with "Win" / Total Spam Emails = 4 / 5 = 0.8
- P(Offer|Spam) = Spam Emails with "Offer" / Total Spam Emails = 3 / 5 = 0.6
- Assuming independence, P(Win, Offer|Spam) = P(Win|Spam) * P(Offer|Spam) = 0.8*0.6 = 0.48

Likelihoods P(Win, Offer|Not Spam):

- P(Win|Not Spam) = Not Spam Emails with "Win" / Total Not Spam Emails = 1 / 5 = 0.2
- P(Offer|Not Spam) = Not Spam Emails with "Offer" / Total Not Spam Emails = 2 / 5 = 0.4
- Assuming independence, P(Win, Offer|Not Spam) = P(Win|Not Spam)·P(Offer|Not Spam) = 0.2 * 0.4=0.08

Now we can compare probabilities

- P(Spam|Win, Offer)∝P(Win, Offer|Spam)·P(Spam) = 0.48 * 0.5 = 0.24
- P(Not Spam|Win, Offer)∝P(Win, Offer|Not Spam)·P(Not Spam) = 0.08 * 0.5 = 0.04

Since P(Spam|Win, Offer) > P(Not Spam|Win, Offer), the email is classified as **Spam**.

📖 4.3.6

Advantages and limitations of Naive Bayes

Advantages:

1. Simplicity - Naive Bayes is easy to understand and implement, even for beginners in machine learning.
2. Efficiency - it can handle small amounts of training data quickly and still give good results.
3. Good performance - it works well for tasks like text classification, such as spam filtering or sentiment analysis.

Limitations:

1. Assumption of independence - Naive Bayes assumes all features are independent, which is not always true in real-world data.
2. Sensitivity to irrelevant features - if the data includes irrelevant features, the model might not perform as well.
3. Continuous features - some versions of Naive Bayes (like Gaussian Naive Bayes) don't handle continuous data very effectively.

Despite these limitations, Naive Bayes remains popular because it often works surprisingly well, especially in situations like email filtering or diagnosing diseases

📝 4.3.7

What is the advantage of the Naive Bayes classifier?

- It is simple to understand and implement.
- It handles complex continuous data perfectly.
- It never makes assumptions about feature independence.
- It is always the most accurate model available.

📖 4.3.8

Applications of Naive Bayes

Naive Bayes is widely used in several areas because of its simplicity and efficiency:

- Text classification represented by Spam filtering where it determines whether an email is spam or not; and Sentiment analysis where figures out if a review is positive or negative based on the words used.
- Medical diagnosis where it helps predict the presence or absence of a disease based on symptoms. For example, it might calculate the likelihood of flu based on symptoms like fever, sore throat, and fatigue.
- Recommendation systems where it suggests items (like products or movies) to users by analyzing their preferences and behavior.

These are just a few examples. In general, Naive Bayes is a good choice when the data is well-organized and the assumptions of independence are close enough to reality.

4.3.9

Where is Naive Bayes commonly applied?

- Spam filtering.
- Medical diagnosis.
- Sorting numerical data in ascending order.
- Calculating exact distances between data points.

4.3.10

Project: Bayes classifier sentiment analyse example

Create a **sentiment analysis tool** that identifies whether a given sentence is **positive**, **negative**, or **neutral**. Sentiment analysis is a popular application of natural language processing used in areas like customer feedback analysis, product reviews, and social media monitoring.

The sample **dataset** consists of sentences labeled with their corresponding sentiments:

- Positive - sentences expressing favorable or happy feelings (e.g., "I feel great about this").
- Negative - sentences expressing unfavorable or unhappy feelings (e.g., "I do not like this").
- Neutral - sentences expressing neither strong positive nor negative sentiment (e.g., "I have no strong feelings").

The corpus is stored as a list of tuples, where each tuple contains a sentence and its sentiment label.

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
classification_report

# Sample data
corpus = [
    ('This is a positive statement', 'positive'),
    ('I feel great about this', 'positive'),
    ('This is a negative statement', 'negative'),
    ('I do not like this', 'negative'),
    ('Neutral statement here', 'neutral'),
    ('I have no strong feelings', 'neutral')
]

# Split data into features and labels
X, y = zip(*corpus)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Text vectorization

The classifier cannot directly process text, so the sentences are converted into numerical features using CountVectorizer:

- **Tokenization -** breaks the text into individual words (tokens).
- **Bag of words (BoW) - c**reates a matrix where each row represents a sentence, and each column represents a word's occurrence in the corpus.

For example, if the vocabulary consists of words ["this", "is", "a", "positive"], the sentence "This is a positive statement" becomes: [1,1,1,1], The vectorized data is then used as input for the classifier.

```
# Vectorize the text data
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
```

Model training

- Naive Bayes Classifier is trained using MultinomialNB, which is well-suited for text classification tasks, particularly for datasets with word counts (like BoW).

- The model learns the probability distribution of words for each sentiment category based on the training data.

```
# Train the Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train_vectorized, y_train)
```

Making predictions - the trained model predicts the sentiment of unseen sentences (in the test set). For example:

- Input: "I do not like this“.
- Output: Negative

Model evaluation

- Accuracy is the proportion of correctly classified sentences in the test set.
- Classification report provides metrics like precision, recall, and F1-score for each sentiment class, helping assess model performance in detail.

```
# Make predictions on the test set
predictions = clf.predict(X_test_vectorized)

# Example: Evaluating your own sentence
new_sentence = ["I am not sure about this decision"]  #
Replace with your sentence
new_sentence_vectorized = vectorizer.transform(new_sentence)
# Transform the sentence into the BoW format
prediction = clf.predict(new_sentence_vectorized)  # Predict
sentiment
print(f"Predicted Sentiment: {prediction[0]}")  # Display the
prediction

# Evaluate the performance
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report with zero_division parameter
print("Classification Report:")
print(classification_report(y_test, predictions,
zero_division=1))
```

Program output:

```
Predicted Sentiment: negative
Accuracy: 0.00
```

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.00      1.00      0.00       0.0
    positive       1.00      0.00      0.00       2.0

    accuracy                           0.00       2.0
   macro avg       0.50      0.50      0.00       2.0
weighted avg       1.00      0.00      0.00       2.0
```

4.4 Learning types

🕮 4.4.1

Machine learning involves teaching computers to learn from data. Two important types of learning are **supervised learning** and **unsupervised learning**.

In **supervised learning**, the model is trained on a dataset where each input has a known output, called a label. For example, if you're teaching a program to recognize pictures of cats, the dataset might include images labeled as "cat" or "not cat“. The goal is to teach the model to predict the correct label for new, unseen data.

In **unsupervised learning**, there are no labels. The model looks for patterns in the data on its own. For example, it might group similar items together, like putting pictures of cats in one group and dogs in another. This type of learning is often used for clustering and finding hidden patterns.

📝 4.4.2

What is the main difference between supervised and unsupervised learning?

- Supervised learning uses labeled data, while unsupervised learning does not.
- Supervised learning groups data into clusters, while unsupervised learning predicts labels.
- Both require labeled data to find patterns.
- Supervised learning cannot be used for predictions.

🕮 4.4.3

Reinforcement learning is different from supervised and unsupervised learning. Here, the model learns by interacting with its environment. It receives feedback in the form of rewards for good actions and penalties for bad ones. For example, a robot

learning to walk might get a reward for moving forward and a penalty for falling over. Over time, the robot learns the best way to walk to maximize rewards.

Semi-supervised learning combines labeled and unlabeled data. Imagine having a small set of labeled images (e.g., "cat" or "not cat") and a large set of unlabeled ones. The model learns from the labeled data first, then improves its understanding using the unlabeled data.

📝 4.4.4

Which types of learning involve rewards or penalties and use a mix of labeled and unlabeled data?

- Reinforcement learning
- Semi-supervised learning
- Supervised learning
- Unsupervised learning

🕮 4.4.5

Self-supervised learning is when the model creates its own labels from the input data. For example, it might take a sentence, remove a word, and train itself to predict the missing word. This type of learning is often used in tasks like understanding language or generating images.

Active learning involves a model that starts with a small amount of labeled data. The model then chooses new data it finds confusing or uncertain and asks for it to be labeled. This way, it improves quickly with less data. It’s like a student asking questions only when they are stuck, making the learning process more efficient.

📝 4.4.6

What is self-supervised learning?

- The model creates its own labels to learn.
- The model is trained on a mix of labeled and unlabeled data.
- The model actively chooses data for labeling.
- The model only learns from rewards and penalties.

🕮 4.4.7

Combining learning types

In real-world tasks, machine learning often combines these learning types. For example:

- A chatbot might use supervised learning to understand user questions and reinforcement learning to improve its responses.
- A recommendation system could use unsupervised learning to group similar items and active learning to refine its recommendations.

By combining the strengths of different approaches, machine learning can solve complex problems more effectively.

4.4.8

Why might machine learning combine different learning types?

- To solve complex problems.
- To use the strengths of various approaches.
- To avoid using labeled data.
- To reduce the need for algorithms.

Principle of Neural Network

Chapter 5

5.1 Artificial neuron

📖 5.1.1

A **neural network** is a type of computer program that tries to mimic how the human brain works. Just like our brain has neurons connected to each other, a neural network has "artificial neurons" that are connected in layers. These connections help the network learn from data and make predictions.

Imagine you're teaching a friend how to recognize cats. You might show them many pictures of cats and explain what features to look for (like whiskers, tails, or pointy ears). A neural network works similarly. It learns patterns and features in the data you give it, like pictures, text, or numbers.

The **input layer** is where the data enters the network (like a picture of a cat). The **hidden layers** process the data to find patterns, and the **output layer** gives the final answer, such as "cat" or "not cat".

📝 5.1.2

What is the main goal of a neural network?

- To learn patterns from data and make predictions.
- To copy information directly from the human brain.
- To store pictures of cats for later use.
- To create outputs without learning.

📖 5.1.3

Neural and human neuron

Neural networks (NNs) are inspired by how the human brain works. While they don't function exactly the same, they share some similarities in how they process information and learn from it.

Human neuron (Source: OpenClipart)

In your brain, **neurons** are cells that process and transmit information. They are connected to other neurons through pathways called synapses. When a neuron receives a strong enough signal, it "fires" and sends the signal to the next neuron.

In a neural network, **artificial neurons** work similarly:

1. They receive inputs (like numbers or features).
2. They process the inputs using weights and activation functions.
3. They send the output to other neurons if the signal is strong enough.

When your brain learns something new, it strengthens or weakens connections between neurons based on experience. For example, if you practice throwing a ball, your brain strengthens the pathways related to that movement. Similarly, a neural network learns by adjusting the **weights** of connections between artificial neurons during training. This adjustment helps it improve over time, just like practicing helps you get better.

Imagine teaching a child to recognize animals. You show them pictures of cats and dogs and explain the differences. Over time, they learn to recognize them on their own. A neural network does something similar: it looks at many examples, finds patterns, and improves its ability to make correct predictions.

5.1.4

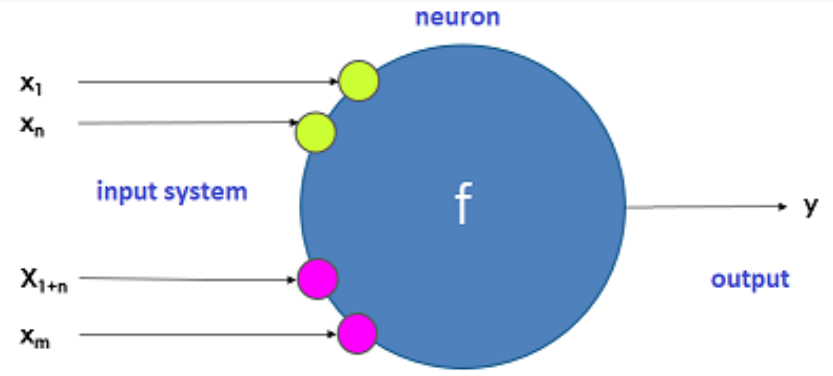
Which aspects of neural networks are inspired by the human brain?

- Neurons passing signals to each other.
- Adjusting connections based on learning.
- Processing signals with DNA.
- Processing signals with DNA.

5.1.5

Neuron in NN

A **neuron** in a neural network is like a decision-maker. It takes **inputs**, processes them, and gives an **output**. Think of it as a box that takes some information, performs a simple calculation, and produces a result.



Imagine a neuron designed to decide if it's raining. The inputs could be:

- Dark clouds - 1 if there are dark clouds, 0 if not.
- Umbrellas seen - 1 if people are carrying umbrellas, 0 if not.
- Wet ground - 1 if the ground is wet, 0 if not.

The neuron processes these inputs and gives an output:

- **1** if it thinks it's raining.
- **0** if it thinks it's not raining.

How the Neuron works:

1. The neuron receives the inputs (e.g., Dark clouds = 1, Umbrellas seen = 1, Wet ground = 0).
2. It processes these inputs by combining them in a simple calculation (like adding them together).
3. If the total passes a certain **threshold** (e.g., 2 or more), the neuron outputs **1** (it's raining). If not, it outputs **0**.

Example:

- Inputs: Dark clouds = 1, Umbrellas seen = 1, Wet ground = 0.
- Calculation: 1 + 1 + 0 = 2
- Threshold: **If total ≥ 2, output = 1** (Rain); otherwise, output = 0 (No Rain).
- Output: 1 (It's raining!).

This simple process helps the neuron decide, and multiple neurons work together in a network to solve complex problems.

5.1.6

What does the output of a neuron represent?

- The result after processing its inputs.
- The size of the network.
- The number of inputs it received.
- The speed of the calculation.

5.1.7

More neurons in NN

A single neuron can make simple decisions, but it's not enough for solving complex problems. Let's see why.

If we're using one neuron to decide if it's raining based on Dark clouds and Umbrellas seen. The neuron works fine for this simple problem because it only needs to process

two inputs. If both are true (e.g., Dark clouds = 1, Umbrellas seen = 1), it outputs "Yes, it's raining".

Now, imagine we want to identify whether a fruit is an apple or a banana. The features (inputs) could be:

1. Shape - is it round?
2. Color - is it red or yellow?
3. Size - is it small or medium?

A single neuron cannot handle this complexity. It would struggle to understand combinations of features, like "round and red" meaning apple or "long and yellow" meaning banana. This is because one neuron only makes basic decisions.

When we use **many neurons**, they work together to handle complex tasks. Each neuron focuses on a part of the problem:

- One neuron might look at the shape.
- Another neuron might focus on the color.
- A third neuron might consider the size.

The **output of these neurons** is combined in the next layer of neurons to make a more accurate decision. Output of first layer of neurons is input of next layer.

5.1.8

Why do we need multiple neurons in a neural network?

- To handle complex problems by focusing on different parts of the data.
- To reduce the size of the input data.
- To speed up individual neurons.
- To replace the need for inputs entirely.

5.1.9

Weights

Weights are one of the most important parts of a neural network. They help the network decide which inputs are more important for making a prediction. Let's explore how weights work using our banana/apple example.

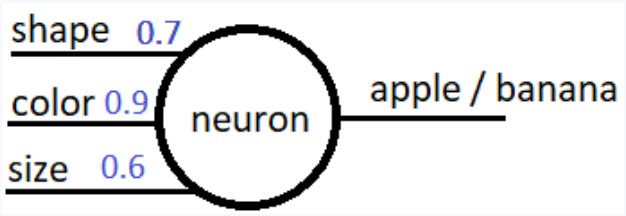
Inputs and weights - imagine we have three inputs into one neuron to help classify fruit:

- Shape - is it round? (1 for Yes, 0 for No)
- Color - is it yellow? (1 for Yes, 0 for No)
- Size - is it small? (1 for Yes, 0 for No)

Each input is connected to a neuron, and each connection has a weight. These weights tell the neuron how important that input is for predicting whether the fruit is an apple or a banana.

Let's make an example and assign weights to the inputs:

- Shape weight = 0.7 (important for apples because they are round)
- Color weight = 0.9 (important for bananas because they are yellow)
- Size weight = 0.6 (important for bananas because they are usually small)



If we input the features of a fruit:

- Shape = 0 (not round),
- Color = 1 (yellow),
- Size = 1 (small),

the neuron calculates a **weighted sum**:

```
WeightedSum = Shape * WeightShape + Color * WeightColor +
Size * WeightSize = 0 * 0.7 + 1 * 0.9 + 1 * 0.6 = 1.5
```

The weighted sum is passed to an **activation function** (explained earlier) to decide the output. If the output is closer to "banana", the network predicts banana. If the output is closer to "apple", it predicts apple.

Why adjust weights? During training, the network adjusts these weights to improve predictions. For example, if it wrongly predicts an apple as a banana, it might lower the weight for "yellow color" for apples and increase the weight for "round shape".

5.1.10

What do weights in a neural network do?

- Show the importance of each input feature.
- Help calculate the weighted sum in neurons.
- Store the final prediction of the network.
- Control the size of the input data.

5.2 Neural network - input processing

🕮 5.2.1

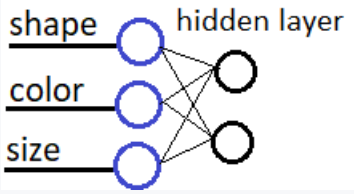
The number of neurons in the input layer depends on the number of features we are using to describe the data. In our banana/apple example, we are using three features:

1. **Shape** - is it round?
2. **Color** - is it yellow?
3. **Size** - is it small?

Each feature is a piece of information the network needs to make a decision. This means we need **one input neuron per feature**. So, in this case, the input layer must have **three neurons**—one for each feature.

Each input neuron represents one aspect of the data:

- The **Shape neuron** only cares about whether the fruit is round.
- The **Color neuron** only processes whether the fruit is yellow.
- The **Size neuron** only considers whether the fruit is small.



These neurons pass their information to the next (hidden) layer, where it gets combined and processed further.

If we decide to add another feature, like **texture** (e.g., smooth or rough), we would need to add a fourth neuron to the input layer. The number of input neurons must match the number of features in the data.

The input layer always has the same number of neurons as the features you are analyzing. Without enough neurons, the network won't get all the information it needs to make accurate predictions.

📝 5.2.2

Why does the input layer need one neuron per feature?

- Each neuron stores the final output prediction.
- Each neuron represents a specific piece of input information.
- More neurons always make the network faster.
- Neurons in the input layer control the training process.

📖 5.2.3

Activation function

The **activation function** is a key part of a neuron. It decides whether the neuron should pass its information to the next layer. Without an activation function, the neural network would only perform simple calculations and couldn't handle complex problems.

Imagine you're deciding if a fruit is an apple or banana based on inputs like shape, color, and size. The inputs are combined in each neuron to create a **weighted sum**. The activation function then checks if the sum is meaningful enough to send forward. For example:

- If the neuron is looking for a "round shape" and the sum matches this feature well, the neuron "fires" and sends a signal forward.
- If the sum doesn't match, the neuron stays silent.

Example of an activation function is **ReLU (Rectified Linear Unit)** and is very common:

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- If the weighted sum is positive, ReLU passes it as-is.
- If the sum is negative or zero, ReLU outputs 0.

ReLU is simple and works well in practice. It allows the network to handle complex data by introducing non-linearity.

📝 5.2.4

What is the main role of an activation function?

- To decide whether the neuron passes its information forward.
- To store data in the network.
- To control the size of the input features.
- To adjust the weights during training.

📖 5.2.5

ReLU application

If a neural network didn't have an activation function, all neurons would just perform simple calculations, like adding up inputs and multiplying by weights. This means the network could only solve very basic problems, like drawing a straight line to separate apples and bananas based on one feature.

But real-world problems, like recognizing fruits based on multiple features, are more complex. For example, bananas are yellow and long, while apples are red and round. These features need to be combined in a way that separates the two types of fruit.

The **ReLU** activation function introduces **non-linearity**. This means it allows the network to create more flexible decision boundaries, like curves, to better separate apples from bananas.

Imagine a neuron in receives three inputs (shape, color, size) and calculates a weighted sum:

- Shape = 1 (round), Color = 0 (not yellow), Size = 1 (small)
- Weighted sum = (1 * 0.8) + (0 * 0.5) + (1 * 0.6) = 1.4

Now, the ReLU activation function is applied:

- ReLU(1.4)=1.4(since 1.4>0)

If the weighted sum had been negative (not in this structure), the ReLU output would be:

- ReLU(−0.5)=0

This ability to "switch off" neurons with negative outputs helps the network focus only on important patterns.

By using ReLU the network can learn complex relationships between features and it can handle patterns that aren't simple straight lines, like the roundness and redness of apples versus the yellowness and length of bananas.

📝 5.2.6

What is one key benefit of the ReLU activation function?

- It introduces non-linearity to solve complex problems.
- It increases the size of the input layer.
- It ensures all outputs are positive, even if not needed.
- It directly calculates weights for the neurons.

🕮 5.2.7

Hidden layer

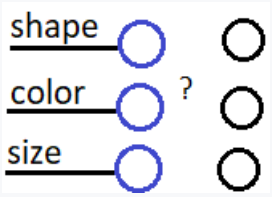
Now that we know how activation functions work, let's decide what happens after the input layer. The next layer is usually called the **hidden layer**, and it plays a crucial role in solving complex problems.

The **input layer** only passes the raw features (e.g., shape, color, size). A hidden layer helps the network learn patterns by combining these features. For example:

- One neuron in the hidden layer might combine shape and size to decide if the fruit is likely an apple.
- Another neuron might combine color and size to decide if it's likely a banana.

The number of neurons in the hidden layer depends on the problem's complexity. For our banana/apple example:

- A hidden layer with **3 neurons** is a good start. Each neuron can focus on a different aspect of the input data (e.g., roundness, yellowness, and smallness).



Each neuron in the hidden layer:

- Takes the outputs from the input layer as its inputs.
- Processes the inputs (weighted sums) and applies an activation function (e.g., ReLU).
- Passes its result to the **output layer**, where the final decision is made (e.g., apple or banana).

📝 5.2.8

Why is the hidden layer important?

- It combines input features to find patterns.
- It directly stores the final prediction.
- It helps the network solve complex problems.
- It controls the size of the input layer.

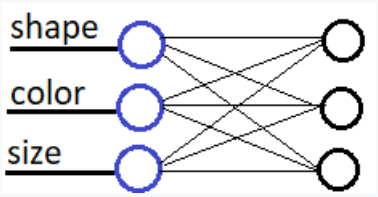
🕮 5.2.9

Let's move on to connecting the **input layer** to the **hidden layer**. This connection involves passing outputs from the input neurons to the hidden neurons, using **weights** to decide the importance of each input feature.

Each input neuron (representing features like shape, color, and size) sends its value to **every neuron in the hidden layer**. For example:

- The output of the **Shape neuron** is sent to all hidden neurons.

- Similarly, the outputs of the **Color** and **Size neurons** are sent to all hidden neurons.



This ensures that every hidden neuron has access to all the input information.

Each neuron in the hidden layer has its own job:

- One neuron might focus on detecting "roundness" (Shape).
- Another might focus on "yellow color" (Color).
- A third might focus on "small size" (Size).

To achieve this specialization, the weights connected to each neuron are set **independently** during training. This allows the network to process the data more effectively. We have three inputs (Shape, Color, Size) and three neurons in the hidden layer:

Hidden Neuron 1 focuses on Shape and Color.

- Weight for Shape = 0.7
- Weight for Color = 0.8
- Weight for Size = 0.3

Hidden Neuron 2 focuses on Color and Size.

- Weight for Shape = 0.2
- Weight for Color = 0.9
- Weight for Size = 0.7

Hidden Neuron 3 focuses on Shape and Size.

- Weight for Shape = 0.9
- Weight for Color = 0.4
- Weight for Size = 0.6

Each hidden neuron calculates a **different weighted sum**, which helps the network understand various patterns in the data.

5.2.10

Why do hidden neurons have different weights?

- To specialize in processing different combinations of inputs.
- To ensure all neurons receive equal contributions from inputs.
- To make the network faster during training.
- To avoid using an activation function.

5.3 Neural network - output

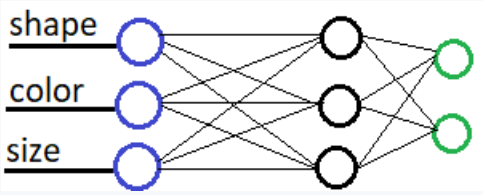
5.3.1

Output layer

Let's see how the hidden layer connects to the **output layer** to make a final prediction.

The output layer is where the network makes its final decision. For our banana/apple example the output layer might have two neurons:

- One neuron predicts the probability of **apple**.
- The other neuron predicts the probability of **banana**.



The network chooses the class (apple or banana) with the higher probability.

Each hidden neuron sends its **processed output** to every neuron in the output layer. Just like before, each connection has a **weight** that determines how much influence the hidden neuron's output has on the final decision. For example:

- If a hidden neuron detects "roundness", it might have a stronger connection to the "apple" neuron in the output layer.
- If another hidden neuron detects "yellow color", it might have a stronger connection to the "banana" neuron.

Each output neuron calculates a **weighted sum** of the inputs it receives from the hidden layer:

```
Weighted_Sum(Output) = (Hidden_Neuron_1_Output * W1) +
(Hidden_Neuron_2_Output * W2) + (Hidden_Neuron_3_Output * W3)
```

Let's assume the hidden neurons produce these outputs:

- Hidden Neuron 1 (roundness) = 1.2
- Hidden Neuron 2 (yellowness) = 0.8
- Hidden Neuron 3 (small size) = 1.5

For the **apple neuron**, the weights might be: W1 = 0.9, W2 = 0.4, W3 = 0.7

The weighted sum for the apple neuron would be: Weighted_Sum(Apple) = (1.2 * 0.9) + (0.8 * 0.4) + (1.5 * 0.7) = 2.45

Similarly, we calculate the weighted sum for the **banana neuron**. The output neuron with the higher weighted sum determines the final prediction.

📝 5.3.2

What is the role of the output layer in a neural network?

- To make the final prediction based on the hidden layer's outputs.
- To calculate probabilities for each class.
- To directly adjust the weights in the input layer.
- To pass raw data back to the hidden layer.

🕮 5.3.3

Softmax

Once the **output layer** calculates its weighted sums, we need to interpret these values as probabilities. This is where the **Softmax function** comes into play.

Softmax is a mathematical function that converts the raw outputs (weighted sums) of the output neurons into probabilities. These probabilities represent how likely the input belongs to each class (e.g., apple or banana).

How does Softmax work?

1. The Softmax function takes the weighted sum from each output neuron.
2. It applies a formula that makes all the outputs positive and scales them to add up to 1, like probabilities.

The formula for Softmax is:

$$P_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Where:

- z_i is weighted sum for the i-th output neuron.
- e^{z_i} - exponent of the weighted sum (ensures positivity).
- $\sum_j e^{z_j}$ is sum of all exponentials (ensures the outputs sum to 1).

Let's say the output layer has these raw scores:

- Apple neuron: $z_1=2.45$
- Banana neuron: $z_2=1.80$

- Compute the exponential values: $e^{z1} = e^{2.45} \approx 11.57$, $e^{z2} = e^{1.80} \approx 6.05$
- Compute the total sum: Sum = $e^{z1} + e^{z2}$ = 11.57 + 6.05 = 17.62
- Compute probabilities: P(Apple) = e^{z1} / Sum = 11.57 / 17.62≈0.66 (66%)
- P(Banana) = e^{z2} / Sum = 6.05 / 17.62≈0.34 (34%)

The Softmax function predicts **Apple** because it has the highest probability (66%).

Softmax ensures that all outputs are interpreted as probabilities (between 0 and 1), and the sum of probabilities is always 1, making it easy to compare and choose the most likely class.

5.3.4

What does the Softmax function do in a neural network?

- Converts raw outputs into probabilities.
- Adjusts the weights of hidden neurons.
- Creates new connections between layers.
- Replaces the activation function in the input layer.

5.3.5

Project: Neural network manual training

Let's train our banana/apple neural network step by step using a small dataset and simple math. The training process involves making predictions, calculating errors, and adjusting weights to improve the network.

We'll use the following features as inputs:

- **Shape**: 1 if round, 0 if not.
- **Color**: 1 if yellow, 0 if not.
- **Size**: 1 if small, 0 if not.

We have **one hidden neuron** (because of simple understanding) and one output neuron. The hidden neuron uses ReLU, and the output neuron uses Softmax to classify the fruit.

Here's our small training dataset:

```
Shape  Color  Size   Label
  1        0        1      Apple (1)
  0        1        1      Banana (0)
```

We start with random weights:

- Hidden neuron weights: $W_1 = 0.5$, $W_2 = 0.3$, $W_3 = 0.2$
- Output neuron weight: $W_{hidden\_output} = 0.6$

Let's process the first example (Shape = 1, Color = 0, Size = 1, Label = Apple

1. Calculate Hidden neuron output: Weighted_Sum(Hidden) = (Shape * W1) + (Color * W2) + (Size - W3) = (1 * 0.5) + (0 * 0.3) + (1 * 0.2) = 0.7
2. Apply ReLU: Hidden Output = ReLU(0.7) = 0.7 (since 0.7>0)
3. Calculate output neuron prediction: Weighted_Sum(Output) = Hidden_Output * W_{hidden_output} = 0.7 * 0.6 = 0.42
4. Apply Softmax (for simplicity, assume this is the only class, so no exponentiation needed): Predicted Output=0.42 (interpreted as probability for Apple)

Calculate error: The true label is Apple = 1. The error is calculated using **Mean squared error (MSE)**:

- Error = 1 / 2 * (True_Label − Predicted_Output)^2 = 1 / 2 * (1 − 0.42)^2 = 0.1682

Backward pass - adjusting weights

Weights are adjusted using **gradient descent**, where we calculate how the weights contributed to the error and update them. The update rule is:

$$W = W - \eta \cdot \frac{\partial \text{Error}}{\partial W}$$

Where:

- η - learning rate (set to 0.1 for simplicity).
- ∂Error / ∂W - gradient of the error with respect to the weight.

Update Hidden-to-Output weight

- For W_{hidden_output}: ∂Error / $\partial W_{hidden\_output}$ = (Predicted_Output − True_Label) * Hidden_Output = (0.42 − 1) * 0.7 = −0.58 * 0.7 = −0.406
- Update weight: W_{hidden_output} = 0.6 − 0.1 * (−0.406) = 0.6406

The process is repeated for all examples in the dataset. Over time, the weights adjust to reduce the error and improve predictions.

📝 5.3.6

What is the main goal of the backward pass during training?

- To adjust weights and reduce error.
- To calculate the final prediction.
- To decide the size of the network.
- To process input features into weighted sums.

🕮 5.3.7

Random weights

When training a neural network, the weights of connections between neurons are often initialized with random values. At first, this might seem odd - how can a network learn anything if we start with random numbers?

Why we use random weights?

- It is good starting point for learning because random weights give the network a place to start learning. Think of it like guessing on your first try when solving a new puzzle. It doesn't matter if your first guess is wrong - what matters is that you can improve from there.
- If all weights started with the same value (e.g., all zeros), every neuron would process inputs in exactly the same way. This would make the network behave like it had only one neuron, defeating the purpose of having multiple neurons.
- Random weights also allow neurons to explore different ways of combining inputs. For example: one neuron might initially focus more on color, another might focus more on shape. Over time, training adjusts these weights to better solve the problem.

Starting with random weights doesn't mean the network will always give random outputs.

- Training helps the network adjust weights. The weights are fine-tuned to reduce error with each iteration. For example, if a neuron misclassifies an apple as a banana, its weights are adjusted to focus more on features that are important for apples.
- The training process (using backpropagation and gradient descent) gradually moves the weights from their random starting values to optimal values that give accurate predictions.

If we started with fixed weights (e.g., all zeros) then all neurons would process data the same way, making the network less flexible. The network might get stuck and fail to learn patterns in the data.

By using random weights, the network starts with variety and can adjust to learn more effectively.

5.3.8

Why do we start with random weights in a neural network?

- To give neurons diversity and encourage learning.
- To ensure the network always gives correct predictions.
- To keep the training process faster.
- To avoid having too many hidden neurons.

5.3.9

Project: Train a neural network to classify fruit

In this project, we will create a simple neural network to classify fruit as **banana** or **apple** based on three features: **Shape**, **Color**, and **Size**. The dataset will be defined directly in the Python code.

1. Dataset:

- The dataset X contains features: **Shape** (1 = round, 0 = not round), **Color** (1 = yellow, 0 = not yellow), and **Size** (1 = small, 0 = not small).
- Labels y classify the fruit: **1 = Apple**, **0 = Banana**.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score,
classification_report

# Define the dataset
# Features: Shape, Color, Size
# Labels: 1 = Apple, 0 = Banana
X = np.array([
    [1, 0, 1],  # Apple
    [0, 1, 1],  # Banana
    [1, 1, 1],  # Apple
    [0, 0, 0]   # Banana
```

```
])

y = np.array([1, 0, 1, 0])  # Labels: Apple = 1, Banana = 0
```

2. Train/Test Split:

- The dataset is split into training (75%) and testing (25%) subsets using train_test_split.

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
```

3. Neural network definition:

- A Multi-Layer Perceptron (MLP) with one hidden layer (3 neurons) is defined.
- Activation function: ReLU (Rectified Linear Unit).
- Solver: Stochastic gradient gescent (SGD). A **solver** in a neural network is the algorithm used to optimize the model during training. It adjusts the weights of the neurons to minimize the error between the predicted output and the true output.

```
# Define the neural network
# One hidden layer with 3 neurons, ReLU activation, and
stochastic gradient descent solver
clf = MLPClassifier(
    hidden_layer_sizes=(3,),  # One hidden layer with 3
neurons
    activation='relu',        # ReLU activation function
    solver='sgd',             # Stochastic Gradient Descent
solver
    max_iter=1000,            # Maximum iterations
    random_state=42,
    verbose=True              # Enable output to show training
progress
)

# Train the neural network and capture loss curve
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)
```

4. Evaluation:

- The accuracy_score and classification_report functions evaluate how well the network predicts labels on the test set
- Accuracy is a percentage showing how many predictions were correct.
- Classification report contains precision, recall, and F1-score for each class (Banana and Apple)..

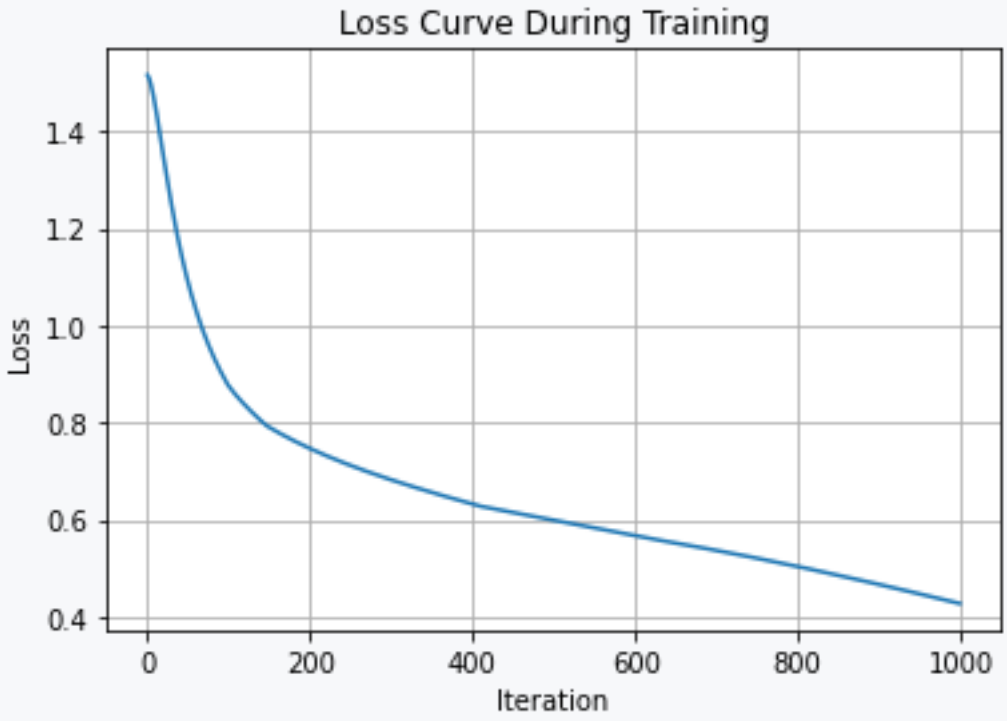
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:")
print(classification_report(y_test, y_pred,
target_names=["Banana", "Apple"]))
```

5. Visualization

- Loss curve - the loss_curve_ attribute of the MLPClassifier stores the error (loss) at **each iteration** during training. Plotting this curve shows how the loss decreases as the model learns from the data.
- Axes - X-Axis represent iterations (number of times the network processes the training data), Y-Axis represent loss (how far the predictions are from the true labels).
- A steadily decreasing curve indicates that the model is learning. If the curve flattens, the model has converged (reached its best performance for the given settings).

```
import matplotlib.pyplot as plt

# Visualize the loss curve
plt.plot(clf.loss_curve_)
plt.title("Loss Curve During Training")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.grid()
plt.show()
```

Program output:



Neural Networks

Chapter 6

6.1 Perceptron

🕮 6.1.1

A **perceptron** is one of the simplest types of artificial neural networks. It's like a single neuron that can solve simple classification problems. The perceptron takes inputs, processes them, and decides between two classes, such as **apple** or **banana**.

How does it work?

- The perceptron receives inputs, like features of a fruit (e.g., Shape, Color, Size).
- Each input is multiplied by a weight, which tells the perceptron how important that input is.
- The perceptron adds up the weighted inputs to calculate a total score (weighted sum).
- If the score is above a certain threshold, the perceptron outputs **1** (e.g., apple). Otherwise, it outputs **0** (e.g., banana).

A perceptron can only solve problems where the data can be separated with a straight line, called **linearly separable problems**.

📝 6.1.2

What is the main job of a perceptron?

- To classify data into one of two categories.
- To process multiple layers of data.
- To predict probabilities for multiple classes.
- To store information about the dataset.

🕮 6.1.3

Let's see how a perceptron works step by step using the banana/apple example.

1. Inputs - the perceptron takes three features as inputs: Shape (1 if round, 0 if not), Color (1 if yellow, 0 if not), Size (1 if small, 0 if not).

2. Weights - each input is multiplied by its corresponding weight:

- Weighted_Input = (Shape * W1) + (Color * W2) + (Size * W3)

3. Threshold - the perceptron compares the weighted sum to a threshold:

- If the score is greater than or equal to the threshold, it outputs **1** (e.g., apple).
- If the score is below the threshold, it outputs **0** (e.g., banana).

Example inputs: Shape = 1, Color = 0, Size = 1

- Weights: W1 = 0.7, W2 = 0.5, W3 = 0.6
- Threshold: 1.0
- Weighted_Sum = (1 * 0.7) + (0 * 0.5) + (1 * 0.6) = 1.3
- Since 1.3>1.0, the perceptron outputs **1 (Apple)**.

📝 6.1.4

What happens if the perceptron's weighted sum is below the threshold?

- It outputs 0 (e.g., banana)
- It increases the threshold automatically.
- It doubles the weights of the inputs.
- It stops processing the inputs.

📝 6.1.5

What happens if the perceptron's weighted sum is below the threshold?

- It outputs 0 (e.g., banana)
- It increases the threshold automatically.
- It doubles the weights of the inputs.
- It stops processing the inputs.

🕮 6.1.6

Training a perceptron involves teaching it the correct weights for its inputs so it can make accurate predictions.

1. Provide data: input features (e.g., Shape, Color, Size), correct labels (e.g., Apple = 1, Banana = 0).

2. Calculate Error: the perceptron compares its output to the correct label.

- Error = Correct_Label − Predicted_Output

3. Update weights

- If the prediction is wrong, the weights are adjusted to reduce the error: $W_i = W_i + \eta * (Error) * (Input_i)$
- η is learning rate (how big the weight changes should be).

For the example:

- Input: Shape = 1, Color = 0, Size = 1
- Initial weights: W1 = 0.7, W2 = 0.5, W3 = 0.6
- Learning rate: η = 0.1

- True label: 0 (Banana)
- Predicted output: 1 (Apple)
- Error = 0 − 1 = −1

Weights are updated as follows:

- W1 = 0.7 + (0.1 * −1 * 1) = 0.7 − 0.1 = 0.6
- W2 = 0.5 + (0.1 * −1 * 0) = 0.5
- W3 = 0.6 + (0.1 * −1 * 1) = 0.6 − 0.1 = 0.5

📝 6.1.7

What happens during the training of a perceptron?

- Weights are adjusted based on the error.
- The perceptron compares its output to the correct label.
- The threshold changes automatically.
- The input features are modified.

🕮 6.1.8

While perceptrons are simple and useful for basic problems, they have some limitations. Main limitation is linearly separable data - a perceptron can only classify data that is linearly separable. This means there must be a straight line (or plane, in higher dimensions) that can divide the data into two classes.

For example:

- A perceptron can easily separate bananas and apples if they differ clearly by one feature, like color.
- However, if bananas and apples overlap in their features (e.g., both can be small and round), a single perceptron won't work.

For more complex problems, we use a **multi-layer perceptron (MLP)**, which has multiple layers of neurons. These extra layers allow the network to handle non-linear problems.

📝 6.1.9

What is the main limitation of a single perceptron?

- It can only classify linearly separable data.
- It requires multiple output neurons.
- It cannot adjust weights during training.
- It cannot handle small datasets.

6.1.10

Project: Fruit classification by perceptron

Implement a perceptron to classify fruits as banana or apple based on the features Shape, Color, and Size.

1. Dataset:

- The X matrix contains features: Shape, Color, Size.
- The y array contains labels: 1 for Apple, 0 for Banana.

```
import numpy as np

# Define the dataset
# Features: Shape, Color, Size
# Labels: Apple = 1, Banana = 0
X = np.array([
    [1, 0, 1],  # Apple
    [0, 1, 1],  # Banana
    [1, 1, 1],  # Apple
    [0, 0, 0]   # Banana
])
y = np.array([1, 0, 1, 0])
```

2. Perceptron parameters settings:

- weights - each feature has an associated weight initialized to 0.
- bias - an additional parameter initialized to 0 to shift the decision boundary.
- learning_rate controls how much weights and bias are updated.

```
# Perceptron Parameters
learning_rate = 0.1
epochs = 10

# Initialize weights and bias
weights = np.zeros(X.shape[1])  # [0, 0, 0]
bias = 0
```

4. Training - for each example, the perceptron:

- Calculates the weighted sum - combines inputs using the weights and bias.
- Makes a prediction by using the activation function (step function) to decide the output.

- Adjusts weights and bias - if the prediction is wrong, it updates them using the error and learning rate.

```
# Activation function (Step function)
def activation_function(weighted_sum):
    return 1 if weighted_sum >= 0 else 0

# Training the Perceptron
for epoch in range(epochs):
    print(f"Epoch {epoch + 1}/{epochs}")
    for i in range(len(X)):
        # Calculate weighted sum
        weighted_sum = np.dot(weights, X[i]) + bias

        # Predict output using activation function
        prediction = activation_function(weighted_sum)

        # Calculate error
        error = y[i] - prediction

        # Update weights and bias if there's an error
        if error != 0:
            weights += learning_rate * error * X[i]
            bias += learning_rate * error

        # Print weights and bias after each update
        print(f"Data: {X[i]}, True Label: {y[i]}, Prediction:
{prediction}, Weights: {weights}, Bias: {bias}")
    print("-" * 50)
```

5. Testing the perceptron on new data to classify it as Apple or Banana.

```
# Testing the Perceptron
print("\nTesting the Perceptron:")
test_data = np.array([[1, 0, 0], [0, 1, 0], [1, 1, 0]])
test_labels = ["Apple", "Banana", "Apple"]

for i, data in enumerate(test_data):
    weighted_sum = np.dot(weights, data) + bias
    prediction = activation_function(weighted_sum)
    predicted_label = "Apple" if prediction == 1 else "Banana"
    print(f"Test Data: {data}, Predicted Label:
{predicted_label}, True Label: {test_labels[i]}")
```

Program output:

```
Testing the Perceptron:
Test Data: [1 0 0], Predicted Label: Apple, True Label: Apple
Test Data: [0 1 0], Predicted Label: Banana, True Label:
Banana
Test Data: [1 1 0], Predicted Label: Apple, True Label: Apple
```

📖 6.1.11

Multilayer perceptron

A **Multilayer Perceptron (MLP)** is a type of neural network that improves on the basic perceptron by adding more layers and neurons. This allows it to solve more complex problems, such as recognizing handwritten digits or classifying fruits with overlapping features.

An MLP consists of at least three layers:

- Input layer is where the network receives data (e.g., Shape, Color, Size for fruits). Each neuron in the input layer corresponds to a feature of the data.
- Hidden layer(s) are the "thinking" layers where the network processes and combines information. Hidden neurons are fully connected to neurons in the previous and next layers. They use nonlinear activation functions (like ReLU or sigmoid) to handle complex relationships.
- Output layer provides the final prediction. The number of output neurons depends on the task: For binary classification (e.g., apple or banana), there is one output neuron. For multi-class classification, there is one output neuron per class.

In a basic perceptron, the output depends only on a linear combination of inputs. This limits it to solving problems that can be separated with a straight line (linearly separable problems).

An MLP uses **nonlinear activation functions**, such as ReLU or sigmoid, to allow the network to learn complex patterns, like distinguishing between overlapping classes (e.g., a round, small fruit could be either an apple or an orange).

📝 6.1.12

What is the main improvement of a multilayer perceptron over a basic perceptron?

- It uses hidden layers to learn complex patterns.
- It always has more neurons in the input layer.
- It eliminates the need for training.
- It processes only linearly separable data.

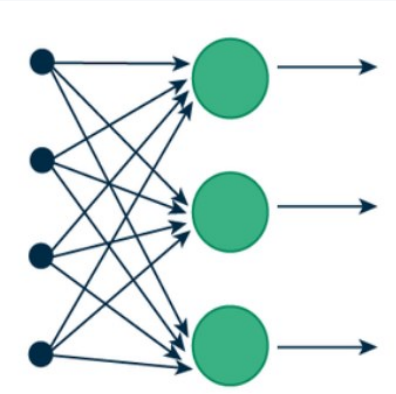
6.2 Topology of a neural network

📖 6.2.1

Feedforward neural networks

One of the simplest types of neural networks is the feedforward neural network (FNN). In this network, information flows in one direction: from the input layer, through the hidden layers, to the output layer. This straightforward design allows the network to learn how to associate input data with specific outputs.

For example, a feedforward neural network can classify fruits based on their features. It might use the color, shape, and size of the fruit to predict whether it's an apple or a banana. However, FNNs are best suited for tasks where the data doesn't involve sequences or time, as they process all inputs at once without considering the order.



📝 6.2.2

What is a key characteristic of a feedforward neural network?

- It processes sequential data.
- Information flows in one direction, from input to output.
- It uses memory to remember past inputs.
- It loops back information from the output.

📖 6.2.3

Convolutional neural networks

When it comes to analyzing images, convolutional neural networks (CNNs) are widely used. CNNs are specialized neural networks designed to detect patterns like edges, shapes, and textures in pictures. Instead of looking at the entire image at once, a CNN focuses on small sections, scanning the image piece by piece.

For example, a CNN can identify objects in a photograph, such as cars, trees, or people. It's also used in facial recognition technology, like unlocking your phone with your face. Beyond images, CNNs are applied in fields like medical imaging, where they help doctors detect tumors in scans.

6.2.4

What is the primary task of a convolutional neural network?

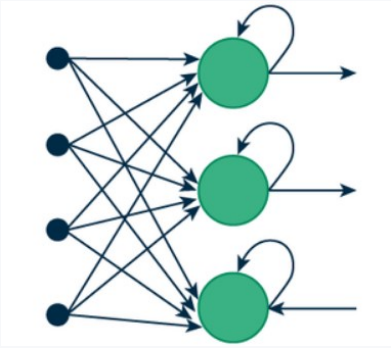
- Analyzing patterns in images.
- Translating text into different languages.
- Processing sequential data like audio.
- Predicting time-series trends.

6.2.5

Recurrent neural network

Recurrent neural networks (RNNs) are built for handling sequential data. Unlike feedforward networks, RNNs have loops that allow them to remember information from earlier inputs. This memory makes them ideal for tasks where the order of data matters.

For instance, RNNs can predict the next word in a sentence based on the words that came before it. If you type "I love," an RNN might predict that the next word could be "chocolate" or "coding." They are also used in speech recognition and stock price prediction, where patterns change over time.



6.2.6

What makes recurrent neural networks different from feedforward neural networks?

- They scan images for patterns.
- They process data without considering sequences.
- They can remember information from earlier inputs.
- They use pooling layers to reduce input size.

📖 6.2.7

Transformer networks

Transformer networks are a newer type of neural network designed to handle sequential data more efficiently than RNNs. Unlike RNNs, transformers process all parts of the input at once, which allows them to focus on the most important pieces of information. This is done using a technique called self-attention.

Transformers are widely used in language tasks, like translating text from one language to another. For example, a transformer network could translate “I love programming” from English to French as “J’aime programmer.” These networks are also behind many virtual assistants and chatbots, helping them understand and generate human-like responses.

📝 6.2.8

Which tasks are commonly solved by transformer networks?

- Translating text.
- Generating chatbot responses.
- Detecting patterns in images.
- Predicting numerical trends in data.

📖 6.2.9

Generative Adversarial Networks

Generative adversarial networks (GANs) consist of two parts: a generator and a discriminator. The generator creates new data, like realistic images, while the discriminator evaluates the data to see if it looks real. These two networks compete with each other, which helps them improve over time.

GANs are used to create lifelike images, enhance low-quality pictures, and even generate realistic simulations for video games or movies. For example, a GAN can create a fake photo of a person who doesn’t exist, but the image looks completely real.

📝 6.2.10

What is the main role of a generator in a GAN?

- To create new data, like realistic images.
- To classify the input data.
- To process sequential data like text.
- To identify patterns in pictures.

6.3 Deep learning

🕮 6.3.1

A neural network is a system of artificial neurons designed to mimic how the human brain processes information. It consists of layers of neurons:

- Input layer receives the data (e.g., images, text, numbers).
- Hidden layer(s) processes the data to find patterns.
- Output layer provides the final result (e.g., a prediction).

Neural networks are powerful tools for solving problems like classification, regression, and pattern recognition.

Deep learning is a **subset of machine learning** that uses **large, complex neural networks** with many hidden layers to solve even more complex problems.

The term "deep" refers to the fact that the network has **many layers**:

- Basic neural networks might have 1-2 hidden layers.
- Deep learning models have dozens or even hundreds of layers, enabling them to process massive amounts of data and learn intricate patterns.

For example:

- A basic neural network might classify fruits based on simple features like color and size.
- A deep learning network might analyze images pixel by pixel to identify fruits, even if they’re partially hidden or in different lighting.

Neural networks are the foundation of deep learning. All deep learning models are built from neural networks, but not all neural networks are deep learning models. Deep learning networks often require more data and computational power than basic neural networks.

Deep learning allows us to:

- Recognize complex patterns - identify faces, objects, or even emotions in pictures.
- Understand languages - translate text, generate responses in chatbots, or summarize documents.
- Drive innovations - power technologies like self-driving cars and medical imaging.

📝 6.3.2

What is the main difference between neural networks and deep learning?

- Deep learning uses neural networks with many layers.
- Neural networks are only used for image recognition.
- Deep learning does not require training.
- Neural networks cannot handle complex problems.

📖 6.3.3

Self-driving cars are one of the most exciting applications of deep learning. These vehicles rely on **neural networks** to interpret their surroundings and make real-time decisions to navigate roads safely.

Deep learning models, especially **convolutional neural networks (CNNs)**, are used to process data from cameras mounted on the car. These cameras capture images of the road, traffic signs, pedestrians, and other vehicles. A CNN detects objects, such as stop signs or lane markings, by recognizing patterns in the images.

Another type of network, **recurrent neural networks (RNNs)**, processes sequential data, like predicting how a nearby vehicle might move based on its current speed and direction. Combining these networks allows the car to understand both its current environment and what might happen next.

Self-driving cars reduce human error, which is a leading cause of accidents. With deep learning, these vehicles can make decisions faster than humans, ensuring safer and more efficient travel.

📝 6.3.4

Which types of neural networks are commonly used in self-driving cars?

- CNNs
- RNNs
- GANs
- Transformers

📖 6.3.5

Deep learning has transformed healthcare by enabling more accurate diagnosis and treatment recommendations. Neural networks can analyze complex medical data, such as images, patient histories, and genetic information.

Convolutional neural networks (CNNs) are used to examine medical images, like X-rays or MRIs, to detect diseases. For example, a CNN can identify early signs of cancer in a scan by spotting subtle patterns that might be missed by the human eye. Similarly, **deep learning models** analyze patient records to predict the risk of developing chronic conditions, like diabetes or heart disease.

Deep learning is also applied in drug discovery, where models test how different compounds might interact with diseases. By simulating millions of scenarios, these networks speed up the process of finding new treatments.

Deep learning reduces diagnosis errors, enables earlier detection of diseases, and accelerates the development of life-saving drugs, ultimately improving patient outcomes and saving lives.

📝 6.3.6

What is one way deep learning is used in healthcare?

- To analyze medical images and detect diseases.
- To generate patient records automatically.
- To predict weather conditions for hospitals.
- To classify DNA into numerical groups.

📖 6.3.7

Deep learning has revolutionized language translation, making tools like Google Translate highly effective. These systems can now understand the context and meaning of sentences, producing more accurate translations.

Transformer networks, like the ones used in models such as GPT or BERT, are at the heart of language translation. They use a technique called **self-attention**, which allows them to focus on the most important words in a sentence. For example, in the sentence "She took the book to the library", the word "library" helps clarify where "book" is being taken.

The model processes the entire sentence at once, rather than word by word, ensuring the translation preserves the correct meaning and grammar. This is especially important for languages with complex structures, like German or Japanese.

Deep learning enables seamless communication across languages, breaking down barriers in travel, business, and education. It has also made real-time translation possible in apps and devices, helping people communicate instantly without learning a new language.

📝 6.3.8

What is a key advantage of using transformer networks for language translation?

- They focus on the most important words in a sentence.
- They analyze medical records.
- They scan images for objects.
- They predict stock prices over time.

Search and Problem Solving

Chapter 7

7.1 Introduction

📖 7.1.1

Search is a process used in Artificial intelligence to find solutions to problems. Imagine you are playing a maze game on your phone. The goal is to find the best path from the start to the finish. AI does something similar but much faster and more efficiently. It explores all possible paths (or solutions) and selects the best one based on specific rules.

In AI, a search problem has three main components: the initial state (where the problem starts), a goal state (the desired outcome), and a set of actions (the steps that can be taken to move from one state to another). For example, in a puzzle game, the initial state is the scrambled pieces, the goal state is the completed puzzle, and the actions are moving or rotating pieces.

There are two types of search: **uninformed search** and **informed search**. Uninformed search does not have any extra information about the problem other than the rules, while informed search uses additional data, like estimates of how close a state is to the goal. Informed searches are faster and more efficient because they can make smarter decisions.

AI search is not just for games! It is used in navigation apps like Google Maps, planning delivery routes for packages, or even solving complex scientific problems.

📝 7.1.2

What is a key difference between uninformed and informed search in AI?

- Informed search uses additional information about the problem.
- Uninformed search relies on problem rules alone.
- Uninformed search is always faster than informed search.
- Informed search never finds the best solution.

📖 7.1.3

AI problem solving is about finding the best or most efficient solution to a given challenge. Imagine you're planning a road trip with multiple stops. You want to visit all the places without driving extra miles. This type of problem is common in AI and is called an **optimization problem**.

To solve problems, AI often represents them as a graph or a tree. In a graph, locations (or states) are shown as points, and possible actions are shown as lines connecting these points. The AI algorithm explores this graph to find the best route or solution. For example, in the road trip problem, the AI would calculate the shortest path connecting all stops.

Some common problem-solving strategies include:

- Breadth-first search (BFS) explores all possible options level by level. It is thorough but can be slow.
- Depth-first search (DFS) explores one path deeply before moving to others. It uses less memory but might miss better solutions.
- A Search* combines information about the current state and estimates how close each state is to the goal, making it both effective and efficient.

So, Breadth-first search and Depth-first search are examples of uninformed search methods because they do not use any additional information about the goal state. In contrast, A Search* is an informed search because it uses a heuristic to estimate the cost of reaching the goal from a given state. AI problem solving helps with scheduling flights, diagnosing diseases, and even organizing sports tournaments.

7.1.4

Which AI search strategy is most efficient for solving problems with known goal distances?

- A* Search
- Breadth-First Search
- Depth-First Search
- Random Search

7.1.5

AI search and problem solving play a big role in the technology you use daily. For example, when you search for the quickest way home using a GPS app, the app uses search algorithms to find the shortest or fastest route. It considers all possible roads and traffic conditions to suggest the best option.

In video games, AI opponents use search to decide their moves. For instance, in chess, the AI looks ahead at possible moves to select the one that gives it the best chance to win. This process involves searching through millions of possible game states.

AI also helps companies make decisions. Airlines use AI to schedule flights, ensuring that planes and crews are in the right places. In warehouses, robots use search to find the best path to pick up items and deliver them to the packing area efficiently.

Understanding how AI works in these situations can inspire you to think about how problems in your own life might be solved using similar methods.

7.1.6

Which of the following is an example of AI search in everyday life?

- A GPS app finding the shortest route
- A chess AI deciding its next move
- A robot cleaning a room randomly
- A student solving math homework manually

7.2 Search problem components

📖 7.2.1

A search problem is a way of organizing a challenge so that AI can systematically find a solution. It is made up of three main parts: the **initial state**, the **goal state**, and the **actions**. These components work together to define the problem clearly.

1. **Initial state** is the starting point of the problem. For example, if you are solving a maze, the initial state is the position where you begin. In a delivery problem, the initial state could be the location of the delivery truck with its list of packages.
2. **Goal state** is the desired result or the solution to the problem. For the maze, the goal state is the position where you exit the maze. In the delivery problem, the goal state might be all packages delivered to their destinations.
3. **Actions** are the possible steps or moves that can be taken to get closer to the goal. For instance, in the maze, actions could be moving left, right, up, or down. In the delivery problem, actions could include driving to a specific street or dropping off a package.

By breaking a problem into these parts, AI can analyze and figure out the best way to solve it. This structure is used in various applications, from games to navigation systems.

📝 7.2.2

Which of the following is part of a search problem?

- Initial state
- Goal state
- Actions
- Graph nodes

📖 7.2.3

The **state space** is like a map of all possible situations you can reach from the starting point by taking different actions. Each state represents a possible condition or arrangement in the problem. For example:

- In a maze, each position you can stand in is a state.
- In a chess game, each possible arrangement of pieces on the board is a state.

AI uses the state space to explore all possibilities and find the best path to the goal. The process involves moving from one state to another by applying actions. If the AI is solving a puzzle, the state space includes all the possible arrangements of the pieces, starting from the scrambled puzzle to the solved one.

One challenge in search problems is that the state space can get very large. For example, in a maze with 100 positions, the state space might have hundreds of possibilities. In a chess game, there are millions of possible states. AI uses smart strategies to explore only the most relevant parts of the state space.

📝 7.2.4

What does the state space represent?

- All possible states in a problem
- A list of actions
- Only the starting and goal states
- Every incorrect solution

🕮 7.2.5

Graphs

AI often represents search problems as **graphs** because graphs make it easy to visualize and analyze the problem. A graph is made up of:

1. **Nodes (or vertices)** represent states. For example, in a maze, each position you can stand in is a node.
2. **Edges (or connections)** represent actions or transitions. For instance, moving from one position in the maze to another creates an edge.

Imagine a road trip. In a graph, each city is a node, and each road connecting the cities is an edge. If you want to find the shortest route from one city to another, AI will explore the graph to figure it out.

Graphs are helpful because they allow AI to calculate distances, costs, or time between nodes. In a navigation app, the graph might show the distance between two cities as a number on each edge. AI can then use this information to find the quickest or shortest path.

📝 7.2.6

In a graph representation, what do nodes and edges represent?

- Nodes are states; edges are actions
- Nodes are goals; edges are paths
- Nodes are actions; edges are states
- Nodes are guesses; edges are goals

🕮 7.2.7

Trees

Another way to represent problems is by using **trees**. A tree starts with a single point called the **root node**, which represents the initial state. From there, it branches out into other nodes, which represent new states created by applying actions.

For example, imagine a simple puzzle with a starting position. The root node is the starting position. If you make a move, like sliding a piece, a new node (state) is created. The tree keeps growing as more actions are taken. Each branch of the tree represents a sequence of actions leading to a new state.

A tree is different from a graph because it does not have loops or cycles. Trees are helpful when there is a clear hierarchy of states. For example, in a math problem where each step builds on the previous one, a tree can be used to represent all the possible solutions.

In many AI problems, the goal is to search the tree to find the shortest or most efficient path from the root node (initial state) to a specific goal node (goal state).

📝 7.2.8

What is the root node in a tree representation?

- The initial state
- A random state
- The goal state
- A final action

7.3 Problem solving

🕮 7.3.1

The early days of artificial intelligence were filled with big dreams. Researchers believed that with just a few smart techniques, they could create programs that could solve any problem - what we might call "all-solving" programs. These programs were built around two main ideas:

1. An internal model of the world where the program created a simplified version of reality inside the computer.
2. Planning solutions by using this model, the program could plan a sequence of steps to solve a problem.

However, solving problems is not as simple as it sounds. The first and most important step is defining the problem. This involves identifying:

- Choices and their consequences, for example, if you're playing a board game, each move you make opens up new possibilities and limits others.
- The goal must be clearly defining when the problem is solved. For instance, in chess, the goal is to checkmate the opponent's king.

Once the problem is well-defined, the next step is finding a series of actions that will take you from the starting point (initial state) to the solution (goal state). AI problems can be grouped into two categories:

1. Search and planning in static environments which involve just one "agent" (like a robot or an AI system) trying to solve a problem without competition. For example, a GPS app finding the fastest route to your destination.
2. Games with two players (agents) which involve competition, like in chess or tic-tac-toe, where the AI must consider what the other player might do.

Understanding these concepts is essential to see how AI systems think and solve problems.

📝 7.3.2

Which step is essential before solving a problem in AI?

- Defining choices and their consequences
- Creating a sequence of actions
- Guessing randomly
- Ignoring the goal state

🕮 7.3.3

Project: Fox and chicken by the river

As an introduction to Search and planning methods, we would like to explain the main concepts using a simple example of the Chicken Crossing Puzzle.

The puzzle - a man has to get a fox, a chicken, and a sack of corn across a river.

- Only the farmer can operate the rowboat.
- Only two of the pieces of cargo can fit on the rowboat together with the man.
- If the fox and the chicken are left together, the fox will eat the chicken.
- If the chicken and the corn are left together, the chicken will eat the corn.

How does the man do it?

We will model the puzzle by noting that some movable things have been identified: the farmer, the rowboat, the fox, the chicken, and the sack of corn. Only the farmer can operate the rowboat; the two will always be on the same side. Thus, there are four things with two possible positions for each, which makes for sixteen combinations, which we will call states. We have added identification to the states. We assume a fixed order of things (the farmer, the fox, the chicken, and the last one is the sack of corn) and state 1 or 2 to determine whether the thing is on one side (side1) of the river or on the other (side2).

The starting state is 1111, and the goal state is 2222 instead of something like “in the starting state, the robot is on the near side, the fox is on the near side, the chicken is on the near side, and also the chicken-feed is on the near side, and in the goal state the robot is on the far side”, and so on.

State	Farmer	Fox	Chicken	Sack of corn
1111	side1	side1	side1	side1
1112	side1	side1	side1	side2
1121	side1	side1	side2	side1
1122	side1	side1	side2	side2
1211	side1	side2	side1	side1
1212	side1	side2	side1	side2
1221	side1	side2	side2	side1
1222	side1	side2	side2	side2
2111	side2	side1	side1	side1
2112	side2	side1	side1	side2
2121	side2	side1	side2	side1
2122	side2	side1	side2	side2
2211	side2	side2	side1	side1
2212	side2	side2	side1	side2
2221	side2	side2	side2	side1
2222	side2	side2	side2	side2

We deleted some of these states that are forbidden by the puzzle conditions. For example, the fourth state, 1122 (meaning that the farmer is on the side1 with the fox, but the chicken and the sack of corn are on the other side - side2), the chicken will eat the corn, which we cannot have.

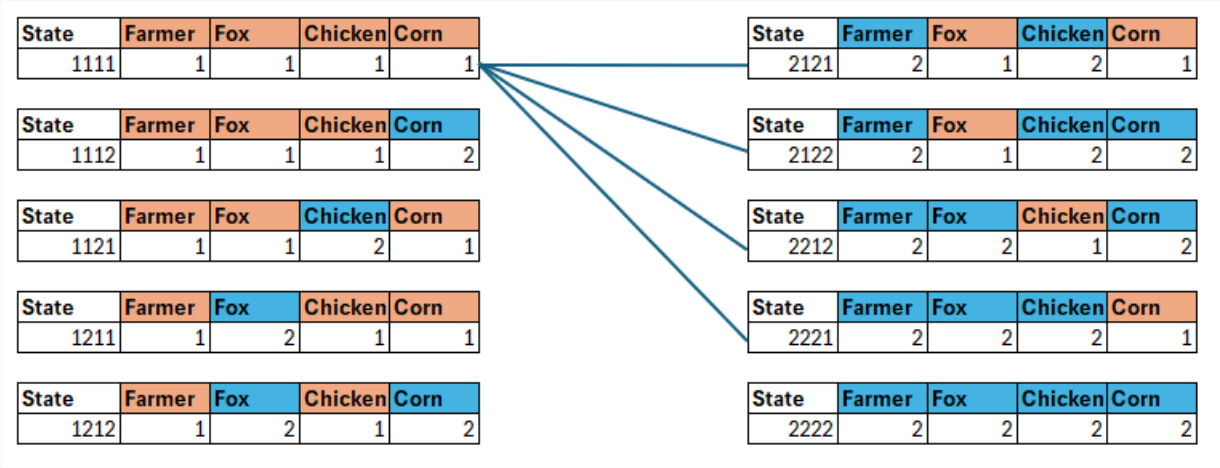
State	Farmer	Fox	Chicken	Sack of corn
1111	side1	side1	side1	side1
1112	side1	side1	side1	side2
1121	side1	side1	side2	side1
1122	side1	side1	side2	side2
1211	side1	side2	side1	side1
1212	side1	side2	side1	side2
1221	side1	side2	side2	side1
1222	side1	side2	side2	side2
2111	side2	side1	side1	side1
2112	side2	side1	side1	side2
2121	side2	side1	side2	side1
2122	side2	side1	side2	side2
2211	side2	side2	side1	side1
2212	side2	side2	side1	side2
2221	side2	side2	side2	side1
2222	side2	side2	side2	side2

We are left with the following ten states:

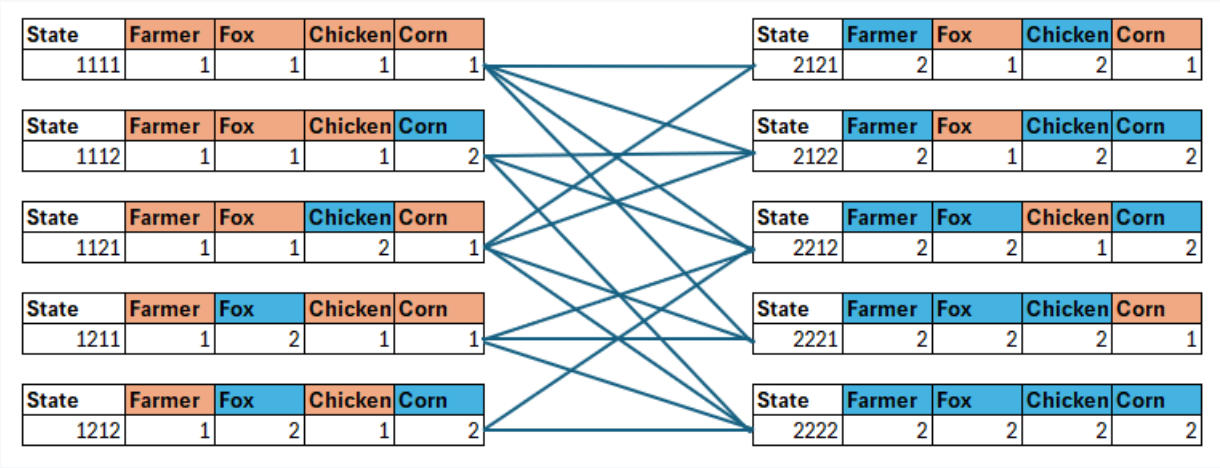
State	Farmer	Fox	Chicken	Sack of corn
1111	side1	side1	side1	side1
1112	side1	side1	side1	side2
1121	side1	side1	side2	side1
1211	side1	side2	side1	side1
1212	side1	side2	side1	side2
2121	side2	side1	side2	side1
2122	side2	side1	side2	side2
2212	side2	side2	side1	side2
2221	side2	side2	side2	side1
2222	side2	side2	side2	side2

Next we will figure out which state transitions are possible, meaning simply that as the farmer rows the boat with some of the items as cargo, what the resulting state is in each case. It's best to draw a diagram of the transitions, and since in any transition the first number alternates between 1 and 2, it is convenient to draw the states starting with 1 (so the farmer is on the side1) in one row and the states starting with 2 in another row:

We could also draw arrows that have a direction so that they point from one node to another, but in this puzzle, the transitions are symmetric: if the farmer can row from one state to another, it can equally well row the other way. Thus it is simpler to draw the transitions simply with lines that don't have a direction.

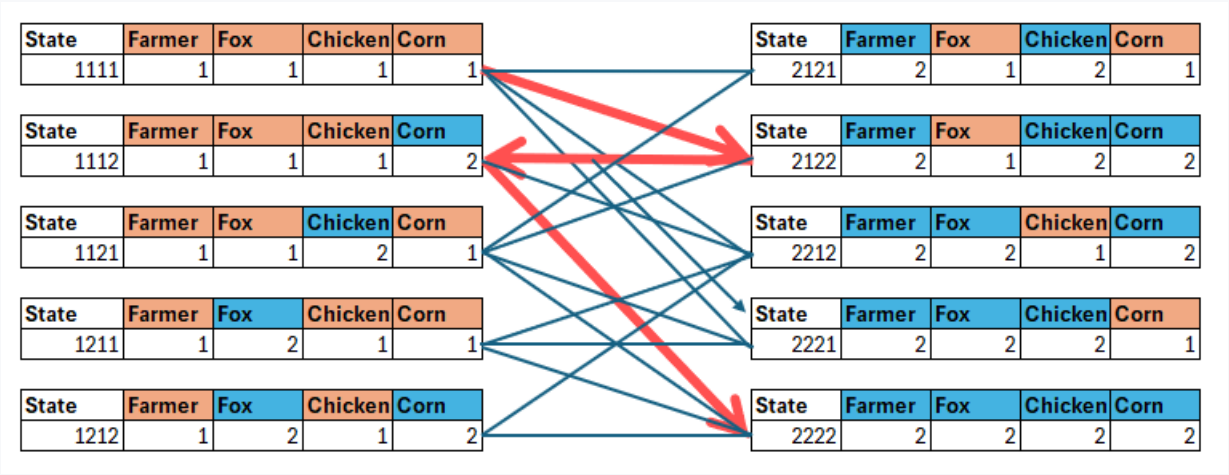


Then we fill in the rest:



Now that we have formulated the alternative states and transitions between them, the rest becomes a mechanical task: find a path from the initial state 1111 to the final state 2222.

One such path is colored in the following picture.



📝 7.3.4

Arrange the following steps in the correct order to solve the Chicken crossing puzzle:

- Take the chicken across the river again.
- Take the chicken across the river.
- Take the fox across the river.
- Return to the starting side alone.
- Return to the starting side alone.
- Take the sack of corn across the river.
- Bring the chicken back to the starting side.

📖 7.3.5

In the Chicken crossing puzzle example, we saw how important it is to **know the goal**. Without a clear understanding of the goal (getting all the items across safely), we couldn't plan the correct sequence of actions. This shows why the concept of **search** is essential in artificial intelligence (AI). Search is how AI finds solutions to problems by exploring all possible paths.

In general, search involves:

1. Exploring a **state space** that represents all the possible configurations of a problem.
2. Finding a path from the **initial state** (starting point) to the **goal state** (solution).

Search is like solving a maze: you start from the beginning, explore paths systematically, and aim to find the best route to the exit.

📝 7.3.6

Why is knowing the goal essential in problem-solving?

- It allows for planning the sequence of actions.
- It makes actions irrelevant.
- It prevents searching the state space.
- It creates an initial state automatically.

📖 7.3.7

Let's dive into the main components of searching in AI:

1. State space is the set of all possible states or configurations in the problem. Each state represents a specific situation, such as the positions of the farmer, fox, chicken, and corn.

2. Initial state is starting point of the search. In the Chicken Puzzle, it's when all items are on one side of the river.
3. Goal state is the desired solution. In this case, it's when all items are safely on the other side of the river.
4. Actions are steps that move the problem from one state to another. For instance, taking the chicken across the river is an action.

The goal of searching is to find a **sequence of actions** that connects the initial state to the goal state.

📝 7.3.8

What does the state space represent in problem-solving?

- All possible states in the problem
- Only the starting state
- A list of all actions
- Random guesses

🕮 7.3.9

A search strategy is an algorithm that decides which action to take at each step. To be effective, a search strategy must:

- Prevent loops (getting stuck repeating the same states).
- Be systematic, exploring all possibilities to avoid missing the solution.

Search often involves building a solution tree, which represents the sequence of states and actions explored. Each branch shows the result of an action, while the nodes represent different states.

Additionally, a goal can be defined in two ways:

- Explicitly as a specific set of states.
- Implicitly by describing a property that defines the goal (e.g., "all items are safely across the river").

Finally, there are two more important concepts:

- Goal test - a function that checks if the current state meets the goal.
- Path cost - a function that assigns a numeric value to the sequence of actions, showing how efficient the path is.

📝 7.3.10

What are the key requirements for a search strategy?

- Must prevent loops
- Must be systematic
- Must avoid defining the goal
- Must ignore the initial state

Uninformed Search

Chapter 8

8.1 Uninformed search

📖 8.1.1

Not all problems can be solved by algorithms, and even when they can, solving them might require enormous computing resources. This means that although we have a method for solving the problem, the solution might take too long to compute or require more memory than is available. For example: Calculating all possible outcomes in a game like chess would take an immense amount of time, even for modern computers.

Instead of focusing on solving the problem directly, one approach is to design an **algorithm that searches for a solution**. This means starting with a basic framework that explores possible answers until it finds one that works.

This concept lays the foundation for **search-based problem-solving methods** used in AI, which explore the "state space" to find paths from the starting state to the goal state.

📝 8.1.2

What is a common challenge of algorithmically solvable problems?

- They may require too much computing power.
- They can always be solved quickly.
- They never involve state spaces.
- They ignore constraints.

📖 8.1.3

Uninformed search is a type of problem-solving where the AI system has no prior knowledge about the problem’s structure or goal. It simply explores all possibilities in the **state space** without any hints or shortcuts. This approach is often used when there is no guidance on where the solution might be found.

Uninformed search methods rely on basic strategies like:

- Exploring all possible paths systematically.
- Using brute force to evaluate every possible state until the goal is found.

While uninformed search is simple and reliable, it can be slow and resource-intensive for large problems. However, it is very useful in situations where no extra information is available about the problem.

📝 8.1.4

How does uninformed search approach a problem?

- By systematically exploring all possibilities.
- By using prior knowledge of the goal.
- By ignoring the state space.
- By guessing randomly.

📖 8.1.5

Uninformed search algorithms can solve a wide range of problems, including:

- Traveling salesman problem to finding the shortest route between multiple cities, visiting each city once and returning to the start.
- Planning - creating a sequence of actions to achieve a goal, such as robotic arm movements in factories.
- Optimization problems solving problems like finding the best route for deliveries or controlling processes in manufacturing.
- Wayfinding focus on finding paths between two points on a map, as used in GPS systems.
- State space search exploring all possible states of a system to find solutions, often used in machine learning and AI.

These problems highlight how uninformed search can be applied to diverse fields, even though it does not leverage additional information about the problem.

📝 8.1.6

Which problem is commonly solved using uninformed search?

- Wayfinding
- Traveling salesman problem
- Ignoring all constraints
- Random guessing

8.2 Breadth-first search

📖 8.2.1

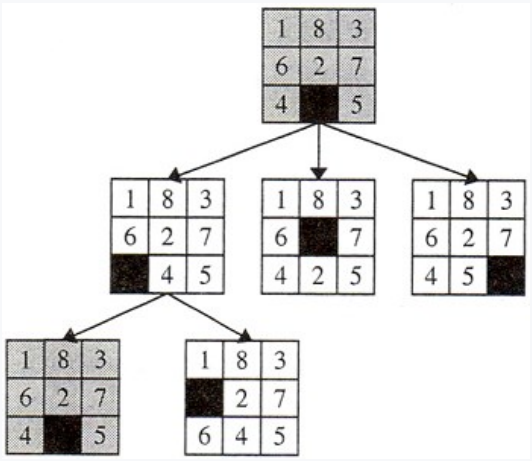
Search strategies

The 9-number puzzle is a classic problem where you move tiles on a grid to arrange them in a specific order, like solving a sliding puzzle. To solve this problem, we use search strategies that guide the process of finding the solution.

When solving the puzzle, we begin with the initial arrangement of tiles. From this state, we can generate all possible moves (called "followers of the state"). Each move creates a new state, like swapping the empty tile with a neighboring number.

Since we can't explore all states at once, we need to:

- Choose one state to work on next (this becomes the **current state**).
- Postpone other states for later exploration (store them for reference).



The order in which we select the next state to explore is determined by the **search strategy**. These strategies have no extra information about which state is better or closer to the solution. They only use the rules of the problem and explore all possibilities systematically.

8.2.2

What do uninformed search strategies rely on when solving a problem?

- Information provided in the problem definition
- Exploring all possibilities systematically
- Random guesses
- Additional hints about promising states

8.2.3

Breadth-first search

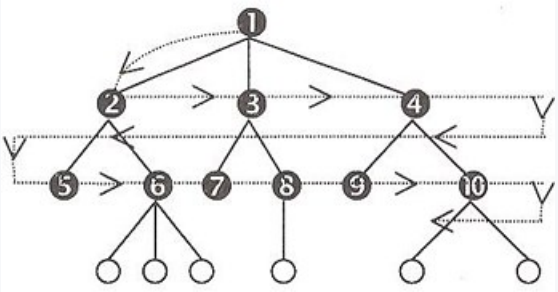
Breadth-first search (BFS) is a straightforward search strategy where nodes are explored level by level. This means the algorithm starts at the root node (the initial state), explores all its direct neighbors, then moves to the neighbors of those neighbors, and so on, until it goes through the entire connection component.

How does it work?

A breadth-first search is an algorithm or method that proceeds by systematically searching the graph through all nodes.

1. BFS systematically searches a graph by expanding all nodes at the current depth before moving to the next level.
2. It does not use heuristics or additional knowledge - just a simple rule to explore all possibilities.
3. As it explores the nodes, BFS keeps track of which nodes lead to others, creating a tree of shortest paths from the starting node to all other nodes in an unweighted graph.

In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.



A strategy is based on FIFO - the nodes that are visited first will be expanded first. If a solution exists, it will certainly be found when searching, i.e. the strategy is complete.

In summary, BFS is a systematic and reliable strategy for exploring graphs, especially useful when the shortest path in an unweighted graph is required.

8.2.4

What is the key characteristic of Breadth-first search?

- It explores nodes level by level.
- It guarantees finding a solution if one exists.
- It uses a Last In, First Out (LIFO) approach.
- It skips some nodes based on heuristics.

8.2.5

The **8-puzzle** is a classic problem consists of a 3x3 board with eight numbered tiles and one blank space. The goal is to slide tiles into the blank space, rearranging them to match a specific configuration. A stone adjacent to the blank space can slide into the space.

Initial state:

```
1 8 3
6 2 7
4 - 5
```

Goal state:

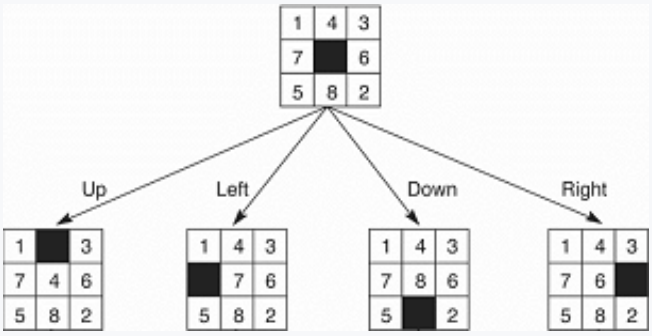
```
1 2 3
8 - 4
7 6 5
```

The 8-puzzle problem has following components:

- States - a state describes the exact positions of the eight tiles and the blank space.
- Initial state is any arrangement of the tiles and blank space can be chosen as the starting point.
- Actions are four possible actions—moving the blank space Left, Right, Up, or Down, provided the move is legal.
- Goal test - the goal state is a specific arrangement of tiles, like the one shown in the example. Other goal configurations can also be used.
- Path cost - each move counts as one step, so the total cost is the number of steps in the solution path.

Breadth-first search explores the problem level by level:

1. It starts with the **initial state** and expands all possible moves.
2. Then it evaluates all the states resulting from those moves, one level at a time.
3. BFS keeps track of visited states to avoid revisiting the same configurations.
4. The algorithm guarantees finding the shortest path (minimum moves) to the goal state because it explores all nodes at the current level before moving deeper.



8.2.6

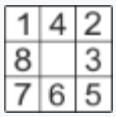
Which characteristic of BFS is demonstrated in the 8-puzzle problem?

- BFS finds the shortest path to the goal.
- BFS explores all states at one depth before moving deeper.
- BFS skips exploring some states randomly.
- BFS uses heuristics to decide the next move.

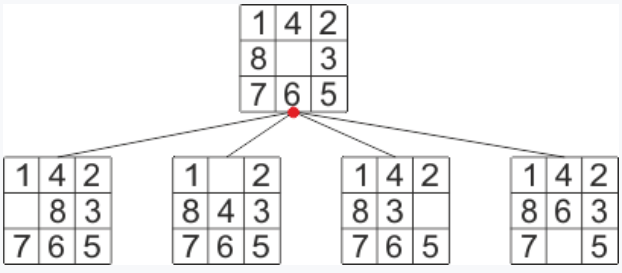
8.2.7

Project: Puzzle Breadth-first search

Let's solve puzzle with visualization using Breadth-first search. This is the initial state of our problem:



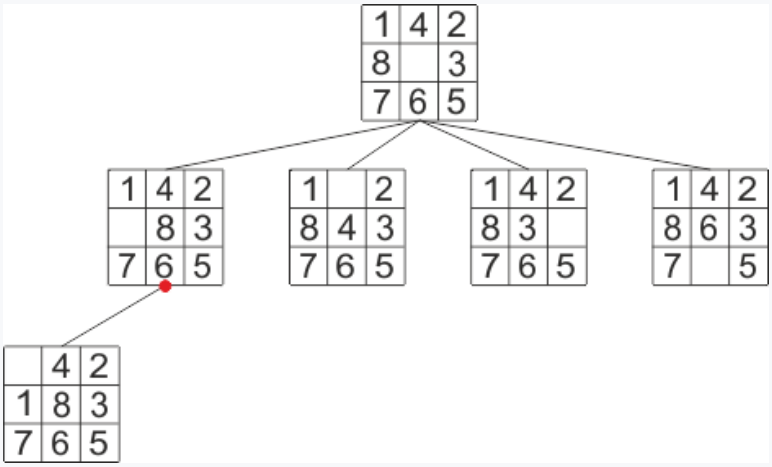
Here, we are exploring all possible states of the initial matrix using a tree structure.

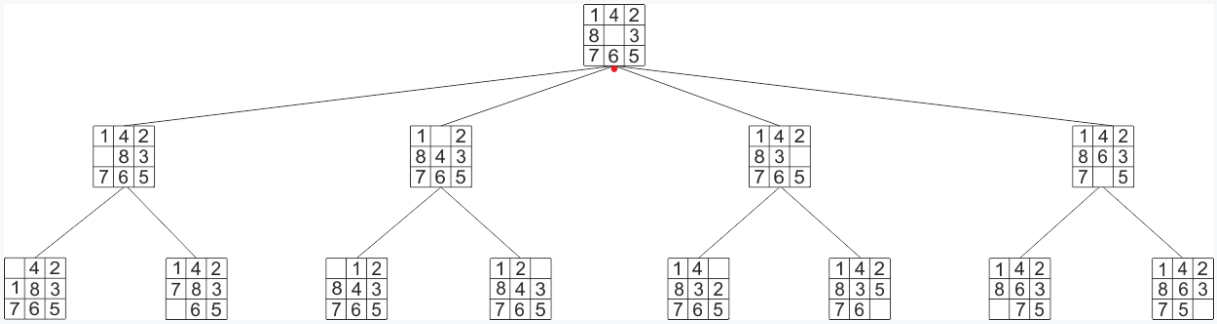


As you can see above we have created new states from our initial grid by sliding stone to the blank space.

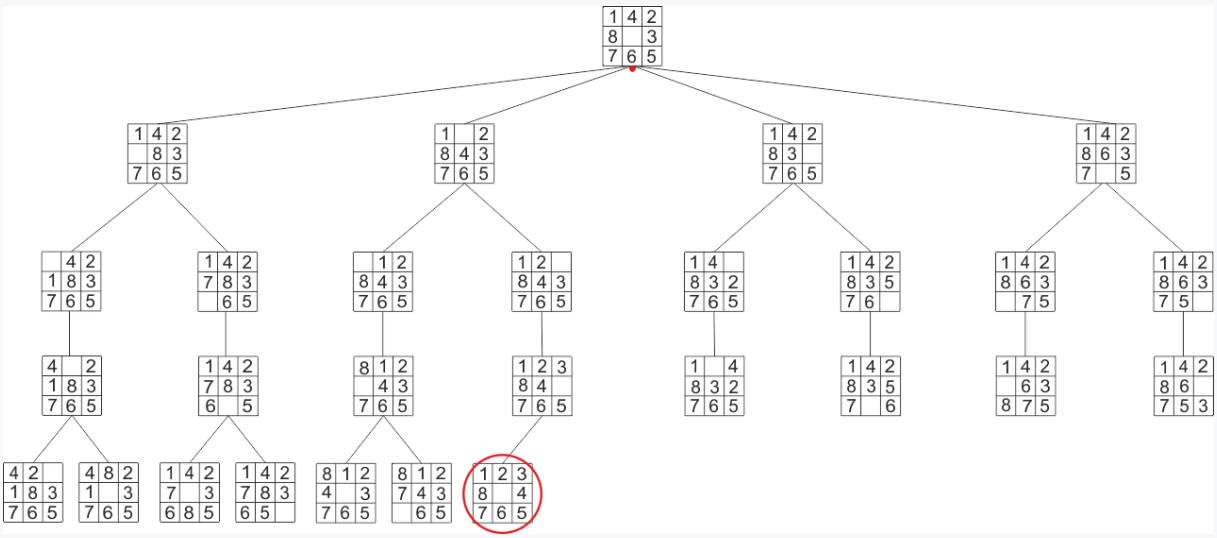
Now we will be exploring. We have already explored depth zero and found out that there are four possible moves. Now, we will find out what the possible moves of this node are. So here you can see this could be moved from top to bottom or left to bottom.

But if you see we have given only four nodes, if you move it to the left you will reach the initial State again. We are just using the unique states. so what we did is I have just moved it to bottom top and right so what I did is I have explored all the possible State except.





Finally, we can find the solution.



📖 8.2.8

Space and time complexity

When analyzing BFS, it's important to understand its space and time complexity, as they determine how efficiently the algorithm performs. BFS can quickly become resource-intensive for problems with many possible states.

Assume every state has **b** possible successors (where **b** is the branching factor). The root node generates **b** nodes at the first level, and each of those generates **b** more nodes at the second level. This pattern continues for **d** levels (where **d** is the depth of the solution). The total number of nodes generated is:

$$1+b^1+b^2+b^3+...+b^d$$

This exponential growth means that BFS can become inefficient as **b** or **d** increases, i.e. a very inefficient method.

The **time complexity** of BFS is directly related to the total number of nodes generated, which is $O(b^d)$**.** This exponential dependence makes BFS computationally expensive for large search spaces.

BFS requires storing every node in memory as it explores the graph or tree. This means the **space complexity** is also $O(b^d)$. Since BFS keeps track of all nodes in the current and previous levels, its memory requirements grow at the same rate as its time requirements. For large problems, this can quickly overwhelm system resources.

While BFS guarantees finding a solution (if one exists), its exponential space and time complexity make it unsuitable for problems with high branching factors or deep solutions.

📝 8.2.9

What is a key drawback of Breadth-first search?

- It has exponential time complexity.
- It requires storing all nodes in memory.
- It does not guarantee finding a solution.
- It skips exploring some states randomly.

8.2.10

Project: Breadth-first search (FIFO)

BFS is an algorithm designed to explore nodes in a graph or tree level by level. It systematically examines all possible states or vertices, ensuring the shortest path is found in an unweighted graph. Let's break down the process step-by-step, focusing on how to implement BFS.

Core concept is based on queue (First In, First Out - FIFO) to manage nodes during traversal. Nodes are visited in the order they are added to the queue, ensuring level-by-level exploration.

1. Initialization - create two lists or sets:

- OPEN - stores nodes currently being explored.
- CLOSED - stores nodes already fully processed.
- Insert the starting node (S_0) into OPEN. If S_0 is also the goal, terminate the algorithm.

2. Iterative search

- If the OPEN list is empty, there is no solution. Stop the search.
- Remove the first node (let's call it current_node) from OPEN. Mark it as CLOSED.
- Process current_node - generate its successors (children nodes).
- For each successor: If it is already in CLOSED, skip it; If it matches the goal state, terminate the search—solution found!
- Otherwise, add the successor to the OPEN list.

3. Repeat steps - continue expanding nodes in **OPEN** until the queue is empty or the goal state is reached.

4. Output

- If the goal state is found, output the path by tracing back the **ancestors** of each node starting from the goal.
- If the queue is exhausted, report that no solution exists.

```
def bfs(graph, start, goal):
    # Initialize structures
    open_list = [start]  # FIFO queue
    closed_list = set()  # Visited nodes
    parent = {}  # Tracks ancestors for path reconstruction

    # BFS loop
    while open_list:
```

```
        # Dequeue the first node
        current_node = open_list.pop(0)

        # Goal test
        if current_node == goal:
            return reconstruct_path(parent, start, goal)

        # Mark as visited
        closed_list.add(current_node)

        # Expand successors
        for neighbor in graph[current_node]:
            if neighbor not in closed_list and neighbor not in
open_list:
                open_list.append(neighbor)
                parent[neighbor] = current_node

    # If no solution
    return None

def reconstruct_path(parent, start, goal):
    path = []
    current = goal
    while current != start:
        path.append(current)
        current = parent[current]
    path.append(start)
    return path[::-1]
```

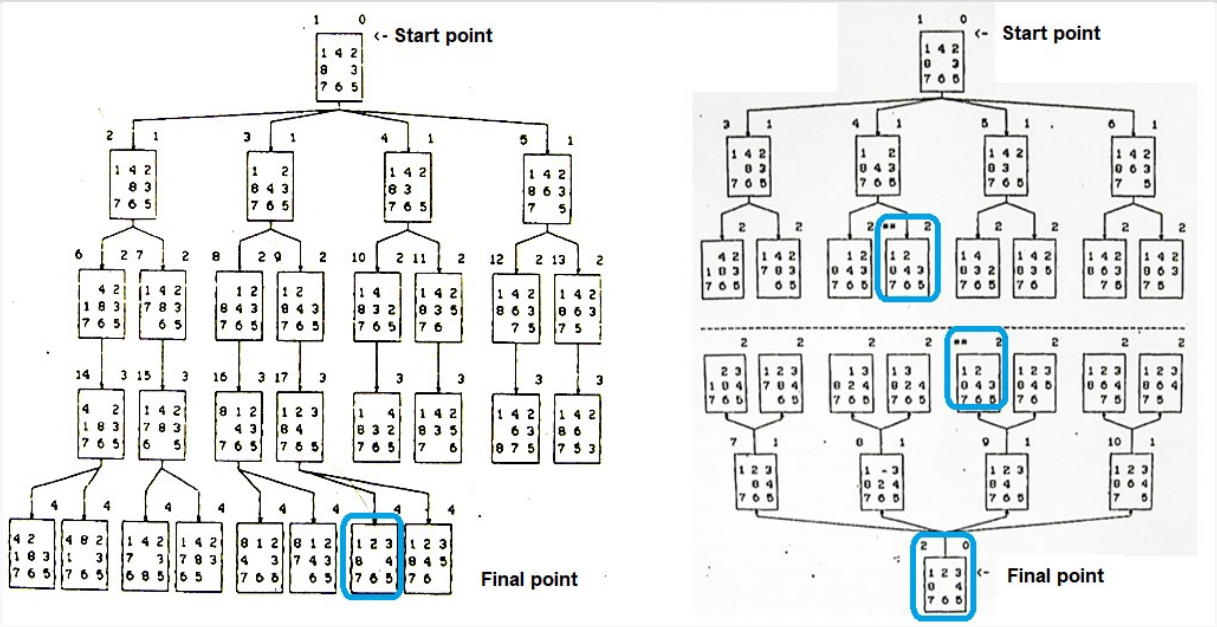


Figure: Standard and extended (combined) variant of the algorithm.

Example execution

- Graph:

```
A -> B, C
B -> D, E
C -> F
D -> Goal
E, F -> None
```

Initial State: A

Goal State: Goal

1. **OPEN = [A]** → Dequeue A → Add successors B, C.
2. **OPEN = [B, C]** → Dequeue B → Add successors D, E.
3. **OPEN = [C, D, E]** → Dequeue C → Add successor F.
4. **OPEN = [D, E, F]** → Dequeue D → Goal found!

There are different variations of BFS, such as Bidirectional Search, which starts the search simultaneously from the start and destination nodes. This method can significantly increase the search efficiency in some cases.

Breadth-first Search is a key tool in every computer scientist's arsenal. Its ability to efficiently traverse graphs and trees makes BFS an invaluable tool for solving a wide variety of problems, from network analysis to pathfinding in game environments.

8.2.11

What structure is used to manage nodes in Breadth-first search?

- FIFO Queue
- Stack
- Priority queue
- Indirect access

8.2.12

Project: Finding the way between two cities (Bread-First search)

Imagine you want to travel between two cities. The road network connecting the cities can be represented as a **graph**, where:

- Nodes (or vertices) represent the cities.
- Edges (or lines) represent the roads connecting the cities.

- BFS is a graph traversal algorithm that explores all neighbors of a node before moving to the next level of nodes. It guarantees finding the shortest path **if all edge costs are equal**.

```
From  To  Cost g(n)
 A    B    1
 A    C    2
 B    D    4
 C    D    7
 B    E    6
 D    E    2

Start city: A
Destination city: E
```

BFS uses a **queue** to keep track of nodes to visit. It explores nodes level by level.

1. Start at Node A.

- Add A's neighbors (B and C) to the queue.
- Queue: [B,C]

2. Visit node B (dequeue the front of the queue).

- Add B's neighbors (A, D, E) to the queue. Ignore A (already visited).
- Queue: [C,D,E]

3. Visit node C (dequeue the front of the queue).

- Add C's neighbors (A, D) to the queue. Ignore A and D (already in the queue).
- Queue: [D,E]

4. Visit node D (dequeue the front of the queue).

- Add D's neighbors (B, E) to the queue. Ignore B (already visited) and E (already in the queue).
- Queue: [E]

5. Visit node E (dequeue the front of the queue).

- Destination reached!

Summarisation:

Step	Node	Neighbors	Queue	Decision
1	A	B,C	[B,C]	Add B, C
2	B	D,E	[C,D,E]	Add D, E
3	C	D	[D,E]	Ignore D
4	D	E	[E]	Ignore E
5	E	-	[]	Destination Reach

BFS explores all neighbors at the current depth before moving to the next level. It ensures the shortest path in terms of the number of edges, not necessarily the total cost. In this example, BFS will find the path A→B→E (3 edges) because it checks all nodes systematically.

8.3 BFS extensions

📖 8.3.1

Uniform cost search

Uniform cost search (UCS) is a search algorithm that finds the cheapest path from a starting node to a destination node in a graph. Unlike BFS, UCS considers the costs of edges between nodes, ensuring that the solution path is the most cost-effective.

Key idea is based on exploring nodes based on their total path cost from the start node, not just their depth in the search tree. It uses a priority queue to always expand the node with the lowest path cost next.

How UCS differs from BFS?

- BFS explores nodes level by level without considering costs.
- UCS explores nodes in the order of their total path cost. If all edges have the same cost, UCS behaves like BFS.

Imagine a map where cities are connected by roads with different distances. UCS ensures you find the shortest route, even if it's not the most direct.

📝 8.3.2

What does Uniform cost search prioritize when exploring nodes?

- Total path cost
- Depth in the search tree
- Random order
- Node visited earlier

8.3.3

UCS works systematically to find the cheapest path. Here's how it operates:

1. Initialization

- adds the start node to the priority queue with a cost of 0
- creates an empty explored set to track visited nodes.

2. Node expansion

- removes the node with the lowest total cost from the queue
- if this node is the goal, stop - the cheapest path is found!
- otherwise, add this node to the explored set.

3. Adding successors

- for each child node of the current node:
- if the child is not in the explored set or queue, add it with its total path cost.
- if the child is already in the queue with a higher cost, replace it with the lower cost.

4. Repeat

- continue until the queue is empty or the goal is found.

```
function UniformCostSearch(graph, start, goal):
    frontier = PriorityQueue()
    frontier.add(start, 0)  # Start node with cost 0
    explored = set()

    while not frontier.isEmpty():
        node = frontier.pop()  # Node with the lowest cost
        if node == goal:
            return reconstructPath(node)  # Cheapest path
found

        explored.add(node)
        for child, cost in graph[node]:
            new_cost = node.path_cost + cost
            if child not in explored or frontier:
                frontier.add(child, new_cost)
            elif frontier.hasHigherCost(child, new_cost):
                frontier.replace(child, new_cost)
    return "Failure: No path found"
```

📝 8.3.4

What data structure is key to implementing UCS?

- Priority queue
- Stack
- FIFO queue
- Array

📖 8.3.5

Properties of UCS

Advantages:

- Optimality - UCS always finds the cheapest path if all edge costs are positive.
- Systematic - It explores nodes in order of increasing total path cost, ensuring no cheaper solution is missed.

Challenges:

- Time Complexity where $O(b^{1+\lfloor C*/\epsilon \rfloor})$, where:
- b -branching factor of the graph.
- $C*$ - cost of the optimal path.
- ϵ - smallest edge cost.
- Memory Complexity is the same as time complexity, as UCS must store all nodes in memory.

UCS is useful in scenarios where the lowest cost path is crucial:

- Route planning to find the shortest or cheapest route in logistics.
- Robot navigation to Guide robots along energy-efficient paths.
- Optimal routing to select the best network route with minimal delay or cost.

📝 8.3.6

In what type of problem is UCS especially useful?

- Planning the lowest-cost route
- Finding a random path
- Exploring all paths equally
- Ignoring edge costs

8.3.7

Depth-first search

Depth-first search (DFS) iis a graph traversal algorithm that explores as far as possible along a branch before backtracking to explore other branches. This behavior makes it a **backtracking algorithm** that systematically explores potential solutions.

- DFS starts at the initial node and explores its first descendant.
- It continues exploring down the branch until it reaches a "dead-end“, where no unvisited nodes remain.
- At this point, it **backtracks** to the previous node to explore other unvisited branches.
- DFS stores nodes in a **stack** (Last In, First Out - LIFO), ensuring that the most recently visited node is processed next.

Algorithm:

1. Initialization: CLOSED = ∅ (empty set of visited nodes); OPEN = {S_0} (stack containing the initial state).

2. If S_0 is the goal, terminate the search.

3. If OPEN is empty, terminate - no solution exists.

4. Remove the top node from OPEN, mark it as CLOSED, and expand it.

5. For each successor:

- If it’s the goal, stop the search.
- If it’s not in CLOSED, add it to OPEN.

6. Repeat from Step 3 until the search completes.

```
def depth_first_search(graph, start, goal):
    stack = [start]  # Initialize stack with the starting node
    visited = set()  # Set to keep track of visited nodes

    while stack:
        current = stack.pop()  # Get the last node added (LIFO
behavior)

        if current == goal:
            return f"Goal {goal} found!"

        if current not in visited:
```

```
            visited.add(current)  # Mark the node as visited

            # Add all unvisited neighbors to the stack
            for neighbor in graph[current]:
                if neighbor not in visited:
                    stack.append(neighbor)

    return "Goal not found!"
```

Example:

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],
    'F': [],
    'G': []
}

print(depth_first_search(graph, 'A', 'G'))
```

Program output:

```
Goal G found!
```

Characteristics of DFS

- Backtracking - a trial-and-error method that avoids testing every solution by ruling out invalid paths.
- Completeness - DFS is guaranteed to find a solution if one exists.
- Non-optimality - DFS may not find the shortest path to the goal in graphs with cycles or multiple paths.
- Complexity - DFS has a time complexity of O(|V|+|E|), where |V| is number of vertices in the graph and |E| is number of edges in the graph.

8.3.8

What is the primary data structure used in DFS?

- Stack
- FIFO queue
- Priority queue
- Array

8.3.9

DFS variant - Depth-limited search

A modification of DFS that sets a limit on how deep the search can go. This prevents infinite loops in graphs with cycles or paths that extend indefinitely.

Nodes deeper than MAX are not expanded. If the solution lies beyond MAX depth, the search must be adjusted or restarted with a higher limit.

Main properties of DFS in this case are cost evaluation and space efficiency. The cost of a node is proportional to its depth. Nodes generated earlier at maximum depth are expanded first. DFS only stores nodes along the current branch, making it more space-efficient than BFS. Memory complexity is O(d), where d is the maximum depth of the search.

DFS in this form is useful for scenarios where finding any solution is acceptable, even if it's not the shortest. Applications include maze solving, puzzle solving, and analyzing hierarchies.

Algorithm steps are at the begin same as initialization as DFS. Then:

- Check the depth of each node before expanding it. If depth > MAX, skip expansion.
- Backtrack and repeat until a solution is found or all nodes are processed.

DFS is a powerful algorithm for exploring graphs with minimal memory overhead. However, its non-optimality and potential for infinite loops in cyclic graphs make **Depth-Limited Search (DLS)** a practical modification.

```
def depth_limited_search(graph, start, goal, max_depth):
    def dls_recursive(node, depth):
        if depth > max_depth:
            return False  # Depth limit reached

        if node == goal:
            return True  # Goal found

        visited.add(node)  # Mark the node as visited

        for neighbor in graph[node]:
            if neighbor not in visited:
                if dls_recursive(neighbor, depth + 1):
                    return True
        return False
```

```
    visited = set()
    return dls_recursive(start, 0)
```

Example:

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],
    'F': [],
    'G': []
}

print(depth_limited_search(graph, 'A', 'G', 3))  # Output:
True
print(depth_limited_search(graph, 'A', 'G', 2))  # Output:
False
```

Program output:

```
True
False
```

8.3.10

What happens if the depth limit is too low in DLS?

- The algorithm may miss the solution.
- The algorithm will still find the goal.
- The algorithm terminates immediately.
- The algorithm switches to BFS.

8.3.11

Depth-limited search advantages and shortcomming

DLS introduces a depth limit to prevent the algorithm from exploring nodes beyond a predefined depth. This limitation brings several advantages, making it suitable for specific scenarios. One significant benefit is its ability to prevent infinite loops in cyclic graphs by stopping exploration once the depth limit is reached. This feature ensures the algorithm remains efficient and avoids unnecessary computations.

Another advantage of DLS is its ability to control memory requirements. By limiting the depth of the search, DLS significantly reduces the number of nodes that need to

be stored in memory compared to standard DFS, making it a more memory-efficient option. Additionally, DLS can be particularly effective in targeted searches when there is a good estimate of how deep the goal node might be. In such cases, focusing the search within a specific range saves time and resources.

However, DLS comes with certain disadvantages. The most notable drawback is its incompleteness. If the depth limit is set too low, the algorithm may fail to find a solution, even if one exists in the graph. This makes it critical to choose an appropriate depth limit. Furthermore, DLS does not guarantee the shortest path. If the shortest path lies beyond the defined depth limit, it will be missed, making DLS less accurate in such cases.

DLS is frequently used in scenarios where graph search needs to be constrained. For instance, it is applied in games like chess or tic-tac-toe, where there is a known maximum number of moves to analyze. In very large graphs, DLS prevents time and memory exhaustion by limiting the search depth. It is also an integral part of more advanced algorithms like Iterative Deepening Depth-First Search (IDDFS), which combines DLS with the completeness of Breadth-First Search (BFS), ensuring efficient and exhaustive exploration.

8.3.12

Why is Depth-limited search useful in cyclic graphs?

- It prevents infinite loops by limiting depth.
- It ignores nodes beyond a certain depth.
- It always finds the shortest path.
- It guarantees finding a solution without depth adjustments.

Informed Search

Chapter 9

9.1 Informed search

📖 9.1.1

Informed search is a problem-solving approach in Artificial intelligence that uses additional information about the problem to guide the search. Unlike **uninformed search**, which explores the state space blindly, informed search uses **heuristics** - rules or estimates that provide hints about which paths are more promising.

For example, imagine you're playing a treasure hunt game where you're trying to find the shortest path to a hidden treasure. If you know that the treasure is to the east, you can focus your search on paths leading in that direction, saving time and effort. This guidance is the key advantage of informed search.

Heuristics, or "rules of thumb", are not guaranteed to always be correct but are useful for making better decisions. For instance, if you are navigating a map, a heuristic could be the straight-line distance to your destination, which helps estimate how far away it is. Informed search combines this heuristic knowledge with the problem's structure to find solutions more efficiently.

Informed search strategies are particularly useful when dealing with large or complex problems, where blindly exploring all possibilities would be too slow or resource-intensive.

📝 9.1.2

What is the primary difference between informed and uninformed search?

- Informed search uses heuristics to guide the search.
- Informed search avoids exploring any unnecessary paths.
- Uninformed search requires no additional information.
- Uninformed search guarantees the shortest path.

📖 9.1.3

Heuristic

A **heuristic** is a key part of informed search. It is a rule or estimate that helps decide which paths to explore first. Heuristics are not always perfect, but they are designed to make the search faster and more efficient.

For example, in a GPS system, the straight-line distance between two points (called the **Euclidean distance**) is often used as a heuristic. While it doesn't account for real-world factors like traffic or road curves, it gives a good estimate of how close you are to your destination. Similarly, in a puzzle like the 8-puzzle, a heuristic could measure the number of tiles out of place compared to the goal state.

The better the heuristic, the faster the search. A **good heuristic** is one that provides an accurate estimate of how close a state is to the goal without being too computationally expensive to calculate. A poor heuristic might mislead the search, wasting time exploring unnecessary paths.

9.1.4

What is a heuristic in informed search?

- A rule or estimate that guides the search.
- A measure of how promising a path might be.
- A random guess about the goal state.
- A guarantee of finding the shortest path.

9.1.5

Greedy Best-First search

One common informed search strategy is **Greedy Best-First search**. This algorithm uses a heuristic to always expand the node that seems closest to the goal. The idea is simple: if you're trying to reach a goal, it makes sense to prioritize the paths that appear most promising.

Greedy Best-First Search uses a **priority queue**, where nodes are sorted by their heuristic value. For example, in a maze, the heuristic might be the straight-line distance to the exit. The algorithm repeatedly selects the node with the smallest heuristic value (i.e., the node that appears closest to the goal).

However, Greedy Best-First search is not always perfect. While it is faster than uninformed search methods, it does not guarantee finding the shortest path. It might follow a promising path that leads to a dead-end, forcing the algorithm to backtrack and explore other options.

9.1.6

What does Greedy Best-First search prioritize when selecting nodes to expand?

- Nodes that appear closest to the goal based on heuristics.
- Nodes with the lowest cost from the start.
- Random nodes in the graph.
- All nodes at the same depth.

9.1.7

A* search algorithm

The A* search algorithm is one of the most popular informed search methods because it combines the strengths of other algorithms. It considers both the **cost of**

the path so far and the **estimated cost to the goal**. These two components are combined into a formula:

f(n) = g(n) + h(n)

Where:

- f(n) - total estimated cost of a path through node n.
- g(n) - actual cost from the start to node n.
- h(n) - heuristic estimate of the cost from nto the goal.

By balancing actual costs with heuristic estimates, A* guarantees finding the shortest path if the heuristic is **admissible** (never overestimates the cost) and **consistent** (satisfies certain mathematical properties).

For example, in a map navigation problem, g(n) might represent the distance traveled so far, and h(n) could be the straight-line distance to the destination. A* finds the optimal path efficiently, making it a preferred choice for many real-world problems.

9.1.8

What does the A algorithm use to select nodes for expansion?

- The sum of the actual path cost and heuristic estimate.
- Only the heuristic estimate.
- Only the actual path cost.
- The total number of expanded nodes.

9.1.9

Applications

Informed search is used in many real-world applications where finding the best solution efficiently is critical. Some examples include:

- GPS navigation - algorithms like A* help calculate the fastest or shortest route by combining travel distance and estimated traffic.
- Robotics where informed search is used to plan energy-efficient paths for robots in complex environments.
- Puzzle solving - heuristic-based search solves puzzles like the 8-puzzle or Sudoku quickly by focusing on promising moves.
- Video game uses AI informed search to make intelligent decisions, like finding the shortest path to a player or planning strategies in games like chess.

These examples show how informed search helps solve problems in a smarter way, saving both time and computational resources.

9.1.10

Which of the following is an application of informed search?

- GPS navigation
- Robotic path planning
- Guessing in puzzles
- Exploring all possible paths equally

9.2 Greedy Best-First search

9.2.1

Greedy Best-First search (GBFS) is a search algorithm that uses a **heuristic** to decide which path to explore. It always chooses the node that appears to be **closest to the goal**, based on the heuristic value. This makes GBFS faster than uninformed search methods like Breadth-First search because it focuses on the most promising paths.

Imagine trying to find the shortest way out of a maze. If you know which direction leads closer to the exit, you can follow that path instead of blindly exploring every possibility. This "greedy" approach is what GBFS does - it prioritizes the shortest-looking route to the solution.

However, GBFS is not perfect. Since it only considers how close a node seems to the goal, it can get stuck in dead-ends or take longer paths, missing the optimal solution.

9.2.2

What does Greedy Best-First search prioritize when selecting nodes?

- Nodes that appear closest to the goal based on the heuristic.
- Nodes with the lowest total cost from the start.
- Nodes chosen at random.
- Nodes at the same depth in the search tree.

9.2.3

Project: Greedy Best-First search algorithm explanation

The Greedy Best-First Search algorithm works as follows:

1. Start with the initial node - add the starting node to a priority queue, where nodes are sorted by their heuristic values (lower is better).
2. Expand the most promising node - remove the node with the lowest heuristic value from the queue and expand its neighbors.

3. Update the priority queue - add unvisited neighbors to the queue, sorting them based on their heuristic values.
4. Repeat until goal is found - continue expanding nodes with the smallest heuristic values until the goal is reached or the queue is empty.

The algorithm uses a **priority queue** to efficiently manage nodes. It does not consider the path cost so far, focusing only on the heuristic estimate.

Imagine we have a graph where nodes represent locations, and edges represent possible paths between them. Each node has a heuristic value representing the estimated distance to the goal.

```
        A
       / \
      B   C
     / \   \
    D   E   F
             /
            G

Goal node: G
Start node: A
Heuristic values: A: 6, B: 4, C: 5, D: 4, E: 7, F: 3, G: 0
```

Step-by-step procedure

1. Initialization

- start at node A; add A to the priority queue with its heuristic value:
- Queue: [(6,A)]

2. Expand node A

- remove A from the queue (lowest heuristic).
- Expand its neighbors B and C. Add them to the queue sorted by their heuristic values:
- Queue: [(4,B),(5,C)]

3. Expand node B

- Remove B from the queue (lowest heuristic).
- Expand its neighbors D and E. Add them to the queue:
- Queue: [(4,D),(5,C),(7,E)

4. Expand node D

- Remove D from the queue (lowest heuristic).

- Node D has no neighbors, so move to the next node
- Queue: [(5,C),(7,E)]

5. Expand node C

- Remove C from the queue (lowest heuristic).
- Expand its neighbor F and add it to the queue
- Queue: [(3,F),(7,E)]

6. Expand node F

- Remove F from the queue (lowest heuristic).
- Expand its neighbor G (the goal)
- Queue: [(0,G),(7,E)]

7. Expand node G

- Remove G from the queue. Since G is the goal node, the search terminates.

Result:

- Path Found: A → C → F → G

GBFS always selects the node with the lowest heuristic value, focusing on the path that seems closest to the goal.

Pseudocode:

```
def greedy_best_first_search(graph, start, goal, heuristic):
    from queue import PriorityQueue

    frontier = PriorityQueue()
    frontier.put((heuristic[start], start))
    explored = set()

    while not frontier.empty():
        _, current = frontier.get()

        if current == goal:
            return f"Goal {goal} found!"

        explored.add(current)

        for neighbor in graph[current]:
            if neighbor not in explored:
                frontier.put((heuristic[neighbor], neighbor))
```

```
    return "Goal not found!"
```

GBFS does not consider the cost of the path so far. It only uses the heuristic to decide which node to expand. The algorithm may take a longer or suboptimal path because it doesn't account for actual distances, only estimates.

📝 9.2.4

What data structure is used in Greedy Best-First search?

- Priority queue
- Linked list
- Stack
- FIFO queue

📝 9.2.5

Why did GBFS expand node F before node E?

```
       A
      / \
     B   C
    / \   \
   D   E   F
          /
         G

Goal node: G
Start node: A
Heuristic values: A: 6, B: 4, C: 5, D: 4, E: 7, F: 3, G: 0
```

- F had a lower heuristic value than E.
- F was added to the queue first.
- E had no neighbors.
- The goal was directly connected to F.

🕮 9.2.6

Advantages and disadvantages

Greedy Best-First Search has its advantages and disadvantages:

Strengths:

- Speed - by focusing on the most promising paths, GBFS can quickly find solutions in many cases.

- Simplicity - the algorithm is straightforward and easy to implement.
- Low memory usage - it only needs to store the current frontier of nodes, not the entire search tree.

Weaknesses:

- No guarantee of optimality - GBFS may find a solution, but it might not be the shortest or best one.
- Risk of dead ends - the algorithm can get stuck exploring a path that looks good initially but does not lead to the goal.
- Heuristic dependency - the quality of the solution depends heavily on the heuristic. A poor heuristic can lead to inefficient searches.

9.2.7

What is the limitation of Greedy Best-First search?

- It can get stuck in dead ends.
- It always finds the shortest path.
- It does not prioritize nodes with low heuristic values.
- It guarantees optimal solutions.

9.2.8

Project: GBFS graph application

Let's apply GBFS to a simple graph:

```
Graph:
A -> B (2), C (1)
B -> D (4), E (3)
C -> F (2), G (1)

Heuristic:
A: 3, B: 4, C: 2, D: 6, E: 5, F: 2, G: 1

Start node: A
Goal node: G
```

- Start at A. Add A to the queue: (3,A)(3, A)(3,A).
- Expand A. Add B and C to the queue: (2,C),(4,B)(2, C), (4, B)(2,C),(4,B).
- Expand C (lowest heuristic). Add F and G: (1,G),(4,B),(2,F)(1, G), (4, B), (2, F)(1,G),(4,B),(2,F).
- Expand G (goal reached).

Result path: A → C → G.

9.2.9

Why did GBFS expand C before B in the example?

- C had a lower heuristic value than B.
- C was at the same depth as B.
- B had no successors.
- C was randomly selected.

9.2.10

Project: Finding the way between two cities (Greedy Best-First search)

Imagine you want to travel between two cities. The road network connecting the cities can be represented as a **graph**, where:

- Nodes (or vertices) represent the cities.
- Edges (or lines) represent the roads connecting the cities.
- Each edge has a cost or "heuristic“, such as distance, travel time, or fuel cost.

The goal is to find the path from the **start city** to the **destination city** that looks the "best" based on the heuristic. Greedy Best-First search is a search algorithm that always chooses the path that appears to bring you closest to the destination based only on the heuristic at each step. It does not guarantee the shortest path, but it is fast and simple to implement.

The algorithm uses a heuristic h(n), which estimates the distance from the current city (n) to the destination city.

```
From  To  Heuristic cost h(n)
 A    B    4
 A    C    2
 B    D    5
 C    D    8
 B    E    6
 D    E    1

Start city: A
Destination city: E
```

1. Start at the initial city (A).

- From A, look at the neighboring cities (B and C) and choose the one with the lowest heuristic.
- Heuristic values: h(B) = 4, h(C) = 2. **Choose C** (because 2 is smaller than 4).

2. Move to C.

- From C, look at its neighbors (A and D). Ignore A since we already visited it.
- Heuristic value: h(D) = 8. **Move to D.**

3. Move to D.

- From D, look at its neighbors (B and E).
- Heuristic values: h(B) = 4, h(E) = 1. **Choose E** (because 1 is smaller than 4).

4. Destination reached!

- You are now at city E.

9.3 A* search algorithm

📖 9.3.1

The A* search algorithm is one of the most powerful and widely used informed search strategies. It is designed to find the shortest path from a starting node to a goal node in a graph or tree. A* combines two key components to guide its search:

- Actual path cost (g(n)) is the cost of the path from the starting node to the current node.
- Heuristic estimate (h(n) is the estimated cost from the current node to the goal.

These two values are summed to compute the total cost:

f(n) = g(n) + h(n)

This makes A* both efficient and optimal. By balancing the actual cost and estimated cost, A* avoids unnecessary exploration while guaranteeing the shortest path if the heuristic is admissible (never overestimates the actual cost).

Example: Imagine navigating a city map. g(n) represents the distance traveled so far, while h(n) estimates the remaining distance to your destination. A* finds the shortest route by combining these values.

📝 9.3.2

What does f(n) represent in A* search?

- The total estimated cost of the path through the node.
- The depth of the node in the tree.
- The heuristic value alone.

- The actual path cost alone.

9.3.3

Project: How does A* search work

A* works systematically, selecting nodes based on their f(n) values, where lower values are prioritized. The algorithm maintains two data structures:

- **Open list (frontier)** - nodes that are yet to be explored, stored in a priority queue sorted by f(n).
- **Closed list (explored set)** - nodes that have already been fully explored.

Procedure:

1. Initialize

- Add the starting node to the open list with f(n) = g(n) + h(n).

2. Expand nodes

- Remove the node with the lowest f(n) from the open list.
- If it's the goal node, terminate the search - solution found!
- Otherwise, expand its neighbors, calculate their f(n), and add them to the open list.

3. Update costs

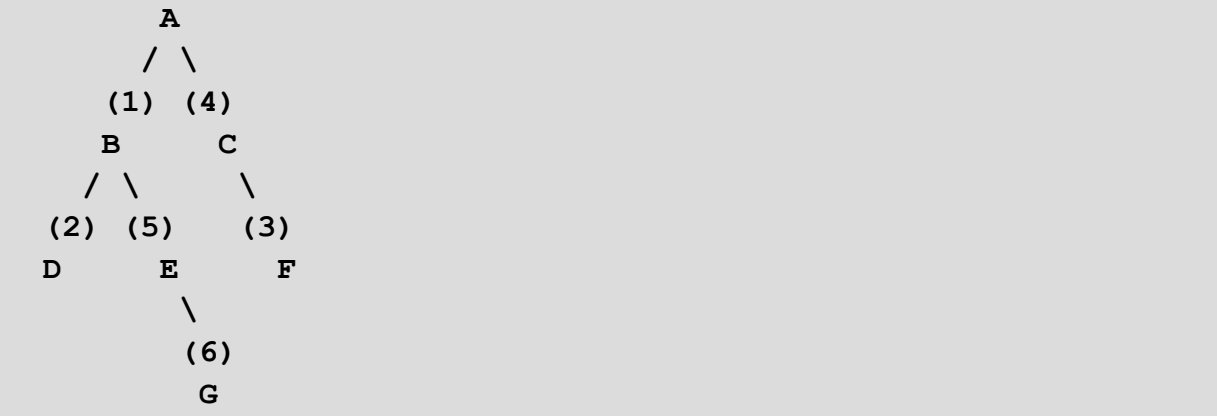
- If a node is already in the open list but a cheaper path to it is found, update its g(n) and recalculate f(n).

4. Repeat

- Continue until the goal node is found or the open list is empty.

Let's explore how the **A*** search algorithm works step by step with an example.

- We have a graph where nodes represent locations, and edges represent paths between them. Each edge has a cost, and each node has a heuristic value representing the estimated cost to the goal.



- Costs (g) - the numbers on edges represent the cost to move from one node to another.
- Heuristic values (h(n)):

```
A: 7, B: 6, C: 5, D: 3, E: 2, F: 6, G: 0

Start node: A
Goal node: G
```

A* search uses the formula f(n) = g(n) + h(n), where:

- g(n) is actual cost from the start to node n.
- h(n) is estimated cost (heuristic) from n to the goal.
- f(n) is total estimated cost of the path through n.

A* uses a **priority queue** to expand the node with the smallest f(n). by following procedure:

1. Initialization:

- Start at node A.
- Add A to the queue with f(A) = g(A) + h(A) = 0 + 7 =7
- Queue: [(7,A)]

2. Expand node A:

- Remove A from the queue.
- Expand neighbors B and C:
- f(B) = g(B) + h(B) = 1 + 6 = 7
- f(C) = g(C) + h(C) = 4 + 5= 9
- Add nodes to the queue - Queue: [(7,B),(9,C)]

3. Expand node B:

- Remove B from the queue (smallest f).

- Expand neighbors D and E:
- f(D) = g(D) + h(D) = (1 + 2) + 3 = 6
- f(E) = g(E) + h(E) = (1 + 5) + 2 = 8
- Add nodes to the queue - Queue: [(6,D),(9,C),(8,E)]

4. Expand node D:

- Remove D from the queue.
- Node D has no neighbors, so continue: Queue: [(8,E),(9,C)]

5. Expand node E:

- Remove E from the queue.
- Expand its neighbor G:
- f(G) = g(G) + h(G) = (1 + 5 + 6) + 0 = 12
- Add G to the queue - Queue: [(9,C),(12,G)]

6. Expand node C:

- Remove C from the queue.
- Expand its neighbor F:
- f(F) = g(F) + h(F) = (4 + 3) + 6 = 13
- Add F to the queue - Queue: [(12,G),(13,F)]

7. Expand node G:

- Remove **G** from the queue.
- Since **G** is the goal, the search terminates.

Result:

- Path found: A → B → E → G
- Total cost: 12

A* finds the optimal path by balancing actual costs (g) and estimated costs (h).

Program code:

```
def a_star_search(graph, start, goal, heuristic):
    from queue import PriorityQueue

    frontier = PriorityQueue()
    frontier.put((0, start))
    came_from = {}
    cost_so_far = {start: 0}

    while not frontier.empty():
```

```
        _, current = frontier.get()

        if current == goal:
            return reconstruct_path(came_from, start, goal)

        for neighbor, cost in graph[current]:
            new_cost = cost_so_far[current] + cost
            if neighbor not in cost_so_far or new_cost <
cost_so_far[neighbor]:
                cost_so_far[neighbor] = new_cost
                priority = new_cost + heuristic[neighbor]
                frontier.put((priority, neighbor))
                came_from[neighbor] = current

    return "Goal not found!"

def reconstruct_path(came_from, start, goal):
    path = []
    current = goal
    while current != start:
        path.append(current)
        current = came_from[current]
    path.append(start)
    return path[::-1]
```

A* guarantees finding the shortest path if the heuristic is admissible (never overestimates the cost). By combining g(n) and h(n), A* avoids dead ends and explores efficiently.

9.3.4

Why does A* guarantee the shortest path if the heuristic is admissible?

- It combines actual path cost with heuristic values.
- It overestimates the cost to reach the goal.
- It expands nodes randomly.
- It prioritizes nodes based only on depth.

9.3.5

What happens when A* finds a cheaper path to a node already in the open list?

- It updates the node's g(n) and f(n).
- It ignores the new path.
- It removes the node from the list.
- It starts the search over.

📖 9.3.6

A* has several important properties that make it an effective search algorithm:

- Optimality - A* guarantees finding the shortest path if the heuristic is admissible (it never overestimates the actual cost) and consistent (it satisfies a mathematical property ensuring reliability).
- Efficiency - A* avoids unnecessary exploration by focusing on nodes that minimize f(n). This makes it faster than uninformed methods like BFS or DFS.
- Time complexity - $O(b^d)$, where b is the branching factor and d is the depth of the solution.
- Space complexity - $O(b^d)$, as A* must store all nodes in memory.
- Heuristic dependency - the quality of the heuristic greatly impacts the algorithm's performance. A good heuristic leads to faster searches, while a poor heuristic may result in slower performance or even incorrect results.

📝 9.3.7

What ensures that A* finds the shortest path?

- An admissible heuristic that never overestimates the cost.
- A priority queue sorted by g(n) alone.
- Ignoring the g(n)values.
- Exploring nodes in random order.

📝 9.3.8

Example: The way A-G

In a following graph find the "best" way A->G.

1. Start at A

- f(A) = g(A) + h(A) = 0 + 7 = 7

2. Expand A neighbors:

- f(B) = g(B) + h(B) = 1 + 6 = 7
- f(C) = g(C) + h(C) = 4 + 5 = 9
- Add to open list: (7,B), (9,C)

3. Expand B neighbors:

- f(D) = g(D) + h(D) = (1 + 2) + 3 = 6
- f(E) = g(E) + h(E) = (1 + 5) + 2=8
- Add to open list: (6,D), (9,C), (8,E)

4. Expand D - no neighbors. Continue.

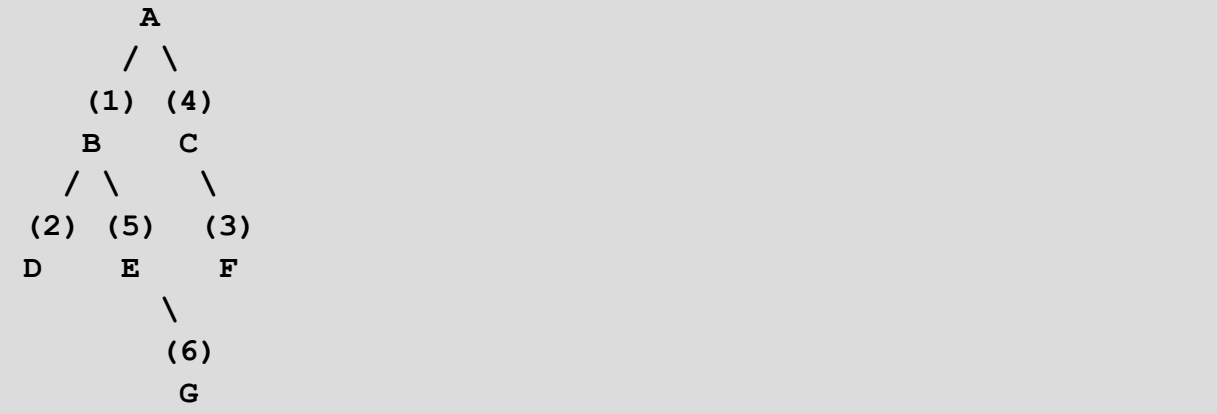
5. Expand E - neighbor:

- f(G) = g(G) + h(G) = (1 + 5 + 6) + 0 = 12
- Add to open list: (9,C), (12,G)

6. Expand G - Goal found!

9.3.9

Why did A* expand D before E in the example?



- D had a lower f(n) value than E.
- D was closer to the goal.
- E had no neighbors.
- D was added to the queue first.

9.3.10

Project: Finding the way between two cities (A* search)

Imagine you want to travel between two cities. The road network connecting the cities can be represented as a **graph**, where:

- Nodes (or vertices) represent the cities.
- Edges (or lines) represent the roads connecting the cities.
- The **A*** algorithm improves upon Greedy Best-First Search by combining the heuristic h(n) with the actual cost g(n) to reach the current city. The total cost f(n) = g(n) + h(n)

The goal is to find the path from the **start city** to the **destination city** that looks the "best" based on the heuristic.

```
From    To    Cost g(n)    Heuristic cost h(n)
  A      B       1              4
  A      C       2              2
  B      D       4              5
  C      D       7              8
  B      E       6              6
  D      E       2              1

Start city: A
Destination city: E
```

1. Start at node A.

- f(A) = g(A) + h(A)
- g(A) = 0 (cost to itself), h(A) = 0 (destination not yet estimated).
- Evaluate neighbors (B and C): f(B) = g(A) + Cost(A→B) + h(B) = 0 + 1 + 4 = 5
- f(B) = g(A) + Cost(A→B) + h(B) = 0 + 1 + 4 = 5
- f(C) = g(A) + Cost(A→C) + h(C) = 0 + 2 + 2 = 4
- Choose C (lowest fff).

2. Move to C.

- Update the actual cost g(C) = g(A) + Cost(A→C) = 2
- Evaluate neighbors (A and D). Ignore A (already visited).
- f(D) = g(C) + Cost(C→D) + h(D) = 2 + 7 + 8 = 17
- **Choose D** (next unexplored node with lowest f).

3. Move to D.

- Update the actual cost g(D) = g(C) + Cost(C→D) = 9
- Evaluate neighbors (B and E).
- f(B) = g(D) + Cost(D→B) + h(B) = 9 + 4 + 4 = 17
  (already visited; skip updating)
- f(E) = g(D) + Cost(D→E) + h(E) = 9 + 2 + 1 = 12
- **Choose E** (lowest fff).

4. Move to E.

- Destination reached with g(E) = g(D) + Cost(D→E) = 11.

Summarisation:

Step	Current	Explored	g(n)	h(n)	f(n)=g(n)+h(n)	decision
1	A	B,C	0	4,2	5(B),4(C)	Move to C
2	C	D		2	8	17	Move to D
3	D	E		9	1	12	Move to E

A* combines the actual cost (g) with the heuristic (h) to make more informed decisions. Unlike Greedy Best-First Search, A* balances exploration of paths based on both the cost already incurred and the estimated cost to the goal. In this example, A* reaches the destination more reliably and often with the shortest path.

9.4 Genetic algorithms

🕮 9.4.1

A Genetic algorithm (GA) is a type of heuristic search inspired by the process of natural evolution. It reflects Charles Darwin's theory of natural selection, where the "fittest" individuals in a population are more likely to survive and reproduce. In GAs, potential solutions to a problem are treated as individuals in a population. The goal is to evolve these solutions over multiple generations to find the best or optimal solution.

The key idea is simple: successful solutions survive and reproduce, while unsuccessful ones "die out". Over time, the population adapts and evolves toward better solutions through the processes of selection, crossover (recombination), and mutation.

For example, imagine solving a puzzle. Each possible arrangement of puzzle pieces is a "chromosome", and the algorithm evolves these chromosomes to improve their fitness, or how close they are to solving the puzzle.

9.4.2

What concept from nature does a Genetic algorithm mimic?

- Mutation and natural selection
- Gravity and momentum
- Random guessing
- Evolution of weaker solutions

9.4.3

Principles of GA

Genetic algorithms operate on populations of solutions, treating each solution as a **chromosome**. These chromosomes are typically represented as **binary strings**, where each bit (0 or 1) encodes some information about the solution.

Key principles are base on following elements:

- Chromosomes and population - a chromosome represents a single solution. A population consists of many chromosomes, representing a diverse set of solutions.
- Fitness function - each chromosome is evaluated using a fitness function, which measures how good a solution it is to the given problem. Chromosomes with higher fitness are more likely to survive and reproduce.
- Natural selection - chromosomes with higher fitness are selected to reproduce, ensuring that "better" solutions have a higher chance of passing their "genes" to the next generation.
- Crossover combines parts of two chromosomes (parents) to create new chromosomes (offspring).
- Mutation randomly alters parts of a chromosome to introduce variety and prevent the algorithm from getting stuck in local optima.

9.4.4

What is the role of the fitness function in a Genetic algorithm?

- To evaluate how good a solution is.
- To randomly change chromosomes.
- To combine chromosomes into a population.
- To ensure weaker solutions dominate the population.

📖 9.4.5

Representation in GA

In GA, solutions are represented as **binary strings** called chromosomes. Each chromosome is a fixed-length string of bits (0s and 1s), where each bit encodes information about the solution.

Consider a problem where a chromosome is a binary string of length 5:

```
X = [1, 0, 1, 1, 0]
```

Here, each position in the string represents a specific feature or parameter of the solution.

Why binary strings? The use of binary strings simplifies the algorithm by standardizing all chromosomes to the same length and format. However, this also means that GAs are only applicable to problems that can be represented in binary form. For instance, a numerical optimization problem can map values to binary, but a problem requiring complex symbolic representation may not fit this model.

While binary strings are a powerful way to represent solutions in GA, they also impose limitations. Not all problems can be easily converted into binary form. For example:

- A problem that requires representing continuous variables might need additional mapping to binary.
- A problem with symbolic or textual data might not fit into the binary model at all.

This means that GA work best for problems that can naturally encode their solutions as fixed-length binary strings. For other problems, adaptations to the algorithm are needed.

Despite this limitation, GAs are widely applicable in areas like optimization, scheduling, and machine learning, where binary representation is practical.

📝 9.4.6

What is a chromosome in a Genetic algorithm?

- A binary string representing a solution.
- A random sequence of numbers.
- A fitness function.
- A solution without structure.

9.4.7

Why are Genetic algorithms limited to specific problems?

- They require solutions to be represented as binary strings.
- They cannot work with optimization problems.
- They avoid using a fitness function.
- They depend on random guessing alone.

9.4.8

Project: How binary strings work in GA

In GAs, solutions are represented as binary strings called chromosomes. These strings encode information about the solution in a format that the algorithm can process. Each bit (0 or 1) in the string represents a feature or parameter of the solution, and the entire string represents one possible solution to the problem.

Let's use a simple problem to demonstrate how binary strings work. Suppose we want to maximize the function:

$f(x) = x^2$

where x is an integer in the range [0,15]. We'll use GA to find the value of xxx that gives the maximum value of f(x).

Each possible value of xxx is encoded as a **binary string** of length 4. For example:

- x = 0 ⇒ 0000
- x = 1 ⇒ 0001,...
- x = 15 ⇒ 1111

This encoding allows us to represent all integers in the range [0,15] using binary strings.

A **population** of chromosomes is generated randomly. Each chromosome is a binary string representing a possible value of xxx. For example, the initial population might be:

```
1010, 0011, 1100, 0111
```

- these strings represent the values: x = 10, x = 3, x = 12, x = 7

The **fitness function** evaluates how good each chromosome is. In this case, the fitness of a chromosome is the value of the function f(x):

- $f(10) = 10^2 = 100$
- $f(3) = 3^2 = 9$

- $f(12) = 12^2 = 144$
- $f(7) = 7^2 = 49$

Fitness scores:

```
1010 → 100, 0011 → 9, 1100 → 144, 0111 → 49
```

Selection - chromosomes with higher fitness are more likely to be selected for reproduction. For example:

- 1100 (fitness 144) and 1010 (fitness 100) are selected due to their higher scores.

Crossover - selected chromosomes exchange parts of their binary strings to create offspring. For example, if we cross 1100 and 1010 at the second position:

```
Parent 1: 1100
Parent 2: 1010
Crossover point: after second bit
Child 1: 1110
Child 2: 1000
```

Mutation - introduces random changes to maintain diversity in the population. For example, flipping the third bit of 1110 gives:

```
Original:1110 → Mutated:1100
```

New population - consists of the offspring and possibly some unmutated parents:

```
1110,1000,1100,1010
```

Iteration - the algorithm repeats the process of selection, crossover, and mutation over multiple generations. Over time, the population evolves toward the optimal solution.

Result - after several generations, the algorithm converges on the chromosome representing x = 15 (binary: 1111), which gives the maximum value of f(x) = 225.

9.4.9

What does each bit in a binary string represent in GA?

- A feature or parameter of the solution.
- A random guess about the solution.
- A fitness value of the solution.
- A mutation point in the chromosome.

9.4.10

Project: GA example - Scheduling tasks

Let’s understand how GAs work with a practical example like scheduling tasks.

We have:

- five tasks (T1, T2, T3, T4, T5)
- two workers (W1, W2)
- Each task must be assigned to one of the workers, and your goal is to minimize the total time required to complete all tasks.

Task durations (in hours):

```
T1: 2, T2: 4, T3: 6, T4: 8, T5: 3
```

If worker W1 gets tasks [T1, T2], and worker W2 gets tasks [T3, T4, T5], the total time is:

```
max(Sum of W1’s tasks, Sum of W2’s tasks) = max(6,
17)=17 hours
```

Our goal is to minimize this total time using a GA.

1. Encoding solutions as chromosomes

Each chromosome represents an assignment of tasks to workers. A chromosome is encoded as a binary string:

- 0 means the task is assigned to W1.
- 1 means the task is assigned to W2.

Example chromosome:

```
Chromosome: 01011
Interpretation:
- T1 → W1
- T2 → W2
- T3 → W1
- T4 → W2
- T5 → W2
```

2. Initial population

The algorithm starts with a **random population** of solutions (chromosomes). For example:

```
Population:
10101, 01011, 00110, 11000
```

3. Fitness function

The fitness function evaluates how good each chromosome is. In this case, it calculates the total time based on task assignments:

- Sum the durations for W1 and W2.
- Compute the maximum of the two sums.

Example for chromosome 01011:

- W1: T1 (2), T3 (6) → Total = 8 hours
- W2: T2 (4), T4 (8), T5 (3) → Total = 15 hours
- Fitness = max(8, 15) = 15

4. Selection

Chromosomes with lower fitness (better solutions) are more likely to be selected for reproduction. For example:

```
Selected parents: 01011 (Fitness 15), 11000 (Fitness 14)
```

5. Crossover

Crossover combines parts of two parents to create offspring. For example, crossing 01011 and 11000 at the third bit:

```
Parent 1: 01011
Parent 2: 11000
Crossover Point: After third bit
Child 1: 01000
Child 2: 11011
```

6. Mutation

Mutation introduces random changes to maintain diversity. For example, flipping the second bit of 11011:

```
Original: 11011 → Mutated: 10011
```

7. New Population

The offspring form the new population. For example:

```
New Population: 01000, 10011, 00110, 10101
```

8. Iteration

Steps 3–7 are repeated for multiple generations. Over time, the population evolves toward better solutions. After several iterations, the algorithm might produce a chromosome like:

```
Optimal Chromosome: 10101
Interpretation:
- W1: T1 (2), T3 (6), T5 (3) → Total = 11 hours
- W2: T2 (4), T4 (8) → Total = 12 hours
Fitness = max(11, 12) = 12
```

Final result - the best chromosome found represents an optimized task assignment with a total time of 12 hours.

9.5 Fitness function in GA

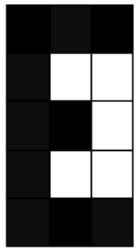
🕮 9.5.1

Real application: Optimizing font design using GAs

This example of using binary strings to represent characters (like the letter E) can be extended into all characters to optimizing font design.

Designing a visually appealing and functional font set involves balancing aesthetics, readability, and efficient storage of character representations. Each character (like E) must fit into a predefined grid size (e.g., 5x7 or 3x5) to ensure uniformity in fonts used for digital displays, low-resolution devices, or pixel art. The goal is to automatically optimize the representation of these characters using GAs.

Assume that the solution state is the letter E. On an area of 3x5 squares, this letter could be drawn as follows in 5x3 matrix:



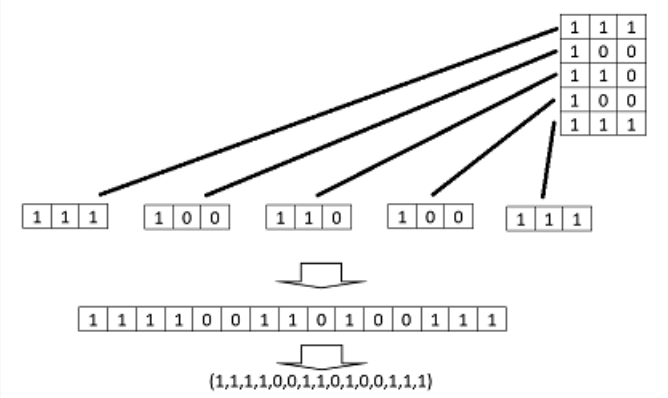
If we replace the black squares with the number 1, we can represent the letter E using a matrix as follows:

1	1	1
1		
1	1	
1		
1	1	1

Subsequently, we add the number 0 to the white, empty squares. This way we get a binary (consisting only of zeros and ones) representation of the character E. However, this representation is a matrix, not a string.

1	1	1
1	0	0
1	1	0
1	0	0
1	1	1

We can "store" each row of the matrix in sequence in one string. This is how we get a binary string representing the character E.



9.5.2

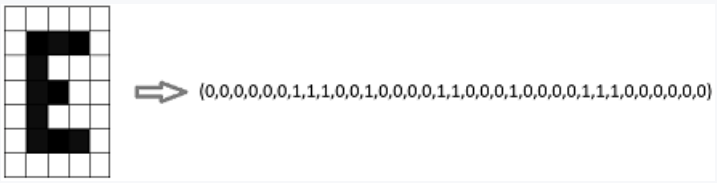
What character does the given binary matrix represent?

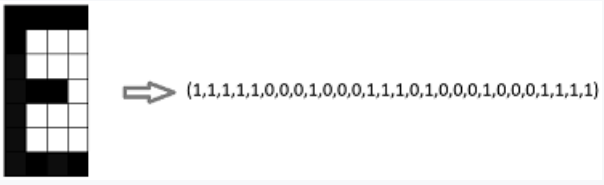
1	0	0
1	0	0
1	0	0
1	0	0
1	1	1

- character L
- character I
- character E
- character 0

9.5.3

The length of the vector depends on the space in which the problem solution states are represented. For example the character E can be displayed on a 3x5 surface (vector of length 15) as well as on a 5x7 surface (vector of length 35), on a 4x7 surface (vector of length 28), etc.





9.5.4

1	0	0
1	0	0
1	0	0
1	0	0
1	1	1

Which of the given strings represents letter in image?

- (1,0,0,1,0,0,1,0,0,1,0,0,1,1,1)
- (1,1,1,1,0,0,1,0,0,1,0,0,1,1,1)
- (1,0,0,1,0,0,1,1,1,0,0,0,1,1,1)
- (1,0,0,1,1,1,1,0,0,1,1,1,1,0,0)

9.5.5

In GAs, solutions are often represented as **binary strings** (sequences of 0s and 1s). The **length of the binary string**, or vector, depends on the space in which the solution states are represented. This is especially important when dealing with problems that involve graphical or spatial representation, like designing characters.

The binary string's length is directly tied to the **dimensions of the grid** used for representing the problem's solution. For instance:

- A **3x5 grid** can display 15 squares (3 rows and 5 columns), so the binary string must have 15 bits.
- A **5x7 grid** can display 35 squares, requiring a binary string of 35 bits.
- A **4x7 grid** would require a string of 28 bits.

Each bit in the string corresponds to a square on the grid:

- A 1 represents a filled square (e.g., part of the character).
- A 0 represents an empty square.

In GAs, the **grid size must remain constant** throughout the task. If the algorithm is working on a 3x5 grid, all solutions in the population must be encoded as binary strings of length 15. This ensures:

1. Uniformity - all solutions (chromosomes) have the same structure and length.
2. Comparability - solutions can be evaluated and manipulated consistently.
3. Integrity - the grid size reflects the nature of the problem, as chosen by the creator of the algorithm.

Consistency in grid size ensures that every solution represents the problem correctly. For example:

- If a 3x5 grid is used to design characters, switching to a 5x7 grid mid-task would invalidate the evaluation, selection, crossover, and mutation processes.

Thus, the binary string length is not arbitrary—it is determined by the **problem's requirements** and must remain constant.

If we are encoding the letter **E** on a 3x5 grid:

```
111
100
111
100
111
```

This would be represented as the binary string:

```
111100111100111
```

For a 5x7 grid, the same character might look more detailed, requiring 35 bits to represent it.

📝 9.5.6

Why must the length of binary strings remain constant in GAs?

- To ensure solutions are uniform and comparable.
- To allow random string lengths for diversity.
- To make all solutions shorter over time.
- To switch grid sizes during the task.

🕮 9.5.7

In a GA, each potential solution to the problem is represented as a chromosome. A chromosome is a binary string (X) of fixed length N, where each bit (0 or 1) encodes some information about the solution. The length N is constant across all chromosomes in the algorithm, ensuring uniformity in how solutions are represented.

A chromosome is essentially a blueprint for one possible solution to the problem. For example:

- In a **character design task**, a chromosome might represent a 3x5 grid as a binary string of length 15.
- In an **optimization problem**, a chromosome could represent the configuration of parameters being optimized.

The binary representation allows the algorithm to manipulate solutions systematically through genetic operations like selection, crossover, and mutation.

A **population (P)** is a group of chromosomes (solutions) that the algorithm works with at any given time. Each population contains M chromosomes, ensuring diversity among potential solutions. Over time, the algorithm evolves these populations to improve the quality of solutions.

1. Initial population

- The first population is generated randomly to ensure a wide variety of starting points.
- Example:

```
111100111100111
101010101010101
000111000111000
```

2. Evolution of populations:

- Populations evolve over multiple **generations**. In each generation chromosomes are evaluated using a **fitness function** to measure their quality.
- The best chromosomes are selected to create a new population through **crossover** and **mutation**.
- Poorly performing chromosomes are replaced by better ones.

Population P_1 (after 1st generation):

```
111100111100111
101110101100111
111101001111000
```

Having a population of chromosomes instead of a single solution helps the algorithm:

- The diversity in the population allows the algorithm to search more broadly and avoid getting stuck in local optima.
- Operations like crossover and mutation ensure that new, potentially better solutions are continuously introduced.
- Each generation refines the population, gradually evolving better solutions.

The GA repeatedly creates new populations by evaluating, selecting, and modifying chromosomes. Just like in biological evolution, this process continues until:

- A desired fitness level is reached.
- A specific number of generations have passed.
- The population stabilizes with no significant improvement.

9.5.8

What is the main role of a population in a GA?

- To store multiple potential solutions at a time.
- To hold only one solution for the entire algorithm.
- To ensure solutions change randomly.
- To switch the representation of chromosomes during the task.

9.5.9

Fitness function

At first glance, it might seem unnecessary to use a GA if we already know the solution. However, GAs are not designed to work with predefined solutions; instead, they are powerful tools for finding unknown solutions to complex problems. Let's explore why GAs are useful even when we have a "benchmark" solution for comparison. In the case of representing the letter **E** using a binary matrix, the **ideal binary matrix** provides a **target** or benchmark for evaluation. However, the challenge lies not in knowing the target but in figuring out the **best way to reach it**. This is where GAs come into play.

Lets go to propose it. Key fitness criteria are:

- Visual similarity - compare the binary matrix of a chromosome to a predefined "ideal" matrix for the target character. The more bits that match the ideal matrix, the higher the fitness score. Fitness calculation: Count the number of matching bits (correct pixels). In this case, 14 out of 15 bits match, resulting in a high fitness score.
- Legibility - assess the overall readability of the character. This involves checking for gaps (missing parts of the character that make it unrecognizable, e.g., a broken horizontal bar in the E), noise (extra bits that distort the intended shape - ff a chromosome introduces gaps or fills unrelated pixels, its fitness score should decrease).
- Symmetry - evaluate the balance of the character, especially for shapes that require symmetry (like E). A visually balanced character is more appealing and legible, e.g. if the top and bottom horizontal bars of E differ significantly, reduce the fitness score

A simple fitness function could combine these criteria as:

```
Fitness = Matching_bits_score + Symmetry_bonus –
Gap/Noise_penalty
```

where:

- Matching_bits_score is number of bits matching the ideal matrix.
- Symmetry_bonus add points for well-balanced characters.
- Gap/Noise_penalty subtract points for missing parts or distortions.

Calculation

Ideal binary string for E (3x5 Grid):

```
111100111100111
```

Candidate chromozome:

```
111100110100111
```

- Matching bits - matches 14 out of 15 bits → Score = 14.
- Symmetry check - top and bottom bars are symmetrical → Bonus = 1.
- Gap/Noise penalty - one incorrect bit creates a small gap → Penalty = -1.
- Final fitness score: Fitness = 14 + 1 − 1 = 14

The fitness function ensures that:

- High-quality solutions (well-drawn characters) are selected for reproduction.
- Poor solutions (unrecognizable or distorted characters) are removed over generations.
- The algorithm converges toward an ideal representation of the character.

9.5.10

What is the primary role of the fitness function in the character representation problem?

- To evaluate how well a binary string matches the ideal character representation.
- To randomly select solutions without evaluation.
- To ensure chromosomes are completely random.
- To adjust the grid size dynamically.

Natural Language Processing

Chapter 10

10.1 Introduction

📖 10.1.1

Natural language processing (NLP) is a part of computer science that focuses on helping machines understand and work with human language. Just like humans can talk, read, and write, NLP helps computers do these things too. It allows computers to understand text, speech, and even respond intelligently. For example, when you ask your phone, "What's the weather today?" and it gives you an answer, that's NLP at work.

NLP isn't just about understanding words; it also involves understanding context and meaning. Computers use patterns in language to figure out what words mean in different situations. For instance, the word "bank" can mean a place to store money or the side of a river. NLP helps computers decide which meaning is correct based on the other words in the sentence.

Today, NLP is everywhere, from search engines like Google to voice assistants like Siri and Alexa. It is also used in chatbots, translation tools, and even autocorrect on your phone. This makes it one of the most important areas of technology, as it connects computers with the way humans communicate naturally.

📝 10.1.2

Which of the following describes Natural language processing?

- A technique to understand and work with human language.
- A method to teach computers to solve math problems.
- A system for creating images from text.
- A way to help humans learn programming languages.

📖 10.1.3

NLP has many practical applications that you might use every day without realizing it. One common example is **voice assistants** like Siri, Alexa, or Google Assistant. These systems use NLP to understand your spoken commands and respond with useful information or perform tasks like setting alarms or playing music.

Another important application is **translation tools**, such as Google Translate. These tools take text or speech in one language and convert it to another language almost instantly. NLP makes this possible by breaking sentences into smaller parts, understanding their meaning, and reassembling them in another language.

Chatbots are another popular example of NLP in action. They are used by companies to provide customer support, answer questions, or even make recommendations. Chatbots save time for businesses and make it easier for people to get quick answers to their questions.

📝 10.1.4

Which of the following are examples of NLP applications?

- Voice assistants like Siri and Alexa
- Chatbots for customer support
- Using GPS for navigation
- Writing computer code manually

🕮 10.1.5

In the early days of NLP, computers followed strict sets of rules to process language. This was called a **rule-based approach**, where programmers wrote instructions for every possible situation. However, this method was limited because language is full of exceptions and complex patterns that are hard to predict.

Today, NLP relies more on **machine learning (ML)** and **deep learning**. These methods allow computers to learn language patterns from large amounts of data, such as books, articles, or conversations. Instead of writing rules, programmers use algorithms to train computers to understand language on their own.

Deep learning, a type of machine learning, has made NLP even more powerful. It uses neural networks, which are inspired by the human brain, to analyze language. For example, tools like GPT and BERT use deep learning to understand and generate human-like text. This shift has made NLP more accurate and capable of handling complex tasks like translation and text generation.

📝 10.1.6

How has NLP evolved from rule-based systems to modern approaches?

- It now uses machine learning and deep learning instead of strict rules.
- It has replaced computers with human interpreters.
- It focuses only on spoken language instead of written text.
- It no longer requires any data to learn patterns.

🕮 10.1.7

For machines to understand people, they need to learn a lot about language. This includes not just knowing words, but also figuring out what those words mean in different situations. Machines must also learn how to respond in a way that makes sense to humans. To achieve this, we use three key concepts:

- Natural language understanding (NLU) helps a machine understand the meaning behind what someone says or writes. It’s not just about knowing words; it’s about understanding context. For example, if someone says, "I saw a bat", NLU can figure out whether they mean a baseball bat or a flying

animal. This involves identifying keywords, analyzing sentence structure, and determining the overall meaning.

- Natural language generation (NLG) is about making machines talk or write like humans. Once a machine understands what you mean, it needs to respond in a way that feels natural. For example, when you ask a chatbot for the weather, NLG helps it form a reply like, "It's sunny today". This involves choosing the right words, organizing them into sentences, and ensuring the response makes sense.
- Machines also need a source of information to answer questions. This is called a **knowledge base**. It's like a library of facts that the machine can search to find answers. For example, when you ask, "Who is the president of the United States?" the machine looks up the answer in its knowledge base and uses NLG to tell you.

📝 10.1.8

Which of the following are needed for machines to understand and respond to people?

- Natural language understanding
- Natural language generation
- Advanced gaming graphics
- Physical sensors for meassuring

🕮 10.1.9

Natural language understanding

NLU is like teaching a machine how to "read between the lines". When people talk or write, they don't always say things clearly or directly. NLU helps computers figure out the meaning behind words, even if the language is complicated or ambiguous.

For example, think about the sentence, "I'm feeling blue today". NLU would recognize that the speaker doesn't mean the color blue but is talking about feeling sad. To do this, the machine analyzes the context of the sentence and compares it to other examples it has learned.

NLU also deals with recognizing synonyms and related words. If someone says, "I need a cab", the machine understands they mean "taxi". This ability to connect similar words is important for understanding human language. Without NLU, machines would only match exact words and miss the meaning behind what we say.

📝 10.1.10

What is the main goal of Natural language understanding?

- To recognize and understand the meaning of human language.
- To create sentences for the computer to say.
- To build a library of physical objects.
- To measure the speed of language processing.

🕮 10.1.11

Natural language generation

NLG helps machines talk or write in a way that makes sense to humans. This is important because once a machine understands what you mean, it needs to explain its response clearly. If it can't do that, it won't feel natural or helpful.

For example, imagine asking a voice assistant, "What's the weather like?" NLG is the part that turns data (like temperature and forecast) into a sentence like, "It's 75 degrees and sunny today". Without NLG, the machine might just show raw numbers, which wouldn't be very useful for most people.

NLG also helps machines personalize their responses. If you're chatting with a customer service bot, it might say, "Hi Alex, how can I help you today?" Personal touches like this make interactions feel more human and engaging.

📝 10.1.12

What does Natural language generation allow machines to do?

- Communicate in a human-like way by generating sentences.
- Store large amounts of information.
- Understand the context of a sentence.
- Improve computer processing speed.

🕮 10.1.13

Knowledge base

Machines need a source of information to answer questions or provide useful responses. This source is called a **knowledge base**. A knowledge base is like a digital library where all kinds of facts and data are stored. The machine searches through this information whenever you ask a question.

For example, when you ask, "What is the capital of France?" the machine looks up "France" in its knowledge base and finds the answer: "Paris". The machine then uses NLG to share this information with you.

Knowledge bases are used in many applications. Search engines like Google, for instance, rely on enormous knowledge bases to find answers. They're also used in healthcare, where a medical chatbot might check its knowledge base to provide information about symptoms or treatments.

📝 10.1.14

What is the purpose of a knowledge base in NLP?

- To store and retrieve information for answering questions.
- To translate text into different languages.
- To analyze the structure of sentences.
- To teach computers how to recognize speech.

10.2 Communication

🕮 10.2.1

Chatbots and the evolution of conversational AI

The Turing test focuses on one specific human ability: the ability to communicate through written language. While this is just one aspect of human intelligence, it has driven the development of AI systems known as chatbots. Chatbots are AI programs designed to simulate conversations with humans. Over the years, chatbots have evolved significantly, from simple rule-based systems to the advanced conversational AI we see today.

The concept of chatbots isn't new. In the 1960s, Joseph Weizenbaum created **ELIZA**, one of the first chatbots. ELIZA used basic pattern-matching techniques to simulate conversations, such as pretending to be a therapist by repeating parts of the user's input as questions. Another early chatbot, **PARRY**, was created in the 1970s to mimic the speech patterns of someone with paranoid schizophrenia. While these systems were groundbreaking at the time, they were very limited and relied on pre-defined rules and scripts.

Modern chatbots like **GPT, Siri, Gemini, and Rabbit** are a major leap forward. Unlike earlier chatbots, these systems use advanced techniques like machine learning and natural language processing (NLP) to generate responses. They can understand context, adapt to different topics, and hold more natural conversations. This advancement has made chatbots not only more interactive but also useful in a variety of fields, from customer service to education and healthcare.

While chatbots have come a long way, they still simulate communication rather than genuinely understanding it. They represent how far AI has progressed in mimicking

human abilities, especially in language, but also remind us of the complexities of human intelligence.

📝 10.2.2

Which of the following are true about chatbots?

- ELIZA was one of the first chatbots, created in the 1960s.
- Modern chatbots like GPT and Siri rely on advanced machine learning techniques.
- Early chatbots were primarily rule-based, relying on pre-defined scripts.
- PARRY was designed to mimic a therapist.

🕮 10.2.3

Conversational software isn't a new idea! Even command-line apps allow users to type instructions in a structured language that's much closer to human language than the raw instructions understood by machines. Today, conversational AI has advanced to include chatbots and virtual assistants that can interact naturally with people. But where do we begin if we want to build our own?

In this lesson, you'll learn how to create a simple conversational application called an **AI-Bot**. The first AI-Bot you'll build is named **EchoBot**, and its job is straightforward - it echoes back to the user whatever they say. This basic bot helps us understand how AI interacts through a simple structure: receiving a message and responding.

For simplicity, the EchoBot you build will process messages written in Python code and display its responses on the screen. Here's an example of how EchoBot works:

```
AI-BOT: Hello
USER:   How are you?
AI-BOT: I can hear you! You said: How are you
```

This simple interaction shows the foundation of conversational AI: input, processing, and output. EchoBot doesn't understand the meaning of the input - it simply repeats it. As you progress, you'll learn how to add more complexity, such as understanding context, generating meaningful responses, and even handling multiple topics.

📝 10.2.4

Build your chatbot

To build an simple AI-Bot you need to create cycle WHILE, which shows text-input and takes a message

If the message is "bye", the conversation will be canceled. And the program will be canceled too.

Command **print**, in the last row, prints what the user just said.

```
conversation = True
print('AI-BOT: Hello!')

#cycle WHILE
while conversation:
    #read user input, and save it into the variable with the
name "message"
    message = input('USER:    ')

    #check if the user input is "bye"
    if message == "bye":
      #if yes, the conversation (and cycle "while") will be
canceled
        conversation = False
    #show message to user
    print('AI-BOT: I can hear you! You said: ', message)
```

10.2.5

What message did you have to write to cancel the conversation in the previous example?

- bye
- nothing
- I closed the internet browser
- See you

10.2.6

How we can improve previous example? The simplest thing we can do is use a python dictionary, with user messages as the keys and responses as the values. For example, here we define a dictionary called `responses`, with the messages "what's your name?" and "what's today's weather" as keys, and suitable responses as values.

```
#set of responses
responses = {
    "what is your name?": "my name is AI-BOT",
    "what is the weather today?": "it is sunny"
}

conversation = True
print('AI-BOT: Hello!')
```

```
while conversation:
    #read user input, and save it into the variable with the
name "message"
    message = input('USER:    ')

    #check if the user input is "bye" and cancel the
conversation
    if message == "bye":
        conversation = False

    #check if the message is in the set of responses
    if message in responses:
        #select the correct response and show it to the user
        print('AI-BOT: ',responses[message])
```

Notice that this will only work if the user's message *exactly* matches a key in the dictionary. In later chapters, you will build much more robust solutions.

What we can do if there isn't a matching message? We can improve our solution to this:

```
    #check if the message is in the set of responses
    if message in responses:
        #select the correct response and show it to the user
        print('AI-BOT: ',responses[message])
    else:
        if conversation:
            print('AI-BOT: I did not understand')
```

Full source code seems like this:

```
#set of responses
responses = {
    "what is your name?": "my name is AI-BOT",
    "what is the weather today?": "it is sunny"
}

conversation = True
print('AI-BOT: Hello!')
while conversation:
    #read user input, and save it into the variable with the
name "message"
    message = input('USER:    ')
```

```
    #check if the user input is "bye" and cancel the
conversation
    if message == "bye":
        conversation = False

    #check if the message is in the set of responses
    if message in responses:
        #select the correct response and show it to the user
        print('AI-BOT: ',responses[message])
    else:
        if conversation:
            print('AI-BOT: I did not understand')
```

10.2.7

Project: Customize your bot

Add reaction on the question "what is your favorite color?". The answer to this question will be "I like blue"

```
#set of responses
responses = {
    "what is your name?": "my name is AI-BOT",
    "what is the weather today?": "it is sunny"
}

conversation = True
print('AI-BOT: Hello!')
while conversation:
    #read user input, and save it into the variable with the
name "message"
    message = input('USER:    ')

    #check if the user input is "bye" and cancel the
conversation
    if message == "bye":
        conversation = False

    #check if the message is in the set of responses
    if message in responses:
        #select the correct response and show it to the user
        print('AI-BOT: ',responses[message])
    else:
```

```
        if conversation:
            print('AI-BOT: Tell me more....')
```

10.2.8

Randomization

A great way to keep users engaged is to offer them various answers.

Instead of using a bland default message like "I did not understand", you can use some phrases that invite further conversation. Questions are a great way to achieve this. "Why do you think that?", "How long have you felt this way?", and "Tell me more!" are appropriate responses to many different kinds of message, and even when they don't quite match are more entertaining than a boring fallback.

```
import random
#set of responses
no_responses = ["I'm sorry, I didn't understand you",
                "Why do you think that?",
                "How long have you felt this way?",
                "Tell me more!"
               ]
```

We can select an answer randomly from the set of answers with command:

```
print('AI-BOT: ', random.choice(no_responses))
```

Full code can see like this:

```
import random
#set of responses
no_responses = ["I'm sorry, I didn't understand you",
                "Why do you think that?",
                "How long have you felt this way?",
                "Tell me more!"
               ]

responses = {
    "what is your name?": "my name is AI-BOT",
    "what is the weather today?": "it is sunny"
}

conversation = True
```

```
print('AI-BOT: Hello!')
while conversation:
    #read user input, and save it into the variable with the
name "message"
    message = input('USER:    ')

    #check if the user input is "bye" and cancel the
conversation
    if message == "bye":
        conversation = False

    #check if the message is in the set of responses
    if message in responses:
        #select the correct response and show it to the user
        print('AI-BOT: ',responses[message])
    else:
        if conversation:
            print('AI-BOT: ', random.choice(no_responses))
```

10.2.9

We can improve our AI-Bot and add some variation.

We'll use the random module - specifically random.choice(ls) - which randomly selects an element from a list.

- If the message is in responses, we will use random.choice() in the respond() function to choose a random matching response.
- If the message is not in responses, we will choose a random default response.

```
responses = {
  "what's your name?": [
      "my name is AI-BOT",
      "they call me AI-BOT",
      "I go by AI-BOT"
   ],
  "what's today's weather?": [
      "the weather is sunny",
      "it's sunny today"
    ]
}
```

And, do not forget to change the message selection:

```
#check if the message is in the set of responses
    if message in responses:
        #select the correct response and show it to the user
        print('AI-BOT: ',random.choice(responses[message]))
    else:
        if conversation:
            print('AI-BOT: ', random.choice(no_responses))
```

Full source code seems like this:

```
import random
#set of responses
no_responses = ["I'm sorry, I didn't understand you",
                "Why do you think that?",
                "How long have you felt this way?",
                "Tell me more!"
               ]

responses = {
  "what's your name?": [
      "my name is AI-BOT",
      "they call me AI-BOT",
      "I go by AI-BOT"
   ],
  "what's today's weather?": [
      "the weather is sunny",
      "it's sunny today"
    ]
}

conversation = True
print('AI-BOT: Hello!')
while conversation:
    #read user input, and save it into the variable with the
name "message"
    message = input('USER:    ')

    #check if the user input is "bye" and cancel the
conversation
    if message == "bye":
        conversation = False

    #check if the message is in the set of responses
```

```
    if message in responses:
        #select the correct response and show it to the user
        print('AI-BOT: ',random.choice(responses[message]))
    else:
        if conversation:
            print('AI-BOT: ', random.choice(no_responses))
```

10.3 ELIZA

🕮 10.3.1

In the previous lesson, we explored the basics of building human-computer interactions. Now, let's dive into one of the first and most iconic systems in this field: **ELIZA**. Created in the mid-1960s by MIT researcher Joseph Weizenbaum, ELIZA was one of the earliest attempts to mimic human conversation using computerized natural language processing. It is considered a pioneer in AI and NLP, laying the groundwork for modern chatbots and conversational agents.

ELIZA worked by analyzing input from a user and responding based on simple pattern-matching rules. One of its most famous implementations was simulating a therapist, often responding to user statements with open-ended questions. For example, if a user said, "I feel sad", ELIZA might reply, "Why do you feel sad?" This approach made conversations feel more interactive and engaging, even though ELIZA didn't truly understand the input.

Despite being built on relatively simple code, ELIZA proved to be a compelling conversational partner for many users. Its ability to mimic human-like interaction was groundbreaking at the time, showing how even basic NLP techniques could simulate meaningful dialogue. ELIZA's success also highlighted the potential of AI to engage with humans naturally, inspiring decades of research in conversational AI.

```
Welcome to

                  EEEEEE  LL      III   ZZZZZZ   AAAAAA
                  EE      LL       II       ZZ   AA  AA
                  EEEEEE  LL       II     ZZZ    AAAAAA
                  EE      LL       II    ZZ      AA  AA
                  EEEEEE  LLLLLLL  III   ZZZZZZ  AA  AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in
1966.
This implementation by Norbert Landsteiner 2005.
```

```
ELIZA: Is something troubling you?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you
come here?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy?
YOU:
```

Source: Wikipedia

📝 10.3.2

What are some key facts about ELIZA?

- ELIZA was created by Joseph Weizenbaum in the 1960s.
- ELIZA simulated a therapist as one of its implementations.
- ELIZA relied on simple pattern-matching rules.
- ELIZA used advanced machine learning techniques.

🕮 10.3.3

How ELIZA simulated conversations

The ELIZA program worked by recognizing keywords and patterns in the text that the user typed and responding with predefined answers. This approach created the illusion of understanding, making users feel like they were having a real interaction, even though ELIZA didn't truly understand the conversation.

One of the most famous implementations of ELIZA was the "DOCTOR" script, which simulated a conversation with a psychotherapist. In this script, ELIZA often repeated or rephrased what the user said to keep the dialogue going. For example:

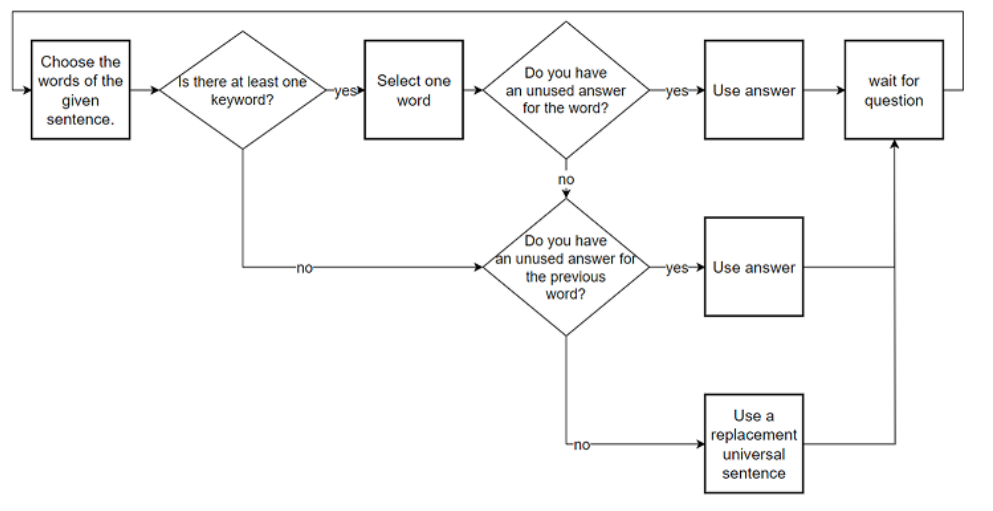
- If the user said, "I feel sad", ELIZA might respond with, "Why do you feel sad?"

This gave the impression of empathy, even though ELIZA was simply following programmed patterns.

ELIZA became notable for a phenomenon known as the **"ELIZA effect".** This refers to the tendency of people to attribute human-like understanding to machines, even when the AI is relatively simple. Users often felt that ELIZA truly understood them, which demonstrated how even basic conversational systems could evoke strong

emotional reactions. The ELIZA effect highlights how humans naturally seek connection, even with machines, and how easy it is to mistake simulated behavior for real understanding.

The ELIZA program worked based on this schema:



10.3.4

What was ELIZA's primary function?

- To mimic human conversation using natural language processing
- To solve complex mathematical equations
- To enhance graphics rendering in video games
- To manage databases efficiently

10.3.5

Which script is the most famous created for ELIZA and what did it simulate?

- "DOCTOR“, simulating a conversation with a psychotherapist
- "LAWYER“, simulating a legal consultation
- "ENGINEER“, simulating technical support

10.3.6

Why ELIZA simulated psychotherapist?

The psychotherapy model, specifically the non-directive style of a therapist, was well-suited to this method because it often involved reflecting the patient's statements back to them in a slightly rephrased or generalized form. A psychotherapist often encourages the patient to continue talking by reflecting their statements back to them. This technique doesn't require the therapist to provide new information but rather to mirror what the patient says.

Simulating a psychotherapist allowed ELIZA to engage in a conversation without needing to understand the content deeply. Instead of requiring complex reasoning or factual knowledge, ELIZA could respond effectively by rephrasing or asking open-ended questions that prompted further input from the user. This approach made it easier to create the illusion of an intelligent conversation without the need for complex language processing.

Joseph Weizenbaum, ELIZA's creator, discovered that users quickly formed **emotional connections** with the program, even when they knew it was just a computer. The role of a psychotherapist helped achieve this because it encouraged users to talk about personal thoughts and feelings. This emotional engagement made ELIZA feel more real and interactive, demonstrating how simple techniques can create the appearance of a thoughtful and empathetic conversation.

10.3.7

Why was the psychotherapy model ideal for ELIZA?

- It relied on reflecting and rephrasing user statements.
- It required deep understanding of user input.
- It needed extensive factual knowledge to respond.
- It involved providing detailed advice to users.

10.3.8

Project: ELIZA development

ELIZA used a method called "pattern matching" where the program searched for keywords in the users' sentences and then used predefined scripts to formulate answers. This process allowed ELIZA to guide the conversation without really understanding the content that was being presented.

To create a simple ELIZA application, it is necessary to make several changes in the previous solution. The main change will be in the database of answers, which will no longer be understood as answers to a specific question, but rather a reaction to some word appearing in the question.

For faster search of topics/intents for dialogue, we have also created a list of topics/intents in the variable **intent_dababase**.

```
responses = {
  "your name": [
      "my name is AI-BOT",
      "they call me AI-BOT",
      "I go by AI-BOT"
   ],
```

```
  "weather": [
      "the weather is sunny",
      "it's sunny today"
    ],
    "hockey": [
      "good winter game!",
      "we have to wait for the lake will be frozen"
    ],
    "football": [
      "millions of people follow their favorite team in every
game",
      "this game helps kids stay active",
      "it is also called as “Soccer” in North America",
      "ronaldo is the best",
    ]
}

intent_database = []
for response in responses:
    intent_database.append(response)

print("Intent database: ",intent_database)
```

Program output:

```
Intent database:  ['your name', 'weather', 'hockey',
'football']
```

Our AI-Bot will look for only those words in the questions to which it has a reaction in the "**responses**" database.

Therefore, first, we have to change all characters such as commas, full stops, question marks, and exclamation marks to spaces. We change it with function **sub()** from library r**e**. We then extract individual words from the question using the **.split()** function. Based on the space, this creates a list of words from the string.

We will show it in the next example:

```
import re

message = "Could we talk about hockey or weather?"
message = re.sub("[ ,.?!:;]", " ", message)
print('message = ',message)
words = message.split()
print('words = ', words)
```

Program output:

```
message =  Could we talk about hockey or weather
words =  ['Could', 'we', 'talk', 'about', 'hockey', 'or',
'weather']
```

In the next step, we have to select the word for conversation, as a intent-word. From the list of all words in the user question, we will find the word that is in the database of intent.

```
intent_word = ""
for word in words:
    if word in intent_database:
        intent_word = word
print("intent-word = ", intent_word)
```

Program output:

```
intent-word =  weather
```

You can see that our question has two words from the list of responses. We selected only the last one. We would like to show you the simplest solution, but you have many approaches to improve it.

The last change is to rewrite the key for finding answers in responses.

```
import random
print('AI-BOT: ', random.choice(responses[intent_word]))
```

Program output:

```
AI-BOT:  it's sunny today
```

10.3.9

Now, we can combine all previous approaches into a full, simple ELIZA application:

```
import random, re
#set of responses
no_responses = ["I'm sorry, I didn't understand you",
                "Why do you think that?",
                "How long have you felt this way?",
                "Tell me more!"
               ]
```

```
responses = {
  "your name": [
      "my name is AI-BOT",
      "they call me AI-BOT",
      "I go by AI-BOT"
   ],
  "weather": [
      "the weather is sunny",
      "it's sunny today"
    ],
    "hockey": [
      "good winter game!",
      "we have to wait for the lake will be frozen"
    ],
    "football": [
      "millions of people follow their favorite team in every
game",
      "this game helps kids stay active",
      "it is also called as “Soccer” in North America",
      "ronaldo is the best",
    ]
}


intent_database = []
for response in responses:
    intent_database.append(response)


conversation = True
print('AI-BOT: Hello! Is something troubling you?')
while conversation:
    #read user input, and save it into the variable with the
name "message"
    message = input('USER:    ')

    #check if the user input is "bye" and cancel the
conversation
    if message == "bye":
        conversation = False

    message = re.sub("[ ,.?!:;]", " ", message)
    words = message.split()
```

```
    intent_word = ""
    for word in words:
        if word in intent_database:
            intent_word = word

    #check if the message is in the set of responses
    if intent_word != "":
        #select the correct response and show it to the user
        print('AI-BOT:
',random.choice(responses[intent_word]))
    else:
        if conversation:
            print('AI-BOT: ', random.choice(no_responses))
```

You can see that to improve communication, you have to add new words and answers to the database of responses only.

However, ELIZA has its limitations. Because the program worked on the basis of simple keyword recognition and transformation, it lacked the ability for real comprehension or contextual analysis. ELIZA was unable to learn from previous interactions (to retain the context of the communication) or develop a conversation beyond its pre-programmed scripts.

📝 10.3.10

What was a major limitation of ELIZA?

- Its inability to retain context from previous interactions
- Its ability to display graphics during interactions
- Its capacity to operate without electricity
- Its use in non-English speaking countries

10.4 ELIZAs impact

🕮 10.4.1

ELIZA represented a significant milestone in the history of artificial intelligence and natural language processing (NLP). Its ability to simulate human conversation, albeit based on very limited algorithms, opened the door for further research in the field of human-computer interaction. ELIZA demonstrated the potential of computer programs for natural language processing, inspiring other researchers and developers to explore the possibilities of AI.

ELIZA has been an inspiration for many other researchers in the field of NLP. Its simple but effective approach to conversation simulation laid the groundwork for the development of more sophisticated systems that were better able to understand and respond to human language. ELIZA also helped raise public awareness of the potential of AI, leading to greater interest and investment in the field.

ELIZA is not just a historical note, but a key step on the path to advanced natural language processing and the development of artificial intelligence. Its legacy is still evident in current advances in NLP and AI, showing how early experiments can lead to large-scale and long-lasting impacts in technological fields.

The "ELIZA effect" describes the tendency of humans to attribute more intelligence to a machine than it actually has, based on the machine's ability to carry on a superficial conversation. This effect has had a significant impact on the public perception of AI, both in a positive and negative sense.

📝 10.4.2

Which of the following are true about ELIZA's impact on artificial intelligence and NLP?

- ELIZA demonstrated the potential of computer programs for natural language processing.
- ELIZA inspired further research and development in human-computer interaction.
- The "ELIZA effect" describes the tendency to overestimate a machine's intelligence.
- ELIZA used advanced machine learning algorithms to simulate conversation.

🕮 10.4.3

Since the creation of ELIZA, there have been dramatic advances in the technologies and methods used in **natural language processing (NLP)**, transforming how computers interpret and generate human language. While ELIZA relied on basic pattern recognition and substitution techniques to simulate conversation, modern NLP systems use advanced machine learning methods to provide much more accurate and relevant responses.

One major difference lies in the use of **deep neural networks** in modern systems like GPT-3. These models analyze language in a way that allows them to understand context, meaning, and subtle variations in communication. As a result, modern NLP systems can do much more than imitate human dialogue. They can perform complex tasks such as translating languages, automatically summarizing long texts, and answering detailed questions with a high degree of accuracy and relevance.

A key milestone in this evolution is the development of **generative pretrained transformers (GPTs)**. Models like GPT-3, which are based on this technology, are capable of generating coherent and contextually relevant text that often feels

indistinguishable from something written by a human. These systems have not only revolutionized conversational AI but have also opened up new possibilities for applications such as automatic content creation, translation, and personalized learning tools. This leap in NLP technology demonstrates how far we've come since ELIZA, and the potential for even greater advances in the future.

10.4.4

What is a key advancement of modern NLP systems like GPT-3 compared to ELIZA?

- They use deep neural networks to analyze and generate contextually relevant responses.
- They rely on basic pattern recognition techniques.
- They only imitate human dialogue without performing other tasks.
- They are limited to pre-programmed conversations.

10.4.5

What was one of the direct outcomes of ELIZA's approach to conversation simulation?

- It discouraged further research in AI due to its simplicity
- It discouraged further research in AI due to its simplicity. B) It led to the development of more sophisticated systems for understanding human language
- It proved that AI could not effectively process natural language
- It shifted the focus of AI research from language to robotics

10.4.6

What was a significant milestone achieved by ELIZA in the field of artificial intelligence?

- It was the first program to beat humans at chess
- It introduced the first use of graphics in user interfaces
- It demonstrated the potential of computer programs for natural language processing
- It established the first AI-based global network

10.4.7

What long-term impact did ELIZA have on the field of natural language processing and AI?

- It is considered a foundational step that continues to influence current advancements in AI and NLP
- Its techniques have become outdated and are no longer relevant
- It led to the discontinuation of research into AI for several years
- It redirected all AI research into purely theoretical studies

Modern NLP

Chapter 11

11.1 Text preprocessing

📖 11.1.1

Text preprocessing is the first and most important step in preparing text data for analysis. Computers don't understand language the way humans do - they see text as a collection of letters and symbols. Text preprocessing helps clean and organize the text so that computers can process it effectively.

Text data is often messy and inconsistent. For example, there might be extra spaces, capital letters, punctuation marks, or words that don't add much meaning, like "the" or "is". Text preprocessing removes unnecessary elements and simplifies the text so that only the most important parts remain. This makes it easier for machines to analyze and understand language.

Several common methods are used in text preprocessing:

- Tokenization breaks text into smaller parts like words or sentences.
- Removing stop words and punctuation taking out unimportant words and symbols.
- Lowercasing text ensures all text is in the same case for consistency.
- Stemming and kemmatization reduce words to their root or dictionary form.

Suppose we have the sentence, "The Cat is Running!" After preprocessing, it might look like this: ["cat", "run"]. The cleaned-up version is much easier for a machine to analyze.

📝 11.1.2

Why is text preprocessing important in NLP?

- It cleans and organizes text so computers can process it effectively.
- It allows computers to understand raw, messy text directly.
- It changes text into a numerical format automatically.
- It helps create graphics from text data.

📝 11.1.3

Tokenization

Tokenization is the process of breaking down a big piece of text into smaller parts called "tokens". These tokens can be words, sentences, or even individual characters. For example, if we take the sentence "I love learning about NLP", tokenization can split it into tokens like ["I", "love", "learning", "about", "NLP"].

Computers don't understand language like humans do. They see text as one big string of letters. Tokenization helps break text into manageable pieces so that a

machine can analyze it. By working with tokens, the computer can focus on the meaning of each part of the text.

In Python, you can use the split() method for simple tokenization or libraries like **NLTK** and **spaCy** for more advanced tokenization. For example:

```
text = "I love learning about NLP‟.
tokens = text.split()
print(tokens)  # Output: ['I', 'love', 'learning', 'about',
'NLP']
```

Program output:

```
['I', 'love', 'learning', 'about', 'NLP.']
```

- or using NLTK

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize("I love learning about NLP‟.)
print(tokens)
```

Program output:

```
['I', 'love', 'learning', 'about', 'NLP', '.']
```

11.1.4

What is the main purpose of tokenization in text preprocessing?

- To break text into smaller parts called tokens.
- To combine sentences into one word.
- To translate text into different languages.
- To measure the size of a text.

11.1.5

Removing stop words

Stop words are common words like "the", "is", "in", and "and" that don’t carry much meaning on their own. In most text analysis tasks, we remove these words because they don’t add valuable information.

When analyzing text, we want to focus on the important words that contribute to the meaning. Removing stop words makes the text smaller and easier to process without losing critical information. For example, in the sentence "The cat is sleeping on the bed". removing stop words leaves us with "cat sleeping bed".

We need a database of stop word, but python libraries like **NLTK** and **spaCy** can help:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "The cat is sleeping on the bed".
tokens = word_tokenize(text)
filtered_tokens = [word for word in tokens if word.lower() not
in stopwords.words('english')]
print(filtered_tokens)  # Output: ['cat', 'sleeping', 'bed']
```

Program output:

```
['cat', 'sleeping', 'bed', '.']
```

11.1.6

Why do we remove stop words in text preprocessing?

- They are common words that don't contribute much to the text's meaning.
- They are rare words that add extra meaning to sentences.
- They make the text harder to process.
- They are specific terms important for analysis.

11.1.7

Lowercasing text

Lowercasing means converting all letters in a text to lowercase. For example, "NLP is Fun" becomes "nlp is fun". This step ensures consistency when analyzing text because computers treat "NLP" and "nlp" as two different things.

Text analysis is case-sensitive by default. If we don't lowercase, the computer might count "Apple" and "apple" as two separate words, even though they mean the same thing. Lowercasing ensures that words are treated equally, making the analysis more accurate.

Lowercasing is simple in Python:

```
text = "NLP is Fun"
lowercased_text = text.lower()
print(lowercased_text)  # Output: 'nlp is fun'
```

Program output:

```
nlp is fun
```

11.1.8

What is the main benefit of converting text to lowercase during preprocessing?

- It ensures consistency by treating words like "Apple" and "apple" as the same.
- It reduces the size of the text.
- It adds capitalization to important words.
- It helps remove punctuation.

11.1.9

Stemming

Stemming is a process that reduces words to their root form. For example, words like "running", "runs", and "runner" are reduced to "run". It focuses on chopping off endings like "-ing" or "-ed" to get the base form of the word.

Stemming helps simplify text by grouping similar words together. This is especially useful when analyzing large amounts of text because it reduces redundancy and focuses on the core meaning of words.

In Python, you can use the **NLTK** library:

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()

words = ["running", "runner", "runs"]
stemmed_words = [ps.stem(word) for word in words]
print(stemmed_words)  # Output: ['run', 'runner', 'run']
```

Program output:

```
['run', 'runner', 'run']
```

11.1.10

What does stemming do in text preprocessing?

- It reduces words to their root form, like "running" to "run".
- It translates words into other languages.
- It removes punctuation from the text.
- It makes words longer for better analysis.

11.1.11

Lemmatization

Lemmatization is like stemming, but smarter. Instead of chopping off endings, it uses a dictionary to find the base or dictionary form of a word. For example, "better" becomes "good", and "running" becomes "run",

While stemming can sometimes produce meaningless roots (like turning "running" into "runn"), lemmatization ensures the output is an actual word. This makes it more accurate for tasks like text analysis or language modeling.

You can use **NLTK** or **spaCy** for lemmatization:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

words = ["running", "better", "cats"]
lemmatized_words = [lemmatizer.lemmatize(word) for word in
words]
print(lemmatized_words)  # Output: ['running', 'better',
'cat']
```

Program output:

```
['running', 'better', 'cat']
```

11.1.12

How is lemmatization different from stemming?

- Lemmatization uses a dictionary to find the proper base form of a word.
- Lemmatization produces meaningless word roots.
- Lemmatization is faster but less accurate.
- Lemmatization removes all stop words from the text.

11.2 Feature extraction

11.2.1

Feature extraction is the process of turning text into numbers so that computers can work with it. Computers don’t understand words like humans do - they need a way to represent text mathematically. Feature extraction helps convert text into numerical formats that can be analyzed by machine learning models.

Imagine you're training a computer to recognize spam emails. The computer needs to look at patterns in the words and phrases of emails to decide if they are spam. Feature extraction allows us to represent those words and phrases in numbers, so the computer can learn these patterns.

There are many ways to extract features from text. A simple method is to count how often each word appears in a document, while more advanced techniques look at the relationships between words. These methods create a numerical representation of the text that the computer can process.

11.2.2

What is the main purpose of feature extraction in text processing?

- To convert text into numerical formats so machines can process it.
- To count the number of letters in a text.
- To change text into different languages.
- To summarize long texts into shorter ones.

11.2.3

Bag-of-Words

Bag-of-Words is a simple method of feature extraction. It looks at all the words in a text and counts how many times each word appears. For example, if the sentence is "I like cats and I like dogs", Bag-of-Words would create a list like this:

- I: 2
- like: 2
- cats: 1
- and: 1
- dogs: 1

Bag-of-Words is useful because it creates a clear numerical representation of text. Even though it doesn't look at word order or meaning, it works well for tasks like spam detection or analyzing the popularity of certain words.

Here's an example using Python:

```
from sklearn.feature_extraction.text import CountVectorizer

text = ["I like cats and I like dogs‟.]
vectorizer = CountVectorizer()
features = vectorizer.fit_transform(text)
```

```
print(vectorizer.get_feature_names_out())  # Output: ['and',
'cats', 'dogs', 'i', 'like']
print(features.toarray())  # Output: [[1, 1, 1, 2, 2]]
```

Program output:

```
['and' 'cats' 'dogs' 'like']
[[1 1 1 2]]
```

11.2.4

What does the Bag-of-Words method do?

- Counts how often each word appears in a text.
- Analyzes the order of words in a sentence.
- Translates text into another language.
- Predicts the next word in a sentence.

11.2.5

TF-IDF

TF-IDF stands for Term Frequency-Inverse Document Frequency. It's a method that not only counts words but also looks at how important each word is in a document compared to a whole collection of documents. For example, common words like "the" or "and" appear frequently in all documents, so they get lower scores. Important words that are unique to a document get higher scores.

TF-IDF helps focus on meaningful words instead of common ones. For example, if you're analyzing movie reviews, TF-IDF will highlight unique words like "amazing" or "terrible" rather than generic words like "the".

Here's an example:

```
from sklearn.feature_extraction.text import TfidfVectorizer

texts = ["I like cats", "I like dogs"]
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(texts)

print(vectorizer.get_feature_names_out())  # Output: ['cats',
'dogs', 'like']
print(features.toarray())
```

Program output:

```
['cats' 'dogs' 'like']
[[0.81480247 0.         0.57973867]
 [0.         0.81480247 0.57973867]]
```

11.2.6

What does TF-IDF measure in text processing?

- How frequently a word appears and how unique it is.
- The total number of letters in a document.
- The translation of words into numbers.
- The length of sentences in a document.

11.2.7

Word embeddings

Word embeddings are advanced techniques for feature extraction that represent words as vectors in a mathematical space. Unlike Bag-of-Words or TF-IDF, embeddings capture the meaning of words and their relationships. For example, the words "king" and "queen" might be close together in the embedding space because they are related.

Word embeddings are powerful because they understand context. For instance, in the sentence "bank of the river", the word "bank" will have a different representation than in "bank account". This makes embeddings much better for tasks like machine translation and chatbots.

Popular libraries like **Word2Vec** or **GloVe** can generate word embeddings. Here's an example using a pre-trained model:

```
import spacy

# Load a pre-trained spaCy model
nlp = spacy.load("en_core_web_md")

# Process a sentence
doc = nlp("I like cats and dogs")

# Get the vector representation of a word
vector = nlp("cats").vector
print(vector)  # Output: A 300-dimensional vector representing
"cats"

# Compare similarity between words
similarity = nlp("cats").similarity(nlp("dogs"))
print(f"Similarity between 'cats' and 'dogs': {similarity}")
```

Program output:

```
[-0.61268   -0.064734  -0.56699   -0.39144   -0.41979
0.48678
 -0.31294    0.29624   -0.25772    2.2192    -0.33147
0.0039392
 -0.042447   0.37442   -0.18652    0.029715  -0.23983    1.416
  0.034493  -0.29434   -0.39891    0.4299    -0.099272
0.22192
 -0.30787    0.11238    0.40169   -0.20075   -0.043359  -
0.1548
 -0.61004   -0.53272   -0.025946   0.11539    0.49905
0.38587
  0.26251   -0.65631   -0.2507     0.13714   -0.20439   -
0.17916
 -0.071594   0.22207    0.036067  -0.60358   -0.61661
0.37028
  0.41655    0.1133    -0.42513   -0.1309    -0.10668   -
0.29307
 -0.060162   0.24548   -0.048741  -0.14913   -0.47725
0.71574
  0.27106   -0.034144   0.033213   0.15999   -0.50639   -
0.54717
  0.26809   -0.16382    0.35391   -0.45869   -0.15397   -
0.17281
  0.073191  -0.16188    0.54544   -0.14076    0.084263  -
0.14443
  0.63365   -0.6294     0.076101   0.13539   -0.31274   -
0.0053902
  0.044966  -0.83299    0.43895    0.48711    0.01288   -
0.14965
 -0.035441   0.39192    0.049251  -0.50649   -0.11186
0.25613
 -0.69311    0.040452   0.083796  -0.15211    0.15537
0.75328
 -0.077697   0.40442    0.41926   -1.2635    -0.36362   -
0.029733
 -0.19177   -0.0073138  0.57964    0.014645   0.17581   -
1.1712
  0.29575   -0.24117    0.32103    0.22099    0.22961
0.12315
 -0.4646    -0.33675    1.1535    -1.0596    -0.2786    -
0.43555
  0.046055   0.17818   -0.54411   -0.39807    0.13058   -
0.29633
```

```
 -0.11394    0.12298   -0.044114   0.23358   -0.25521
0.005093
  0.52585    0.036619  -1.7155    -0.26673   -0.090739
0.51008
  0.84245   -0.22871   -0.079482  -0.0029213 -0.35907   -
0.050142
 -0.076525   0.08398    0.075674  -0.27701   -0.085819
0.15038
 -0.03102    0.032134  -0.24743   -0.36245    0.63206   -
0.1389
  0.12904   -0.047932  -0.39091    0.57811   -0.061526  -
0.74497
  0.02629    0.011687   0.5746     0.6314     0.51488
0.16019
  0.382      0.71822   -0.054051   0.41295    0.39132
0.17327
  0.5453     0.30652    0.20403    0.23124   -0.19699   -
0.51232
  0.29342   -0.17117    0.48804   -0.55928   -0.064253
0.36009
 -0.37384   -0.29146   -0.22348    0.57183   -0.13826   -
0.23661
  0.51335   -0.40382   -0.40917    0.23571    0.092329
0.070955
  0.61405    0.44471    0.76898    0.54311    0.50808   -
0.3372
  0.29676    0.20609    0.24101   -0.23887   -0.36182   -
0.23638
  0.44887    0.16563    0.35734   -0.016492   0.74391
0.028051
 -0.19458   -0.57536   -0.2026     0.11047    0.049983
0.89323
 -0.19994    0.44014   -0.49289    0.018244  -0.14263   -
0.08619
  0.56226    0.44891   -0.19197    0.099574   0.18063   -
0.60151
 -0.83685   -0.020899  -0.089601   0.11795   -0.26522
0.29178
 -0.53655   -0.0075586 -0.68878   -0.032507  -0.057263
0.76552
 -0.42193    0.23629    0.5668     0.23398    0.0064155
0.44912
  0.67651   -0.25687    0.096718  -0.069202  -0.39039   -
0.041272
```

```
  0.023674    0.31629     0.1043    -0.27229     0.52119
0.32055
 -0.76582   -0.020925   -0.14284     0.22171    -0.1568
0.33978
 -0.36838    0.63171    -0.29093     0.18553     0.42925
0.25329
 -0.35271    -0.5044      0.3635     0.27235     0.077282   -
0.45261
 -0.48192    0.019771   -0.53193    -0.24074    -0.39771    -
0.20727
 -0.46444    0.77485     0.23406     0.18958    -0.29641    -
0.18323  ]
Similarity between 'cats' and 'dogs': 0.48349690447919175
```

11.2.8

What is the key advantage of word embeddings over simpler methods?

- They capture the meaning and relationships between words.
- They focus only on word counts.
- They ignore the context of words.
- They are only used for counting punctuation.

11.2.9

Feature extraction methods have different strengths and weaknesses:

- Bag-of-Words is simple and easy to use but ignores word order and context.
- TF-IDF adds importance to unique words but still doesn't consider word relationships.
- Word embeddings captures the meaning and context of words but requires more data and computation.

When to Use Each Method?

- Use Bag-of-Words for quick and simple tasks like spam detection.
- Use TF-IDF for tasks where unique words matter, like document classification.
- Use Word Embeddings for advanced tasks like chatbots or sentiment analysis.

Let's say you're analyzing these two sentences: "I like cats"., "I like dogs".

- Bag-of-Words will treat "cats" and "dogs" as unrelated.
- TF-IDF will highlight whichever word is rarer.
- Word embeddings will show that "cats" and "dogs" are semantically similar.

11.2.10

Which method captures the meaning and relationships between words?

- Word embeddings
- Bag-of-Words
- TF-IDF
- Character creating

11.3 Essencial tools

11.3.1

NER

Named Entity Recognition (NER) is a process in Natural Language Processing (NLP) that identifies and extracts specific types of information from text. These pieces of information, called **entities**, can include names of people, places, organizations, dates, or even monetary values. For example, in the sentence "Barack Obama was born in Hawaii on August 4, 1961", NER identifies:

- **Barack Obama**: Person
- **Hawaii**: Location
- **August 4, 1961**: Date

NER helps machines quickly pick out the most relevant pieces of information from text. This is useful in real-world applications like summarizing news articles, extracting dates from emails, or identifying company names in resumes. Without NER, a machine would treat all text as equally important, making it harder to focus on what really matters.

NER systems are built using machine learning models trained on labeled examples of entities. Libraries like **spaCy** or **NLTK** make it easy to implement NER in Python. For example:

```
import spacy

# Load a pre-trained NER model
nlp = spacy.load("en_core_web_sm")

# Process a sentence
doc = nlp("Barack Obama was born in Hawaii on August 4,
1961".)
```

```
# Extract named entities
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Program output:

```
Barack Obama PERSON
Hawaii GPE
August 4, 1961 DATE
```

📝 11.3.2

What is the main purpose of Named entity recognition?

- To identify key elements like names, dates, and locations in text.
- To create summaries of text.
- To generate new sentences from text.
- To convert text into numerical formats.

🕮 11.3.3

NER is widely used in real-world applications. Here are some examples:

1. Customer support to automatically extract customer names or product information from emails.
2. News summarization to Identify key people, places, and dates in articles.
3. Healthcare to extract drug names or symptoms from medical records.

These applications save time and reduce human effort. For example, instead of manually finding all dates in an email thread, an NER system can highlight them instantly.

Imagine a company that gets hundreds of resumes. Instead of reading each one manually, NER can scan for names, degrees, and skills, making the process faster and more efficient.

📝 11.3.4

Which of the following is a real-world application of NER?

- Extracting dates from emails
- Identifying key people in news articles
- Translating text into another language
- Counting the number of words in a document

11.3.5

How are NER systems built

NER systems rely on machine learning models trained on labeled data. The data includes examples of sentences where entities are marked, like:

```
"Barack Obama [PERSON] was born in Hawaii [LOCATION]".
```

Several libraries make it easy to build or use NER systems:

- spaCy - pre-trained models that are easy to use.
- NLTK - provides basic NER capabilities.
- Hugging face transformers - advanced models like BERT for better accuracy.

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Google was founded on September 4, 1998, in Menlo
Park".)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Program output:

```
Google ORG
September 4, 1998 DATE
Menlo Park GPE
```

11.3.6

What is needed to train an NER system?

- Sentences with entities labeled for training.
- Unlabeled text data.
- Large amounts of numerical data.
- Punctuation removal scripts.

11.3.7

Text similarity

Text similarity measures how close the meaning or structure of two pieces of text is. For example, the sentences "I enjoy reading books" and "I love reading books" are similar because they express a similar idea, even though the words are not identical.

Text similarity is used in many real-world applications, such as:

- Plagiarism detection - for identifying copied content by comparing texts.
- Search engines from finding documents that match a search query.
- Chatbots for understanding when a user's question is similar to a previously answered one.

One common way to measure similarity is by converting text into numerical representations (vectors) and comparing those vectors. Techniques like **Cosine similarity** calculate the angle between two vectors in a multi-dimensional space. Smaller angles mean the texts are more similar. We'll imagine these two sentences:

1. "I enjoy reading books".
2. "I love reading novels and books".

For simplicity, let's assume the vocabulary consists of these words: ["I", "reading", "books", "love", "novels", "enjoy"].

Sentence 1 word vector:

- "I" appears once → 1
- "reading" appears once → 1
- "books" appears once → 1
- "love" → 0
- "novels" → 0
- "enjoy" appears once → 1
- Final vector for Sentence 1: **[1, 1, 1, 0, 0, 1]**

Sentence 2 word vector:

- "I" appears once → 1
- "reading" appears once → 1
- "books" appears once → 1
- "love" appears once → 1
- "novels" appears once → 1
- "enjoy" → 0
- Final vector for Sentence 2: **[1, 1, 1, 1, 1, 0]**

Using these vectors:

1. Calculate the **Dot product**: Dot product = (1 * 1) + (1 * 1) + (1 * 1) + (0 * 1) + (0 * 1) + (1 * 0) = 3

2. Calculate the **Magnitude** of each vector:

- Magnitude of Sentence 1: sqrt(1^2 + 1^2 + 1^2 + 0^2 + 0^2 + 1^2) = sqrt(4) = 2
- Magnitude of Sentence 2: sqrt(1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2) = sqrt(5) =

3. Calculate **Cosine similarity**:

Cosine Similarity = Dot Product / (Magnitude of Sentence 1 * Magnitude of Sentence 2) = 3 / (2 * sqrt(5)) ≈0.67

Cosine similarity = 0.67 indicates that the two sentences are moderately similar. Both sentences share the words "I", "reading", and "books", but Sentence 2 introduces additional words ("love" and "novels") that lower the similarity score slightly.

11.3.8

What does Cosine similarity measure?

- The angle between the vectors of two pieces of text.
- The difference in sentence length.
- The total number of words in two sentences.
- The number of identical words.

11.3.9

Applications of text similarity

We know text similarity is applied in a wide range of tasks:

- Plagiarism detection - universities and organizations use it to find copied content in essays or research papers.
- Search engines when you type a query, the search engine finds documents that are similar to your query.
- Recommendation systems suggesting similar articles or videos based on what you've previously read or watched.

When you search for "best laptops for students", the engine compares your query with indexed documents using text similarity metrics. Documents with higher similarity scores are ranked higher.

How to calculate similarity in Python?

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

texts = ["I enjoy reading books", "I love reading books"]
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(texts)

similarity = cosine_similarity(vectors[0], vectors[1])
```

```
print(similarity[0][0])  # Output: Similarity score (closer to
1 means more similar)
```

Program output:

```
0.5031026124151314
```

📝 11.3.10

Which of the following is a real-world application of text similarity?

- Detecting plagiarism
- Identifying key entities in text
- Ranking search engine results
- Tokenizing words in a sentence

11.4 Text clustering

🕮 11.4.1

Text clustering

Text clustering is a method of grouping similar pieces of text into clusters. Each cluster represents a group of texts with similar content or themes. For example, news articles about sports might form one cluster, while articles about technology form another.

Clustering helps organize large amounts of text data. Applications include:

- News categorization automatically groups articles by topic.
- Customer feedback analysis groups similar complaints or reviews.
- Document organization clusters research papers by field.

Clustering uses mathematical algorithms to group data points (text vectors). One popular method is **K-Means clustering**, which divides data into k clusters.

📝 11.4.2

What is the purpose of text clustering?

- To group similar pieces of text into categories.
- To count the number of words in documents.
- To identify key names and dates in text.
- To translate text into different languages.

11.4.3

K-Means clustering

K-Means clustering is a popular algorithm that divides data into a specified number (k) of clusters. It assigns each piece of text to the cluster with the closest "center" based on similarity.

1. Choose k, the number of clusters.
2. Place k random points as initial cluster centers.
3. Assign each text to the nearest cluster.
4. Update cluster centers based on the average of the texts in the cluster.
5. Repeat until the cluster centers don't change much.

If you're clustering customer reviews into three categories:

- Cluster 1: positive reviews
- Cluster 2: negative reviews
- Cluster 3: neutral reviews

The algorithm calculates similarities between reviews and groups them accordingly.

Lets go to analyse 5 sentences

1. "I love cats".
2. "Dogs are great pets".
3. "I enjoy playing with cats and dogs".
4. "The weather is sunny".
5. "It's raining outside".

We will represent these sentences as **word counts** (simplified Bag-of-Words). Here's the feature matrix:

```
Sentence                                  Cats  Dogs  Weather
Sunny  Raining
 1. "I love cats".                         1     0      0
0      0
 2. "Dogs are great".                      0     1      0
0      0
 3. "I enjoy playing with cats and dogs".  1     1      0
0      0
 4. "The weather is sunny".                0     0      1
1      0
 5. "It's raining outside".                0     0      1
0      1
```

1. Initialize k clusters - let's set k = 2 (two clusters: one for pets, one for weather). We initialize the centroids randomly:

- Cluster 1 center: [1, 0, 0, 0, 0]
- Cluster 2 center: [0, 0, 1, 1, 0]

2. Assign points to clusters - we calculate the Euclidean distance of each sentence vector to the cluster centers.

- Euclidean distance = $\text{sqrt}(\sum(xi-ci)^2)$
- For Sentence 1 ("I love cats") distance to Cluster 1: $\text{sqrt}((1-1)^2 + (0-0)^2 + (0-0)^2 + (0-0)^2 + (0-0)^2) = 0$
- distance to Cluster 2: $\text{sqrt}((1-0)^2 + (0-0)^2 + (0-1)^2 + (0-1)^2 + (0-0)^2) = \text{sqrt}(3)$
- Sentence 1 goes to Cluster 1.

3. Repeat this process for all sentences. The cluster assignments are:

- Cluster 1: sentences 1, 2, 3
- Cluster 2: sentences 4, 5

4. Update cluster centers - calculate the new cluster centers as the mean of all points in each cluster:

- Cluster 1 center - average of [1, 0, 0, 0, 0], [0, 1, 0, 0, 0], and [1, 1, 0, 0, 0] = [(1+0+1) / 3, (0+1+1)/3, 0,0,0] = [0.67, 0.67, 0, 0, 0]
- Cluster 2 center - average of [0, 0, 1, 1, 0] and [0, 0, 1, 0, 1]: = [(0+0)/2,(0+0)/2,(1+1)/2,(1+0)/2,(0+1)/2] = [0, 0, 1, 0.5, 0.5]

5. Reassign points - recalculate distances and reassign points based on the updated centers. Repeat steps until cluster assignments no longer change.

Final output after convergence:

- Cluster 1 (Pets) - sentences 1, 2, 3
- Cluster 2 (Weather) - sentences 4, 5

11.4.4

What is the purpose of updating cluster centers in K-Means?

- To reflect the average position of points in the cluster.
- To make the clusters smaller.
- To ensure all clusters have the same number of points.
- To add new points to the dataset.

📖 11.4.5

Measuring clustering quality

Clustering quality is measured using metrics like **Silhouette score**. This score measures how similar texts in the same cluster are to each other compared to texts in other clusters. A higher Silhouette score (closer to 1) indicates better clustering.

Mathematical formula:

s(i) = (b(i) − a(i)) / max⁡(a(i),b(i))

Where:

- a(i) - average distance between iii and other points in the same cluster.
- b(i) - average distance between iii and points in the nearest other cluster.

After clustering articles, you should calculate the score to ensure similar topics are grouped together while different topics are separated.

📝 11.4.6

What does a high Silhouette score indicate?

- Better clustering quality.
- Poor clustering quality.
- More clusters than necessary.
- Random grouping of data points.

📝 11.4.7

Project: Clustering customer reviews into sentiment categories

Group customer reviews into three clusters: **positive**, **negative**, and **neutral**, using K-Means clustering. We'll process text data, represent it numerically, and apply the clustering algorithm to categorize reviews based on their sentiment.

1. Import necessary libraries - we'll use Python libraries for text preprocessing and clustering:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

2. Create a dataset of customer reviews:

```
reviews = [
    "I love this product! It's amazing‟.,
    "Terrible service. I will never come back‟.,
    "The quality is okay, but not the best‟.,
    "Absolutely fantastic experience!",
    "The item was broken when it arrived. Very
disappointing‟.,
    "Decent product for the price. Could be better‟.,
    "I am so happy with this purchase. Excellent quality!",
    "Worst customer support I’ve ever dealt with‟.,
    "It’s neither bad nor great. Just average‟.
]

# Create a DataFrame
df = pd.DataFrame(reviews, columns=["Review"])
```

3. Text preprocessing - convert text into numerical representations using **TF-IDF**:

```
# Convert text to TF-IDF vectors
vectorizer = TfidfVectorizer(stop_words="english")
tfidf_matrix = vectorizer.fit_transform(df["Review"])
```

4. Apply K-Means clustering - cluster the reviews into three groups:

```
# Define the number of clusters
num_clusters = 3

# Apply K-Means
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(tfidf_matrix)

# Add cluster labels to the DataFrame
df["Cluster"] = kmeans.labels_
```

5. Analyze results - interpret the clusters:

```
# Print reviews grouped by cluster
for cluster in range(num_clusters):
    print(f"Cluster {cluster}:")
    print(df[df["Cluster"] == cluster]["Review"].values)
```

```
    print("\n")
```

Program output:

```
Cluster 0:
['The quality is okay, but not the best.'
 'I am so happy with this purchase. Excellent quality!']


Cluster 1:
['Worst customer support I’ve ever dealt with.']


Cluster 2:
["I love this product! It's amazing‟.
 'Terrible service. I will never come back.'
 'Absolutely fantastic experience!'
 'The item was broken when it arrived. Very disappointing.'
 'Decent product for the price. Could be better.'
 'It’s neither bad nor great. Just average.']
```

6. Evaluate clustering quality - use the **Silhouette Score** to measure clustering quality:

```
score = silhouette_score(tfidf_matrix, kmeans.labels_)
print(f"Silhouette Score: {score}")
```

Program output:

```
Silhouette Score: 0.03186963047431916
```

A score closer to 1 indicates well-separated clusters - so this is bad solution.

We can try to use 4 clusters:

```
# Define the number of clusters
num_clusters = 4

# Apply K-Means
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(tfidf_matrix)

# Add cluster labels to the DataFrame
```

```
df["Cluster"] = kmeans.labels_

# Print reviews grouped by cluster
for cluster in range(num_clusters):
    print(f"Cluster {cluster}:")
    print(df[df["Cluster"] == cluster]["Review"].values)
    print("\n")

score = silhouette_score(tfidf_matrix, kmeans.labels_)
print(f"Silhouette Score: {score}")
```

Program output:

```
Cluster 0:
['The quality is okay, but not the best.'
 'I am so happy with this purchase. Excellent quality!']


Cluster 1:
['Worst customer support I’ve ever dealt with.']


Cluster 2:
["I love this product! It's amazing‟.
 'Terrible service. I will never come back.'
 'The item was broken when it arrived. Very disappointing.'
 'Decent product for the price. Could be better.'
 'It’s neither bad nor great. Just average.']


Cluster 3:
['Absolutely fantastic experience!']


Silhouette Score: 0.03319753174408247
```

The reason **K-Means clustering** works in this project, even though we didn’t explicitly define positive, negative, or neutral words, lies in the way the algorithm and **TF-IDF** representation work together.

TF-IDF converts text into numerical vectors that reflect how important each word is in a review relative to the dataset. Words commonly used in all reviews (like "the" or "is") are given low importance, while unique words (like "amazing" or "terrible") are given higher weights.

For example:

- In "I love this product! It's amazing", words like "love" and "amazing" have higher weights because they are less common and carry more meaning.
- In "Worst customer support I've ever dealt with", words like "worst" and "support" will be prominent in the vector.

This representation indirectly captures sentiment because positive, negative, and neutral reviews tend to use distinct sets of words.

K-Means clustering doesn't know about sentiment explicitly. Instead, it groups reviews based on the similarity of their TF-IDF vectors. Here's how:

- Similarity of vectors - reviews with similar words (like "love", "amazing", "happy") will have vectors that are closer in the mathematical space.
- Cluster formation - the algorithm assigns these similar vectors to the same cluster because they are closer to one another than to reviews with words like "worst" or "terrible".

Over time, clusters naturally form based on the dominant patterns in the data. Although we don't explicitly define positive or negative words, the algorithm discovers these patterns based on the language used. Sentiment categories emerge because:

- Positive reviews use common positive words ("amazing", "love", "fantastic").
- Negative reviews use common negative words ("worst", "terrible", "broken").
- Neutral reviews have balanced or vague language ("okay", "decent", "average").

The algorithm doesn't "understand" the meaning of the words but notices that reviews with similar words tend to group together.

We have bad results because we used only **a few sentences** - so the solution might not perform well. K-Means clustering - and most machine learning methods - becomes more effective with larger datasets. Let's explore why this is a limitation and how to improve the solution.

- Limited vocabulary - with only 10 sentences, the range of unique words (vocabulary) is very small. This limits the ability of TF-IDF to assign meaningful weights to words, as most words will appear in a significant portion of the dataset, reducing the importance of unique words. If "terrible" appears in 2 of 10 reviews, it's not as impactful as if it appeared in 2 of 1,000 reviews.
- Sparse data - TF-IDF vectors for small datasets are sparse, meaning many features (words) will have zero values in most vectors. This makes it harder for K-Means to find meaningful clusters.

- Insufficient patterns - clustering relies on patterns in the data. With only 10 sentences, there aren't enough examples of positive, negative, and neutral reviews to form distinct clusters.

📝 11.4.8

What is the purpose of using TF-IDF in this project?

- To calculate the word frequency and ignore irrelevant words.
- To directly classify the reviews into categories.
- To remove punctuation from the reviews.
- To visualize the clusters.

11.5 Text classification

🕮 11.5.1

Text classification is a process in Natural language processing where we assign predefined labels to text. It's like teaching a computer how to organize and categorize information. For example, we might classify emails as "spam" or "not spam" or group reviews into "positive", "negative", or "neutral".

Text classification is one of the most useful NLP tasks. It's used in:

- Email filtering to keep spam out of your inbox.
- Sentiment analysis to find out if reviews or social media posts are positive or negative.
- Topic categorization to group news articles by topic, like sports, politics, or technology.

Text classification uses machine learning to learn patterns in text. We train a model using labeled examples, such as:

- "This is the best product ever!" → Positive
- "Terrible experience. Do not recommend". → Negative

The model learns from these examples and can classify new, unseen text.

📝 11.5.2

What is the purpose of text classification?

- To assign labels to text based on its content.
- To translate text into numerical values.
- To find key entities like names and dates.

- To measure the similarity between two texts.

📖 11.5.3

Text classification involves three main steps:

1. **Preprocessing** cleaning and converting text into numerical data (e.g., TF-IDF or word embeddings).
2. **Training a model** using labeled data to teach the model to recognize patterns.
3. **Prediction** applying the trained model to new text.

Sentiment analysis - let’s classify these two reviews:

- "I love this phone. It works perfectly“. → Positive
- "The screen cracked after one day. Terrible quality!" → Negative

The model is trained to recognize that words like "love" and "perfectly" are positive, while "cracked" and "terrible" are negative. When given a new review, it predicts the sentiment based on these patterns.

Common models for text classification include:

- Naive Bayes are simple and fast, great for tasks like spam filtering.
- Logistic regression works well for small datasets.
- Deep learning models such as BERT or GPT for advanced tasks.

📝 11.5.4

What are the main steps in text classification?

- Preprocessing text
- Predicting labels for new text
- Generating new text
- Measuring cosine similarity

📝 11.5.5

Project: Build a spam detector

Let’s build a simple project to classify emails as "spam" or "not spam“.

1. Import libraries

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,
classification_report
```

2. Prepare the Dataset

```
data = {
    "Email": [
        "Win a $1,000 gift card now!",
        "Important meeting scheduled at 10 AM",
        "Get cheap medicines online",
        "Your invoice is attached",
        "Earn money working from home!"
    ],
    "Label": ["spam", "not spam", "spam", "not spam", "spam"]
}
df = pd.DataFrame(data)
```

3. Preprocess the data - convert the text into numerical form using TF-IDF:

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df["Email"])
y = df["Label"]
```

4. Train the model - split the data and train a Naive Bayes classifier:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
model = MultinomialNB()
model.fit(X_train, y_train)
```

5. Make predictions - evaluate the model's performance:

```
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Program output:

```
Accuracy: 0.5
Classification Report:
              precision    recall  f1-score   support
```

```
    not spam       0.00      0.00      0.00         1
        spam       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

/home/johny/.local/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1334:
UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/johny/.local/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1334:
UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/johny/.local/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1334:
UndefinedMetricWarning: Precision and F-score are ill-defined
and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

11.5.6

Which machine learning model is commonly used for spam detection?

- Naive Bayes
- K-Means
- Cosine similarity
- PCA

11.5.7

Multiclass classification

Multiclass classification is a type of text classification where text is grouped into more than two categories. For example, we might classify customer reviews into:

- **Positive**
- **Negative**
- **Neutral**

Topic categorization based e.g. on these news headlines:

- "The team won the championship!" → Sports
- "The stock market crashed today". → Business
- "A new vaccine is under development". → Health

The model learns from labeled examples and assigns the correct topic to new headlines.

11.5.8

What is the key feature of multiclass classification?

- It assigns text to more than two categories.
- It only handles binary classification tasks.
- It uses clustering algorithms.
- It only works with numerical data.

11.5.9

Project: Topic categorization of news headlines

Build a simple text classification system to categorize news headlines into topics: Sports, Business, and Health.

1. Import necessary libraries - we'll use Python libraries for text preprocessing and classification and prepare some dataset:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,
accuracy_score

data = {
    "Headline": [
        "The team won the championship!",
        "The stock market crashed today".,
        "A new vaccine is under development".,
        "The player scored a hat trick in the game".,
        "Shares of the company dropped significantly".,
        "Scientists discover a breakthrough treatment for
cancer".,
        "The football league kicks off this weekend".,
```

```
        "Investors are worried about inflation‟.,
        "New health guidelines released for public safety‟.
    ],
    "Category": ["Sports", "Business", "Health", "Sports",
"Business", "Health", "Sports", "Business", "Health"]
}

df = pd.DataFrame(data)
```

2. Preprocess the data - convert the text into numerical representations using TF-IDF:

```
# Convert text to TF-IDF vectors
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df["Headline"])
y = df["Category"]
```

3. Split the data and train the model

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

4. Make predictions - evaluate the model on the test set:

```
# Make predictions
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

5. Test with new headlines - test the classifier with new, unseen headlines:

```
new_headlines = [
    "The tennis player broke the world record‟.,
    "Economic growth slows down this quarter‟.,
    "Doctors warn of a new flu outbreak‟.
```

```
]

# Transform the new data
new_vectors = vectorizer.transform(new_headlines)

# Predict categories
predictions = model.predict(new_vectors)
for headline, category in zip(new_headlines, predictions):
    print(f"Headline: '{headline}' → Predicted Category:
{category}")
```

Program output:

```
Headline: 'The tennis player broke the world record.' →
Predicted Category: Sports
Headline: 'Economic growth slows down this quarter.' →
Predicted Category: Sports
Headline: 'Doctors warn of a new flu outbreak.' → Predicted
Category: Health
```

11.5.10

Which of the following was the goal of the previous project?

- To classify news headlines into predefined topics like Sports, Business, and Health.
- To generate new headlines.
- To cluster similar headlines without labels.
- To predict future headlines.

Generative AI

Chapter 12

12.1 Introduction

📖 12.1.1

Generative AI is a type of artificial intelligence that creates something new - like text, images, music, or even video. Generative AI is a field of artificial intelligence that focuses on creating new content. Unlike tasks like text classification or clustering, where AI organizes existing data, generative AI produces original outputs based on patterns it has learned. For example:

- A chatbot generating a response to your question.
- AI creating a painting based on your description.

Natural language understanding (NLU) helps AI understand text or speech. Natural language generation (NLG), a key part of generative AI, allows it to respond. NLU understands, "What's the weather today?" while NLG replies, "It's sunny and 75 degrees".

Generative AI powers tools like:

- Chatbots (e.g., ChatGPT)
- Automatic content creators
- AI artists and music composers

📝 12.1.2

What is the main purpose of generative AI?

- To create new content like text, images, or music.
- To organize and classify data.
- To cluster similar pieces of data.
- To translate text into different languages.

📖 12.1.3

AI generates text by predicting the next word (or token) in a sequence based on patterns it has learned from data. Let's break this down with two types of tasks:

1. Filling text with generation - AI continues a given piece of text naturally. It doesn't "understand" but uses patterns it has seen during training to create coherent text. Example:

- Input: "Once upon a time, there was a brave knight who..".
- Output: "...traveled through the forest in search of a lost treasure".

Here, the AI looks at the words "brave knight" and "forest" to predict what might come next based on similar stories in its training data. It's like finishing someone's sentence when you know the context.

2. Answering questions - when answering a question, the AI uses text generation in a more focused way. Instead of continuing a story, it generates text that directly relates to the question. **Example:**

- Question: "What is the capital of France?"
- Output: "The capital of France is Paris".

The AI processes the question, identifies key elements ("capital of France"), and generates a fact-based response. This involves understanding relationships between words and meanings.

Filling text focuses on generating natural and coherent text without needing specific knowledge. Answering Questions requires the AI to retrieve or infer facts while keeping the response relevant to the question.

Let's compare both tasks:

- Filling text: input: "The cat chased the mouse and then..".; output: "...jumped onto the table to catch it".
- Answering questions: input: "Why did the cat chase the mouse?"; output: "Because the cat wanted to catch the mouse".

In both cases, the AI predicts the next words, but answering questions requires more focus on meaning and facts.

📝 12.1.4

What does AI do when generating text?

- It predicts the next word in a sequence based on learned patterns.
- It directly copies sentences from its training data.
- It identifies key entities like names and dates.
- It clusters text into predefined categories.

🕮 12.1.5

Transformer

A transformer is a type of AI model architecture used for tasks like text generation and answering questions. It is one of the key reasons modern AI, like ChatGPT, is so powerful.

Why it's called a transformer? The model "transforms" input data (e.g., a question or sentence) into something meaningful, like a reply or a continuation of the text.

Transformers process all words (or tokens) in a sentence at once. They use a method called attention to focus on the most important parts of the input. For example, in the question "What is the capital of France?" the model pays more attention to "capital" and "France" while ignoring less relevant words like "What" or "is".

Older models processed words one at a time, which made them slower and less accurate. Transformers process all words simultaneously, making them faster and better at understanding context. This is why they power tools like ChatGPT, BERT, and GPT.

📝 12.1.6

What is a transformer in AI?

- A model architecture that processes all words at once and focuses on key parts using attention.
- A model that predicts the weather.
- A model that clusters text into groups.
- A method for removing unnecessary words from text.

🕮 12.1.7

Text generation is one of the most exciting applications of generative AI. It powers many tools and tasks that are used daily, from chatbots to creative writing. Let's explore these applications in detail,

1. Chatbots and virtual assistants

Generative AI enables chatbots like ChatGPT, Alexa, and Siri to generate human-like responses to questions or commands. When you interact with these systems, they rely on Natural Language Generation (NLG) to craft replies.

Example:

- User: "What's the weather today?"
- Chatbot: "It's sunny with a high of 75°F".

The AI:

1. **Understands** your query (NLU).
2. **Generates** an answer using text generation.

Chatbots save time and effort, providing instant responses for tasks like checking the weather, setting reminders, or answering FAQs.

2. Creative writing

AI can generate stories, poems, or even jokes. Writers use AI tools to brainstorm ideas or create drafts for creative projects. The AI draws on patterns from its training to produce imaginative outputs.

- Example: "Write a poem about the moon“.
- Output:

```
"The moonlight dances on the sea,
A silver glow so wild and free,
Guiding ships across the night,
A beacon of hope, soft and bright".
```

It helps writers overcome creative blocks and inspires new ideas.

3. Summarization

Text generation is used to create concise summaries of long documents or articles. This is especially useful in business, education, or news platforms where people need quick overviews.

- Example: A long article about climate change.
- Output: "Climate change is accelerating due to greenhouse gas emissions, leading to rising temperatures and extreme weather events“.

Summarization saves time by providing key points without reading the full text.

4. Translation

AI-powered translation tools like Google Translate use text generation to convert text from one language to another. For instance:

- Input: "Hello, how are you?" (English)
- Output: "Hola, ¿cómo estás?" (Spanish)

The model learns language patterns and generates grammatically correct translations. Translation tools break language barriers, enabling global communication.

5. Content generation

Businesses use generative AI to create marketing materials, social media posts, or product descriptions. The AI can generate text tailored to specific audiences.

- Example: "Write a product description for a smartwatch“.
- Output: "Stay connected and track your fitness with our latest smartwatch. Featuring heart rate monitoring, GPS, and customizable watch faces“.

It speeds up the content creation process, allowing marketers to focus on strategy.

6. Personalized recommendations

Text generation also powers recommendation systems by creating personalized suggestions for users. For example:

- Netflix might generate "Because you liked ‘Stranger Things,’ we recommend ‘Dark.’"
- E-commerce platforms might generate, "Based on your interest in sneakers, check out these trending styles“.

It enhances user experience by providing relevant suggestions.

In each of these applications, the AI follows similar steps:

1. Input - the user provides a prompt or query.
2. Processing - the model analyzes the input using techniques like transformers and NLU.
3. Output - the AI generates a coherent and contextually relevant response.

For instance:

- In chatbots, the focus is on meaningful replies.
- In summarization, the focus is on extracting key ideas.
- In creative writing, the focus is on imagination and diversity.

12.1.8

Which of the following are real-world applications of text generation?

- Summarizing long articles
- Translating text into another language
- Clustering text into groups
- Measuring sentence similarity

12.2 Text generation

12.2.1

Chatbots and virtual assistants

Chatbots and virtual assistants generate responses to user inputs, enabling conversational interactions. Examples include Alexa, Siri, and customer support bots. They use NLG to respond like humans.

Example:

- User: "How are you?"
- Chatbot: "I am fine“.

The chatbot:

1. Understands the question using NLU.
2. Generates a relevant response using NLG.

Chatbots are widely used in:

- Customer service - answering FAQs.
- Productivity - setting reminders or appointments.

Example:

```
from transformers import AutoModelForCausalLM, AutoTokenizer

# Load GPT-2  model and tokenizer
model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

def generate_response(prompt: str, max_length: int = 50,
temperature: float = 0.7, top_k: int = 50, top_p: float = 0.9)
-> str:
    """
    Generate a single response using GPT-2 Medium.
    """
    inputs = tokenizer(prompt, return_tensors="pt")
    outputs = model.generate(
        inputs.input_ids,
        max_length=max_length,
        pad_token_id=tokenizer.eos_token_id,
```

```
        temperature=temperature,
        top_k=top_k,
        top_p=top_p,
    )
    response = tokenizer.decode(outputs[0],
skip_special_tokens=True).strip()
    return response

# Example usage
prompt = "Hello, how are you?"
response = generate_response(prompt)
print(f"GPT-2 Response: {response}")
```

Program output:

```
GPT-2 Medium Response: Hello, how are you?

I'm a little bit of a nerd. I'm a big nerd. I'm a big nerd.
I'm a big nerd. I'm a big nerd. I'm a big nerd. I'm a
```

The repetitive and nonsensical output you received from GPT-2 likely results from mode collapse or poor context retention during generation. Let's break down what's happening and how to fix it.

Why this happens?

- GPT-2 generates text by predicting the next word in a sequence. Without strong context or constraints, it may fall into repetitive loops, especially when working with short inputs or no clear conversational structure.
- If the **temperature** (randomness control) is too low or **top_k** and **top_p** settings are not optimal, GPT-2 might keep repeating the same predictions because it "thinks" the repeated phrase is the most likely continuation.
- GPT-2 doesn't inherently understand dialogue flow. It works better with structured or descriptive prompts, and short user inputs like "What's your favorite color?" don't provide enough context to generate meaningful responses.

How to fix it?

- Adjust temperature and sampling settings - increase randomness to break repetitive loops. For example set temperature=0.9 to encourage creativity, use top_p=0.95 (nucleus sampling) instead of relying solely on top_k.
- Provide clearer prompts - GPT-2 performs better with descriptive and structured inputs. For example: Instead of: "What's your favorite color?", use: "Pretend you are an AI assistant. What's your favorite color, and why?"
- Trim repetition in output - use post-processing to identify and trim repeated sequences.

12.2.2

What is the main purpose of chatbots?

- To generate human-like responses to user inputs.
- To translate text into another language.
- To summarize long documents.
- To cluster text into categories.

12.2.3

Creative writing

Creative writing by AI refers to using artificial intelligence to generate text that is imaginative or artistic. This includes stories, poems, dialogues, and even jokes. Unlike structured tasks like summarization or translation, creative writing focuses on diversity and creativity. The AI uses patterns learned during training to craft unique outputs.

AI generates creative text by predicting the next word (or sequence of words) based on the input prompt. It doesn't "think" like humans but uses probabilities to choose words that fit well together. For instance:

- "Write a story about a brave knight who…"
- Output: "…traveled through enchanted forests to rescue a lost princess trapped in a castle guarded by a dragon“.

The AI has seen many similar patterns during training and combines these to produce something new. Applications of AI in creative writing ca be:

- Storytelling - writers can use AI to generate ideas for novels, short stories, or screenplays. Example: Starting a sci-fi story with the prompt, "In a future where robots rule the world…"
- Poetry - AI can write rhymes or free verse poetry based on a theme or keyword. Example: "The stars above, a silent glow, whisper secrets we’ll never know“.
- Game Development - AI assists in generating dialogues for characters or descriptions of fantasy worlds. Example: "The village of Eldara lies in a valley surrounded by mist-covered mountains“.
- Humor and jokes - AI creates jokes or witty comments to entertain users. Example: "Why don’t scientists trust atoms? Because they make up everything!"

We can use the **GPT-2** model, which is open-source and requires no API key, to generate creative text.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

# Load GPT-2  model and tokenizer
model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

def generate_response(prompt: str, max_length: int = 50,
temperature: float = 0.7, top_k: int = 50, top_p: float = 0.9)
-> str:
    """
    Generate a single response using GPT-2 Medium.
    """
    inputs = tokenizer(prompt, return_tensors="pt")
    outputs = model.generate(
        inputs.input_ids,
        max_length=max_length,
        pad_token_id=tokenizer.eos_token_id,
        temperature=temperature,
        top_k=top_k,
        top_p=top_p,
    )
    response = tokenizer.decode(outputs[0],
skip_special_tokens=True).strip()
    return response

# Example usage
prompt = "Write a story about a brave knight who"
response = generate_response(prompt, max_length=100)
print(response)
```

Program output:

```
Write a story about a brave knight who saved his family from a
terrible fate.

The story is about a young knight who is a knight in the army
of the king of the land. He is a knight in the army of the
king of the land. He is a knight in the army of the king of
the land. He is a knight in the army of the king of the land.
He is a knight in the army of the king of the land. He is a
knight in
```

How the code works:

- Input prompt - the user provides a starting point for the AI, such as "Write a story about a brave knight who"
- Tokenization - the input is converted into numerical data that the model can process.
- Text generation - the AI predicts the next word repeatedly, choosing words based on probabilities and creative constraints like temperature and top_p.
- Decoding - the numerical output is translated back into human-readable text.

12.2.4

Which of these is an application of creative text generation?

- Creating story drafts
- Translating languages
- Generating customer support responses
- Summarizing research papers

12.2.5

What is the primary focus of creative writing by AI?

- To create imaginative and artistic outputs like stories or poems.
- To summarize long texts.
- To translate text into another language.
- To classify text into categories.

12.2.6

Summarization

Summarization involves reducing a long piece of text to its most important points while retaining the meaning. AI can summarize text in two ways:

1. Extractive summarization selects key sentences or phrases directly from the original text. The AI identifies and selects the most important sentences from the text.

- Example: Given the text: "Climate change is caused by greenhouse gases. These gases trap heat, leading to global warming and extreme weather events“.
- Extractive summary: "Climate change is caused by greenhouse gases“.

2. Abstractive summarization generates a summary in its own words, rephrasing and condensing the information. The AI generates a new, concise version of the text using its language generation capabilities.

- Example: "Climate change is caused by greenhouse gases. These gases trap heat, leading to global warming and extreme weather events".
- Abstractive summary: "Greenhouse gases cause climate change, resulting in global warming and extreme weather".

Applications of summarization

- News summaries quickly condensing news articles for busy readers. Example: Summarizing a 1,000-word article about a political event into a single paragraph.
- Education - creating concise summaries of textbooks or research papers. Example: Summarizing a biology chapter on photosynthesis into key takeaways.
- Business summarizing meeting transcripts or lengthy reports for executives. Example: "The key action items from the meeting are X, Y, and Z".
- Social media condensing long posts into short, shareable snippets.

We'll use a **pre-trained summarization model** like BART (from Hugging Face) or for quicker result gpt2.

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

# Load a pre-trained summarization model and tokenizer
model_name = "facebook/bart-large-cnn"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

# Function to summarize text
def summarize_text(text, max_length=50, min_length=25):
    # Tokenize the input text
    inputs = tokenizer(text, return_tensors="pt",
truncation=True)

    # Generate a summary
    outputs = model.generate(
        inputs.input_ids,
        max_length=max_length,
        min_length=min_length,
        length_penalty=2.0,  # Penalize overly long summaries
        num_beams=4,  # Use beam search for better results
        early_stopping=True  # Stop when the model finds the
best solution
```

```
    )

    # Decode and return the summary
    return tokenizer.decode(outputs[0],
skip_special_tokens=True)

# Example long text
text = """
Climate change is a pressing global issue caused by the
buildup of greenhouse gases in the atmosphere.
These gases trap heat, leading to global warming and extreme
weather events. Rising sea levels,
melting glaciers, and more intense storms are just a few
consequences of this crisis.
"""

# Generate a summary
summary = summarize_text(text)
print("Summary:", summary)
```

Program output:

```
Summary: Climate change is a pressing global issue caused by
the buildup of greenhouse gases in the atmosphere. Rising sea
levels,melting glaciers, and more intense storms are just a
few consequences.
```

AI Summarization summarizes large texts in seconds. Extracts or generates the most relevant information. Adjust summary length and level of detail.

12.2.7

What is the purpose of text summarization?

- To highlight key points from a long text.
- To generate creative stories.
- To categorize documents.
- To answer user queries.

12.2.8

What is the key difference between extractive and abstractive summarization?

- Extractive summarization selects key sentences from the original text, while abstractive summarization generates new sentences.
- Extractive summarization translates text into another language.

- Abstractive summarization is always shorter than extractive.
- Both methods work only on short texts.

12.2.9

Translation

AI-powered translation converts text from one language to another while maintaining its meaning. Unlike older rule-based systems, AI models today, such as **transformers**, understand context and idiomatic expressions, making translations more natural and accurate.

Modern AI translation relies on pre-trained language models like BERT, MarianMT, and M2M100. These models learn from vast datasets of bilingual texts and use probabilities to predict the best translation for a given sentence. Following steps are applied:

- Input text where user provides a sentence in one language (e.g., English).
- Understanding context - the model analyzes the structure and meaning of the input text using attention mechanisms.
- Generating translation predicts the equivalent text in the target language, word by word or token by token.
- Example: "Hello, how are you?" (English), output: "Hola, ¿cómo estás?" (Spanish)

Applications of AI translation are:

- Businesses use AI translators to communicate with global audiences.
- Translation apps help tourists navigate foreign countries.
- Translate educational content into different languages to make it accessible to more people.
- Media and entertainment e.g. by subtitling movies or translating books and articles.

We'll use the **MarianMT model**, an open-source translation model, for this example.

```
from transformers import MarianMTModel, MarianTokenizer

# Load the pre-trained MarianMT model for English to Spanish translation
model_name = "Helsinki-NLP/opus-mt-en-es"
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# Function to translate text
def translate_text(text, src_lang="en", tgt_lang="es"):
```

```
    # Tokenize the input text
    inputs = tokenizer(text, return_tensors="pt",
truncation=True)

    # Generate the translation
    outputs = model.generate(
        inputs.input_ids,
        max_length=50,  # Limit translation length
        num_beams=4,  # Use beam search for better accuracy
        early_stopping=True
    )

    # Decode and return the translation
    return tokenizer.decode(outputs[0],
skip_special_tokens=True)

# Example English text
text = "Hello, how are you?"

# Translate from English to Spanish
translation = translate_text(text)
print("Translation:", translation)
```

Program output:

```
Translation: Hola, ¿cómo estás?
```

The code follows typical steps:

- Tokenization - converts the input sentence into tokens that the AI model understands.
- Translation - generates the equivalent sentence in the target language.
- Decoding - converts the tokenized output back into a human-readable sentence.

Advantages of AI translation are:

- Speed - translate text in seconds compared to manual translation.
- Context awareness - Handles idiomatic phrases and context better than traditional methods. Example: "Break a leg!" is translated contextually to a phrase meaning "Good luck“.
- Scalability - translate large documents or entire websites quickly.

12.2.10

What makes AI translation more effective than traditional methods?

- AI translation learns from large datasets and handles context and idiomatic expressions.
- AI translation uses fixed rules for sentence structure.
- AI translation can only translate short sentences.
- AI translation works without any training data.

12.3 Image generation

12.3.1

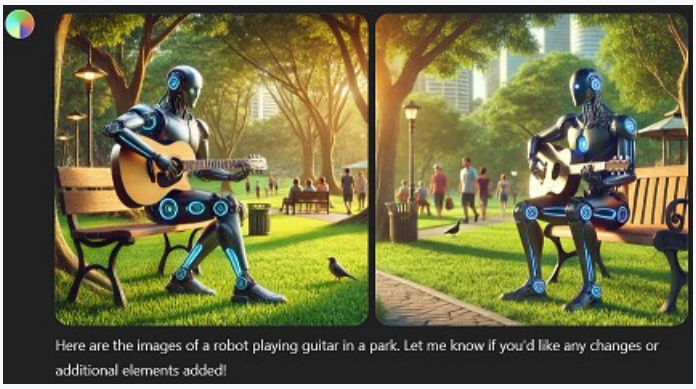
Image generation is a type of generative AI that creates pictures from scratch. It can turn descriptions into images or even modify existing pictures. For example:

- Input: "A cat sitting on a beach at sunset“.
- Output: A completely new image of a cat on a beach.

DALL-E results

AI models like **DALL·E** or **Stable diffusion** learn from millions of images and captions. They understand the relationship between words and visual elements to generate realistic or artistic images based on a description.

- Input: "A robot playing guitar in a park“.
- Output: The AI generates an image of a robot holding a guitar with trees and grass in the background.



DALL-E results

12.3.2

What does image generation do?

- It creates new images based on descriptions or patterns.
- It edits existing photos.
- It classifies images into categories.
- It analyzes text for sentiment.

12.3.3

Where is image generation used?

- Art and design creates custom artwork or logos.

- Gaming designs characters, backgrounds, or assets.
- Marketing generates images for ads or campaigns.
- Education visualizes concepts, like historical events or scientific models.

An artist describes their idea: "A futuristic city at night with flying cars“. The AI generates several images that fit the description, helping the artist refine their vision.

Image generation helps people bring their ideas to life, even if they can’t draw. It’s also faster and more cost-effective for tasks like prototyping and storytelling.

12.3.4

Which of the following is an application of image generation?

- Creating custom artwork for marketing campaigns.
- Translating languages into text.
- Generating music based on input prompts.
- Classifying documents into predefined topics.

12.3.5

How image generation works

Image generation (IG) is a process where AI creates visual content, such as drawings, photos, or digital art, based on a description or a set of instructions. It leverages powerful machine learning models trained on massive datasets of images and associated text to learn how to "imagine" and create new visuals.

Image generation is powered by Generative adversarial networks (GANs), Diffusion models, or Transformers like DALL·E and Stable Diffusion. Let's break down the main steps:

1. Training the model

- To generate images, AI needs to learn patterns and relationships between visual data (images) and text descriptions. This involves a large collection of labeled images (e.g., "a red apple" paired with an image of an apple) and learning relationships when model learns what visual features (colors, shapes, textures) correspond to specific text inputs.

2. Core techniques for IG

- GANs which consist of two neural networks: generator - tries to create fake images; discriminator - evaluates whether an image is real or generated. The generator learns to create realistic images over time by "fooling" the discriminator.
- Diffusion models generate images by reversing a noise process. Starting with random noise, the model iteratively refines it into a clear image: noise is gradually removed, patterns emerge (shapes, colors, and objects).
- Transformers (e.g., DALL·E) that use a text-to-image pipeline: text encoding - the input description (e.g., "A cat on a surfboard") is converted into a mathematical representation, and image generation where the encoded text guides the model to generate an image matching the description.

3. Generating an image

1. Text encoding - the model analyzes the text to understand key elements (e.g. "sunset", "mountain range").

2. Latent space - the model translates these elements into abstract features (e.g., colors, shapes, patterns) in a high-dimensional space.
3. Rendering the image uses techniques like GANs or Diffusion, the AI combines these features to create a realistic or artistic image.

Why does image generation work? Models are trained on large datasets - millions of images and captions, helping them learn the relationship between text and visuals. AI captures high-level features like texture, color, and shape in a "latent space", allowing it to blend concepts creatively. Iterative refinement - diffusion and GANs improve the quality of generated images by iteratively refining noisy or rough visuals.

12.3.6

What is the role of a generator in a GAN?

- To create new images by learning patterns in the training data.
- To evaluate whether an image is real or generated.
- To remove noise from images.
- To encode text into mathematical representations.

12.3.7

To generate an image using Generative AI and display it with matplotlib, you can use an AI model like OpenAI's DALL·E or Hugging Face's diffusers. Here's a code snippet using Hugging Face's diffusers library, which supports stable diffusion pipelines.

```
from diffusers import StableDiffusionPipeline
import torch
import matplotlib.pyplot as plt

# Load the pre-trained Stable Diffusion model
model_id = "runwayml/stable-diffusion-v1-5"
pipe = StableDiffusionPipeline.from_pretrained(model_id,
torch_dtype=torch.float16)

# Move the pipeline to GPU if available
device = "cuda" if torch.cuda.is_available() else "cpu"
pipe = pipe.to(device)

# Define a prompt for the image generation
prompt = "A beautiful serene landscape with mountains, a
river, and a sunrise"

# Generate the image using the pipeline
try:
```

```
    generated_output = pipe(prompt=prompt)
    generated_image = generated_output.images[0]  # Get the
first generated image

    # Display the generated image with Matplotlib
    plt.imshow(generated_image)
    plt.axis("off")  # Remove axes for clean display
    plt.title("Generated Image: Serene Landscape")
    plt.show()
except Exception as e:
    print(f"Error during image generation: {e}")
```

12.3.8

Who typically owns the rights to an image generated by AI?

- The person who used the AI to generate the image.
- The AI model itself.
- The creator of the AI model.
- Nobody, AI-generated images are always in the public domain.

12.3.9

Who owns AI-generated images?

When you use an AI tool to generate an image, you might wonder, "Who owns the rights to this image?" This question isn't as straightforward as it seems. Ownership of AI-generated images depends largely on the terms and conditions of the AI tool and the laws in your country.

For instance, some AI tools, like DALL-E, allow you to own the images you generate, even for commercial use. Others, however, may claim partial ownership or impose restrictions on how you can use the generated content. The user, the creator of the AI model, or even no one at all could be considered the "owner", depending on the circumstances.

The issue gets murkier when we consider copyright laws. Copyright generally protects "original works of authorship", which implies a human creator. AI-generated images, being the result of algorithms processing data, challenge this definition. Can something generated by an algorithm truly be considered "original"?

Another challenge arises from the training data used to build AI models. If an AI model learns from copyrighted works and produces something strikingly similar, legal complications may follow. For example, if an AI generates an image that resembles a famous painting, does it infringe on the original artist's rights? These are questions many legal systems are still grappling with.

📖 12.3.10

The legal landscape surrounding AI-generated images is still evolving. One major challenge is that copyright laws in many countries require a human creator. Courts have ruled in some cases that works created solely by AI cannot be copyrighted, leaving AI-generated content in a legal gray area. For example, in 2022, a U.S. court denied copyright protection for an image created entirely by an AI tool, citing the lack of human authorship.

Another legal issue stems from the training data used by AI models. These models often learn from massive datasets that include copyrighted works. This raises concerns about whether the use of such data without explicit permission constitutes copyright infringement. For example, artists have complained that their artworks were used to train AI models without their consent, yet the models generate outputs that resemble their style.

Ethically, the misuse of AI-generated images is a growing concern. Deepfakes, which are realistic but fake images or videos of people, can be used for harmful purposes like spreading misinformation or violating privacy. Similarly, AI-generated visuals can be used to create misleading content, such as photos that make false claims about events or individuals.

Addressing these challenges requires collaboration between governments, AI developers, and users. Transparency is key - AI companies should disclose the sources of their training data and ensure that the datasets are legally obtained. Governments are also beginning to explore new regulations to address ownership and misuse of AI-generated content.

📝 12.3.11

What is one of the biggest legal challenges with AI-generated images?

- Determining copyright ownership of the generated content
- Ensuring the AI model runs efficiently
- Training the AI on copyrighted data without permission
- Making the images aesthetically pleasing

📖 12.3.12

Beyond legal considerations, ethical responsibility plays a significant role in the use of AI-generated images. One key concern is how these tools might undermine the work of human artists. Many AI models are trained on datasets that include art created by people who were not asked for permission. These artists often feel that their creative labor is being exploited by AI companies and users who benefit from the technology.

Another ethical issue arises with attribution. While some users might claim full authorship of AI-generated images, it’s more ethical to acknowledge the role of the

AI tool in the creation process. Transparency about how an image was created helps maintain fairness and honesty in creative industries.

AI-generated images also raise questions about societal impact. For example, if these tools are used to generate images for advertisements, they might reinforce stereotypes or create unrealistic expectations. Similarly, the misuse of AI-generated images for deepfakes or propaganda can cause significant harm.

To ensure ethical use, it's essential to:

- Attribute credit appropriately.
- Use AI responsibly, avoiding content that could harm individuals or communities.
- Support initiatives that regulate the use of AI-generated content in creative industries.

Ethical AI practices ensure that this technology benefits society without harming individuals or undermining human creativity.

12.3.13

What is the ethical concern about AI-generated images?

- They might undermine the work of human artists.
- They always require attribution.
- They are always free to use for any purpose.
- They are incapable of resembling human creativity.

Robotics

Chapter 13

13.1 Robotics and automation

📖 13.1.1

Automation is the use of technology to complete tasks with little or no human intervention. Think of factory machines sorting packages or assembling products. Once programmed, these systems perform their tasks without needing constant guidance from humans. This makes processes faster, more reliable, and less prone to errors. Automation can be as simple as an automatic door or as complex as a factory assembly line.

Robotics is a branch of technology that focuses on designing, building, and using robots. Robots are machines that can perform specific tasks, often replacing humans in dangerous, repetitive, or time-consuming jobs. Examples include robotic arms in car manufacturing, robots that explore the ocean floor, or even robots designed to clean your home.

Over time, automation and robotics have combined to create advanced systems that not only follow instructions but also adapt to their environment. These systems rely on artificial intelligence (AI), making robots smarter and capable of learning from their experiences.

📝 13.1.2

Which of the following statements about automation and robotics is correct?

- Robots are designed to perform specific tasks, often in place of humans.
- Automation requires constant human guidance.
- Automation and robotics have no connection to artificial intelligence.
- All robots are controlled manually by humans.

📖 13.1.3

The journey of robotics began with simple tools like automated weaving looms. These machines could operate with minimal human input, saving time and effort. Over the years, engineers and inventors developed more sophisticated machines to handle tasks that were too repetitive or dangerous for humans.

In the 20th century, robots started appearing in factories. They were used to assemble cars, weld parts, and even pack products for shipment. These robots were not "smart" - they could only follow specific instructions. However, they transformed industries by making production faster and safer.

With the rise of artificial intelligence, robots have evolved dramatically. Today, they can learn from their mistakes and adapt to new tasks. For example, robots equipped with AI can recognize objects, navigate around obstacles, or perform delicate tasks

like folding laundry. This progression shows how robotics has shifted from simple machines to intelligent systems capable of complex decision-making.

13.1.4

Which of the following is an example of early robotics?

- Automated weaving looms
- Robots learning to fold laundry.
- Robots navigating around obstacles.
- Modern AI-driven robots.

13.1.5

Artificial intelligence (AI) has revolutionized the field of robotics. In the past, robots could only follow strict, pre-written instructions. They couldn't adapt to new situations or learn from their mistakes. However, AI allows robots to think, learn, and make decisions based on the data they receive.

AI technologies like machine learning enable robots to improve their performance over time. For example, a robot vacuum cleaner can learn the layout of your house, avoiding obstacles and cleaning more efficiently. This capability is made possible by algorithms that mimic how the human brain processes information.

These advancements rely on platforms like TensorFlow and PyTorch. These tools help developers create and train AI models that robots can use. As a result, modern robots are becoming increasingly intelligent, capable of handling tasks in various environments, from hospitals to space exploration.

13.1.6

How has artificial intelligence fundamentally changed robotics?

- By limiting the functions robots can perform
- By enabling robots to learn and make autonomous decisions
- By making robots completely dependent on human operators
- By reducing the safety and efficiency of robotic systems

13.1.7

Robots in industry

Robots are widely used in industries to perform tasks that are repetitive, dangerous, or require high precision. In the automotive industry, robotic arms assemble cars, weld parts, and paint surfaces with extreme accuracy. These robots increase efficiency and reduce the risk of workplace injuries.

Outside of industries, robots are becoming more common in everyday life. For example, robotic vacuum cleaners help people keep their homes clean. Smart lawnmowers can trim grass without human supervision. Robots are even being designed to assist the elderly by performing tasks like fetching items or reminding them to take medication.

The integration of robots into daily life shows how technology is becoming more accessible and practical. As robots become smarter, their applications will continue to expand, improving our quality of life and productivity.

13.1.8

What is the primary goal of using robotics in industries?

- To increase human workload
- To automate repetitive or hazardous tasks
- To eliminate all human jobs
- To entertain and engage people

13.1.9

Which of the following is NOT a common use of robots?

- Painting walls without human supervision.
- Assembling cars in factories.
- Cleaning homes with robotic vacuums.
- Assisting the elderly with daily tasks.

13.1.10

The integration of artificial intelligence (AI) into robotics has greatly increased the autonomy of robots. Autonomy means robots can perform tasks on their own with minimal or no human guidance. For example, AI-powered robots are capable of surveying land in remote areas or carrying out search and rescue operations in disaster zones. These robots analyze their surroundings, make decisions, and adapt to challenges in real-time. This level of independence has made them indispensable in situations where human intervention may be dangerous or impractical.

One of the key advancements is in **human-robot interaction (HRI)**. Modern robots are now equipped with technologies that allow them to understand and respond to human speech and gestures. This is made possible by natural language processing (NLP), which enables robots to comprehend spoken or written language. For instance, a robot assistant in a home can understand commands like "turn off the lights" or "play music“.

Additionally, robots are gaining the ability to recognize and interpret non-verbal cues, such as facial expressions or tone of voice, through **emotional recognition**. This helps robots respond in ways that feel more natural and human-like. For example, a

robot working in customer service might recognize if a person is frustrated and respond calmly to assist them. These improvements make robots more effective in environments that require interaction with people, such as healthcare, education, and customer support.

As AI continues to advance, the integration of these technologies into robotics will open up even more possibilities, making robots smarter, more adaptable, and better at collaborating with humans.

📝 13.1.11

Which of the following are examples of how AI enhances robotics?

- Robots surveying land with minimal guidance.
- Robots responding to human speech using natural language processing.
- Robots requiring constant human supervision for every task.
- Robots ignoring non-verbal cues during interactions.

13.2 Computer vision

🕮 13.2.1

Computer vision (CV) is a critical technology in robotics, enabling robots to "see" and interpret their surroundings. It uses advanced algorithms and artificial intelligence to process visual data from cameras and sensors, helping robots understand and respond to the world around them. Without computer vision, robots would be like blind machines, unable to interact effectively with their environments.

One of the main tasks of computer vision in robotics is **object detection and recognition**. For example, a warehouse robot equipped with CV can identify and locate specific packages among thousands. This capability allows robots to perform precise and repetitive tasks, such as sorting objects or assembling products on a production line.

Another key function of computer vision is **environment mapping and navigation**. Robots, especially those used in autonomous vehicles or drones, rely on CV to create detailed maps of their surroundings. Technologies like LiDAR (Light Detection and Ranging) and stereo cameras work with CV algorithms to detect obstacles, measure distances, and plan safe paths. This is essential for tasks such as delivering packages, exploring unknown terrains, or navigating busy streets.

Computer vision also plays a significant role in **human-robot interaction**. Robots can use CV to recognize gestures, facial expressions, or even emotional cues from humans. For instance, a service robot in a store might identify a customer's waving

hand as a signal for assistance. This makes interactions smoother and more intuitive.

Lastly, CV enhances robots' ability to handle **dynamic environments**. Unlike static settings, real-world environments are constantly changing. A robot vacuum cleaner, for example, needs CV to detect new obstacles like toys on the floor and adjust its path accordingly. This adaptability is crucial for robots operating in homes, hospitals, and other unpredictable spaces.

In summary, computer vision is the "eyes and brain" of modern robots, enabling them to perceive, analyze, and respond to their surroundings with intelligence and precision. It is a cornerstone technology that makes robotics useful in a wide range of industries and applications.

13.2.2

Which of the following are key roles of computer vision in robotics?

- Helping robots recognize objects like packages.
- Allowing robots to map and navigate their environment.
- Preventing robots from interacting with humans.
- Making robots blind to changes in their surroundings.

13.2.3

What role does computer vision play in autonomous vehicles?

- It is used to enhance the vehicle's audio systems
- It enables the vehicle to navigate and understand its environment
- It is mainly used for aesthetic enhancements
- It decreases the vehicle's operational efficiency

13.2.4

Object detection and recognition

Object detection and recognition are essential capabilities in robotics, enabling robots to identify and interact with objects in their environment. These technologies allow robots to see and understand the world, making them smarter and more efficient in performing tasks.

Object detection involves identifying the presence and location of objects in an image or video. For example, a robot in a warehouse uses cameras and sensors to detect packages on shelves. Advanced algorithms like convolutional neural networks (CNNs) analyze the visual data, marking the objects and determining their positions. This helps robots to interact accurately with the objects they detect.

Object recognition takes this one step further by identifying what the object actually is. For instance, a robot in a recycling plant can distinguish between a plastic bottle, a glass jar, and a piece of cardboard. This allows it to sort items into the correct bins. Recognition is achieved using machine learning models that are trained on large datasets containing labeled images of different objects.

These capabilities are widely used in various applications:

- Manufacturing - robots detect and assemble parts with precision.
- Healthcare - robots identify medical instruments during surgeries.
- Self-driving cars - vehicles recognize pedestrians, other cars, and road signs.

Object detection and recognition rely on continuous learning. Robots improve their performance over time by analyzing new data, allowing them to adapt to different environments and tasks. This adaptability is crucial for robots working in dynamic or unstructured settings.

In conclusion, object detection and recognition are the "vision and understanding" tools of robots. They make robots capable of interacting with their surroundings intelligently, paving the way for their use in industries, homes, and public spaces.

13.2.5

Which of the following are examples of object detection and recognition?

- A robot sorting recycling materials like plastic and glass.
- A robot detecting and locating packages in a warehouse.
- A robot ignoring objects and only navigating based on predefined paths.
- A robot unable to distinguish between different types of objects.

13.2.6

Environment mapping and navigation

One of the most important abilities for robots is environment mapping and navigation. This capability allows robots to understand their surroundings, plan paths, and move safely and efficiently from one place to another. Robots achieve this through a combination of sensors, algorithms, and artificial intelligence.

Environment mapping refers to the process by which a robot creates a virtual map of its surroundings. Using technologies like LiDAR (Light Detection and Ranging), cameras, and ultrasonic sensors, robots gather data about their environment. For example, a robot in a warehouse uses LiDAR to detect the positions of shelves, walls, and other obstacles, generating a detailed map it can use for navigation.

Navigation is how robots use this map to move from one point to another. Advanced robots use algorithms like SLAM (Simultaneous Localization and Mapping), which allow them to build and update maps in real time while determining their location

within that map. This is particularly useful for dynamic environments, where objects and obstacles may move or change.

Applications of environment mapping and navigation include:

- Autonomous vehicles - cars use sensors and cameras to map roads and avoid pedestrians or other vehicles.
- Drones - aerial robots map terrain and avoid obstacles like trees or buildings.
- Robot vacuums - these devices detect furniture and obstacles in a home, planning paths to clean efficiently.

Dynamic environments present challenges, but AI enables robots to adapt. For instance, if a new obstacle, like a fallen object, appears, the robot updates its map and re-plans its path. This adaptability is crucial for robots working in unstructured settings like homes, hospitals, or outdoor spaces.

In summary, environment mapping and navigation give robots the "sense of direction" they need to operate intelligently. With these abilities, robots can work safely and efficiently, contributing to fields like transportation, logistics, and personal assistance.

13.2.7

What is a significant challenge in computer vision for robotics?

- The inability to operate without internet connectivity
- Difficulty in perceiving environments under varying conditions
- Over-reliance on solar energy
- Robots' inability to move

13.2.8

Which of the following demonstrates environment mapping and navigation?

- A robot vacuum detecting furniture and planning cleaning paths.
- A drone mapping terrain and avoiding obstacles like trees.
- A robot only moving in straight lines without detecting obstacles.
- A robot ignoring changes in its surroundings.

13.2.9

Dynamic environments

Dynamic environments are spaces where conditions change constantly, such as a busy street, a cluttered room, or a hospital ward. For robots to work effectively in such environments, they must adapt quickly to new situations. CV plays a critical role in enabling robots to handle these challenges by providing them with the ability to "see" and interpret their surroundings.

Using CV, robots can detect objects, people, and obstacles in real time. For example, a delivery robot navigating a crowded mall uses cameras and computer vision algorithms to identify moving shoppers, stationary objects, and pathways. By processing this data, the robot can adjust its route to avoid collisions and reach its destination safely.

In addition to obstacle avoidance, CV allows robots to recognize patterns and changes in the environment. For instance, a robot in a warehouse might notice that a package has been moved to a new location. It updates its internal map and modifies its task accordingly. This ability to react to unexpected changes makes CV-equipped robots highly effective in dynamic settings.

Robots with CV also excel in tasks that require precision in shifting environments. A robot vacuum cleaner can identify and navigate around toys scattered on the floor. Similarly, autonomous vehicles use CV to detect and react to sudden changes, like a pedestrian crossing the street or an obstacle appearing unexpectedly.

Key benefits of CV in dynamic environments include:

- Real-time obstacle detection and avoidance where robots process visual data instantly to navigate safely.
- Adaptability to changes allows to robots to update their plans based on new information.
- Improved efficiency is achieved by making complete tasks faster and more accurately, even in unstructured spaces.

📝 13.2.10

What are the underlying technologies that enable AI in robotics

- It allows robots to detect and avoid obstacles in real time.
- Robots can recognize changes and update their tasks accordingly.
- Robots ignore unexpected changes to stay on their preplanned paths.
- Robots stop functioning if their environment changes.

13.3 AI in robotics

🕮 13.3.1

Artificial intelligence tools are the backbone of modern robotics, providing robots with the ability to think, learn, and adapt. These tools enable robots to perform complex tasks that require more than just following pre-programmed instructions. Let’s explore some key AI tools used in robotics and why they are essential.

- Machine learning as a branch of AI that enables robots to learn from data and improve their performance over time. For example, a robot trained with ML algorithms can recognize and sort objects of different shapes and sizes. ML tools are vital because they help robots handle tasks that require adaptation and precision, such as picking delicate items in warehouses or identifying patterns in healthcare diagnostics.
- Deep learning frameworks like TensorFlow and PyTorch are used to create and train neural networks, which mimic how the human brain processes information. These frameworks allow robots to perform tasks such as image recognition, speech understanding, and decision-making. For instance, a robot equipped with TensorFlow can identify faces or detect anomalies in industrial machines. These tools are crucial for enhancing a robot's perception and decision-making abilities.
- Robots that interact with humans often use NLP tools, such as OpenAI's GPT or similar language models. NLP allows robots to understand and respond to spoken or written language. This is essential for robots in customer service, healthcare, or education, where communication with humans is key. For example, a robot nurse might use NLP to understand a patient's request and provide the correct assistance.
- Reinforcement learning is a technique where robots learn by trial and error. Using platforms like OpenAI Gym, robots are trained to solve problems by receiving rewards for successful actions and penalties for mistakes. This approach is vital for tasks like navigation or game-playing robots, where dynamic and unpredictable environments require constant learning and adjustment.
- Computer vision tools like OpenCV allow robots to process visual data and recognize objects, people, or movements. This is critical for robots that operate in environments requiring visual understanding, such as autonomous vehicles, drones, or security robots.

AI tools make robots:

- Smarter because robots can analyze data and make decisions, rather than just following fixed instructions.
- Adaptable because they can adjust to new environments or tasks, such as working in a warehouse one day and a hospital the next.
- Efficient because robots equipped with AI tools can perform tasks faster and with fewer errors, making them invaluable in industries like manufacturing, logistics, and healthcare.

13.3.2

Which AI tools are commonly used in robotics, and why are they important?

- Machine learning algorithms for helping robots improve performance over time.
- NLP tools for enabling robots to understand and respond to human language.

- Ignoring deep learning frameworks to keep robots simple.
- Reinforcement learning platforms for adapting to dynamic environments.

🕮 13.3.3

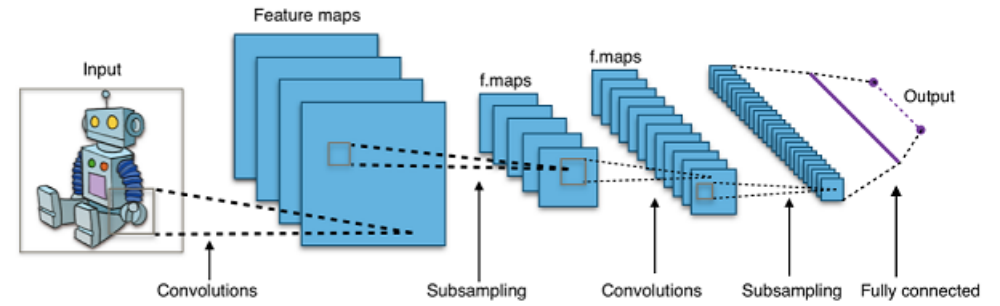
Deep learning networks for object identification

When it comes to identifying objects, deep learning networks are incredibly powerful because they can learn to recognize patterns, shapes, and features in images with high accuracy.

Deep learning networks are composed of layers of artificial neurons. Each neuron processes information and passes it to the next layer. For object identification, these networks are often structured as **Convolutional neural networks (CNNs)**, which are specifically designed for image processing tasks.

When an image is input into the network, the first layers (called **convolutional layers**) identify simple features like edges or corners. As the image passes through deeper layers, the network learns more complex patterns, such as shapes and textures.

Once the features are extracted, the final layers of the network (called **fully connected layers**) analyze the combined information to classify the object. For example, the network might decide whether an image contains a car, a dog, or a tree. The output is typically a probability score for each possible category, and the highest score determines the identified object.



Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

A CNN consists of several layers, each performing a specific function to extract and analyze features from an image. The primary layers in a CNN are:

1. Convolutional layer - the core building block of a CNN. In this layer:

- Small filters (or kernels) slide across the input image, performing a mathematical operation called **convolution**.
- This operation detects patterns like edges, lines, or corners in the image.
- The result is a feature map that highlights where specific patterns occur.

2. Pooling layer - after convolution, the pooling layer reduces the size of the feature map:

- It downsamples the data, keeping only the most important features.
- This process reduces computational complexity and helps the network focus on key patterns.
- Common pooling methods include **max pooling** (keeping the highest value) and **average pooling** (taking the average value).

3. Fully connected layer - at the end of the network, the fully connected layer takes the processed data and uses it to make predictions:

- It connects all the neurons from the previous layer to create a final output.
- For example, if the task is to classify an image of animals, the output might be probabilities for categories like "dog“, "cat“, or "bird“.

When an image is fed into a CNN:

- The convolutional layers extract patterns and features from the image, such as edges, textures, and shapes.
- Pooling layers simplify the data, focusing on the most important features.
- Fully connected layers analyze these features and produce the final result, such as identifying what object is in the image.

Feature maps are intermediate outputs produced by the convolutional layers. These maps are critical because they represent the patterns and features extracted from the input data, like edges, textures, or shapes in an image. Feature maps focus on different aspects of an image:

- In **early layers**, feature maps detect simple patterns like edges and corners.
- In **deeper layers**, feature maps capture more complex patterns, such as textures or specific shapes.
- These progressively abstract features allow the network to recognize objects and make decisions.

13.3.4

What are the main components of a CNN?

- Convolutional layers for extracting patterns like edges and shapes.
- Pooling layers for reducing feature map size and focusing on key patterns.
- Ignoring features and analyzing the image as a whole.
- Fully connected layers for producing final predictions.

13.3.5

What makes deep learning networks effective for object identification?

- They use convolutional layers to extract features like edges and shapes.
- They rely on labeled datasets to learn and improve.
- They cannot process complex or cluttered images.
- They analyze objects without learning patterns.

13.3.6

What do feature maps represent in a CNN?

- Patterns like edges, shapes, and textures in an image.
- Abstract features that increase in complexity with deeper layers.
- Raw pixel values without any processing.
- The final classification output of the network.

13.3.7

Reinforcement learning in robotics

Reinforcement learning (RL) is a type of machine learning where an agent (like a robot) learns to make decisions by interacting with its environment. Reinforcement learning makes robots more adaptive, autonomous, and versatile. They don't just follow pre-programmed rules; they learn from their mistakes, adapt to new challenges, and improve their performance over time. The agent performs actions, observes the results, and receives feedback in the form of rewards or penalties. Over time, the robot learns to choose actions that maximize rewards and avoid penalties.

In RL, the robot interacts with its surroundings in a continuous loop:

- The robot takes an **action**, like turning left or moving forward. This action changes its **state**, such as moving closer to a goal or hitting an obstacle.
- After each action, the robot gets **feedback**. Positive actions, like finding the right path in a maze, are rewarded, while negative actions, like hitting a wall, result in penalties.
- By repeating this process, the robot **adjusts its behavior**. It learns to prioritize actions that lead to better outcomes and avoids those that result in poor results.
- For example, a robot trying to navigate a maze will explore various paths. Initially, it may hit walls or take wrong turns, but over time it learns the most efficient route by associating successful actions with rewards.

Robots use both exploration and exploitation. Exploration describes robot activity focused on trying new actions to discover better strategies, exploitation the use of its past experiences to make the best decision in a given situation.

The reward function defines the robot's goal. For instance, a self-driving car's reward might be reaching its destination safely, while penalties are assigned for actions like collisions or running red lights.

A popular RL method where the robot learns to evaluate the long-term benefits of each action, not just the immediate reward. It assigns a "Q-value" to actions, helping the robot make smarter decisions over time.

Applications of reinforcement learning in robotics:

- Navigation - robots learn to navigate through complex environments like warehouses or disaster zones.
- Manipulation - robots master handling delicate or complex objects, such as assembling parts or sorting items.
- Autonomous vehicles - cars use RL to make decisions like stopping at traffic signals, avoiding pedestrians, or merging into traffic.
- Interactivity - robots learn to interact effectively with humans, such as providing assistance or playing games.

13.3.8

Which of the following are key aspects of reinforcement learning?

- The agent learns by receiving rewards and penalties.
- The agent relies solely on predefined rules for actions.
- Exploration allows the agent to discover new strategies.
- The agent stops learning once it encounters penalties.

13.3.9

How does reinforcement learning improve a robot's performance?

- By allowing the robot to learn from rewards and penalties.
- By enabling the robot to adjust its behavior based on experience.
- By relying on fixed instructions with no adjustments.
- By avoiding exploration to prevent mistakes.

13.4 Robotics and society

📖 13.4.1

Automation powered by artificial intelligence is revolutionizing industries by replacing repetitive, manual tasks with machines and software. For instance, in the **automotive industry**, robotic arms are widely used to assemble cars, weld parts, and paint vehicles with greater speed and precision than humans. These systems work tirelessly, increasing efficiency and reducing errors. Similar automation technologies are being implemented in industries like manufacturing, agriculture, and logistics, where repetitive tasks are common.

While automation brings clear benefits, it also creates challenges for the workforce. Workers who once performed these tasks may find their roles replaced by machines. However, this doesn't mean their careers are over. There is a growing demand for **new skills** to operate, program, and maintain these systems. For example, workers can transition from performing assembly tasks to becoming technicians who ensure that robotic systems are functioning properly.

The impact on the workforce varies across industries. While some jobs may disappear, others are transformed, requiring workers to learn skills like **robotic maintenance, AI system monitoring, or programming industrial machines**. Training programs and government initiatives are essential to help workers adapt to these changes and avoid job losses.

📝 13.4.2

How is automation transforming industries?

- By replacing repetitive tasks with robotic systems.
- By creating a demand for new skills like robotic maintenance.
- By completely eliminating the need for human workers in all industries.
- By maintaining jobs as they were before automation.

📖 13.4.3

The rise of artificial intelligence has given birth to entirely new professions that didn't exist just a decade ago. **Data scientists** are among the most sought-after professionals today, tasked with analyzing massive datasets to extract insights that drive decision-making. Similarly, **machine learning engineers** develop AI models that allow systems to learn from data, improving their performance over time.

Other emerging roles include **AI developers**, who create applications powered by AI, and **AI ethics advisors**, who help ensure these systems are designed responsibly and without harmful biases. These roles are critical in industries ranging from healthcare to finance, where AI technologies are being integrated into core operations.

To succeed in these professions, workers need skills in **programming (e.g., Python, R)**, **data analysis**, and familiarity with **machine learning frameworks like TensorFlow and PyTorch**. Soft skills like ethical decision-making and critical thinking are also essential, particularly in roles focused on AI ethics. As the AI industry continues to grow, these professions contribute significantly to the economy by driving innovation and solving complex problems.

13.4.4

Which of the following is an example of a profession created by AI advancements?

- Machine learning engineer.
- AI ethics advisor.
- Automobile mechanic.
- Grocery store cashier.

13.4.5

AI is transforming the way we work by enabling **flexible working models**, including remote work. With AI-powered tools, employees can collaborate on projects, attend virtual meetings, and access shared documents from anywhere in the world. This shift allows companies to adopt **home office models**, which improve work-life balance for many employees.

However, these new working models come with challenges. For instance, remote workers need strong **self-management skills** to plan their tasks effectively and meet deadlines without direct supervision. Additionally, companies must implement systems to monitor and manage productivity fairly, ensuring that remote work doesn't lead to decreased efficiency.

AI also helps automate administrative tasks, like scheduling meetings or tracking project progress, which allows workers to focus on higher-value activities. While this technology provides convenience, it also requires employees to be comfortable with AI-based tools and software.

In summary, AI is reshaping work by making it more flexible, but workers need to adapt to new expectations, such as being more independent and proficient with digital tools.

13.4.6

What are the benefits and challenges of AI-enabled working models?

- Improved work-life balance through remote work.
- The need for workers to develop self-management skills.
- A decrease in productivity due to AI tools.
- The inability to collaborate remotely on projects.

📖 13.4.7

Artificial intelligence is having a profound impact on society, changing how people live and work. One major effect is the **transformation of job structures**. While AI creates opportunities in high-tech roles like data analysis and machine learning, it also automates many routine jobs, leading to concerns about **job losses** in sectors like manufacturing and retail.

AI also contributes to **economic inequality**, as those with the skills to work with AI technologies benefit the most, while others may struggle to find employment. This creates a gap between workers with advanced technical skills and those in traditional roles. Governments and organizations are debating how to address these challenges, including offering training programs, creating new policies, and implementing universal basic income.

On the positive side, AI can improve efficiency, drive innovation, and solve global challenges, such as improving healthcare systems or addressing climate change. However, society must balance these benefits with strategies to minimize **negative impacts**, like ensuring equal access to education and technology.

In summary, AI's social and economic effects are a mix of opportunities and challenges, requiring careful planning to maximize benefits and reduce inequalities.

📝 13.4.8

How does AI impact society and the economy?

- It transforms job structures, creating new roles and automating routine tasks.
- It increases inequality by benefiting only those with advanced technical skills.
- It eliminates innovation and slows down global progress.
- It ensures equal benefits for all workers, regardless of skill levels.

The Future of AI

Chapter 14

14.1 Trends in AI for technology

📖 14.1.1

Deep learning

Deep learning is one of the most exciting areas in artificial intelligence, driving progress in many fields. Deep learning uses neural networks, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to analyze and learn from massive amounts of data. These advanced architectures enable AI systems to recognize patterns, process language, and make decisions with remarkable accuracy.

Convolutional Neural Networks (CNNs) are widely used in image recognition tasks, allowing AI to identify objects, detect faces, and even analyze medical scans for diseases. Meanwhile, RNNs excel in processing sequences of data, making them invaluable for natural language processing tasks like translation, chatbots, and language models.

The development of transformer-based architectures like GPT has revolutionized AI's ability to understand and generate human-like text, marking a leap forward in natural language processing. These advancements have accelerated AI's growth, enabling applications in healthcare, finance, entertainment, and more.

In the future, we can expect even more efficient and versatile deep learning models. These models will require less data for training, make faster predictions, and integrate seamlessly into everyday technologies, reshaping industries and enhancing human experiences.

📝 14.1.2

What advancements in deep learning are shaping AI's future?

- CNNs for image recognition.
- RNNs for processing sequential data.
- The elimination of neural networks from AI.
- The use of transformer architectures for natural language tasks.

📖 14.1.3

Augmented and virtual reality

Augmented reality (AR) and Virtual reality (VR) are transforming how humans interact with technology. By overlaying digital elements onto the real world (AR) or creating fully immersive virtual spaces (VR), these technologies enhance experiences in entertainment, education, and work. AI plays a pivotal role in making AR and VR more intelligent and interactive.

AI enhances AR by enabling advanced spatial awareness. For example, it allows AR devices to recognize and interact with physical environments, such as furniture in a room, and overlay relevant information or virtual objects. In VR, AI can personalize content, such as generating custom environments or tailoring recommendations to individual users.

Innovative projects like Meta's Metaverse envision virtual spaces where people can socialize, work, and share data in immersive environments. Similarly, devices like Apple’s Vision Pro glasses bring augmented reality into everyday life, allowing users to interact with virtual elements seamlessly integrated into their surroundings.

In the future, AI-driven AR and VR will become more accessible, blending virtual and real worlds in ways that revolutionize communication, learning, and creativity.

📝 14.1.4

What are the roles of AI in AR and VR?

- Enhancing spatial awareness in AR environments.
- Creating personalized VR content.
- Eliminating physical objects in AR.
- Making AR and VR less interactive.

🕮 14.1.5

Robotics and autonomy

Artificial intelligence is driving significant progress in **robotics**, making robots smarter, more autonomous, and better at adapting to new environments. Modern robots can learn on their own, enabling them to handle tasks that were once impossible without human intervention.

In manufacturing, robots are becoming collaborators, working alongside humans to perform repetitive tasks with precision while humans focus on problem-solving. In healthcare, robots assist in surgeries or provide companionship and assistance to patients.

Autonomous technologies are also advancing in vehicles. **Self-driving cars** and systems with partial autonomy rely on AI to navigate, avoid obstacles, and make real-time decisions. Robots like **Tesla Optimus**, designed to look and act human-like, represent a major trend toward creating robots capable of working efficiently in human environments.

These developments promise a future where humans and robots collaborate at unprecedented levels, increasing productivity, safety, and efficiency across industries.

📝 14.1.6

Which advancements are part of AI-driven robotics?

- Robots learning and adapting to new environments.
- Fully autonomous vehicles and human-like robots.
- Robots ignoring collaboration with humans.
- Replacing all human workers in healthcare.

🕮 14.1.7

Smart homes

Artificial intelligence is transforming **smart homes** by automating everyday tasks and optimizing energy consumption. Smart systems equipped with AI can automatically adjust heating, shading, and ventilation based on the user's preferences and environmental conditions. For example, an intelligent thermostat learns when you're home and adjusts the temperature to match your comfort level while conserving energy.

The main goal of AI in smart homes is to make life **simpler and more user-friendly**. By learning from historical data, such as your daily routines and preferences, these systems create a personalized experience. For instance, lights might automatically dim at bedtime, or the coffee maker could start brewing as you wake up.

AI-driven smart homes also enhance energy efficiency by monitoring and optimizing resource usage. This not only saves money but also benefits the environment. As AI in smart homes continues to evolve, it promises to make daily living more convenient, efficient, and sustainable.

📝 14.1.8

What is the role of AI in smart homes?

- Automating tasks like heating and ventilation based on user preferences.
- Learning from historical data to create a personalized experience.
- Ignoring energy efficiency to prioritize user comfort.
- Reducing convenience by adding more manual steps.

🕮 14.1.9

Transport

Artificial intelligence is making transportation systems safer and more efficient. **Autonomous vehicles**, such as self-driving cars, use advanced machine learning algorithms to recognize their surroundings, detect obstacles, and adapt to traffic

conditions. This reduces the risk of accidents and improves overall traffic flow, paving the way for safer roads.

In addition to autonomous vehicles, AI is being used to manage traffic systems in **smart cities**. For example, AI can optimize the timing of traffic signals at busy intersections, reducing congestion and minimizing delays. AI can also improve public transit by predicting passenger demand and suggesting optimal routes for buses and trains.

AI's role in transport extends to **reducing environmental impact**. By improving traffic flow and enabling smarter logistics, AI helps reduce fuel consumption and emissions. As these technologies advance, they will play a vital role in creating safer, greener, and more efficient transportation systems.

📝 14.1.10

What are the uses of AI in transportation?

- Managing traffic systems in smart cities.
- Enabling autonomous vehicles to navigate safely.
- Increasing congestion by ignoring traffic patterns.
- Replacing all human drivers with no oversight.

🕮 14.1.11

Data analysis

As the volume of data grows, **data analysis using AI** has become increasingly important. AI-driven tools can process vast datasets, uncovering complex patterns and relationships that are impossible for humans to detect manually. This capability is transforming fields like **business analytics**, where AI helps companies predict trends and optimize operations, and **biomedicine**, where AI identifies patterns in genetic or clinical data to improve treatments.

Modern techniques like **machine learning** and **big data analytics** allow organizations to derive actionable insights from massive amounts of information. For example, AI can help retailers understand customer behavior, detect fraud in financial systems, or identify new drug candidates in medical research.

This trend is accelerating due to the growth of data storage and computing power. As AI becomes more sophisticated, it will affect all aspects of life, from personalized healthcare to smarter cities. Understanding how to analyze and leverage data effectively will be essential for businesses and individuals alike.

📝 14.1.12

What are key benefits of AI in data analysis?

- Identifying patterns in genetic and clinical data.
- Helping businesses optimize operations and predict trends.
- Reducing the importance of data in decision-making.
- Replacing computing power with manual analysis.

14.2 Trends in AI for society

🕮 14.2.1

Working with language

Natural language processing (NLP) technologies are advancing rapidly, enabling artificial intelligence (AI) to better understand human speech and written text. Modern language models, such as **BERT (Bidirectional Encoder Representations from Transformers)**, achieve a deep understanding of context and nuances in language. This allows for advanced applications like **automatic translations**, **query answering**, and even creative writing.

Today, numerous language models are widely used in everyday applications. **OpenAI's ChatGPT-4**, **Microsoft Bing**, **Google Bard**, and **Microsoft Copilot** are examples of AI tools integrated into various platforms. For instance, Microsoft Copilot works within Office 365 to enhance productivity, while Google Bard assists in conversational tasks. These tools aim to simplify communication, assist with tasks like **online translations**, and enable advanced features like **intelligent searches**.

Additionally, companies are embedding NLP tools into consumer devices. For example, Samsung and Google are integrating AI into smartphones, providing features like **real-time call translations** and **advanced photo editing**. This progress is not just about understanding language but also making daily interactions with technology more natural and intuitive.

📝 14.2.2

Which tasks are enabled by modern NLP technologies?

- Automatic translation of human speech.
- Understanding the context of human language.
- Ignoring nuances in written text.
- Replacing phones with non-interactive systems.

📖 14.2.3

Artificial creativity

Artificial intelligence is revolutionizing **creativity** by generating original artworks, music, and designs. AI algorithms analyze vast amounts of data to create images, compose melodies, or write compelling texts based on user preferences and previous patterns. This has led to innovations in **generative design**, where AI assists designers in creating aesthetically pleasing and functional products for interiors, exteriors, and other fields.

A recent trend involves the generation of images and other media using tools like **Midjourney**, **DALL-E 2**, and similar services. These systems allow users to create stunning visuals simply by describing their ideas. However, they also raise ethical concerns about the potential misuse of AI-generated content, such as creating **falsified materials** for news or social media.

Another exciting development is **Sora**, OpenAI's realistic video generator. Although currently limited to generating one-minute videos, it shows immense potential for transforming the **film industry**. As AI-generated videos improve, they could reduce the demand for certain roles, change the way movies are made, and create new opportunities in digital storytelling.

📝 14.2.4

How is AI transforming creativity?

- By generating original images and music.
- By assisting designers in generative design projects.
- By eliminating the need for creativity in design.
- By ensuring all AI-generated materials are accurate.

📖 14.2.5

Healthcare

Artificial intelligence (AI) is revolutionizing **healthcare** by improving diagnostics and enabling personalized treatment. For example, **machine learning algorithms** can analyze medical images, such as X-rays or MRIs, to detect abnormalities like tumors with incredible accuracy - sometimes even outperforming human doctors. This capability reduces diagnostic errors and speeds up treatment.

AI also helps create **personalized treatment plans** by analyzing historical data about a patient's condition, genetic makeup, and medical history. This allows for more targeted and effective therapies. Additionally, AI systems make healthcare more accessible in areas where specialized medical personnel are unavailable. For instance, chatbots powered by AI can answer health-related questions or triage symptoms for patients in remote locations.

Despite its advantages, AI in healthcare also faces challenges, such as ensuring data privacy and integrating AI systems with traditional healthcare practices. However, as these technologies improve, they hold immense potential to transform how care is delivered, making it more precise, accessible, and efficient.

📝 14.2.6

How does AI improve healthcare?

- By analyzing medical images to detect abnormalities.
- By creating personalized treatment plans based on patient data.
- By making medical care inaccessible in remote areas.
- By replacing traditional healthcare practices entirely.

🕮 14.2.7

Privacy and security

With the increasing integration of artificial intelligence (AI) into everyday life, concerns about **privacy and data protection** have grown significantly. AI systems often rely on vast amounts of user data, which raises questions about how personal information is collected, stored, and used. Examples include user profiles on the internet, data from apps, and even surveillance footage from the dense network of city cameras.

To address these concerns, advancements in **data encryption** and **anonymization techniques** are improving security. Encryption ensures that data transmitted between systems is protected from unauthorized access, while anonymization removes identifiable information, safeguarding user privacy. These technologies are crucial for preventing **security breaches** and ensuring trust in AI systems.

The regulation of AI is another important area. For example, the European Union is incorporating elements of AI regulation to ensure that AI technologies adhere to high standards of privacy and security. These regulations aim to create a balance between innovation and protecting user rights.

AI itself is also being used to enhance security. For example, machine learning algorithms can detect unusual activity, such as attempted hacks, and prevent data breaches in real-time. However, achieving privacy and security in AI systems remains an ongoing challenge as technology continues to evolve.

📝 14.2.8

How is privacy and security being addressed in AI systems?

- By using encryption and anonymization to protect data.
- By implementing regulations like those in the EU.
- By collecting and sharing user data without restrictions.

- By avoiding advancements in data protection technologies.

📖 14.2.9

Justice and equality

Artificial intelligence has the potential to amplify social inequalities if it is not designed with care. AI systems often rely on historical data to make predictions or decisions. If the data contains biases, the AI may unfairly favor or discriminate against certain groups, worsening existing inequalities.

To address this issue, researchers are developing methodologies that emphasize **justice and fairness** in AI. For example, algorithms are being designed to treat all users and groups equally, avoiding practices like scoring or ranking people based on irrelevant characteristics. These approaches aim to ensure that AI systems work fairly and inclusively.

AI's role in promoting equality also extends to accessibility. Systems are being developed to serve diverse populations, including those with disabilities or limited access to technology. For instance, AI-powered tools like screen readers and real-time translations are making digital platforms more inclusive.

In the future, ensuring justice and equality in AI will require collaboration between developers, policymakers, and social organizations. The goal is to create AI systems that serve everyone fairly, without perpetuating discrimination or bias.

📝 14.2.10

What steps are being taken to ensure justice and equality in AI?

- Avoiding biased scoring or ranking of users.
- Designing algorithms that treat all users equally.
- Using historical data without checking for biases.
- Ignoring accessibility for diverse populations.

📖 14.2.11

Employment

The rise of artificial intelligence and automation is reshaping the **labor market**, bringing both opportunities and challenges. On one hand, AI can enhance productivity and efficiency by automating repetitive tasks. For example, companies are using robots for multi-shift operations to reduce costs and improve precision. On the other hand, this shift may lead to job losses, as machines replace human workers in certain roles.

Studies are exploring how the workforce can adapt to these changes. Strategies include providing workers with training and skills development to transition into new roles created by AI, such as data analysts or AI system operators. Policymakers are also discussing ways to minimize negative impacts, like supporting retraining programs and ensuring that workers displaced by automation have access to new opportunities.

Another emerging topic is the taxation of robots and automated systems. As automation reduces the need for human labor, governments may face a decline in wage-based tax revenue. Taxing the "work" done by robots - both hardware and software - could provide a way to offset these losses and support social programs.

In summary, while AI has the potential to transform industries, careful planning and proactive measures are essential to ensure that the benefits are shared and the negative impacts on workers are minimized.

14.2.12

What are potential impacts of AI on employment?

- Increased efficiency through automation.
- The need for workers to transition to new roles.
- Eliminating all job opportunities for humans.
- Ignoring taxation for robot-based labor.

Resources

Chapter 15

15.1 Bibliography

🕮 15.1.1

Bibliography:

1. Accenture. (2024). Technology Trends 2024. Available to: https://www.accenture.com/us-en/insights/technology/technology-trends-2024
2. Alechina. N. Planning and Search. University of Nottingham. Online. 2014. Available to: https://www.cs.nott.ac.uk/~psznza/G52PAS/lecture9.pdf
3. Anandraj, J. Non Linear Planning with Constraint Posting. 2018. Online. Available to: https://aithefuture.wordpress.com/2018/05/08/non-linear-planning-with-constraint-posting/
4. ASI is now ConvergeOne. (2019). Artificial Intelligence (AI) and Natural Language Processing (NLP) Made Easy. Online. Available to: https://www.youtube.com/watch?v=UGWDso6PmmE
5. Augmented Startups. (2023). The Role of Computer Vision in Robotics: Advancements, Applications, and Future Implications. Available to: https://www.augmentedstartups.com/blog/the-role-of-computer-vision-in-robotics-advancements-applications-and-future-implications
6. Autoblocks. Stanford Research Institute Problem Solver (STRIPS). Generated by AI. Online. Available to: https://www.autoblocks.ai/glossary/stanford-research-institute-problem-solver
7. Bishop, Christopher M. Pattern Recognition and Machine Learning. Springer, 2006.
8. Brassai, S. T., Iantovics, B., & Enăchescu, C. (2012). Artificial Intelligence in the Path Planning Optimization of Mobile Agent Navigation. In Procedia Economics and Finance (Vol. 3, pp. 243–250). Elsevier BV. https://doi.org/10.1016/s2212-5671(12)00147-5
9. Brave. (2023) What is natural language processing (NLP)? Online. Available to: https://brave.com/ai/what-is-natural-language-processing/
10. Bundy, A., & Wallen, L. Non-Linear Planning. In Catalogue of Artificial Intelligence Tools. 1984. (pp. 82–83). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-96868-6_158
11. Burkov, Andriy, The Hundred-Page Machine Learning Book, 2019, ISBN 978-1-9995795-0-0
12. Burrows, L. (2021). The present and future of AI. Harvard John A. Paulson School of Engineering and Applied Sciences. Retrieved from https://seas.harvard.edu/news/2021/10/present-and-future-ai
13. Coding and More. (2023). Natural Language Processing - What, Where, How (Google Colab, Huggin Face). Online. Available to: https://www.youtube.com/watch?v=PGczHa_GKkc
14. Crusius, J., Corcoran, K., & Mussweiler, T. (2021). Social Comparison: A Review of Theory, Research, and Applications. ResearchGate. Retrieved from https://www.researchgate.net/publication/353803747_Social_Comparison_A_Review_of_Theory_Research_and_Applications

15. Deshpande, S., Kashyap, A. K., & Patle, B. K. (2023). A review on path planning ai techniques for mobile robots. In Robotic Systems and Applications (Vol. 3, Issue 1, pp. 27–46). JVE International Ltd. https://doi.org/10.21595/rsa.2023.23090
16. Domshlak, K. Automated (AI) Planning Planning tasks & Search. CVUT FEL. Online. Available to: https://cw.fel.cvut.cz/old/_media/courses/a4m33pah/02-search.pdf
17. Ertel, Wolfgang, Introduction to Artificial Intelligence, 2017, ISBN 9783319584874
18. Fikes, R. Nilsson, R. (1971) STRIPS: A New Approach to the Application of .Theorem Proving to Problem Solving. Standford Research Institute. Online. Available to: https://ai.stanford.edu/users/nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf
19. FindLight. (2024). AI in Robotics: Exploring Emerging Trends and Future Prospects. Available to: https://www.findlight.net/blog/ai-in-robotics-exploring-emerging-trends-and-future-prospects/
20. Glover, E., Whitfield, B. (2023). The ELIZA Effect: How Chatbots Impact Human Behavior. Built In. Available to: https://builtin.com/artificial-intelligence/eliza-effect
21. Goodfellow, I., Bengio, Y. & Courville, A., "Deep Learning, MIT Press. Available to: https://www.deeplearningbook.org/
22. Hastie, Trevor, Robert Tibshirani, a Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd vydání. Springer, 2009.
23. https://aidetem.cz/obecny-uvod-do-umele-inteligence
24. https://avs431.medium.com/explain-it-to-me-like-a-5-year-old-beginners-guide-to-deep-learning-neural-network-1eb6a979f74
25. https://blog.ovhcloud.com/deep-learning-explained-to-my-8-year-old-daughter/
26. https://corporatefinanceinstitute.com/resources/data-science/regression-analysis/
27. https://edu.ukf.sk/course/view.php?id=5334
28. https://medium.com/@endothermic-dragon/in-depth-machine-learning-for-teens-neural-networks-ded1af6a84de
29. https://medium.com/@novus_afk/understanding-logistic-regression-a-beginners-guide-73f148866910
30. https://medium.com/@satyarepala/understanding-logistic-regression-a-step-by-step-explanation-9a404344964b
31. https://medium.com/analytics-vidhya/a-comprehensive-guide-to-logistic-regression-e0cf04fe738c
32. https://medium.com/analytics-vidhya/understanding-logistic-regression-b3c672deac04
33. https://medium.com/analytics-vidhya/understanding-the-linear-regression-808c1f6941c0
34. https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389
35. https://scikit-learn.org/1.5/auto_examples/cluster/plot_dbscan.html

36. https://utsavdesai26.medium.com/linear-regression-made-simple-a-step-by-step-tutorial-fb8e737ea2d9
37. https://www.akkio.com/post/the-five-main-subsets-of-ai-machine-learning-nlp-and-more
38. https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/
39. https://www.codingal.com/coding-for-kids/blog/what-is-a-neural-network/
40. https://www.coursera.org/articles/history-of-ai
41. https://www.cuemath.com/data/least-squares/
42. https://www.datacamp.com/tutorial/understanding-logistic-regression-python
43. https://www.geeksforgeeks.org/artificial-intelligence-in-robotics/
44. https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/
45. https://www.geeksforgeeks.org/hierarchical-clustering/
46. https://www.geeksforgeeks.org/informed-search-algorithms-in-artificial-intelligence/
47. https://www.geeksforgeeks.org/k-means-clustering-introduction/
48. https://www.inspiritscholars.com/blog/what-is-ai-for-kids/
49. https://www.kaggle.com/code/prashant111/logistic-regression-classifier-tutorial
50. https://www.researchgate.net/publication/345340198_Depiction_the_Artificial_Intelligence_with_Machine_Learning/figures?lo=1
51. https://www.scribbr.com/statistics/simple-linear-regression/
52. Choudhary, K., DeCost, B., Chen, C., Jain, A., Tavazza, F., Cohn, R., Park, C. W., Choudhary, A., Agrawal, A., Billinge, S. J. L., Holm, E., Ong, S. P., & Wolverton, C. (2022). Recent advances and applications of deep learning methods in materials science. In npj Computational Materials (Vol. 8, Issue 1). Springer Science and Business Media LLC. https://doi.org/10.1038/s41524-022-00734-6
53. IBM. What is artificial intelligence (AI)? Available to: https://www.ibm.com/topics/artificial-intelligence
54. Interaction Design Foundation. (n.d.). Narrow AI. Interaction Design Foundation. Retrieved from https://www.interaction-design.org/literature/topics/narrow-ai
55. Kabudi, T., Pappas, I., & Olsen, D. H. (2021). AI-enabled adaptive learning systems: A systematic mapping of the literature. In Computers and Education: Artificial Intelligence (Vol. 2, p. 100017). Elsevier BV. https://doi.org/10.1016/j.caeai.2021.100017
56. Khermos, L. (2022). How AI technology can aid natural language processing deployment. Online. Available to: https://www.eenewseurope.com/en/how-ai-technology-can-aid-natural-language-processing-deployment/
57. Kristalina Georgieva. (2024, 14. ledna). AI Will Transform the Global Economy. Let's Make Sure It Benefits Humanity. IMF. Available to: https://www.imf.org/en/Blogs/Articles/2024/01/14/ai-will-transform-the-global-economy-lets-make-sure-it-benefits-humanity

58. Kyt Dotson. (2023). Nvidia brings on new advances in robotics and computer vision AI. Available to: https://siliconangle.com/2023/03/21/nvidia-brings-new-advances-robotics-computer-vision-ai/
59. Laskowski, L. What is artificial intelligence (AI)? Everything you need to know. Available to: https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence
60. Law, M. (2022). From ELIZA to ChatGPT: The evolution of chatbots technology. AI & Machine Learning. Available to: https://technologymagazine.com/articles/from-eliza-to-chatgpt-the-evolution-of-chatbots-technology
61. Luhui Hu. (2023). Computer Vision: 2023 Recaps and 2024 Trends. Towards AI. Available to: https://towardsai.net/p/machine-learning/computer-vision-2023-recaps-and-2024-trends
62. Marguerita Lane. (2021). The impact of AI on the labour market: is this time different? OECD. Available from https://oecd.ai/en/wonk/impact-ai-on-the-labour-market-is-this-time-different
63. McCorduck, P., Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence, A. K. Peters/CRC Press.
64. MIT Computer Science & Artificial Intelligence Laboratory. (2022). ELIZA wins Peabody Award. Available to: https://www.csail.mit.edu/news/eliza-wins-peabody-award
65. Natale, S. (2021). The ELIZA Effect. In Deceitful Media (pp. 50–67). Oxford University Press. https://doi.org/10.1093/oso/9780190080365.003.0004
66. Natale, S. (2021). The ELIZA Effect. In Deceitful Media (pp. 50–67). Oxford University Press. https://doi.org/10.1093/oso/9780190080365.003.0004
67. Neil Bowers. Elisa - Doctor script. Available to https://github.com/neilb/Chatbot-Eliza/blob/master/examples/doctor.txt
68. Rajput, H. Non-Linear Planning in AI. 2023. Online. Available to: https://www.codingninjas.com/studio/library/non-linear-planning-in-ai
69. Randy Baraka. (2023). A Vision for the Future: How Computer Vision is Transforming Robotics. Available to: https://www.comet.com/site/blog/a-vision-for-the-future-how-computer-vision-is-transforming-robotics/
70. Roger Brown. (2023). Computer Vision Technologies in Robotics: State of the Art. Available to: https://www.aiplusinfo.com/blog/computer-vision-technologies-in-robotics-state-of-the-art/
71. Ronkowitz, K. (n.d.). Eliza, a chatbot therapist. New Jersey Institute of Technology. Available to: https://web.njit.edu/~ronkowit/eliza.html
72. Russel, Stuart, a Peter Norvig. Artificial Intelligence: A Modern Approach. 4th vydání. Prentice Hall, 2021.
73. Russell, S., & Norvig, P. (2010) - Artificial Intelligence: A Modern Approach. Available to: https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf
74. Sumathi, D. (2020) STandford Research Institute Problem Solver - STRIPS, Problem solver in Artificial Intelligence. Standford Research Institute. Online. Available to: https://www.youtube.com/watch?v=9UKAkyNpgDE
75. Sutton, Richard S., a Andrew G. Barto. Reinforcement Learning: An Introduction. 2nd vydání. The MIT Press, 2018.

76. Tech, K. (2023). AI in Natural Language Processing Models : Guide to Deployment on Google and API Utilization. Online. Available to: https://katchintech.medium.com/ai-in-natural-language-processing-models-guide-to-deployment-on-google-and-api-utilization-f7ac4efc10e9
77. Tobias Sytsma, Éder M. Sousa. (2023). Artificial Intelligence and the Labor Force: A Data-Driven Approach to Identifying Exposed Occupations. RAND Corporation. https://doi.org/10.7249/rra2655-1
78. Turing, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. In Mind: Vol. LIX (Issue 236, pp. 433–460). Oxford University Press (OUP). https://doi.org/10.1093/mind/lix.236.433
79. Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. In Computational Intelligence and Neuroscience (Vol. 2018, pp. 1–13). Hindawi Limited. https://doi.org/10.1155/2018/7068349
80. Weinberg, Gerald M. (2013). Joseph Weizenbaum. IEEE Computer Society. Available to: https://history.computer.org/pioneers/weizenbaum.html
81. Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent Trends in Deep Learning Based Natural Language Processing [Review Article]. In IEEE Computational Intelligence Magazine (Vol. 13, Issue 3, pp. 55–75). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/mci.2018.2840738. https://arxiv.org/pdf/1708.02709.pdf
82. Zhang, J. AI based Algorithms of Path Planning, Navigation and Control for Mobile Ground Robots and UAVs. 2021. Online. Available to: https://arxiv.org/ftp/arxiv/papers/2110/2110.00910.pdf

🕮 15.1.2

Statement regarding the use of Artificial Intelligence in content creation

This content has been developed with the assistance of artificial intelligence tools, specifically ChatGPT, Gemini, and Notebook LM. These AI technologies were utilized to enhance the text by providing suggestions for rephrasing, improving clarity, and ensuring coherence throughout the material. The integration of these AI tools has enabled a more efficient content creation process while maintaining high standards of quality and accuracy.

The use of AI in this context adheres to all relevant guidelines and ethical considerations associated with the deployment of such technologies. We acknowledge the importance of transparency in the content creation process and aim to provide a clear understanding of how artificial intelligence has contributed to the final product.

Appendix I. Brief AI History

Chapter 16

16.1 1940-1980

📖 16.1.1

1940s-1950s

Artificial intelligence has its roots in the idea that machines could mimic human intelligence. This concept started gaining attention in the 1940s and 1950s with the invention of computers. Alan Turing, a British mathematician, played a critical role during this period. In 1950, he published a paper titled Computing Machinery and Intelligence, where he introduced the Turing Test. This test measured a machine's ability to exhibit intelligent behavior indistinguishable from a human.

In 1956, the term "Artificial Intelligence" was officially coined at the Dartmouth Conference, where researchers like John McCarthy, Marvin Minsky, and others gathered to discuss how machines could simulate human thought. This marked the beginning of AI as a formal field of study.

During this time, early programs demonstrated basic problem-solving and logical reasoning. For example, the "Logic Theorist" program developed by Allen Newell and Herbert A. Simon could prove mathematical theorems. Though primitive by today's standards, these achievements set the foundation for AI development.

📝 16.1.2

Which of the following are milestones in the early history of AI?

- The Turing Test
- The Dartmouth Conference
- The invention of the internet
- The release of smartphones

📖 16.1.3

1943: Neural networks

In 1943, Warren McCulloch and Walter Pitts proposed a revolutionary idea—a mathematical model for neural networks. Their work laid the foundation for how computers could mimic the human brain's processing capabilities. Neural networks are a type of machine learning inspired by the way neurons in the human brain interact to solve problems.

This model used binary (on or off) logic to simulate the firing of neurons. While the concept was ahead of its time, it wouldn't be widely applied until much later, as computing power in the 1940s was very limited. Still, their work became the basis for modern deep learning techniques used in AI today.

16.1.4

What did McCulloch and Pitts introduce in 1943?

- A mathematical model for neural networks
- The Turing Test
- Chatbots
- The concept of big data

16.1.5

1950s-1960s: ELIZA

The late 1950s and 1960s saw some significant progress in AI. Computers were programmed to perform tasks like solving algebraic equations, playing games like chess, and even understanding simple language commands. One notable example was "ELIZA", created in 1966 by Joseph Weizenbaum. ELIZA was a chatbot program that mimicked a psychotherapist by responding to user input with questions and statements. It amazed people, showing how machines could interact using language.

Despite these early successes, challenges soon became apparent. Computers lacked the processing power and memory needed for more complex tasks. Researchers also discovered that many problems were harder to solve than initially thought, such as understanding natural language or recognizing images. Funding for AI research declined during the late 1960s, leading to what is often called the "AI Winter".

16.1.6

What was a major AI program developed in the 1960s?

- ELIZA
- Watson
- Siri
- Google Assistant

16.1.7

1951: First AI program run

In 1951, Christopher Strachey wrote one of the first AI programs on the Ferranti Mark 1 computer. This program allowed the computer to play a simple game of checkers. It demonstrated the ability of machines to make decisions based on logical rules, a core idea in AI.

Although basic, this achievement showed that machines could perform tasks previously thought to require human intelligence. It was a stepping stone toward more complex AI programs that would follow in the decades ahead.

16.1.8

What was the significant achievement of Christopher Strachey in 1951?

- Creating a checkers-playing AI program
- Inventing the computer
- Designing the first robot
- Introducing the term "Artificial Intelligence"

16.1.9

1970s-1980s: Expert systems

AI research regained momentum in the 1970s with the development of "expert systems", These systems were designed to simulate the decision-making abilities of a human expert in specific fields. For example, "MYCIN", developed in the 1970s, helped doctors diagnose bacterial infections and recommend treatments.

During the 1980s, businesses began adopting AI technologies for practical purposes. Expert systems were used in industries such as healthcare, finance, and manufacturing. This success led to increased funding and interest in AI research. However, these systems still had limitations, as they relied heavily on predefined rules and struggled with tasks requiring general knowledge or adaptability.

16.1.10

Which AI advancements gained popularity in the 1970s and 1980s?

- Expert systems
- MYCIN
- Neural networks
- Smartphones

16.1.11

1979: First autonomous vehicle prototype

In 1979, Stanford University introduced the "Stanford Cart", one of the earliest prototypes of autonomous vehicles. The cart could navigate around obstacles in a controlled environment using cameras and basic programming.

While far from the self-driving cars of today, this innovation demonstrated the potential of combining robotics and AI to achieve autonomous navigation. It laid the

groundwork for the autonomous vehicle technology we see being developed today by companies like Tesla and Waymo.

📝 16.1.12

What was notable about the Stanford Cart in 1979?

- It was an early prototype of autonomous vehicles
- It navigated around obstacles autonomously
- It generated natural language text
- It defeated a human in Go

16.2 1980-2000

🕮 16.2.1

1980s-1990s: Machine learning

In the late 1980s and 1990s, AI research shifted focus to machine learning. Unlike expert systems, machine learning allowed computers to learn from data and improve performance over time. This shift was driven by increased computing power and the availability of large datasets.

Neural networks, inspired by the structure of the human brain, became an important tool in machine learning. In 1986, Geoffrey Hinton and others developed algorithms to train neural networks effectively. These algorithms laid the groundwork for many modern AI applications. By the 1990s, AI systems could recognize handwritten digits and play competitive games like chess.

One of the most famous AI achievements during this period was IBM's Deep Blue, which defeated world chess champion Garry Kasparov later, in 1997 demonstrating the power of combining machine learning with advanced computation.

📝 16.2.2

What was a major AI milestone between 1980 and 1990?

- The development of algorithms to train neural networks effectively
- The invention of the microchip
- The creation of smartphones
- The Turing Test

📖 16.2.3

1987-1993: The second AI winter

From 1987 to 1993, AI faced a period of reduced funding and interest known as the "AI Winter". This happened because early AI systems, like expert systems, failed to meet high expectations. They struggled with tasks that required adaptability and understanding of complex real-world problems.

The AI Winter taught researchers important lessons: that computing power, data availability, and realistic expectations were critical for AI progress. It was a setback, but it also led to a more measured and robust approach to AI development in the years that followed.

📝 16.2.4

What caused the second AI Winter between 1987 and 1993?

- AI systems failed to meet high expectations
- AI systems exceeded expectations
- AI became too expensive to develop
- AI defeated world chess champions

📖 16.2.5

1990s: Everyday applications

By the 1990s, AI technologies were being integrated into everyday life. Speech recognition systems, such as those used in early voice assistants, started to appear. AI-powered systems also became crucial for search engines, helping organize and retrieve information efficiently. For example, in 1997, Google introduced its early algorithms, which used AI techniques to improve search results.

Other practical uses of AI included automated customer service, financial fraud detection, and email spam filters. These applications demonstrated that AI could solve real-world problems, paving the way for even more advanced technologies in the 21st century.

📝 16.2.6

Which AI applications emerged in the 1990s?

- Speech recognition
- Search engine algorithms
- Cryptocurrency mining
- Augmented reality glasses

📖 16.2.7

1998: AI in gaming

In 1998, real-time strategy games like StarCraft introduced advanced AI opponents. These AI systems could adapt to players' strategies, offering a more challenging and engaging gaming experience. This represented a leap forward in AI application in entertainment.

By using machine learning techniques, game developers created AI that could make decisions and plan strategies. This milestone influenced AI research and its use in complex decision-making environments.

📝 16.2.8

What made AI in games like StarCraft important in 1998?

- AI opponents could adapt to players' strategies
- It advanced decision-making in entertainment AI
- AI began generating images from text prompts
- AI was first used in medical diagnostics

📝 16.2.9

Order the following AI milestones in the correct chronological order (1940-1999)

- Development of the chatbot ELIZA.
- IBM's Deep Blue defeats world chess champion Garry Kasparov.
- Algorithms for training neural networks become effective.
- Dartmouth Conference officially coins the term "Artificial Intelligence".
- Introduction of expert systems like MYCIN.

16.3 2000-2015

📖 16.3.1

2000-2010: Big data

The early 2000s marked a turning point for AI as the world began generating vast amounts of data, known as "big data“. This term refers to the massive collection of digital information created by people using the internet, smartphones, and other devices. Big data became a powerful resource for training AI systems, as more data meant better learning opportunities.

Around this time, breakthroughs in machine learning helped AI make significant progress. Machine learning allows computers to identify patterns in data and improve their performance without being explicitly programmed. Companies like Google began using these techniques to refine search engines and advertising algorithms. Meanwhile, recommendation systems - like those used by Amazon and Netflix - emerged to suggest products or movies based on user preferences.

AI-powered tools also began appearing in other fields. For example, AI helped doctors analyze medical data to diagnose diseases more accurately. This era set the stage for the rapid growth of AI technologies in the coming decade.

📝 16.3.2

Which advancements in AI occurred between 2000 and 2010?

- The use of big data to train AI systems
- The rise of recommendation systems like Netflix's
- The development of neural networks
- The launch of self-driving cars

🕮 16.3.3

2004-2005: DARPA Grand challenge

The DARPA Grand Challenge was a competition to encourage the development of autonomous vehicles. In 2005, a vehicle named Stanley, created by Stanford University, won the challenge by driving autonomously through a 132-mile desert course.

This milestone spurred significant advancements in self-driving car technology. It marked the transition from research prototypes to practical applications in autonomous transportation.

📝 16.3.4

What was the significance of the DARPA Grand challenge?

- It advanced autonomous vehicle technology
- It developed the first AI chatbot
- It created a game-playing AI
- It introduced the concept of big data

📖 16.3.5

2010-2015: Deep learning revolution

In the early 2010s, a new technique called "deep learning" revolutionized AI. Deep learning is a subset of machine learning that uses neural networks with many layers to analyze complex patterns in data. These deep networks can process images, sound, and text with remarkable accuracy.

One of the biggest breakthroughs in deep learning came in 2012 when a neural network developed by Alex Krizhevsky, Geoffrey Hinton, and others won an image recognition competition by a large margin. This success demonstrated that deep learning could outperform traditional methods in tasks like identifying objects in photos.

Companies quickly adopted deep learning for various applications. For example, it became the backbone of voice recognition systems like Apple's Siri and Google's Assistant. Deep learning also improved machine translation, enabling tools like Google Translate to produce more accurate results.

📝 16.3.6

What was a major breakthrough in AI between 2010 and 2015?

- The success of deep learning in image recognition
- The invention of microprocessors
- The introduction of the Turing Test
- The rise of recommendation systems

📖 16.3.7

2011: Watson wins Jeopardy!

In 2011, IBM's Watson defeated human champions on the quiz show Jeopardy! This demonstrated AI's ability to understand and process natural language in real-time. Watson used advanced algorithms to analyze questions and retrieve relevant answers from vast amounts of data.

This achievement showcased the potential for AI in fields requiring complex decision-making and information retrieval, such as customer service, legal analysis, and healthcare.

📝 16.3.8

What did IBM Watson achieve in 2011?

- Winning the quiz show Jeopardy!
- Defeating a human in chess

- Driving autonomously
- Predicting protein structures

16.4 2015-2020

📖 16.4.1

2015-2018: Virtual assistants

By the mid-2010s, AI technologies became an everyday part of life for many people. Virtual assistants like Siri, Alexa, and Google Assistant could perform tasks like setting reminders, answering questions, and controlling smart home devices. These assistants used natural language processing, a branch of AI that helps machines understand and generate human language.

During this period, AI also advanced in facial recognition technology. Systems capable of identifying individuals from images or videos became widely used in security and social media. For example, Facebook implemented facial recognition to tag people in photos automatically.

AI-powered automation also gained traction in industries like manufacturing, where robots performed tasks with greater precision and efficiency. These advancements highlighted AI's ability to enhance both personal convenience and industrial productivity.

📝 16.4.2

Which technologies became popular between 2015 and 2018?

- Virtual assistants like Alexa and Google Assistant
- Facial recognition for photo tagging
- The invention of the internet
- The creation of recommendation systems

📖 16.4.3

2016-2018: Defeating humans in complex games

One of AI's most famous achievements came in 2016 when Google DeepMind's AlphaGo defeated a world champion in Go, a highly complex board game. Unlike chess, Go has a vast number of possible moves, making it much harder for traditional algorithms to handle. AlphaGo used reinforcement learning, where the AI learned by playing millions of games and improving based on its performance.

In 2017, AlphaGo Zero, an even more advanced version, taught itself to play Go without any human input. It became stronger than any previous AI, showcasing the potential of self-learning systems.

These achievements were not just about games. They demonstrated how AI could tackle problems involving strategy, prediction, and decision-making, with applications in areas like logistics, healthcare, and finance.

📝 16.4.4

What was a major AI milestone between 2016 and 2018?

- AlphaGo defeating a world champion in Go
- The invention of the Turing Test
- The creation of deep learning
- The development of emotion recognition technology

🕮 16.4.5

2018: BERT revolutionizes NLP

In 2018, Google introduced BERT (Bidirectional Encoder Representations from Transformers), a model that greatly improved natural language processing. BERT allowed machines to understand the context of words in a sentence more accurately, leading to better language translation, search results, and AI assistants.

This breakthrough made interactions with AI systems more natural and effective, enhancing applications in chatbots, content creation, and search engines.

📝 16.4.6

What were the impacts of BERT in 2018?

- Improved natural language understanding
- Better AI-powered search results
- Enhanced facial recognition
- Advanced autonomous driving

🕮 16.4.7

2018-2020: Creativity and healthcare

Between 2018 and 2020, AI began showing its creative side. Generative AI, which can create new content, gained attention with tools like GPT (Generative Pre-trained Transformer). These systems could write essays, compose music, and even generate realistic images based on prompts. OpenAI's GPT-3, released in 2020, amazed people with its ability to generate human-like text.

AI also transformed healthcare during this time. It was used to analyze medical images, predict patient outcomes, and even assist in drug discovery. For example, during the COVID-19 pandemic, AI helped researchers model the spread of the virus and develop potential treatments.

These advancements showed that AI was not just a tool for automation or problem-solving but also a source of creativity and innovation.

16.4.8

What were notable AI advancements between 2018 and 2020?

- The rise of generative AI like GPT-3
- AI assisting in medical image analysis
- The invention of smartphones
- The development of the Turing Test

16.4.9

Order the following AI milestones in the correct chronological order (2000-2020):

- Virtual assistants like Alexa and Google Assistant become mainstream.
- Deep learning achieves a breakthrough in image recognition.
- AlphaGo defeats a world champion in Go.
- Big data begins powering AI advancements.
- GPT-3 demonstrates advanced generative AI capabilities.

16.5 2021+

16.5.1

Advancements in AI language models

In 2021, OpenAI introduced DALL-E, an AI model capable of generating images from textual descriptions, showcasing the potential of combining language understanding with image generation. This innovation demonstrated AI's ability to create visual content based on human language inputs.

In November 2022, OpenAI launched ChatGPT, an AI chatbot built on the GPT-3.5 architecture. ChatGPT gained widespread attention for its ability to generate human-like text, engage in conversations, and provide detailed responses across various topics. Its release marked a significant milestone in making advanced AI accessible to the general public.

These developments highlighted the rapid progress in AI language models, enabling machines to understand and generate human language with increasing sophistication.

📝 16.5.2

Which AI model, introduced in 2021, can generate images from textual descriptions?

- DALL-E
- ChatGPT
- AlphaGo
- GPT-3

🕮 16.5.3

Creative arts and image generation

The early 2020s witnessed a surge in AI applications within the creative arts, particularly in image generation. In 2022, models like Midjourney and Stable Diffusion were released, enabling users to create detailed images from text prompts. These tools democratized art creation, allowing individuals without formal training to produce intricate artworks.

In 2023, advancements continued with the release of models capable of generating legible text within images, enhancing the realism and utility of AI-generated visuals. These innovations expanded the boundaries of digital art and design, fostering new forms of expression and creativity.

📝 16.5.4

Which AI models, released in 2022, are known for generating images from text prompts?

- Midjourney
- Stable Diffusion
- GPT-4
- BERT

🕮 16.5.5

Scientific research and healthcare

AI has significantly impacted scientific research and healthcare by enhancing data analysis and predictive modeling. In 2022, AI models were utilized to predict protein structures, aiding in understanding diseases and developing new treatments. This capability accelerated biomedical research by providing insights that were previously time-consuming to obtain.

In 2023, AI-assisted drug discovery became more prevalent, with algorithms identifying potential therapeutic compounds faster than traditional methods. Additionally, AI tools improved diagnostic accuracy by analyzing medical images and patient data, supporting healthcare professionals in making informed decisions.

📝 16.5.6

In which area did AI significantly contribute by predicting protein structures in 2022?

- Biomedical research
- Financial modeling
- Autonomous driving
- Social media analysis

🕮 16.5.7

Ethical considerations and AI governance

As AI technologies advanced, concerns regarding ethical implications and governance emerged. In 2023, discussions intensified around AI-generated content, particularly regarding copyright infringement and the use of unlicensed data for training models. Artists and creators raised issues about their work being used without consent, leading to legal actions and demands for clearer regulations.

In response, organizations and governments began formulating guidelines to ensure responsible AI development. This included promoting transparency in AI systems, ensuring data privacy, and preventing biases in AI decision-making processes. The goal was to balance innovation with ethical responsibility, safeguarding societal interests.

📝 16.5.8

Which concerns have been associated with AI advancements in recent years?

- Copyright infringement
- Data privacy issues
- Improved battery life in smartphones
- Faster internet speeds

🕮 16.5.9

2023: AI in Space exploration

In 2023, AI played a significant role in space exploration, assisting in analyzing vast amounts of data collected from missions. AI-powered systems optimized the operations of Mars rovers, making exploration more efficient and effective.

This milestone demonstrated how AI can operate in extreme environments, solving problems and aiding scientific discovery in ways that would be challenging for humans alone.

16.5.10

How did AI contribute to space exploration in 2023?

- Optimized Mars rover operations
- Analyzed data from space missions
- Improved energy storage in batteries
- Generated realistic human faces

16.5.11

AI integration into daily life

By 2024, AI became deeply integrated into daily life, influencing various sectors such as education, entertainment, and personal productivity. AI-powered personal assistants became more intuitive, managing schedules, providing personalized content recommendations, and even assisting with mental health support.

Looking forward, AI is expected to continue evolving, with potential developments in artificial general intelligence (AGI), where machines could perform any intellectual task that a human can. This prospect raises both exciting possibilities and important considerations regarding the future of work, ethics, and human-AI collaboration.

16.5.12

What does AGI stand for in the context of AI development?

- Artificial General Intelligence
- Automated General Interface
- Advanced Graphical Integration
- Augmented Global Information

16.5.13

Self-driving cars

One of the most prominent applications of AI in recent years has been the development of autonomous vehicles, or self-driving cars. These vehicles use AI to navigate roads, detect obstacles, and make real-time driving decisions without human intervention. Companies like Tesla, Waymo, and others have been at the forefront of this innovation.

Autonomous vehicles rely on multiple AI technologies, including computer vision and machine learning. Computer vision helps the car "see" its surroundings by analyzing

data from cameras and sensors, while machine learning enables the system to predict and respond to changes in traffic conditions. For example, these cars can recognize stop signs, pedestrians, and other vehicles to ensure safe travel.

Despite significant progress, there are challenges. Autonomous vehicles must handle unpredictable situations, such as bad weather or erratic human drivers. Governments and companies are also working on ethical guidelines and safety regulations to ensure these vehicles are deployed responsibly. As AI continues to advance, fully autonomous vehicles may become a regular part of everyday transportation.

16.5.14

Which technologies are crucial for autonomous vehicles?

- Computer vision
- Machine learning
- Text generation algorithms
- Recommendation systems

Appendix II. Do You Remember

Chapter 17

17.1 Do you remember? I.

17.1.1

What is AI?

- Correcting words in a text editor
- Calculator
- Accounting software
- Number plate recognition software

17.1.2

Which of the following is a key technique used in AI?

- Statistical analysis
- Quantum mechanics
- Machine learning algorithms
- Geometric modeling

17.1.3

Who created ELIZA, and where was it developed?

- Alan Turing at Cambridge University
- Joseph Weizenbaum at MIT
- Marvin Minsky at Harvard University
- John McCarthy at Stanford University

17.1.4

What was ELIZA's primary function?

- To solve complex mathematical equations
- To mimic human conversation using natural language processing
- To enhance graphics rendering in video games
- To manage databases efficiently

17.1.5

What significant aspect of human-computer interaction did ELIZA help to advance?

- Development of faster processing speeds in computers
- Understanding and building human-computer dialogues
- Increasing storage capacity of machines
- Enhancing security protocols in computing

17.1.6

What is the primary technology used by AI to process visual information?

- Quantum computing
- Convolutional neural networks
- Blockchain technology
- Basic algorithmic functions

17.1.7

Which of the following applications is an example of visual information processing by AI?

- Weather prediction models
- Data encryption
- Object identification and face recognition
- Financial forecasting

17.1.8

What are examples of advanced NLP models?

- GPT-3 and BERT
- SQL and NoSQL databases
- iOS and Android operating systems
- TensorFlow and PyTorch frameworks

17.1.9

In what applications are AI's speech recognition systems typically used?

- Virtual assistants such as Siri and Google Assistant
- Autonomous driving systems
- Online banking systems
- Mechanical engineering software

17.1.10

How is AI affecting work models?

- It is enabling more flexible work models like home offices
- It is making remote work more challenging and less feasible
- It is decreasing work-life balance for most workers
- It has had no significant impact on work models

17.1.11

What is required from workers as industries adopt more AI and robotics?

- Less involvement in operational processes
- New skills for operating and maintaining advanced AI systems
- Only basic computer knowledge
- Traditional skills without any technological updates

17.1.12

What are some of the broader social impacts of AI on employment?

- Decrease in employment opportunities across all sectors
- Increase in job stability and security for all workers
- Elimination of all employment challenges
- Potential increases in employment inequality and job losses

17.2 Do you remember? II.

17.2.1

What is the primary focus of artificial intelligence (AI)?

- Developing software for data analysis
- Creating machines that perform tasks requiring human intelligence
- Improving efficiency of industrial manufacturing
- Enhancing communication networks

17.2.2

Who is considered the father of AI and introduced the term at the 1956 Dartmouth conference?

- Alan Turing
- Marvin Minsky
- John McCarthy
- Herbert Simon

17.2.3

What does the Turing test evaluate?

- A machine's processing speed
- A machine's ability to mimic human intelligence

- The efficiency of a computer program
- The accuracy of robotic movements

17.2.4

How has AI evolved over the years?

- From basic calculators to advanced gaming systems
- From simple algorithms to complex deep learning models
- From manual data entry to automated data processing
- From wired networks to wireless technology

17.2.5

Which script is the most famous created for ELIZA and what did it simulate?

- "DOCTOR“, simulating a conversation with a psychotherapist
- "LAWYER“, simulating a legal consultation
- "TEACHER“, simulating a classroom discussion
- "ENGINEER“, simulating technical support

17.2.6

What is the "ELIZA effect"?

- The program’s ability to improve itself over time
- The phenomenon where users believe the program understands them
- The system's tendency to malfunction in long conversations
- The program’s capability to translate languages automatically

17.2.7

What best describes Weak AI and Narrow AI?

- AI that possesses capabilities across multiple domains like humans
- AI designed for a specific task within a limited scope
- AI that exceeds human intelligence in all areas
- AI capable of emotional perception and social skills

17.2.8

What defines Superintelligence?

- AI that is slightly smarter than humans in certain tasks
- AI that performs well at data analysis but not in other areas
- AI that exceeds the intelligence of the smartest and most capable humans in all areas

- AI designed to perform only voice or image recognition

17.2.9

How does AI process audio information?

- Through natural language processing (NLP) and speech recognition technologies
- Using computer vision technologies
- By utilizing relational database systems
- Through manual input by users

17.2.10

What is machine learning?

- A method that requires explicit programming to analyze data
- A method that allows AI systems to learn from data without being explicitly programmed
- A network of artificial neurons that mimics the human brain
- A system that adapts to new environments without learning from data

17.2.11

What is a common application of machine learning?

- Developing lightweight web browsers
- Powering traditional mechanical systems
- Creating recommendation systems like those used by Netflix and Amazon
- Enhancing the durability of industrial machines

17.2.12

What distinguishes deep learning from traditional machine learning?

- Deep learning requires less data to function effectively
- Deep learning uses a simpler algorithmic approac
- Deep learning involves artificial neural networks that analyze large data sets
- Deep learning is another term for basic machine learning

17.2.13

What is an adaptive system?

- A system that performs consistently regardless of environmental changes
- A system that adapits to new situations or changes in the environment
- A system that cannot learn from its experiences

- A system that solely depends on pre-written software codes

17.2.14

What role do adaptive systems play in robotics?

- They prevent robots from interacting with humans
- They enable robots to react to unpredictable situations and learn from interactions
- They reduce the efficiency of robots in industrial settings
- They are mainly used for aesthetic purposes in robotics

PRISCILLA

priscilla.fitped.eu