UNIVERZITA
KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED A INFORMATIKY

Vybrané kapitoly z dátovej vedy II

FRANTIŠEK FORGÁČ • LÍVIA KELEBERCOVÁ
VALERII VOLODYMYROVYCH POPOVYCH
KRISTÍNA ŠTEFLOVIČOVÁ

NITRA 2023

Vybrané kapitoly z dátovej vedy II

František Forgáč, Lívia Kelebercová,
Valerii Volodymyrovych Popovych, Kristína Šteflovičová

Univerzita Konštantína Filozofa v Nitre

Vybrané kapitoly z dátovej vedy II

Edícia Prírodovedec č. 809

Autori:

František Forgáč

Lívia Kelebercová

Valerii Volodymyrovych Popovych

Kristína Šteflovičová

Recenzenti:

RNDr. Ján Skalka, PhD.

Mgr. Ľubomír Benko, Ph.D.

prof. RNDr. Daša Munková, PhD.

prof. RNDr. Michal Munk, PhD.

doc. Mgr. Martin Drlík, PhD.

doc. PaedDr. Jozef Kapusta, PhD.

ISBN 978-80-558-2020-0

POĎAKOVANIE

Táto publikácia bola podporovaná:

Agentúrou na podporu výskumu a vývoja (APVV)

na základe Zmluvy č. APVV-18-0473

Klasifikačný model chybovosti strojového prekladu: krok k objektívnejšiemu hodnoteniu kvality prekladu

Projektom ERASMUS+ Programme 2021

KA2, grant number: 2021-1-SK01-KA220-HED-000032095

Future IT Professionals Education in Artificial Intelligence

Predslov

Milí čitatelia,

máte v rukách knihu, ktorá Vám poskytne ucelený prehľad o dátovej vede a jej využití v praxi. Prináša vám základné informácie o dôležitých oblastiach dátovej vedy, ktoré majú významný vplyv na súčasný svet informačných technológií.

Prvá kapitola tejto knihy sa zaoberá spracovaním prirodzeného jazyka, ktoré je kľúčovou oblasťou v rámci dátovej vedy. Dozviete sa, ako sa prostredníctvom algoritmov spracovania prirodzeného jazyka dokáže analyzovať a porozumieť textu v ľudskej reči.

Druhá kapitola sa zameriava na získavanie lingvistických dát, čo je nevyhnutné pre správne fungovanie algoritmov spracovania prirodzeného jazyka. V tejto časti sa dozviete, ako sa získavajú a spracovávajú rôzne druhy jazykových dát.

Ďalšia kapitola knihy je venovaná sumarizácii, čo je proces tvorby stručného zhrnutia veľkého množstva informácií. Kapitola prináša zhrnutie algoritmov strojového učenia dokáže automatizovať proces tvorby súhrnu informácií.

V ostatných kapitolách sa podrobne dozviete o strojovom preklade, neurónových a rekurentných sieťach, klasifikácii a evalvácii a nakoniec o dátovom inžinierstve. Každá z týchto oblastí má svoje špecifiká a prínos pre dátovú vedu.

Táto publikácia je vhodná pre študentov, akademikov a odborníkov z oblasti informatiky, ktorí sa chcú dozvedieť viac o dátovej vede a jej využití v praxi. Veríme, že vás táto kniha poteší a pomôže vám získavať nové poznatky a skúsenosti v oblasti dátovej vedy.

S úctou,

Autori knihy

OBSAH

1 Spracovanie prirodzeného jazyka

1.1 Úvod do problematiky NLP

Prirodzené jazyky sa líšia od tých programovacích jazykov. Pri bežnej komunikácií človeka s človekom nespracovávame náročné matematické operácie. Medzi sebou si dokážeme jednoducho zdieľať informácie, či už hovoreným slovom alebo aj čítaním textu z tohto kurzu. No časom vznikla aj potreba komunikácie človeka s počítačom. Nakoľko výpočtová sila každým rokom rastie, dokázali sme si počítače prispôsobiť rýchlosti spracovania informácie a tak s nimi dokážeme v reálnom čase komunikovať.

Spracovanie prirodzeného jazyka sa zaoberá komunikáciou človeka s počítačom a prebieha pomocou prirodzeného jazyka, napr. slovenský alebo anglický jazyk. Je dôležité povedať, že pri riešení tohto problému sa využívajú poznatky nielen z oblasti informatiky (umelá inteligencia), ale aj jazykovedy či psychológie. Po expertných systémoch sa jedná o najväčšiu aplikáciu umelej inteligencie.

DEFINÍCIA

Spracovanie prirodzeného jazyka – angl. Natural language processing (NLP) je oblasťou výskumu počítačovej vedy a umelej inteligencie, ktorá sa zaoberá spracovaním prirodzených jazykov, ako je angličtina alebo slovenčina. Venuje sa analýze, generovaniu textu alebo hovoreného slova, ktoré si vyžadujú určitú mieru porozumenia prirodzenému jazyku počítačom.

V ďalších kapitolách si pojem Spracovanie prirodzeného jazyka budeme zjednodušovať na NLP.V dnešnej dobe prináša NLP človeku veľké využitie vo výpočtovej technike a zjednodušuje nám tak prácu so zariadeniami. Bežne sa stretávame s určitými úlohami, ako napr.:

- Klasifikácia textu – detekcia spamu, filter slov, ...
- Jazykové modelovanie – chatbot, predpovedanie nasledujúcich slov, generovanie textu, preklad, kontrola pravopisu, ...
- Systém otázok a odpovedí,
- Analýza sentimentu textov (tweets) alebo mienkotvorba,
- Text-to-speech (syntéza reči z textu).

1.2 Metódy v programovaní

1.2.1 Výber vhodnej technológie

Ešte predtým, než sa ponoríme do riešenia niektorých problémov spracovania prirodzeného jazyka, je potrebné si zvoliť správny programovací jazyk a prostredie, v ktorom budeme pracovať.

Pri NLP máme vo voľbe vhodného jazyka dve možnosti, a to Javu a Python.

Java

Javu považujeme za jeden z najpoužívanejších programovacích jazykov súčasnosti. Pri riešení problematiky NLP nám avšak neposkytuje takú flexibilitu, ako jej konkurent Python.

Knižnica OpenNLP sa využíva v Jave pre riešenie problémov s NLP. V tejto oblasti je najpoužívanejšou spolu s knižnicou CoreNLP, ktorá má integráciu aj do jazyka Python.

Python

Už spomínaný Python je dlhodobo vedúcim programovacím jazykom práve pre riešenie NLP problémov. Ponúka jednak skvelú podporu pre integráciu s inými nástrojmi a zároveň je populárny svojou jednoduchosťou a rýchlosťou.

Pre jazyk Python existuje obrovské množstvo pravidelne aktualizovaných knižníc a nástrojov, ktoré si prejdeme podrobnejšie v ďalších kapitolách. Pre Python je navrhnuté aj špeciálne prostredie, ktoré sa vynikajúco hodí na riešenie akéhokoľvek problému - Jupyter.

Jupyter notebook

Užitočným nástrojom pre interaktívny režim Pythonu, ktorý dátovým analytikom uľahčuje prácu, je Jupyter Notebook, kedysi označovaný aj ako IPython Notebook. Je to webová verzia Python konzoly, kde môžeme zadávať príkazy a kontrolovať výstup. Tento flexibilný nástroj pomáha vytvárať čitateľné analýzy, pretože umožňuje zachovať kódy, obrázky, komentáre a vzorce. Jupyter Notebook je pomerne rozšíriteľný aj z dôvodu toho, že sa dá ľahko implementovať na bežnom počítači alebo tiež na takmer ľubovoľnom serveri.

V Jupyter Notebooku sa príkazy a ich výsledky ukladajú. Jednoducho sa k nim môžeme vrátiť, upravovať ich a pridávať komentáre. Pomocou značkovacieho jazyka Markdown je možné medzi príkazy vkladať text, t.j. komentáre, a plynule tak prechádzať od jednotlivých poznámok ku kódu. Celý dokument v Jupyter Notebook je možné zdieľať, a vytvoriť z neho napr. tutoriál, snímky v prezentácii alebo dokonca vedecký článok.

1.2.2 Prehľad knižníc na prácu s NLP

NLP knižnice poskytujú vývojárom pripravené nástroje, ktoré zjednodušujú prácu s textom. Dovoľujú im sústrediť sa na vytvorenie kvalitných modelov pre strojové učenie. V minulosti mohli byť súčasťou projektov, ktoré sa zaoberali spracovaním prirodzeného jazyka iba ozajstní experti, ktorí mali veľmi vysoké znalosti v matematike, lingvistike a v strojovom učení. Vďaka týmto knižniciam sa dnes dokážu presadiť aj začiatočníci v programovaní, nakoľko nám ponúkajú prehľadnú dokumentáciu a rozvinutú komunitu v online priestore.

V tejto kapitole si prejdeme najpoužívanejšie knižnice na prácu s dátami. Obsahujú funkcie, ktoré nám pomáhajú aj pri spracovaní prirodzeného jazyka, to znamená, že ich môžeme využiť aj mimo tejto oblati. Sú to konkrétne:

- nltk
- pandas
- numpy
- spaCy
- stanza
- scikit-learn

NLTK

Azda najpoužívanejšou knižnicou na spracovanie prirodzeného jazyka je určite NLTK (Natural Language ToolKit). Poskytuje nám implementácie takmer všetkých možných problémov NLP. Dokážeme ňou tokenizovať, alebo ju používať aj na zložitejšie problémy ako analýza štruktúry a zmyslu viet. Okrem kompletných algoritmov obsahuje aj voľne dostupné korpusy.

Jednou z nevýhod knižnice NLTK je to, že je relatívne pomalá. Avšak pri riešení jednoduchých úloh je plne dostačujúca.

Príkaz na import knižnice vyzerá nasledovne:

```
import nltk
```

Obrázok 1 import nltk

Bohaté príklady jej využitia nájdeme v ďalších kapitolách.

pandas

Pandas viacúčelová knižnica Python používaná na manipuláciu a indexovanie údajov. Pandas je ďalšia Dá sa použiť na dolovanie webu v spojení s BeautifulSoup. Hlavnou výhodou používania pandas je, že analytici môžu vykonávať celý proces analýzy údajov pomocou jedného jazyka (a vyhnúť sa potrebe prepínať na iné jazyky, ako napríklad v R).

Stručne povedané, čo je možné urobiť v Exceli, dá sa urobiť aj v Pandas. Základná implementácia knižnice Pandas sa realizuje štandardne pomocou príkazu import:

```
1 import pandas
```

Obrázok 2 import pandas

V literatúre často nájdete aj volanie import s priradením aliasu, napr.: import pandas as pd, pričom celá funkcionalita knižnice je v nasledujúcom kóde volaná aliasom pd.

```
1 import pandas as pd
```

Obrázok 3 import pandas s aliasom

Knižnica ponúka základný údajový typ DataFrame, teda tabuľka. Jednotlivé záznamy sú organizované v riadkoch a stĺpcoch.

```
data = {
  "kalorie": [420, 380, 390],
  "cas": [50, 40, 45]
}

# nacitame dataframe do objektu
df = pd.DataFrame(data)

print(df)
```

```
   kalorie  cas
0      420   50
1      380   40
2      390   45
```

Obrázok 4 pandas príklad

Okrem klasických zoznamov môžeme dáta definovať pomocou slovníkov alebo zo súboru csv alebo json. Práve posledný prístup je ten najčastejší.

```
df1 = pd.read_csv('data.csv')

df2 = pd.read_json('data.json')
```

Obrázok 5 Možnosti importu dát s pandas

numpy

Knižnica *numpy* je nástupcom knižnice *Numeric* a je určená na prácu s viacrozmernými poľami. Implementuje polia typov int, float, complex a iné ako triedy, kde položky ukladá v pamäti „za sebou“. Operácie s klasickými štruktúrami poľa sú veľmi pomalé a zato nám numpy umožňuje implementovať operácie nad takýmito poľami oveľa efektívnejšie.

Na ďalších stranách nájdete spôsoby definovania poľa prostredníctvom knižnice numpy.

Ako prvé importujeme knižnicu numpy :

```
1 import numpy as np
```

Obrázok 6 Import numpy

Zadefinujeme si prostredníctvom numpy niekoľko polí. V komentároch máte vysvetlené, ako je jednotlivé pole zadefinované.

```
1  a = np.array([1,2,3,4,5]) # celociselny vektor zo zoznamu
2  b = np.zeros(5) # nulove pole
3  c = np.ones(5) # jednotkove pole
4  d = np.array([[1,2], [3,4]]) # celociselna matica zo zoznamu
5  e = np.zeros((3,4),int) # nulova matica 3x4, celociselna
6  f = np.zeros((3,4),float) # nulova matica 3x4, s realnymi cislami
7
8  print(a)
9  print(b) # defaultne je pole vo float, zato obsahuje bodky
10 print(c) # defaultne je pole vo float, zato obsahuje bodky
11 print(d)
12 print(e)
13 print(f)
```

```
[1 2 3 4 5]
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
[[1 2]
 [3 4]]
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Obrázok 7 Príklad použitia knižníce numpy

V prípade, ak máme pole s príliš veľa údajmi, zobrazí sa vo výstupe nasledovne:

```
g = np.ones((100,100))

print(g)
```

```
[[1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 ...
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]]
```

Obrázok 8 Zobrazenie nadmerného množstva údajov v outpute

Zároveň nám numpy umožnuje veľmi jednoducho použiť už spomínané operácie.

```
# majme zaefinované dve polia

a = np.array([1,2,3])
b = np.array([3,4,5])

# operácie riešime nasledovne

a + b
a - b
```

Obrázok 9 Použitia matematických operácii s numpy

spacy

Knižnica spacy je považovaná za silného konkurenta knižnice NLTK. Podporuje 50 rôznych jazykov vrátane slovenského jazyka. Knižica je charakteristická najmä tým, že sú do nej integrované nástroje na prípravu textu pre strojové učenie. Práve strojové učenie je neoddeliteľnou súčasťou NLP, kde po základnej úprave textu je potrebné spracovaný text natrénovať pomocou umelej inteligencie.

Spacy nám poskytuje množstvo vopred natrénovaných modelov. Napríklad model „en_core_web_sm“ je anglický model obsahujúci všetky základné funkcie. Je trénovaný na základe anglických textov z webov. Načítame ho pomocou funkcie

load(). Nasledujúci príklad zobrazuje predikciu POS značiek, čiže slovných druhou použitím knižnice spacy.

```
1 import spacy
2
3 nlp = spacy.load("en_core_web_sm")
4
5 doc = nlp("My brother Jake likes apples.")
6
7 for token in doc:
8     print(token.text, token.pos_)
```

Obrázok 10 Príklad spacy

stanza

Stanza je balík na analýzu prirodzeného jazyka v Pythone. Obsahuje nástroje pre identifikáciu morfologických znakov viet, analýzu syntaktickej štruktúry, lematizáciu a pod. Súbor nástrojov je navrhnutý pre viac ako 70 jazykov.

Stanza pracuje na základe strojového učenia - neurónovej siete. Táto bola natrénovaná pomocou dostupných anotovaných textov.

Pre prácu s knižnicou je ju potrebné importovať a stiahnuť balíček pre slovenský jazyk.

```
1 import stanza
2 stanza.download('sk')
```

Obrázok 11 Import knižnice stanza

scikit-learn

Skvelá knižnica pre začínajúcich programátorov v umelej inteligencii. Poskytuje nástroje nielen pre oblasť NLP, ale aj pre účely dátovej analýzy a strojového

učenia. Pracuje na knižniciach nižšej úrovne, a to numpy a scipy. Jej využitie je hlavne všeobecné strojové učenie, nakoľko má v rámci hlbokého učenia obmedzenú funkcionalitu. Na to nám slúžia knižnice TensorFlow a Keras. Dokážeme s ňou riešiť problémy ako regresia, klasifikácia, clustering a mnoho ďalších.

1.2.3 Nástroje na získavanie dát z webu

Na dolovanie údajov z webu je nevyhnutná znalosť nejakého programovacieho jazyka. V súčasnosti je najpoužívanejší jazyk Python, v ktorom je implemetovaných niekoľko knižníc pre web scraping.

Beautiful Soup

Beautiful Soup je knižnica jazyka Python na dolovanie údajov zo súborov HTML a XML. Využíva parser a poskytuje idiomatické spôsoby navigácie, vyhľadávania a úpravy parsovaného stromu. Bežne šetrí programátorom hodiny alebo dni práce.

Scrapy

Scrapy je aplikačný framework založený na jazyku Python, ktorý prehľadáva a extrahuje štruktúrované údaje z webu. Bežne sa používa na hĺbkovú analýzu údajov, spracovanie informácií a archiváciu historického obsahu. Okrem webového scrapingu (na ktorý bol špeciálne navrhnutý) ho možno použiť aj ako univerzálny webový indexový prehľadávač alebo na extrahovanie údajov prostredníctvom rozhraní API.

1.3 Predspracovanie textu

1.3.1 Úvod do predspracovania dát

Teraz poznáme niekoľko knižníc, ktoré budeme využívať pri práci s NLP. Využitie niektorých z nich si predstavíme na ukážkach predspracovania textu.

Aby počítač mohol pochopiť prirodzenému jazyku, je potrebné text analyzovať z jazykového hľadiska a následne ho premeniť na užitočnú reprezentáciu.

Jednou zo základných funkcií pre prácu s dátami, ktorou môžeme porovnávať veľkosť pôvodného a referenčného textu, je zistenie počtu slov, znakov a priemernej dĺžka slova v texte. Na toto nám poslúži základné funkcie v jazyku python. Napríklad, na získanie počtu znakov môžeme využiť funkciu *len()*. Pre

počet slov slúži metóda *split()*, ktorá nám reťazec rozdelí na zoznam slov a jeho dĺžku následne získame spomínanou funkciou *len()*.

Predspracovanie textu pozostáva z nasledujúcich častí:

- Odstránenie interpunkcie (Removing punctuations), napr: . , ! $( ) * % @
- Odstránenie stopových slov (Removing Stop words)
- Zmena na malé písmená (Lower-casing)
- Tokenizácia (Tokenization)
- Lemantizácia (Lemmatization)
- Nájdenie koreňu slova (Stemming)
- Určenie POS tagov

1.3.2 Odstránenie interpunkcie

Ide o proces, pri ktorom sa z textu odstránia bodky, čiarky a ďalšie špeciálne znaky (napr. @, #, %).

1.3.3 Odstránenie stopových slov

Stop slová možno bezpečne ignorovať vykonaním vyhľadávania vo vopred definovanom zozname kľúčových slov, uvoľnením priestoru v databáze a skrátením času potrebného na spracovanie. Neexistuje univerzálny zoznam stop slov. Môžu byť vopred vybrané alebo úplne nové. Ich odstránením sa zlepší indexovanie, analýza textu a zníži sa veľkosť dát. Stop slová predstavujú 20 až 30 % počtu všetkých slov v dokumente. V nasledujúcom texte uvádzame zoznam stop slov pre slovenský jazyk.

Typické stop-slová pre slovenčinu:

a, aj, aby, ale, ako, áno, alebo, ani, asi, byť, bez, by, či, cez, do, čo, ešte, dnes, iba, ďalší, je, ho, i, jej, ja, jeho, každý, k, kam, ktorý, kde, mať, kto, môj, ku, na, môcť, my, niet, nad, nie, nový, než, nič, po, o, od, on, prečo, pod, pred, podľa, práve, potom, preto, prvý, pri, s, so, sa, si, svoj, späť, tak, ten, takže, tuto, teda, tento, to, už, toto, z, tu, tvoj, ty, u, v, že váš, viac, však, všetko, vy, za, že ...

V rámci knižnice NLTK je možné používať stop slová pre veľa jazykov. Po importe stop slov je možné zobraziť jazyky, pre ktoré existujú stop slová v NLTK. Nižšie môžeme pozorovať zoznam anglických stop slov prostredníctvom knižnice nltk.

```
import nltk
from nltk.corpus import stopwords

stops = set(stopwords.words('english'))
print(stops)
```

```
{'ours', 'it', 'where', 'more', 'what', 'only',
```

Obrázok 12 Výpis stop slov prostredníctvom knižnice nltk

Následne môžeme s databázou slov pracovať. Vymyslíme si nejaký text a prostredníctvom cyklu *for* odstránime stop slová:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

data = "This is just a sample sentence, showing off the stop words filtration."

stopWords =set(stopwords.words('english'))
words = word_tokenize(data)
wordsFiltered = []

for w in words:
  if w not in stopWords:
    wordsFiltered.append(w)

print(wordsFiltered)
```

```
['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

Obrázok 13 Ukážka odstránenia stop slov

Často sa opakujúce slová v analyzovanom texte niekedy zbytočne utlmujú význam ostatných slov. Napr. pri sémantickej analýze knihy "Dejiny hradov a zámkov Severnej Ameriky" nie je dôležité, koľko krát sa v jednotlivých kapitolách vyskytuje slovo hrad alebo zámok, pretože všetky kapitoly sú vlastne o hradoch, a je predpoklad, že týchto slov tam bude pravdepodobne najviac. Za stop slová preto považujeme veľakrát aj slová, ktoré sa často objavujú vo väčšine textov. Z tohto dôvodu sa často slová hľadajú pomocou početností, t.j. zistí sa početnosť všetkých slov a odstránia sa najčastejšie používané slová.

NEVÝHODY

Odstraňovanie stop slov môže vymazať príslušné informácie a upraviť kontext v danej vete, ktorý je potrebný napríklad pri analýze sentimentu. Za týchto

podmienok môžete zvoliť minimálny zoznam stop slov a pridať ďalšie výrazy v závislosti od konkrétneho cieľa.

1.3.4 Zmena na malé písmená

Prevod slova na malé písmená (NLP -> nlp). Slová ako Kniha a kniha znamenajú to isté, ale keď sa neprevedú na malé písmená, sú tieto dve slová reprezentované ako dve rôzne slová v modeli vektorového priestoru (čo má za následok viac dimenzií). Tento krok nie je potrebné robiť vždy, keď predspracovávame text, pretože pri niektorých problémoch môže menšie písmo viesť k strate informácií

1.3.5 Tokenizácia a segmentácia

Tokenizácia je proces rozdelenia textu na množinu (zoznam) menších častí, tzv. tokenov. Token je vlastne časť celku, a dá sa chápať z pohľadu viet, ako aj slov. Slovo je časť vety a veta je časť celého textu, resp. odseku.

Prvú ukážku zameriame na použitie funkcie **sent_tokenize()** z knižnice nltk, ktorá zo súvislého textu vytvorí tokeny vo forme viet.

```
1 from nltk.tokenize import sent_tokenize
2
3 basen = "Ja sladké túžby, túžby po kráse spievam peknotou nadšený, a v tomto duše mojej ol
  letí, ona mi svety pohýna. Ona mi kýva zo sto životov: No centrom, živlom, nebom, jednotou
4
5 print(sent_tokenize(basen))
```

```
['Ja sladké túžby, túžby po kráse spievam peknotou nadšený, a v tomto duše mojej ohlase sv
```

```
1 # vystup mozeme priradit do premennej
2
3 verse =  sent_tokenize(basen)
4
5 print(verse[2])
```

```
Ona mi kýva zo sto životov: No centrom, živlom, nebom, jednotou krás mojich moja Marína!
```

Obrázok 14 Príklad tokenizácie funkciou sent_tokenize()

Funkcia sent_tokenize() je inštanciou PunktSentenceTokenizer. Táto inštancia je trénovaná a veľmi dobre pracuje pre väčšinu hlavných svetových jazykov. Jej činnosť dokážeme vylepšiť nastavením jazyka, v ktorom budeme tokenizáciu robiť.

Druhou a podľa nás častejšie používanou tokenizáciou, je tokenizácia viet do jednotlivých slov, resp. tokenov. Z hľadiska spracovania textu sa jednotlivé vety skladajú z tokenov a medzier. Token je teda akýkoľvek reťazec znakov, ktorí sa zvyčajne musí nachádzať medzi dvoma medzerami (whitespace). Samotná tokenizácia viet nie je jednoduchý proces. Pri tomto procese je potrebné sa vysporiadať aj s nasledovnými problémami:

- Bodka pri desatinných miestach (napr. číslo 2.06 alebo 2,06). Znak bodky alebo čiarky predstavuje pre tokenizátor bod, kedy je možné oddeliť token. Pri číslach sa však musí rozhodnúť, či bude vhodné číslo rozdeliť.

- Druhým problémom s číslami je ich (pre lepšiu čitateľnosť) „tradičné“ uvádzanie s medzerami (napr. 3 113 416 123). V tomto prípade tokenizátor nesmie na medzery pozerať ako na oddeľovače tokenov.

- Zvládnutie rôznych špeciálnych znakov, veľké a malé písmená, dolný a horný index, dvojbodka, bodka, otáznik, výkričník, úvodzovky, hviezdičky, matematické symboly a iné.

- Problém tzv. spojitých tvarov (napr.: v rámci, na čierno, a tak, Nitriansky kraj, Nitriansky samosprávny kraj a pod.), kde je vhodnejšie chápať spojitý tvar ako celok a nedeliť ho na viacero tokenov.

Pre tokenizáciu viet je vhodné použiť funkciu **word_tokenize()**, ktorá ako môžeme vidieť na nasledujúcom príklade zohľadňuje aj interpunkčné znamienka, s ktorými tiež pracuje ako s tokenmi.

```
1 from nltk.tokenize import word_tokenize
2
3 basen = "Ja sladké túžby, túžby po kráse spievam peknotou nadšený, a v tomto duše mo
  letí, ona mi svety pohýna. Ona mi kýva zo sto životov: No centrom, živlom, nebom, je
4
5 print(word_tokenize(basen))
```

```
['Ja', 'sladké', 'túžby', ',', 'túžby', 'po', 'kráse', 'spievam', 'peknotou', 'nadše
```

Obrázok 15 Príklad tokenizácie s funkciou word_tokenize()

V ďalšom texte už nebudeme rozoberať všetky možnosti tokenizácie. Zvlášť zaujímavé sú funkcie pre tokenizáciu s možnosťou definovania vlastných

regulárnych výrazov, alebo možnosť natrénovania si vlastného tokenizéra pre zohľadnenie špeciálnych charakteristík analyzovaného textu. Podrobnejšie informácie o týchto postupoch nájdete v publikácii.

Pri segmentácii textu rozlišujeme nasledujúce kategórie znakov:

- **Alfanumerické** – sem patria písmená, číslice a podobné znaky. Z technického hľadiska sme si situáciu uľahčili tým, že do tejto kategórie patria všetky znaky s unicode kategóriou L (písmená), M (špeciálne značky, napr. kombinované diakritické značky) alebo N (číslice). Tento prístup je v poriadku, ak uvažujeme o bežne rozšírených jazykoch. Pri tokenizácii zlučujeme alfanumerické znaky medzi dvoma medzerami do jedného celku (tokenu).
- **Zložené slová** – je otázkou, či pri tokenizácii slová so spojovníkom spojiť (napr. „Jean-Claude“, „hi-fi“, „MS-DOS“ a pod.) alebo rozdeliť do viacerých tokenov. Odpoveď nie je jednoznačná – spojenie do jedného tokenu môže byť problematické napr. pri tvaroch typu „Prešov-Košice“, kde bol v texte chybne použitý spojovník namiesto pomlčky. Možným riešením je preskúmať, či sa textové jednotky pred a za spojovníkom nachádzajú v slovníku prípustných tvarov – ak áno, potom treba zložený tvar rozdeliť a zložky reprezentovať ako samostatné tokeny.
- **Interpunkcia (dvojbodka, bodka, otáznik, výkričník na konci vety, úvodzovky, hviezdičky, matematické symboly a iné)** – každý znak tvorí samostatný token. Napr. vo vete „Win98 mi nefunguje!!!“ bude 8 tokenov (úvodzovky dolné, Win98, mi, nefunguje, výkričník, výkričník, výkričník, úvodzovky horné). Väčšinou sú interpunkčné znaky interpretované ako oddeľovače, no niekedy nastáva podobný problém ako pri zložených slovách (napr. „verzia 1.32“, „1.44 MB“, „PS/1“, „abc@email.com“, atď.). Potrebné je nastaviť systém tokenizačných pravidiel tak, aby dokázal identifikovať aspoň základné typy zložených tvarov obsahujúcich interpunkčné tvary – adresy elektronickej pošty a webových stránok, desatinné čísla, a podobne.
- **Veľkosť písma** – znaky sa často v procese tokenizácie konvertujú buď na veľké alebo na malé písmo, čo však môže niekedy znamenať stratu

sémantickej informácie. Naopak, veľké písmená vo všeobecnosti nemajú absolútnu sémantickú platnosť (napr. rozhodne neplatí, že vždy indikujú začiatok vety). Odporúčaným postupom je zachovať informáciu o skutočnom tvare tokenu v texte, vrátane veľkých a malých písmen, ako jeden z parametrov tokenu.

Vo všeobecnosti však pre proces tokenizácie platí, že všetky typy tokenov musia byť identifikovateľné a odlíšiteľné od ostatných.

Softvérové nástroje na segmentáciu a tokenizáciu zahŕňajú pomerne širokú škálu pravidlovo-orientovaných systémov (angl. rule-based systems). V týchto nástrojoch používateľ pomocou formálnej gramatiky definuje pravidlá a ohraničenia na identifikáciu žiadaných tvarov reťazcov v texte. Nástroj potom vygeneruje vykonateľný alebo zdrojový kód, ktorý je priamo použiteľný na segmentáciu a tokenizáciu podľa zadaných pravidiel.

Hotové riešenie na segmentáciu a tokenizáciu anglických textov ponúka vyhľadávací systém Apache Lucene (https://lucene.apache.org/).

Bez ohľadu na použitý nástroj, výsledkom tokenizácie je súbor identifikovaných a ohodnotených tokenov. Pre ďalšie spracovanie býva súbor tokenov najčastejšie vyjadrený vo formáte XML. Token je reprezentovaný pozíciou výskytu v texte, svojim pôvodným tvarom v texte a modifikovaným tvarom. Niekedy sa do XML formátu tokenizovaného textu pridávajú aj údaje o začiatku a konci vety.

V neskoršom procese lematizácie a morfologickej analýzy sa tokenu priraďujú atribúty lema a tag.

Proces tokenizácie môže byť obzvlášť problematický pri práci s doménami biomedicínskeho textu, ktoré obsahujú veľa pomlčiek, zátvoriek a iných interpunkčných znamienok.

1.3.6 Lematizácia

Lematizácia je proces zmeny slova na základný tvar. Predstavuje jednu z možností normovania slov v jazyku. V slovenčine existujú určité konvencie pri vytváraní lemy, napr. slovesá sú vždy v neurčitku, podstatné mená v nominatíve jednotného čísla, prídavné mená v predikatívnej pozícii a pod.

Príklad lematizéru prostredníctvom knižnice nltk:

```
import nltk
from nltk.stem import WordNetLemmatizer

wordnet_lemmatizer = WordNetLemmatizer()
text = "studies study student"

tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print("{}, {}".format(w, wordnet_lemmatizer.lemmatize(w)))
```

```
studies, study
study, study
student, student
```

Obrázok 16 Príklad lematizéru prostredníctvom knižnice nltk

Príklad lematizéru prostredníctvom knižnice spacy:

```
import spacy

nlp = spacy.load('en_core_web_sm')

sentence = ("Lemmatization is the process of converting a word to its base form.")

doc = nlp(sentence)

for token in doc:
    print(token, token.lemma_)
```

Obrázok 17 Príklad lematizéru prostredníctvom knižnice spacy

Lematizáciu možno vykonávať pomocou:

- **slovníka koreňov** – výhodou tejto metódy je minimálna chybovosť, nevýhodou rozsiahlosť slovníka a jeho prípadné obmedzenie na špecifický odbor.
- **odstránením afixov, tzn. sufixov (prípon) a prefixov (predpôn)** – ide o najčastejšie používanú metódu. Afixy môžu byť odstraňované na základe zoznamu sufixov a prefixov alebo na základe pravidiel, podľa ktorých sú konkrétne afixy generované.
- **štatisticky** – na základe rôznorodosti po sebe nasledujúcich písmen v slove, kde sa pomocou frekvencie jednotlivých zhlukov písmen stanovuje, či ide o prefix, koreň alebo sufix slova. Táto metóda je nezávislá na jazyku.

1.3.7 Stemming

Týka sa procesu odstránenia konca (postfix, napr. full v slove powerfull) alebo začiatku slov (prefix, napr. predpona astro v slove astrobiologia) s úmyslom odstrániť prípony (lexikálne doplnky ku koreňu slova). Tento postup bol účinný pre vyhľadávače, ktoré využívajú zhlukovanie na získanie relevantnejších výsledkov. Pomocou stemmingu je možné nájsť viac zhody, pretože slovo má širší význam, a dokonca sa dajú spracovať aj také veci, ako sú pravopisné chyby. V aplikáciách umelej inteligencie môže pomôcť zlepšiť celkové porozumenie. Existuje viacero algoritmov na tvorbu slov, z ktorých mnohé sú pomerne jednoduché.

Príklad stemmovania pri rôznych tvaroch slova „close“ (zatvoriť):

{closing, closes, closer, close} → clos

Porter Stemmer

Porterov algoritmus napríklad uvádza, že slovo "univerzálny" má rovnaký základ ako slová "univerzita" a "univerzity", čo môže mať historický základ, ale už nie je sémanticky relevantné. Porterov kmeňový algoritmus tiež neuznáva, že anglické slová "theater" a "theatre" by mali patriť do tej istej kmeňovej triedy. Z týchto dôvodov Watson Explorer Engine nepoužíva Porterov stemmer ako svoj anglický stemmer.

Porterov algoritmus je založený na metóde odstraňovania sufixov (angl. suffix stripping) s využitím vopred definovaného slovníka sufixov a niektorých pravidiel morfológie anglického jazyka. Pravidlá sú uvádzané v tvare:

(podmienka – logický výraz) slovo1 → slovo2

Podmienková časť operácií Porterovho algoritmu môže obsahovať nasledujúce premenné:

- c – spoluhlásky
- v – samohlásky
- m – miera slova (angl. Measure of a Word) – počet samohláskových skupín slova
- *S – slovo končiace na spoluhlásku S (rovnako pre ostatné písmená)
- *v* - slovo obsahujúce samohlásku

- *d – slovo končiace zdvojenou spoluhláskou (napr. –ss)
- *o – slovo končiace cvc, teda spoluhláskou, samohláskou a spoluhláskou, pričom druhá spoluhláska nie je w, x alebo y.

V skutočnosti spoločnosť IBM vytvorila vlastný proprietárny stemmer, ktorý umožňuje značné prispôsobenie. Prípony môžu vytvárať alebo rozširovať nové tvary toho istého slova (nazývané inflexné prípony), alebo dokonca vytvárať samotné nové slová (nazývané odvodzovacie prípony). Jedným z riešení je tieto preddefinované metódy upraviť pridaním alebo odstránením prípon a pravidiel, musíme však vziať do úvahy, že môžete zlepšiť výkon v jednej oblasti a spôsobiť degradáciu v inej oblasti, napríklad vo výkone.

Ďalšie nástroje sú napríklad Snowball stemmer, Lancaster stemmer, Lovinsov stemmer a ďalšie.

Snowball: https://snowballstem.org/

Lancaster: https://www.nltk.org/_modules/nltk/stem/lancaster.html

V slovenčine a ďalších flektívnych jazykoch je situácia komplikovanejšia, pravidlá morfológie sú podstatne zložitejšie a izolácia koreňa sa rieši morfologickou analýzou a komplexným lingvistickým prístupom.

Problémy, s ktorými je potrebné sa vysporiadať pri lematizácii v slovenčine, spôsobujú časté nejednoznačnosti a rôzne typy nepravidelností jazyka, predovšetkým tvarová homonymia. Napríklad token mier nemá jednoznačnú lemu – môže to byť sloveso mieriť, aj substantíva mier a miera. Je potrebné zistiť okolie (kontext) tokenu, odvodiť slovný druh a morfologické kategórie tokenu, a až potom sa dá presne určiť lema.

Na lematizáciu a morfologickú analýzu textov v slovenčine bolo vytvorených niekoľko nástrojov, napríklad webové rozhranie Jazykovedného ústavu Ľ. Štúra SAV Bratislava s názvom Lematizácia, morfologická anotácia a dezambiguácia (https://morphodita.juls.savba.sk/). Analýza textu je založená na morfologickej databáze, ktorá obsahuje 111-tisíc lem; 3,6 milióna záznamov; 1,3 milióna jedinečných slovných tvarov. Samotná analýza a dezambiguácia používa softvér MorphoDita vrátane jednoduchého štatisticko-heuristického guessera, používaného na odhad možnej lemy a gramatických kategórii slova, ktoré sa nenachádza v morfologickej databáze.

1.3.8 Označovanie slovných druhov (POS tagging)

Morfologická anotácia je základnou (a najčastejšou) lingvistickou informáciou vnášanou najmä do flektívnych jazykov. Obsahuje slovnodruhové a tvarové charakteristiky slov v kontexte. Zvyčajne jej predchádza lematizácia – priradenie základného (slovníkového) tvaru každému slovu.

Výsledkom morfologickej analýzy je priradenie tzv. tagov k jednotlivým slovám. Z tohto dôvodu sa hovorí o tzv. anotovaní (slangovo "tagovaní"), t.j. priradeniu krátkej charakteristiky slova v rámci vety. Dobrou pomôckou pri určovaní morfologických tvarov slov je projekt Slovenského národného korpusu - https://korpus.sk/morpho.html. Na tejto adrese nájdete úplný význam jednotlivých častí tagov.

Úloha: Určte morfologické značky jednotlivých slov vo vete: "Moja mama vždy hovorila, že zázraky sa dejú každý deň".

Pre vyriešenie uvedenej úlohy použijeme knižnicu stanza. Už známou funkciou nlp vetu analyzujeme.

```
nlp = stanza.Pipeline(lang='sk')
doc = nlp("Moja mama vždy hovorila, že zázraky sa dejú každý deň.")
```

Okrem lematizácie príkaz vykoná aj morfologickú analýzu. Jej výsledky stačí iba „prečítať". Samotné slovo a lema sa nachádzajú vo vlastnostiach text a lema. Nové vlastnosti, ktoré využijeme sú:

- **upos** (prípadne iba **pos**) - tzv. univerzálna značka (universal part of speech tag),
- **xpos** – špecifická značka pre daný jazyk
- **feats** zoznam morfologických vlastnosti pre špecifický jazyk, t.j "bližšie" vlastnosti ako napr. rod, pád, číslo, atď.

Pre určenie morfologických značiek použijeme vlastnosť xpos.

```
for veta in doc.sentences:
  for slovo in veta.words:
    print(slovo.text + ' ( ' + slovo.xpos + ' ) ')
```

Výsledkom morfologickej anotácie sú značky pre jednotlivé slová. Napr. pre slovo „mama“ bola identifikovaná značka „SSfs1“. Podľa https://korpus.sk/morpho.html môžeme z tejto značky zistiť nasledovné:

Substantívum

Pozícia	Znak	Hodnota	Príklad
1. slovný druh	S	substantívum	slovo, ryba, ústav, muž
2. paradigma	S	substantívna	chlap, žena, srdce
	A	adjektívna	hlavný, vedúci, Mastný, starká, Slaná, vstupné
	F	zmiešaná	kuli, gazdiná
	U	neúplná	kanoe, kupé
3. rod	m	mužský životný	hrdina, hlavný, Mastný
	i	mužský neživotný	strom, rýľ
	f	ženský	ulica, pani, vedúca, Slaná, hradská
	n	stredný	mesto, vysvedčenie, dievča, mláďa
4. číslo	s	jednotné	slovo, ryba, ústav, muž
	p	množné	slová, ryby, ústavy, muži/mužovia
5. pád	1	nominatív	pán, vedúci, matka, Slaná, more, mláďa
	2	genitív	pána, vedúceho, matky, Slanej, mora, mláďaťa
	3	datív	pánovi, vedúcemu, matke, Slanej, moru, mláďaťu
	4	akuzatív	pána, vedúceho, matku, Slanú, more, mláďa
	5	vokatív	pane, mami, Táni, oci
	6	lokál	pánovi, mame, Slanej, mori, mláďati
	7	inštrumentál	pánom, vedúcim, matkou, Slanou, morom, mláďaťom

Obrázok 18 Značky slovných druhov v slovenčine

Prvá pozícia v značke „S“ označuje slovný druh „substantívum“, t.j. podstatné meno. Tretia pozícia „f“ hovorí o ženskom rode, štvrtá „s“ – jednotné číslo, piata pozícia označuje 1. pád. Sumárne teda značka „SSfs1“ hovorí o identifikovanom podstatnom mene, v substantívnej forme, ženského rodu, jednotného čísla v prvom páde – nominatív.

Podobne napr. pre slovo „hovorila“ bola identifikovaná značka „VLescf+“. Táto identifikovala slovo „hovorila“ ako sloveso (verbum), v tz

v. l-ovom príčastí, nedokonavom vide, v jednotnom čísle, tretej osobe, ženskej kongruencie v rode s tzv. afirmáciou.

1.3.9 N-gramy

Za N-gram považujeme skupinu prvkov, zvyčajne slov, ktorá je spojená do jedného celku. Jej veľkosť definujeme písmenom N. Doteraz sme pracovali s unigramom, t.z. iba s jedným tokenom. Pod pojmom bigram rozumieme rozdelenie na prvky do skupín po dvoch prvkoch. Trigramy majú veľkosť troch prvkov a takto pokračujeme až k N-gramom.

```
import nltk

text = "Python je interpretovaný, interaktívny programovací jazyk, ktorý vyt
je často porovnávaný s jazykmi Tcl, Perl, Scheme, Java a Ruby. Python je vyv

tokens = nltk.word_tokenize(text)
```

```
bigram = list(nltk.bigrams(tokens))
trigram = list(nltk.trigrams(tokens))
ngram = list(nltk.ngrams(tokens, 4)) # urcime s N = 4
```

```
print(bigram)
# print(trigram)
# print(ngram)
```

```
[('Python', 'je'), ('je', 'interpretovaný'), ('interpretovaný', ','), (',',
[('Python', 'je', 'interpretovaný'), ('je', 'interpretovaný', ','), ('interp
```

Obrázok 19 Príklad n-gramov

N-gramy považujeme za veľmi pomocný mechanizmus pri identifikovaní entít. Frekvencia výskytu určitých N-gramov v dokumentoch je oveľa väčšia pri známych viacslovných entitách. Ak si zoberieme bi-gramy (New, York) a úplne náhodný bi-gram ako napríklad (walked, him), tak je zrejmé že frekvencia bi-gramu (New, York) bude v dokumente väčšia. N-gramy sú užitočným nástrojom aj pri predikcii ďalších slov vo vete.

Jazykový model môže vypočítať pravdepodobnosť, že dané slovo bude nasledovať po postupnosti predchádzajúcich slov. Určenie pravdepodobnosti dlhej postupnosti slov w (w1, ..., wm) je zvyčajne neuskutočniteľné. To znamená, že príliš dlhý n-gram (n = 10) sa nepoužíva, lebo v celom datasete by sa mohol vyskytovať iba jeden takýto prípad.

Výpočet spojitej pravdepodobnosti P(w1, ... , wm) by sa uskutočnil použitím nasledujúceho reťazového pravidla:

$$P(w_1, \ldots, w_m) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_m|w_1, \ldots, w_{m-1})$$

Pravdepodobnosť nasledujúcich slov vzhľadom na predchádzajúce slová je obzvlášť ťažké odhadnúť z dostupných údajov. Preto sa spojitá pravdepodobnosť zvyčajne aproximuje pomocou predpokladu nezávislosti, t. j. i-te slovo závisí len od n-1 predchádzajúcich slov. Budeme modelovať len spoločné pravdepodobnosti kombinácií n po sebe nasledujúcich slov, tzv. n- gramov. Napríklad vo vete "Toto jedlo je dobré" máme nasledujúce n-gramy:

- 1-gram (unigram): "Toto", "jedlo", "je" a "dobré"
- 2-gramy (bigram): "Toto jedlo", "jedlo je" a "je dobré"
- 3-gramy (trigram): "Toto jedlo je" a "jedlo je dobré"
- 4-gramy: "Toto jedlo je dobré"

Odvodenie spojitého rozdelenia sa aproximuje pomocou n-gramových modelov, ktoré rozdeľujú spojité rozdelenie na viacero nezávislých častí. Termín n-gramy možno použiť na označenie iných typov sekvencií dĺžky n, napríklad n znakov.

Ak máme veľký korpus, môžeme nájsť všetky n-gramy až do určitého n (zvyčajne 2 až 4) a spočítať výskyt každého n-gramu v tomto korpuse. Na základe týchto počtov môžeme odhadnúť pravdepodobnosť posledného slova každého n-gramu vzhľadom na predchádzajúcich n-1 slov:

- **1-gram**: $P(word) = \frac{count(word)}{\text{total number of words in corpus}}$
- **2-gram**: $P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$
- **N-gram**: $P(w_{n+i}|w_n, \ldots, w_{n+i-1}) = \frac{count(w_n, \ldots, w_{n+i-1}, w_{n+i})}{count(w_n, \ldots, w_{n+i-1})}$

(27)

Predpoklad nezávislosti, že i-te slovo závisí len od predchádzajúcich n-1 slov možno teraz použiť na aproximáciu spoločného rozdelenia.

Napríklad pre unigram môžeme aproximovať spoločné rozdelenie pomocou nasledujúceho vzorca:

$$P(w_1, \ldots, w_m) = P(w_1)P(w_2)P(w_3)\ldots P(w_m)$$

(28)

Pre trigram môžeme aproximovať spoločné rozdelenie pomocou nasledujúceho vzorca:

$$P(w_1, \ldots, w_m) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_m|w_{m-2}, w_{m-1})$$

(29)

Keď sa zvyšuje počet možných vstupných premenných (slov), exponenciálne sa zvyšuje počet rôznych kombinácií týchto vstupných hodnôt. Čím väčšie je naše n, tým lepšie dokážeme aproximovať pôvodné rozdelenie a tým viac údajov by sme potrebovali na dobrý odhad n-gramu pravdepodobnosti.

1.4 Dáta

Všetky spomenuté možnosti predspracovania dát sa využívajú hlavne pri väčších rozmeroch textu. To, čo potrebujeme, je ucelená množina súvisiacich dát, ktorú nazývame dataset. Pri jeho správnom výbere musíme brať ohľad v jeho relevantnosť na náš problém – PRÍKLAD: pokiaľ chceme z dostupných dát analyzovať sentiment, dataset so značkami áut nebude pre tento problém vhodným riešením. Okrem vhodnej kvantity dát je dôležitá aj ich kvalita, hlavne z dôvodu lepšej úspešnosti použitého algoritmu, zároveň nestrácame čas s jeho očistením.

V tematike NLP často môžeme stretnúť aj s pojmami ako slovník alebo korpus. Rozdiel medzi nimi je ten, že slovník je zoznam slov (vo väčšine prípadov zoradený abecedne), ku ktorým je podané vysvetlenie. Jeho nevýhoda spočíva hlavne vo vynechaní veľkého množstva slov a tak tak môže dôjsť k zámene kontextu slov. Korpus je štruktúrovaný, unifikovaný a veľmi rozsiahly zdroj jazykových dát. Vďaka korpusom môžeme napr. pozorovať vývoj jazyka. K zarovnaniu korpusov sa bližšie venujeme v kapitole 2.3.

1.4.1 WordNet

Jednou z veľmi populárnych lexikálnych databáz je WordNet. Ide o slovník slov, kde dané slová usporiadané podľa sémantických vzťahov. Knižnica nltk nám ponúka zaujímavý nástroj pre zisťovanie týchto vzťahov. Slovník wordnet je potrebné pred jeho používaním stiahnuť pomocou metódy download().

```
1 import nltk
2
3 nltk.download('wordnet')
4
5 from nltk.corpus import wordnet
```

```
[nltk_data] Downloading package wordnet to /home/johny/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Obrázok 20 Ukážka importu Wordnetu do nášho prostredia

Pomocou slovníka wordnet je možné zistiť význam slov, synonymá a antonymá slov. Pri práci so slovníkom budeme používať tzv. synset. Je to jednoduchá možnosť, ktorá sa v nltk používa na čítanie vzťahov. Inštancie synset sú zoskupenia synonymických slov, ktoré vyjadrujú rovnaký koncept. Niektoré slová majú iba jeden synset a niektoré majú niekoľko. Skúsme nájsť vo wordnete slovo „joy“.

```
1 syns = wordnet.synsets("joy")
2
3 print(syns)
```

```
[Synset('joy.n.01'), Synset('joy.n.02'), Synset('rejoice.v.01'), Synset('gladden.v.01')]
```

Obrázok 21 Synsety

Podľa výsledku hľadania sa slovo „joy“ nachádza v štyroch synsetoch. Každý synset má názov, ktorý končí číslom synsetu. V našom výsledku je medzi slovom a číslom ešte znak „n“ alebo „v“. Znak „n“ označuje podstatné meno, „v“ zasa sloveso.

Ak chceme vidieť slová, ktorá majú sémantický, t.j. významový vzťah k „joy“ (v preklade “radosť”), pomocou metódy lemmas() ich môžeme vypísať.

```
for syn in syns[0].lemmas():
  print(syn.name())
```

```
joy
joyousness
joyfulness
```

```
for syn in syns[1].lemmas():
  print(syn.name())
```

```
joy
delight
pleasure
```

Obrázok 22 Výpis synsetov 1

Všimnime si, že prvý synset obsahuje tri slová, druhý síce rovnaké množstvo, ale niektoré slová sú iné. Často sa stáva, že synset obsahuje iba jedno slovo. Výpis slov všetkých synsetov, kde sa nachádza slovo „joy“ môžeme realizovať nasledovne.

```
for syn in syns:
  for lema in syn.lemmas():
    print(lema.name())
```

```
joy
joyousness
joyfulness
joy
delight
pleasure
rejoice
joy
gladden
joy
```

Obrázok 23Výpis synsetov 2

V synsetoch sú uložené aj ďalšie informácie. Napríklad, väčšina synsetov obsahuje aj definíciu skúmaného slova.

```
1 print(syns[0].definition())
```

```
the emotion of great happiness
```

Obrázok 24 Výpis definície synsetu

Zaujímavé je, že slovo „joy“ má rozdielnu definíciu v prvom synsete a rozdielnu v druhom. Je to bežné, pretože napr. slovo „table“ môže byť v synsete so slovami označujúcimi nábytok, ale aj v synsete slov označujúcich štruktúru údajov - tabuľku.

Ďalšou možnosťou je zobrazenie príkladov použitia slova v synsete. Tento príklad podobne ako definícia nemusí byť súčasťou každého synsetu.

```
1 print(syns[0].examples())
2 print(syns[1].examples())
```

```
[]
['a joy to behold', 'the pleasure of his company', 'the new car is a delight']
```

Obrázok 25 Príklady použitia slova v synsete

Okrem podobných slov je možné nájsť v synsete aj tzv. antonymá. Tieto označujú kontrastné pojmy (opozitá) k danému slovu, t.j. slová s protikladným významom. Zaujímavosťou je, že pokiaľ pre skúmané slovo vieme nájsť niekedy celý zoznam synoným, antonymá sa nachádzajú iba vo dvojiciach, t.j. slovo a jeho antonymum. V nasledujúcej ukážke uvádzame výpis všetkých synoným pre slovo „joy“ spolu s antonymom týchto synoným (ak existujú).

```python
from nltk.corpus import wordnet

slovo = "joy"
synonyma = []
antonyma = []

for syn in wordnet.synsets(slovo):
  for lema in syn.lemmas():
    synonyma.append(lema.name())
    if lema.antonyms():
      antonyma.append(lema.antonyms()[0].name())

print("Synonymá:")
print(set(synonyma))
print("Antonymá:")
print(set(antonyma))
```

```
Synonymá:
{'joyousness', 'delight', 'rejoice', 'pleasure', 'gladden', 'joy', 'joyfulness'}
Antonymá:
{'sorrow', 'sadden'}
```

Obrázok 26 Výpis synonym a antoným

Zaujímavou funkcionalitou knižnice nltk je zisťovanie tzv. významovej podobnosti slov. Pre dve slová vieme zistiť ich úroveň podobnosti, táto metrika je počítaná pomocou vyhľadávania počtov spoločných synsetov, synsetov synoným atď. Najväčšia miera podobnosti je 1, významovo rozdielne slová majú podobnosť blízku 0.

```
w1 = wordnet.synset('joy.n.01')
w2 = wordnet.synset('joyousness.n.01')

print(w1.wup_similarity(w2))
```

```
1.0
```

```
w1 = wordnet.synset('joy.n.01')
w2 = wordnet.synset('sorrow.n.01')

print(w1.wup_similarity(w2))
```

```
0.7142857142857143
```

```
w1 = wordnet.synset('joy.n.01')
w2 = wordnet.synset('mouse.n.01')

print(w1.wup_similarity(w2))
```

```
0.1
```

Obrázok 27 Podobnosť slov prostredníctvom synsetov 1

V ukážke uvádzame výpočet významovej podobnosti slov „joy“ a „joyousness“ (nie je to presne podobnosť slov ale skôr podobnosť synsetov, v ktorých sa slová nachádzajú). Vzhľadom k tomu, že sa jedná o synonymá, ich podobnosť je rovná 1. Slovo “sorrow” (smútok) má tiež pomerne veľkú významovú blízkosť. Je to vlastne antonymum slova “joy”. Pre kontrolu uvádzame v poslednom príklade podobnosť slova “joy” a “mouse” (myš), ktoré nie sú vôbec významovo blízke. Hodnota ich významovej blízkosti je veľmi malá.

Na záver uvádzame “dôkaz”, že “človeku spôsobí väčšiu radosť futbal ako šach”.

```
w1 = wordnet.synset('joy.n.01')
w2 = wordnet.synset('football.n.01')

print(w1.wup_similarity(w2))
```

```
0.23529411764705882
```

```
w1 = wordnet.synset('joy.n.01')
w2 = wordnet.synset('chess.n.01')

print(w1.wup_similarity(w2))
```

```
0.1
```

Obrázok 28 Podobnosť slov prostredníctvom synsetov 2

WordNet je v rámci knižnice nltk vynikajúce spracovaný pre angličtinu. Slovenský WordNet je zatiaľ spracovaný vo forme súboru obsahujúceho: číslo synsetu, slovný druh, slová obsahujúce daný synset (sú oddelené bodkočiarkou) a bližšie vysvetlenie synsetu. Slovenský WordNet je možné nájsť na https://korpus.sk/WordNet.html

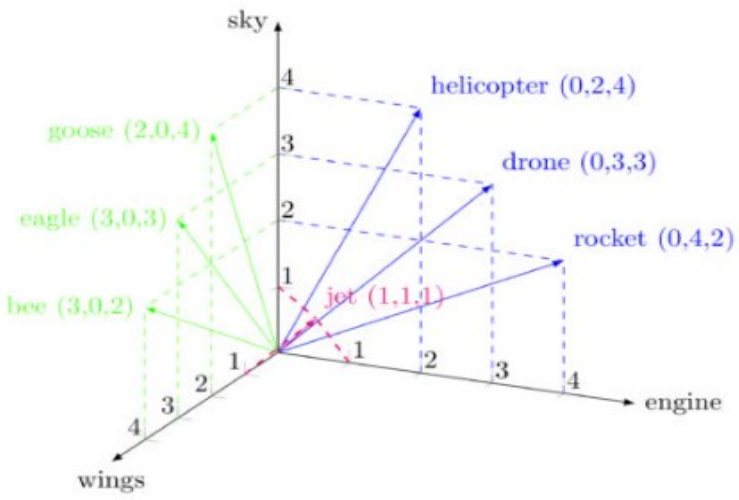
1.5 Vektorová reprezentácia textu

1.5.1 Úvod do vektorových modelov

Aby bol počítač schopný pochopiť textu, musíme daný text premeniť na čísla. Tomuto procesu hovoríme **word embedding**, prekl. vnorenie slov.

Vektor slova

Aby významu slov v prirodzenom jazyku čo najlepšie porozumel počítač, je nutné slová určitým spôsobom reprezentovať do nejakej vodnej podoby. Najbežnejším spôsobom strojovej reprezentácie slova je niekoľkorozmerný číselný vektor. Ten sa následne používa ako vstup do zložitejších modelov strojového učenia - porozumenie významu textu, poznávanie hlasu, strojový preklad a podobne. Príklad, ako sa môže takýto slovný vektor zobrazovať:



Obrázok 29 Zobrazenie slovného vektora

Najjednoduchším spôsobom ako strojovo reprezentovať text zaznačením počtu výskytov jednotlivých slov v texte do vektoru. Túto metódu nazývame **aj Bag of words** (Batoh slov).

1.5.2 Bag of Words (Batoh slov)

Je jeden z najjednoduchšch modelov, ktoré existujú. Jeho cieľom je započítať, ako často sa v texte nejaké slovo vyskytuje, nerieši slovosled a zoženie vety.

Bag-of-words je jednoduchý model na pochopenie, implementáciu a ponúka nám veľkú flexibilitu prispôsobovania špecifických textových dát. Napriek tomu má niekoľko nedostatkov:

- **Veľkosť vektora** – pri rozsiahlych slovníkoch môže mať obrovskú veľkosť
- **Významovosť** – nezohľadňuje kontext a význam slov vo vetách. Takže slovo, ktoré sa v texte vyskytuje často, bude mať najväčšiu váhu, hoci nejde o slovo, ktoré vystihuje celý dokument.

Práve z toho dôvodu sa budeme ďalej zaoberať modelmi zamerané na vektorovú reprezentáciu slov.

Nižšie môžete vidieť, ako algoritmus BoW pracuje. Pre zlepšenie algoritmu sme využili aj kroky predspracovania textu ako tokenizácia a odstránenie stop slov.

```
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer

docs = ["Machine learning uses historical data to predict output values.",
             "It is seen as a part of artificial intelligence.",
             "Machine learning programs can perform tasks without being explicit
```

```
['Machine learning uses historical data to predict output values.', 'It is seen a
```

```
def preprocess(document):
    # vsetky pismena nastavime na male
    document = document.lower()
    # tokenizuj do slov
    words = word_tokenize(document)
    # vymaz stop slova
    words = [word for word in words if word not in stopwords.words("english")]
    # spoj slova, nech su naspat vety
    document = " ".join(words)

    return document

docs = [preprocess(doc) for doc in docs]
```

```
vectorizer = CountVectorizer()
bow_model = vectorizer.fit_transform(docs)
print(bow_model.toarray())
```

```
[[0 1 0 1 0 1 1 1 0 0 1 0 0 0 0 1 1 0]
 [1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0]
 [0 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1]]
```

Obrázok 30 Príklad algoritmu BoW

1.5.3 Modely reprezentácie textových dokumentov

Využívame ich vtedy, keď chceme text efektívne spracovať prostredníctvom metód text miningu. Vďaka týmto modelom môžeme dokumenty formalizovať, najmä algoritmami klasifikácie a zhlukovania. Pri textových dokumentoch poznáme dva druhy prístupov k dátam:

· údaje so závislosťami - prístup sa využíva v prípadoch, kedy je potrebné poznať ako slová vo vete za sebou nasledujú, t.j. kde presne vo vete sa nachádzajú. Takýto prístup sa používa pri skúmaní gramatických pravidiel jazyka, štýlu jazyka a pod. Väčšinou sa však jedná o výpočtovo a pamäťovo náročné operácie.

· údaje bez závislostí - Vo väčšine text miningových problémov však nie je potrebné vedieť, ako slová za sebou nasledujú. Postačí, ak vieme informáciu, že slovo sa v danej vete nachádza (a nie je dôležité v ktorej časti vety). Týmto prístupom dokážeme významne zredukovať pamäťovú náročnosť riešeného problému.

Najrozšírenejšou metódou analýzy textu je pravdepodobne vyhľadávanie dokumentov. Vzhľadom na skutočnosť, že sa často jedná o rozsiahle súbory, porovnávanie celých dokumentov, odsekov alebo viet nie je vhodné. V praxi sa preto začali používať tzv. modeli dokumentov, ktoré reprezentujú zastúpenie slov v dokumente.

Boolovský model

Najjednoduchším modelom je tzv. boolovský model. Už z jeho názvu vyplýva, že využíva operátory Boolovskej logiky, ktoré poznáme ako AND a NOT, resp. 1 a 0. O každom dokumente si zachováva informáciu o slovách, ktoré sa v ňom nachádzajú. Výhody takejto reprezentácie sú hlavne vo veľmi jednoduchom a prehľadnom vyhľadávaní termu.

Povedzme, že máme k dispozícií tri vektory:

· Prvý (1,0,1,0,1)

· Druhý (1,1,1,0,0)

· Tretí (0,1,0,1,0)

Ak pracujeme s boolovskými vektormi, vieme ich pomerne jednoducho násobiť, spočítať a pod. Našou úlohou je nájsť taký vektor, ktorý patrí pod prvý vektor a zároveň, nepatrí pod tretí vektor. Výsledok by bol nasledovný:

Prvý AND NOT (Tretí)

(1,0,1,0,1) AND NOT (0,1,0,1,0) = 10101 * 10101 = 10101

Ďalšou výhodou boolovského modelu je jeho použitie pri vyhľadávaní podobností. Pre vypočítanie podobnosti vektorov používame skalárny súčin vektorov. Ide o operáciu na dvoch n-rozmerných vektoroch. Výsledkom tejto operácie je číslo (čiže skalár, nie vektor).

Skalárny súčin vypočítame nasledovne:

$$\vec{x} \cdot \vec{y} = (x_1 y_1 + x_2 y_2 + \cdots + x_n y_n)$$

ÚLOHA

K dispozícií máme vektory A (1,1,0,1), B (0,1,1,1), C (1,1,0,0), D (0,0,1,1), E (1,0,0,0). Zistite, akú podobnosť má vektor A s ostatnými vektormi.

Riešenie:

A * B =(1,1,0,1)*(0,1,1,1)= 1*0 + 1*1 + 0*1 + 1*1 = 2

A * C =(1,1,0,1)*(1,1,0,0)= 1*1+ 1.1 + 0*0 + 1*0 = 2

A * D =(1,1,0,1)*(0,0,1,1)= 1*0 + 1*0 + 0*1 + 1*1 = 1

A * E =(1,1,0,1)*(1,0,0,0)= 1*1 + 1*0 + 0*0 + 1*0 = 1

Z výsledkov vyplýva, že vektor A je najpodobnejší vektoru B a C.

Vektorový model

Boolovský model uvedený v predchádzajúcej podkapitole je veľmi jednoduchý pre reprezentáciu dokumentu ale hlavne pre samotné vyhľadávanie dokumentov. Nedokáže však zachytiť množstvo alebo kvalitu uložených vlastností/slov v dokumente. Z tohto dôvodu sa prakticky používa iba pre štúdium úvodu do oblasti vyhľadávania dokumentov.

Použiteľnejší je vektorový model dokumentu (boolovský model využíval tiež vektory, tzv. boolovské vektory, ktorých prvky nadobúdajú hodnotu 0 alebo 1). Vo vektorom modeli tiež predpokladáme nemenné poradie dokumentov (v našom príklade rozprávkových bytostí) a tiež slov/termov (v našom príklade vlastností pytačov). Na rozdiel od boolovského modelu vieme vo vektorovom zohľadniť frekvenciu výskytu slov, dôležitosť slov a pod.

V nasledujúcej ukážke môžeme vidieť typický výsledok tabuľky početností slov vo vete. V ukážke je použitá metóda **CountVectorizer()**. Slúži na vytvorenie

vektora početnosti slov priamo z textov. Vstupom do metódy je zoznam textov. Metóda je súčasťou knižnice **scikit-learn**.

```
1  from sklearn.feature_extraction.text import CountVectorizer
2
3  vectorizer = CountVectorizer()
4
5  doc = ["good, good good good bad good bad good good good good good","good bad bad bad bad bad"]
6
7  vectorizer.fit(doc)
8
9  print("Slovník: ", vectorizer.vocabulary_)
10
11 vector = vectorizer.transform(doc)
12
13 print(vector.toarray())
```

```
Slovník:  {'good': 1, 'bad': 0}
[[ 2 10]
 [ 5  1]]
```

Obrázok 31 Príklad vektorového modelu

Pravdepodobnostný model

Tento model bližšie popisuje ako model fungujúci na princípe pravdepodobnostného ohodnocovania. Systém sa snaží čo najlepšie odhadovať pravdepodobnosti výskytu a výsledky radí vzostupne podľa relevancie. Tento model taktiež vychádza z Boolovského modelu. Jeho hlavný princíp vychádza z výpočtu pravdepodobnosti, do akej miery dokument patrí do množiny dokumentov, ktoré sú relevantné k ideálnej množine dokumentov v odpovedi na dopyt.

Tento vzťah je vyjadrený prostredníctvom vzorca:

$$sim(d_j, q) = \frac{p(\frac{R}{q}, \bar{d}_j)}{p(\frac{\bar{R}}{q}, \bar{d}_j)}$$

Medzi známe modely patrí aj model distribuovanej sémantiky, ktorý zovšeobecnením vektorového modelu.

TF-IDF

Relatívne početnosti sú základom štatistického modelu TF-IDF. V modeli je definovaná frekvencia termu TF (Term frequency), táto predstavuje spomínané relatívne početnosti. TF je teda počet výskytov termu/slov v dokumente normalizovaný delením celkovým počtom termov/slov v dokumente. Vypočíta sa ako pomer frekvencie termu v dokumente ku celkovému počtu termov.

Nech t je term/slovo, d je dokument, w je akýkoľvek term v dokumente, potom frekvenciu termu/slova t v dokumente d vypočítame ako:

$$tf(t,d) = \frac{f(t,d)}{f(w,d)},$$

kde f(t,d) je počet termov/slov v dokumente d a f(w,d) je počet všetkých termov v dokumente.

Pri výpočte TF-IDF sa ešte zohľadňuje počet všetkých dokumentov, v ktorých sa term/slovo vyskytuje. Tento počet označíme df(t,D), teda dokumentová frekvencia (document frequency) a vyjadríme ako

$$df(t,D) = \sum (d \in D : t \in d),$$

, kde D je korpus všetkých dokumentov, s ktorými pracujeme.

Na základe dokumentovej frekvencie je možné vypočítať Inverznú dokumentovú frekvenciu, ktorá vyjadruje ako bežný je term/slovo v korpuse dokumentov. Čím je term bežnejší, tým je jej hodnota nižšia. Vypočítame ju ako:

$$idf(t,D) = \log \frac{N}{df(t,D)+1}$$

, kde N je počet dokumentov v korpuse a df(t,D) je už spomínaná dokumentová frekvencia. Hodnota +1 vo vzorci je z dôvodu ošetrenia delenia nulou, pretože dokumentová frekvencia môže byť aj 0.

Uvedené vzorce predstavujú čiastočné výpočty pre výpočet TF-IDF (Term frequency – inverse document frequency), ktorá určuje ako dôležité je vybrané slovo pre daný dokument v korpuse dokumentov. TD-IDF vypočítame ako:

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

teda presne podľa jej názvu ako frekvencia termu/slova násobená inverznou dokumentovou frekvenciou.

Vektory TF-IDF vypočítame pomocou metódy TfidfVectorizer knižnice scikit-learn. Na rozdiel od príkladu pre CountVectorizer() z predchádzajúcej kapitoly, potrebujeme uvažovať s úplným korpusom dokumentov, t.j všetkými vizitkami. Väčšina zdrojového kódu je prakticky rovnaká ako v predchádzajúcej kapitole, nahradená je iba metóda pre výpočet TF-IDF.

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

doc = ["good, good good good bad good bad good good good good good","good bad bad bad bad bad"]

vectorizer = TfidfVectorizer()

vectors = vectorizer.fit_transform(doc)

feature_names = vectorizer.get_feature_names_out()
dense = vectors.todense()
denselist = dense.tolist()

df = pd.DataFrame(denselist, columns=feature_names)

print(df)
```

```
        bad      good
0  0.196116  0.980581
1  0.980581  0.196116
```

Obrázok 32 Príklad TF-IDF

1.5.4 Metriky podobnosti

Základom určovania podobnosti je navrhnúť vhodnú reprezentáciu objektov. Nielen pre dokumenty je najvhodnejšia reprezentácia formou vektorov, ktorú sme

popísali v predchádzajúcom texte. Poznáme veľa druhov metrík, my si ale uvedieme základné dve metriky pre NLP:

- euklidovu vzdialenosť
- kosínusovú podobnosť

Euklidova vzdialenosť

Základnou metrikou pre určenie podobnosti je euklidovská vzdialenosť. Euklidovská vzdialenosť je známa aj ako jednoduchá vzdialenosť, prezýva sa aj ako L2-Norm. Euklidovská vzdialenosť medzi dvoma bodmi je dĺžka cesty, ktorá ich spája.

Na vypočítanie euklidovej vzdialenosti potrebujeme knižnicu numpy.

```
1 import numpy as np
2
3 vektor1 = np.array([4,2])
4 vektor2 = np.array([10,2])
5 vektor3 = np.array([1,5])
```

Obrázok 33 Výpočet euklidovej vzdialenosti pomocou knižnice numpy

Euklidovu vzdialenosť vektorov x a y vypočítame ako:

$$m_e(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2},$$

,pričom x a y sú vektory s rovnakým počtom prvkov. V našom príklade vieme jednoducho urobiť funkciu pre výpočet euklidovskej vzdialenosti, kde volaním vytvorenej funkcie **euclidean_distance()** môžeme zistiť podobnosť (vzdialenosť) medzi jednotlivými vektormi.

```
1 def euclidean_distance(x, y):
2     return np.sqrt(np.sum(x - y) ** 2)
```

Obrázok 34 Funkcia euclidean_distance()

Pre definíciu funkcie sme použili knižnicu NumPy s nasledovnými funkciami:

	Popis	Príklad
sqrt	Funkcia slúži na výpočet druhej odmocniny	In [16]: `np.sqrt(81)` Out[16]: 9.0
sum	Funkcia na súčet jednotlivých prvkov vektorov	In [15]: `v = np.array([1,4,2])` `np.sum(v)` Out[15]: 7
**	Operátor pre mocninu daného čísla	In [17]: `4 ** 3` Out[17]: 64 In [18]: `4 ** 2` Out[18]: 16

Obrázok 35 Vysvetlenie matematických operácii

Následne vypíšeme vzdialenosť medzi jednotlivými vektormi:

```
1 print(euclidean_distance(vektor1, vektor2))
2 print(euclidean_distance(vektor1, vektor3))
```

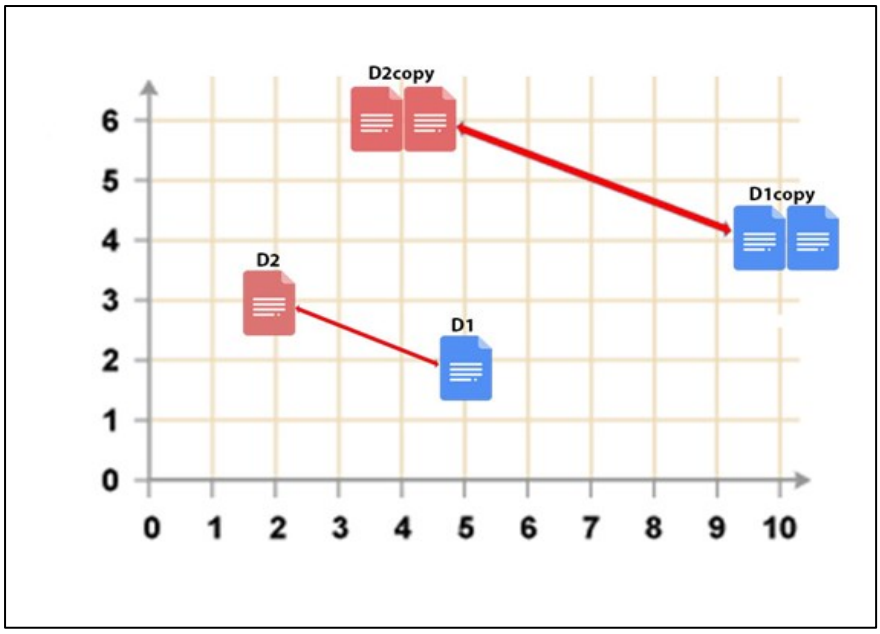
```
6.0
0.0
```

Obrázok 36 Výstup funkcie euclidean_distance()

Kosínusová podobnosť

Bežne používaný prístup vektorovej reprezentácie textu je založený na počte slov v dokumentoch. Tento prístup má však jednu chybu. S rastúcou veľkosťou dokumentu sa počet bežných slov zvyšuje.

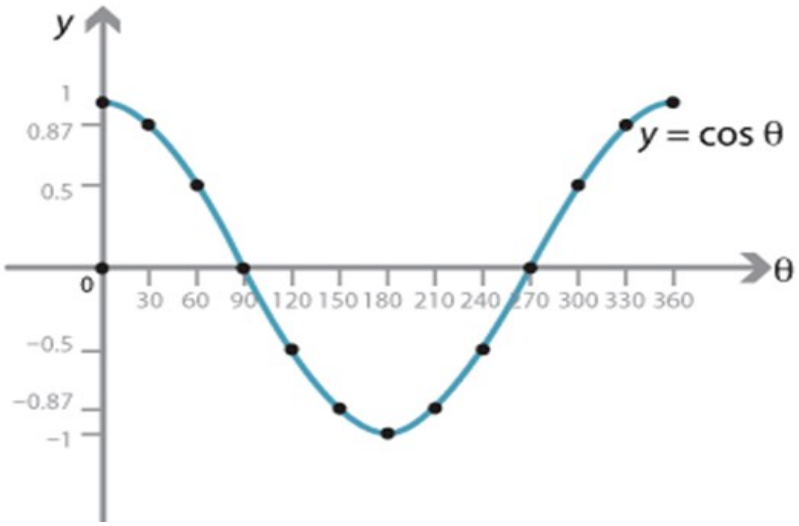
Ak by sme uvažovali s dvoma dokumentami D1 a D2. Následne by sme vytvorili dokument D1 copy tak, že by sme iba zduplikovali obsah dokumentu (t.j. dokument D1 by sa v ňom nachádzal 2x). Podobne by sme vytvorili aj dokument D2copy. Pri vektorovej reprezentácii dokumentov, by jednotlivé prvky vektora boli v dokumentoch D1copy a D2copy dvojnásobné. Pri metrikách ako euklidovská vzdialenosť by sme potom zistili, že D1 a D2 majú k sebe bližšie (sú si viac podobné) ako D1copy a D2copy. To ale nie je pravda.



Obrázok 37 Ukážka podobnosti dokumentov pri duplikácii

Kosínová podobnosť spolu s modelom TF-IDF pomáha prekonať túto zásadnú chybu v prístupe „počet slov“ alebo euklidovskej vzdialenosti. Kosínová podobnosť je metrika, ktorá sa používa na meranie podobnosti dokumentov bez ohľadu na ich veľkosť. Je vlastne vyjadrením veľkosti uhla, ktorý zvierajú dva vektory reprezentujúce dva dokumenty.

Kosínus 0° je 1, a je menší ako 1 pre akýkoľvek iný uhol. Je to teda pohľad na orientáciu a nie na veľkosť vektorov. Dva vektory s rovnakou orientáciou majú kosínusovú podobnosť 1, dva vektory pri 90° majú podobnosť 0 a dva vektory diametrálne protikladné majú podobnosť -1, nezávisle od ich veľkosti.



Pre dva dokumenty (vektory) a a b vypočítame kosínusovú podobnosť nasledovne:

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \ \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \ \sqrt{\sum_1^n b_i^2}}$$

kde:

$$\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

je vektorový súčin.

PRÍKLAD

Majme tri vektory:

- vektor1 = (4,2)
- vektor2 = (10,2)
- vektor3 = (1,5)

V jazyku Python môžeme potom kosínusovú podobnosť vypočítať nasledovne:

```
1 import numpy as np
2
3 vektor1 = np.array([4,2])
4 vektor2 = np.array([10,2])
5 vektor3 = np.array([1,5])
```

```
1 def cosine_similarity(x,y):
2     return np.dot(x,y) / (np.sqrt(np.dot(x,x)) * np.sqrt(np.dot(y, y)))
```

```
1 print(cosine_similarity(vektor1, vektor2))
2 print(cosine_similarity(vektor1, vektor3))
```

```
0.9647638212377322
0.6139406135149205
```

Obrázok 38 Výpočet kosínusovej vzdialenosti v Pythone

Najčastejším prístupom je použitie TF-IDF modelu dokumentov v kombinácii s kosínusovou podobnosťou. V praxi však výpočet TF-IDF modelu dokumentov namiesto vektorovému modelu početností je iba záležitosťou zmeny metódy v zopár riadkoch zdrojového kódu. Z tohto dôvodu sme v kapitole pracovali s vektorovým modelom početnosti slov v dokumente. Kombináciu TF-IDF a kosínusovej podobnosti uvedieme v Aplikačnom príklade v závere tejto publikácie.

1.6 Pokročilejšie nástroje na reprezentáciu slov

1.6.1 Knižnice

TensorFlow

Knižnica, ktorá je navrhnutá a vyvíjaná spoločnosťou Google patrí medzi najpopulárnejšie knižnice pre vývoj aplikácií pomocou strojového učenia. Poskutuje extrémnu prispôsobiteľnosť rôznych možností komplexných návrhov algoritmov pre neurónové siete, spolu s vysokou výkonnosťou pri numerických výpočtoch. Netreba dodávať, že túto knižnicu využíva Google aj pri práci na svojich projektoch, a teda o kvalite tejto knižnice netreba pochybovať.

Zároveň je TensorFlow nízkoúrovňová knižnica, čo považujeme za jej nevýhodu. Náročnosť pri pracovaní s ňou je pomerne veľká a zato s tým súvisí aj náročnosť pri hľadaní chýb v kóde. Toto negatívum mierne odstraňuje použitie knižnice Keras, ktorá ku knižnici TensorFlow poskytuje jednoduchšie rozhranie na vyššej úrovni.

Keras

Keras poskytuje vysokú výpočtovú silu vďaka backendu, na ktorom beží knižnica TensorFlow. Práve TensorFlow umožňuje Keras vytvárať sofistikované modely neurónových sietí akými sú viacvrstvové perceptrónové siete, konvolučné neurónové siete alebo rekurentné neurónové siete. Výhodou Keras oproti TensorFlow je jednoduchosť pri pracovaní s ňou. Vďaka tomu, že Keras pracuje na vyššej úrovni je mnoho detailov presunutých do abstraktu, a tým sa kód stáva omnoho prehľadnejším a stručným. Namiesto riešenia ako správne získať výstup jednej funkcie a vložiť ho ako vstup do druhej sa vývojár môže sústrediť na podstatu svojho cieľa pri riešení problému umelej inteligencie a detaily týchto

procesov sú na backende riešené za nás. Vďaka týmto výhodám je Keras jedna z najpopulárnejších variantov pri pracovaní s neurónovými sieťami.

PyTorch

PyTorch je open-source knižnica založená na knižnici Torch. Je odporúčaným nástrojom na vytváranie projektov strojového učenia a hĺbkového učenia. Má vlastnú základnú dátovú štruktúru známu pod názvom tenzor – označenie pre číslo, vektor, maticu ale akékoľvek n-rozmerné pole obsahujúce prvky rovnakého dátového typu.

S knižnicou, ktorá sa zaoberá viacrozmernými poliami sme sa už stretli – numpy. Rozdiel medzi PyTorch a numpy je v tom, že tenzory sú schopné automaticky počítať gradient a podpora GPU, vďaka ktorej sa tenzorové operácie vykonávajú oveľa efektívnejšie.

Gensim

GenSim dokáže efektívnejšie pracovať s veľkými textovými korpusmi ako iné knižnice. Špecializuje sa na identifikáciu sémantickej podobnosti medzi dvoma dokumentmi prostredníctvom modelovania vektorového priestoru a súboru nástrojov na modelovanie tém. Obsah dokumentov pritom vníma ako sekvencie vektorov a zhlukov. Pomocou knižnice NumPy dokáže optimalizovať využitie pamäte a rýchlosť spracovania údajov.

Knižnica GenSim má rovnako široké využitie, napríklad:

- vektorizácia textu,
- sumarizácia textu,
- generovanie textu,
- analýza textu,
- sémantické vyhľadávanie,
- prieskum údajov.

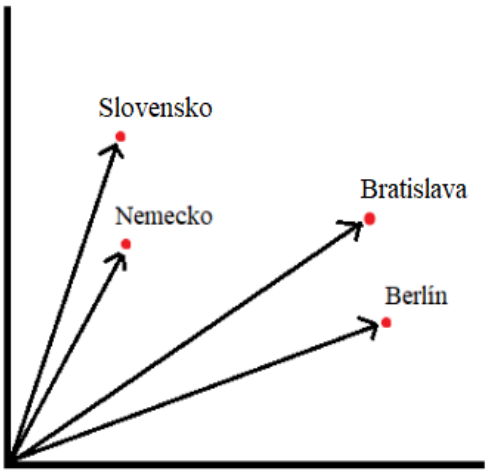
1.6.2 Word2Vec

Ide o jeden z najpoužívanejších modelov na tvorbu vektorovej reprezentácie slov, ktorá sa často označuje aj ako vnorenie slov.

Word2Vec používa na určenie podobnosť medzi jednotlivými vektormi kosínusovú podobnosť:

$$cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} AiBi}{\sqrt{\sum_{i=1}^{n} Ai^2}\sqrt{\sum_{i=1}^{n} Bi^2}}$$

Čím je uhol menší, tým je podobnosť vyššia. Príklad:



Obrázok 39 Príklad podobnosti

Model Word2Vec používa dvojvrstvovú neurónovú sieť ktorá spracováva text pomocou vektorizácie slov. Vstupom je textový korpus a výstupom je sada vektorov. Vznikol v roku 2013 z potreby zachytiť sémantickú podobnosť slov.

Vektory pre jednotlivé slová sa snaží získať dvoma algoritmami:

- **CBOW** (Continuous bag of words) – algoritmus sa snaží predpovedať chýbajúce slovo v závislosti na ostatných slovách v dokumente.
- **Skip-gram** – funguje ako opak CBOW. Dostane na vstup jedno slovo a hľadá tie slová, ktoré sú v rámci kontextu v jeho blízkom okolí.

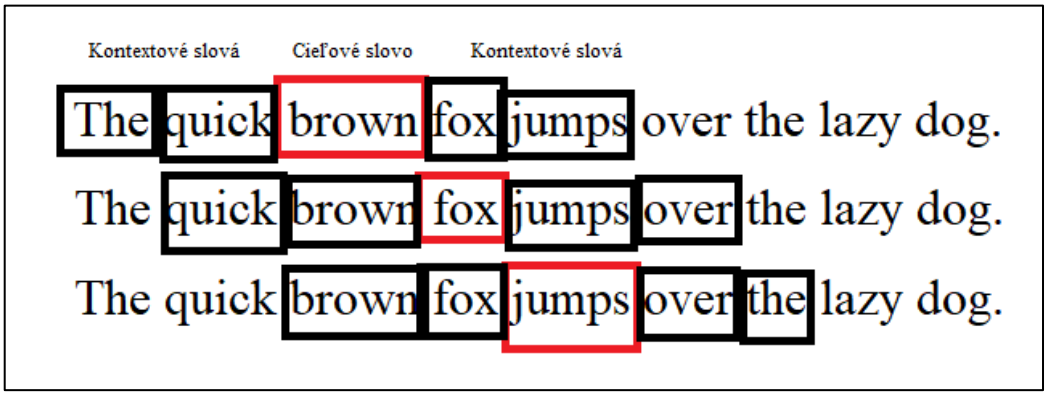
Princíp algoritmu spočíva v zoskupení vektorov podobných slov vo vektorovom priestore, pričom samotná podobnosť vektorov je počítaná matematicky.

V momente, ako má Word2Vec dostatok informácií o dátach, použití a kontexte slov, je schopný veľmi presne odhadnúť význam vlastného slova na základe jeho predchádzajúcich výskytov. Jeho výstupom je spomínaná „sada vektorov“, čo je v podstate slovník, kde je ku každému slovu pripojené aj jeho numerické vyjadrenie.

CBOW

Continuous Bag of Words vznikol z pôvodnej metódy BOW (Bag of Words) jeho rozšírením. Jeho cieľom je predikovať cieľové slovo na základe okolitých slov v jeho okolí.

Na obrázku nižšie je vizualizácia kontextového okna architektúry CBOW o dĺžke 5.



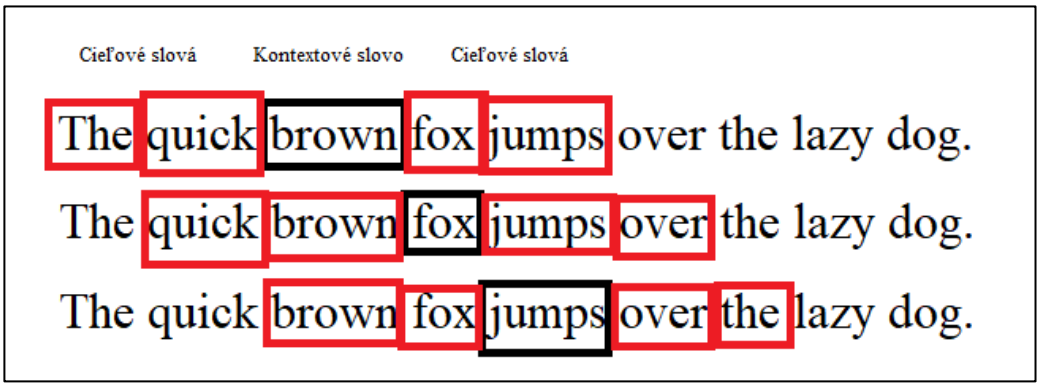
Obrázok 40 Príklad použitia CBOW architektúry

S väčšou hodnotou dĺžky okna je možné zachytiť viac súvislostí medzi susediacimi slovami. Prednosťou tejto metódy je vysoká rýchlosť a lepšia reprezentácia slov, ktoré sa v texte vyskytujú častejšie.

Skip-gram

Tento model je opakom modelu CBOW. Na základe cieľového slova sa predikujú všetky okolité slová. Metóda dokazuje výrazne lepších výsledkov pri menších objemoch dát a lepšie reprezentuje vzácne slová. Na druhú stranu ide o pomalý model než CBOW.

Na obrázku nižšie je zobrazená vizualizácia kontextového okna architektúry Skip-gram o dĺžke 5.



Obrázok 41 Príklad použitia Skip-gram architektúry

Word2Vec môžeme určite popísať ako jednu z najlepších implementácií vektorovej reprezentácie slov. Poskytuje nám široké možnosti využitia nielen s textovými dátami (napr. pri vytváraní hudobných playlistov).

Word2Vec nie je ale jediná technika, ktorá reprezentuje slová pomocou unikátneho vektora. Z Word2Vec vychádza novšia a rýchlejšia verzia označovaná ako FastText. ďalšia alternatíva je GloVe a v neposlednom rade aj Doc2Vec. Všetky spomenuté techniky si popíšeme v ďalších podkapitolách.

FastText

Metóda FastText bola navrhnutá v roku 2016 spoločnosťou Meta.

Algoritmus metódy FastText obsahuje predučené modely s anglickými vektormi sov naučených z wikipédie a webcrawlera. Využíva techniku rozdelenia slov na znakové N-gramy, často medzi tri až šesť znakov. Príklad:

slovo = where

n = 3

<wh, whe, her, ere, re>

Rozdeľovaním slov by sa mala zaručiť lepšia vektorová reprezentácia slov, ktoré sa tak často nevyskytujú.

GloVe

Jedna z variant Word2Vec, ktorá slúži rovnako na natrénovanie vektorovej reprezentácie slov je GloVe (Global Vectors).

Učenie modelu Word2Vec je založené na závislostiach cieľových slov s ich kontextom. Chybou je ale to, že metóda nerešpektuje častejšiu frekvenciu niektorých slov v texte. A práve GloVe dáva dôraz na frekvenciu výskytu slov v texte. Kombinácia vektorov slov teda priamo súvisí aj s pravdepodobnosťou, že sa dve slová vyskytnú v tejto kombinácií spoločne.

VÝHODY

GloVe je veľmi rýchla metóda na trénovanie a je škálovateľná aj pre veľké slovníky. Poskytuje dobré výsledky aj s menším slovníkom a vektory s menším počtom dimenzií.

NEVÝHODY

Potrebuje veľa pamäti a je citlivá na inicializáciu parametrov.

1.6.3 Doc2Vec

Doc2Vec je nadstavba metódy Word2Vec a je silne založená na jej princípe. Jej cieľom je vypočítať vektorovú reprezentáciu celých dokumentov nezávislé na ich dĺžke. Problém môže spôsobiť to, že dokumenty nemusia byť nutne zložené z logickej štruktúry.

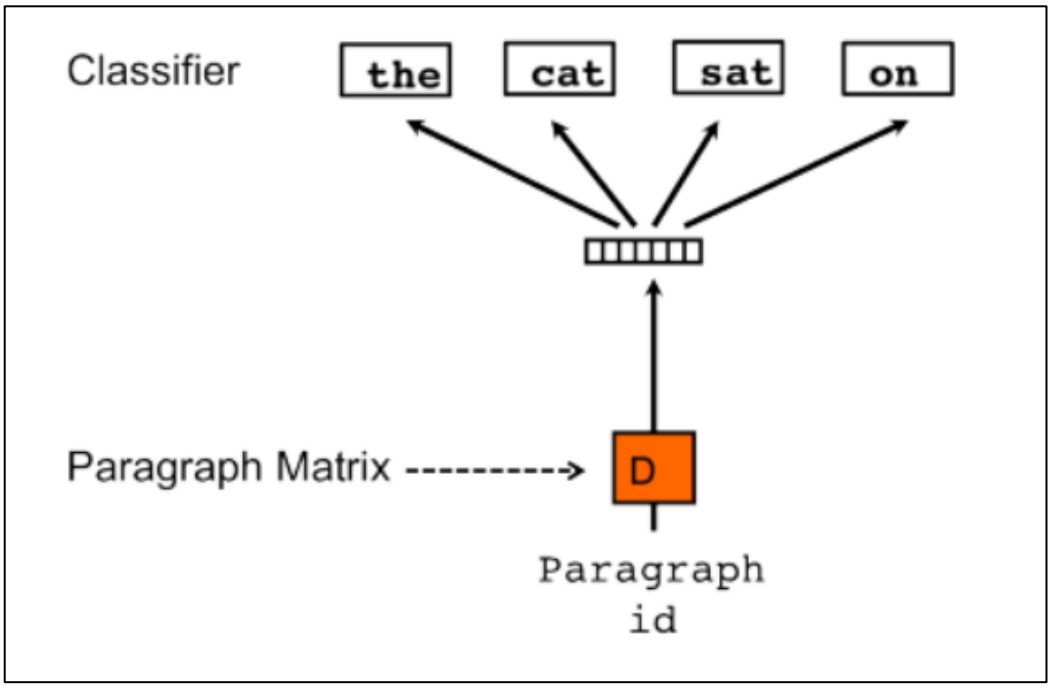
Hlavným rozdielom s Word2Vec je rozšírenie o tzv. *Paragraph ID*, ktoré slúži práve k identifikácií konkrétneho dokumentu.

Tak ako Word2Vec, aj Doc2Vec používa dve techniky na trénovanie vektorov viet, a to:

- **PV-DBOW** (angl. Distributed Bag of Words of Paragraph Vector), ktorá zanedbáva kontext slov vo vstupe,
- **PV-DM** (angl. Distributed memory model of Paragraph Vectors) funguje na princípe CBOW.

PV-DBOW

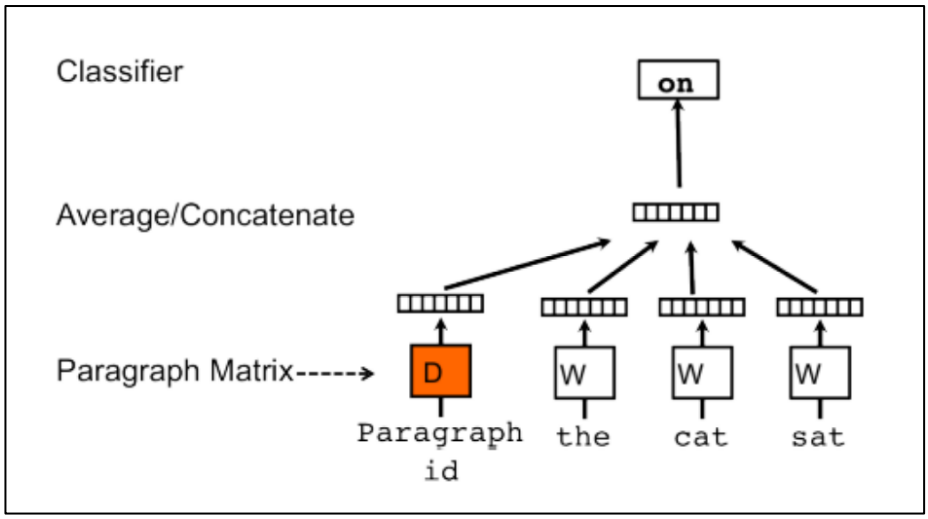
Je rozšírením modelu Skip-gram. Namiesto predikcie slova, ako je v pôvodnom modeli, využíva vektor odstavca ku klasifikácií dokumentu. Prednosťou tohto modelu je jeho rýchlosť, nakoľko využíva málo pamäti, pretože nepotrebuje ukladať vektory slov. Obrázok nižšie je grafické zobrazenie modelu PV-DBOW.



Obrázok 42 Princíp PV-DBOW architektúry

PV-DM

Je rozšírením pôvodného modelu CBOW a je pridaný o hodnotu Paragraph ID. Tento vektor slúži ako unikátny identifikátor dokumentu, časti textu a vyjadruje numerickú reprezentáciu dokumentu. Samotný model slúži ako pamäť, ktorá si pamätá, čo v texte v danom kontexte chýba alebo vyjadruje tému dokumentu. Jeho grafické zobrazenie na obrázku nižšie.



Obrázok 43 Princíp PV-DM architektúry

Priebeh modelu:

1. Vektory slov a vektory odstavcov sa inicializujú náhodne,
2. Vektory odstavcov sú vždy priradené len jednému dokumentu. Vektory slov sú zas priradené každému dokumentu.
3. Vektory slov a odstavcov sa následne spriemerujú ale sa zreťazia a vložia do stochastického gradientu a gradient sa následne získa spätným šírením.

2 Získavanie lingvistických dát

2.1 Získavanie dát - úvod

Na akúkoľvek analýzu prirodzeného jazyka je potrebné analyzovať údaje. Prvým krokom je preto extrahovanie textu z jeho natívnej formy (v tomto prípade súborov pdf alebo html) do textových súborov. Paralelný korpus je súbor textov zarovnaný prekladmi do iného jazyka. V súčasnosti je k dispozícii niekoľko paralelných korpusov. Často ide o desiatky až stovky miliónov slov. Niektoré sú dostupné na internete, mnohé ďalšie distribuje Linguistic Data Consortium z Pensylvánskej univerzity alebo je k dispozícii aj korpus Europarl.

Mnohé webové stránky sú preložené do viacerých jazykov a niektoré spravodajské organizácie, ako napríklad BBC, uverejňujú svoje články pre viacjazyčných recipientov. Získavanie paralelných korpusov z webu je niekedy jednoduché, ale často je komplikované z viacerých dôvodov.

Práca s dobrým dátovým setom je kľúčová na zabezpečenie dobrej výkonnosti modelu strojového učenia, takže prijatie dobrej metódy extrakcie údajov môže priniesť nespočetné výhody pre nasledujúce procesy.

2.1.1 Získavanie jazykových dát z internetu

Web scraping je technika používaná na zhromažďovanie obsahu a údajov z internetu. Tieto údaje sa zvyčajne ukladajú do lokálneho súboru, aby sa s nimi dalo podľa potreby manipulovať a analyzovať. Webové aplikácie na scraping (alebo "boti") sú naprogramované tak, aby navštevovali webové stránky, zachytili príslušné stránky a extrahovali užitočné informácie. Automatizáciou tohto procesu môžu títo roboti extrahovať obrovské množstvo údajov vo veľmi krátkom čase. To má zjavné výhody v digitálnej dobe, keď veľké dáta – ktoré sa neustále aktualizujú a menia – zohrávajú významnú úlohu.

Informácie, ktoré je možné získať z webu zahŕňajú:

- obrázky,
- videá,
- text,
- produktové informácie,

- recenzie (napr. TripAdvisor),
- a iné.

Pre účely vytvorenia korpusu sú najzaujímavejšie textové dáta.

2.1.2 Fungovanie web scrapera

Aj keď sa konkrétna metóda líši v závislosti od použitého softvéru alebo nástroju, všetky web scrapre sa riadia tromi základnými princípmi:

- odoslanie požiadavky HTTP na server,
- extrahovanie a parsovanie (alebo preloženie) kódu webovej stránky, a
- lokálne uloženie relevantných údajov.

HTTP request

Keď navštívite ako jednotlivec webovú stránku prostredníctvom prehliadača, odošlete takzvanú požiadavku HTTP. Toto je v podstate digitálny ekvivalent klopania na dvere a žiadania o vstup. Po schválení vašej žiadosti môžete získať prístup na túto stránku a všetky informácie na nej. Rovnako ako osoba, aj web scraper potrebuje povolenie na prístup na stránku. Preto prvá vec, ktorú web scraper urobí je, že odošle požiadavku HTTP na web, na ktorý sa zameriava.

Extrahovanie a parsovanie kódu webovej stránky

Akonáhle webová stránka poskytne prístup scraperovi, bot môže čítať a extrahovať kód HTML alebo XML stránky. Tento kód určuje štruktúru obsahu webovej stránky. Scraper potom parsuje kód (čo v podstate znamená jeho rozdelenie na jednotlivé časti), aby mohol identifikovať a extrahovať prvky alebo objekty, ktoré boli preddefinované tým, kto bota spustil. Môžu zahŕňať konkrétny text, hodnotenia, triedy, značky, identifikátory alebo iné informácie.

Lokálne uloženie relevantných údajov

Po prístupe, získaní a analýze html alebo XML sa do web scrapera lokálne uložia príslušné údaje. Ako už bolo spomenuté, extrahované údaje sú preddefinované – botovi je určené, čo má zhromažďovať). Údaje sa zvyčajne ukladajú ako štruktúrované údaje, často v excelovom súbore, napríklad vo formáte .csv alebo .xls. Tento proces nevykonáva len raz, ale nespočetne veľakrát. To prichádza s vlastnou škálou problémov, ktoré je potrebné vyriešiť. Napríklad zle

kódované scrapre môžu odosielať príliš veľa požiadaviek HTTP, čo môže zlyhať na webe. Každá webová stránka má tiež iné pravidlá pre to, čo roboti môžu a nemôžu robiť. Spustenie kódu na web scraping je len jednou časťou procesu.

Zoradte kroky fungovania web scrapera

- extrahovanie a parsovanie (alebo preloženie) kódu webovej stránky
- lokálne uloženie relevantných údajov
- odoslanie požiadavky HTTP na server

2.1.3 Získavanie textových dát z PDF dokumentov

Získavanie textových dát z PDF dokumentov

Extrahovanie textov z PDF dokumentov je neľahký proces, nakoľko ide o „layout-based“ formát, ktorý neobsahuje sémantické informácie o častiach textu.

PDF patrí medzi najpopulárnejšie formáty pre elektronické dokumenty. Samotný Google prehliadač v súčasnosti indexuje viac ako 3 miliardy PDF dokumentov, čo je viac ako pre akýkoľvek iný dokumentový formát, s výnimkou HTML. Avšak, PDF je „layout-based“ formát, čo znamená, že špecifikuje na každej strane pozíciu a font jednotlivých znakov, z ktorých sa text skladá. Jazyky popisujúce strany (PDLs – page description languages) sa používajú ako komunikácia medzi formátovaným popisom strany alebo dokumentu v kompozičnom systéme a výstupným zariadením, napríklad monitorom alebo tlačiarňou. Výhodou dokumentu, ktorý je zapísaný v PDL (Page Description Language) formáte je plná kontrola nad vzhľadom. Nevýhodou týchto dokumentov je veľkosť súboru a chýbajúca logická štruktúra v porovnaní so značkovacím jazykom HTML. V súčasnosti v spracovaní prirodzeného jazyka dominuje dátami riadená technika. Veľký pokrok vo viacerých odvetviach spracovania prirodzeného jazyka je možné vidieť vďaka pokroku v strojovom učení a narastajúcim dátovým sadám. Avšak, dostupnosť čistých dátových setov je stále pretrvávajúcim problémom pre väčšinu jazykov na svete. Bežne sa dáta získavajú dolovaním z World Wide Web, aby sa rozšírili dátové sety. S rastúcim dopytom po stále väčších dátových setoch je nevyhnutné hľadať aj ďalšie zdroje textov. PDF formát vo svojej podstate reprezentuje papier v digitálnej forme, preto prirodzene obsahuje veľké množstvo textov. Tento formát bol optimalizovaný na tlač papiera, a preto sa zameriava najmä na vizuálnu stránku dokumentov. Formát

nie je štruktúrovaný a nerozlišuje sémantický význam objektov na strane. To spôsobuje, že pri extrahovaní informácií sa nerobia rozdiely medzi nadpismi, paragrafmi, textom v tabuľkách alebo popismi obrázkov. PDF asociácia, ktorá definuje ISO štandard pre formát PDF sa v súčasnosti zameriava na sprístupnenie PDF dokumentov. Cieľom je, aby aj ľudia s postihnutím (napr. zrakovým) mali plnohodnotný prístup k informáciám v dokumente. Základom je možnosť pridať logickú štruktúru do formátu, ktorá sa odkazuje na konkrétne časti strán v dokumente a pridáva im sémantický význam. Rovnako ako v HTML, aj v PDF dokumentoch je sémantický význam obsahu predstavovaný tagmi. Otagovaný PDF dokument následne umožňuje presné extrahovanie paragrafov, anotácií, nadpisov, ale aj správne poradie čítania pri viac-stĺpcových dokumentoch. Uvádzanie logickej štruktúry v formáte PDF nie je povinné, preto existuje málo dokumentov s danou štruktúrou. Niektoré programy ako napríklad balíček kancelárskych programov od Apple, pridávajú tagy pri exportovaní do PDF automaticky.

2.1.4 Problémy pri extrahovaní textu z PDF dokumentov

Keďže PDF zobrazuje text na základe fontu a pozície na danej strane pre jednotlivé znaky, nástroje, ktoré extrahujú text sa stretávajú s niekoľkými problémami. Uvedieme niektoré:

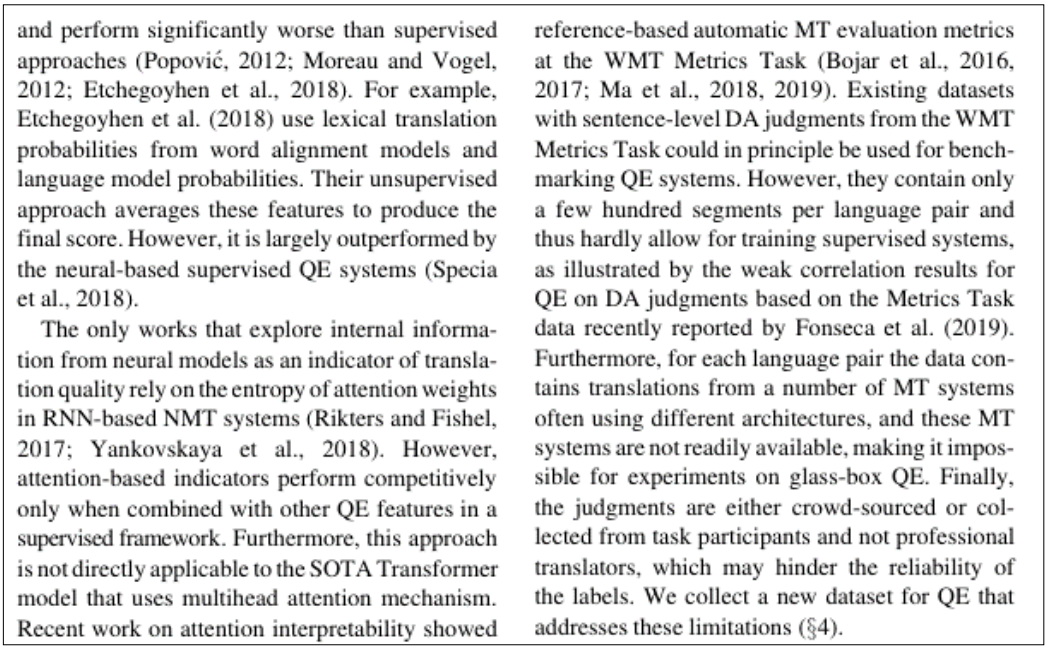
- Identifikácia slov
- Poradie slov
- Ohraničenie paragrafov
- Ligatúry
- Rozdelenie slov
- Sémantický význam

Identifikácia slov

Dôležité pri vyhľadávaní slov v dokumente. Ak sa slovo nepodarí správne identifikovať, hľadané slovo sa nemusí podariť nájsť. Problém pri identifikovaní slov spočíva v tom, že medzery medzi písmenami sa môžu líšiť nielen medzi riadkami, ale aj v rámci jedného riadku. Dlhé slová môžu byť na konci riadku oddelené pomlčkou, čím sa jedno slovo vizuálne rozdelí a v dokumente sa bude vyskytovať na dvoch miestach.

Poradie slov

Určiť poradie slov je podstatné pri aplikáciách ako sú napríklad čítačky e-kníh alebo ak chceme dostať neformátovaný text. Poradie slov sa dá určiť na základe pozície slov v PDF. Avšak, pri dokumentoch, ktoré majú viac-stĺpcové rozloženie sa poradie nedá určiť na základe pozície.



and perform significantly worse than supervised approaches (Popović, 2012; Moreau and Vogel, 2012; Etchegoyhen et al., 2018). For example, Etchegoyhen et al. (2018) use lexical translation probabilities from word alignment models and language model probabilities. Their unsupervised approach averages these features to produce the final score. However, it is largely outperformed by the neural-based supervised QE systems (Specia et al., 2018).

The only works that explore internal information from neural models as an indicator of translation quality rely on the entropy of attention weights in RNN-based NMT systems (Rikters and Fishel, 2017; Yankovskaya et al., 2018). However, attention-based indicators perform competitively only when combined with other QE features in a supervised framework. Furthermore, this approach is not directly applicable to the SOTA Transformer model that uses multihead attention mechanism. Recent work on attention interpretability showed

reference-based automatic MT evaluation metrics at the WMT Metrics Task (Bojar et al., 2016, 2017; Ma et al., 2018, 2019). Existing datasets with sentence-level DA judgments from the WMT Metrics Task could in principle be used for benchmarking QE systems. However, they contain only a few hundred segments per language pair and thus hardly allow for training supervised systems, as illustrated by the weak correlation results for QE on DA judgments based on the Metrics Task data recently reported by Fonseca et al. (2019). Furthermore, for each language pair the data contains translations from a number of MT systems often using different architectures, and these MT systems are not readily available, making it impossible for experiments on glass-box QE. Finally, the judgments are either crowd-sourced or collected from task participants and not professional translators, which may hinder the reliability of the labels. We collect a new dataset for QE that addresses these limitations (§4).

Obrázok 44 Ukážka viacstĺpcového dokumentu

Na ukážke je vidieť časť textu z odborného článku, ktorý je písaný v dvoch stĺpcoch. Niektoré nástroje na extrahovanie nedokážu rozpoznať takýto layout a výsledkom je text extrahovaný po riadkoch.

Ohraničenie paragrafov

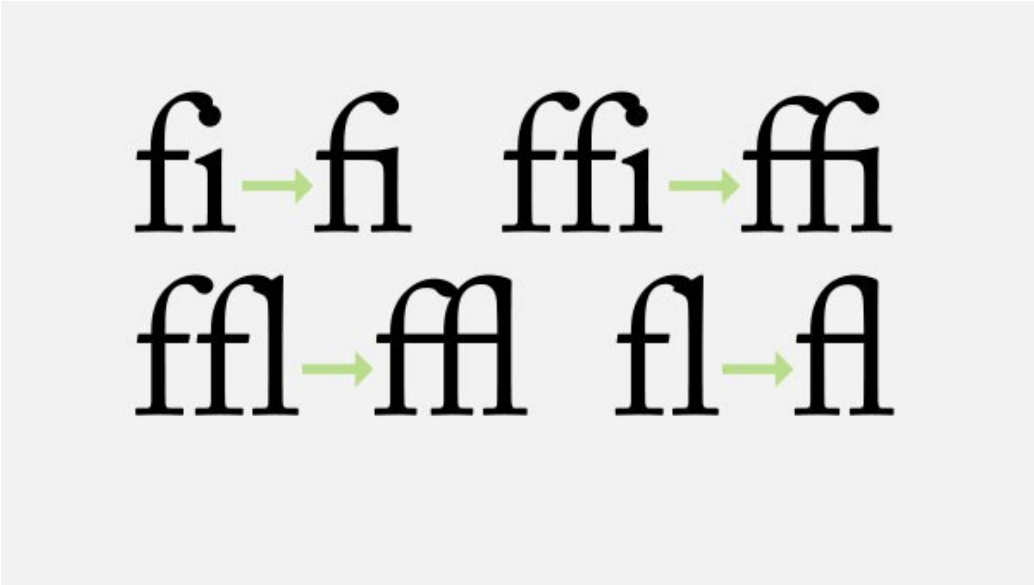
Určiť začiatok a koniec paragrafu je opäť dôležité pre aplikácie, ktoré čítajú dokument. Tento problém je spôsobený tým, ak je napríklad v paragrafe vložený obrázok alebo tabuľka. Aplikácia môže nesprávne vyhodnotiť koniec paragrafu, aj keď narazí na koniec strany. Napriek tomu, že paragraf pokračuje na druhej strane, aplikácia to vyhodnotí ako dva rôzne paragrafy.

Obrázok 45 Ukážka tagovaného PDF

Na obrázku je ukážka tagovaného PDF, ktoré pridáva do dokumentu informácie ako je ohraničenie paragrafov, alebo aj reading order (smer čítania). V takomto prípade je extrahovanie textu jednozačné čo sa týka paragrafov, ale rieši aj problém poradia slov.

Ligatúry

Ďalším problémom je rozpoznávanie znakov. Napríklad znak „fi“ (Unicode U+FB01), predstavuje iba jeden znak. Pri extrahovaní slova „fialka“ môžu byť prvé dve písmená „f“ a „i“ nesprávne identifikované a vo výstupe sa vôbec nebudú nachádzať.



Obrázok 46 Ukážka ligatúr

Rozdelenie slov

Rozdelenie slov pomlčkou spôsobuje problém v prípadoch, keď chceme extrahovaný text znovu použiť, napríklad pri spracovaní prirodzeného jazyka.

Väčšina nástrojov využíva na rozlíšenie pomlčky v zmysle rozdelenia slova heuristické pravidlá.


program and get errors indi-
cating that a linker

Obrázok 47 Ukážka rozdelenia slov na konci riadkov

Na obrázku je zvýraznené slovo "indicating", ktoré je rozdelené pomlčkou. Pri extrahovaní textu z PDF, dostaneme výsledok presne v tomto tvare. Takéto údaje nie sú vhodné na učelý trénovania umelej inteligencie, lebo slová "indi" a "cating" neexistujú.

Sémantický význam

Nakoľko PDF dokument nie je štruktúrovaný formát, nerozlišuje medzi tým, či je text v zmysle paragrafu, nadpisu, poznámky alebo čísla strany. PDF tento nedostatok odstraňuje práve možnosťou doplniť štruktúru dokumentu a vytvoriť tak tagované PDF dokumenty.

3.1.1 Nástroje na extrahovanie dát z PDF dokumentov

Pdftotext

Jeden z najpoužívanejších nástrojov na extrahovanie dát z PDF. Konvertuje PDF do obyčajného textu, ale neidentifikuje paragrafy, ani sémantický význam textov.

Program: https://pypi.org/project/pdftotext/

Pdfix

Ide o SDK na čítanie, zápis, renderovanie a manipuláciu s PDF súbormi. Vďaka pokročilým algoritmom dokáže rozpoznať logickú štruktúru ako texty, nadpisy, obrázky, tabuľky alebo hlavičku a pätu dokumentu. Všetky tieto údaje sú potom dostupné vo formátoch HTML, CSV, JSON alebo XML.

Program: https://pdfix.net/download/

PdfMiner

Nástroj, ktorý dokáže analyzovať štruktúru PDF súboru a konvertuje ho do textového, XML alebo HTML výstupu. Text je rozdelený na paragrafy, riadky a znaky.

Program: https://pypi.org/project/pdfminer/

Pdfbox

Knižnica od spoločnosti Apache, napísaná v jazyku Java. Rovnako ako predchádzajúci nástroj, konvertuje PDF do textového formátu, pričom nerozlišuje paragrafy, ani sémantický význam textov.

Program: https://pdfbox.apache.org/download.html

Pdfforge

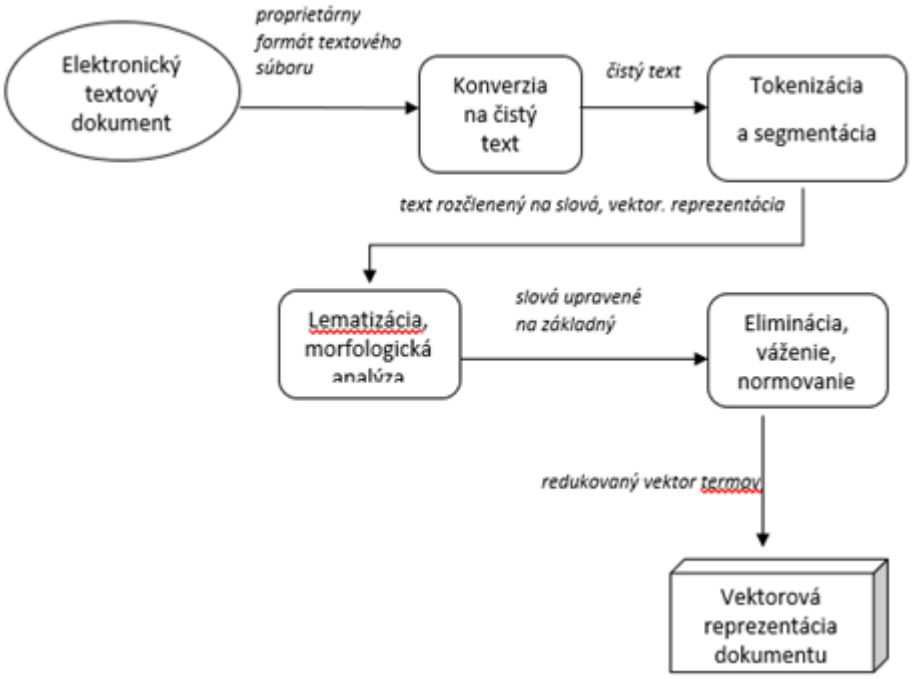
Voľne dostupný online nástroj, ktorého výstupom je textový súbor.

Program: https://download.pdfforge.org/

2.2 Predspracovanie textu elektronických dokumentov

Hlavným cieľom predspracovania textu je rozdeliť text do podoby, ktorú dokážu algoritmy strojového učenia spracovať. Textové údaje získané z prirodzeného jazyka sú neštruktúrované a obsahujú nečisté údaje, tzv. šumy (angl. noisy data). Predspracovanie textu zahŕňa transformáciu textu do čistého a konzistentného formátu, ktorý sa potom môže vložiť do modelu na ďalšiu analýzu a učenie. Predspracovanie textových údajov sa súhrnne označuje ako získanie príznakových popisov pre všetky textové dokumenty zo skúmaného korpusu.

Fázy predspracovania textových údajov:



Obrázok 48 Fázy predspracovania textu

Na vstupe sa predpokladajú elektronické textové dokumenty v rôznych formátoch. Prvým krokom je odstránenie redundantných formátovacích znakov a konverzia dokumentu na tzv. „čistý text“ (angl. plain text). Tento text sa ďalej delí na elementárne textové jednotky, tzv. tokeny (angl. tokens). Následne sa v texte identifikujú slová, tzv. lexikálne jednotky, pre ktoré sa určí príslušný základný tvar (lema) a morfologické kategórie. Napokon sa odstránia neplnovýznamové slová, tzn. tie, pri ktorých sa predpokladá malý prínos k vyjadreniu celkového kontextu dokumentu. Zostávajúce plnovýznamové slová, ohodnotené vhodnou váhovou funkciou, potom tvoria hľadanú vektorovú reprezentáciu vstupného dokumentu. V nasledujúcich podkapitolách podrobnejšie popíšeme jednotlivé fázy predspracovania textu.

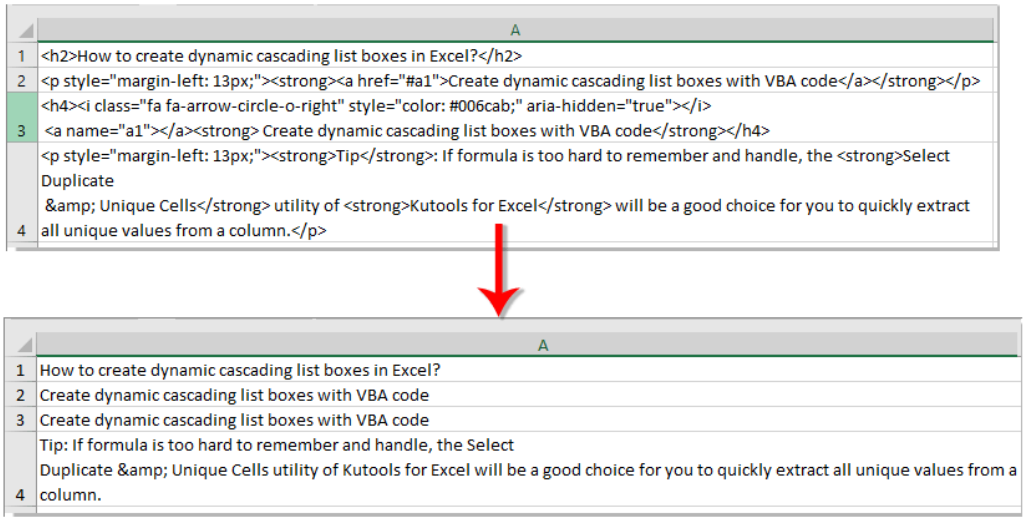
2.2.1 Konverzia elektronických dokumentov na čistý textový formát

Prvým krokom predspracovania dokumentov obsahujúcich textovú informáciu je jej získanie v podobe čistého textu.

Čistý textový formát alebo čistý text možno popísať ako sekvenciu alfanumerických, interpunkčných a rôznych iných oddeľovacích grafických znakov alebo špeciálnych symbolov, ku ktorým môžeme zaradiť znaky, ako napr.

percento (%), ampersand(&), zavináč(@). Každá z týchto skupín znakov plní v texte svoju funkciu. Sekvencie alfanumerických znakov predstavujú lexikálnu a fonetickú hodnotu. V dôsledku toho ich možno označiť za priamych nositeľov obsahu textu. K členeniu obsahu textu sa využívajú interpunkčné a oddeľovacie znaky ako napr. bodka, čiarka, lomka, zátvorky, pomlčka, medzera, tabelátor.

Hlavnou úlohou procesu konverzie elektronických dokumentov na čistý textový formát je odstránenie všetkých netextových informácií a typografických značiek zo vstupného textu. V praxi ide najčastejšie o odstránenie formátovacích značiek, ako napr. typ, veľkosť, farba písma, odsadzovanie textu a pod., ale častokrát aj odstraňovanie obrázkov, grafických elementov, grafov, či tabuliek.



	A
1	How to create dynamic cascading list boxes in Excel?
2	Create dynamic cascading list boxes with VBA code
3	 Create dynamic cascading list boxes with VBA code
4	Tip: If formula is too hard to remember and handle, the Select Duplicate & Unique Cells utility of Kutools for Excel will be a good choice for you to quickly extract all unique values from a column.

	A
1	How to create dynamic cascading list boxes in Excel?
2	Create dynamic cascading list boxes with VBA code
3	Create dynamic cascading list boxes with VBA code
4	Tip: If formula is too hard to remember and handle, the Select Duplicate & Unique Cells utility of Kutools for Excel will be a good choice for you to quickly extract all unique values from a column.

Obrázok 49 Text očistený od netextových informácií

Elektronické dokumenty sú zvyčajne k dispozícii v rôznych formátoch, napr. v MS Word, RTF, PDF, HTML, XML a iné. Každý formát má svoje vlastnosti a iný spôsob konverzie do čistého textového formátu. Hlavným rozdielom medzi formátmi elektronických dokumentov je spôsob akým informácie reprezentujú. Formáty ako HTML alebo formáty vychádzajúce z XML (napr. docx) sú tzv. "structure-based". Na druhej strane, formát PDF je "layout-based".

Pri získavaní čistého textu, predovšetkým pri extrakcii z bohatších formátov (PDF, MS word a HTML), sa často strácajú relevantné a dôležité informácie o obsahu, štruktúre textu ako aj o dokumente ako celku. Viaceré z týchto informácií môžu byť v ďalšom procese veľmi užitočné, a preto je dôležité ich istým spôsobom zachovať. Jednou z možností je archivácia pôvodného

dokumentu spolu s referenciou na príslušný čistý text. Ďalším spôsobom je kódovanie dôležitých údajov v modifikovanom formáte čistého textu, napríklad pomocou XML notácie – týmto spôsobom možno definovať hlavičku s globálnymi informáciami o pôvodnom dokumente, čistý text členiť do kapitol a odsekov, zachovať označenia nadpisov a zvýraznených častí textu, a podobne.

Údaje, ktoré sa pri konverzii elektronických dokumentov na čistý textový formát strácajú, ale dajú sa odvodiť z pôvodného elektronického textového dokumentu nazývame metainformácie. Medzi metainformácie môžeme zaradiť:

- **informácie o štruktúre textu** – štruktúrovanie textu na odseky, kapitoly, nadpisy, zalomenia riadkov a strán alebo typografické informácie ako veľkosť a typ písma, odsadzovanie atď..,
- **metaúdaje** – informácie, ktoré charakterizujú obsah textu a sú zväčša vkladané do dokumentu autorom,
- **bibliografické údaje** – autor, miesto a dátum publikovania, zdroj, vydavateľ,
- **vlastnosti dokumentu** – názov, umiestnenie alebo veľkosť elektronického textového dokumentu.

Predspracovanie textu pozostáva z nasledujúcich častí:

- Odstránenie interpunkcie (Removing punctuations), napr: . , ! $( ) * % @
- Odstránenie stopových slov (Removing Stop words)
- Zmena na malé písmená (Lower-casing)
- Tokenizácia (Tokenization)
- Nájdenie koreňu slova (Stemming)
- Lemantizácia (Lemmatization)

Všetky tieto časti podrobne popisujeme bližšie v kapitole 1.3 Predspracovanie textu.

2.3 Zarovnanie korpusov

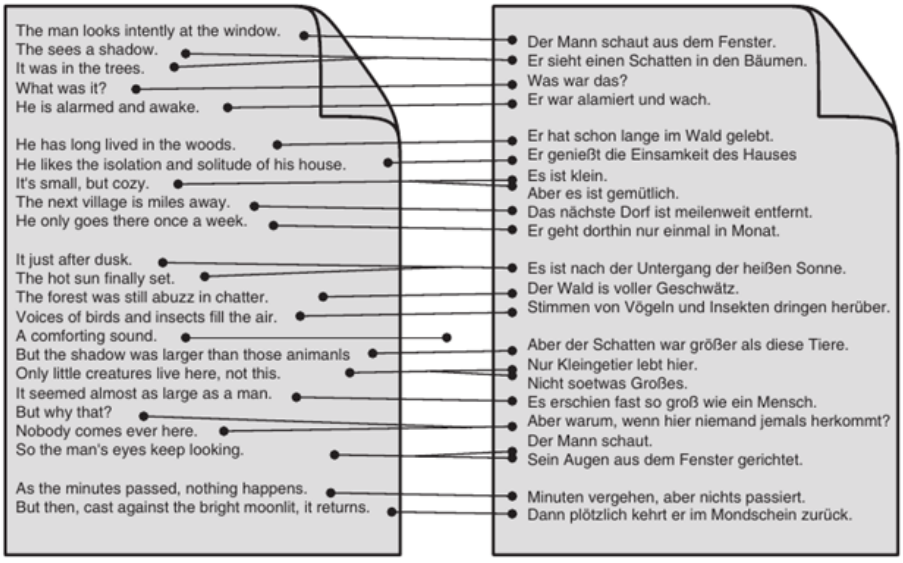
2.3.1 Zarovnanie

Zarovnávanie viet je úloha, pri ktorej sa berú korešpondujúce dokumenty (originál a jeho korešpondujúci preklad) rozdelené na vety a hľadá sa bipartitný

graf, ktorý zodpovedá minimálnym skupinám viet, ktoré sú vzájomnými prekladmi.

Na natrénovanie takmer všetkých systémov strojového prekladu (MT) sa používa bitext (dva korešpondujúce texty (originál a preklad) zarovnané podľa viet). Zistilo sa, že chyby v zarovnávaní majú malý vplyv na výkonnosť štatistického MT (SMT), avšak ukázalo sa, že nesprávne zarovnané vety oveľa viac ovplyvňujú výkom neurónový MT (NMT).

Zriedkavo sa text prekladá slovo za slovo a nie vždy sa prekladá veta za vetu. Dlhé vety sa môžu rozdeliť alebo krátke vety sa môžu spojiť. Dobré zarovnanie je kľúčové aj pre lexikografiu, pretože sa dá využiť na zobrazenie paralelných zhodností a na hľadanie prekladových ekvivalentov, ako aj na extrakciu terminológie. Zarovnávanie paralelných korpusov sa využíva aj v digitálnych humanitných vedách (DH) na rôzne účely, napríklad na učenie historického jazyka alebo zarovnávanie verzií stredovekých textov.



Obrázok 50 Ukážka zarovnania

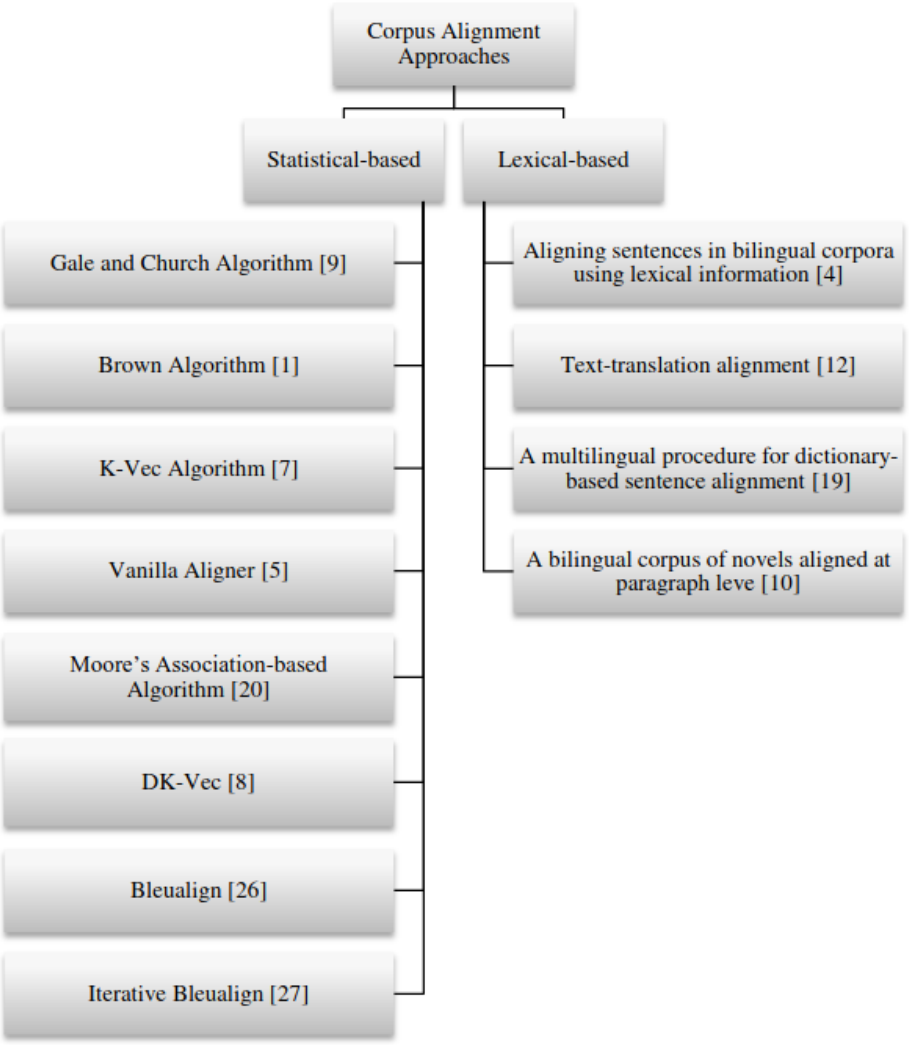
Existujú dva hlavné prístupy k zarovnávaniu textov:

- prístup založený na štatistických údajoch,
- prístup aplikuje lingvistické - lexikálne poznatky.

Na základe týchto prístupov bolo vyvinutých niekoľko techník, pričom každá z nich má svoje výhody a nevýhody.

Prístupy založené na lexikálnych vlastnostiach textu sa opierajú o existujúce lexikálne zdroje, ako sú rozsiahle dvojjazyčné slovníky a glosáre. Tieto techniky sú pomalšie ako techniky založené na štatistických informáciách a sú závislé od jazyka. Hlavnou nevýhodou týchto techník je, že ich výkonnosť do značnej miery závisí od lexikálnych informácií použitých pri procese zarovnávania. Mnohé z týchto metód a techník sa však vyvíjajú, pretože sa očakáva, že budú generovať lepšie výsledky ako štatistické metódy.

Prístupy založené na štatistike sa opierajú o extraligválne kvantitatívne charakteristiky, ako je dĺžka vety, pozícia vety, frekvencia spoluvýskytu, pomer dĺžok viet v dvoch jazykoch atď. Tieto techniky urýchľujú proces zarovnávania a vo všeobecnosti sú nezávislé od jazyka. Hlavnou nevýhodou týchto techník je však to, že ich výkon do veľkej miery závisí od štrukturálnej podobnosti medzi cieľovým a zdrojovým textom (bitextom).



Obrázok 51 Prehľad algoritmov na zarovnanie textu

2.3.2 Gale a Church algoritmus

Ide o jeden z prvých algoritmov na automatické zarovnanie textov. Hoci nepatrí k najlepším, je jazykovo nezávislý.

Algoritmus využíva skutočnosť, že dlhšie vety v jednom jazyku majú tendenciu byť preložené do dlhších viet v druhom jazyku a naopak. Kratšie vety majú tendenciu byť preložené do kratších viet. Každej navrhovanej dvojici viet sa priradí pravdepodobnostné skóre na základe pomeru dĺžok oboch viet (v znakoch) a rozptylu tohto pomeru. Toto pravdepodobnostné skóre sa používa v rámci dynamického programovania s cieľom nájsť maximálne pravdepodobné zarovnanie viet.

Dynamické programovanie sa používa na zarovnanie dvoch sekvencií znakov v rôznych prostrediach, ako sú sekvencie genetickéh kódu rôznych druhov, rečové sekvencie od rôznych hovoriacich, sekvencie plynových chromatografov od rôznych zlúčenín a geologické sekvencie z rôznych lokalít. Podrobnosti techník zarovnávania sa v jednotlivých aplikáciách značne líšia, ale všetky používajú mieru vzdialenosti na porovnanie dvoch jednotlivých prvkov v rámci sekvencií a algoritmus dynamického programovania na minimalizáciu celkovej vzdialenosti medzi zarovnanými prvkami v rámci dvoch sekvencií.

Našou mierou vzdialenosti je odhad - logProb(match I δ), kde δ závisí od l1 a l2, dĺžok dvoch uvažovaných častí textu. Táto miera vzdialenosti je založená na predpoklade, že každý znak v jednom jazyku, L1, dáva predpoklad vzniku náhodného počtu znakov v druhom jazyku, L2. Predpokladáme, že tieto náhodné premenné sú nezávislé a identicky rozdelené s normálnym rozdelením. Model je potom špecifikovaný na základe strednej hodnoty c a rozptylu s2 tohto rozdelenia, c je očakávaný počet znakov v L2 na jeden znak v L1 a s2 je rozptyl počtu znakov v L2 na jeden znak v L1. Definujeme δ ako (l2 - l1c)/ tak, že má normálne rozdelenie so strednou hodnotou nula a rozptylom jedna (aspoň vtedy, keď sú dve uvažované časti textu skutočne vzájomnými prekladmi).

Algoritmus je zhrnutý v nasledujúcej rekurzívnej rovnici. Nech si, i = 1...I, je veta jedného jazyka a tj, j = 1...J, sú preklady týchto viet v druhom jazyku. Nech d je funkcia vzdialenosti opísaná v predchádzajúcej časti a nech D(i,j) je minimálna vzdialenosť medzi vetami s1,...si a ich prekladmi t1,...tj, v rámci

zarovnania s maximálnou dôveryhodnosťou. D(i,j) sa vypočíta minimalizáciou v šiestich prípadoch (substitúcia, vymazanie, vloženie, kontrakcia, expanzia a zlúčenie), ktoré v podstate ukladajú súbor obmedzení sklonu. To znamená, že D(i,j) je definovaný nasledujúcou rekurenciou s počiatočnou podmienkou D(i,j) = 0.

$$D(i,j) = \min \begin{cases} D(i,j-1) + d(0,t_j;0,0) \\ D(i-1,j) + d(s_i,0;0,0) \\ D(i-1,j-1) + d(s_i,t_j;0,0) \\ D(i-1,j-2) + d(s_i,t_j;0,t_{j-1}) \\ D(i-2,j-1) + d(s_i,t_j;s_{i-1},0) \\ D(i-2,j-2) + d(s_i,t_j;s_{i-1},t_{j-1}) \end{cases}$$

Zdroj: https://aclanthology.org/J93-1004.pdf

Moore algoritmus

Ide o algoritmus, ktorý kombinuje techniky adaptované z predchádzajúceho algoritmu na zarovnávanie viet a slov v troj-krokovom postupe. Najprv sa zarovná korpus pomocou modifikovanej verzie Brownovho modelu založeného na dĺžke vety. Potom sa použije technika orezávania pri vyhľadávaní na efektívne nájdenie dvojíc viet, ktoré sa zarovnajú s najvyššou pravdepodobnosťou bez použitia oporných bodov alebo väčších predtým zarovnaných jednotiek. Následne sa použije dvojica viet, ktorým bola priradená najvyššia pravdepodobnosť zosúladenia, na trénovanie modifikovanej verzie translačného modelu IBM Model 1. Nakoniec sa korpus znovu zarovná, pričom pôvodný model zarovnania je rozšírený o IBM Model 1, aby sa vytvorilo zarovnanie založené na dĺžke viet a korešpondencii slov. Konečné vyhľadávanie sa obmedzuje iba na minimálne segmenty zarovnania, ktorým bola podľa pôvodného modelu zarovnania priradená nezanedbateľná pravdepodobnosť, čo zmenšuje veľkosť priestoru vyhľadávania natoľko, že toto zarovnanie je v skutočnosti rýchlejšie ako pôvodné zarovnanie, hoci model je oveľa náročnejší na aplikáciu pre každý segment.

Zdroj: https://link.springer.com/chapter/10.1007/3-540-45820-4_14

Program: https://www.microsoft.com/en-us/download/details.aspx?id=52608

Príkaz: ./align-sents-all.pl <source_text> <target_text>

Hunalign

Hunalign zarovnáva dvojjazyčný text na úrovni viet. Jeho vstupom je tokenizovaný a segmentovaný text v dvoch jazykoch (originál a jeho preklad). V najjednoduchšom prípade je jeho výstupom postupnosť dvojjazyčných dvojíc viet (bitext). V prípade existencie slovníka, ho hunalign kombinuje s informáciou o dĺžke vety podľa Gale-Churchovho algoritmu. Ak slovník neexistuje, najprv sa vráti k informáciám o dĺžke vety a potom na základe tohto zarovnania vytvorí automatický slovník. Potom pri druhom prechode textom zarovná text pomocou automatického slovníka. Podobne ako väčšina zarovnávačov viet, ani hunalign sa nezaoberá zmenami poradia viet, t. j. nedokáže krížovo zarovnávať (segmenty A a B vo východiskovom jazyku by zodpovedali segmentom B' a A' v cieľovom jazyku).

Zdroj: http://mokk.bme.hu/resources/hunalign/

Zdroj k slovníkom: https://github.com/coezbek/hunalign-dict-muse/tree/main/dict[1]

Program: https://github.com/danielvarga/hunalign

BleuAlign

Základnou myšlienkou Bleualignera je použiť výstup zo strojového prekladu (MT) a skóre automatickej metriky BLEU, ktorá charakterizuje podobnosť MT výstupu s referenčným prekladom na nájdenie spoľahlivých zarovnaní, ktoré sa používajú ako oporné body. Medzery medzi týmito opornými bodmi sa potom vyplnia pomocou heuristiky založenej na metrike BLEU a na dĺžke. Tento prístup prekonáva najmodernejšie algoritmy v našej úlohe zarovnania a toto zlepšenie kvality zarovnania sa premieta do lepšej výkonnosti SMT. To znamená, že pokiaľ by sme na zarovnanie paralelného korpusu použili BleuAlign a na takomto korpuse vytvorili model SMT, jeho výsledky by boli lepšie, ako keby sme ho natrénovali na korpuse, ktorý bol zarovnaný predošlými algoritmami.

Algoritmus počíta zarovnanie viet pre paralelný text (bitext). Vstupom pre algoritmus sú dve množiny dokumentov, články alebo odseky oddelené tvrdými

oddeľovačmi a ľubovoľný počet prekladov. Výber a počet tvrdých oddeľovačov závisí od textov, ktoré sa majú zarovnať, ale musia byť spoľahlivé, pretože algoritmus nehľadá zarovnania, ktoré tieto oddeľovače presahujú. Algoritmus bol vyvinutý pre články s rozsahom niekoľkých desiatok strán (500 a viac viet) a pre tieto dĺžky textov je dostatočne rýchly, ale kvôli svojej kvadratickej zložitosti nie je vhodný na spracovanie celých zbierok textov bez použitia tvrdých vymedzovačov.

Hlavný algoritmus zarovnania sa počíta pre každý textový segment medzi dvoma tvrdými vymedzovačmi (vrátane začiatku a konca súboru) a pozostáva z dvoch krokov. Najprv sa identifikuje súbor oporných bodov pomocou metriky BLEU. V druhom kroku sa vety medzi týmito opornými bodmi zarovnajú buď pomocou heuristiky založenej na metrike BLEU alebo algoritmu založeného na dĺžke podľa Galea a Churcha.

Zdroj: https://aclanthology.org/2010.amta-papers.14/

Program: https://github.com/rsennrich/Bleualign

Príkaz: ./bleualign.py -s sourcetext.txt -t targettext.txt --srctotarget sourcetranslation.txt -o outputfile

Spanalign

Chousa a kol.(2020) navrhli metódu automatického zarovnávania viet z nečistých paralelných dokumentov. Najprv formalizovali problém zarovnania viet ako nezávislé predpovede rozptylov v cieľovom dokumente z viet v zdrojovom dokumente. Potom zaviedli metódu celkovej optimalizácie pomocou celočíselného lineárneho programovania s cieľom zamedziť prekrývaniu rozpätí a získať nemonotónne zarovnania. Implementovali predpovedanie rozpätí medzi jazykmi jemným doladením vopred natrénovaných viacjazyčných modelov založených na architektúre BERT a ich trénovaním pomocou pseudoznačených údajov získaných z metódy zarovnania viet bez dohľadu. Zatiaľ čo základné metódy používajú vety a predpokladajú monotónne zarovnanie, táto metóda dokáže zachytiť interakciu token-to-token medzi tokenmi zdrojového a cieľového textu a zvládnuť nemonotónne zarovnanie.

Chousa a kol. (2020) uskutočnili experiment na vyhodnotenie presnosti zarovnania viet s použitím reálneho súboru so šumom, išlo o paralélne novinové

články v angličtine a japončine. Vyhodnotili dva varianty navrhnutej metódy využívajúce rôzne optimalizačné metódy: monotónne DP (Gale a Church, 1993) a ILP (Integer Linear Programming). Ako východiskový variant použili Vecalign zarovnávač (Thompson a Koehn, 2019). Svoju metódu porovnávali aj Utiyama a Isahara (2003), keďže ide o štandardný prístup na vytváranie anglicko-japonských paralelných korpusov. Na vyhodnotenie metrík použili skóre F1 zarovnania viet, ktoré je jednou zo štandardných metrík pre určenie kvality zarovnania viet. Skóre bolo vypočítané podľa správnych a predpovedaných párov zarovnania. Táto metrika však priamo nehodnotila presnosť extrakcie neparalelných viet, hoci v zašumených dvojjazyčných dokumentoch je týchto neparalelných viet veľa. Preto pre ďalšiu podrobnú analýzu vypočítali aj skóre Precision/Recall/F1 pre každý počet zdrojových a cieľových viet v pároch zarovnania na základe ich implementácie.

Zdroj: https://aclanthology.org/2020.coling-main.418/

Program: https://github.com/nttcslab-nlp/spanalign

Vecalign

Vecalign je presný algoritmus na zarovnávanie viet, ktorý je rýchly aj pri veľmi dlhých dokumentoch. V spojení so systémom LASER funguje Vecalign v približne 100 jazykoch (t. j. 100^2 jazykových párov) bez potreby systému strojového prekladu alebo lexikónu (slovníka).

Vecalign používa na posúdenie podobnosti viet podobnosť viacjazyčných vnorení a aproximáciu dynamického programovania založenú na rýchlom dynamickom časovom oscilovaní, ktorá je lineárna v čase a v priestore vzhľadom na počet zarovnávaných viet.

Vecaling navrhuje novú skórovaciu funkciu založenú na podobnosti dvojjazyčných vnorení viet. Metóda počíta skóre podobnosti vložených viet s kosínusovou podobnosťou normalizovanou pomocou náhodne vybraných vložených viet. Potom spriemeruje susedné dvojice vložených viet v oboch dokumentoch a tieto približné vložené vety zarovná, pričom toto zarovnanie iteratívne spresňuje pomocou pôvodných vložených viet a malého priestoru okolo nich.

Na výpočet vložených viet sa použil program LASER4. Tento nástroj je založený na architektúre pre jazykovo agnostické vkladanie viet (Artetxe a Schwenk).

Zdroj: https://aclanthology.org/D19-1136/

Zdroj LASER: https://github.com/facebookresearch/LASER

Program: https://github.com/thompsonb/vecalign

Príkaz: ./vecalign.py --alignment_max_size 8 --src bleualign_data/dev.de --tgt bleualign_data/dev.fr --src_embed bleualign_data/overlaps.de bleualign_data/overlaps.de.emb --tgt_embed bleualign_data/overlaps.fr bleualign_data/overlaps.fr.emb

3 Sumarizácia

Na internete sa nachádza čoraz viac elektronických dokumentov a text obsiahnutý v nich môže byť príliš zdĺhavý a ťažko pochopiteľný. S rozvojom oblasti umelej inteligencie, ktorá sa nazýva spracovanie prirodzeného jazyka boli vyvinuté algoritmy, pomocou ktorých možno text skrátiť, teda sumarizovať. Ako hovorí kapitola 1, Spracovanie prirodzeného jazyka (Natural Language Processing) sa zaoberá algoritmami, ktoré dokážu porozumieť ľudskému jazyku. Typické úlohy, ktorými sa spracovanie prirodzeného jazyka zaoberá sú napríklad určovanie morfologických vlastností textov (napríklad určovanie slovných druhov), prekladanie dokumentov, dopĺňanie textov (napríklad našepkávač vo vyhľadávaní Google) ale aj algoritmami, ktoré dokážu zjednodušiť množstvo dlhého textu do súdržného a plynulého súhrnu. Podoblasť spracovania jazyka, ktorá sa zaoberá zjednodušovaním textu sa nazýva sumarizácia.

3.1 Úvod do sumarizácie

Pomocou sumarizácie textu možno zjednodušiť dlhý text na odseky, vety alebo zachytiť kľúčové frázy. Takýmto spôsobom dokážeme skrátiť čas potrebný na pochopenie dlhých materiálov, ako sú výskumné články, bez vynechania dôležitých informácií. Základom sumarizácie textu je vytvorenie zhrnutia, ktoré možno definovať ako text vytvorený z jedného alebo viacerých dokumentov, ktorý poskytuje dôležité informácie v pôvodných dokuentoch a neprekračuje dĺžku polovice pôvodného dokumentu. Úlohou automatickej sumarizácie textu je vytvoriť stručné a plynulé zhrnutie, ktoré bude zachovávať obsah kľúčových informácií a celkového významu. Automatická sumarizácia sa využíva napríklad pri generovaní úryvkov článkov napríklad na spravodajských weboch. Automatická sumarizácia textu je veľmi náročná. Keď sa človek pokúša o zhrnutie textu, zvyčajne si ho prečíta celý a na základe toho, ako mu porozumel dokáže napísať zhrnutie. Počítače však nedokážu rozmýšľať o texte rovnako ako človek, a preto je sumarizácia náročná a netriviálna úloha.

Sumarizácia textu je dôležitou úlohou, ktorá v dátovej vede umožňuje odstrániť prebytočný informačný šum, vďaka čomu môžeme pracovať len s podstatnými informáciami z pôvodných textov. Existujú štyri kritéria, podľa ktorých možno posúdiť kvalitu sumarizácie:

- **Pokrytie informácií** - Pokrytie informácií hovorí o tom, akú časť dôležitých informácií v texte sumarizácia dokázala obsiahnuť.
- **Súdržnosť sumarizácie** - Súdržnosť sumarizácie je miera, ktorá vyjadruje vzťah a nadväznosť medzi vetami.
- **Minimalizácia redundancie** - Minimalizovaním redundacie sa rozumie minimalizovanie duplicitných informácií v sumarizácií.
- **Stručnosť** - Stručnosť je metrika, ktorá vyjadruje koľko slov sumarizácia potrebuje pre obsiahnutie dôležitých informácií.

3.2 Prístupy k sumarizácií

Existujú dva hlavné prístupy k úlohe sumarizácie – extrakcia a abstrakcia. Extrakcia je založená na tom, že z existujúceho dokumentu vyberie najdôležitejší obsah, ktorý v dokumente existuje, zatiaľ čo abstrakčné prístupy dokážu generovať nové vety.

3.2.1 Extraktívna sumarizácia

Extraktívna sumarizácia pracuje na takom princípe, že rozhoduje o tom, ktoré vety z textu sú významné a je potrebné ich zahrnúť do zhrnutia. K tomuto účelu sa používa tzv. bodovanie viet, čo znamená, že každej vete sa priradí skóre a následne sa vety zoradia. Vety, ktoré majú najvyššie skóre musia byť zahrnuté do zhrnutia.

Zjednodušene povedané, ak použijeme extraktívny prístup, pokúsime sa nájsť najrelevantnejšie informatívne vety v dokumente, potom ich „vytiahneme“ z textu a znova ich skombinujeme, aby sme vytvorili novú kratšiu verziu pôvodného textu. Pri tomto prístupe sa nevytvoria žiadne nové vety, ktoré by predtým v texte neexistovali. Extrahované vety z pôvodného dokumentu sa len prekombinujú. Extraktívna sumarizácia môže využívať niekoľko metód:

- **TF-IDF (Term Frequency - Inverse Document Frequency) -** Táto štatistická metóda sa používa na posúdenie dôležitosti slov. Frekvencia výrazov sa používa na zistenie, koľkokrát sa výraz vyskytuje v dokumente. Frekvencia výrazov ako „the“ môže byť veľmi vysoká. Inverzná frekvencia dokumentov sa vypočíta ako logaritmus celkového počtu dokumentov vydelený celkovým počtom dokumentov, v ktorých sa termín vyskytuje. Inverzná frekvencia dokumentu výrazu môže byť nízka, aj keď je jeho frekvencia výrazov veľmi vysoká.
- **Metódy založené na grafoch -** Pri tejto technike sa vytvorí graf. Uzly grafu predstavujú vety. Okraje grafu symbolizujú spojenie medzi vetami, ktoré zdieľajú rovnaké slová. Uzly, ktoré majú viac hrán, obsahujú dôležité vety a majú väčšiu prioritu pri sumarizácii.
- **Princípy využívajúce strojové učenie -** S využitím strojového učenie môžeme na sumarizáciu textu nazerať ako na dvojtriedny klasifikačný problém. Vety sú zoskupené do súvetí a nesúhrnných viet. Sumarizátor je trénovateľný, súbor trénovacích údajov a ich extrakčné súhrny sa používajú ako referencia.
- **LSA (Latent Semantic Analysis) -** LSA je robustná algebricko-štatistická metóda, ktorá extrahuje skryté sémantické štruktúry slov a viet, teda extrahuje vlastnosti, ktoré nemožno priamo spomenúť. Tieto funkcie sú nevyhnutné pre údaje, ale nie sú pôvodnými funkciami množiny údajov. Je to prístup bez dozoru spolu s používaním spracovania prirodzeného jazyka (NLP).
- **Metódy využívajúce neurónové siete –** Neurónové siete sa snažia napodobniť činnosť ľudského mozgu pri učení sa. Podobne ako ľudský mozog obsahujú neuróny, pomocou ktorých spracovávajú údaje. Neurónová sieť sa teda snaží rozmýšľať podobne ako človek k tomu, aby dokázala posúdiť, ktoré z viet sú dôležité a majú byť zahrnuté do súhrnu. Pomocou trénovacích dát sa snaží naučiť typy viet, ktoré by mali byť zahrnuté do súhrnu. Keď sa sieť naučí vlastnosti, ktoré musia existovať v súhrnných vetách, musíme určiť trendy a vzťahy medzi vlastnosťami, ktoré sú vlastné väčšine viet.
- **Metódy založené na Fuzzy logike -** Táto metóda považuje každú charakteristiku textu, ako je dĺžka vety, podobnosť s názvom, podobnosť

s kľúčovým slovom atď. za vstup fuzzy systému. Všetky pravidlá potrebné na sumarizáciu sú tiež vstupom do znalostnej bázy. Každá veta získa skóre v rozsahu od 0 do 1. Získaná hodnota určuje dôležitosť viet pre generovanie súhrnu.

3.2.2 Abstraktívna sumarizácia

Abstraktívná sumarizácia je inteligentnejšia forma sumarizácie v porovnaní s extraktívnou sumarizáciou, pretože dokáže generovať nové vety. Pri tomto prístupe musíme najskôr vytvoriť prechodnú reprezentáciu vstupného textu. Tá sa obvykle vytvára reprezentáciou témy, teda transformáciou textu s cieľom identifikovať hlavné témy textu a reprezentáciou indikátora, kde súbor „ukazovateľov“ – napr. dĺžky viet alebo viet obsahujúcich určité „indikátorové“ slová vyjadruje dôležitosť časti textu. Jednotlivé vety sa opäť zoradia podľa toho, aké získali skóre a na zostavenie súhrnu sa použijú vety s najvyšším skóre.

Abstraktívna sumarizácia sa pokúša porozumieť textu pomocou pokročilých techník spracovania prirodzeného jazyka a vytvoriť nové vety parafrázovaním tak, ako by to urobil človek. Tento prístup je podstatne zložitejší, nakoľko vyžaduje sémantické porozumenie textu a väzbu medzi pojmami, kontextom a témami. Podobne ako pri extraktívnej sumarizácií aj abstraktívnu sumarizáciu možno docieliť niekoľkými metódami:

- **Metódy založené na stromoch -** Hlavnou myšlienkou tejto skupiny metód je použitie stromu závislostí, ktorý predstavuje text alebo obsah dokumentu. Príkladom takéhoto algoritmu je fúzia viet, ktorá dokáže spracovávať viacero dokumentov.
- **Metódy založené na šablónach -** Pri týchto metódach je celý dokument reprezentovaný pomocou určitého sprievodcu. Jazykové vzory alebo pravidlá extrakcie sa priraďujú k bodovým textovým úryvkom, ktoré možno namapovať do vodiacich slotov (na vytvorenie databázy).

- **Metódy založené na pravidlách -** Metódy založené na pravidlách zobrazujú vstupné dokumenty z hľadiska tried a zoznamov aspektov. Na vytvorenie pravidiel extrakcie sa identifikujú slovesá a podstatné mená s podobným významom. Vyberie sa niekoľko pravidiel kandidátov, ktoré sa prenesú do súhrnu.
- **Metódy založené na grafoch -** Podobne ako pri extraktívnej sumarizácií aj abstraktívna sumarizácia môže byť dosiahnutá s využitím grafov.
- **Metódy založené na ontológií -** Ontológie sú v NLP mimoriadne populárne, vrátane extrakčných aj abstraktných sumarizácií, kde sú vhodné, pretože sa zvyčajne obmedzujú na rovnakú tému alebo doménu. Okrem toho má každá doména svoju vlastnú znalostnú štruktúru, ktorú možno lepšie znázorniť pomocou ontológie. Aj keď sa líšia vo svojich špecifických prístupoch, všetky metódy sumarizácie založené na ontológii zahŕňajú redukciu viet komprimovaním a preformulovaním pomocou lingvistických aj NLP techník. Typickým zástupcom je Fuzzy ontológia.
- **Multimodálny sémantický model -** Sémantický model je spočiatku vytvorený pomocou reprezentácie znalostí založenej na objektoch. Uzly predstavujú pojmy a väzby medzi týmito pojmami predstavujú vzťah medzi nimi. Dôležité nápady sa hodnotia pomocou metriky hustoty informácií, ktorá kontroluje úplnosť, vzťah s ostatnými a počet výskytov výrazu. Vybrané pojmy sú nakoniec transformované do viet, aby vytvorili zhrnutie.
- **Model reprezentácie sémantického textu -** Cieľom tejto techniky je analyzovať vstupný text pomocou sémantiky slov, a nie pomocou syntaxe alebo štruktúry textu.

3.2.3 Kombinovaná sumarizácia

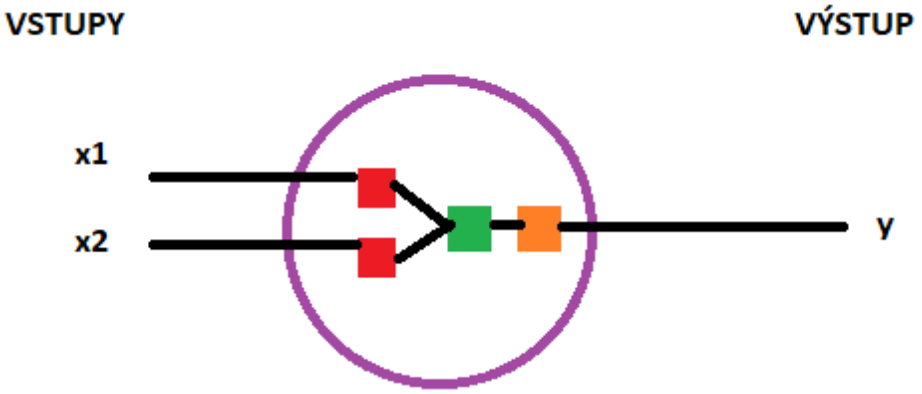
Existujú kombinované prístupy, ktoré využívajú abstraktný generátor. Abstraktný generátor prijíma ako vstup text, ktorý pochádza z extraktívneho sumarizátora. Kombinovaná sumarizácia je efektívnejšia, pretože pracuje s textom, ktorý je už zbavený všetkých nadbytočných a nepodstatných informácií. Mnoho súčasných algoritmov a medzi nimi aj algoritmus BART je založených na tomto prístupe.

3.2.4 Koncepty a algoritmy využívané v sumarizácií textu

Aby sme pochopili princíp činnosti jednotlivých algoritmov, ktoré slúžia ku sumarizácií textu, musíme sa najskôr oboznámiť s neurónovými sieťami. Ľudský mozog obsahuje desiatky tisíc mozgových buniek, ktoré sa nazývajú neuróny. Tieto neuróny sú medzi sebou pospájané, dokážu medzi sebou komunikovať a vytvárajú komplexné štruktúry. V umelej inteligencií existujú metódy, ktorá učí počítače spracovávať dáta podobne ako ľudský mozog. Tieto metódy sa nazývajú neurónové siete.

3.2.5 Neurón

Skôr, než si vysvetlíme architektúru neurónových sietí je potrebné sa oboznámiť s tým, ako pracuje neurón. Najjednoduchšou možnou implementáciou je neurón s dvoma vstupmi, podobne, ako na obrázku 38, kde x_1 x_2 predstravujú vstupy a y predstavuje výstup. Tematiku neurónov si viac priblížime v kapitole 5.



Obrázok 52 Neurón

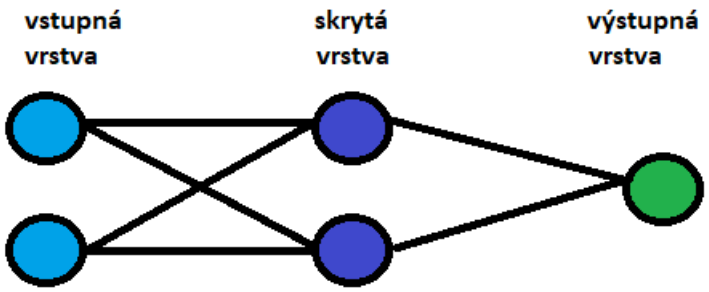
Dejú sa tu tri veci:

- Najskôr sa každý vstup vynásobí nejakou váhou *w*.
- Vážené vstupy sa spočítajú a pripočíta sa k ním tzv. bias (nejaká konštanta)
- Výsledok z druhého kroku sa vloží do aktivačnej funkcie, ktorá sa používa na premenu výsledku vstupu na výstup, ktorý má peknú, čitateľnú formu. Bežne používaná aktivačná funkcia je **sigmoidná funkcia**, ktorá prevedie číslo na hodnotu od 0 do 1.

3.2.6 Neurónové siete

Ak spojíme viaceré neuróny dohromady, hovoríme o tzv. neurónových sieťach. Základná neurónová sieť (Obrázok 53) má prepojené umelé neuróny v troch vrstvách:

- **Vstupná vrstva** - Ide o naše pôvodné vstupy, podobne ako na predchádzajúcom obrázku x_1 a x_2.
- **Skrytá vrstva** - Skrytá vrstva analyzuje výstup zo vstupnej vrstvy. Ak má neurónová vrstva niekoľko skrytých vrstiev hovoríme o **hlbokých neurónových sieťach.**
- **Výstupná vrstva** - Výstupná vrstva dáva konečný výsledok.



Obrázok 53 Základná neurónová sieť

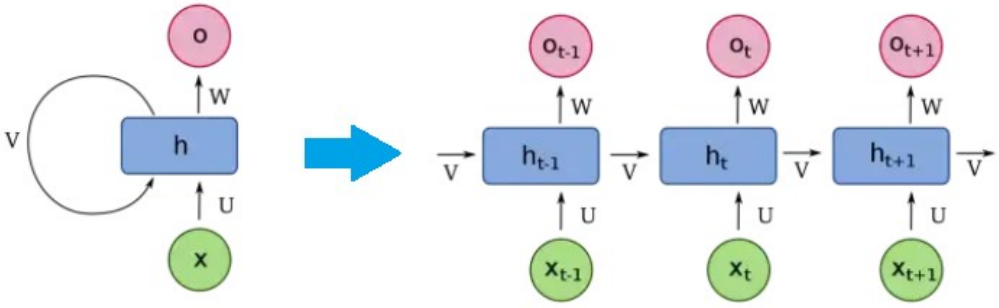
Existuje niekoľko typov neurónových sietí. Popíšeme si tie najznámejšie z nich.

Dopredné neurónové siete

Dopredné neurónové siete spracúvajú dáta v jednom smere, od vstupnej vrstvy k výstupnej. Každý uzol v jednej vrstve je spojený s každým ďalším v nasledujúcej vrstve. Podrobne si ich vysvetlíme v kapitole 6.

Rekurentné neurónové siete (RNN)

Rekurentné neurónové siete (Obrázok 54) sú špeciálne architektúry, ktoré berú do úvahy časové informácie. Skrytý stav neurónovej siete v čase *t* preberá informácie zo vstupu v čase *t* a aktivácií zo skrytých jednotiek v čase *t*-1 na výpočet výstupov pre čas *t*. To je možné vidieť na obrázku nižšie. Takýmto spôsobom si rekurentná neurónova sieť dokáže pamätať si predchádzajúce vstupy a ich výstupy.



Obrázok 54 Rekurentná neurónová sieť

Zapamätať si predchádzajúce vstupy je obzvlášť dôležité pri úlohách spracovania prirodzeného jazyka, a to z toho dôvodu, že vstupné slová nemajú rovnakú veľkosť a nasledujúce slovo je veľmi závislé od predchádzajúcich slov. Rekurentné neurónové siete si teda dokážu zapamätať kontext.

Siete s dlhou krátkodobou pamäťou (LSTM)

Pamäť RNN je krátka, čo môže predstavovať problém. Optimalizáciou tohto problému bolo vytvorenie sietí s dlhšou pamäťou, ktoré sa nazývajú LSTM a používajú tzv. stav bunky. Tento stav bunky je stav v akomkoľvek čase a aktualizuje sa o relevantné informácie v každom časovom kroku. Výstup v každom časovom kroku je odvodený zo vstupu, predchádzajúceho výstupu a aktualizovaného stavu bunky.

Transformátorové siete

Zavedením LSTM a ich schopnosti zapamätať si stav buniek sa pamäť neurónových sietí zlepšila, no stále bola obmedzená. Na riešenie problémov s pamäťou boli vyvinuté transformátorové siete, ktoré zavádzajú koncept blokov pozornosti. Bloky pozornosti vypočítali, ako každé slovo vo vstupe súvisí s inými slovami vo vstupe. Čím vyššia hodnota, tým väčšia pozornosť sa venuje týmto slovám a tým závislejšia je množina slov. Pozornosť zvyšuje počet kontextových spojení, ktoré môže sieť vytvoriť, a sieť sa môže naučiť vzťahy a kontext z veľkých množín údajov.

3.2.7 Algoritmy využívané pri sumarizácií textu

V súčasnosti, keďže ide o oblasť, ktorá prináša skutočne užitočné výsledky, sa používa a neustále zdokonaľuje mnoho rôznych algoritmov.

3.2.7.1 GPT-3

GPT-3 je automatický regresívny algoritmus umelej inteligencie vyvinutý spoločnosťou OpenAI , výskumným laboratóriom poháňaným AI so sídlom v San Franciscu v Kalifornii. Ide o masívnu umelú neurónovú sieť, ktorá využíva hlboké učenie na generovanie ľudského textu a je trénovaná na

obrovských textových súboroch s tisíckami miliárd slov. Je to model predikcie jazyka AI tretej generácie v sérii GPT-n a nástupca GPT-2.

Tento algoritmus umelej inteligencie je program, ktorý dokáže vypočítať slovo alebo dokonca znak, ktorý sa musí objaviť v texte vo vzťahu k slovám okolo neho. Toto sa nazýva podmienená pravdepodobnosť slov. Je to generatívna neurónová sieť, ktorá umožňuje číselné skóre alebo odpoveď áno alebo nie. Ako výstup generuje aj dlhé sekvencie pôvodného textu. Celkový počet váh, ktoré OpenAI GPT-3 dynamicky uchováva vo svojej pamäti a využíva na spracovanie každého dopytu, je 175 miliárd.

3.2.7.2 BERT

BERT (obojsmerná reprezentácia kódovača pre transformátory) využíva úplný obojsmerný prístup bez dozoru a je vopred pripravený iba na čistý textový korpus (Wikipedia). Obojsmerný prístup znamená že slová vo vete posudzuje nie len zľava do prava a zhora nadol, ako by to robil človek ale aj v opačnom smere. Zaujímavosťou tohto algoritmu je, že učenie sa pomocou maskovacej funkcie („modelovanie maskovaného jazyka“: niektoré slová sú zamaskované do vety) a potom musí BERT predpovedať, ktoré z nich je chýbajúce slovo alebo či veta nasleduje za inou vetou. BERT používa mechanizmus pozornosti, ktorý je schopný naučiť sa kontextové vzťahy medzi slovami v texte. Nižšie si stručne popíšeme algoritmus BART, ktorý je v súčasnosti najmodernejším v oblasti sumarizácie a je odvodený od BERT.

3.2.7.3 BART

BART (obojsmerná reprezentácia autoenkódera pre transformátory). Algoritmus BART zovšeobecňuje prístup GPT aj BERT, pričom berie to najlepšie z týchto dvoch modelov. BART je trénovaný na poškodzovanie textu pomocou funkcie šumu (ktorá pridáva k textu „šum“, nielen masky) a potom sa učí model, aby znovu získal pôvodný text. Je založený na architektúre prekladu neurónového stroja založenej na transformátore s obojsmerným kódovačom (ako BERT) a dekodérom zľava doprava (ako GPT). Algoritmus BART mapuje poškodené dokumentu do vstupného dokumentu a možno ho použiť na

akýkoľvek typ poškodenia dokumentu (maskovanie tokenu, vymazanie tokenu, vyplnenie textu, permutácia viet, rotácia dokumentu atď.). Algoritmus BART dosahuje nové, najmodernejšie výsledky v oblasti abstraktného dialógu, generovania textu, odpovedí na otázky a sumarizačných úloh.

3.3 Extrakcia kľúčových slov

Extrakcia kľúčových slov je jednou z techník sumarizácie, ktorá slúži k zachyteniu najdôležitejších slov alebo fráz z dokumentu. Pomocou tejto techniky je možné extrahovať malý súbor jednotiek zložených z jedného alebo viacerých fráz. Kľúčové slová frázy zohrávajú dôležitú úlohu pri rýchlom získavaní myšlienky textových údajov bez toho, aby bolo potrebné čítať celý text. Táto technika sumarizácie textu nachádza uplatnenie v oblasti správy obsahu, ako je optimalizácia pre vyhľadávače, reklama a systémy odporúčaní pre používateľov. Napríklad pri návšteve reklamy alebo webovej stránky sú koncoví používatelia priťahovaní, ak sú kľúčové slová relevantné pre ich potreby.

Prístupy k extrakcií kľúčových slov možno najjednoduchšie rozdeliť do dvoch základných skupín, a to jednoduché štatistické prístupy a prístupy založené na strojovom učení. Ak by sme sa chceli na extrakciu kľúčových slov pozrieť podrobnejšie, mohli by sme princípy extrakcie rozdeliť do piatich kategórií, a to jednoduché štatistické prístupy, prístupy založené na grafe, lingvistické prístupy, prístupy založené na strojovom učení a hybridné prístupy (Beliga, 2014).

3.3.1 Štatistické prístupy

Štatistické prístupy extrahujú kľúčové slová tak, že pri tom využívajú štatistické funkcie, ako napríklad TF-IDF (Term Frequency-Inverse Document Frequency), n-gramová štatistika, spoločné výskyty slov a iné štatistiky. Väčšina štatistických prístupov je jazykovo nezávislá, čo znamená, že ich možno použiť pre texty v akomkoľvek jazyku v prípade, že je k dispozícií dostatočne veľký korpus. Okrem použiteľnosti na ľubovoľný jazyk je nespornou výhodou štatistických prístupov rýchlosť. Takéto algoritmy sú podstatne rýchlejšie na rozdiel od prístupov, ktoré sú založené na strojovom učení.

TF-IDF (Term Frequency – Inverse Document Frequency)

TF-IDF je jedným z najznámejších možných prístupov k zisteniu dôležitých slov z dokumentu. TF-IDF hovorí o dôležitosti slov v dokumente vzhľadom na celý korpus. Túto metódu sme si opísali bližšie v podkapitole 1.5.3.

RAKE (Rapid Automatic Keyword Extraction)

RAKE sa teší najväčšej obľube medzi algoritmami extrakcie kľúčových slov založených na štatistike. Myšlienka tohto algoritmu je, že kľúčové slová často obsahujú viacero slov, ale zriedka obsahujú interpunkciu, stop slová alebo iné slová s minimálnym lexikálnym významom. Algoritmus je založený predovšetkým na spoločnom výskyte slov, napríklad pri extrakcií kľúčových slov zo spätnej väzby zákazníkov na konkrétny telefón by kľúčovú frázu mohol predstavovať bigram ako „dobry fotoaparat“, „kvalitny zvuk.“ Tieto slová v doméne spätnej väzby konkrétneho produktu sa často vyskytovali spoločne. Je to kolokácia. Vstupom do algoritmu je text očistený od stopových slov a interpunkcie. Algoritmus následne vypočíta maticu spoločných výskytov (Obrázok 55).

	feature	extraction	complex	algorithm	available	help	rapid	automatic	keyword
feature	2	2	0	0	0	0	0	0	0
extraction	2	3	0	0	0	0	0	1	1
complex	0	0	1	0	0	0	0	0	0
algorithm	0	0	0	1	1	0	0	0	0
available	0	0	0	1	1	0	0	0	0
help	0	0	0	0	0	1	0	0	0
rapid	0	1	0	0	0	0	1	1	1
automatic	0	1	0	0	0	0	1	1	1
keyword	0	1	0	0	0	0	1	1	1
TOTAL SUM	**4**	**8**	**1**	**2**	**2**	**1**	**3**	**4**	**4**

2 kandidátne kľúčové frázy pre slovo feature

Obrázok 55 RAKE

Každému slovu sa následne pridelí skóre. Vypočíta sa stupeň slova v matici - súčet počtu spoločných výskytov vydelený frekvenciou ich výskytu. Frekvencia výskytu znamená, koľkokrát sa slovo vyskytuje v korpuse (Obrázok 56).

Word	Degree OfWord	Word Frequency	Degree Score
feature	4	2	2
extraction	8	3	2.66
complex	1	1	1
algorithm	2	1	2
available	2	1	2
help	1	1	1
rapid	3	1	3
automatic	4	1	4
keyword	4	1	4

Obrázok 56 Výpočet metrík

Finálne skóre pre identifikované kľúčové frázy budú súčtom skóre jednotlivých slov, ktoré kľúčová fráza obsahuje. V prípade kľúčovej frázy „feature extraction“ bude teda hodnota rovná 4,66.

Implementácia algoritmu RAKE

K implementácií algoritumu RAKE si budeme najskôr musieť nainštalovať knižnicu nltk, rake-nltk. Po inštalácií si môžeme knižnice naimportovať.

Zoznamy stop slov sú k dispozícií na rôznych internetových stránkach. Mohli by sme si stiahnuť ľubovoľný z nich a implementovať ho do nášho kódu ako list. Môžeme však využiť aj zoznam stop slov, ktoré ponúka knižnica nltk.V našom prípade si ukážeme extrakciu kľúčových slov z jednoduchého textu, ktorý budeme mať uložený v premennej typu string. Tento text budeme musieť tokenizovať na vety, k čomu využijeme Punkt Sentence Tokenizer, ktorý rozdeľuje text na zoznam viet.

Majme nasledovný text: "*Text summarization is a methond which belongs to the area of Natural Language Processing. Keyword extraction is a process of obtaining the most important keywords in document. Keyword extraction is usefull text sumarization technique.*" Uložme si tento text ako premennú typu string. Preveďme si rovno tento text na malé písmená. Do premennej stop_words si uložme zoznam našich stop slov (Obrázok 57).

```
[1] !pip install nltk
    !pip install rake-nltk

    import nltk
    from rake_nltk import Rake

[3] nltk.download('stopwords')
    from nltk.corpus import stopwords
    nltk.download('punkt')

    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Unzipping corpora/stopwords.zip.
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    True

[4] text = '''Text summarization is a method which belongs to the area of Natural Language Processing.
    Keyword extraction is a process of obtaining the most important words in document. Keyword extraction
    is usefull text summarization technique.'''
    text = text.lower()

    stop_words = nltk.corpus.stopwords.words('english')
```

Obrázok 57 Načítanie knižnice a predspracovanie

Do premennej rake_extractor si inicializujeme triedu Rake, ktorá bude extrakciu vykonávať. Parameter stopwords určuje zoznam slov, ktoré sa z textu odstránia. Rozsah n-gramov, teda počet slov, ktoré chceme aby naše kľúčové slová obsahovali určuje parameter min_length, ktorý definuje minimálny počet slov, ktoré frázy musia obsahovať a parameter max_length, ktorý definuje maximálny počet slov, ktoré môžu extrahované kľúčové frázy obsahovať. V našom prípade chceme frázy, ktoré majú presne dve slová. Parameter include_repeated_phrases určuje, či chceme aby sa nám vo výsledku opakovali extrahované kľúčové slová. Následne zavoláme funkciu extract_keywords_from_text, ktorá prijme ako parameter našu premennú s názvom text.

Pre získanie kľúčových slov využijeme metódu get_ranked_phrases alebo get_ranked_phrases_with_scores podľa toho, či chceme vidieť aj

bodovacie skóre pre naše kľúčové slová. Naše kľúčové slová sú teda text summarization, keyword extraction a important words (Obrázok 58).

```
[5]  rake_extractor = Rake(stopwords = stop_words, min_length=2, max_length=2, include_repeated_phrases=False)
     rake_extractor.extract_keywords_from_text(text)

[6]
     rake_extractor.get_ranked_phrases_with_scores()
     rake_extractor.get_ranked_phrases_with_scores()

     [(4.0, 'text summarization'),
      (4.0, 'keyword extraction'),
      (4.0, 'important words')]
```

Obrázok 58 Zavolanie extraktora a výpis kľúčových slov

KP-miner

Existujú komplexnejšie metódy extrakcie kľúčových slov, ktoré využívajú TF-IDF iba ako štatistickú metódu na výpočet dôležitosti kľúčových fráz. Patrí sem napríklad metóda KP-miner, ktorá je rozdelená do troch krokov. Prvým krokom je vybrať kandidátske slová z dokumentov, druhým krokom je vypočítať skóre kandidátskych slov a tretím krokom je vybrať kandidátske slovo s najvyšším skóre ako konečnú kľúčovú frázu. KP-miner zaviedol dve nové štatistické funkcie vo fáze výberu kandidátskeho slova. Faktor najnižšej prípustnej frekvencie videnia (Least Allowable Seen Frequency) znamená, že za kandidátske slová možno považovať iba slová, ktoré sa v dokumente vyskytujú viac ako n-krát. Druhá štatistická funkcia zavedená KP-minerom sa nazýva CutOff a je založená na skutočnosti, že ak sa slovo objaví po danej prahovej pozícii v dlhom dokumente, nebude to kľúčová fráza, čo znamená, že slovo, ktoré sa objaví po CutOff, bude odfiltrované. Nakoniec sa konečné kľúčové frázy vyberú kombináciou pozícií kandidátskych slov a skóre TF-IDF.

Implementácia algoritmu KP-miner

Prvým krokom je nainštalovanie a importovanie potrebnej knižnice. Metóda load_document prijíma ako parametre vstupný text a jeho jazyk. Následne zavoláme metódu candidate_weighting a v poslednom kroku vypíšeme kľúčové frázy pozostávajúce z dvoch slov (Obrázok 59).

```
[12] !pip install git+https://github.com/boudinfl/pke.git
     import pke

[13] kp_extractor = pke.unsupervised.KPMiner()
     kp_extractor.load_document(text, language='en')

[15] kp_extractor.candidate_weighting()
     keywords = kp_extractor.get_n_best(n=2)
     keywords
```

Obrázok 59 KP-miner

YAKE

YAKE je typická metóda extrakcie kľúčových fráz s využitím TF-IDF. Rozdiel medzi YAKE a KP-miner spočíva v tom, že YAKE používa umiestnenie kandidátskych slov alebo informácie TF-IDF a zavádza novú sadu piatich funkcií. Veľkosť písmen *WC* odráža veľkosť písmen kandidátskych slov. Premenná *WP* vo vzorci odráža pozíciu slova, čo znamená, že čím častejšie je slovo v prednej časti dokumentu, tým väčšia je jeho hodnota. Frekvencia slov je *WF* vyjadruje, že čím vyššia je frekvencia slova v dokumente, tým väčšia je jeho hodnota. Súvislosť slov s kontextom *WRC* označuje počet rôznych slov vyskytujúcich sa na oboch stranách kandidátskeho slova. Word DifSentence *WD* označuje frekvenciu kandidátskeho slova v rôznych vetách. Týchto päť hodnôt sa skombinuje na výpočet S(w), ako je znázornené vo vzorci (19).

$$S(w) = \frac{WR * WP}{WC + \frac{WF}{WRC} + \frac{WD}{WR}} \tag{19}$$

Nakoniec sa vypočíta konečné *S*(*kw*) každého kandidátskeho slova pomocou 3-gramového modelu, ako je znázornené v nasledovnej rovnici (20), kde *kw* predstavuje kandidátske slovo a *TF* predstavuje frekvenciu kľúčovej frázy. Čím menšia je hodnota *S*(*kw*), tým je pravdepodobnejšie, že *kw* bude kľúčová fráza.

$$S(w) = \frac{\pi_{w \in kw} S(w)}{TF(kw) * (1 + \sum w \in kw\ (Sw)} \tag{20}$$

Implementácia algoritmu YAKE

Pre implementáciu algoritmu si potrebujeme stiahnuť a naimportovať príslušnú knižnicu. Základná implementácia algoritmu je jednoduchá, stačí definovať jazyk, v ktorom sa náš text nachádza a maximálny počet n-gramov (Obrázok 60).

```
[ ]  !pip install yake
     import yake

     yake_extractor = yake.KeywordExtractor(lan="en", n=2)
     keywords = yake_extractor.extract_keywords(text)
     for kw in keywords:
         print(kw)
```

Obrázok 60 Yake

3.3.2 Prístupy založené na grafe

Všetky prístupy založené na grafoch počítajú dôležitosť vrcholu v grafe, pričom sa nespoliehajú len na informácie, ktoré sú špecifické pre lokálny vrchol, ale berú do úvahy aj globálne informácie, ktoré sú pomocou rekurzie vypočítané z celého grafu. Základom mnohých algoritmov založených na grafe je algoritmus PageRank

PageRank

PageRank je algoritmus, ktorý bol vyvinutý ako systém hodnotenia webových stránok podľa na základe množstva a kvality odkazov, ktoré na ňu odkazujú. PageRank sa počíta podľa vzorca (21), kde *A* predstavuje stránku, pre ktorú sa počíta PageRank a T_1 až T_n sú stránky, ktoré na stránku *A* odkazujú. *PR* predstavuje PageRank stránky, *C* predstavuje odchádzajúce odkazy zo stránky a *d* je faktor tlmenia, ktorá sa postará o zníženie prílišného vplyvu určitej stránky, čo môže mať za následok používanie falošných robotov. 1-*d* je faktor, ktorá sa stará o každú novú stránku, na ktorú neukazuje žiadny odkaz.

$$PR(A) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + \cdots + \frac{PR(T_n)}{C(T_n)}\right) \quad (21)$$

TextRank

Najznámejším algoritmom extrakcie kľúčových slov, ktorý je založený na grafe je TextRank, ktorý vychádza z algoritmu PageRank. TextRank v podstate aplikuje algoritmus PageRank na graf, kde vrcholy zodpovedajú slovám. Dôležitým aspektom TextRank je, že nevyžaduje hlboké jazykové znalosti, ani anotované korpusy špecifické pre danú doménu alebo jazyk, vďaka čomu je vysoko prenosný do iných domén, žánrov alebo jazykov.

Pôvodný algoritmus PageRank predpokladá nevážený graf. Ale grafy pre TextRank sú vytvorené z textu v prirodzenom jazyku, a preto by zahŕňali veľa prepojení medzi tokenmi extrahovanými z textu. Preto by bolo užitočné zahrnúť silné stránky týchto vzťahov medzi vrcholy grafu. Z tohto dôvodu sa TextRank aplikuje na vážený graf.

Zaujímavou vlastnosťou algoritmu TextRank je, že obsahuje taktiež črty lingvistických prístupov. Prvým krokom algoritmu je totižto tokenizovať text a následne ho anotovať POS tagmi. Autori TextRank vykonali experimenty a pozorovali najlepšie výsledky, keď zvažovali iba podstatné mená a prídavné mená. Vrcholy, ktoré prejdú syntaktickým filtrom, v ktorom majú značky POS iba podstatné meno alebo prídavné mená, sa pridajú do grafu ako vrcholy.

Implementácia TextRank

Do premennej pos si definujeme tagy, ktoré nás zaujímaju. Metóda load_document prijíma ako parametre vstupný text a jeho jazyk. Následne v metóde candidate_weighting vytvoríme grafovú reprezentáciu tak, že do parametra pos priradíme našu premennú pos a povieme, že nás zaujíma horných 33% (Obrázok 61).

```
[ ]  pos = {'NOUN', 'PROPN', 'ADJ'}
     text_rank_extractor = pke.unsupervised.TextRank()
     text_rank_extractor.load_document(input=text,language='en')
     text_rank_extractor.candidate_weighting(pos=pos, top_percent=0.33)
     keywords = text_rank_extractor.get_n_best(n=2)
```

Obrázok 61 TextRank

SingleRank

Ďalším algoritmom extrakcie kľúčových slov, ktorý je založený na grafe je SingleRank, ktorý rozširuje TextRank s dvoma hlavnými rozdielmi. Rovnako ako v algoritme TextRank aj pri SingleRank vrcholy prejdú syntaktickým filtrom a hrany sú tiež priradené na základe spoločného výskytu slov v okne. Prvým hlavným rozdielom je, že týmto hranám je priradená váha na základe vzdialenosti medzi dvoma slovami, ktoré sa nachádzajú v preddefinovanom okne. Druhý rozdiel spočíta v počte vrcholov, ktoré si ponechá ako potenciálne kľúčové slová po spustení algoritmu PageRank. SingleRank zachováva všetky slová, zatiaľ čo pri TextRanku je to obvykle horných 30%.

Implementácia algoritmu SingleRank

Pri implementácií algoritmu SingleRank môžeme využiť knižnicu, ktorú sme si nainštalovali v predošlom kroku. Postup bude identický ako pri implementácií algoritmu TexRank (Obrázok 62).

```
[ ]  pos = {'NOUN', 'PROPN', 'ADJ'}
     single_rank_extractor = pke.unsupervised.SingleRank()
     single_rank_extractor.load_document(input=text,language='en')
     single_rank_extractor.candidate_weighting(pos=pos, top_percent=0.33)
     keywords = single_rank_extractor.get_n_best(n=2)
```

Obrázok 62 SingleRank

TopicRank

TopicRank využíva mierne rozlišný spôsob od algoritmov TextRank a SingleRank. Jeho úlohou je extrahovať kľúčové frázy z najdôležitejších tém prítomných v dokumente. Tento algoritmus považuje tému za zhluk podobných

kandidátov na kľúčové frázy. Tieto témy sú následne zoradené podľa ich dôležitosti v dokumente a pre každú tému sa vyberie najdôležitejšia kľúčová fráza. Algoritmus TopicRank pozostáva z nasledujúcich krokov:

- identifikácia témy,
- hodnotenie založené na grafe,
- výber kľúčových slov.

Na nájdenie najlepšej kľúčovej frázy pre danú tému sa používajú tri stratégie. Jedna zo stratégií konvertuje všetky kľúčové frázy späť na ich generickú formu a vyberie kľúčovú frázu, ktorá sa v dokumente objavila ako prvá. Druhá stratégia vyberá najčastejšiu kľúčovú frázu, zatiaľ čo tretia vyberá na základe ťažiska klastra (centroidu). Ťažisko je imaginárne alebo skutočné miesto predstavujúce stred klastra. Každý dátový bod je priradený každému z klastrov znížením súčtu štvorcov v klastri.

Implementácia algoritmu TopicRank

Pre implementovanie algoritmu TopicRank môžeme znovu využiť knižnicu, ktorú sme si predstavili v predošlých dvoch prípadoch. Kód bude vyzerať veľmi podobne.

```
[ ]  pos = {'NOUN', 'PROPN', 'ADJ'}

     topic_rank_extractor = pke.unsupervised.TopicRank()
     topic_rank_extractor.load_document(input=text,language='en')
     topic_rank_extractor.candidate_weighting(pos=pos,top_percent=0.33)
     keywords = topic_rank_extractor.get_n_best(n=2)
     keywords
```

Obrázok 63 TopicRank

3.3.3 Prístupy založené na strojovom učení

Prístupy extrakcie kľúčových slov, ktoré sú založené na strojovom učení využívajú učenie s učiteľom (s dohľadom) a transformujú úlohu extrakcie kľúčových fráz do klasifikačného alebo predikčného problému. Model natrénovaný na označenej množine sa používa na určenie, či je kandidátske

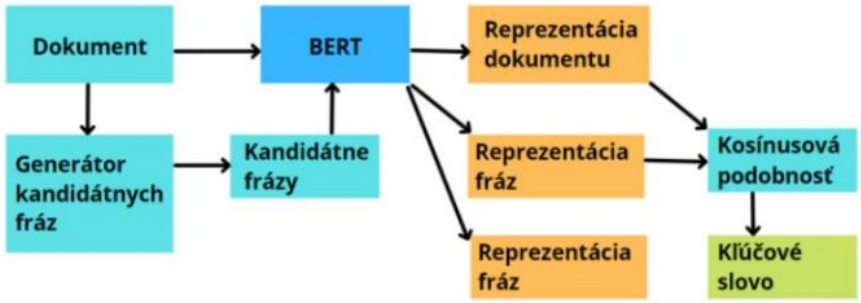
slovo v texte kľúčovou frázou alebo nie. Výhodou prístupov založených na strojovom učení je, že vyžadujú menej prípadne žiadne predspracovanie textu a extrahujú kľúčové frázy s vysokou sémantickou relevanciou. Nevýhodou je, že modely sú jazykovo a niekedy aj kontextuálne závislé a zmena korpusu môže vyžadovať trénovanie modelu na novo, prípadne výber iného modelu. Ďalšou nevýhodou je, že strojové učenie so sebou prináša vyššiu výpočtovú náročnosť, kvôli čomu je extrakcia pomocou prístupov založených na strojovom učení pomalšia nar ozdiel od prístupov, ktoré sú založené výlučne na štatistike a nevyžadujú trénovacie dáta.

KEA

Jednou z prvých metód extrakcie kľúčových slov, ktorá využíva strojové učenie je KEA, ktorá a spočíva v určení, či je kandidátske slovo kľúčovou frázou vypočítaním TF-IDF každého kandidátskeho slova a miesta, kde sa prvýkrát objaví v texte, a vložení týchto hodnôt hodnoty do Naive Bayes.

KeyBERT

Najpoužívanejšou technikou, ktorá dokáže extrahovať kľúčové slová s vysokou sémantickou relevanciou je KeyBERT. Táto technika využíva vopred natrénovaný model BERT (Bidirectional Encoder Representations from Transformes). Samotný algoritmus KeyBERT začína poslaním dokumentu do BERT modelu, ktorý vytvorí reprezentáciu dokumentu tak, že rozdelí text na vektory pevnej veľkosti, predstavujúce sémantiku dokumentu (Obrázok 64).



Obrázok 64 KeyBERT

V druhom kroku generátor kandidátnych fráz extrahuje z dokumentu kandidátne frázy pomocou jednoduchých techník, ako počet výskytov, TF-IDF a podobne. V ďalšom kroku sa tieto dáta opäť pošlú do BERT modelu a získa sa reprezentácia na úrovni fráz. Následne sa medzi reprezentáciou na úrovni dokumentu a reprezentáciou na úrovni fráz vypočíta kosínusová podobnosť, pre získanie najpodobnejších slov k dokumentovej reprezentácií, ktoré sú výslednými kľúčovými slovami. Výpočet prebieha podľa vzorca (22). Výsledky sú následne zoradené zostupne a vyberie sa top *n* položiek.

$$podobnosť(A,B) = \frac{A * B}{|A| * |B|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (22)$$

Implementácia algoritmu KeyBERT

Prvým krokom, ktorý je potrebné urobiť, je nainštalovať knižnicu, ktorá implementuje KeyBERT a následne ju naimportovať. Následne si môžeme implementovať algoritmus pomocou dvoch riadkov kódu. Parameter *keyphrase_ngram_range* definuje rozsah želaných n-gramov. Do parametra *stop_words* vložíme zoznam želaných stop slov. Prednastavený model, ktorý sa ku extrakcií používa je model "all-MiniLM-L6-v2". Tento model pracuje s anglickým jazykom.

```
[12] !pip install keybert
     from keybert import KeyBERT

[13] keybert_extractor = KeyBERT()
     keywords = keybert_extractor.extract_keywords(text, keyphrase_ngram_range=(2, 2), stop_words=stop_words)

[14] keywords

     [('keyword extraction', 0.7149),
      ('text summarization', 0.7084),
      ('summarization method', 0.6712),
      ('summarization technique', 0.6682),
      ('document keyword', 0.5379)]
```

Obrázok 65 KeyBERT implementácia

3.3.4 Hybridné prístupy

Hybridné prístupy kombinujú predchádzajúce metódy. Využívajú heuristické znalosti, ako je poloha, dĺžka slov, HTML tagy okolo slov a iné metódy.

3.4 Evalvácia algoritmov pre extrakciu kľúčových slov

Nie je ľahké navrhnúť hodnotiacu metriku, ktorá by mohla odrážať výhody a nevýhody algoritmu. Keďže hodnotiaca metrika môže hodnotiť iba jeden aspekt algoritmu, viacero metrík môže presnejšie a komplexnejšie vyhodnotiť algoritmus. Napríklad výskumníci zvyčajne používajú presnosť (precision), pokrytie (recall) a F1-skóre (harmonický priemer) na vyhodnotenie metódy z viacerých perspektív.

3.4.1 Metriky založené na štatistike

Hodnotiace metriky založené na štatistike analyzujú výkonnosť metódy výpočtom podielu počtu rôznych kľúčových fráz, ako je počet extrahovaných kľúčových fráz, správnych kľúčových fráz, nesprávnych kľúčových fráz a manuálne priradených kľúčových fráz. Štandardné metriky založené na štatistike zahŕňajú presnosť, pokrytie a harmonický priemer. Majú široké využitie a podrobné informácie o nich nájdete v kapitole 3.4.

3.4.2 Metriky založené na lingvistike

Doposiaľ uvedené hodnotiace metriky sú založené na predpoklade, že kľúčové frázy sú navzájom nezávislé, ale na základe zvykov ľudského jazyka dúfame, že dôležitejšie kľúčové frázy by mali byť umiestnené vyššie. Nasledujúce tri metriky hodnotenia môžu odrážať funkcie poradia medzi výstupmi kľúčových fráz pomocou algoritmu.

Mean Reciprocal Rank (MRR)

Stredná recipročná hodnosť je miera na vyhodnotenie modelov, ktoré vracajú dokumentom zoradený zoznam kľúčových fráz. MRR sa stará len o jednu najvyššie hodnotenú relevantnú položku. Ak model vráti relevantnú kľúčovú frázu na treťom najvyššom mieste, potom sa o to MRR stará. Nezáleží na tom, či sú ostatné relevantné kľúčové frázy (za predpokladu, že nejaké existujú) v poradí číslo 1 alebo číslo 10. MRR udáva priemerné hodnotenie prvej správnej predpovede, kde d je počet dokumentov a $rank_i$ je poradie, v ktorom

bola nájdená prvá správna kľúčová fráza dokumentu *i. MMR možno vypočítať podľa vzorca (24)*

$$MMR = \frac{1}{|d|}\sum_{i=1}^{d}\frac{1}{rank_i} \tag{24}$$

Mean Average Precision (MAP)

MAP berie do úvahy poradie konkrétneho vráteného zoznamu kľúčových fráz. Priemerná presnosť AP je definovaná podľa rovnice (25),

$$AP = \frac{\sum_{n=1}^{|N|} P(n)gd(n)}{|LN|} \tag{25}$$

kde |*N*| dĺžka zoznamu, |*LN*| predstavuje počet relevantných položiek, *P*(*n*) je presnosť a *gd*(*n*) sa rovná jednej, ak je *n*-tá položka zlatá kľúčová fráza a 0 v opačnom prípade. Spriemerovaním *AP* cez množinu *n* dokumentov je stredná priemerná presnosť (*MAP*) definovaná podľa (26):

$$Bref = \frac{1}{C}\sum c \in C\ 1 - \frac{|I|}{M} \tag{26}$$

4 Strojový preklad

4.1 Úvod do strojového prekladu

Strojový preklad (MT) alebo automatický preklad na internete už prestal pútať pozornosť zvedavcov alebo „malých“ používateľov, ktorí používajú MT pre vlastné účely. V poslednom období MT vstupuje do „reálnej“ ekonomiky jazykového odvetvia a mení ho od základov. MT systémy, podobne ako jazyk, môžeme chápať ako komplexné adaptívne systémy spracovania prirodzeného jazyka (NLP), ktoré pozostávajú z agentov, ktorí na seba navzájom pôsobia, aby dosiahli spoločný cieľ úlohy, ktorá pozostáva z text-to-text transformácie, t. j. prekladu. Inými slovami povedané, vzhľadom na daný vstup - východiskový text v prirodzenom jazyku, MT systémy musia vytvoriť výstup ako verziu cieľového textu tiež v prirodzenom jazyku. Spomedzi aplikácií, ktoré ako výstup vytvárajú prirodzený jazyk je strojový preklad najvýznamnejší.

Strojový preklad je automatické konvertovanie textu z jedného prirodzeného jazyka do druhého prirodzeného jazyka, pričom kvalitu, ktorú produkujú systémy, resp. nástroje strojového prekladu lavíruje od veľmi nízkej po veľmi vysokú v závislosti od vnútorných (jazykových) vlastností daného textu a jazykového páru, teda od samotného spracovania daného prirodzeného jazyka a pravidiel transfer.

Automatické preklady je možné rozdeliť do troch kategórií, pričom každá kategória má svoje využitie. V prvej kategórii sa požaduje od prekladu kvalita vhodná na publikovanie, čo však nie je možné bez zásahu prekladateľa (buď na vstupe formou preeditácie textu, pomoou kontrolovaného jazyka alebo na výstupe formou posteditácie textu). Strojový preklad môže byť prvým krokom k vytvoreniu hrubého prekladu a následne, v druhom kroku môže posteditor zeditovať výstup na požadovanú úroveň kvality. V súčasnosti vygenerovanie plne automatického vysoko kvalitného strojového prekladu (FAHQMT) nie je možné, avšak v kombinácii so zásahom prekladateľa (v posteditovaní) sa dá dosiahnuť vysoká kvalita prekladu vhodná na publikovanie.Tieto preklady sa používajú hlavne na diseminácu informácií vo viacerých jazykoch (napr. letáky k produktom). Vzhľadom na zložitosť jazyka sa tento cieľ podarilo dosiahnuť len v

prípade niektorých aplikácií ako napr. predpoveď počasia (systém Metéo, ktorý sa dodnes používa) alebo súhrn športových podujatí. Množina slovnej zásoby a gramatiky je v týchto doménach dostatočne obmedzená na to, aby bolo možné napísať pravidlá transferu.

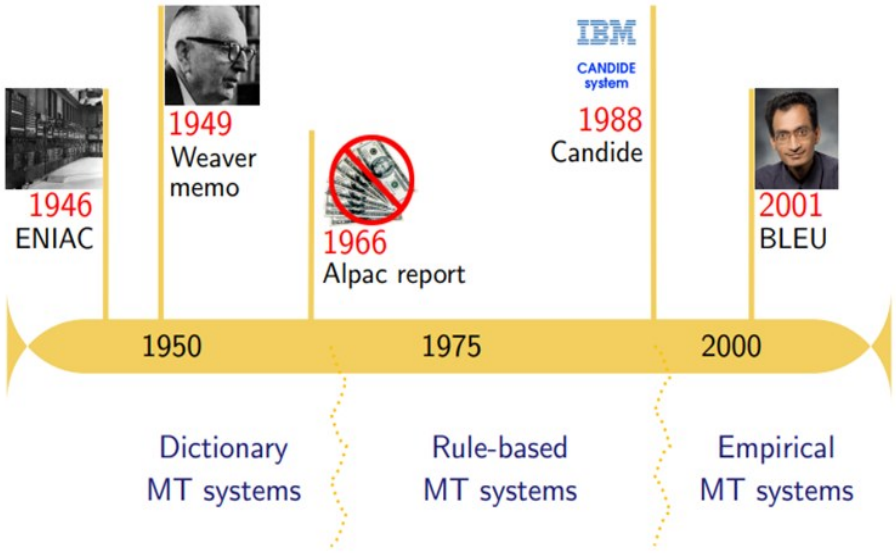
Druhá kategória automatických prekladov obsahuje preklady „krátkodobých" textov, ktoré neprekladajú profesionálni prekladatelia, používajú sa iba na asimiláciu, teda porozumenie obsahu daného textu. Napr. preklad webových stránok pomocou online prekladateľských služieb samotnými používateľmi, ktorí chcú porozumieť tomu, čo je napísané v cudzom jazyku. Preklad nemusí byť dokonalý, aby vystihol význam. Ak používateľ dostatočne pochopí informácie na vyriešenie svojho technického problému, potom strojový preklad preukázal svoju užitočnosť. Práve gisting je najrozšírenejším využitím strojového prekladu.

Posledná kategória pokrýva preklady rýchlej komunikácie (napr. email), ktorá používa automatický preklad na výmenu informácií.

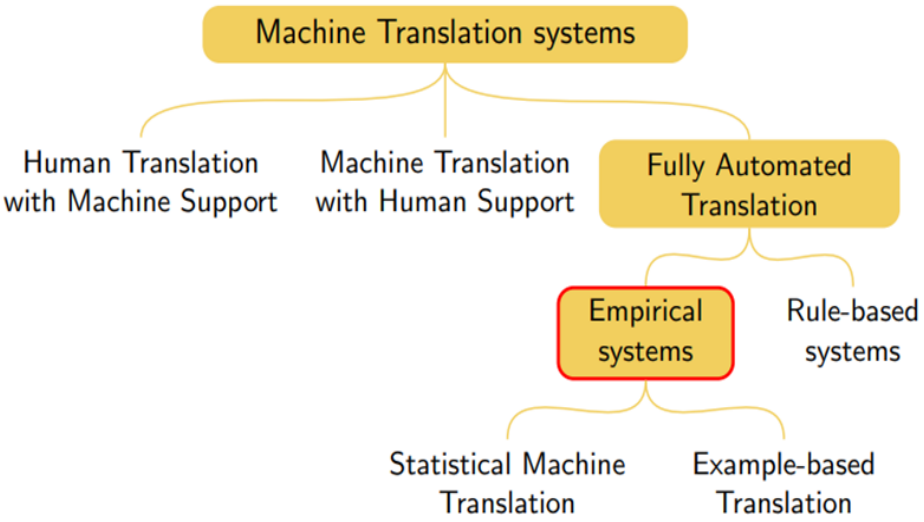
Koehn (2010) tvrdí, že automatický preklad nemusí byť dokonalý, aby bol použiteľný. Hovorí, že aj „mizerný" strojový preklad má svoje využitie. Stotožňuje sa s prvými dvoma kategóriami, avšak poslednú kategóriu, jej uplatnenie vníma ako využívanie v interkultúrnej komunikácii, ktorá v sebe zahŕňa aj online komunikáciu, či už v synchrónnej alebo asynchrónnej forme, ako aj v písanej alebo hovorenej forme.

Vzhľadom na možnosti širokého uplatnenia jazykových technológií v rôznych sférach života spoločnosti, nadobúda dnes problematika okrem vedeckej dimenzie aj spoločenský rozmer, pričom každé si vyžaduje inú rýchlosť a kvalitu.

Zaujímavým smerom pre strojový preklad je jeho kombinácia s rečových technológií (jeho integrácia). To otvára široké možnosti využitia, ako napríklad preklad telefonických rozhovorov alebo zvukových vysielaní. Súčasnými testovacími aplikáciami sú preklady vysielaných správ, parlamentných prejavov a rozhovorov v oblasti cestovania. V súčasnosti sa používajú systémy, ktoré monitorujú spravodajské relácie v cudzích jazykoch a vykonávajú preklad reči v reálnom čase.



Obrázok 66 História strojového prekladu



Obrázok 67 Rozdelenie typov strojového prekladu

História strojového prekladu

Prvé príklady systémov strojového prekladu (MT) boli financované vládou. Prehnané náklady na počítače znamenali, že neexistoval trh s osobnými počítačmi. Technológia, ktorá bola na to potrebná, bola príliš drahá na to, aby sa MT stal ekonomicky výhodným podnikaním.

Typy strojových prekladov:

- Na základe pravidiel (rule-based)

- Na základe štatistiky (statistics-based)

Neurónový strojový preklad (Neural Machine Translation)

Pôvodným cieľom MT bolo zostrojiť počítače, ktoré by samostatne vykonávali preklad na základe pravidiel. Inými slovami, stroj by sa naučil úplnú slovnú zásobu a gramatiku viacerých jazykov, aby mohol samostatne prekladať.

V roku 1954 počítač IBM 701 úspešne preložil 49 viet na tému chémia z ruštiny do angličtiny. Približne v tomto období zaznamenal MT posun v záujmoch od vojenských k civilným. Globalizácia si vyžadovala väčšiu integráciu trhov na celom svete. Základná znalosť jazykov, ktorými sa hovorí na celom svete, sa stala nevyhnutnosťou pre každú obchodnú organizáciu.

V roku 1997 bezplatná online prekladateľská služba BabelFish umožnila preklady medzi jazykmi angličtina, nemčina, francúzština, španielčina, portugalčina a taliančina. Táto služba mala svoje problémy s výkonom. Často sa zamotávala, keď mala vybrať medzi slovami, ktoré mali v cieľovom jazyku rôzny význam. MT systémom založeným na pravidlách chýbali mimotextové znalosti.

Druhá etapa MT prišla v podobe štatistického MT. Je to tá istá technológia, ktorá sa používa pri preklade Google Translate. SMT funguje na myšlienke, že ak do rozsiahlej pamäte počítača vložíte dostatok údajov v podobe paralelných textov v dvoch jazykoch, bude schopný odhaliť a obnoviť štatistické vzory medzi nimi. Stane sa samoučiacim sa tým, že zásobuje svoj stroj, aby sa zväčšil jeho korpus. Prekladač Google má aj možnosť spätnej väzby pre používateľov, ktorí môžu navrhovať lepšie preklady, čím si jeho systém môže zväčšiť svoju pokladnicu slov a fráz, z ktorej môže neskôr čerpať.

SMT je síce vylepšením RBMT, ale počas prekladu rozpoznáva len príkazy. Nedokáže prekladať významy, ktoré si ľudia spájajú s prekladaným jazykom. Vyniká pri preklade vedeckých a technických textov, ale nedokáže preložiť hovorový alebo umelecký jazyk. Napríklad mnohé názvy tradičnej čínskej medicíny sa nedajú preložiť, pretože nemajú paralelné anglické výrazy. Ďalej sú viazané na čínsku kultúru, ktorú SMT nepozná.

Poslednou a najnovšou fázou MT je neurónová MT. NMT pozostáva z neurónových sietí natrénovaných a optimalizovaných na vykonávanie prekladateľských služieb. Využíva hlboké učenie na analýzu obrovského

množstva prekladov, ktoré už vykonali ľudskí prekladatelia. NMT dokáže zohľadniť celé vety, pochopiť kontext, zohľadniť rôzne varianty a pracovať s jazykovými jemnosťami, ktoré by sa nikdy nedali naprogramovať do štatistického modelu. Výsledkom je, že NMT prekladá plynulejšie a prirodzenejšie. Napodobňuje fungovanie ľudského mozgu v jeho schopnosti učiť sa a vytvárať nervové dráhy. Vďaka štruktúre neurónovej siete je systém adaptívnejší na spracovanie zložitých modelov ako systém založený na pravidlách a štatistike. Dokáže sa tiež učiť zo svojich chýb a podľa toho sa prispôsobiť, aby nabudúce fungoval efektívne.

NMT sa už ukázala byť oveľa lepšia ako SMT. K nahradeniu ľudských prekladateľov je však ešte dlhá cesta. Následná úprava MT otvorí poskytovateľom prekladateľských služieb nové možnosti rastu. Budúcnosť MT bude symbiotický vzťah medzi ľudskými prekladateľmi a strojovými prekladateľmi.

Výhody strojového prekladu

Strojový preklad je nenahraditeľným nástrojom v procese prekladu. Môže sa používať samostatne alebo v kombinácii s ľudskou následnou úpravou.

MT ponúka tri hlavné výhody:

- Šetrí čas - Strojový preklad dokáže ušetriť veľa času, pretože dokáže preložiť celé textové dokumenty v priebehu niekoľkých sekúnd. Majte však na pamäti, že prekladatelia by mali vždy dodatočne upravovať preklady vykonané MT.
- Znižuje náklady - Strojový preklad môže podstatne znížiť vaše náklady, pretože si vyžaduje menšiu účasť človeka.
- Zapamätá si výrazy -Ďalšou výhodou strojového prekladu je jeho schopnosť zapamätať si kľúčové pojmy a opätovne ich použiť všade tam, kde by sa mohli hodiť.

4.2 Typy strojového prekladu

Existujú štyri rôzne typy strojového prekladu - štatistický strojový preklad (SMT), strojový preklad založený na pravidlách (RBMT), hybridný strojový preklad (HMT) a neurónový strojový preklad (NMT).

Strojový preklad založený na pravidlách (RBMT)

RBMT - najstaršia forma MT - prekladá obsah na základe gramatických pravidiel. Od vzniku RBMT došlo k výraznému pokroku v technológii strojového prekladu, takže má niekoľko nevýhod. Medzi tieto nevýhody patrí potreba veľkého množstva ľudskej následnej úpravy a ručného pridávania jazykov. Napriek tejto nízkej kvalite prekladu je RBMT užitočný v základných situáciách, keď sa vyžaduje len rýchle pochopenie významu.

Štatistický strojový preklad (SMT)

SMT funguje na základe vytvorenia štatistického modelu vzťahov medzi slovami, frázami a vetami textu. Potom tento model prekladu aplikuje na druhý jazyk a prevádza tie isté prvky do nového jazyka. SMT do určitej miery zlepšuje RBMT, ale stále má mnoho rovnakých problémov.

Táto metóda patrí taktiež medzi tie jednoduchšie. Nepoužíva žiadne informácie závislé na konkrétnom jazyku, a preto je použiteľná pre ich ľubovoľnú kombináciu. Na tvorbu slovníka, ktorým budeme prekladať, potrebujeme bilingválny zarovnaný korpus. Z neho si zistíme všetky možné preklady daného slova a ich frekvenciu výskytu. Do slovníka následne uložíme dvojicu: slovo a jeho najčastejší preklad. Takto získaným slovníkom následne môžeme prekladať. Ak by sme napríklad tvorili česko-slovenský prekladač a zistili by sme, že slovo hovoriť sa dá preložiť ako mluvit alebo říkat a slovo mluvit sa v korpuse vyskytuje častejšie, tak by sme všetky výskyty slova hovoriť prekladali na mluvit. Ak chceme dosiahnuť väčšiu úspešnosť, nepočítame počet výskytov jednotlivých slov, ale bigramov, trigramov a podobne. Týmto získame väčšiu úspešnosť slovníka, ale jeho veľkosť začne značne narastať, čo sa môže odraziť aj na výkone samotného prekladača. Táto metóda má však aj veľa nevýhod. Je to napríklad tvorba bilingválneho zarovnaného korpusu, z ktorého sa počítajú frekvencie výskytu. Na to, aby sme získali dostatočne veľké a smerodajné údaje, potrebujeme obrovské korpusy, ktorých tvorba nie je ľahká ani lacná.

Ďalším problémom je aj ich samotná tvorba. Často ich totiž nedokážeme správne zarovnať'. Napríklad sa často stáva, že na preklad slova nepoužijeme jedno slovo, ale celú vetnú frázu. Ďalším problémom je rôzna štylizácia vety. V

češtine môžeme napríklad povedať: „Nestalo se ti nic?" a do slovenčiny to môže byť preložené ako „Si v poriadku?"

Preklad s použitím tretieho jazyka

Ďalšou možnosťou, ako je možné urobiť preklad z jedného jazyka do druhého, je použiť nejaký tretí jazyk. To sa nám môže hodiť, ak chceme prekladať a k dispozícii máme len dva prekladače, ktoré dokážu prekladať z nášho jazyka do nejakého tretieho a následne z neho do nami požadovaného jazyka. Problémom tejto metódy je to, že má dve slabé miesta, a to dva prekladače. Celkový preklad môže byť teda horší ako kvalita horšieho z nich. V najhoršom možnom prípade budú oba robiť chyby na rôznych miestach. Táto metóda je použiteľná len ak máme k dispozícii dva kvalitné prekladače.

Ďalšou variantou takéhoto prekladu môže byť použitie „medzijazyka" (interlingua). Princíp je takmer identický ako v prípade použitia tretieho jazyka. Rozdielnym je preklad do určitej formy jazyka, ktorý nemá konkrétnu komunikačnú podobu, ale obsahuje len význam vety a sémantiku jednotlivých slov. Takýto preklad má tú výhodu, že po preložení textu do medzijazyka je možné z neho prekladať do ľubovoľného jazyka, pre ktorý máme vytvorené pravidlá.

Hybridný strojový preklad (HMT)

HMT je kombináciou RBMT a SMT. HMT využíva prekladovú pamäť, vďaka čomu je oveľa efektívnejší z hľadiska kvality. Aj HMT má však svoje nevýhody, z ktorých najväčšou je potreba ľudskej úpravy.

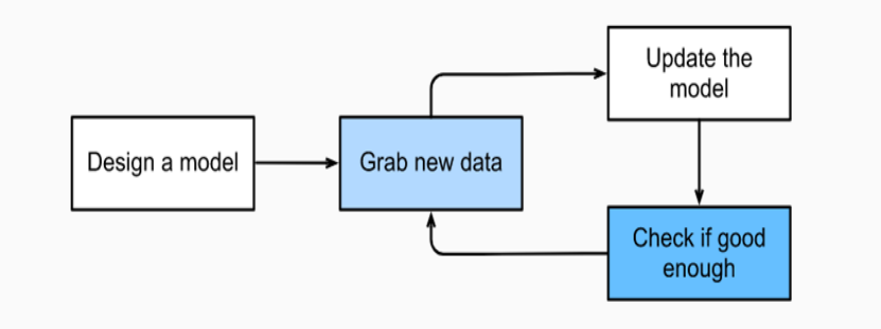
Neurónový strojový preklad (NMT)

NMT využíva umelú inteligenciu na učenie sa jazykov a neustále zlepšovanie týchto znalostí. Týmto spôsobom sa snaží napodobniť neurónové siete v ľudskom mozgu. NMT je presnejší ako iné typy prekladov s umelou inteligenciou. Pomocou NMT je jednoduchšie pridávať jazyky a prekladať obsah. Keďže NMT poskytuje lepšie preklady, rýchlo sa stáva štandardom pri vývoji MT nástrojov.

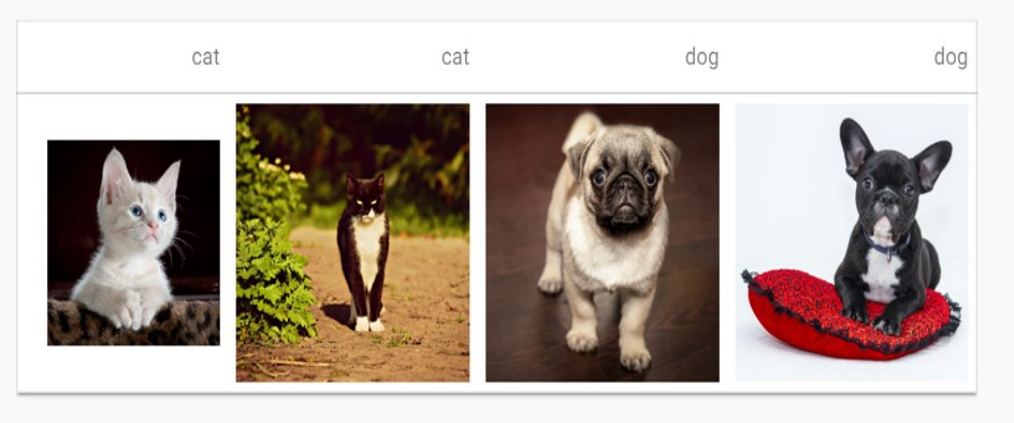
NMT funguje tak, že zahŕňa trénovacie údaje. V závislosti od potrieb používateľa môžu byť tieto údaje všeobecné alebo vlastné.

- **Všeobecné** údaje: Ide o súhrn všetkých údajov, ktoré sa naučili z prekladov vykonaných v priebehu času strojovým prekladom (MTE). Tieto údaje vytvárajú všeobecný prekladový nástroj pre rôzne aplikácie vrátane textu, hlasu a dokumentov.
- **Vlastné alebo špecializované údaje:** Ide o trénované údaje, ktoré sa dodávajú strojovému prekladaču s cieľom vytvoriť špecializáciu v danej oblasti. Medzi predmety patrí strojárstvo, dizajn, programovanie alebo akýkoľvek odbor s vlastnými špecializovanými slovníkmi a slovníkmi.

Hlboké učenie je len jednou z mnohých populárnych metód na riešenie problémov strojového učenia.



Obrázok 68 Proces trénovania



Obrázok 69Ukážka výsledku rozpoznávania

5 Neurónová sieť v strojovom preklade

5.1 Neurónové siete

Neurónovú sieť môžeme opísať ako matematický model spracovania informácií. Neurónová sieť nie je fixný program, ale skôr model (systém), ktorý spracováva informácie (vstupy).

Charakteristiky neurónovej siete sú nasledovné:

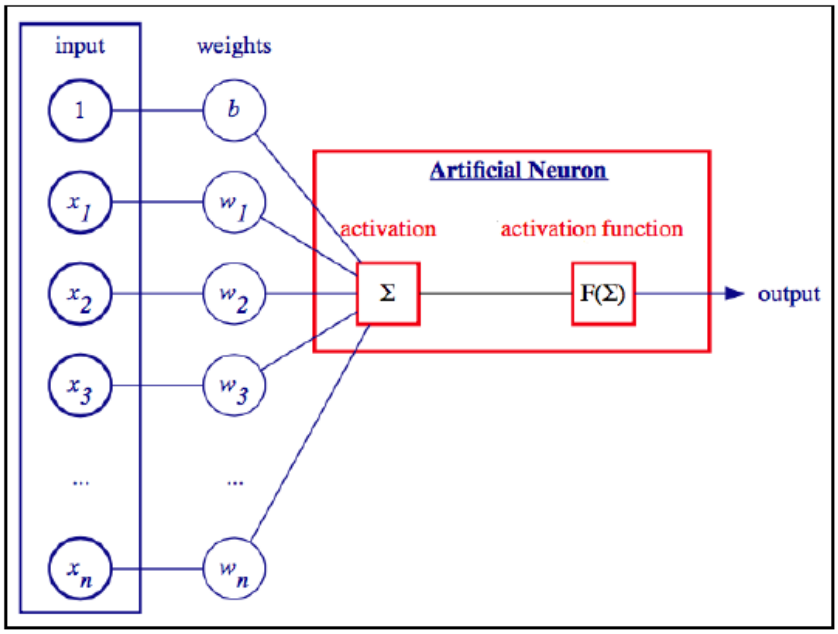
- Spracovanie informácií prebieha v najjednoduchšej forme, nad jednoduchými prvkami, ktoré sa nazývajú neuróny.
- Neuróny sú prepojené a vymieňajú si medzi sebou signály prostredníctvom spojovacích liniek. Môžu byť silnejšie alebo slabšie, a to určuje spôsob spracovania informácií.
- Každý neurón má vnútorný stav, ktorý je určený všetkými prichádzajúcimi spojeniami z iných neurónov.
- Každý neurón má inú aktivačnú funkciu, ktorá sa počíta na základe jeho stavu a určuje jeho výstupný signál.

Dve hlavné charakteristiky neurónovej siete:

- **Architektúra neurónovej siete** - opisuje súbor spojení medzi neurónmi (feedforward, rekurentné, viacvrstvové alebo jednovrstvové a pod.), tak aj počet vrstiev a počet neurónov v každej vrstve.
- **Učenie (learning)** - trénovanie. Najčastejším a najbežnejším, ale nie jediným spôsob trénovania neurónovej siete je pomocou gradientového klesania a spätného šírenia.

5.1.1 Neurón

Neurón je matematická funkcia, ktorá prijíma jednu alebo viac vstupných hodnôt a na jej výstupe je jedna číselná hodnota:



Obrázok 70 Ukážka rozličných elementov neurónu

Pre výpočet neurónu postupujeme nasledovne:

$$y = f(\sum_i x_i w_i + b) \tag{30}$$

kde xi číselná hodnota, ktorá predstavuje vstupné údaje alebo výstupy z iných neurónov, ak je neurón súčasťou neurónovej siete. Váhy wi sú číselné hodnoty, ktoré predstavujú buď silu vstupov alebo silu spojení medzi jednotlivými neurónmi. Váha b je špeciálna hodnota nazývaná odchýlka (bias), ktorej vstup je vždy 1.

1. Najprv vypočítame súčet vstupov xi a váh wi (tzv. aktivačná hodnota).
2. Potom použijeme výsledok váženého súčtu ako vstup pre aktivačnú funkciu f, ktorá je známa aj ako prenosná funkcia (transfer function). Existuje mnoho typov aktivačných funkcií, ale všetky musia spĺňať požiadavku nelineárnosti.

Aktivačnú hodnotu možno interpretovať ako skalárny súčin dvoch vektorov w a x:

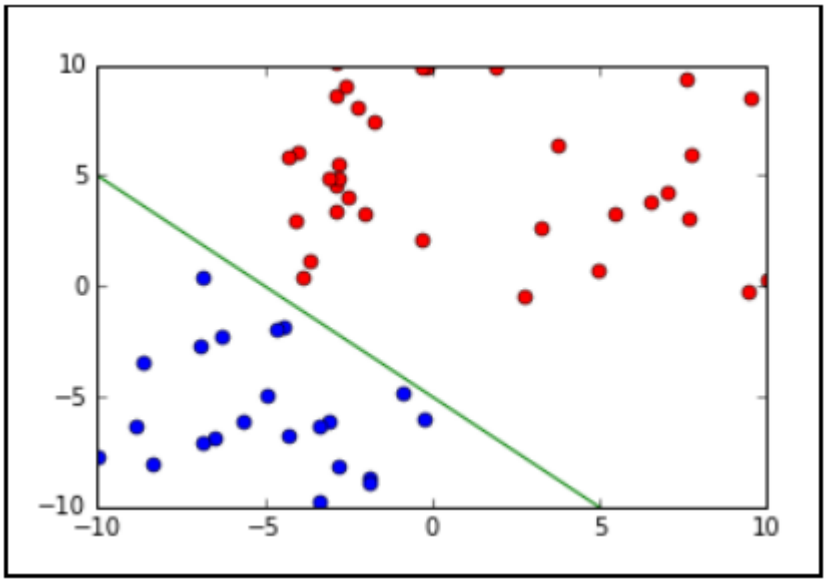
$$y = f(\vec{x} \cdot \vec{w} + b).$$

Vektor x bude kolmý na váhový vektor w, keď $\vec{x} \cdot \vec{w} = 0$.

Všetky vektory x, pre ktoré platí $\vec{x} \cdot \vec{w} = 0$ definujú hyperrovinu (angl. hyperplane) v priestore Rn , kde n je dimenzia x.

Pre lepšie pochopenie, uvažujme o špeciálnom prípade, v ktorom máme len jednu vstupnú hodnotu x s aktivačnou funkciou f(x) = x. Výstupom je neurón y = wx + b, čo je v podstate lineárna rovnica. Z toho vyplýva, že v jednodimenzionálnom vstupnom priestore neurón definuje (zobrazuje) priamku. Ak by sme uvažovali o dvoch a viac vstupoch, potom neurón definuje rovinu alebo hyperrovinu pre ľubovoľný počet vstupných rozmerov.

Úlohou odchýlky b je umožniť posun hyperroviny od stredu súradnicového systému. Ak nepoužijeme odchýlku, neurón bude mať obmedzenú reprezentačnú schopnosť.



Obrázok 71 Diagram hyperroviny

Perceptron (teda neurón) pracuje len s lineárne oddeliteľnými triedami z dôvodu definovania hyperrovinu. Z tohto dôvodu sa neuróny usporiadávajú do neurónovej siete.

5.1.2 Vrstvy v neurónovej sieti

Neurónová sieť môže mať neobmedzený počet neurónov, ktoré sú usporiadané do vzájomne prepojených vrstiev. Vstupná vrstva predstavuje súbor údajov s počiatočnými podmienkami (hodnotami).

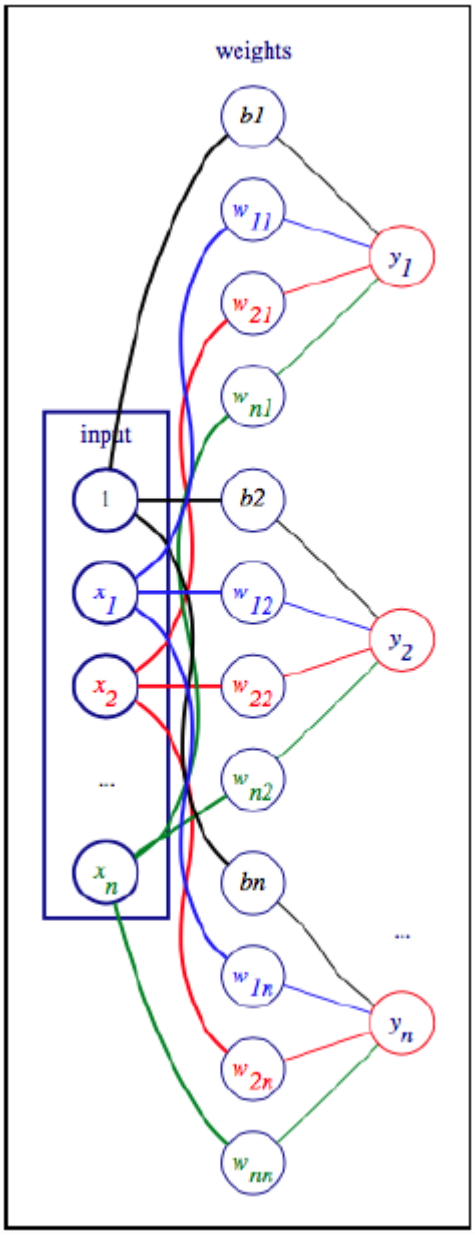
Napríklad ak je vstupom obrázok v odtieňoch sivej, výstupom každého neurónu vo vstupnej vrstve je intenzita jedného pixelu obrazu.

Práve z tohto dôvodu vo všeobecnosti nepočítame vstupnú vrstvu ako súčasť ostatných vrstiev. Keď hovoríme o jednovrstvovej sieti, myslíme tým, že je to jednoduchá sieť, ktorá má okrem vstupnej vrstvy už len jednu, výstupnú vrstvu.

Výstupná vrstva môže mať viac ako jeden neurón. To je užitočné najmä pri klasifikácii, kde každý výstupný neurón predstavuje jednu triedu.

Napríklad v prípade MNIST databázy (rozsiahla databáza obrázkov, na ktorých sú ručne písané číslice) budeme mať 10 výstupných neurónov, pričom každý neurón zodpovedá číslici od 0 do 9. Týmto spôsobom môžeme na klasifikáciu číslice použiť jednovrstvovú sieť pre každý obrázok. Číslicu určíme tak, že zoberieme výstupný neurón s najvyššou hodnotou aktivačnej funkcie. Ak je to y7, budeme vedieť, že sieť si myslí, že obrázok zobrazuje číslo 7.

V prípade jednovrstvovej doprednej siete sa váhy w pre každé spojenie medzi neurónmi zobrazujú explicitne, avšak zvyčajne hrany spájajúce neuróny predstavujú váhy implicitne. Váha wij spája i-ty vstupný neurón s j-tym výstupným neurónom. Prvý vstup 1 je jednotkou skreslenia a váha b1 je váha odchýlky:



Obrázok 72 1-vrstvova dopredná sieť

Neuróny vľavo predstavujú vstup s odchýlkou b, stredný stĺpec predstavuje váhy pre každé spojenie a neuróny vpravo predstavujú výstup vzhľadom na váhy *w* (Obrázok 72).

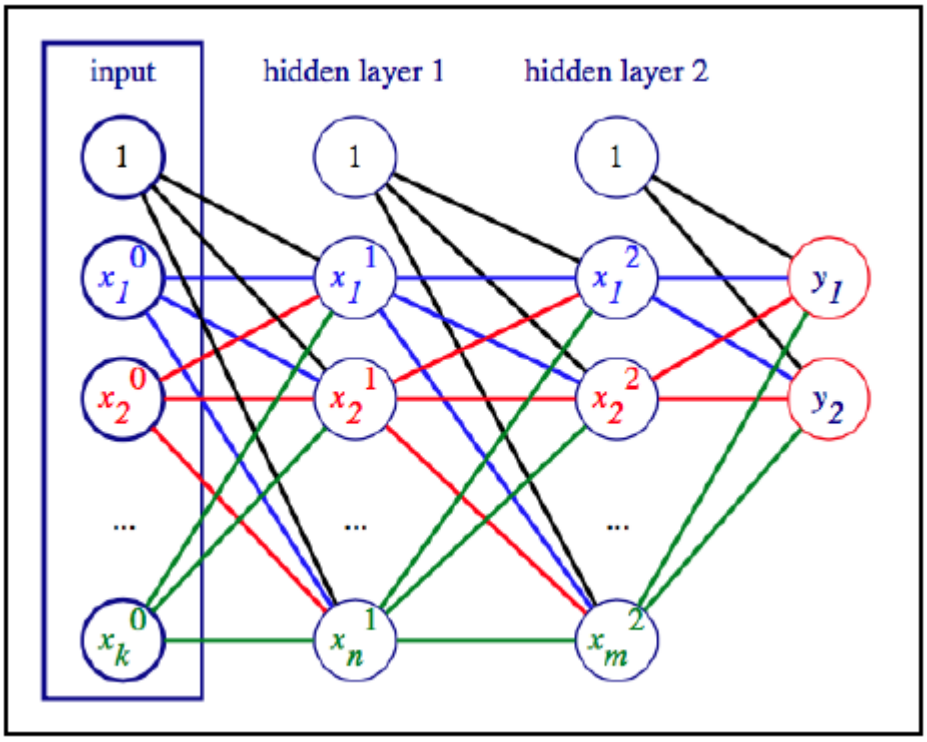
Neuróny jednej vrstvy môžu byť prepojené s neurónmi iných vrstiev, ale nie s nerónmi v tej istej vrstve. V tomto prípade sú vstupné neuróny $x_1,...,x_n$ pripojené len k výstupným neurónom $y_1,...,y_n$.

Dôvodom usporiadania neurónov do siete je ten, že neurón môže sprostredkovať len obmedzenú informáciu (len jednu hodnotu). Keď však neuróny spojíme do vrstiev, ich výstupy tvoria vektor a namiesto jednej aktivácie

môžeme teraz uvažovať o celom vektore. Týmto spôsobom môžeme sprostredkovať oveľa viac informácií, nielen preto, že vektor má viacero hodnôt, ale aj preto, že relatívne pomery medzi nimi nesú ďalšie informácie.

Viacvrstvové neurónové siete

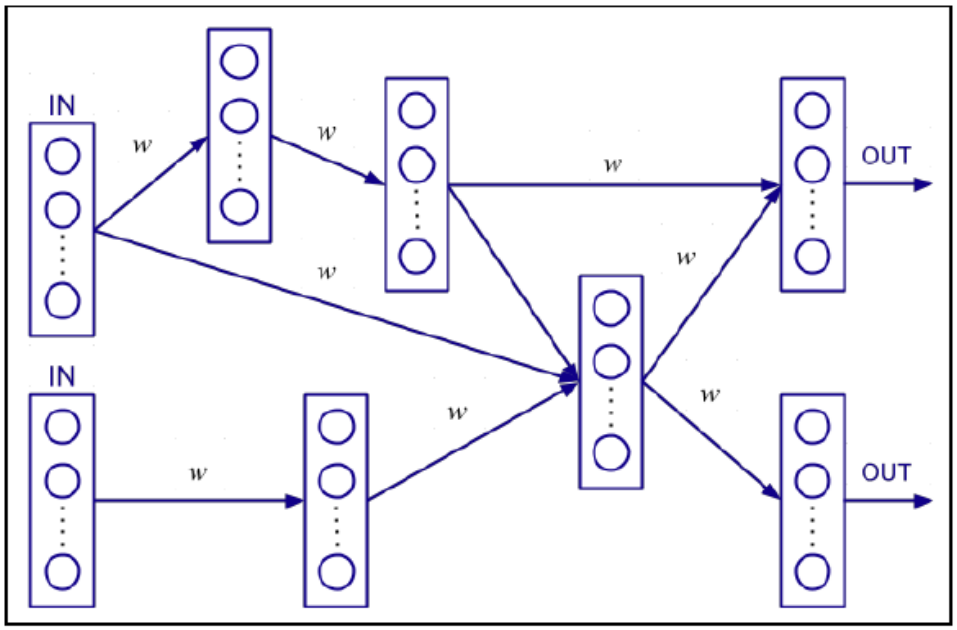
Jednovrstvové neurónové siete dokážu klasifikovať len lineárne oddeliteľné triedy. Nič nám však nebráni zaviesť ďalšie vrstvy medzi vstupom a výstupom. Tieto prídavné vrstvy sa nazývajú skryté vrstvy (angl. hidden layers). Trojvrstvová plne prepojená neurónová sieť s dvoma skrytými vrstvami je znázornená na obrázku. Vstup má k vstupných neurónov, prvá skrytá vrstva má n skrytých neurónov a druhá skrytá vrstva má m skrytých neurónov. Výstupom sú v tomto prípade dve triedy y_1 a y_2. V hornej časti je vždy zapnutý skresľovací neurón. Jednotka z jednej vrstvy je pripojená k všetkým jednotkám z predchádzajúcej a nasledujúcej vrstvy (teda plne prepojené). Každé spojenie má svoju vlastnú váhu w, ktorá sa kvôli jednoduchosti nezobrazuje. V tomto prípade ide o neurónovú sieť so sekvenčnými vrstvami.



Obrázok 73 Viacvrstvová sekvenčná sieť

Neuróny a ich spojenia môžu tvoriť aj orientované cyklické grafy. V takomto grafe informácia nemôže prejsť dvakrát z toho istého neurónu (žiadne slučky), tečie len jedným smerom, zo vstupu na výstup. Sieť je len špeciálnym

prípadom grafu, ktorého vrstvy sú prepojené sekvenčne. Nasledujúci obrázok tiež znázorňuje korektnú neurónovú sieť s dvoma vstupnými vrstvami, dvoma výstupnými vrstvami a náhodne prepojenými skrytými vrstvami. Kvôli zjednodušeniu sú znázornené viacnásobné váhy w, spájajúce vrstvy ako jedna čiara.



Obrázok 74 Neurónová sieť

Neurónová sieť ako zloženie neurónov je matematická funkcia, v ktorej vstupné údaje predstavujú argumenty funkcie a váhy siete w predstavujú jej parametre.

5.1.3 Aktivačné funkcie

Viacvrstvové siete dokážu klasifikovať lineárne neseparovateľné triedy za podmienky, že neuróny nemajú aktivačné funkcie, potom ich výstupom by bol súčet vážených vstupov, [redacted], čo je lineárna funkcia. Potom sa celá neurónová sieť (zloženie neurónov) je zložením lineárnych funkcií, čo je opäť lineárna funkcia. Aj keď pridáme skryté vrstvy, sieť bude stále ekvivalentná jednoduchému lineárnemu regresnému modelu so všetkými jej obmedzeniami. Aby sme sieť premenili na nelineárnu funkciu, použijeme nelineárnu aktivačnú funkcie pre neuróny. Zvyčajne majú všetky neuróny v tej istej vrstve rovnakú aktivačnú funkciu, avšak rôzne vrstvy môžu mať rôzne aktivačné funkcie.

Najbežnejšie aktivačné funkcie sú tieto:

- **funkcia identity (identity function)** - táto funkcia prepúšťa aktivačnú hodnotu a.

$$f(a) = a$$

(31)

- **funkcia hraničnej aktivity (threshold activity function)** - táto funkcia aktivuje neurón. Ak je aktivácia vyššia ako určitá hodnota a, tak ide o funkciu hraničnej aktivity.

$$f(a) = \begin{cases} 1 \text{ if } a \geq 0 \\ 0 \text{ if } a < 0 \end{cases}$$

(32)

- **logistická funkcia (logistic function) alebo logistický sigmoid (logistic sigmoid)** - táto funkcia je jednou z najčastejšie používaných, pretože jej výstup je v rozmedzí medzi 0 a 1 a možno ju stochasticky interpretovať ako pravdepodobnosť aktivácie neurónu.

$$f(a) = \frac{1}{1+exp(-a)}$$

(33)

- **bipolárny sigmoid (bipolar sigmoid)** - je to v podstate logistický sigmoid preškálovaný (rescaled) a preložený tak, aby mal rozsah (-1, 1).

$$f(a) = \frac{2}{1+exp(-a)} - 1 = \frac{1-exp(-a)}{1+exp(-a)}$$

(34)

- **hyperbolický tangens (Hyperbolic tangent)**

$$f(a) = \frac{exp(a)-exp(-a)}{exp(a)+exp(-a)} = \frac{1-exp(-2a)}{1+exp(-2a)}$$

(35)

- **usmerňovač (rectifier) alebo ReLU (Rectified Linear Unit)** - táto aktivačná funkcia je pravdepodobne najbližšie k jej biologickému náprotivku. Je to kombinácia funkcie identity a funkcie hraničnej aktivity.

Existujú rôzne varianty aktivačnej funkcie ReLU, ako napríklad Noisy ReLU, Leaky ReLU alebo ELU (Exponential Linear Unit).

$$f(a) = \begin{cases} a \text{ if } a \geq 0 \\ 0 \text{ if } a < 0 \end{cases} \tag{36}$$

Aktivačná funkcia identity alebo hraničná funkcia boli používané pri vzniku neurónových sietí s implementáciami ako perceptron alebo Adaline (adaptívny lineárny neurón), ale neskôr sa prestali presadzovať v prospech logistického sigmoidu, hyperbolického tangensu alebo ReLU a jeho variácií. Posledné tri aktivačné funkcie sa líšia v nasledujúcich dvoch aspektoch:

Ich rozsah je odlišný.

Rozsah logistickej funkcie je (0,1), čo je jeden z dôvodov, prečo je táto funkcia preferovaná pre stochastické siete (siete s neurónmi, ktoré sa môžu aktivovať na základe pravdepodobnostnej funkcie). Hyperbolická funkcia je veľmi podobná logistickej funkcii, ale jej rozsah je (-1, 1). Naproti tomu ReLU má rozsah (0, ∞).

Ich derivácie sa počas trénovania správajú odlišne.

- V prípade logistickej funkcie f je derivácia f * (1-f).
- Ak je funkcia f hyperbolický tangens, jej derivácia je (1+f) * (1-f).
- Ak je funkcia f ReLU funkcia, jej derivácia je jednoduchá:

$$f'(a) = \begin{cases} 1 \; if \; a \geq 0 \\ 0 \; if \; a < 0 \end{cases} \tag{37}$$

Výhodou ReLU funkcie je, že jej derivácia je konštantná a nemá tendenciu klesať k nule, keď sa zväčšuje.

5.2 Rekurentné neurónové siete

Rekurentné neurónové siete (angl. Recurrent Neural Networks - RNN) pracujú so sekvenciami s premenlivou dĺžkou a definujú rekurentný vzťah nad týmito sekvenciami:

$$s_t = f(s_{t-1}, x_t)$$

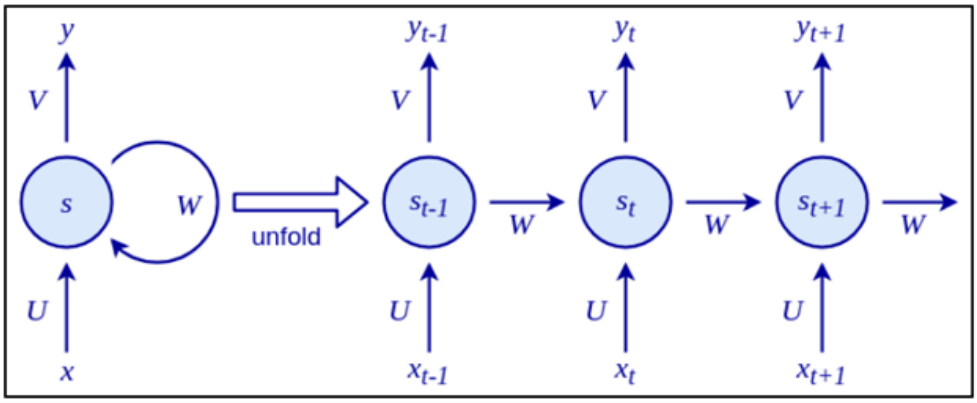
(38)

kde f je diferencovateľná funkcia, st je vektor hodnôt nazývaný vnútorný stav siete (v kroku t) a xt je vstup siete v kroku t. Na rozdiel od bežných sietí, kde stav závisí len od aktuálneho vstupu (a váh siete), tu je st funkciou oboch vstupov aktuálneho vstupu, ako aj predchádzajúceho stavu st-1. Stav st-1 môžeme považovať za súhrn všetkých predchádzajúcich vstupov siete.

Vďaka schopnosti spracovať ľubovoľné sekvencie vstupov sú rekurentné NN používané pri spracovaní prirodzené jazyka (NLP) a pri úlohách rozpoznávania reči. V skutočnosti sa RNN dajú použiť na akýkoľvek problém, pretože bolo dokázané, že sú turingovo úplné - teda teoreticky môžu simulovať akýkoľvek program, ktorý by bežný počítač nebol schopný vypočítať. Napríklad, DeepMind spoločnosti Google navrhol model s názvom Differentiable Neural Computer, ktorý sa dokáže naučiť a vykonávať jednoduché algoritmy, napríklad zoraďovanie.

Príkladom údajov s variabilnou dĺžkou sú slová vo vete alebo cena akcií v rôznych v časových intervaloch. Rekurentné NN dostali svoj názov z dôvodu opakovania tej istej funkcie nad sekvenciou.

Vzťah rekurencie je definovaný podľa toho ako sa stav vyvíja, krok za krokom cez sekvenciu prostredníctvom spätnej väzby cez predchádzajúce stavy, ako je znázornené na nasledujúcom diagrame:



Obrázok 75 Ukážka vzťahov v rekurentnej neurónovej sieti

RNN má tri sady parametrov (alebo váh):

- U transformuje vstup xt na stav st
- W transformuje predchádzajúci stav st-1 na aktuálny stav st
- V mapuje (transformuje) novo vypočítaný vnútorný stav st na výstup y

Parametre U, V a W aplikujú lineárnu transformáciu na svoje príslušné vstupy. Najzákladnejší prípad transformácie je vážený súčet. Teraz môžeme definovať vnútorný stav a výstup siete nasledovne:

$$s_t = f(s_{t-1} * W + x_t * U)$$

$$y_t = s_t * V$$

(39)

kde f je nelineárna aktivačná funkcia (napríklad hyperbolický tangens, sigmoid alebo ReLU).

V jazykovom modeli na úrovni slov bude vstupom x postupnosť slov zakódovaná vo vstupných vektoroch (x1 ... xt ...). Stav s bude postupnosť stavových vektorov (s1 ... st ...) a výstup y bude postupnosť pravdepodobnostných vektorov (y1 ... yt ...) nasledujúcich slov v postupnosti.

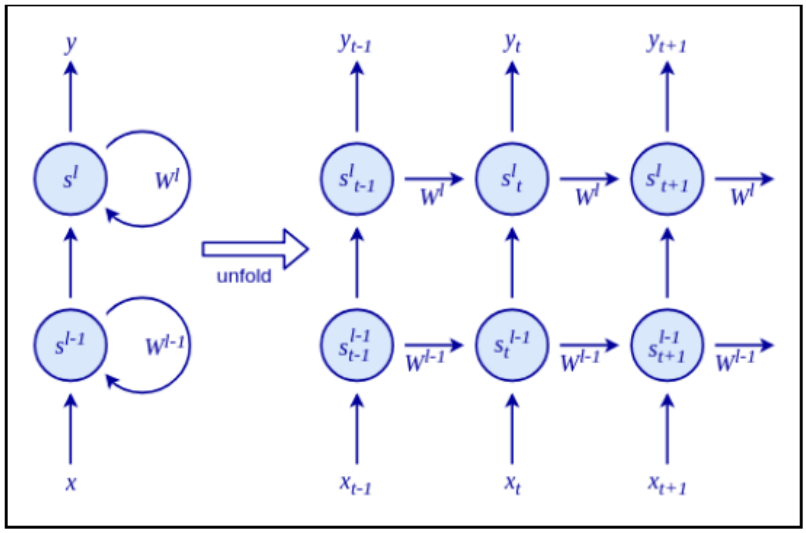
V RNN je každý stav závislý na všetkých predchádzajúcich výpočtoch prostredníctvom rekurentného vzťahu. Dôležitým dôsledkom toho je, že RNN majú pamäť v čase, pretože stavy s obsahujú informácie založené na predchádzajúcich krokoch. Dokonca RNN si môžu pamätať informácie na ľubovoľne dlhé obdobie, ale v praxi sú obmedzené na pamäť len niekoľkých krokov dozadu.

Rovnako ako pri bežných neurónových sieťach, môžeme navrstviť viac RNN a vytvoriť tak navrstvenú RNN (stacked RNN). Stav bunky slt RNN na úrovni l v čase t prevezme výstup ytl-1 bunky RNN z úrovne l-1 a predchádzajúci stav bunky slt-1 bunky RNN na rovnakej úrovni l ako vstup:

$$s_t^l = f(s_{t-1}^l, y_t^{l-1})$$

(40)

Na nasledujúcom obrázku vidíme rozvinutú, navrstvenú RNN:

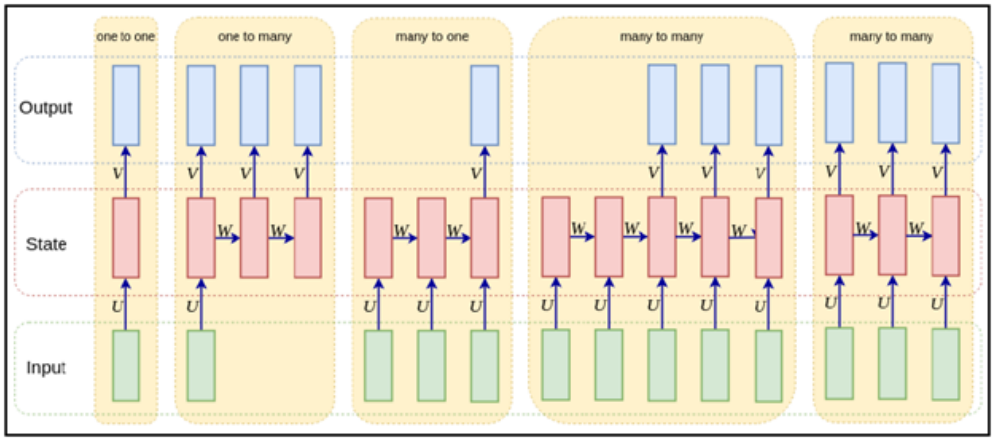


Obrázok 76 Navrstvená rekurentná neurónová sieť

Keďže RNN nie sú obmedzené na spracovanie vstupov fixnej veľkosti, výrazne rozširujú možnosti toho, čo môžeme pomocou neurónových sietí vypočítať, napríklad sekvencie rôznych dĺžok alebo obrázky rôznych veľkostí. Uvedieme niekoľko kombinácií spracovania:

- One-to-one: ide o nesekvenčné spracovanie, ako je napríklad dopredná neurónová sieť a konvolučné neurónové siete. Nie je veľa rozdielov medzi doprednou sieťou a použitím RNN na jeden časový krok. Príkladom jednosmerného spracovania je klasifikácia obrazu.
- One-to-many: generuje sekvenciu na základe jedného vstupu, napr. generovanie titulkov z obrázka.
- Many-to-one: výstupom je jeden výsledok na základe sekvencie, napr. klasifikácia sentimentu z textu.
- Many-to-many indirect: sekvencia je zakódovaná do stavového vektora, po ktorom sa sa tento stavový vektor dekóduje do novej sekvencie, napr. preklad jazyka.
- Many-to-many direct: Tento postup vypisuje výsledok pre každý vstupný krok, napr., označovanie foném v rámci rozpoznávania reči.

Nasledujúci obrázok znázorňuje predchádzajúce kombinácie vstupov a výstupov:



Obrázok 77 Kombinácie vstupov a výstupov v RNN

1.1.1 Implementácia a trénovanie rekurentnej NN

Úloha: Naučiť RNN počítať jednotky v sekvencii.

Ukážeme pomocou jazyka Python a knižnice NumPy ako naučiť jednoduchú RNN počítať počet jednotiek na vstupe a výsledok zobraziť na konci postupnosti. Je to príklad "many-to-one" vzťahu.

Príklad vstupu a výstupu je nasledujúci:

Vstup: (0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0)

Výstup: 3

Sieť bude mať len dva parametre: vstupnú váhu U a váhu rekurentnosti W. Výstupná váha V je nastavená na 1, aby sme ako výstup y prečítali len posledný stav.

Prvým krokom je import knižnice numpy, definovanie trénovacích dáta x a značky y. x je dvojrozmerné pole, keďže prvá dimenzia predstavuje vzorku v našom malom dátovom sete (použijeme malú sadu s jednou vzorkou):

```
import numpy as np

# The first dimension represents the mini-batch
x = np.array([[0, 0, 0, 0, 1, 0, 1, 0, 1, 0]])

y = np.array([3])
```

Vzťah rekurentnosti definovaný touto sieťou je st = st-1 * W + xt * U. Ide o lineárny model preto rekurentný vzťah implementujeme nasledovne:

```
def step(s, x, U, W):
  return x * U + s *
```

Stavy st a váhy W a U sú jednotlivé skalárne hodnoty. Už len získať súčet vstupov v celej postupnosti. Ak nastavíme U=1, potom vždy, keď je prijatý vstup dostaneme jeho plnú hodnotu. Ak nastavíme W=1, potom hodnota, ktorú by sme akumulovali by sa nikdy nezmížila. V tomto príklade by sme teda dostali požadovaný výstup: 3.

Spätné šírenie v čase

Spätné šírenie (angl. Back propagation) v čase je typický algoritmus, ktorý sa používa na trénovanie rekurentných sietí.

Rozdiel medzi bežným spätným šírením a spätným šírením v čase spočíva v tom, že rekurentná NN sa rozvíja v čase po určitý počet časových krokov.[5] Po dokončení procesu rozkladu dostávame model, ktorý je dosť podobný bežnej viacvrstvovej doprednej NN. Jedna skrytá vrstva tejto siete predstavuje jeden krok v čase. Jediné rozdiely sú v tom, že každá vrstva má viacero vstupov: predchádzajúci stav st-1 a aktuálny vstup xt. Parametre U a W sú spoločné pre všetky skryté vrstvy.

Dopredný prechod rozvíja RNN pozdĺž sekvencie a vytvára zásobník stavov pre každý krok. Nasleduje implementácia dopredného prechodu, ktorý vracia aktiváciu s pre každý rekurentný krok a každú vzorku v sade:

```
def forward(x, U, W):
    # Number of samples in the mini-batch
    number_of_samples = len(x)

    # Length of each sample
    sequence_length = len(x[0])

    # Initialize the state activation for each sample along the sequence
    s = np.zeros((number_of_samples, sequence_length + 1))

    # Update the states over the sequence
    for t in range(0, sequence_length):
        s[:, t + 1] = step(s[:, t], x[:, t], U, W)  # step function

    return s
```

Keď máme krok vpred(step forward) a funkciu chybovosti (loss function), môžeme definovať gradient šírenia dozadu (späť). Keďže rozvinutá RNN je ekvivalentná bežnej doprednej sieti, môžeme použiť reťazové pravidlo.

Váhy W a U sú spoločné pre všetky vrstvy, čím budeme akumulovať chybu derivácie pre každý rekurentný krok, pričom na konci budeme aktualizovať váhy pomocou akumulovanou hodnotou.

Implementácia spätného prechodu je nasledovná:

Gradienty pre U a W sa akumulujú v gU a gW:

```
def backward(x, s, y, W):
    sequence_length = len(x[0])

    # The network output is just the last activation of sequence
    s_t = s[:, -1]

    # Compute the gradient of the output w.r.t. MSE cost function
at final state
    gS = 2 * (s_t - y)

    # Set the gradient accumulations to 0
    gU, gW = 0, 0

    # Accumulate gradients backwards


   for k in range(sequence_length, 0, -1):
       # Compute the parameter gradients and accumulate the
results.
       gU += np.sum(gS * x[:, k - 1])
       gW += np.sum(gS * s[:, k - 1])

       # Compute the gradient at the output of the previous layer
       gS = gS * W

   return gU, gW
```

Použijeme gradientný zostup na optimalizáciu našej siete. Použijeme strednú kvadratickú chybu:

```
def train(x, y, epochs, learning_rate=0.0005):
    """Train the network"""

    # Set initial parameters
    weights = (-2, 0)  # (U, W)

    # Accumulate the losses and their respective weights
    losses = list()
    weights_u = list()
    weights_w = list()

    # Perform iterative gradient descent
    for i in range(epochs):
        # Perform forward and backward pass to get the gradients
        s = forward(x, weights[0], weights[1])

        # Compute the MSE cost function
        loss = (y[0] - s[-1, -1]) ** 2

        # Store the loss and weights values for later display
        losses.append(loss)

        weights_u.append(weights[0])
        weights_w.append(weights[1])

        gradients = backward(x, s, y, weights[1])

        # Update each parameter `p` by p = p - (gradient *
learning_rate).
        # `gp` is the gradient of parameter `p`
        weights = tuple((p - gp * learning_rate) for p, gp in
zip(weights, gradients))

    print(weights)
```

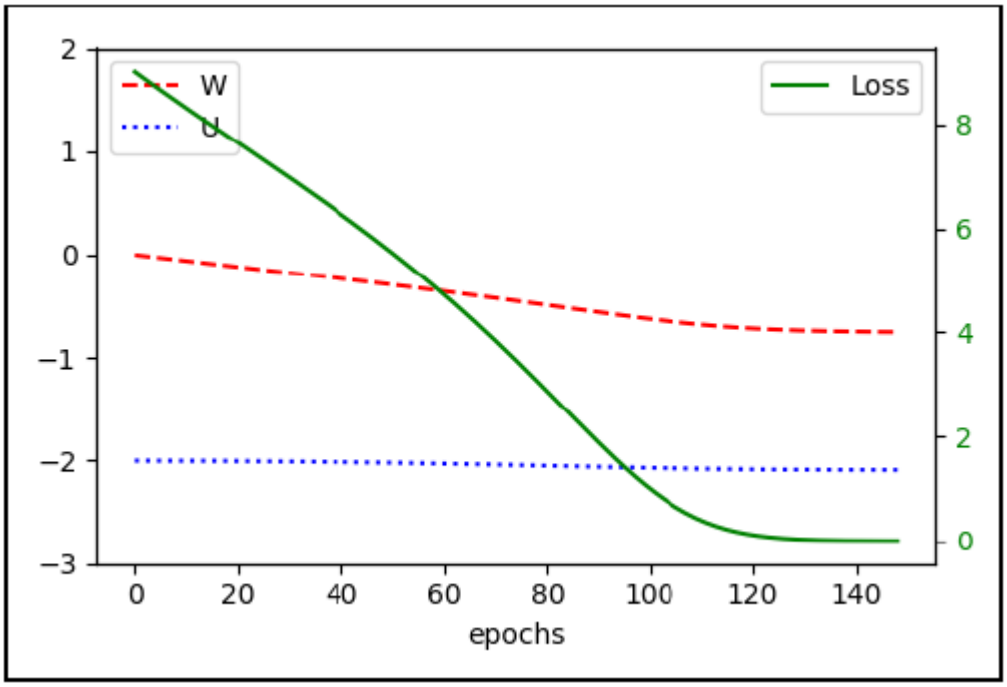
Ďalej implementujeme súvisiacu funkciu plot_training, ktorá zobrazí váhy a stratu:

```
def plot_training(losses, weights_u, weights_w):
    import matplotlib.pyplot as plt

    # remove nan and inf values
    losses = losses[~np.isnan(losses)][:-1]
    weights_u = weights_u[~np.isnan(weights_u)][:-1]
    weights_w = weights_w[~np.isnan(weights_w)][:-1]

    # plot the weights U and W
    fig, ax1 = plt.subplots(figsize=(5, 3.4))

    ax1.set_ylim(-3, 2)
    ax1.set_xlabel('epochs')
    ax1.plot(weights_w, label='W', color='red', linestyle='--')
    ax1.plot(weights_u, label='U', color='blue', linestyle=':')
    ax1.legend(loc='upper left')

    # instantiate a second axis that shares the same x-axis
    # plot the loss on the second axis
    ax2 = ax1.twinx()

    # uncomment to plot exploding gradients
    ax2.set_ylim(-3, 200)
    ax2.plot(losses, label='Loss', color='green')
    ax2.tick_params(axis='y', labelcolor='green')
    ax2.legend(loc='upper right')

    fig.tight_layout()

    plt.show()
```

Spustíme nasledujúci kód:

```
losses, weights_u, weights_w = train(x, y, epochs=150)
plot_training(losses, weights_u, weights_w)
```

Výsledkom bude nasledujúci graf:

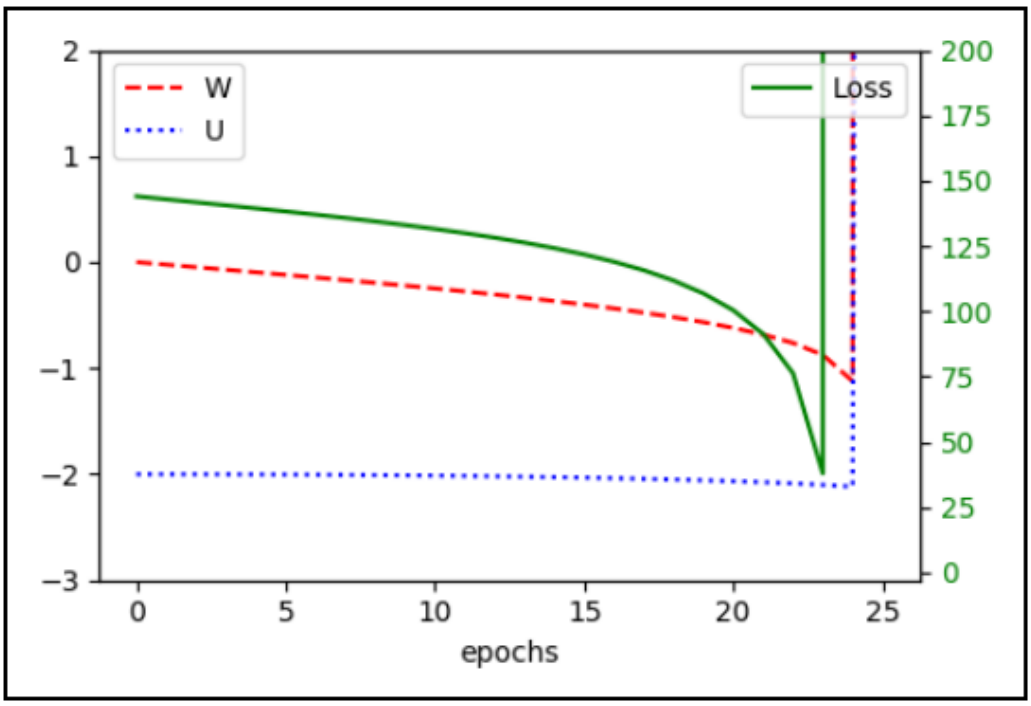


Obrázok 78 RNN strata

Vo funkcii plot_training je potrebné nastaviť ax2.set_ylim(-3, 200) z pôvodného ax2.set_ylim(-3, 10).

Výstupom je varovanie:

```
rnn_example.py:5: RuntimeWarning: overflow encountered in
multiply return x * U + s * W

rnn_example.py:36: RuntimeWarning: invalid value encountered
in multiply gU += np.sum(gS * x[:, k - 1])

rnn_example.py:37: RuntimeWarning: invalid value encountered
in multiply gW += np.sum(gS * s[:, k - 1])
```



Obrázok 79 Prípad explodujúceho gradientu

Váhy sa pomaly približujú k optimálnej hodnote a strata klesá, až kým neprekročí hodnotu v epochu 23 (presná epocha však nie je dôležitá). Nastane situácia, že cena plochy, na ktorej trénujeme je veľmi nestabilná. Pomocou malých krokov sa môžeme presunúť do stabilnej časti cenovej funkcie, kde je gradient nízky, a zrazu narazíme na skok v cenách a zodpovedajúci obrovský gradient. Keďže gradient je tak rýchlo vzrastajúci, bude mať veľký vplyv na naše váhy prostredníctvom aktualizácií váh - stanú sa z nich NaN (Not a Number) (ako ilustruje skok mimo grafu). Tento problém je známy ako rýchlo vzrastajúci (explodujúci) gradient.

Existuje tiež problém zanikajúcich (opak explodujúcich) gradientov. Gradient sa exponenciálne rozpadá v priebehu počtu krokov až do bodu, keď sa v skorších stavoch stáva extrémne malým. V podstate sú zatienené väčšími gradientmi z novších časových krokov a schopnosť siete zachovať históriu týchto skorších stavov sa vytráca. Tento problém je ťažšie odhaliť, pretože trénovanie bude stále prebiehať a sieť bude produkovať platné výstupy (na rozdiel od rýchlo vzrastajúce gradientov). Len nebude schopná učiť sa dlhodobé závislosti.

Hoci zanikajúce a rýchlo vzrastajúce gradienty sú prítomné v bežných neurónových sieťach, v RNN sú obzvlášť výrazné. Dôvody sú nasledovné:

V závislosti od dĺžky sekvencie môže byť rozvinutá RNN oveľa hlbšia v porovnaní s bežnou sieťou.

Váhy W sú spoločné pre všetky kroky. To znamená, že rekurentný vzťah, ktorý šíri gradient späť v čase tvorí geometrickú postupnosť:

$$\frac{\partial s_t}{\partial s_{t-m}} = \frac{\frac{\partial s_t}{\partial s_{t-1}} * \ldots * \partial s_{t-m+1}}{\partial s_{t-m}} = W^m$$

(41)

V našej jednoduchej lineárnej RNN rastie gradient exponenciálne, ak |W| > 1 (exploduje gradient). Napríklad 50 časových krokov v priebehu W=1,5 je W50 = 1,550 ≈ 6 * 108. Gradient sa exponenciálne zmenšuje, ak |W| <1 (zanikajúci gradient). Napríklad 20 časových krokov na W=0,6 je W20 = 0,620 ≈ 3*10-5. Ak je váhový parameter W matica a nie skalár, tento rýchlo vzrastajúci alebo zanikajúci gradient súvisí s najväčšou vlastnou hodnotou (ρ) W (známou aj ako spektrálny polomer). Ak ρ < 1, gradienty sa strácajú a ak ρ > 1, gradienty rýchlo vzrastajú.

6 Neurónové jazykové modely

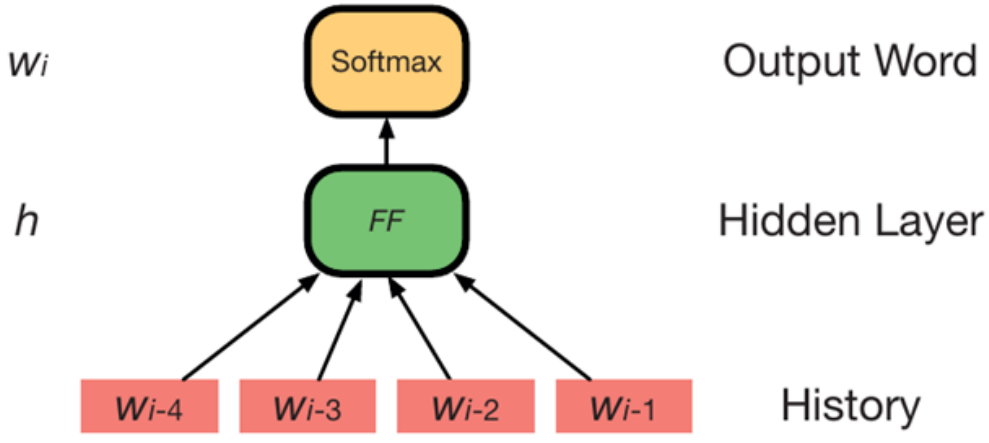
6.1 Neurónové jazykové modely

Neurónové siete sú veľmi efektívnou metódou na modelovanie podmienených pravdepodobnostných rozdelení s viacerými vstupmi p(d|a,b,c). Sú spoľahlivé na nepozorované dátové body - napríklad na nepozorované (a,b,c,d) v trénovacích dátach. Pomocou tradičných metód štatistického odhadu môžeme riešiť problém s rozptýlenými údajmi pomocou vynechania a zhlukovania (ktorú časť kontextu podmienených údajov treba vynechať ako prvú a koľko zhlukov).

N-gramové jazykové modely redukujú pravdepodobnosť vety na súčin pravdepodobností slov v kontexte niekoľkých predchádzajúcich slov, napr. štyroch - p(wi |wi-4,wi-3,wi-2,wi-1). Takéto modely sú najlepším príkladom pre podmienené rozdelenie pravdepodobnosti s rozsiahlym podmieňujúcim kontextom, pre ktoré nám často chýbajú dátové body a chceli by sme informácie zoskupiť. V štatistických jazykových modeloch sa používajú zložité diskontné a back-off schémy na vyváženie bohatých zdrojov z modelov nižšieho rádu - napr. bigramového modelu p(wi |wi-1)-s riedkymi odhadmi z modelov vyššieho rádu.

6.1.1 Dopredné neurónové jazykové modely

Ukážka 5-gramovej neurónovej siete jazykového modelu. Uzly siete reprezentujúce kontextové slová majú spojenie do skrytej vrstvy, ktorá je spojená s výstupnou vrstvou pre predpovedané slovo.



Obrázok 80 Neurónový jazykový model

6.1.2 Reprezentácia slov

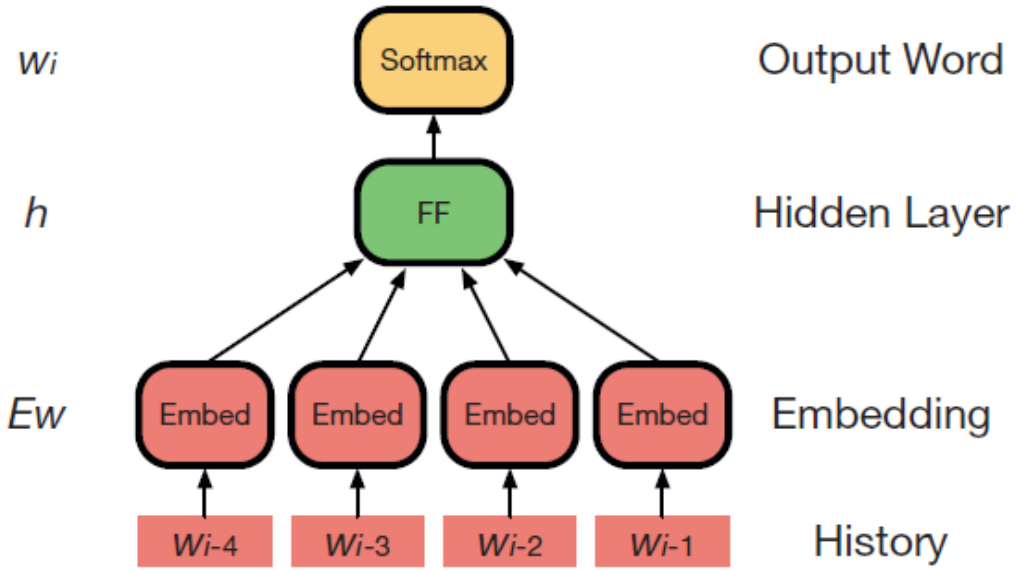
Uzly v neurónovej sieti nadobúdajú hodnoty reálnych čísel, avšak slová sú diskrétne položky z rozsiahlej slovnej zásoby. Nemôžeme použiť identifikátory tokenov, pretože neurónová sieť bude predpokladať, že token 124 je veľmi podobný tokenu 125 - zatiaľ čo v praxi sú tieto čísla úplne náhodné. Rovnaký argument platí aj pre použitie bitového kódovania pre identifikátory tokenov. Slová (1,1,1,1,1,0,0,0,0)T a (1,1,1,1,1,0,0,0,1)T majú veľmi podobné kódovanie, ale nemusia mať navzájom nič spoločné.

Každé slovo budeme reprezentovať viacrozmerným vektorom s jednou dimenziou pre každé slovo v slovníku. Hodnotou 1 pre dimenziu, ktorá zodpovedá danému slovu a 0 pre ostatné. Tento typ vektorov sa nazýva jeden horúci vektor (one-hot vector). Napríklad:

- pes = (0,0,0,0,1,0,0,0,0,...)T,
- mačka = (0,0,0,0,0,0,0,1,0,...)T,
- jesť = (0,1,0,0,0,0,0,0,0,...)T.

Keďže Ide o veľmi rozsiahle vektory, budeme musieť riešiť problémy spojené s touto voľbou reprezentácie slov. Jedným z dočasných riešení je obmedziť slovnú zásobu, napríklad na 20 000 najfrekventovanejších slov a spojiť všetky ostatné slová do iného tokenu. Mohli by sme tiež použiť triedy slov (buď automatické zhluky alebo lingvisticky motivované triedy, ako napríklad PoS (part-of-speech) tagy) na zníženie dimenzionality vektorov.

Na zhromažďovanie informácií medzi slovami zavedieme ďalšiu vrstvu medzi vstupnou a skrytou vrstvou. V tejto vrstve sa každé kontextové slovo individuálne premieta do priestoru s nižšou dimenziou. Pre každé z kontextových slov používame rovnakú váhovú maticu, čím vytvárame spojitú priestorovú reprezentáciu pre každé slovo nezávisle od jeho pozície v podmieňujúcom kontexte. Táto reprezentácia sa bežne označuje ako vnorené slovo (angl. word embedding).



Obrázok 81 Architektúra celého dopredného neurónového jazykového modelu

Slová, ktoré sa vyskytujú v podobných kontextoch, by mali mať podobné vnorené slová. Napríklad, ak údaje na trénovanie jazykového modelu často obsahujú n-gramy:

- ale milý pes skočil,
- ale milá mačka skočila,
- dieťa mačku pevne objalo,
- dieťa pevne objalo psa,
- radi pozerajú videá s mačkami,
- rád pozerá videá so psami,

potom by jazykový model využíval poznatok, že pes a mačka sa vyskytujú v podobných kontextoch, čím sú do istej miery zameniteľné. Ak by sme chceli predpovedať na základe kontextu, v ktorom sa vyskytuje pes, ale videli sme tento kontext len so slovom mačka, potom by sme to stále chceli považovať za pozitívny znak. Vnorené slová umožňujú zovšeobecnenie medzi slovami (zhlukovanie), a teda mať robustné predpovede v nevidených kontextoch (back off).

6.2 Architektúra neurónovej siete

Na obrázku (hore) je znázornená architektúra plnohodnotného jazykového modelu doprednej neurónovej siete, ktorý pozostáva z kontextových slov ako

vstupnej vrstvy s jedným horúcim vektorom, vrstvy vnorených slov (angl. embedding layer), skrytej vrstvy a vrstvy predpovedaných výstupných slov.

Kontextové slová sú najprv zakódované ako one-hot vektory. Následne prejdú cez maticu vnorení E, čím vznikne vektor čísel s pohyblivou desatinnou čiarkou, tzv. vnorené slovo. Tento vnorený vektor má zvyčajne 500 alebo 1 000 uzlov. Pričom používame rovnakú vnorenú maticu E pre všetky kontextové slová.

Z matematického hľadiska sa v tomto prípade toho až tak veľa nedeje. Keďže vstupom do násobenia do maticu E je one-hot vektor, väčšina vstupných hodnôt pre násobenie matice sú nuly. V podstate vyberáme jeden stĺpec matice, ktorý zodpovedá ID-čku vstupného slova. Z toho vyplýva, že aktivácia je zbytočná funkcia. V istom zmysle je matica vnorení vyhľadávacou tabuľkou E(wj) pre vnorenie slov, indexovaná ID-čkom slova wj:

$$E(w_j) = E\ w_j.$$

Mapovanie na skrytú vrstvu v modeli si vyžaduje spojenie všetkých kontextových vnorených slov E(wj) ako vstup do klasickej doprednej vrstvy, napríklad s použitím hyperbolického tangensu (tanh) ako aktivačnej funkcie:

$$h = \tanh\left(b_h + \sum_j H_j E(w_j)\right).$$

Výstupná vrstva sa interpretuje ako rozdelenie pravdepodobnosti nad slovami. Najprv sa vytvorí lineárna kombinácia si váh wij a hodnoty skrytých uzlov hj sa vypočítavajú pre každý uzol i:

$$s = W\ h.$$

Aby sme sa uistili, že ide o správne rozdelenie pravdepodobnosti, použijeme aktivačnú funkciu softmax, aby sme zabezpečili, že všetky hodnoty sa budú rovnať jednej:

$$p_i = \text{softmax}(s_i, \vec{s}) = \frac{e^{s_i}}{\sum_j e^{s_j}}.$$

Je to blízke neurónovému pravdepodobnostnému jazykovému modelu, ktorý navrhli Bengio a kol. (2003). Tento model mal ešte jeden problém, pridával priame spojenia kontextových slovných vnorení do výstupu slova, pričom pre každé slovo wj pridával do slovných vnorení E (wj) po lineárnej transformácii s váhovou maticou Uj. Rovnica s = W h sa teda nahradí rovnicou:

$$s = W\,h + \sum_j U_j\,E(w_j).$$

V ich práci sa uvádza, že takéto priame spojenia z kontextových slov k výstupným slovám urýchľuje trénovanie, hoci v konečnom dôsledku nezlepšuje výkon. Nazývajú sa aj reziduálne spojenia (angl. residual connections), preskočené spojenia (skip connections) alebo dokonca hlavný ťah spojení (highway connections).

6.2.1 Trénovanie

Parametre neurónového jazykového modelu (matica vnorených slov, váhové matice, vektory odchýlok) trénujeme spracovaním všetkých n-gramov v trénovacom korpuse. Pre každý n-gram vložíme do siete kontextové slová a porovnáme výstup siete s jedným horúcim vektorom správneho slova, ktoré sa má predpovedať. Váhy sa aktualizujú pomocou spätného šírenia.

Jazykové modely sa bežne hodnotia pomocou perplexity, ktorá súvisí s pravdepodobnosťou prisudzovanou k správnemu východiskovému textu. Preto cieľom trénovania jazykových modelov je zvýšiť pravdepodobnosť trénovaných údajov.

Počas trénovania, vzhľadom na kontext x = (wn-4,wn-3,wn-2,wn-1), máme správnu hodnotu pre jeden horúci vektor →y . Pre každý trénovací príklad (x, →y) je cieľ trénovania definovaný na základe zápornej logaritmickej pravdepodobnosti ako

$$L(\mathbf{x}, \vec{y}; W) = -\sum_k y_k \log p_k.$$

Jediná hodnota yk je rovná 1, ostatné sú rovné 0. Ide o pravdepodobnosť pk priradenú správnemu slovu k. Definovanie nám umožňuje aktualizovať všetky váhy, vrátane tých, ktoré vedú k nesprávnym výstupným slovám.

6.3 Vnorenie slov

V NLP oblasti sa pojem vnorenie slov (angl. word embedding) používa na reprezentáciu slov pri analýze textu, zvyčajne vo forme vektora s reálnymi hodnotami, ktorý kóduje význam slova tak, že sa očakáva, že slová, ktoré sú v priestore vektora bližšie, budú mať podobný význam.

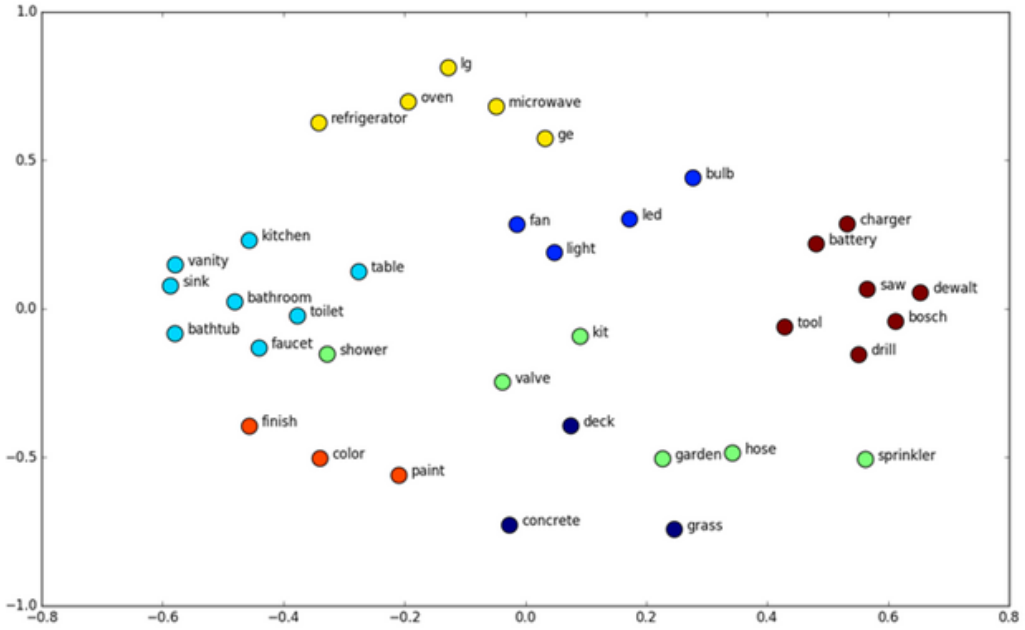
Vnorenie slov má dôležitú úlohu v neurónových jazykových modeloch. Predstavujú kontextové slová, ktoré umožňujú predikovať nasledujúce slovo v sekvencii. Napríklad:

- ale milý pes skočil,
- ale milá mačka skočila.

Keďže pes a mačka sa vyskytujú v podobných kontextoch, ich vplyv na predikovanie slova “skočil” by mal byť podobný. Mal by sa líšiť od slov ako “auto”, ktoré pravdepodobne nevyvolajú dokončenie “skočil”. Myšlienka, že slová, ktoré sa vyskytujú v podobných kontextoch, sú si sémanticky podobné je významnou myšlienkou v lexikálnej sémantike.

Význam a sémantika sú pomerne zložité pojmy s prevažne distribučnými lexikálnymi sémantickými nevyriešenými definíciami. Podstata distribučnej lexikálnej sémantiky spočíva v definovaní slov na základe ich distribučných vlastností, t. j. v akých kontextoch sa vyskytujú. Slová, ktoré sa vyskytujú v podobných kontextoch (pes a mačka), by mali mať podobné reprezentácie. Vo vektorových priestorových modeloch, ako sú vnorené slová sa podobnosť môže merať funkciou vzdialenosti, napr. kosínusovou vzdialenosťou - uhlom medzi vektormi.

Ak premietneme viacrozmerné vnorené slová do dvoch rozmerov, môžeme vnorené slová vizualizovať tak, ako je to znázornené na obrázku. Slová, ktoré sú si podobné (dráma, divadlo, festival) sú zoskupené spolu.



Obrázok 82 Vnorenie slov zobrazené v dvoch dimenziách

6.4 Rekurentné neurónové jazykové modely

Dopredný neurónový jazykový model je schopný používať dlhšie kontexty ako tradičné štatistické back-off modely, pretože má flexibilnejšie prostriedky na riešenie neznámych kontextov - konkrétne použitie slovných vnorení na vyžitie podobných slov a robustné spracovanie nevidených slov v akejkoľvek kontextovej pozícii. Čo umožňuje podmieňovať oveľa väčšie kontexty ako tradičné štatistické modely, napr. 20-gramové modely.

Prípadne namiesto používania fixnej dĺžky kontextových slov môžu rekurentné neurónové siete podmieniť kontextové sekvencie ľubovoľnej dĺžky. Čo spočíva v opätovnom použití skrytej vrstvy, ktorá bola použitá pri predpovedaní slova wn, ako dodatočného vstupu na predpovedanie slova wn+1.

Spočiatku sa model nijako nelíši od dopredného neurónového modelu jazyka. Vstup do siete je prvé slovo vety w1 a druhá sada neurónov, ktoré na tomto mieste označujú začiatok vety. Vnorenie slova w1 a neuróny začiatku vety sa najprv mapujú do skrytej vrstvy h1, ktorá sa potom použije na predpovedanie výstupného slova w2.

Tento model používa rovnakú architektúru, t. j. slová (vstupné a výstupné) sú reprezentované jedným horúcim vektorom, vnorenými slovami a skrytou vrstvou, ktorá používa, napr. 500 neurónov s reálnou hodnotou. Používame

aktivačnú funkciu tanh na skrytej vrstve a funkciu softmax na výstupnej vrstve. Jedným zo vstupov do tretieho slova w3 v sekvencii je bezprostredne predchádzajúce (už známe) slovo w2. Avšak neuróny v sieti, ktoré sme použili na reprezentáciu začiatku vety, sú teraz naplnené hodnotami zo skrytej vrstvy predchádzajúcej predpovede slova w2.

V predchádzajúcej architektúre dopredného typu NN sa na predikciu i-teho slova vypočítal skrytý stav hi z predchádzajúcich slov E(wi-j), napr. troch slov. Toto používa zdieľanú maticu vnorených slov E a váhovú maticu Hj pre každé z predchádzajúcich slov, plus výraz odchýlky bh:

$$h_i = \tanh\left(b_h + \sum_{1 \le j \le 3} H_j E(w_{i-j})\right).$$

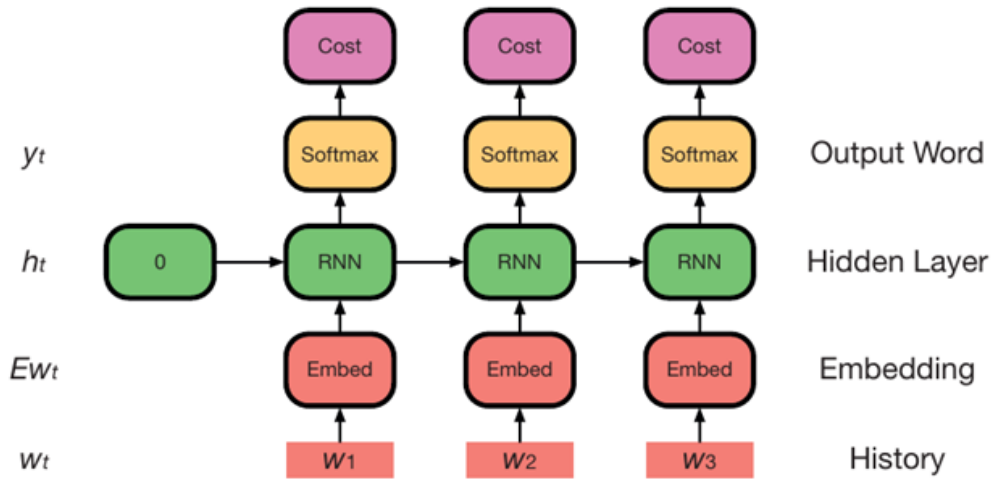
Upravením na rekurzívnu definíciu skombinujeme jedno predchádzajúce slovo wi-1 s predchádzajúcim skrytým stavom hi-1. Pridáme váhovú maticu V, aby sme parametrizovali mapovanie z predchádzajúceho skrytého stavu hi-1:

$$h_i = \tanh\Big(b_h + HE(w_{i-1}) + Vh_{i-1}\Big).$$

Neuróny v skrytom stave hi-1 kódujú kontext predchádzajúcej vety. V každom kroku sú obohatené o informácie o novom vstupnom slove, a teda sú podmienené celou históriou vety. Takže aj posledné slovo vety je čiastočne podmienené prvým slovom vety. Okrem toho je model jednoduchší: má menej váh ako 3-gramový dopredný neurónový model jazyka.

6.4.1 Trénovanie s ľubovoľne dlhými kontextami

V počiatočnej fáze (predpovedanie druhého slova na základe prvého) máme rovnakú architektúru, a teda aj rovnaký postup trénovania ako pri dopredných neurónových sieťach. Vyhodnotíme chybu vo výstupnej vrstve a aktualizácie šírime späť do vstupnej vrstvy. Takto by sme mohli spracovať každý trénovací príklad, v podstate by sme považovali skrytú vrstvu z predchádzajúceho tréningového príkladu ako fixný vstup aktuálneho príkladu. Týmto spôsobom však nikdy neposkytujeme spätnú väzbu k reprezentácii predchádzajúcej histórie v skrytej vrstve.



Obrázok 83 Ukážka spätného šírenia v čase

Postup trénovania so spätným šírením v čase rozvíja rekurentnú neurónovú sieť v pevne stanovenom počte krokov, pričom sa vracia späť napríklad k predikciám piatich slov.

Spätné šírenie v čase sa môže použiť pre každý trénovací príklad (časový krok), ale je to výpočtovo dosť náročné. Zakaždým sa musia vykonať výpočty v niekoľkých krokoch. Namiesto toho môžeme vypočítať a aplikovať aktualizácie váh po menších častiach. Najprv spracujeme väčší počet trénovacích príkladov (napríklad 10 až 20 alebo celú vetu) a potom aktualizujeme váhy.

Vzhľadom na moderný výpočtový výkon je úplné rozvinutie rekurentnej neurónovej siete bežnejšie. Zatiaľ čo rekurentné neurónové siete majú teoreticky ľubovoľnú dĺžku, vzhľadom na konkrétny trénovací príklad je jej veľkosť známa a pevne daná, takže môžeme plne skonštruovať graf výpočtu pre každý daný trénovací príklad, definovať chybu ako súčet chýb predpovede slov a potom vykonať spätné šírenie nad celou vetou. To si však vyžaduje, aby sme vedeli rýchlo zostaviť výpočtové grafy - dynamické výpočtové grafy - ktoré sú v súčasnosti niektorými sadami nástrojov podporované lepšie a inými horšie.

7 Neurónové translačné modely

7.1 Enkóder-dekóder

Neurónový model prekladu je priamym rozšírením jazykového modelu. Rekurentná neurónová sieť na modelovanie jazyka sa môže považovať za rovnorodý proces. Vzhľadom na všetky predchádzajúce slová takýto model predpovedá nasledujúce slovo. Keď sa dostaneme na koniec vety, pokračujeme teraz v predpovedaní prekladu vety, a to po jednom slove.

Ak chceme takýto model natrénovať, jednoducho spojíme vstup a výstup vety a použijeme rovnakú metódu ako na trénovanie jazykového modelu. Na dekódovanie vety vložíme vstupnú vetu a potom prejdeme predikciami modelu, kým nepredpokladá token na ukončenie vety (bodka, otáznik alebo výkričník).

7.1.1 Kódovacia fáza

Keď spracovanie dosiahne koniec vstupnej vety (po predpovedaní značky konca vety </s>), skrytý stav zakóduje jej význam. Vektor, ktorý nadobudne hodnoty uzlov tejto poslednej skrytej vrstvy je vnorenie vstupnej vety. Toto je kódovacia fáza modelu. Potom sa tento skrytý stav použije na vytvorenie prekladu vo fáze dekodéra.

7.1.1.1 Dekódovacia fáza

Počas fázy kódovania, sieť musí zahrnúť všetky informácie o vstupnej vete. Nemôže zabudnúť na prvé slová na konci vety. Počas dekódovacej fázy nie je potrebné, aby mala dostatok informácií na predpovedanie každého ďalšieho slova. Potrebné je mať určitý prehľad o tom, aká časť vstupnej vety už bola preložená a čo je ešte potrebné preložiť.

V praxi navrhované modely fungujú primerane dobre pri krátkych vetách (povedzme do 10 až 15 slov), ale pri dlhých vetách zlyhávajú. Boli navrhnuté niektoré menšie vylepšenia tohto modelu, napríklad použitie stavu vety ako vstupu do všetkých skrytých stavov dekódovacej fázy modelu. Tým sa dekodér štrukturálne líši od kodéra a znižuje sa časť záťaže zo skrytého stavu počas dekódovania, pretože si už nemusí pamätať vstup. Ďalším nápadom je obrátenie poradia výstupnej vety tak, aby posledné slová vstupných viet boli blízko posledných slov výstupnej vety.

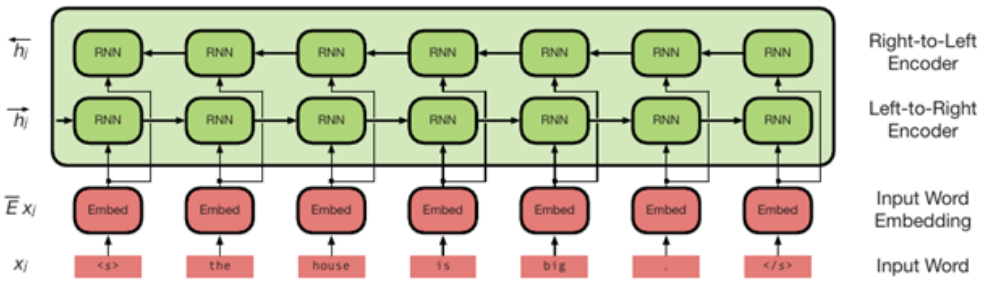
7.2 Zarovnávací model (Alignment model)

7.2.1 Zarovnávací model (Alignment model)

Prvým úspešným modelom neurónového strojového prekladu bol sequence-to-sequence (seq2seq) encoder–decoder s pozornosťou (attention). V podstate ide o model s explicitným mechanizmom zarovnávania. Vo svete hlbokého učenia sa toto zarovnanie nazýva pozornosť.

7.2.2 Enkóder

Úlohou kódera je poskytnúť reprezentáciu vstupnej vety. Vstupná veta je postupnosť slov, pre ktorú najprv získame vnorenú maticu (embedding matrix). Rovnako ako v jednoduchom jazykovom modeli, spracujeme tieto slová pomocou rekurentnej neurónovej siete. Výsledkom sú skryté stavy, ktoré kódujú každé slovo s jeho ľavým kontextom, t. j. všetkými predchádzajúcimi slovami. Na získanie správneho kontextu tiež vytvoríme rekurentnú neurónovú sieť, ktorá pracuje sprava doľava, resp. od konca vety na začiatok.



Obrázok 84 Sekvenčno-sekvenčný enkóder-dekóder model

Dve rekurentné neurónové siete pracujúce v dvoch smeroch sa nazývajú obojsmerná rekurentná neurónová sieť. Matematicky sa kódovač skladá z vnoreného vyhľadávania pre každé vstupné slovo xj a mapovania, ktoré prechádza skrytými stavmi →hj a →hj:

$$\overleftarrow{h_j} = f(\overleftarrow{h_{j+1}}, \bar{E}\, x_j)$$

$$\overrightarrow{h_j} = f(\overrightarrow{h_{j-1}}, \bar{E}\, x_j).$$

Vo vyššie uvedených rovniciach sme použili všeobecnú funkciu f pre bunku v rekurentnej neurónovej sieti. Táto funkcia môže byť jednoduchou vrstvou doprednej neurónovej siete - napríklad f(x) = tanh(Wx + b) - alebo zložitejšie bunky GRU alebo LSTM.

Tieto modely by sme mohli trénovať pridaním kroku, ktorý predpovedá ďalšie slovo v sekvencii, ale v skutočnosti ich trénujeme v kontexte celého modelu strojového prekladu. Obmedzenie opisu na dekodér, jeho výstupom je postupnosť reprezentácií slov, ktoré spájajú dva skryté stavy (→hj , →hj).

7.2.3 Dekóder

Dekodér je tiež rekurentná neurónová sieť. Preberá určitú reprezentáciu vstupného kontextu, predchádza skryté stavy a predikciu výstupného slova a generuje nový skrytý stav dekodéra a novú predikciu výstupného slova.

Matematicky vychádzame z rekurentnej neurónovej siete, ktorá si udržiava postupnosť skrytých stavov, ktoré sú vypočítané z predchádzajúceho skrytého stavu si-1 , vloženia predchádzajúceho výstupného slova Eyi-1 a vstupného kontextu ci:

$$s_i = f(s_{i-1},\ Ey_{i-1}, c_i).$$

Opäť existuje niekoľko možností voľby funkcie f, ktorá kombinuje tieto vstupy na generovanie ďalšieho skrytého stavu: lineárne transformácie s aktivačnou funkciou, GRU, LSTM atď. Najčastejšie sa voľba zhoduje s kóderom. Ak teda použijeme LSTM pre kodér, potom použijeme LSTM aj pre dekodér.

Zo skrytého stavu predpovedáme výstupné slovo. Táto predikcia má podobu rozdelenia pravdepodobnosti nad celým výstupným slovníkom. Ak máme slovnú zásobu, povedzme 50 000 slov, potom predikcia je 50 000-rozmerný vektor, ktorého každý prvok zodpovedá pravdepodobnosti predikovanej pre jedno slovo v slovníku.

Vektor predikcie ti je podmienený skrytým stavom dekodéra si-1 a opäť vnorením predchádzajúceho výstupného slova Eyi-1 a vstupného kontextu ci:

$$t_i = \text{softmax}\big(W(Us_{i-1} + VEy_{i-1} + Cc_i) + b\big).$$

Treba si uvedomiť, že opakujeme podmienenie na Eyi-1, pretože používame skrytý stav si-1 a nie si. Tým sa oddelí postup stavu kódera od si-1 po si od predikcie výstupného slova ti.

Aktivačná funkcia softmax sa používa na prevod nespracovaného vektora na rozdelenie pravdepodobnosti, kde súčet všetkých hodnôt je rovné 1. Najvyššia hodnota vo vektore označuje výstupný slovný token yi. Jeho vnorené slovo Eyi-1 informuje ďalší časový krok rekurentnej neurónovej siete.

Počas trénovania je známe správne výstupné slovo yi, takže trénovanie pokračuje s týmto slovom. Cieľom trénovania je poskytnúť čo najviac pravdepodobnostnej miery správnemu výstupnému slovu. Funkcia produkcie (cost function) ktorá riadi trénovanie, je teda záporný logaritmus pravdepodobnosti danej správnemu prekladu slova:

$$\text{cost} = -\log t_i[y_i].$$

V ideálnom prípade chceme dať správnemu slovu pravdepodobnosť 1, čo by znamenalo zápornú logaritmickú pravdepodobnosť 0, ale zvyčajne je to nižšia pravdepodobnosť, a teda aj vyššia cena. Všimnime si, že funkcia produkcie je viazaná na jednotlivé slová, celková cena vety je súčtom cien všetkých slov.

Počas odvodzovania novej testovacej vety sme zvyčajne vybrali slovo yi s najvyššou hodnotou v ti, t. j. najpravdepodobnejší preklad. Používame jeho vnorenie Eyi pre ďalšie kroky inferencie (odvodzovania).

7.2.4 Mechanizmus pozornosti

Mechanizmus pozornosti v hlbokom učení je technika používaná na zlepšenie výkonu neurónovej siete tým, že umožňuje modelu zamerať sa na najdôležitejšie vstupné údaje pri generovaní predpovedí.

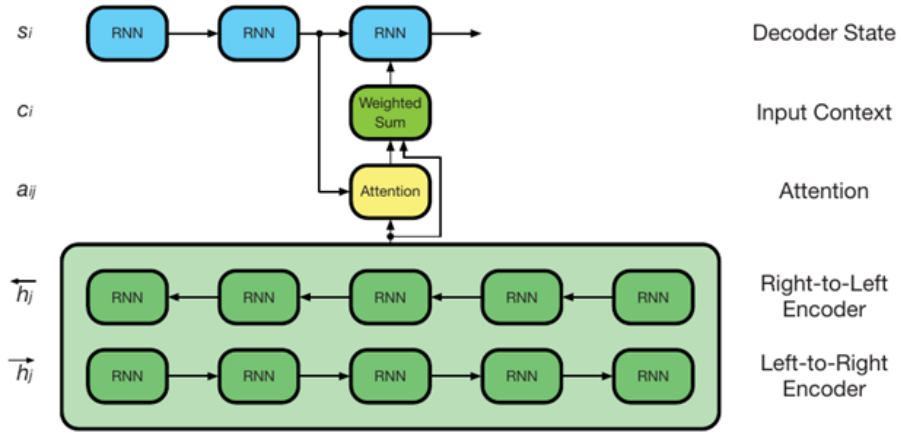
To sa dosiahne vážením vstupných údajov tak, že model uprednostňuje niektoré vstupné vlastnosti pred ostatnými. Výsledkom je, že model môže produkovať presnejšie predpovede tým, že zohľadní len najvýznamnejšie vstupné premenné.

Mechanizmus pozornosti sa často používa pri úlohách spracovania prirodzeného jazyka, ako je strojový preklad, kde model musí venovať pozornosť rôznym častiam vstupnej frázy, aby plne pochopil jej význam a poskytol vhodný preklad.

V súčasnosti máme dva otvorené konce. Kóder nám dal sekvenciu reprezentácií slov hj = (→hj , →hj) a dekodér očakáva kontext ci v každom kroku i. Teraz opíšeme mechanizmus pozornosti, ktorý spája tieto konce dohromady.

Mechanizmus pozornosti je ťažké ilustrovať pomocou našich typických neurónových siete, ale obrázok (dole) poskytuje aspoň predstavu o tom, čo sa na vstupe a výstupné vzťahy sú. Mechanizmus pozornosti je informovaný všetkými vstupnými reprezentáciami slov (→hj , →hj) a predchádzajúcim skrytým stavom dekodéra si-1 a vytvára kontextový stav ci.

Dôvodom je, že chceme vypočítať asociáciu medzi stavom dekodéra (ktorý obsahuje informáciu o tom, kde sa nachádzame na výstupe produkcii vety) a každým vstupným slovom. Na základe toho, ako silná je táto väzba alebo inými slovami, ako relevantné je každé konkrétne vstupné slovo na produkciu ďalšieho výstupného slova, chceme určiť aký má vplyv na jeho slovnú reprezentáciu.



Obrázok 85 NMT model - model pozornosti

Matematicky najprv vypočítame asociáciu s doprednou vrstvou pomocou váhových matíc (weight matrices) Wa , Ua a váhového vektora (weight vector) va):

$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j).$$

Výstupom tohto výpočtu je skalárna hodnota, ktorá udáva, ako dôležité je vstupné slovo j pre vytvorenie výstupného slova i.

Túto hodnotu pozornosti normalizujeme tak, aby sa hodnoty pozornosti v rámci všetkých vstupných slov j rovnali 1 pomocou softmax:

$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}.$$

Na záver použijeme normalizovanú hodnotu pozornosti na zváženie prínosu vstupnej reprezentácie slova hj ku kontextovému vektoru ci:

$$c_i = \sum_j \alpha_{ij} h_j.$$

Jednoduché sčítanie vektorov reprezentácie slov (vážených alebo nevážených) môže na prvý pohľad zdať zvláštne a zjednodušujúce. Je to však veľmi bežná prax v hlbokom učení v spracovaní prirodzeného jazyka.

8 Klasifikácia

Myšlienka klasifikácie je v princípe veľmi jednoduchá. Predstavme si, že nám niekto predkladá obrázky a my sa musíme rozhodnúť, či je na fotografii pes alebo mačka. Inými slovami, zviera na obrázku je určitý prípad a našimi vstupnými premennými sú vlastnosti obrázka, na základe ktorých sa snažíme sa určiť, do akej kategórie (pes alebo mačka) patrí. Kategórie sa niekedy označujú aj, ako triedy, štítky alebo označenia. My sme ale počas života videli veľa psov a mačiek, a preto je pre nás jednoduché určiť, čo sa na obrázku nachádza. Ako to ale naučíme počítač?

8.1 Úvod do klasifikácie

Aby sme mohli klasifikovať, potrebujeme teda dve veci. Klasifikátor, teda algoritmus, ktorý dokáže aplikovať klasifikáciu na nejaký dátový súbor. Predstavme si, že ideme riešiť jednu z typických úloh klasifikácie, a to či je správa hoax, alebo nie. Dátový súbor by mohol byť v našom prípade .csv súbor, v ktorom by sme mali v jednom stĺpci správu a v druhom stĺpci binárne označenie (0/1). V prípade, že je daná správa hoax, mala by označenie 1 a v prípade, že by daná spáva nebola hoax, mala by označenie 0.

V tejto chvíli potrebujeme náš model naučiť (natrénovať). Štandardne sa to robí takým spôsobom, že sa vezme nejaká časť dátového súboru (napríklad 75%) pomocou ktorej sa model bude učiť. Táto časť súboru sa nazýva aj trénovacia množina. Model sa bude učiť podobne, ako by to robil človek. Pozrie sa správu, prečíta si či je to hoax alebo nie a bude sa snažiť hľadať vzory medzi dátami aby sa naučil, ktoré správy vyzerajú, ako hoax. Keď sa proces trénovania ukončí, nasleduje druhá fáza, testovanie. V tejto fáze model dostane zvyšných 25% dát, ktoré sa označujú aj, ako testovacia množina a bude sa snažiť zaradiť správy do tried bez toho, aby sa pozeral na výstupy. Výsledkom modelu bude s akou pravdepodobnosťou model predpokladá, že daná správa je hoax.

8.1.1 Rozdelenie klasifikátorov

Klasifikátory môžeme rozdeliť podľa prístupu k učeniu do dvoch skupín:

- **Leniví študenti (Lazy Learners)** – Takéto algoritmy si najskôr uložia trénovaciu množinu, počkajú kým príde testovacia množina a zostavia si model. Takéto algoritmy venujú menej času trénovaniu ale viacej času im zaberie odhad triedy. Patrí sem napríklad algoritmus k-najbližších susedov (KNN).
- **Usilovní študenti (Eager Learners)** – Usilovní alebo dychtiví študenti si zostavia model ešte pred získaním testovacej množiny. Takéto algoritmy trávia viac času štúdiom a menej času predpovedaním triedy. Niektoré z príkladov sú umelé neurónové siete (ANN), Naivný Bayesov klasifikátor alebo rozhodovacie stromy.

8.1.2 Typy klasifikačných úloh

Klasifikačné úlohy môžeme rozdeliť do troch základných skupín:

- binárna klasifikácia,
- viactriedna klasifikácia,
- klasifikácia s viacerými štítkami.

8.1.2.1 Binárna klasifikácia

O binárnej klasifikácií sme hovorili už v predošlých príkladoch, keď sme si spomínali klasifikačné úlohy typu či je na obrázku pes alebo mačka alebo či je správa hoaxom. Ide teda o taký typ klasifikácie, kde náš prípad môžeme zaradiť len do dvoch tried. Napríklad, či je email spam. Vstupné premenné by predstavovali charakteristiky/vlastnosti emailu. Táto premenná by mohla byť reprezentovaná jednou z dvoch hodnôt (tried). V prípade, že by bol email spam, bola by mu priradená hodnota 1, v opačnom prípade by mu bola priradená hodnota 0.

8.1.2.2 Viactriedna klasifikácia

Viactriedna klasifikácia sa používa v prípadoch, že naša vstupná premenná môže nadobúdať viac, ako dve hodnoty. Typickou úlohou viactriednej klasifikácie je kategorizácia tvárí alebo druhov rastlín.

Predstavme si, že by sme chceli vytvoriť model, ktorý by dokázal určiť, o ktorý z kvetov ide. V takomto prípade by sme mali teda tri možné hodnoty premennej na výstupe, a preto hovoríme, že ide o viactriednu klasifikáciu.

8.1.2.3 Klasifikácia s viacerými štítkami

Predstavme si, že chceme klasifikovať objekty na fotografii. Na jednej fotografií sa môže nachádzať človek, stôl, pes atď. Oproti binárnej alebo viactriednej klasifikácií, pri ktorých sme predpokladali jediné označenie triedy, konkrétna fotografia teda môže mať na scéne viacero objektov.

8.2 Klasifikátory

Klasifikátory sú algoritmy, ktoré dokážu implementovať klasifikáciu. Medzi základné klasifikátory patrí:

- Logistická regresia,
- Naivný Bayesov klasifikátor,
- Klasifikátor K- najbližších susedov,
- Podporné vektorové stroje,
- Rozhodovací strom,
- Náhodný les.

8.2.1 Logistická regresia

Logistická regresia (Jammal, Ali-Hassan, El-Hallak, & El Morr, 2022) je klasifikačný algoritmus, ktorý možno rozdeliť do troch typov:

- **Binomická**: V binomickej logistickej regresii môžu existovať iba dve možné hodnoty závislej premennej, ako napríklad 0 alebo 1, vyhovelo alebo zlyhalo atď.
- **Multinomiálna**: V multinomiálnej logistickej regresii môžu existovať 3 alebo viac možných neusporiadaných hodnôt/kategórií závislej premennej. Závislá premenná nadobúda tri hodnoty/kategórie: mačka, pes, ovca, teda ide o polytomickú premennú,

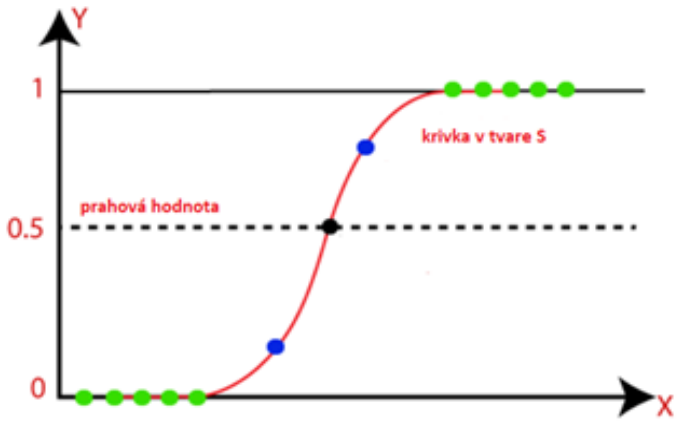
v predchádzajúcom prípade ide o dichotomickú premennú, čo majú spoločné je to, že sú nominálne, to znamená je tam iba rozlišovacia schopnosť, kým v nasledujúcom prípade viem zohľadniť aj poradie.

- **Ordinálna**: V ordinálnej logistickej regresii môžu existovať 3 alebo viac možných usporiadaných hodnôt/úrovní závislej premennej, ako napríklad "nízka", "stredná" alebo "vysoká".

Pomocou logistickej regresie by sme vedeli natrénovať model, ktorý by sa naučil odhadovať, či človek daného pohlavia s určitým vekom a hmotnosťou môže v budúcnosti trpieť na infarkt. Výsledok modelu by bola hodnota medzi 0 a 1, ktorá by udávala pravdepodobnosť. K tomu, aby logistická regresia rozložila pravdepodobnosť medzi hodnoty 0 a 1 využíva sigmoidnú funkciu (sigmoid) podľa vzorca (1).

$$\sigma = \frac{1}{1 + e^{-y}} \qquad (1)$$

Na obrázku 81 potom môžeme vidieť, že takáto krivka nebude priamkou, ako pri lineárnej regresií ale bude mať tvar písmena S. V logistickej regresii používame koncept prahovej hodnoty, ktorá definuje pravdepodobnosť buď 0 alebo 1. Napríklad hodnoty nad prahovou hodnotou majú tendenciu k 1 a hodnota pod prahovými hodnotami má tendenciu k 0.



Obrázok 86 Logistická regresia

Rovnicu logistickej regresie možno najjednoduchšie pochopiť z rovnice lineárnej regresie. Vieme, že rovnicu priamky možno vypočítať pomocou vzorca (2).

$$y = b_0 + b_1 x_1 + \ldots + b_n x_n \tag{2}$$

Pri logistickej regresií sa musí *y* rovnať hodnote medzi 0 a 1. K tomu sa dopracujeme tak, že upravíme ľavú stranu rovnice podľa vzorca (3).

$$\sigma = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + \ldots + b_n x_n)}} \tag{3}$$

8.2.2 Naivný Bayesov klasifikátor

Klasifikátor sa nazýva naivný, pretože predpokladá, že každá vstupná premenná je nezávislá. Naivný Bayesov klasifikátor sa používa na binomické aj multinomiálne dáta. Je založený na Bayesovej vete, ktorá je známa aj ako Bayesovo pravidlo alebo Bayesov zákon. Myšlienka Bayesovej vety spočíva v určení pravdepodobnosti výskytu výsledku. Túto pravdepodobnosť nazývame podmienená pravdepodobnosť a odvíja sa od predošlých výsledkov. V praxi by sa dala Bayesová veta použiť napríklad pri určovaní Alzheimeru na základe veku. Ak Alzheimer zodpovedá veku človeka, potom by sme vedeli presnejšie určiť, pravdepodobnosť Alzheimeru. Bayesovú vetu by sme vedeli zapísať podľa vzorca (4), kde *A*, *B* predstavujú javy, *P*(*A*), *P*(*B*) ich pravdepodobnosti výskytu a *P*(*A*|*B*) je podmienená pravdepodobnosť javu *A* za predpokladu, že nastal jav *B* a *P*(*B*|*A*) je podmienená pravdepodobnosť javu *B* za predpokladu, že nastal jav *A* (Murty & Devi, 2011).

$$P(A|B) = P(B|A)P(A)/P(B) \tag{4}$$

Podľa rozdelenia dát existujú tri typy Naivných Bayesových modelov:

- **Gausiálny** – Dáta sú spojité a majú normálne (Gaussovo) rozdelenie.
- **Multinomiálny** – Pri mulitunomiálnom modeli sú premenné reprezentované frekvenciami výskytov. V prípade klasifikácie článkov do kategórií by mohlo ísť o to, s akou frekvenciou sa slová vyskytujú v jednotlivých kategóriach.

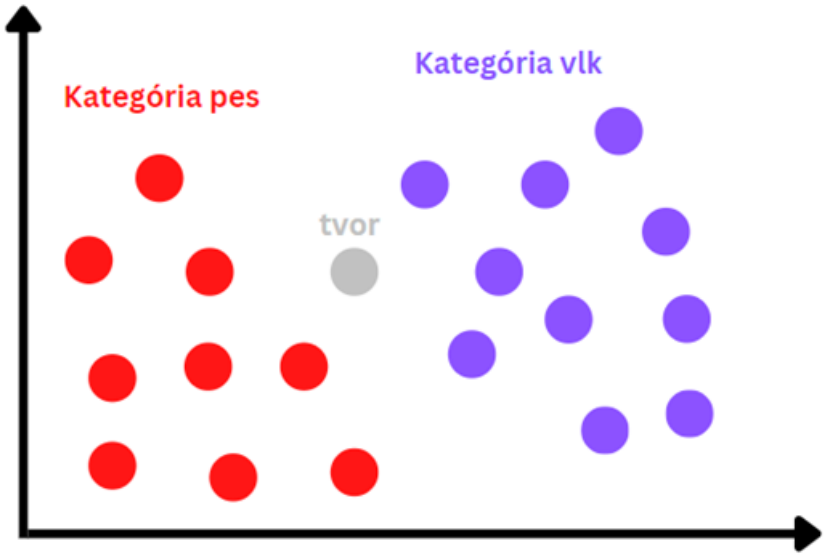
- **Bernoulliho** – Bernoulliho klasifikátor funguje podobne, ako multinomický klasifikátor, ale premenné sú booleovské. Napríklad, či konkrétne slovo je alebo nie je prítomné v dokumente. Tento model je známy aj úlohami klasifikácie dokumentov.

Bayesov klasifikátor so sebou prináša aj určité problémy, ako napríklad, že všetky údaje musia byť diskrétne. Ďalším problémom algoritmu je, že malá testovacia množina môže skresľovať výpočet relatívnych pravdepodobnosti. Ak sa určitá hodnota v testovacej množine vôbec nenachádza, pravdepodobnosť jej výskytu mala hodnotu 0.

8.2.3 Klasifikátor K-najbližších susedov (KNN)

Algoritmus *K* - najbližších susedov patrí medzi algoritmy lenivého učenia, pretože sa neučí z trénovacej množiny okamžite, namiesto toho ukladá množinu údajov a v čase klasifikácie vykonáva na množine údajov akciu (Russel, 2018).

Predpokladajme, že máme obrázok tvora, ktorý vyzerá podobne, ako pes a vlk, ale chceme vedieť, či je to pes alebo vlk (Obrázok 87).



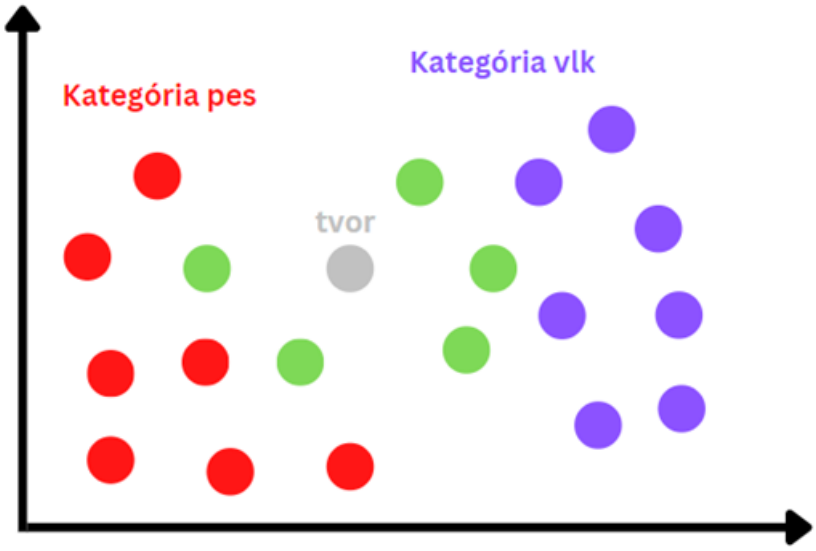
Obrázok 87 Nezaradený prípad

Prvým krokom algoritmu je nájdenie vhodného čísla *K*, ktoré bude označovať počet najbližších susedov (Obrázok 88). Výber správneho *K* je dôležitou úlohou. Veľmi nízke hodnoty *K* by mohli viesť k nestabilným hraniciam rozhodovania sa a vysoké hodnoty *K* by

mohli byť výpočtovo náročné. Po výbere *K* algoritmus nájde k nášmu tvorovi *K*-najbližších susedov podľa Euklidovskej vzdialenosti (5).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5}$$

Následne algoritmus spočíta, koľko z najbližších susedov patrí do kategórie pes a koľko z nich patrí do kategórie vlk. Tvorovi priradí kategóriu, ktorá sa v *K*-susedoch vyskytovala najčastejšie. Uvažujme, že naše *K*=5. Tvorovi bude priradená kategória vlk.



Obrázok 88 K-najbližších susedov

8.2.4 Podporný vektorový stroj (SVM)

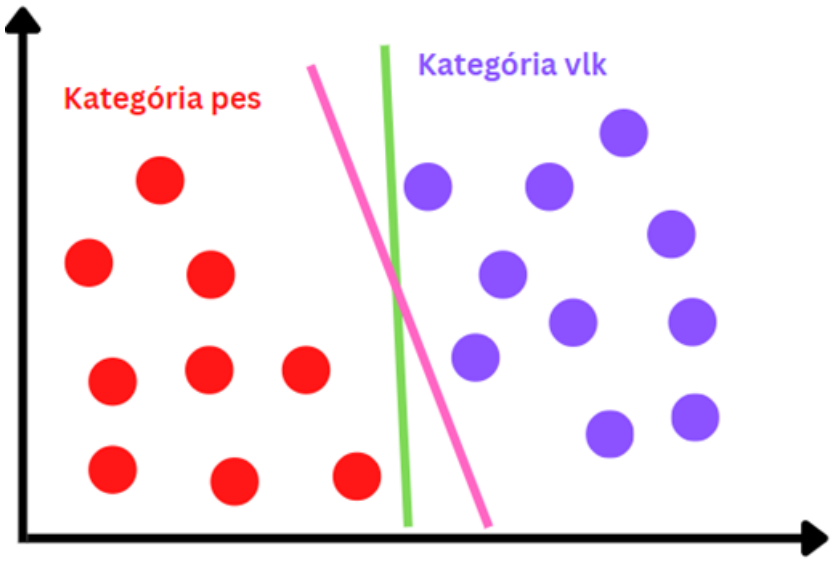
Cieľom podporných vektorových strojov je nájsť nadrovinu, teda rozhodovaciu čiaru alebo hranicu rozhodovania, ktorá pomôže klasifikovať dátové body do tried. Rozmery tejto nadroviny závisia od prítomných prvkov. Dátové body, ktoré nadrovinu podporujú (sú k nadrovine najbližšie) a ovplyvňujú jej polohu sa nazývajú podporné vektory (Russel, 2018).

Poznáme dva typy podporných vektorových strojov:

- **Lineárne** – ak je možné, dátový súbor rozdeliť priamkou.
- **Nelineárne** – ak dátový súbor nie je možné rozdeliť priamkou.

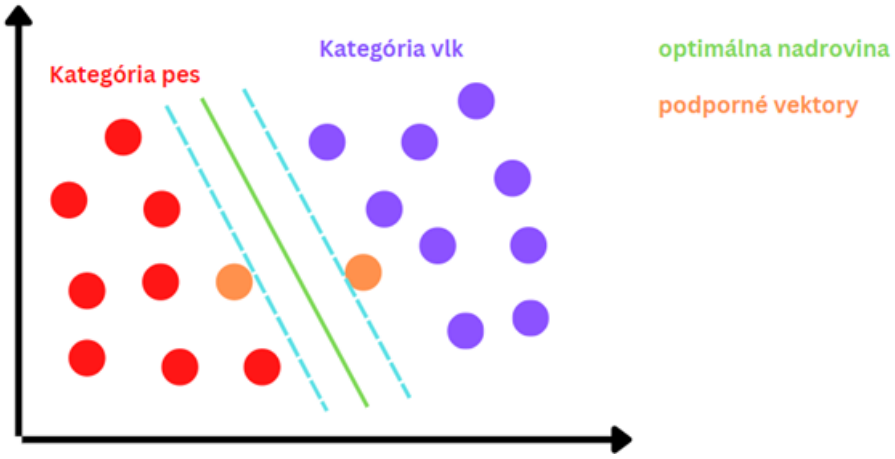
Lineárne SVM

Uvažujme predošlý príklad, pomocou ktorého sme si vysvetľovali KNN. Existuje mnoho možných spôsobov, ktorými by bolo možné tieto triedy odlíšiť priamkou (Obrázok 89).



Obrázok 89 Oddelenie priamkou

Algoritmus SVM bude hľadať takú priamku, ktorej vzdialenosť od najbližších bodov z oboch tried je maximálna. Takúto priamku (nadrovinu) bude pokladať za optimálnu (Obrázok 90).

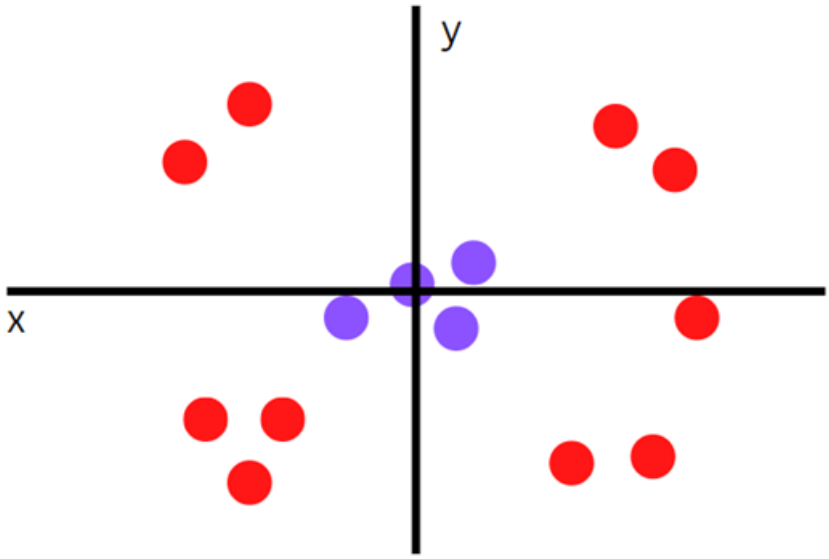


Obrázok 90 Optimálna nadrovina

V prípade, že by sme potrebovali klasifikovať doposiaľ neznámy údajový bod, zaradili by sme ho podľa toho, na ktorej strane nadroviny sa nachádza.

Nelineárne SVM

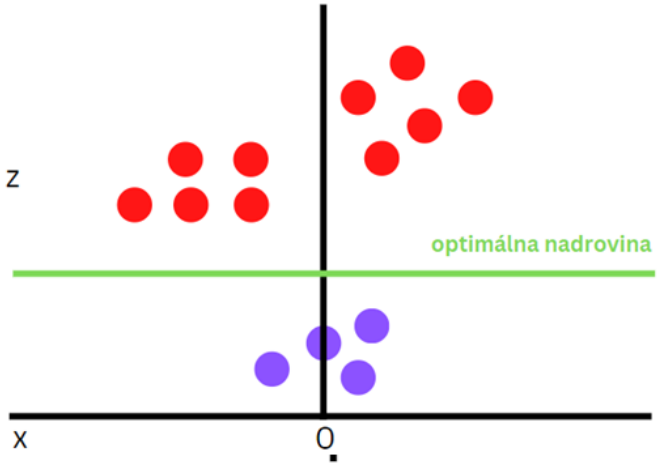
Existujú prípady, keď dáta nemožno rozdeliť pomocou priamky (Obrázok 91).



Obrázok 91 Nelineárne dáta

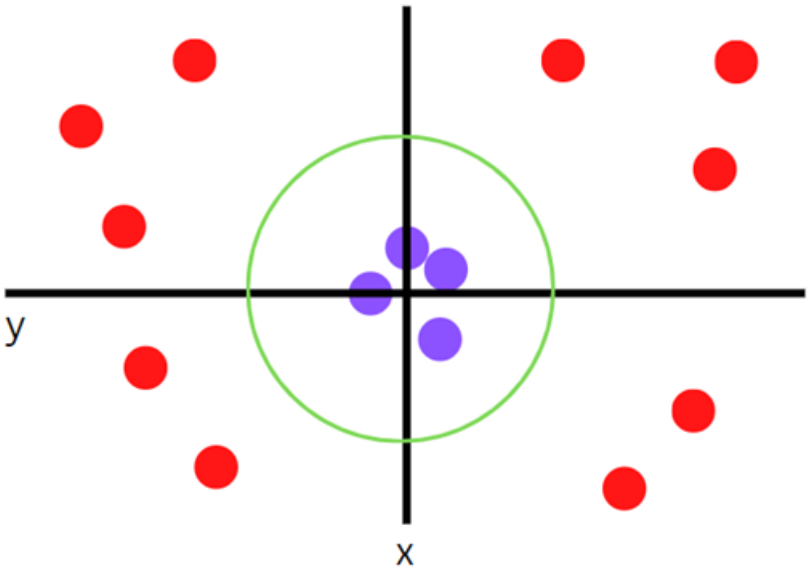
V takomto prípade môžeme pridať tretí rozmer *z* (Obrázok 7) a môžeme ho vypočítať podľa vzorca (6).

$$z = x^2 + y^2 \tag{6}$$



Obrázok 92 Pridanie tretieho rozmeru

Všimnite si, že keďže sme teraz v troch dimenziách, nadrovina je rovina rovnobežná s osou x v určitom z (povedzme $z = 1$). Zostáva mapovanie späť do dvoch dimenzií (Obrázok 8).



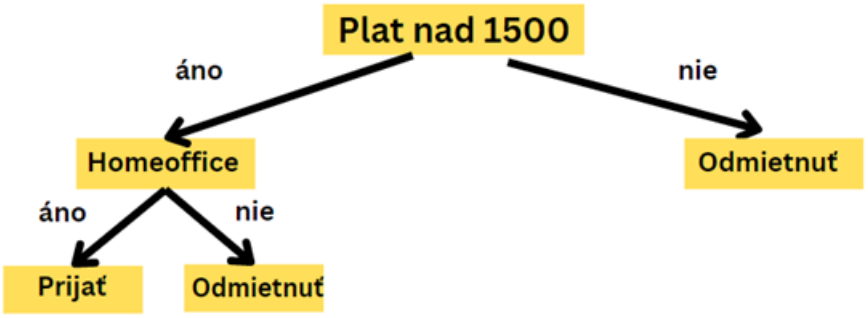
Obrázok 93 Mapovanie naspäť do dvoch dimenzií

8.2.5 Rozhodovací strom

Rozhodovací strom je klasifikátor so stromovou štruktúrou. Pri rozhodovacom strome zavádzame pojmy:

- **Koreň** – miesto, kde sa strom začína. Z tohto miesta (vrcholu / uzlu) sa ďalej strom vetví na dve alebo viac častí.
- **List** - konečný výstup, ktorý sa ďalej nevetví.
- **Vetva** – podstrom, ktorý sa vytvoril vetvením.

Rozhodovací strom je veľmi jednoducho pochopiteľný, pretože sa rozhoduje podobne ako človek. Na vrchole stromu sa nachádza otázka alebo hlavné kritérium, z ktorého sa strom vetví. Povedzme, že si hľadáme prácu s platom nad 1500€ a s možnosťou homeoffice (Obrázok 9).



Obrázok 94 Rozhodovací strom

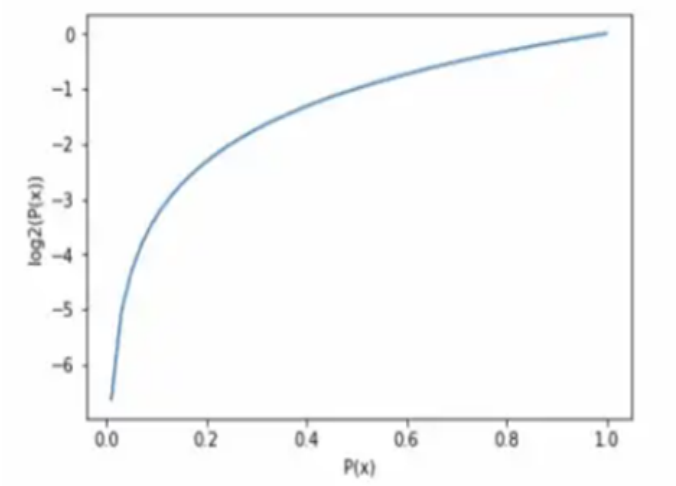
Meranie kvality rozdelenia stromu

K meraniu kvality rozdelenia sa zvyčajne používajú dve metriky, a to Gini index a entropia.

Gini index (7) je metrika pre klasifikačné úlohy, ktorej hodnota sa pohybuje v hodnotách od 0 do 1, kde 0 značí, že všetky prvky sú pridružené k určitej triede alebo existuje iba jedna trieda. V prípade, že Giniho index má hodnotu 1, prvky sú náhodne rozdelené do rôznych tried. Ak Gini index dosahuje hodnotu 0,5, znamená to, že prvky sú rovnomerne rozdelené do niektorých tried. Atribút s nízkym Gini indexom by mal byť preferovaný v porovnaní s vysokým Gini indexom. Gini index sa počíta podľa vzorca nižšie, kde pi je pravdepodobnosť, že objekt bude zaradený do určitej triedy.

$$Gini = 1 - \sum_{i=1}^{n} (pi)^2 \qquad (7)$$

Predstavme si, že si vyberieme bielu šachovú figúrku z krabice, v ktorej sa nachádza 100 bielych figúrok. Potom môžeme povedať, že krabica má nulovú entropiu. Teraz si predstavme, že 50 z nich je nahradených čiernou. Pravdepodobnosť vytiahnutia bielej klesne z 1,0 na 0,5 a entropia sa zvýšila. Shannonov entropický model používa logaritmickú funkciu so základom 2 *(*log2(*P*(*x*)) na meranie entropie, pretože ako sa zvyšuje pravdepodobnosť *P*(*x*) náhodného vytiahnutia bielej figúrky, výsledok sa približuje bližšie k hodnote 1 logaritmu pri základe 2, ako je znázornené na obrázku 10.



Obrázok 95 Shannonov entropický model

Entropia je metrika na meranie neurčitosti, s ktorou je prípad klasifikovaný do triedy a úlohou algoritmu je túto neurčitosť minimalizovať. Podobne ako pri Giniho indexe sa optimálne rozdelenie volí vlastnosťou s menšou entropiou. Entropiu možno vypočítať podľa vzorca (8), kde $P(x=k)$ vyjadruje pravdepodobnosť, že cieľový znak bude mať hodnotu k.

$$E = -\sum_j (P(x = k) * log_2(P(x = k)) \quad (8)$$

Logaritmus zlomkov dáva zápornú hodnotu, a preto sa vo vzorci entropie používa znamienko mínus na negáciu týchto záporných hodnôt. Maximálna hodnota entropie závisí od počtu tried. Aby sme našli najlepšiu vlastnosť, ktorá slúži ako koreňový uzol z hľadiska získavania informácií, najprv použijeme každú popisnú funkciu a rozdelíme množinu údajov podľa hodnôt týchto opisných vlastností a potom vypočítame entropiu množiny údajov. To nám dáva zostávajúcu entropiu, keď rozdelíme množinu údajov podľa hodnôt funkcií. Potom túto hodnotu odpočítame od pôvodne vypočítanej entropie množiny údajov, aby sme zistili, do akej miery toto rozdelenie prvkov znižuje pôvodnú entropiu, ktorá poskytuje informačný zisk prvku a vypočíta sa podľa vzorca (9), podľa ktorého funkcia s najvyšším informačným ziskom bude použitá ako koreňový uzol.

$$Informačný\ zisk(x) = E(dataset) - E(x) \quad (9)$$

8.2.5.1 Algoritmy pre vytvorenie stromu

ID3 (Iterative DiChaudomiser 3) algoritmus vytvára z trénovacej množiny údajov stromovú štruktúru, ktorá slúži k tomu, aby klasifikovala ešte nezatriedené dáta. Snaží sa o to, aby našiel kategorickú vlastnosť, ktorá prinesie najväčší informačný zisk pre kategorické ciele. Informačný zisk je vypočítaný pomocou entropie. Algoritmus prehľadáva jednotlivé vetvy a zastaví sa, keď každý podpriestor obsahuje iba prvky jednej triedy. Stromy vytvorené týmto algoritmom sú náchylné na pretrénovanie (Pandey & Mohurle, 2022).

Lepšou verziou je algoritmus C4.5, ktorý odstránil problém pri klasifikácii datasetov, ktoré obsahujú atribúty s veľkým množstvom hodnôt. Entropia atribútov, ktoré nadobúdajú veľké množstvo hodnôt je veľmi nízka. Preto sa zaviedol tzv. normalizovaný informačný zisk. Ďalším rozdielom, oproti predošlému algoritmu je, že dokáže pracovať aj s takými atribútmi, ktoré majú prázdne hodnoty. Po vytvorení modelu ešte raz preskúma strom a odstraňuje uzly, ktoré nemajú významný vplyv na klasifikáciu (Pandey & Mohurle, 2022).

Ďalšia úprava algoritmu získala názov C5.0. Tento algoritmus bol rýchlejší, efektívnejší a bolo možné paralelizovať algoritmus použitím vláken.

CART je rozhodovací strom, kde je každá vetva rozdelená na prediktorovú premennú a každý uzol má na konci predikciu pre cieľovú premennú. V rozhodovacom strome sú uzly rozdelené na poduzly na základe prahovej hodnoty atribútu. Koreňový uzol sa berie ako trénovacia množina a je rozdelený na dve časti zvážením najlepšieho atribútu a prahovej hodnoty. Ďalej sú podmnožiny tiež rozdelené pomocou rovnakej logiky. Toto pokračuje, kým sa v strome nenájde posledná čistá podmnožina alebo maximálny možný počet listov v tomto rastúcom strome. Algoritmus funguje podľa nasledovného postupu:

- získa sa najlepší bod rozdelenia každého vstupu,
- na základe týchto bodov rozdelenia sa identifikuje nový najlepší bod rozdelenia,rozdelí zvolený vstup podľa najlepšieho deliaceho bodu, pokračuje sa v delení, kým nie je splnené pravidlo zastavenia alebo nie je k dispozícii žiadne ďalšie požadované delenie.

Príliš veľký strom zvyšuje riziko preučenia sa a malý strom nemusí zachytiť všetky dôležité vlastnosti súboru. Preto sa využíva technika s názvom prerezávanie, ktorá zmenšuje strom bez zníženia presnosti.

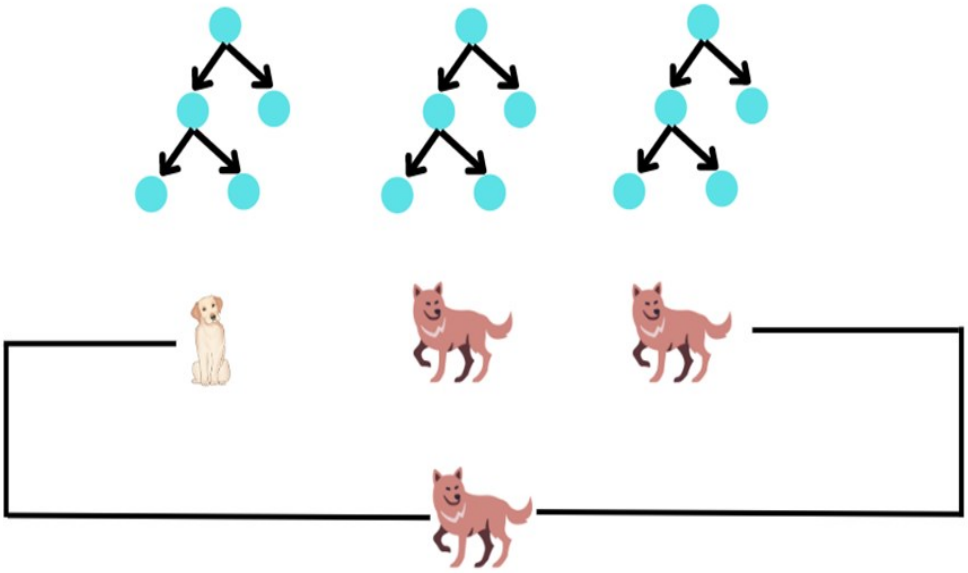
Existujú dva spôsoby, ktorými možno znížiť veľkosť stromu. Prvým spôsobom, je **predrezanie**, ktoré spočíva v ladení hyperparametrov pred trénovaním. Zahŕňa heuristiku známu ako „skoré zastavenie“, ktorá zastaví rast rozhodovacieho stromu – bráni mu dosiahnuť jeho plnú hĺbku. Zastavuje proces budovania stromov, aby sa zabránilo vytváraniu listov s malými vzorkami. Počas každej fázy rozdelenia stromu sa bude monitorovať chyba krížovej validácie. Ak už hodnota chyby neklesá - zastavíme rast rozhodovacieho stromu. Hyperparametre, ktoré je možné vyladiť na skoré zastavenie a zabránenie preťaženiu, sú maximálna hĺbka, minimálny počet vzoriek potrebných na rozdelenie vnútorného uzla a minimálny počet vzoriek požadovaných na to, aby sa nachádzali v liste.

Druhým spôsobom je **post-prerezávanie**, ktoré nezabraňuje stromu v raste ale prerezáva strom po tom, strom vyrastie do plnej hĺbky. Pre každý nelistový uzol v strome algoritmus vypočítava očakávanú chybovosť, ktorá sa môže objaviť, ak sa podstrom v tomto uzle skráti. Ďalej sa vypočíta očakávaná chybovosť, ktorá sa objaví, ak uzol nebol orezaný, pomocou chybovosti pre každú vetvu, spojenej vážením podľa rozmeru pozorovaní pozdĺž každej vetvy. Ak orezanie uzla vedie k vyššej očakávanej chybovosti, potom sa podstrom zachová.

8.2.6 Náhodný les

Klasifikátor s názvom náhodný les obsahuje množstvo rozhodovacích stromov pre rôzne podmnožiny daného súboru údajov a berie priemer na zlepšenie prediktívnej presnosti tohto súboru údajov. Je presnejší, pretože namiesto spoliehania sa na výsledok jedného stromu náhodný les berie predpoveď z každého stromu a na základe toho, aký výsledok dala väčšina z nich predpovedá konečný výstup. Náhodný les nepotrebuje veľa času na trénovanie, a dokáže klasifikovať s vysokou presnosťou aj v prípade, že časť údajov chýba.

Uvažujme náš prípad s klasifikáciou tvora do triedy pes alebo vlk. Predstavme si, že by sme vytvorili náhodný les s troma rozhodovacími stromami. Dva zo stromov sa zhodli, že na obrázku sa nachádza vlk, výslednou predpoveďou bude teda vlk (Obrázok 11).



Obrázok 96 Náhodný les

8.3 Súborové učenie

Doposiaľ sme si ukazovali, ako trénovať jeden konkrétny model. V praxi sa viacej využíva koncept súborového učenia (Ensemble Learning), ktorého myšlienkou je trénovať viacero modelov s využitím rovnakého algoritmu. Kombinácia niekoľkých modelov umožňuje dosiahnuť lepšiu klasifikáciu, oproti použitiu jedného klasifikátora (Russel, 2018).

Hlavné príčiny chýb pri učení klasifikátorov sú šum, rozptyl alebo skreslenie. Súborové učenie pomáha eliminovať tieto chyby obzvlášť pri nestabilných klasifikátoroch. Existujú tri hlavné metódy súborového učenia, pričom každá z nich sa používa podľa toho, ako chceme klasifikáciu vylepšiť. Pre zníženie rozptylu je vhodné použiť bagging metódu, pre zníženie kreslenia boosting metódu a pre zlepšenie presnosti klasifikácie stacking metódu. Súborové metódy môžeme v princípe rozdeliť do dvoch hlavných skupín:

- **Sekvenčné súborové metódy** – žiaci sú vytváraní sekvenčne a cieľom týchto metód je využiť závislosť medzi žiakmi. Výkon metód sa obvvykle zvyšuje tak, že sa pridáva váha nesprávne klasifikovaným prípadom. Typickým zástupcom je Adaboost.
- **Paralelné súborové metódy** – žiaci sú vytváraní paralelne, príkladom takéhoto algoritmu je aj samotný random forest, ktorý paralelne vytvára stromy. Oproti sekvenčným súborovým metódam, v tomto prípade je cieľom využiť nezávislosť

8.3.1 Bagging

Pojem Bagging vznikol zo slov Bootstrap Aggregating. Bootstrap je založený na náhodnom vzorkovaní malých častí datasetu, pričom tieto množiny môžu byť nahradené. Takéto náhodné vzorkovanie môže pomôcť k lepšej interpretácií štandardnej odchylky v datasete.

Bagging je jednoduchá metóda, pomocou ktorej možno trénovať množstvo rôznych modelov na rôznych náhodne vybraných podskupinách z trénovacej množiny a potom skombinovať ich predpovede pomocou hlasovania. Údaje sa teda najskôr rozdelia do niekoľkých tréningových a testovacích sád, na ktorých sa následne trénujú modely a predpoveď väčšiny je výsledkom. Výhody vrecovania:

- zvyšuje skóre presnosti modelu,
- dokáže si poradiť s overfittingom,
- znižuje chyby skreslenia a rozptylu,
- jednoduchá implementácia.

8.3.2 Boosting

Aby sme pochopili Boosting, je dôležité uvedomiť si, že Boosting je skôr generický algoritmus než špecifický model. Boosting je metóda učenia, ktorá spája skupinu slabých klasifikátorov do silného klasifikátora, aby sa minimalizovali chyby pri tréningu. Na rozdiel od baggingu, môžeme hovoriť o „tímovej práci“, nakoľko modely bežia paralelne. Pri boostingu sa vyberie náhodná vzorka údajov, ktorá sa osadí modelom a potom sa postupne trénuje – to znamená, že každý model sa snaží kompenzovať slabé stránky svojho predchodcu. Pri každej iterácii sa slabé pravidlá z každého jednotlivého klasifikátora kombinujú do jedného, silného predikčného pravidla.

Adaboost (Adaptive Boosting)

Adaboost je technika založená na boostingu, ktorá je založená na kombinovaní viacerých slabých klasifikátorov do jedného silného klasifikátora. Slabým klasifikátorom v Adaboost môže byť rozhodovací strom s jedným rozdelením. V momente vytvorenia prvého rozdelenia získajú všetky pozorovania rovnakú váhu. Pre korekciu chyby teda nesprávne klasifikované prípady získajú vyššiu váhu.

Gradient Boosting

Podobne ako Adaboost, aj Gradient Boosting sa snaží o to zlepšiť svojho predchodcu ale používa k tomu mierne odlišný spôsob. Nesnaží sa meniť váhy pre nesprávne klasifikované prípady ale pokúša sa opraviť svojho prechodcu za účelom zníženia chybovosti. Modifikáciou Gradient Boostingu je tzv. XGBoost, ktorý pozostáva zo zosilnených rozhodovacích stromov pre vyšší výkon.

8.3.3 Stacking

Stacking je ďalšou metódou súborového učenia. Je založený na skladaní klasifikačných modelov a pozostáva z dvojvrstvových odhadov. Prvá vrstva je zložená zo všetkých základných modelov, ktoré sa používajú na predpovedanie výstupov na testovacích súboroch údajov. Druhá vrstva pozostáva z meta klasifikátora, ktorý berie všetky predpovede základných modelov ako vstup a generuje nové predpovede. Výhodou stackingu je, že dokáže využiť možnosti radu dobre fungujúcich modelov na klasifikačnú alebo regresnú úlohu a vytvárať predpovede, ktoré majú lepší výkon ako ktorýkoľvek jednotlivý model v súbore.

8.4 Evalvácia klasifikačných modelov

Evalvácia znamená, že sa pokúšame vyhodnotiť, ako dobre náš model klasifikuje. K vyhodnoteniu úspešnosti vytvoreného modelu existuje niekoľko metrík.

8.4.1 Accuracy

Accuracy jednoducho meria, ako často klasifikátor správne predpovedá. Accuracy môžeme definovať ako pomer počtu správnych predpovedí a celkového počtu predpovedí. Accuracy možno vypočítať podľa vzorca (10), kde *TP* predstavuje pravdivo pozitívne prípady, *TN* pravdivo negatívne, *FP* falošne pozitívne a *FN* falošne negatívne prípady.

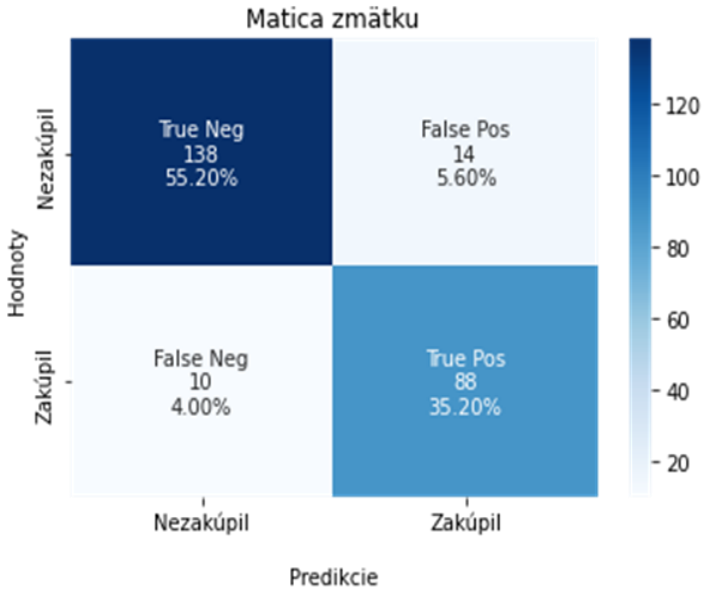
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

Keď accuracy dosahuje vysoké hodnoty, nemusí to nutne znamenať, že náš model klasifikuje správne. Predstavme si, že máme klasifikátor, ktorý určuje, či sa na obrázku nachádza pes, alebo vlk. Máme množinu testovacích obrázkov spolu s označeniami a vložíme do modelu prvý obrázok psa. Predpokladáme, že náš model predpovedá, že ide o psa, a potom predpoveď porovnáme so správnym označením. Ak model predpovedá, že ide o vlka a porovnáme ho so správnym označením, model sa pomýlil.

Tento proces opakujeme pre všetky obrázky v testovacej množine. Nakoniec budeme mať počty *TP, TN, FP, FN*. V skutočnosti je však veľmi zriedkavé, že všetky nesprávne alebo správne zhody budú vyvážené.

8.4.2 Matica zmätku

Matica zmätku je tabuľka, v ktorej sa nachádzajú hodnoty *TP*, *TN*, *FP* a *FN*. Čítame ju tak, že riadok predstavuje triedu a stĺpce predstavujú počet pravdivých a nepravdivých prípadov v triede (Obrázok 12).



Obrázok 97 Príklad matice zmätku

8.4.3 Presnosť (precision)

Táto metrika vysvetľuje, koľko zo správne predpovedaných prípadov sa skutočne ukázalo ako pozitívnych a je definovaná ako počet skutočných pozitívnych výsledkov vydelený počtom predpokladaných pozitívnych výsledkov (11).

$$precision = \frac{TP}{TP + FP} \tag{11}$$

8.4.4 Pokrytie (recall)

Recall vysvetľuje, koľko skutočných pozitívnych prípadov sme boli schopní správne predpovedať pomocou nášho modelu a je definovaná ako počet skutočných pozitívnych výsledkov vydelený celkovým počtom skutočných pozitívnych výsledkov (12).

$$recall = \frac{TP}{TP + FN} \tag{12}$$

8.4.5 Harmonický priemer (F1 score)

Harmonický priemer poskytuje kombinovanú predstavu o metrikách precision a recall. Maximálny je, keď sa presnosť rovná Recall. Jeho výhodou je, že trestá extrémne hodnoty (13).

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{13}$$

8.4.6 AUC-ROC

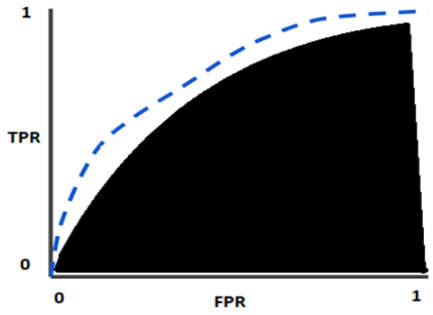
Krivka ROC (krivka prevádzkovej charakteristiky prijímača) znázorňuje výkonnosť modelu pri rôznych prahoch. Obsahuje dva parametre a to skutočne pozitívna miera *TPR* a falošne pozitívna miera *FPR (14, 15)*.

$$TPR = \frac{TP}{TP + FN} \tag{14}$$

(15)

$$FPR = \frac{FP}{FP + TN}$$

Oblasť pod krivkou (AUC) je mierou schopnosti klasifikátora rozlišovať medzi triedami. Čím väčšia je AUC, tým lepšia je výkonnosť modelu a teda keď sa AUC rovná 1, klasifikátor je schopný dokonale rozlíšiť medzi všetkými pozitívnymi a negatívnymi bodmi triedy. Keď sa AUC rovná 0, klasifikátor by predpovedal všetky negatívne body ako pozitívne a naopak. Keď je AUC 0,5, klasifikátor nie je schopný rozlíšiť medzi pozitívnou a negatívnou triedou (Obrázok 13).



Obrázok 98 AUC ROC

8.4.7 Strata logaritmu (Log Loss, Cross Entropy Loss)

Strata logaritmu alebo krížovej entropie je jednou z hlavných metrík na posúdenie výkonnosti klasifikačného problému. Pre jednu vzorku so skutočným označením $y \in \{0,1\}$ a odhadom pravdepodobnosti $p = Pr(y = 1)$ je strata logaritmu počítaná podľa vzorca (16).

$$logloss_{(N=1)} = y \log(p) + (1 - y) \log(1 - p) \quad (16)$$

Strata krížovej entropie alebo logaritmická strata meria výkon klasifikačného modelu, ktorého výstupom je hodnota pravdepodobnosti medzi 0 a 1. Strata krížovej entropie sa zvyšuje, keď sa predpokladaná pravdepodobnosť líši od skutočného označenia. Dokonalý model by mal stratu logaritmu 0.

8.5 Implementácia v Pythone

V tejto časti si ukážeme, ako možno klasifikáciu jednoducho implementovať v jazyku Python. Postačí, ak si vytvoríme jeden python súbor alebo jupyter notebook. Skôr, než začneme písať kód si potrebujeme nainštalovať knižnice numpy pandas, matplotlib, seaborn a sklearn.

8.5.1 Dataset

Dátový súbor[1], s ktorým budeme pracovať je databáza potenciálnych zákazníkov firmy, ktorá sa zaoberá predajom krbov. Obsahuje tisíc záznamov a s využitím klasifikátorov sa pokúsime modelovať, koľko ľudí z nášho datasetu zakúpilo krb za posledný rok. V datasete sa nachádza vek zákazníka, mesačný plat a informácia o tom, či zákazník daný produkt zakúpil.

Importujeme si knižnicu numpy ,pre prácu s poliami, pandas, pre prácu s našim dátovým súborom. Knižnice seaborn a matplotlib budú neskôr slúžiť na vykreslenie dát a sklearn bude slúžiť k trénovaniu našich klasifikačných modelov (Obrázok 14).

```
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 import sklearn
```

[0]

Obrázok 99 Import knižníc

Do premennej dataset si pomocou knižnice pandas a metódy read_csv načítame náš dataset. Táto premenná bude typu Dataframe. Pomocou metódy print() vypíšeme prvých a posledných 5 záznamov. Z výpisu ďalej vidíme, že náš dataset má tisíc riadkov a tri stĺpce (Obrázok 15).

[1] Zdroj datasetu: https://github.com/livi83/customers-dataset

```
#načítanie datasetu, v našom prípade sú v .csv hodnoty oddelené bodkočiarkou
dataset = pd.read_csv("https://raw.githubusercontent.com/livi83/customers-
dataset/main/dataset.csv", sep=";")
print(dataset)
```

```
     Vek  Mesačný plat  Zakúpil
0     37           833        0
1     42          1813        0
2     51          3083        0
3     42          4479        1
4     27          3292        0
..   ...           ...      ...
995   40          2458        0
996   49           979        0
997   30          5771        1
998   50          5583        1
999   46          3063        0

[1000 rows x 3 columns]
```

Obrázok 100 Načítanie datasetu

Pre lepší prehľad o našom datasete (Obrázok 101) môžeme použiť metódu describe().

```
print(dataset.describe())
```

```
              Vek  Mesačný plat      Zakúpil
count  1000.000000     1000.0000  1000.000000
mean     42.106000     3028.8050     0.402000
std      10.707073     1437.0232     0.490547
min      20.000000      625.0000     0.000000
25%      34.000000     1932.7500     0.000000
50%      42.000000     3000.0000     0.000000
75%      50.000000     3750.0000     1.000000
max      65.000000     6354.0000     1.000000
```

Obrázok 101 Metóda describe()

Skontrolujeme, či nemáme v datasete prázdne hotnoty pomocou metódy isnull. Ich počty si sčítame metódou sum() (Obrázok 102).

```
print(dataset.isnull().sum())
```

```
Vek           0
Mesačný plat  0
Zakúpil       0
dtype: int64
```

Obrázok 102 Sčítanie prázdnych hodnôt

Pomocou knižnice matplotlib si môžeme vizualizovať, koľko zákazníkov produkt zakúpilo a koľko zákazníkov produkt nezakúpilo (Obrázok 103).

```
dataset.Zakúpil.value_counts().plot(kind='bar')
plt.xlabel('Zakúpil')
plt.ylabel('počet zákazníkov')
plt.title('Počet zákazníkov vzhľadom k tomu, či produkt zakúpili')
plt.show()
```

Obrázok 103 Vizualizácia cieľovej premennej

K jednotlivým riadkom a stĺpcom budeme pristupovať pomocou metódy iloc. Tá prijíma ako parametre zoznam riadkov a stĺpcov. Ak dáme na niektorú z pozícií dvojbodku, hovoríme, že chceme všetky hodnoty (v našom prípade všetky riadky). Zoznam *x* predstavuje naše nezávislé premenné a obsahuje všetky riadky nultého a prvého stĺpca (číslujeme od nuly) a *y* predstavuje našu závislú premennú, ktorá obsahuje všetky riadky posledného stĺpca (Obrázok 104).

```
x = dataset.iloc[:,[0,1]].values
y = dataset.iloc[:,2].values
print(x)
print(y)
```

Obrázok 104 Výpis hodnôt x, y

8.5.2 Rozdelenie datasetu na trénovaciu a testovaciu množinu

Dátový súbor máme načítaný, potrebujeme si ho rozdeliť na trénovaciu a testovaciu časť. K tomuto využijeme knižnicu sklearn a jej metódu train_test_split, pomocou ktorej rozdelíme dáta v pomere 75:25 (Obrázok 105).

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25, random_state=0)
```

Obrázok 105 Rozdelenie na trénovaciu a testovaciu množinu

V tomto kroku nám vznikli premenné x_train, x_test, y_train, y_test. Pokiaľ ide o *y_train, y_test*, hodnoty sú v rozsahu 0-1 a preto nie je potrebné ich upravovať ale *x_train, x_test* musia pred trénovaním prejsť štandadizáciou. K tomuto účelu môžeme využiť triedu StandardScaler z knižnice sklearn, ktorá z ich hodnôt odpočíta priemer a vydelí ich smerodajnou odchylkou, aby upravila údaje na jednotkový rozptyl (Obrázok 106).

```
#normalizacia
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

Obrázok 106 Úprava na jednotkový rozptyl

8.5.3 Import evalvačných metrík

Skôr, než začneme s implementáciou klasifikátorov si môžeme importovať evalvačné metriky. Importujeme si maticu zmätku, accuracy, precision, recall a harmonický priemer F1 (Obrázok 107).

```
# Evalvácia
from sklearn import metrics
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score
```

Obrázok 107 Import evalvačných metrík

Ďalej si vytvoríme funkciu con_matrix, ktorá prijíma ako parameter predpovedané hodnoty a vykreslí maticu zmätku (Obrázok 108).

```
def conf_matrix(y_pred):
  cm = confusion_matrix(y_test, y_pred)

  names = ['True Neg','False Pos','False Neg','True Pos']
  counts = ["{0:0.0f}".format(value) for value in
                  cm.flatten()]
  percentages = ["{0:.2%}".format(value) for value in
                      cm.flatten()/np.sum(cm)]


  labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
            zip(names,counts,percentages)]

  labels = np.asarray(labels).reshape(2,2)

  ax = sns.heatmap(cm, annot=labels, fmt='', cmap='Blues')

  ax.set_title('Matica zmätku');
  ax.set_xlabel('\nPredikcie')
  ax.set_ylabel('Hodnoty');
  ax.xaxis.set_ticklabels(['Nezakúpil','Zakúpil'])
  ax.yaxis.set_ticklabels(['Nezakúpil','Zakúpil'])

  plt.show()
```

Obrázok 108 Funkcia pre výpis matice zmätku

8.5.4 Implementácia Logistickej regresie

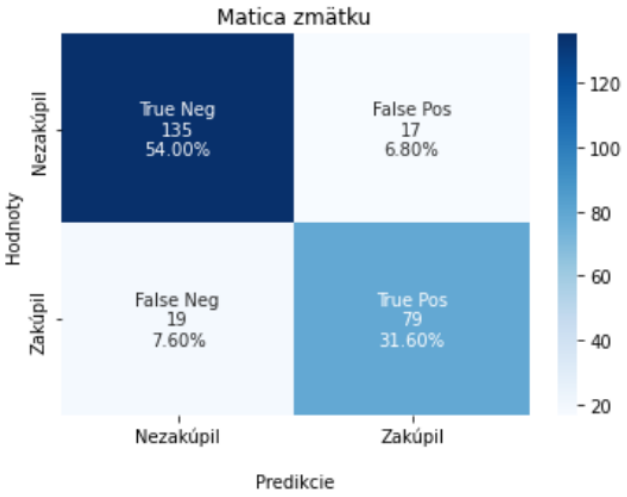
V základnej verzii možno logistickú regresiu implementovať zavolaním triedy LogisticRegression. Následne môžeme pomocou importovaných metrík zistiť výkonnosť nášho klasifikátora. Pomer počtu správnych predpovedí a celkového počtu predpovedí je 85,6%, miera správne identifikovaných pozitívnych prípadov zo všetkých predpovedaných pozitívnych prípadov je 82,2%, miera správne identifikovaných pozitívnych prípadov zo všetkých skutočných pozitívnych prípadov je 80,6% a harmonický priemer mier predstavuje 81,4% (Obrázok 109).

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression().fit(x_train,y_train)
y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')
conf_matrix(y_pred)
```

```
Accuracy: 0.856
Precision: 0.8229166666666666
Recall: 0.8061224489795918
F1-score: 0.8144329896907215
```

Obrázok 109 Evalvácia logistickej regresie

Zavolaním funkcie conf_matrix dokážeme vypísať maticu zmätku. Náš model klasifikoval správne 135 prípadov a nesprávne 17 pre triedu nezakúpil. Zvyšok testovacích dát patrí teda triede zakúpil, pričom model odhadol 79 z nich správne a 19 nesprávne (Obrázok 110).



Obrázok 110 Matica zmätku pre logistickú regresiu

8.5.5 Implementácia Naivného Bayesovho klasifikátora

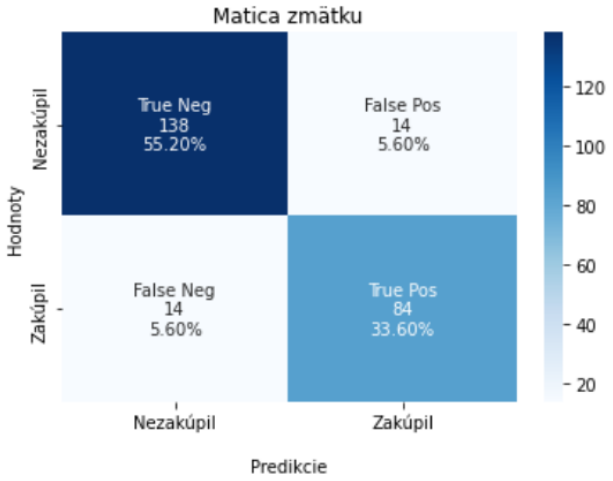
Pomer počtu správnych predpovedí a celkového počtu predpovedí je 88,8%, miera správne identifikovaných pozitívnych prípadov zo všetkých predpovedaných pozitívnych prípadov je 85,7%, miera správne identifikovaných pozitívnych prípadov zo všetkých skutočných pozitívnych prípadov je 85,7% a harmonický priemer mier predstavuje 85,7% (Obrázok 111).

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB().fit(x_train,y_train)
y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

```
Accuracy: 0.888
Precision: 0.8571428571428571
Recall: 0.8571428571428571
F1-score: 0.8571428571428571
```

Obrázok 111 Evalvácia Naivného Bayesa

Zavolaním funkcie conf_matrix dokážeme vypísať maticu zmätku. Náš model klasifikoval pre triedu nezakúpil správne 138 prípadov a nesprávne 14. Zvyšok testovacích dát patrí teda triede zakúpil, pričom model odhadol 84 z nich správne a 14 nesprávne (Obrázok 112).



Obrázok 112 Matica zmätku pre Naivného Bayesa

8.5.6 Implementácia klasifikátora K- najlbližších susedov

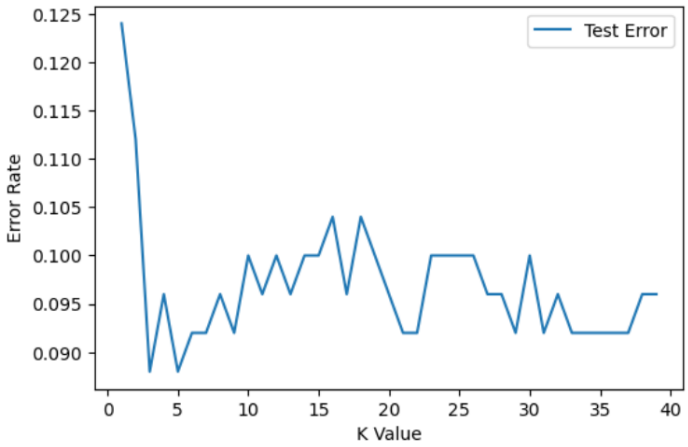
Pre implementáciu klasifikátora je dôležité nájsť optimálnu hodnotu *K*. Vyskúšajme hodnoty K od 1 do 40 a skúsme zistiť, pri ktorej hodnote dosiahne klasifikátor najvyššiu accuracy. Ako môžeme vidieť z grafu, optimálny počet najbližších susedov bude 2 (Obrázok 113, Obrázok 114).

```
from sklearn.neighbors import KNeighborsClassifier
test_error_rates = []

for k in range(1,40):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(x_train,y_train)

    y_pred = model.predict(x_test)

    test_error = 1 - accuracy_score(y_test,y_pred)
    test_error_rates.append(test_error)

min_value = min(test_error_rates)
print(test_error_rates.index(min_value))

plt.figure(figsize=(6,4),dpi=100)
plt.plot(range(1,40),test_error_rates,label='Test Error')
plt.legend()
plt.ylabel('Error Rate')
plt.xlabel("K Value")
```

[0]

```
2
```

Obrázok 113 Výpočet najlepšieho K



Obrázok 114 Vizualizácia najlepšieho K

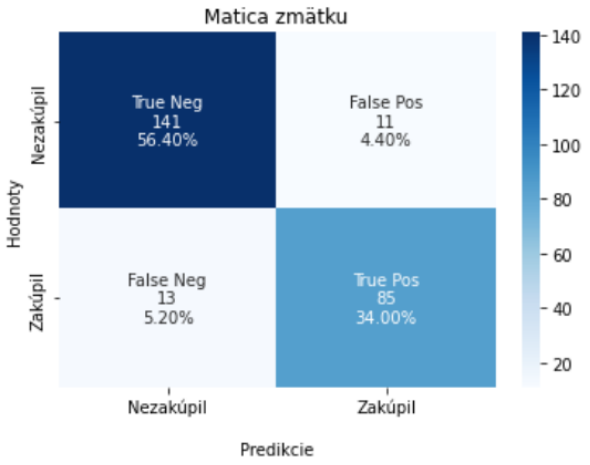
Pomer počtu správnych predpovedí a celkového počtu predpovedí je 90%, miera správne identifikovaných pozitívnych prípadov zo všetkých predpovedaných pozitívnych prípadov je 88,5%, miera správne identifikovaných pozitívnych prípadov zo všetkých skutočných pozitívnych prípadov je 86,7% a harmonický priemer mier predstavuje 87,6% (Obrázok 115).

```
model = KNeighborsClassifier(n_neighbors=4)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

```
Accuracy: 0.904
Precision: 0.8854166666666666
Recall: 0.8673469387755102
F1-score: 0.8762886597938144
```

Obrázok 115 Evalvácia K - najbližších susedov

Zavolaním funkcie conf_matrix dokážeme vypísať maticu zmätku. Náš model klasifikoval pre triedu nezakúpil správne 141 prípadov a nesprávne 11. Zvyšok testovacích dát patrí teda triede zakúpil, pričom model odhadol 85 z nich správne a 13 nesprávne (Obrázok 116).



Obrázok 116 Matica zmätku pre K-najbližších susedov

8.5.7 Implementácia SVM

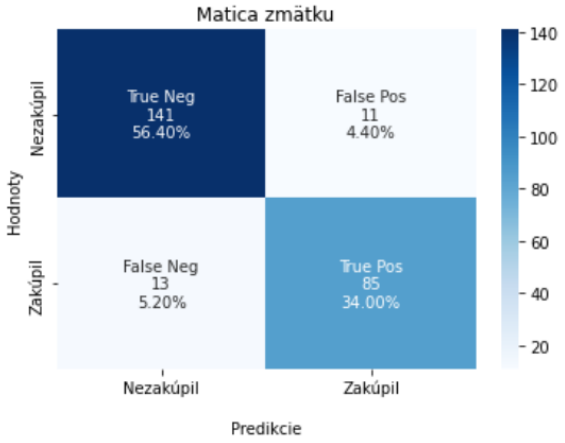
Naše dáta sú lineárne a teda využijeme implementáciu lineárneho SVM. Pomer počtu správnych predpovedí a celkového počtu predpovedí je 90,4%, miera správne identifikovaných pozitívnych prípadov zo všetkých predpovedaných pozitívnych prípadov je 88,5%, miera správne identifikovaných pozitívnych prípadov zo všetkých skutočných pozitívnych prípadov je 86,7% a ich harmonický priemer predstavuje 87,6% (Obrázok 117).

```
from sklearn.svm import SVC
model =SVC(kernel='linear').fit(x_train,y_train)
svm_y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

```
Accuracy: 0.904
Precision: 0.8854166666666666
Recall: 0.8673469387755102
F1-score: 0.8762886597938144
```

Obrázok 117 Evalvácia SVM

Zavolaním funkcie conf_matrix dokážeme vypísať maticu zmätku. Náš model klasifikoval pre triedu nezakúpil správne 141 prípadov a nesprávne 11. Zvyšok testovacích dát patrí teda triede zakúpil, pričom model odhadol 85 z nich správne a 13 nesprávne (Obrázok 118).



Obrázok 118 Matica zmätku pre SVM

8.5.8 Implementácia rozhodovacieho stromu

Pre implementáciu rozhodovacieho stromu pomocou knižnice sklearn si zavoláme triedu DecisionTreeClassifier. Prednastavenou metrikou pre meranie kvality rozdelenia je Gini index. V prípade, že by sme chceli zmeniť metriku na entropiu, postačilo by do klasifikátora vložiť parameter *criterion* s hodnotou entropy ().

Už pri rozdeľovaní datasetu na trénovaciu a testovaciu množinu sme použili parameter ***random_state***. Tento parameter riadi akým spôsobom, budú dáta rozdelené a môže nadobúdať hodnotu none, čím dostaneme náhodné rozdelenie dát pri každom spustení, doň môžeme vložiť celé číslo. Obvykle sa volí hodnota od 0 po 42 pre zaistenie náhodného ale vždy rovnakého rozdelenie dát. Inými slovami, ak dáme parametru random_state celočíselnú hodnotu, určujeme náhodnosť vo výbere. Po každom spustení bude však výber dát rovnaký. Rovnako ak chceme riadiť náhodnosť delenia pri rozhodovacích stromoch alebo náhodnách lesoch, parameter *random_state* je užitočný.

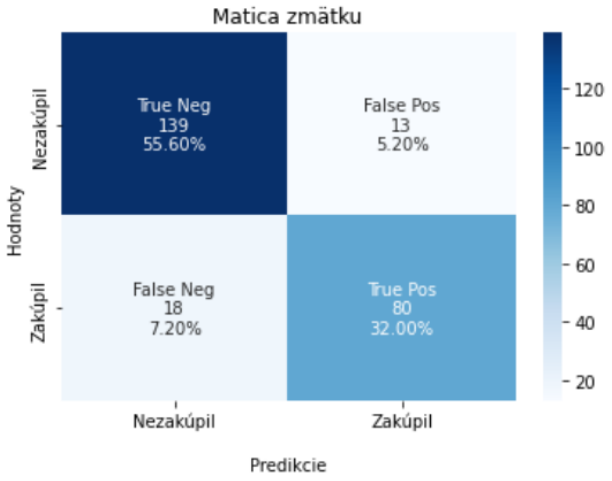
Pomer počtu správnych predpovedí a celkového počtu predpovedí je 87,6%, miera správne identifikovaných pozitívnych prípadov zo všetkých predpovedaných pozitívnych prípadov je 86%, miera správne identifikovaných pozitívnych prípadov zo všetkých skutočných pozitívnych prípadov je 81,6% a ich harmonický priemer predstavuje 83,7% (Obrázok 119).

```python
from sklearn.tree import DecisionTreeClassifier
model= DecisionTreeClassifier(random_state=0).fit(x_train, y_train)
y_pred = model.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,y_pred)}')
print('Precision: '+ f'{precision_score(y_test,y_pred)}')
print('Recall: '+ f'{recall_score(y_test,y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,y_pred)}')

conf_matrix(y_pred)
```

```
Accuracy: 0.876
Precision: 0.8602150537634409
Recall: 0.8163265306122449
F1-score: 0.837696335078534
```

Obrázok 119 Evalvácia rozhodovacieho stromu

Zavolaním funkcie conf_matrix dokážeme vypísať maticu zmätku. Náš model klasifikoval pre triedu nezakúpil správne 139 prípadov a nesprávne 13. Zvyšok testovacích dát patrí teda triede zakúpil, pričom model odhadol 80 z nich správne a 18 nesprávne (Obrázok 120).



Obrázok 120 Matica zmätku pre rozhodovací strom

8.5.9 Implementácia náhodného lesa

Posledný klasifikátor, ktorý si ukážeme je náhodný les. Možno ho implementovať pomocou triedy RandomForestClassifier, do ktorej pomocou parametra n_estimators definujeme počet stromov, ktorých hlasovanie bude ovplyvňovať zaradenie prvku do triedy. V našom prípade sme zvolili 10 stromov.

Pomer počtu správnych predpovedí a celkového počtu predpovedí je 90%, miera správne identifikovaných pozitívnych prípadov zo všetkých predpovedaných pozitívnych prípadov je 89,5%, miera správne identifikovaných pozitívnych prípadov zo všetkých

skutočných pozitívnych prípadov je 85,7% a harmonický priemer predstavuje 87,5% (Obrázok 121).

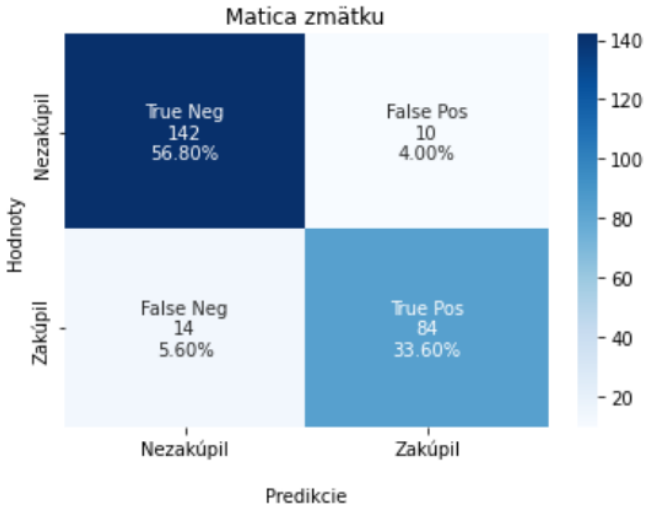
```
#random forest
from sklearn.ensemble import RandomForestClassifier
forest= RandomForestClassifier(n_estimators=10, random_state=0).fit(x_train, y_train)
forest_y_pred = forest.predict(x_test)

#Celkové vyhodnotenie modelu
print('Accuracy: '+ f'{accuracy_score(y_test,forest_y_pred)}')
print('Precision: '+ f'{precision_score(y_test,forest_y_pred)}')
print('Recall: '+ f'{recall_score(y_test,forest_y_pred)}')
print('F1-score: '+ f'{f1_score(y_test,forest_y_pred)}')

conf_matrix(forest_y_pred)
```

[0]

```
Accuracy: 0.904
Precision: 0.8936170212765957
Recall: 0.8571428571428571
F1-score: 0.875
```

Obrázok 121 Evalvácia náhodného lesa

Zavolaním funkcie conf_matrix dokážeme vypísať maticu zmätku. Náš model klasifikoval pre triedu nezakúpil správne 142 prípadov a nesprávne 10. Zvyšok testovacích dát patrí teda triede zakúpil, pričom model odhadol 84 z nich správne a 14 nesprávne (Obrázok 122).



Obrázok 122 Matica zmätku pre náhodný les

9 Dátové inžinierstvo

Projekty strojového učenia prediktívneho modelovania, ako je klasifikácia a regresia, vždy zahŕňajú určitú formu prípravy údajov. Špecifická príprava údajov potrebná pre súbor údajov závisí od špecifík údajov, ako sú typy premenných, ako aj od algoritmov, ktoré sa použijú na ich modelovanie, ktoré môžu klásť očakávania alebo požiadavky na údaje. Napriek tomu existuje súbor štandardných algoritmov na prípravu údajov, ktoré možno použiť na štruktúrované údaje (napr. údaje, ktoré tvoria veľkú tabuľku ako v tabuľkovom procesore). Tieto algoritmy prípravy údajov môžu byť organizované alebo zoskupené podľa typu do rámca, ktorý môže byť užitočný pri porovnávaní a výbere techník pre konkrétny projekt.

V tejto kapitole objavíte bežné úlohy prípravy údajov vykonávané v úlohe strojového učenia prediktívneho modelovania. A po jej dokončení budete vedieť:

- Techniky, ako je čistenie údajov, môžu identifikovať a opraviť chyby v údajoch, ako sú chýbajúce hodnoty.
- Transformácie údajov môžu zmeniť rozsah, typ a rozdelenie pravdepodobnosti premenných v súbore údajov.
- Techniky ako výber prvkov a redukcia rozmerov môžu znížiť počet vstupných premenných

9.1 Príprava dát

Prípravu dát môžeme definovať ako transformáciu nespracovaných dát do formy vhodnejšej na modelovanie. Napriek tomu existujú kroky v projekte prediktívneho modelovania pred a po kroku prípravy údajov, ktoré sú dôležité a informujú o príprave údajov, ktoré sa majú vykonať. Proces aplikovaného strojového učenia pozostáva zo sekvencie krokov. Môžeme skákať tam a späť medzi krokmi v skúmanom projekte, ale všetky projekty majú rovnaké všeobecné kroky, ktoré sú nasledovne (Crickard, 2020):

- **Krok 1** : Definujte problém.
- **Krok 2** : Pripravte dáta.
- **Krok 3** : Vyhodnoťte modely.
- **Krok 4** : Dokončite model.

Nás v tomto prípade teda zaujíma krok prípravy údajov (krok 2) a existujú bežné alebo štandardné úlohy, ktoré môžete použiť alebo preskúmať počas kroku prípravy údajov v projekte strojového učenia.

Existuje veľké množstvo rôznych typov techník prípravy údajov , ktoré by sa dali použiť v projekte prediktívneho modelovania. V niektorých prípadoch môže distribúcia údajov alebo požiadavky modelu strojového učenia naznačovať potrebnú prípravu údajov. Namiesto toho môže byť príprava údajov považovaná za ďalší hyperparameter, ktorý sa má vyladiť ako súčasť modelovacieho potrubia. To vyvoláva otázku, ako zistiť, aké metódy prípravy údajov je potrebné zvážiť pri vyhľadávaní, čo môže odborníkom aj začiatočníkom pripadať zdrvujúce. Riešením je premýšľať o rozsiahlom poli prípravy údajov štruktúrovaným spôsobom a systematicky vyhodnocovať techniky prípravy údajov na základe ich vplyvu na nespracované údaje.

V tejto kapitole objavíte rámec, ktorý poskytuje štruktúrovaný prístup k premýšľaniu a zoskupovaniu techník prípravy údajov pre prediktívne modelovanie so štruktúrovanými údajmi. Kapitola je rozdelená na tri časti a to: výzva na prípravu dát; rámec na prípravu údajov; techniky prípravy dát.

Začneme prvou častou a to výzvou na prípravu dát. Príprava údajov sa vzťahuje na transformáciu nespracovaných údajov do formy, ktorá je vhodnejšia na prediktívne modelovanie. Môže to byť potrebné, pretože samotné údaje obsahujú chyby. Môže to byť aj preto, že zvolené algoritmy majú očakávania týkajúce sa typu a distribúcie údajov.

Aby bola úloha prípravy údajov ešte náročnejšia, je tiež bežné, že príprava údajov potrebná na získanie najlepšieho výkonu z prediktívneho modelu nemusí byť zrejmá a môže ohnúť alebo porušiť očakávania používaného modelu. Ako také je bežné považovať výber a konfiguráciu prípravy údajov aplikovanú na nespracované údaje za ďalší hyperparameter modelovacieho potrubia, ktorý sa má vyladiť.

Toto rámcovanie prípravy údajov je v praxi veľmi efektívne, pretože vám umožňuje používať techniky automatického vyhľadávania, ako je vyhľadávanie v mriežke a náhodné vyhľadávanie, na objavenie neintuitívnych krokov prípravy údajov, ktorých výsledkom sú zručné prediktívne modely. Riešením tohto preťaženia je premýšľať o technikách prípravy údajov systematickým spôsobom.

Ďalšou zaujímavou pre nás častou bude tzv. rámec na prípravu údajov. Efektívna príprava údajov si vyžaduje, aby dostupné techniky prípravy údajov boli organizované a posudzované štruktúrovaným a systematickým spôsobom. To vám umožňuje zabezpečiť, aby sa pre vašu množinu údajov preskúmali techniky prístupu a že potenciálne efektívne techniky nebudú preskočené alebo ignorované. Dá sa to dosiahnuť pomocou rámca na organizáciu techník prípravy údajov, ktoré zohľadňujú ich vplyv na surový súbor údajov.

Napríklad štruktúrované údaje strojového učenia, ako sú údaje, ktoré môžeme uložiť do súboru CSV na klasifikáciu a regresiu, pozostávajú z riadkov, stĺpcov a hodnôt. Mohli by sme zvážiť techniky prípravy údajov, ktoré fungujú na každej z týchto úrovní.

a. Príprava údajov pre riadky.
b. Príprava dát pre stĺpce.
c. Príprava dát na hodnoty.

Príprava údajov pre riadky môžu byť techniky, ktoré pridávajú alebo odstraňujú riadky údajov z množiny údajov. Podobne príprava údajov pre stĺpce môžu byť techniky, ktoré pridávajú alebo odstraňujú stĺpce (funkcie alebo premenné) z množiny údajov. Zatiaľ čo príprava údajov na hodnoty môžu byť techniky, ktoré menia hodnoty v súbore údajov, často pre daný stĺpec.

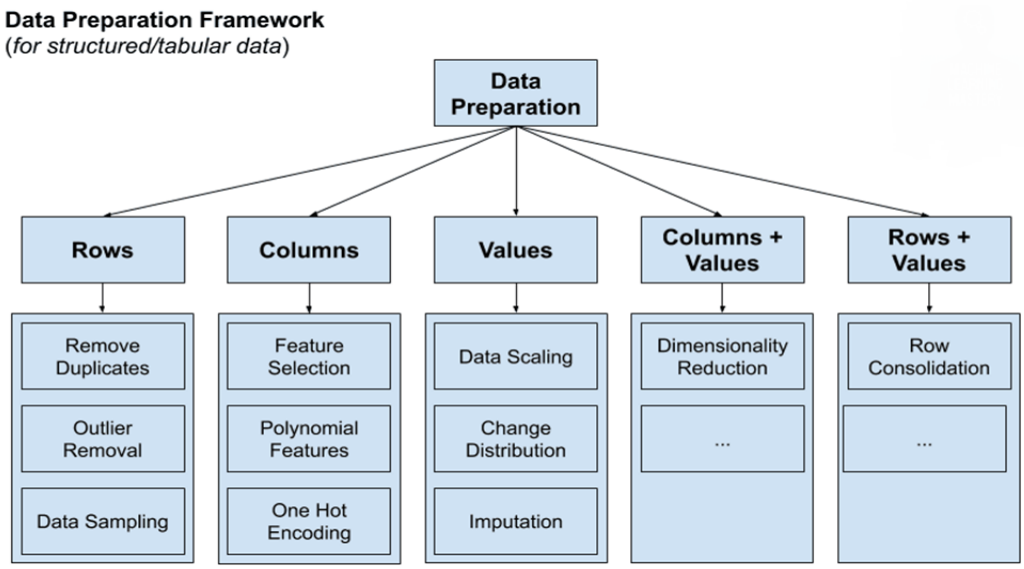
Existuje ešte jeden typ prípravy údajov, ktorý presne nezapadá do tejto štruktúry, a to sú techniky redukcie rozmerov. Tieto techniky menia stĺpce a hodnoty súčasne, napr. premietaním údajov do priestoru nižšej dimenzie.

Príprava údajov pre stĺpce + hodnoty.

To vyvoláva otázku techník, ktoré by sa mohli použiť na riadky a hodnoty súčasne. To môže zahŕňať prípravu údajov, ktorá nejakým spôsobom konsoliduje riadky údajov.

Príprava údajov pre riadky + hodnoty.

Tento rámec a niektoré skupiny metód prípravy údajov na vysokej úrovni môžeme zhrnúť na nasledujúcom obrázku.



Obrázok 123 Rámec prípravy údajov

Teraz, keď máme rámec na premýšľanie o príprave údajov na základe ich vplyvu na údaje, pozrime sa na príklady techník, ktoré zapadajú do každej skupiny.

No a teda tretou častou nasej kapitoly budú práve príklady technik prípravy dát. My skúsime preskúmať päť skupín techník prípravy údajov na vysokej úrovni definovaných v predchádzajúcej časti, ktoré navrhujú špecifické techniky, ktoré môžu patriť do každej skupiny.

Prvou nami skúmanou skupinou bude príprava dát pre riadky. Táto skupina je určená pre techniky prípravy údajov, ktoré pridávajú alebo odstraňujú riadky údajov.

V strojovom učení sa riadky často označujú ako vzorky, príklady alebo inštancie. Tieto techniky sa často používajú na rozšírenie obmedzeného súboru trénovacích údajov alebo na odstránenie chýb alebo nejednoznačnosti zo súboru údajov.

Hlavnou triedou techník, ktoré prichádzajú na myseľ, sú techniky prípravy údajov, ktoré sa často používajú na nevyváženú klasifikáciu. To zahŕňa techniky ako SMOTE, ktoré vytvárajú syntetické riadky tréningových údajov pre nedostatočne zastúpené triedy a náhodné podvzorkovanie, ktoré odstraňuje príklady pre nadmerne zastúpené triedy.

Zahŕňa tiež pokročilejšie kombinované techniky prevzorkovania a podvzorkovania, ktoré sa pokúšajú identifikovať a odstrániť nejednoznačné príklady pozdĺž rozhodovacej hranice problému klasifikácie a odstrániť ich alebo zmeniť ich označenie triedy.

Táto trieda techník prípravy údajov zahŕňa aj algoritmy na identifikáciu a odstránenie odľahlých hodnôt z údajov. Ide o riadky údajov, ktoré môžu byť ďaleko od centra hmotnosti pravdepodobnosti v súbore údajov a naopak nemusia reprezentovať údaje z domény.

Ako ďalšiu skúmanú skupinu vezmeme prípravu dát pre stĺpce. Táto skupina je určená pre techniky prípravy údajov, ktoré pridávajú alebo odstraňujú stĺpce údajov.

V strojovom učení sa stĺpce často označujú ako premenné alebo funkcie. Tieto techniky sú často potrebné na zníženie zložitosti (rozmernosti) problému predikcie alebo na rozbalenie zložených vstupných premenných alebo komplexných interakcií medzi funkciami.

Hlavnou triedou techník, ktoré prichádzajú na myseľ, sú techniky výberu funkcií. To zahŕňa techniky, ktoré používajú štatistiky na hodnotenie relevantnosti vstupných premenných pre cieľovú premennú na základe typu údajov každej z nich.

Zahŕňa to aj techniky výberu funkcií, ktoré systematicky testujú vplyv rôznych kombinácií vstupných premenných na predikčnú schopnosť modelu strojového učenia.

Súvisiace techniky, ktoré používajú model na hodnotenie dôležitosti vstupných funkcií na základe ich použitia prediktívnym modelom, označované ako metódy dôležitosti funkcie. Tieto metódy sa často používajú na interpretáciu údajov, hoci sa dajú použiť aj na výber prvkov.

Táto skupina metód tiež prináša do úvahy techniky na vytváranie alebo odvodzovanie nových stĺpcov údajov, nové funkcie. Tieto sa často označujú ako inžinierstvo funkcií, hoci niekedy sa celá oblasť prípravy údajov označuje ako inžinierstvo funkcií. Napríklad nové funkcie, ktoré predstavujú hodnoty povýšené na exponenty alebo multiplikatívne kombinácie funkcií, možno vytvoriť a pridať do množiny údajov ako nové stĺpce.

Môže to zahŕňať aj transformácie údajov, ktoré menia typ premennej, ako napríklad vytváranie fiktívnych premenných pre kategorickú premennú, ktoré sa často označuje ako jednorazové kódovanie.

Tretou skúmanou nami skupinou technik bude príprava dát na hodnoty. Táto skupina je určená pre techniky prípravy údajov, ktoré menia nespracované hodnoty v údajoch. Tieto techniky sú často potrebné na splnenie očakávaní alebo požiadaviek špecifických algoritmov strojového učenia.

Hlavnou triedou techník, ktoré prichádzajú na myseľ, sú transformácie údajov, ktoré menia rozsah alebo distribúciu vstupných premenných. Napríklad transformácie údajov, ako je štandardizácia a normalizácia , menia rozsah číselných vstupných premenných. Transformácie údajov, ako je ordinálne kódovanie, menia typ kategorických vstupných premenných.

Existuje tiež veľa transformácií údajov na zmenu distribúcie vstupných premenných. Napríklad diskretizácia alebo binning mení distribúciu numerických vstupných premenných na kategorické premenné s poradovým poradím.

Transformáciu výkonu možno použiť na zmenu rozdelenia údajov, aby sa odstránilo zošikmenie a aby sa rozdelenie stalo normálnejším (gaussovským).

Kvantilová transformácia je flexibilný typ techniky prípravy údajov, ktorá môže mapovať numerickú vstupnú premennú alebo na rôzne typy distribúcií, ako je normálne alebo Gaussovské.

Ďalším typom techniky prípravy údajov, ktorý patrí do tejto skupiny, sú metódy, ktoré systematicky menia hodnoty v súbore údajov. To zahŕňa techniky, ktoré identifikujú a nahradia chýbajúce hodnoty, často označované ako imputácia chýbajúcej hodnoty. Dá sa to dosiahnuť pomocou štatistických metód alebo pokročilejších metód založených na modeloch.

Všetky diskutované metódy by sa tiež mohli považovať za metódy inžinierstva funkcií (napr. zapadajúce do predtým diskutovanej skupiny metód prípravy údajov), ak sa výsledky transformácií pridajú k nespracovaným údajom ako nové stĺpce.

Predposlednou v zozname na obrázku skupinou techník je príprava údajov pre stĺpce a hodnoty. Táto skupina je určená pre techniky prípravy údajov, ktoré menia počet stĺpcov aj hodnoty v údajoch.

Hlavnou triedou techník, ktoré to prináša na myseľ, sú techniky redukcie rozmerov, ktoré špecificky znižujú počet stĺpcov a rozsah a distribúciu numerických vstupných premenných. To zahŕňa metódy faktorizácie matice používané v lineárnej algebre, ako aj rôzne algoritmy učenia používané vo vysokorozmerných štatistikách.

Aj keď sú tieto techniky navrhnuté tak, aby vytvárali projekcie riadkov v priestore nižšej dimenzie, možno to tiež ponecháva dvere otvorené technikám, ktoré robia opak. To znamená, použiť všetky alebo podmnožinu vstupných premenných na vytvorenie projekcie do priestoru vyššej dimenzie, možno dekompilovať zložité nelineárne vzťahy.

Možno by do tejto triedy metód prípravy údajov zapadali polynómové transformácie, kde výsledky nahrádzajú nespracovaný súbor údajov.

A poslednou skupinou v našom skúmanom zozname je príprava údajov pre riadky a hodnoty. Táto skupina je určená pre techniky prípravy údajov, ktoré menia počet riadkov aj hodnoty v údajoch.

Skupina metód, ktoré prichádzajú na myseľ, sú klastrovacie algoritmy, kde sú všetky alebo podmnožiny riadkov údajov v množine údajov nahradené vzorkami údajov v centrách klastrov, ktoré sa označujú ako centroidy klastrov.

Súvisiace môže byť nahradenie riadkov príkladmi (agregátmi riadkov) prevzatými zo špecifických algoritmov strojového učenia, ako sú podporné vektory z podporného vektorového stroja alebo vektory z číselníka prevzaté z učiacej sa vektorovej kvantizácie.

Prirodzene, tieto súhrnné riadky sa jednoducho pridajú do množiny údajov, a nie nahradzujú riadky, potom by prirodzene zapadali do skupiny prípravy údajov pre riadky, ktorú sme opísali vyššie.

Typy vykonanej prípravy údajov závisia od vašich údajov. Napriek tomu, keď pracujete na viacerých projektoch prediktívneho modelovania, znova a znova vidíte a vyžadujete rovnaké typy úloh prípravy údajov. Tieto úlohy zahŕňajú:

- **Čistenie údajov** : Identifikácia a oprava chýb v údajoch.
- **Výber funkcie** : Identifikácia tých vstupných premenných, ktoré sú pre danú úlohu najrelevantnejšie.
- **Transformácie údajov** : Zmena mierky alebo distribúcie premenných.
- **Inžinierstvo funkcií** : Odvodzovanie nových premenných z dostupných údajov.
- **Redukcia rozmerov** : Vytváranie kompaktných projekcií údajov.

Je to základ, ktorý môžeme použiť na premýšľanie a navigáciu v rôznych algoritmoch prípravy údajov, ktoré môžeme zvážiť v danom projekte so štruktúrovanými alebo tabuľkovými údajmi. Pozrime sa postupne na každú z nich.

9.2 Čistenie dát

Čistenie údajov zahŕňa opravu systematických problémov alebo chýb v „ *neporiadnych* “ údajoch. Najužitočnejšie čistenie údajov zahŕňa hlboké odborné znalosti a môže zahŕňať identifikáciu a riešenie konkrétnych pozorovaní, ktoré môžu byť nesprávne.

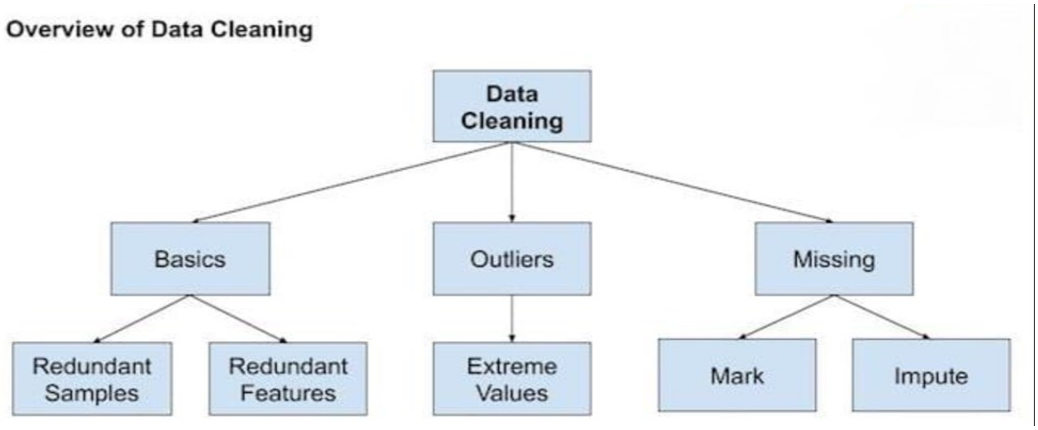
Existuje mnoho dôvodov, prečo môžu mať údaje nesprávne hodnoty, napríklad nesprávne zadané, poškodené, duplikované atď. Odbornosť domény môže umožniť identifikáciu zjavne chybných pozorovaní, pretože sa líšia od očakávaných, ako napríklad výška osoby 200 stôp.

Keď sa identifikujú chaotické, hlučné, skorumpované alebo chybné pozorovania, možno ich riešiť. Môže to zahŕňať odstránenie riadka alebo stĺpca. Alternatívne to môže zahŕňať nahradenie pozorovaní novými hodnotami.

Napriek tomu existujú všeobecné operácie čistenia údajov, ktoré možno vykonať, ako napríklad:

- Použitie štatistík na definovanie normálnych údajov a identifikáciu odľahlých hodnôt.
- Identifikácia stĺpcov, ktoré majú rovnakú hodnotu alebo žiadnu odchýlku, a ich odstránenie.
- Identifikácia duplicitných riadkov údajov a ich odstránenie.
- Označenie prázdnych hodnôt ako chýbajúcich.
- Dopočítanie chýbajúcich hodnôt pomocou štatistík alebo naučeného modelu.

Čistenie údajov je operácia, ktorá sa zvyčajne vykonáva ako prvá pred ďalšími operáciami prípravy údajov.



Obrázok 124 Prehľad čistenia dát

Čistenie dát je kriticky dôležitý krok v každom projekte strojového učenia. V tabuľkových údajoch existuje veľa rôznych techník štatistickej analýzy a vizualizácie údajov, ktoré môžete použiť na preskúmanie údajov s cieľom identifikovať operácie čistenia údajov, ktoré možno budete chcieť vykonať(Abeykoon et al., 2020).

Predtým, ako prejdete na sofistikované metódy, existuje niekoľko veľmi základných operácií čistenia údajov, ktoré by ste pravdepodobne mali vykonať v každom jednom projekte strojového učenia. Sú také základné, že ich skúsení odborníci na strojové učenie často prehliadajú, no sú také kritické, že ak sa preskočia, modely sa môžu pokaziť alebo vykazovať príliš optimistické výsledky.

V tejto časti objavíte základné čistenie údajov, ktoré by ste mali vždy vykonávať vo svojom súbore údajov. Na konci tejto časti budete vedieť:

- Ako identifikovať a odstrániť premenné stĺpca, ktoré majú iba jednu hodnotu.
- Ako identifikovať a zvážiť stĺpcové premenné s veľmi malým počtom jedinečných hodnôt.
- Ako identifikovať a odstrániť riadky, ktoré obsahujú duplicitné pozorovania.

Skusmé uviesť všetky možne kroky na čistenie údajov, je ich 7 a to sú:

- Chaotické množiny údajov
- Identifikácia stĺpcov, ktoré obsahujú jednu hodnotu
- Odstránenie stĺpcov, ktoré obsahujú jednu hodnotu
- Zváženie/identifikácia stĺpcov, ktoré majú veľmi málo hodnôt
- Odstránenie stĺpcov, ktoré majú nízky rozptyl
- Identifikácia riadkov, ktoré obsahujú duplicitné údaje
- Odstránenie riadkov, ktoré obsahujú duplicitné údaje

Čistenie údajov sa týka identifikácie a opravy chýb v súbore údajov, ktoré môžu negatívne ovplyvniť prediktívny model. Čistenie údajov sa používa na označenie všetkých druhov úloh a činností na zistenie a opravu chýb v údajoch.

Aj keď je čistenie dát kriticky dôležité, nie je vzrušujúce, nezahŕňa vymyslené techniky. Stačí dobrá znalosť dátového súboru. V množine údajov existuje mnoho typov chýb, aj keď niektoré z najjednoduchších chýb zahŕňajú stĺpce, ktoré neobsahujú veľa informácií, a duplicitné riadky.

Identifikácia stĺpcov, ktoré obsahujú jednu hodnotu - stĺpce, ktoré majú jedno pozorovanie alebo hodnotu, sú pravdepodobne na modelovanie zbytočné. Tieto stĺpce alebo prediktory sa označujú ako prediktory s nulovým rozptylom, ako keby sme merali rozptyl (priemernú hodnotu od priemeru), bol by nulový. „Keď prediktor obsahuje jednu hodnotu, nazývame to prediktor s nulovým rozptylom, pretože prediktor skutočne nezobrazuje žiadnu odchýlku.“

Jedna hodnota tu znamená, že každý riadok pre daný stĺpec má rovnakú hodnotu. Napríklad stĺpec X1 má hodnotu 1,0 pre všetky riadky v množine údajov:

```
1 X1
2 1.0
3 1.0
4 1.0
5 1.0
6 1.0
7 ...
```

Obrázok 125 Ukážka hodnôt stĺpcov

Stĺpce, ktoré majú jednu hodnotu pre všetky riadky, neobsahujú žiadne informácie pre modelovanie.

V závislosti od výberu algoritmov prípravy údajov a modelovania môžu premenné s jednou hodnotou spôsobiť chyby alebo neočakávané výsledky. Riadky, ktoré majú túto vlastnosť, môžete zistiť pomocou funkcie unique() NumPy , ktorá bude hlásiť počet jedinečných hodnôt v každom stĺpci. Nižšie uvedený príklad načíta súbor údajov klasifikácie ropných škvŕn, ktorý obsahuje 50 premenných a sumarizuje počet jedinečných hodnôt pre každý stĺpec.

```
# summarize the number of unique values for each column using numpy
from urllib.request import urlopen
from numpy import loadtxt
from numpy import unique
# define the location of the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/oil-spill.csv'
# load the dataset
data = loadtxt(urlopen(path), delimiter=',')
# summarize the number of unique values in each column
for i in range(data.shape[1]):
 print(i, len(unique(data[:, i])))
```

Obrázok 126 Ukážka načítania súboru údajov

Spustenie príkladu načíta množinu údajov priamo z adresy URL a vytlačí počet jedinečných hodnôt pre každý stĺpec Vidíme, že index stĺpca 22 má iba jednu hodnotu a mal by byť odstránený.

```
0 238
1 297
2 927
3 933
4 179
5 375
6 820
7 618
8 561
9 57
10 577
11 59
12 73
13 107
14 53
15 91
16 893
17 810
18 170
19 53
20 68
21 9
22 1
23 92
24 9
25 8
26 9
27 308
28 447
29 392
30 107
31 42
32 4
33 45
34 141
35 110
36 3
37 758
38 9
39 9
40 388
41 220
42 644
43 649
44 499
45 2
46 937
47 169
48 286
49 2
```

Obrázok 127 Ukážka indexov stĺpcov

Jednoduchším prístupom je použitie funkcie nunique() Pandas , ktorá urobí ťažkú prácu za vás. Skusmé aj uviesť rovnaký príklad ale s použitím funkcie Pandas.

```
# summarize the number of unique values for each column using numpy
from pandas import read_csv
# define the location of the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/oil-spill.csv'
# load the dataset
df = read_csv(path, header=None)
# summarize the number of unique values in each column
print(df.nunique())
```

Obrázok 128 Ukážka načítania súboru s použitím funkcie Pandas

Spustením príkladu dostaneme rovnaký výsledok, index stĺpca a počet jedinečných hodnôt pre každý stĺpec.

```
0     238
1     297
2     927
3     933
4     179
5     375
6     820
7     618
8     561
9      57
10    577
11     59
12     73
13    107
14     53
15     91
16    893
17    810
18    170
19     53
20     68
21      9
22      1
23     92
24      9
25      8
26      9
27    308
28    447
29    392
30    107
31     42
32      4
33     45
34    141
35    110
36      3
37    758
38      9
39      9
40    388
41    220
42    644
43    649
44    499
45      2
46    937
47    169
48    286
49      2
dtype: int64
```

Obrázok 129 Ukážka indexov stĺpcov s použitím funkcie Pandas

Odstránenie stĺpcov, ktoré obsahujú jednu hodnotu - premenné alebo stĺpce, ktoré majú jednu hodnotu, by mali byť pravdepodobne odstránené z našej množiny údajov.

Stĺpce sa dajú pomerne ľahko odstrániť z poľa NumPy alebo Pandas DataFrame. Jedným z prístupov je zaznamenať všetky stĺpce, ktoré majú jednu jedinečnú hodnotu, a potom ich odstrániť z Pandas DataFrame zavolaním funkcie drop().

Zváženie/identifikácia stĺpcov, ktoré majú veľmi málo hodnôt - v predchádzajúcej časti sme videli, že niektoré stĺpce vo vzorovej množine údajov mali veľmi málo jedinečných hodnôt. Napríklad existovali stĺpce, ktoré mali iba 2, 4 a 9 jedinečných hodnôt. To môže mať zmysel pre ordinálne alebo kategorické premenné. V tomto prípade súbor údajov obsahuje iba číselné premenné. Ako také, len 2, 4 alebo 9 jedinečných číselných hodnôt v stĺpci môže byť prekvapujúce. Tieto stĺpce alebo prediktory môžeme označovať ako prediktory takmer nulového rozptylu, keďže ich rozptyl nie je nula, ale veľmi malé číslo blízke nule.

V závislosti od výberu algoritmov prípravy údajov a modelovania môžu premenné s veľmi malým počtom číselných hodnôt tiež spôsobiť chyby alebo neočakávané výsledky. Napríklad, môže sa stáť že spôsobujú chyby pri používaní výkonových transformácií na prípravu údajov a pri zostavovaní lineárnych modelov, ktoré predpokladajú „ rozumné “ rozdelenie pravdepodobnosti údajov.

Ak chceme zvýrazniť stĺpce tohto typu, da sa vypočítať počet jedinečných hodnôt pre každú premennú ako percento z celkového počtu riadkov v množine údajov. Urobme to ručne pomocou NumPy. Úplný príklad môžeme vidieť na obrázku.

```
# summarize the percentage of unique values for each column using numpy
from urllib.request import urlopen
from numpy import loadtxt
from numpy import unique
# define the location of the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/oil-spill.csv'
# load the dataset
data = loadtxt(urlopen(path), delimiter=',')
# summarize the number of unique values in each column
for i in range(data.shape[1]):
 num = len(unique(data[:, i]))
 percentage = float(num) / data.shape[0] * 100
 print('%d, %d, %.1f%%' % (i, num, percentage))
```

Obrázok 130 Príklad výpočtu počtu jedinečných hodnôt pre každú premennú ako percento z celkového počtu riadkov v množine údajov

Spustenie príkladu uvádza index stĺpca a počet jedinečných hodnôt pre každý stĺpec, za ktorým nasleduje percento jedinečných hodnôt zo všetkých riadkov v množine údajov.

Tu vidíme, že niektoré stĺpce majú veľmi nízke percento jedinečných hodnôt, napríklad menej ako 1 percento.

```
0, 238, 25.4%
1, 297, 31.7%
2, 927, 98.9%
3, 933, 99.6%
4, 179, 19.1%
5, 375, 40.0%
6, 820, 87.5%
7, 618, 66.0%
8, 561, 59.9%
9, 57, 6.1%
10, 577, 61.6%
11, 59, 6.3%
12, 73, 7.8%
13, 107, 11.4%
14, 53, 5.7%
15, 91, 9.7%
16, 893, 95.3%
17, 810, 86.4%
18, 170, 18.1%
19, 53, 5.7%
20, 68, 7.3%
21, 9, 1.0%
22, 1, 0.1%
23, 92, 9.8%
24, 9, 1.0%
25, 8, 0.9%
26, 9, 1.0%
27, 308, 32.9%
28, 447, 47.7%
29, 392, 41.8%
30, 107, 11.4%
31, 42, 4.5%
32, 4, 0.4%
33, 45, 4.8%
34, 141, 15.0%
35, 110, 11.7%
36, 3, 0.3%
37, 758, 80.9%
38, 9, 1.0%
39, 9, 1.0%
40, 388, 41.4%
41, 220, 23.5%
42, 644, 68.7%
43, 649, 69.3%
44, 499, 53.3%
45, 2, 0.2%
46, 937, 100.0%
47, 169, 18.0%
48, 286, 30.5%
49, 2, 0.2%
```

Obrázok 131 Výsledok výpočtu počtu jedinečných hodnôt

Ak chceme napríklad vymazať všetkých 11 stĺpcov s jedinečnými hodnotami menšími ako 1 percento riadkov, vieme to urobiť pomocou príkladu nižšie.

```
# delete columns where number of unique values is less than 1% of the rows
from pandas import read_csv
# define the location of the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/oil-spill.csv'
# load the dataset
df = read_csv(path, header=None)
print(df.shape)
# get number of unique values for each column
counts = df.nunique()
# record columns to delete
to_del = [i for i,v in enumerate(counts) if (float(v)/df.shape[0]*100) < 1]
print(to_del)
# drop useless columns
df.drop(to_del, axis=1, inplace=True)
print(df.shape)
```

Obrázok 132Príklad odstránenia všetkých stĺpcov s jedinečnými hodnotami menšími ako 1 percento riadkov

Spustenie príkladu najprv načíta množinu údajov a vykáže počet riadkov a stĺpcov. Vypočíta sa počet jedinečných hodnôt pre každý stĺpec a identifikujú sa stĺpce, ktoré majú počet jedinečných hodnôt menší ako 1 percento riadkov. V tomto prípade 11 stĺpcov. Identifikované stĺpce sa potom odstránia z dátového rámca a nahlási sa počet riadkov a stĺpcov v dátovom rámci, aby sa potvrdila zmena.

Ďalším prístupom k problému odstraňovania stĺpcov s malým počtom jedinečných hodnôt je zvážiť rozptyl stĺpca. Pripomeňme, že rozptyl je štatistika vypočítaná na premennej ako priemerná štvorcová odchýlka hodnôt vo vzorke od priemeru.

Odchýlku možno použiť ako filter na identifikáciu stĺpcov, ktoré sa majú odstrániť z množiny údajov. Stĺpec, ktorý má jednu hodnotu, má rozptyl 0,0 a stĺpec, ktorý má veľmi málo jedinečných hodnôt, bude mať malú hodnotu rozptylu. Trieda *VarianceThreshold* z knižnice scikit-learn to podporuje ako typ výberu funkcií. Inštanciu triedy je možné vytvoriť špecifikovaním argumentu „ prah “, ktorého predvolená hodnota je 0,0 na odstránenie stĺpcov s jednou hodnotou. Potom sa dá prispôsobiť a aplikovať na množinu údajov zavolaním funkcie *fit_transform()* na vytvorenie transformovanej verzie množiny údajov, z ktorej sa automaticky odstránia stĺpce, ktoré majú odchýlku nižšiu ako prahová hodnota. Potom sa vytvorí čiarový graf zobrazujúci vzťah medzi prahovou hodnotou a počtom prvkov v transformovanom súbore údajov.

```
...
# define the transform
transform = VarianceThreshold()
# transform the input data
X_sel = transform.fit_transform(X)
```

Obrázok 133Príklad s zavolaním funkcie fit_transform()

Identifikácia riadkov, ktoré obsahujú duplicitné údaje - riadky, ktoré majú identické údaje, sú pri vyhodnocovaní modelu pravdepodobne zbytočné, ak nie nebezpečne zavádzajúce.

Z pravdepodobnostného hľadiska si duplicitné údaje môžete predstaviť ako úpravu priorít pre označenie triedy alebo distribúciu údajov. To môže pomôcť algoritmu, ako je Naive Bayes , ak chcete cielene skresľovať priority. Zvyčajne to tak nie je a algoritmy strojového učenia budú fungovať lepšie, ak identifikujú a odstránia riadky s duplicitnými údajmi.

Na zistenie vieme použiť funkciu pandas *duplicated()* ktorá oznámi, či je daný riadok duplikovaný alebo nie. Všetky riadky sú označené buď ako False, čo znamená, že nejde o duplikát, alebo ako True, čo znamená, že ide o duplikát. Ak existujú duplikáty, prvý výskyt riadka je označený ako False (predvolene), ako by sme mohli očakávať.

Uvedieme aj príklad ktorý kontroluje duplikáty.

```
# locate rows of duplicate data
from pandas import read_csv
# define the location of the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'
# load the dataset
df = read_csv(path, header=None)
# calculate duplicates
dups = df.duplicated()
# report if there are any duplicates
print(dups.any())
# list all duplicate rows
print(df[dups])
```

Obrázok 134 Príklad ktorý kontroluje duplikáty

Spustenie príkladu najprv načíta množinu údajov a potom vypočíta duplikáty riadkov. Najprv sa nahlási prítomnosť akýchkoľvek duplicitných riadkov a v tomto prípade môžeme vidieť, že existujú duplikáty (True). Potom sa nahlásia všetky duplicitné riadky. V tomto prípade vidíme, že sa zobrazili tri duplicitné riadky, ktoré boli identifikované.

```
True
      0    1    2    3              4
34   4.9  3.1  1.5  0.1    Iris-setosa
37   4.9  3.1  1.5  0.1    Iris-setosa
142  5.8  2.7  5.1  1.9  Iris-virginica
```

Obrázok 135 Výsledok výpočtu duplikátov riadkov

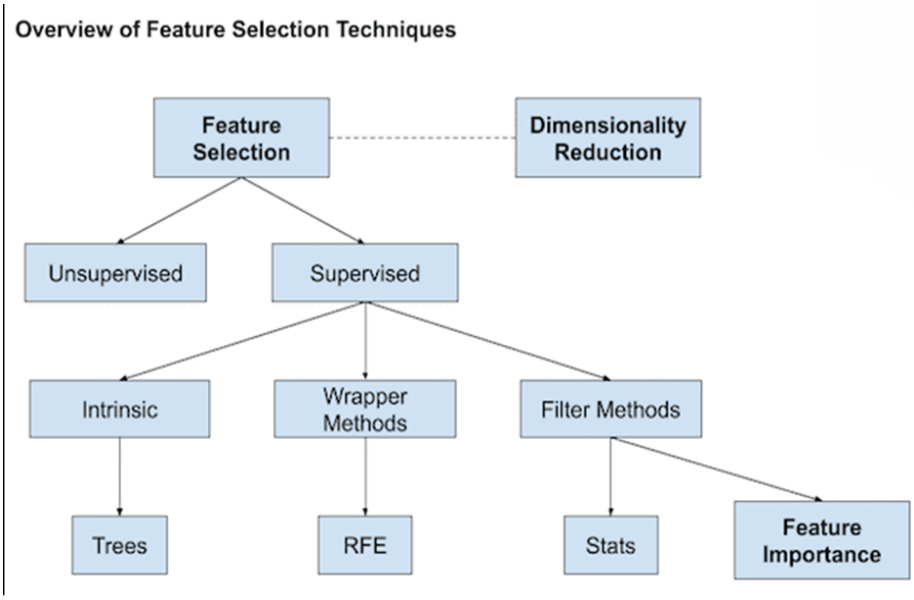
Riadky duplicitných údajov by sa pravdepodobne mali pred modelovaním z vašej množiny údajov odstrániť. Existuje mnoho spôsobov, ako to dosiahnuť, hoci Pandas poskytuje funkciu *drop_duplicates()*, ktorá presne toto dosahuje. Princíp fungovania tejto funkcie je nasledovný: po spustení príkladu najprv sa načíta množina údajov a vypíše počet riadkov a stĺpcov. Ďalej sa identifikujú riadky duplicitných údajov a odstránia sa z dátového rámca. Potom sa ohlási tvar DataFrame, aby sa potvrdila zmena.

9.3 Výber funkcií

Výber funkcii sa týka techník na výber podmnožiny vstupných vlastností, ktoré sú najrelevantnejšie pre cieľovú premennú, ktorá sa predpovedá.

Je to dôležité, pretože irelevantné a nadbytočné vstupné premenné môžu rozptyľovať alebo zavádzať algoritmy učenia, čo môže viesť k nižšej prediktívnej výkonnosti. Okrem toho je žiaduce vyvíjať modely len s použitím údajov, ktoré sú potrebné na predpovedanie, napr. na uprednostnenie najjednoduchšieho možného dobre fungujúceho modelu(Housley, 2022).

Techniky výberu funkcií sú vo všeobecnosti rozdelené na tie, ktoré používajú cieľovú premennú **supervised (pod dohľadom)** a na tie, ktoré ju nepoužívajú **unsupervised** (**bez dozoru**)**.** Okrem toho možno techniky pod dohľadom ďalej rozdeliť na modely, ktoré automaticky vyberajú funkcie ako súčasť prispôsobenia modelu **vnútorný** (**intrinsic)**, tie, ktoré si explicitne vyberajú funkcie, ktorých výsledkom je najlepší model **wrapper** (**obal**) a tie, ktoré hodnotia každú vstupnú funkciu a umožňujú podmnožinu, ktorá sa má vybrať (**filter**).



Obrázok 136 Prehľad budúcich techník výberu

Štatistické metódy sú obľúbené na hodnotenie vstupných funkcií, ako je korelácia. Funkcie potom možno zoradiť podľa ich skóre a podmnožiny s najväčším skóre použitým ako vstup do modelu. Výber štatistickej miery závisí od typov údajov vstupných premenných a prehľadu rôznych štatistických mier, ktoré je možné použiť.

Výber funkcií je proces znižovania počtu vstupných premenných pri vývoji prediktívneho modelu.

Je žiaduce znížiť počet vstupných premenných, aby sa znížili výpočtové náklady modelovania a v niektorých prípadoch sa zlepšila výkonnosť modelu. Štatistické metódy výberu vlastností zahŕňajú vyhodnotenie vzťahu medzi každou vstupnou premennou a cieľovou premennou pomocou štatistiky a výber tých vstupných premenných, ktoré majú najsilnejší vzťah s cieľovou premennou. Tieto metódy môžu byť rýchle a efektívne, hoci výber štatistických mier závisí od typu údajov vstupných aj výstupných premenných.

Pre odborníka na strojové učenie môže byt náročné vybrať vhodnú štatistickú mieru pre súbor údajov pri výbere funkcií na základe filtra. V tejto časti zistíte, ako zvoliť štatistické miery pre výber funkcií na základe filtra s číselnými a kategorickými údajmi. Po ich prečítaní budete vedieť:

- Existujú dva hlavné typy techník výberu prvkov: pod dohľadom a bez dozoru a kontrolované metódy možno rozdeliť na obalové, filtračné a vnútorné.
- Metódy výberu funkcií založené na filtri používajú štatistické merania na hodnotenie korelácie alebo závislosti medzi vstupnými premennými, ktoré možno filtrovať a vybrať tie najrelevantnejšie funkcie.
- Štatistické miery pre výber vlastností musia byť starostlivo zvolené na základe dátového typu vstupnej premennej a výstupnej alebo odozvovej premennej.

Celý postup je rozdelený na 4 časti:

- Metódy výberu funkcií
- Štatistika pre metódy výberu prvkov filtra
- Tipy a triky pre výber funkcií
- Spracované príklady

Metódy výberu prvkov sú určené na zníženie počtu vstupných premenných na tie, o ktorých sa predpokladá, že sú pre model najužitočnejšie, aby bolo možné predpovedať cieľovú premennú. Výber funkcií je primárne zameraný na odstránenie neinformatívnych alebo nadbytočných prediktorov z modelu.

Niektoré problémy s prediktívnym modelovaním majú veľký počet premenných, ktoré môžu spomaliť vývoj a trénovanie modelov a vyžadujú veľké množstvo systémovej pamäte. Okrem toho sa môže výkon niektorých modelov zhoršiť, ak zahrniete vstupné premenné, ktoré nie sú relevantné pre cieľovú premennú.

Jedným zo spôsobov, metód výberu funkcií, sú metódy pod **dohľadom** a **bez dozoru**. Rozdiel súvisí s tým, či sú funkcie vybrané na základe cieľovej premennej alebo nie. Techniky výberu funkcií bez dozoru ignorujú cieľovú premennú, ako sú metódy, ktoré odstraňujú nadbytočné premenné pomocou korelácie. Techniky výberu funkcií pod dohľadom používajú cieľovú premennú, napríklad metódy, ktoré odstraňujú irelevantné premenné.

Ďalší spôsob, ako zvážiť mechanizmus používaný na výber funkcií, ktoré možno rozdeliť na metódy **wrapper** a **filtre.** Tieto metódy sú takmer vždy pod dohľadom a sú hodnotené na základe výkonu výsledného modelu na množine údajov.

Metódy výberu funkcií Wrapper vytvárajú mnoho modelov s rôznymi podmnožinami vstupných funkcií a vyberajú tie funkcie, ktorých výsledkom je model s najlepšou výkonnosťou podľa metriky výkonu. Tieto metódy sa netýkajú typov premenných, hoci môžu byť výpočtovo nákladné. RFE je dobrým príkladom metódy výberu prvkov. Metódy výberu prvkov filtra používajú štatistické techniky na vyhodnotenie vzťahu medzi každou vstupnou premennou a cieľovou premennou a tieto skóre sa používajú ako základ na výber (filtrovanie) tých vstupných premenných, ktoré sa použijú v modeli.

Nakoniec existuje niekoľko algoritmov strojového učenia, ktoré vykonávajú výber funkcií automaticky ako súčasť učenia sa modelu. Tieto techniky by sme mohli označovať ako metódy výberu **vnútorných** funkcií. Patria sem algoritmy, ako sú penalizované regresné modely ako Lasso a rozhodovacie stromy, vrátane súborov rozhodovacích stromov, ako je náhodný les.

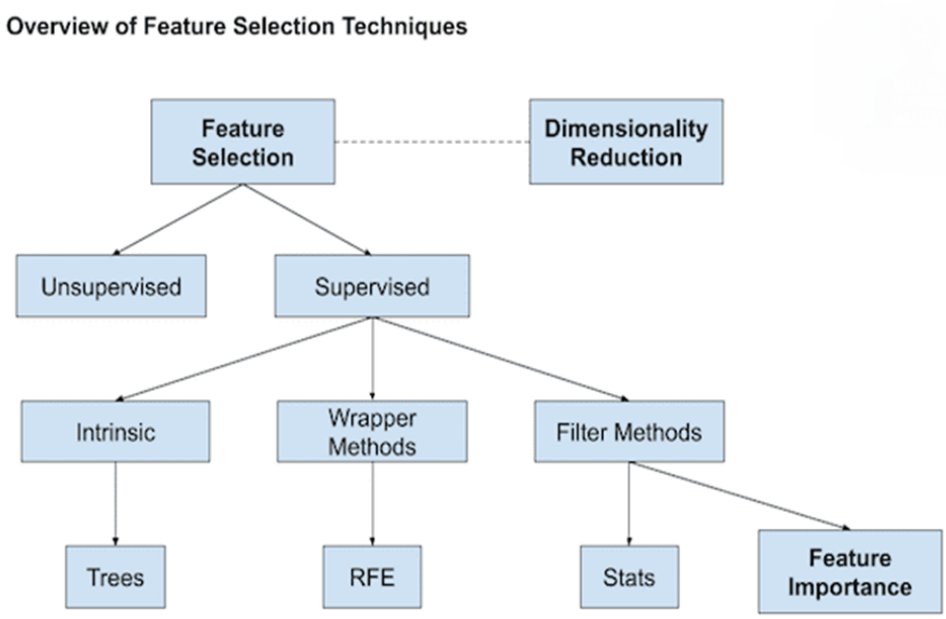
Výber prvkov tiež súvisí s technikami redukcie rozmerov v tom, že obe metódy hľadajú menej vstupných premenných do prediktívneho modelu. Rozdiel je v tom, že výber funkcií vyberá funkcie, ktoré sa majú zachovať alebo odstrániť z množiny údajov, zatiaľ čo redukcia rozmerov vytvára projekciu údajov, ktorá vedie k úplne novým vstupným vlastnostiam. Redukcia rozmerov ako taká je skôr alternatívou k výberu prvkov než typom výberu prvkov.

Výber funkcií môžeme zhrnúť nasledovne:

- **Výber funkcií**: Vyberte podmnožinu vstupných funkcií z množiny údajov.
  - **Unsupervised**: Nepoužívajte cieľovú premennú (napr. odstráňte nadbytočné premenné).
    - Korelácia
  - **Pod dohľadom:** Použite cieľovú premennú (napr. odstráňte irelevantné premenné).
    - **Wrapper** : Vyhľadajte dobre fungujúce podmnožiny funkcií.
      - **RFE**
    - **Filter** : Vyberte podmnožiny prvkov na základe ich vzťahu k cieľu.
      - Štatistické metódy
      - Metódy dôležitosti funkcie
    - **Intrinsic**: Algoritmy, ktoré vykonávajú automatický výber funkcií počas tréningu.
      - Rozhodovacie stromy

- **Redukcia rozmerov** : Premietajte vstupné údaje do priestoru prvkov nižšej dimenzie.

Obrázok nižšie poskytuje súhrn tejto hierarchie techník výberu funkcií.



Obrázok 137 Prehľad budúcich techník výberu – súhrn hierarchie

V ďalšej časti preskúmame niektoré štatistické opatrenia, ktoré možno použiť na výber funkcií na základe filtra s rôznymi typmi údajov vstupných a výstupných premenných. Je bežné používať štatistické merania typu korelácie.

Výber štatistických ukazovateľov ako taký veľmi závisí od rôznych typov údajov. Bežné typy údajov zahŕňajú numerické (napríklad výšku) a kategorické (napríklad štítok), hoci každý môže byť ďalej rozdelený, ako napríklad celé číslo a pohyblivá rádová čiarka pre numerické, boolovské a ordinálne premenné:

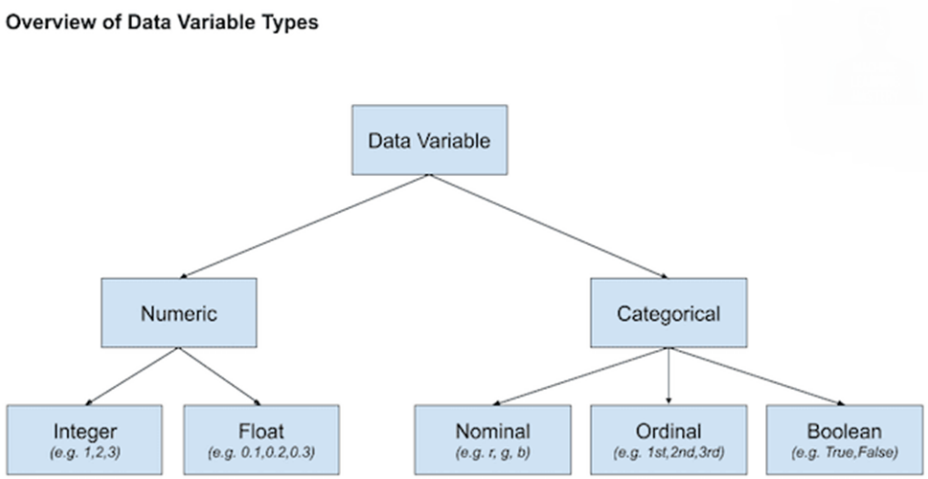
Numerické premenné

- Celočíselné premenné.
- Premenné s pohyblivou rádovou čiarkou.

Kategorické premenné

- Booleovské premenné (dichotomické).
- Ordinálne premenné.
- Nominálne premenné.

Na obrázku môžeme presne vidieť prehľad typov dátových premenných.



Obrázok 138 Prehľad typov dátových premenných

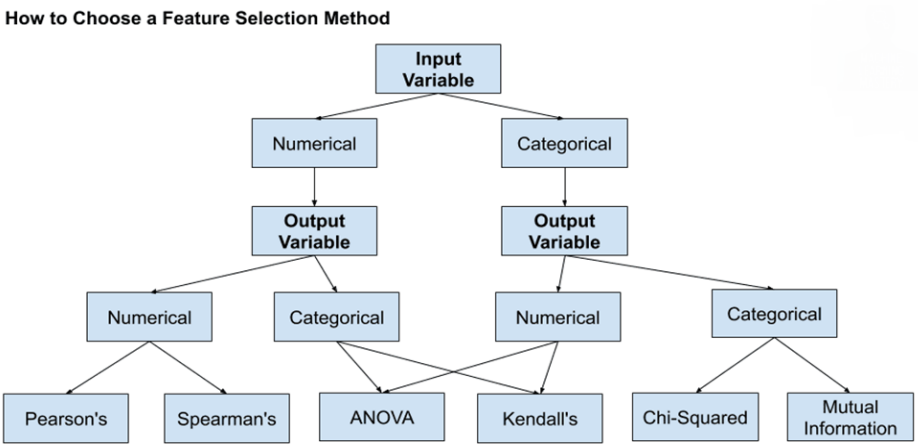
Čím viac je známe o type údajov premennej, tým ľahšie je vybrať vhodnú štatistickú mieru pre metódu výberu prvkov založenú na filtri. V tejto časti zvážime dve široké kategórie typov premenných: číselné a kategorické; tiež dve hlavné skupiny premenných, ktoré treba zvážiť: vstup a výstup.

Vstupné premenné sú tie, ktoré sa poskytujú ako vstup do modelu. Pri výbere funkcií chceme zmenšiť veľkosť tejto skupiny premenných. Výstupné premenné sú tie, pre ktoré má model predpovedať, často nazývané premenná odozvy. Typ premennej odozvy zvyčajne označuje typ vykonávaného problému prediktívneho modelovania. Napríklad numerická výstupná premenná označuje problém regresného prediktívneho modelovania a kategorická výstupná premenná označuje problém prediktívneho modelovania klasifikácie.

- **Numerický výstup** : Problém regresného prediktívneho modelovania.
- **Kategorický výstup** : Klasifikačný problém prediktívneho modelovania.

Štatistické miery používané pri výbere funkcií na základe filtra sa vo všeobecnosti vypočítavajú po jednej vstupnej premennej s cieľovou premennou. Ako také sa označujú ako jednorozmerné štatistické miery. To môže znamenať, že v procese filtrovania sa nezohľadňuje žiadna interakcia medzi vstupnými premennými.

S týmto frameworkom si pozrime niektoré jednorozmerné štatistické miery, ktoré možno použiť na výber funkcií na základe filtra.



Obrázok 139 Ako si vybrať metódu výberu funkcie

Numerický vstup, Numerický výstup. Toto je problém regresného prediktívneho modelovania s numerickými vstupnými premennými. Najbežnejšími technikami sú použitie korelačného koeficientu, ako je Pearsonov pre lineárnu koreláciu, alebo metódy založené na poradí pre nelineárnu koreláciu.

- Pearsonov korelačný koeficient (lineárny).
- Spearmanov koeficient poradia (nelineárny).

Numerický vstup, kategorický výstup. Toto je problém prediktívneho modelovania klasifikácie s numerickými vstupnými premennými. Toto môže byť najbežnejší príklad klasifikačného problému. Najbežnejšie techniky sú opäť založené na korelácii, aj keď v tomto prípade musia brať do úvahy kategorický cieľ.

- ANOVA korelačný koeficient (lineárny).
- Kendallov koeficient poradia (nelineárny).
- Kendall predpokladá, že kategorická premenná je ordinálna.

Kategorický vstup, číselný výstup. Ide o problém regresného prediktívneho modelovania s kategorickými vstupnými premennými. Toto je zvláštny príklad regresného problému (s nim sa nestretávame často). Napriek tomu môžete použiť rovnaké metódy „ Numerický vstup, Kategorický výstup “ (popísané vyššie), ale v opačnom poradí.

Kategorický vstup, Kategorický výstup. Ide o problém prediktívneho modelovania klasifikácie s kategorickými vstupnými premennými. Najbežnejšou mierou korelácie pre kategorické údaje je test chí-kvadrát . Využiť môžete aj vzájomné informovanie sa (informačný zisk) z oblasti teórie informácie.

- Chí-kvadrát test (kontingenčné tabuľky).
- Vzájomné informácie.

Vzájomné informácie sú v skutočnosti mocnou metódou, ktorá sa môže ukázať ako užitočná pre kategorické aj číselné údaje, napr. je agnostická voči typom údajov.

Tipy a triky pre výber funkcií. Táto časť poskytuje niekoľko ďalších úvah pri používaní výberu funkcií na základe filtra.

Korelačná štatistika. Knižnica scikit-learn poskytuje implementáciu väčšiny užitočných štatistických opatrení. Napríklad:

- Pearsonov korelačný koeficient: f_regression()
- ANOVA: f_classif()
- Chi-kvadrát: chi2()
- Vzájomné informácie: mutual_info_classif() a mutual_info_regression()
- Knižnica SciPy tiež poskytuje implementáciu mnohých ďalších štatistík, ako je Kendallovo tau (kendalltau) a Spearmanova korelácia hodnosti (spearmanr).

Spôsob výberu. Knižnica scikit-learn tiež poskytuje mnoho rôznych metód filtrovania po vypočítaní štatistík pre každú vstupnú premennú s cieľom. Dve z najpopulárnejších metód zahŕňajú:

- Výber horných k premenných: SelectKBest
- Výber premenné najvyššieho percentilu: SelectPercentile

Transformácia premenných. Zvážte transformáciu premenných, aby ste získali prístup k rôznym štatistickým metódam. Môžete napríklad transformovať kategorickú premennú na ordinálnu, aj keď nie je, a zistiť, či sa objavia nejaké zaujímavé výsledky.

Niektoré štatistické miery predpokladajú vlastnosti premenných, ako napríklad Pearsonova, ktorá predpokladá Gaussovo rozdelenie pravdepodobnosti k pozorovaniam a lineárny vzťah. Dáta môžete transformovať tak, aby splnili očakávania testu a vyskúšať test bez ohľadu na očakávania a porovnávať výsledky.

Neexistuje najlepší spôsob výberu funkcií. Rovnako ako neexistuje najlepšia sadá vstupných premenných alebo najlepší algoritmus strojového učenia. Aspoň nie univerzálne. Namiesto toho musíte pomocou starostlivého systematického experimentovania zistiť, čo najlepšie funguje pre váš konkrétny problém. Vyskúšajte rad rôznych modelov, ktoré sa hodia na rôzne podskupiny funkcií vybraných prostredníctvom rôznych štatistických meraní a zistite, čo najlepšie.

Výber regresnej funkcie: (numerický vstup, numerický výstup). Táto časť demonštruje výber funkcií pre regresný problém ako numerické vstupy a numerické výstupy. Skúšobný regresný problém sa pripraví pomocou funkcie *make_regression()*. Výber vlastností sa vykonáva pomocou Pearsonovho korelačného koeficientu prostredníctvom funkcie *f_regression()*.

```
# pearson's correlation feature selection for numeric input and numeric output
from sklearn.datasets import make_regression
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
# generate dataset
X, y = make_regression(n_samples=100, n_features=100, n_informative=10)
# define feature selection
fs = SelectKBest(score_func=f_regression, k=10)
# apply feature selection
X_selected = fs.fit_transform(X, y)
print(X_selected.shape)
```

Obrázok 140 Príklad - výber funkcií pre regresný problém ako numerické vstupy a numerické výstupy

Spustením príkladu sa najskôr vytvorí regresná množina údajov, potom sa definuje výber funkcií a aplikuje sa postup výberu funkcií na množinu údajov, čím sa vráti podmnožina vybratých vstupných vlastností.

```
(100, 20)
```

Obrázok 141 Vrátený výsledok

Výber funkcie klasifikácie: (číselný vstup, kategorický výstup). Táto časť demonštruje výber vlastností pre problém klasifikácie, ktorý predstavuje numerické vstupy a kategorické výstupy. Pomocou funkcie make_classification() sa pripraví testovací regresný problém. Výber vlastností sa vykonáva pomocou merania ANOVA F prostredníctvom funkcie f_classif().

```
# ANOVA feature selection for numeric input and categorical output
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
# generate dataset
X, y = make_classification(n_samples=100, n_features=20, n_informative=2)
# define feature selection
fs = SelectKBest(score_func=f_classif, k=2)
# apply feature selection
X_selected = fs.fit_transform(X, y)
print(X_selected.shape)
```

Obrázok 142 Príklad - výber vlastností pre problém klasifikácie

Spustením príkladu sa najprv vytvorí množina údajov klasifikácie, potom sa definuje výber vlastností a aplikuje sa postup výberu prvkov na množinu údajov, čím sa vráti podmnožina vybratých vstupných vlastností.

```
(100, 2)
```

Obrázok 143 Vrátený výsledok

V tejto časti sme zistili, ako zvoliť štatistické miery pre výber funkcií na základe filtra s číselnými a kategorickými údajmi. Konkrétne sme sa dozvedeli a naučili, že existujú dva hlavné typy techník výberu prvkov: pod dohľadom a bez dozoru a kontrolované metódy možno rozdeliť na obalové, filtračné a vnútorné. Metódy výberu funkcií založené na filtri používajú štatistické merania na hodnotenie korelácie alebo závislosti medzi vstupnými premennými, ktoré možno filtrovať a vybrať tie najrelevantnejšie funkcie. Štatistické miery pre výber vlastností musia byť starostlivo zvolené na základe dátového typu vstupnej a výstupnej premennej.

Okrem toho existujú bežné prípady použitia výberu funkcií, s ktorými sa môžete ešte stretnúť pri prediktívnom modelovaní, ako napríklad: Kategorické vstupy pre cieľovú premennú klasifikácie; Číselné vstupy pre cieľovú premennú klasifikácie; Číselné vstupy pre cieľovú premennú regresie.

Ak je prítomná zmes dátových typov vstupných premenných, môžu sa použiť rôzne metódy filtrovania. Alternatívne možno použiť metódu wrapper, ako je napríklad populárna metóda RFE, ktorá je agnostika k typu vstupnej premennej. Širšia oblasť hodnotenia relatívnej dôležitosti vstupných funkcií sa označuje ako dôležitosť funkcie a existuje veľa techník založených na modeli, ktorých výstupy možno použiť na pomoc pri interpretácii modelu, interpretácii súboru údajov alebo pri výbere funkcií na modelovanie.

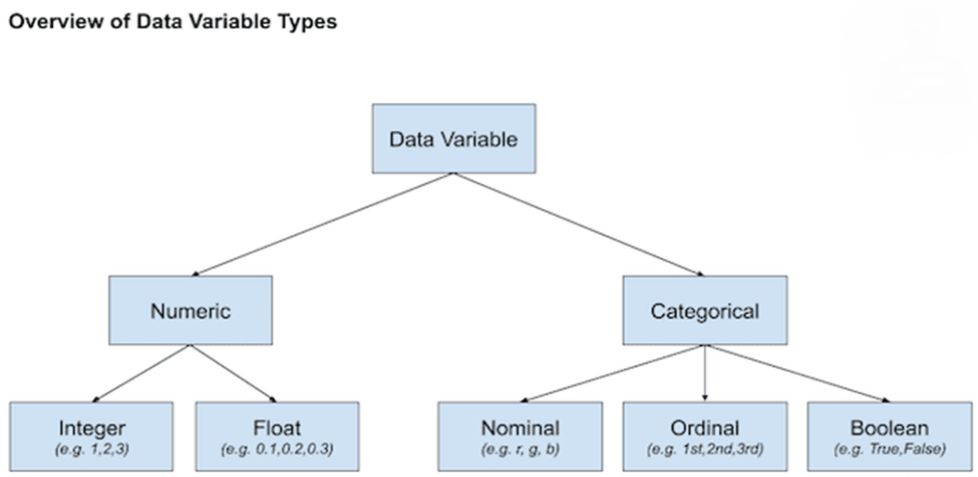
9.4 Transformácia údajov

Dátové transformácie sa používajú na zmenu typu alebo distribúcie dátových premenných. Toto je veľký dáždnik rôznych techník a možno ich rovnako ľahko aplikovať na vstupné a výstupné premenné.

Pripomeňme, že údaje môžu mať jeden z niekoľkých typov, napríklad numerický alebo kategorický , pričom každý má podtypy, napríklad celé číslo a reálnu hodnotu pre numerické a nominálne, ordinálne a boolovské pre kategorické. Skusmé o týchto typoch povedať trosku viac.

- **Číselný typ údajov** : číselné hodnoty.
- **Celé číslo** : Celé čísla bez zlomkovej časti.
- **Skutočné** : Hodnoty s pohyblivou rádovou čiarkou.
- **Typ kategorických údajov** : Hodnoty štítkov.
- **Ordinal** : Štítky s poradím podľa hodnosti.
- **Nominálne** : Štítky bez poradia podľa poradia.
- **Boolean** : Hodnoty True a False.

Obrázok nižšie poskytuje prehľad rovnakého rozdelenia typov údajov na vysokej úrovni.



Obrázok 144 Prehľad typov dátových premenných rozdelenia typov údajov na vysokej úrovni

Môžeme si želať previesť číselnú premennú na ordinálnu premennú v procese nazývanom diskretizácia. Alternatívne môžeme zakódovať kategorickú premennú ako celé čísla alebo booleovské premenné, ktoré sa vyžadujú pri väčšine klasifikačných úloh:

- **Diskretizačná transformácia** : Kódovanie číselnú premennú ako ordinálnu premennú.
- **Ordinal Transform** : Kódovanie kategorickej premennej do celočíselnej premennej.
- **One-Hot Transform** : Kódovanie kategorickú premennú do binárnych premenných.

Pre číselné premenné s reálnou hodnotou spôsob, akým sú reprezentované v počítači , znamená, že v rozsahu 0-1 je výrazne väčšie rozlíšenie ako v širšom rozsahu typu údajov. Ako také môže byť žiaduce škálovať premenné do tohto rozsahu, nazývaného normalizácia. Ak majú údaje gaussovské rozdelenie pravdepodobnosti, môže byť užitočnejšie posunúť údaje na štandardné gaussovské s priemerom nula a štandardnou odchýlkou jedna.

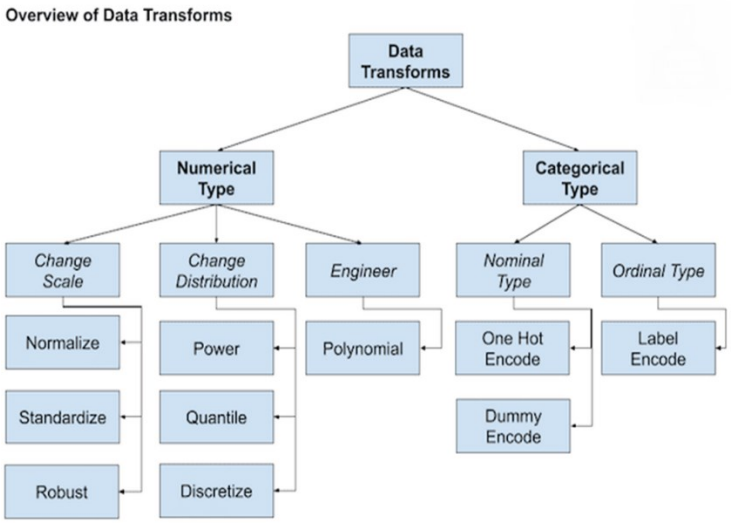
- **Normalizačná transformácia** : Škálovanie premennej na rozsah 0 a 1.
- **Štandardizačná transformácia** : Škálovanie premennej na štandardný Gaussian.

Rozdelenie pravdepodobnosti pre číselné premenné možno zmeniť.

Napríklad, ak je rozdelenie takmer gaussovské, ale je zošikmené alebo posunuté, môže byť gaussovské pomocou transformácie výkonu. Alternatívne možno použiť kvantilové transformácie na vynútenie rozdelenia pravdepodobnosti, ako je rovnomerné alebo Gaussovské na premennej s nezvyčajným prirodzeným rozdelením.

- **Power Transform** : Zmeňte rozdelenie premennej tak, aby bolo viac Gaussovské.
- **Quantile Transform**: Zaveďte rozdelenie pravdepodobnosti, ako napríklad rovnomerné alebo Gaussovské.

Dôležitým aspektom pri transformáciách údajov je, že operácie sa vo všeobecnosti vykonávajú samostatne pre každú premennú. Preto môžeme chcieť vykonávať rôzne operácie s rôznymi typmi premenných.



Obrázok 145 Prehľad transformácií údajov

V budúcnosti ak budeme chcieť použiť transformáciu aj na nové údaje. Dá sa to dosiahnuť uložením transformačných objektov do súboru spolu s konečným modelom natrénovaným na všetkých dostupných údajoch.

9.5 Funkcia inžinierstva (Feature Engineering)

Funkcia inžinierstva sa týka procesu vytvárania nových vstupných premenných z dostupných údajov.

Vytváranie nových funkcií je vysoko špecifické pre vaše údaje a typy údajov. Preto si to často vyžaduje spoluprácu odborníka na danú problematiku, ktorý pomôže identifikovať nové funkcie, ktoré by sa dali skonštruovať z údajov.

Táto špecializácia z neho robí náročnú tému na zovšeobecnenie na všeobecné metódy. Napriek tomu existuje niekoľko techník, ktoré možno opätovne použiť, ako napríklad:

- Pridanie booleovskej premennej príznaku pre nejaký stav.
- Pridanie skupinovej alebo globálnej súhrnnej štatistiky, ako je napríklad priemer.
- Pridanie nových premenných pre každý komponent zloženej premennej, ako je napríklad dátum a čas.

Obľúbeným prístupom čerpaným zo štatistík je vytváranie kópií numerických vstupných premenných, ktoré boli zmenené jednoduchou matematickou operáciou, ako je ich zvýšenie na mocninu alebo vynásobenie s inými vstupnými premennými, ktoré sa označujú ako polyfonické znaky. **Polynomiálna transformácia** : Vytvorte kópie numerických vstupných premenných, ktoré sú umocnené.

Témou inžinierstva funkcií je pridať širší kontext k jednému pozorovaniu alebo rozložiť komplexnú premennú, oboje v snahe poskytnúť priamočiarejší pohľad na vstupné údaje. Dá sa povedať o inžinierstve funkcií ako o type transformácie údajov, hoci by bolo rovnako rozumné považovať transformáciu údajov za typ inžinierstva funkcií.

9.6 Redukcia rozmerov

Počet vstupných funkcií pre súbor údajov možno považovať za rozmernosť údajov. Napríklad dve vstupné premenné spolu môžu definovať dvojrozmernú oblasť, kde každý riadok údajov definuje bod v tomto priestore. Táto myšlienka môže byť potom škálovaná na ľubovoľný počet vstupných premenných, aby sa vytvorili veľké multidimenzionálne hyper-objemy.

Problém je v tom, že čím viac rozmerov má tento priestor (napr. čím viac vstupných premenných), tým je pravdepodobnejšie, že množina údajov predstavuje veľmi riedky a pravdepodobne nereprezentatívny výber tohto priestoru. Toto sa označuje ako prekliatie dimenzionality.

To motivuje výber prvkov, hoci alternatívou k výberu prvkov je vytvorenie projekcie údajov do priestoru nižšej dimenzie, ktorý stále zachováva najdôležitejšie vlastnosti pôvodných údajov. Toto sa všeobecne označuje ako redukcia rozmerov a poskytuje alternatívu k výberu prvkov. Na rozdiel od výberu funkcií premenné v projektovaných údajoch priamo nesúvisia s pôvodnými vstupnými premennými, čo sťažuje interpretáciu projekcie.

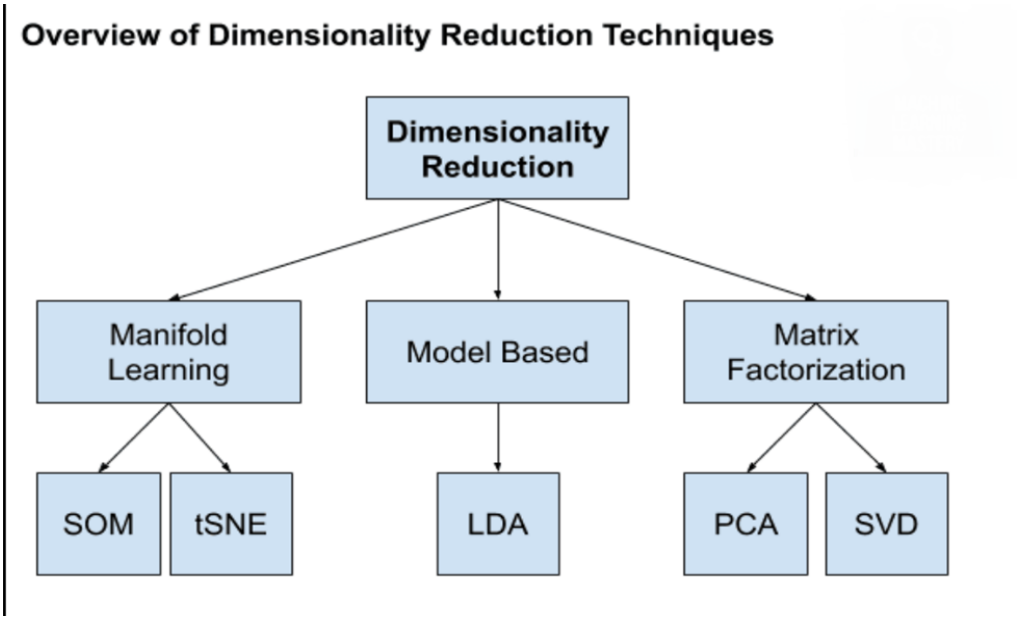
Najbežnejším prístupom k redukcii rozmerov je použitie techniky faktorizácie matice:

- Analýza hlavných komponentov (PCA)
- Dekompozícia singulárnej hodnoty (SVD)

Hlavným vplyvom týchto techník je, že odstraňujú lineárne závislosti medzi vstupnými premennými, napr. korelovanými premennými.

Existujú aj iné prístupy, ktoré objavujú nižšiu dimenzionálnu redukciu. Mohli by sme ich označovať ako metódy založené na modeloch, ako sú LDA a možno aj automatické kódovače. Lineárna diskriminačná analýza (LDA).

Niekedy sa môžu použiť aj rôzne algoritmy učenia, ako napríklad Kohonenove samoorganizujúce sa mapy a t-SNE.



Obrázok 146 Prehľad techník redukcie rozmerov

Počet vstupných premenných alebo funkcií pre súbor údajov sa označuje ako jeho rozmernosť. Zníženie rozmerov sa týka techník, ktoré znižujú počet vstupných premenných v súbore údajov. Viac vstupných funkcií často spôsobuje, že úloha prediktívneho modelovania je náročnejšia na modelovanie, všeobecnejšie označované ako prekliatie dimenzionality.

Na vizualizáciu údajov sa často používajú vysokorozmerné štatistiky a techniky redukcie rozmerov. Napriek tomu sa tieto techniky môžu použiť v aplikovanom strojovom učení na zjednodušenie klasifikácie alebo regresného súboru údajov, aby lepšie vyhovovali prediktívnemu modelu.

A teraz vám predstavíme jemný úvod do znižovania rozmerov (Dimensionality Reduction) pre strojové učenie. Hlavne časti ktoré budete vedieť po preštudovaní tejto finálnej kapitoly: veľké množstvo vstupných funkcií môže spôsobiť slabý výkon algoritmov strojového učenia; Redukcia rozmerov je všeobecná oblasť štúdia zameraná na redukciu počtu vstupných prvkov; Metódy redukcie rozmerov zahŕňajú výber prvkov, metódy lineárnej algebry, projekčné metódy a autoenkódery.

Túto kapitolu sme si rozdelili do troch častí, ktorými sa postupne budeme zaoberať, a to práve:

- Problém s mnohými vstupnými premennými
- Zníženie rozmerov
- Techniky redukcie rozmerov

Začneme prvou časťou a to **problém s mnohými vstupnými premennými**. Výkonnosť algoritmov strojového učenia sa môže zhoršiť pri príliš veľkom množstve vstupných premenných. Ak sú vaše údaje reprezentované pomocou riadkov a stĺpcov, ako napríklad v tabuľkovom hárku, vstupné premenné sú stĺpce, ktoré sa privádzajú ako vstup do modelu na predpovedanie cieľovej premennej. Vstupné premenné sa tiež nazývajú funkcie. Stĺpce údajov predstavujúce dimenzie v n-rozmernom priestore prvkov a riadky údajov môžeme považovať za body v tomto priestore. Toto je užitočná geometrická interpretácia súboru údajov.

Veľký počet rozmerov v priestore prvkov môže znamenať, že objem tohto priestoru je veľmi veľký a body, ktoré v tomto priestore máme (riadky údajov), často predstavujú malú a nereprezentatívnu vzorku. To môže dramaticky ovplyvniť výkon algoritmov strojového učenia prispôsobených údajom s mnohými vstupnými funkciami, ktoré sa všeobecne označujú ako „prekliatie dimenzionality“. Preto je často žiaduce znížiť počet vstupných funkcií. Tým sa znižuje počet rozmerov priestoru prvkov, preto sa nazýva „redukcia rozmerov“.

Druhou podstatnou pre nás častou bude **zníženie rozmerov**. Zníženie rozmerov sa týka techník na zníženie počtu vstupných premenných v tréningových údajoch.

Vysoká dimenzionalita môže znamenať stovky, tisíce alebo dokonca milióny vstupných premenných. Menej vstupných rozmerov často znamená zodpovedajúcim spôsobom menej parametrov alebo jednoduchšiu štruktúru v modeli strojového učenia, ktorý sa označuje ako stupne voľnosti. Model s príliš veľkým počtom stupňov voľnosti pravdepodobne preplní tréningový súbor údajov, a preto nemusí fungovať dobre na nových údajoch. Je žiaduce mať jednoduché modely, ktoré dobre zovšeobecňujú, a následne vstupné údaje s malým počtom vstupných premenných. To platí najmä pre lineárne modely, kde počet vstupov a stupne voľnosti modelu často úzko súvisia.

Zníženie rozmerov je technika prípravy údajov vykonaná na údajoch pred modelovaním. Môže sa vykonať po vyčistení údajov a škálovaní údajov a pred trénovaním prediktívneho modelu.

Akákoľvek redukcia rozmerov vykonaná na trénovacích údajoch sa ako taká musí vykonať aj na nových údajoch, ako je súbor testovacích údajov, overovací súbor údajov a údaje pri vytváraní predpovede s konečným modelom.

No a tretou podstatnou častou pre nás budú **techniky redukcie rozmerov.** Existuje mnoho techník, ktoré možno použiť na zníženie rozmerov. My skúsime preskúmať hlavné techniky.

Prvou z nich - **metódy výberu funkcií.** Snáď najbežnejšie sú takzvané techniky výberu prvkov, ktoré používajú skórovacie alebo štatistické metódy na výber prvkov, ktoré sa ponechajú a ktoré sa odstránia. Dve hlavné triedy techník výberu prvkov zahŕňajú metódy wrapper a metódy filtrovania.

Metódy Wrapper, ako už názov napovedá, zabalia model strojového učenia, prispôsobia a vyhodnotia model rôznymi podmnožinami vstupných funkcií a vyberú podmnožinu výsledkov s najlepším výkonom modelu. RFE je príkladom metódy výberu prvkov obalu.

Metódy filtrovania používajú metódy hodnotenia, ako je korelácia medzi objektom a cieľovou premennou, na výber podmnožiny vstupných funkcií, ktoré sú najviac prediktívne. Príklady zahŕňajú Pearsonovu koreláciu a Chí-kvadrát test.

Druhá metóda - **maticová faktorizácia.** Techniky z lineárnej algebry možno použiť na redukciu rozmerov. Konkrétne, metódy faktorizácie matice možno použiť na redukciu matice množiny údajov na jej jednotlivé časti. Príklady zahŕňajú vlastný rozklad a rozklad singulárnych hodnôt. Časti potom možno zoradiť a vybrať podmnožinu týchto častí, ktorá najlepšie vystihuje podstatnú štruktúru matice, ktorú možno použiť na reprezentáciu súboru údajov. Najbežnejšou metódou hodnotenia komponentov je analýza hlavných komponentov alebo skrátene PCA.

Ďalšou zaujímavou metódou je - **mnohostranné učenie.** Techniky z vysokorozmernej štatistiky možno použiť aj na redukciu rozmerov. Tieto techniky sa niekedy označujú ako „rôzne učenie“ a používajú sa na vytvorenie nízkorozmernej projekcie vysokorozmerných údajov, často na účely vizualizácie údajov.

Projekcia je navrhnutá tak, aby vytvorila nízkorozmernú reprezentáciu súboru údajov a zároveň čo najlepšie zachovala výraznú štruktúru alebo vzťahy v údajoch. Príklady rôznych techník učenia zahŕňajú (Naeem et al., 2022):

- Kohonen Self-Organizing Map (SOM)
- Sammonsovo mapovanie

- Viacrozmerné škálovanie (MDS)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

Prvky v projekcii majú často malý vzťah s pôvodnými stĺpcami, napr. nemajú názvy stĺpcov, čo môže byť pre začiatočníkov mätúce.

Štvrtá metóda o ktorú vám chceme predstaviť sú - **metódy automatického kódovania.** Neurónové siete s hlbokým učením môžu byť skonštruované tak, aby vykonávali redukciu rozmerov. Populárny prístup sa nazýva autoenkódery. Zahŕňa to zostavenie vzdelávacieho problému s vlastným dohľadom, kde model musí správne reprodukovať vstup.

Používa sa sieťový model, ktorý sa snaží skomprimovať dátový tok do vrstvy úzkeho miesta s oveľa menšími rozmermi ako pôvodné vstupné dáta. Časť modelu pred a vrátane úzkeho miesta sa označuje ako kodér a časť modelu, ktorá číta výstup úzkeho miesta a rekonštruuje vstup, sa nazýva dekodér.

Po naučení sa dekodér zahodí a výstup z úzkeho miesta sa použije priamo ako zmenšená dimenzionalita vstupu. Vstupy transformované týmto kódovačom môžu byť potom privedené do iného modelu, nie nevyhnutne modelu neurónovej siete.

Výstup kodéra je typ projekcie a podobne ako iné metódy projekcie neexistuje žiadny priamy vzťah k výstupu úzkeho miesta späť k pôvodným vstupným premenným, čo sťažuje ich interpretáciu.

Skusmé teda zhrnúť prejdené metódy a dáme vám tipy na zmenšenie rozmerov. Neexistuje žiadna najlepšia technika na redukciu rozmerov a žiadne mapovanie techník na problémy. Namiesto toho je najlepším prístupom použiť systematické kontrolované experimenty, aby ste zistili, aké techniky redukcie rozmerov, keď sa spárujú s modelom podľa vášho výberu, vedú k najlepšiemu výkonu vášho súboru údajov. Typicky lineárna algebra a rôzne metódy učenia predpokladajú, že všetky vstupné funkcie majú rovnakú mierku alebo distribúciu. To naznačuje, že je dobrou praxou buď normalizovať alebo štandardizovať údaje pred použitím týchto metód, ak majú vstupné premenné rôzne stupnice alebo jednotky.

Literatúra

1. Natural Language Processing in Radiology: Update on Clinical Applications. (2022). Journal of the American College of Radiology. doi:10.1016/j.jacr.2022.06.016
2. AIS2. (n.d.). Dostupné na https://ais2.ukf.sk/ais/portal2/ucitel/
3. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval (Illustrated edition). New York: Cambridge University Press.
4. Goldberg, Y. (2017). Neural Network Methods in Natural Language Processing (G. Hirst, Ed.). San Rafael, Calif.: Morgan & Claypool Publishers.
5. Slovenský WordNet. (n.d.). Slovenský národný korpus. Dostupné na https://korpus.sk/korpusy-a-databazy/databazy/wordnet/
6. Čermák, F. (2011). Jazyk a jazykověda. Karolinum Press.
7. Paralič, J. (2010). Dolovanie znalostí z textov (1st ed.). Košice: Equilibria.
8. Bella, M. (2010). Metody indexace dokumentů (Masaryk University, Faculty of Informatics). Dostupné na https://theses.cz/id/em7osa/?lang=en
9. Natural Language Query to SQL Conversion Using Machine Learning Approach. (n.d.-a). Dostupné na https://ieeexplore-1ieee-1org-1025mrzpa182c.erproxy.cvtisr.sk/document/9732586/
10. Natural Language Query to SQL Conversion Using Machine Learning Approach. (n.d.-b). Dostupné na https://ieeexplore-1ieee-1org-1025mrzpa182c.erproxy.cvtisr.sk/document/9732586
11. Arefin, M., Hossen, K. M., & Uddin, M. N. (2021, December). Natural Language Query to SQL Conversion Using Machine Learning Approach. 2021 3rd International Conference on Sustainable Technologies for Industry 4.0 (STI), 1–6. doi:10.1109/STI53101.2021.9732586
12. Kynabay, B., Aldabergen, A., & Zhamanov, A. (2021, April). Automatic Summarizing the News from Inform.kz by Using Natural Language Processing Tools. 1–4. doi:10.1109/SIST50301.2021.9465885
13. Dátová veda a jej aplikácie. (n.d.). Dostupné na https://unibook.upjs.sk/sk/informatika/1420-datova-veda-a-jej-aplikacie

14. Ľubomír Antoni, Peter Butka, Martin Sarnovský, Peter Gurský, Miroslav Opiela, Miron Kuzma, … František Babič. (2020). Dátová veda a jej aplikácie (1st ed.). Košice: Vydavateľstvo ŠafárikPress.
15. Subedi, N. (2018, July). FastText: Under the Hood. Medium. Dostupné na https://towardsdatascience.com/fasttext-under-the-hood-11efc57b2b3
16. Goldberg, Y., & Levy, O. (2014, February). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. Dostupné na http://arxiv.org/abs/1402.3722
17. Shperber, G. (2019, November). A gentle introduction to Doc2Vec. Wisio. Dostupné na https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e
18. McKinney, W. (2010). Data Structures for Statistical Computing in Python. 56–61. doi:10.25080/Majora-92bf1922-00a
19. Bird, S., Klein, E., & Loper, E. (2009b). Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit (1st edition). Beijing ; Cambridge Mass.: O'Reilly Media.
20. Sarkar, D. (2019b). Python for Natural Language Processing. In D. Sarkar (Ed.), Text Analytics with Python: A Practitioner's Guide to Natural Language Processing (pp. 69–114). doi:10.1007/978-1-4842-4354-1_2
21. Sarkar, D. (2019a). Processing and Understanding Text. In D. Sarkar (Ed.), Text Analytics with Python: A Practitioner's Guide to Natural Language Processing (pp. 115–199). doi:10.1007/978-1-4842-4354-1_3
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12(85), 2825–2830. Dostupné na http://jmlr.org/papers/v12/pedregosa11a.html
23. Sanjana Reddy. (n.d.). GloVe and fastText — Two Popular Word Vector Models in NLP. SAP. Dostupné na https://blogs.sap.com/2019/07/03/GLOVE-AND-FASTTEXT-TWO-POPULAR-WORD-VECTOR-MODELS-IN-NLP/
24. Řehůřek, R., & Sojka, P. (2011). Gensim—Statistical Semantics in Python.
25. Jozef Kapusta. (2020). Metódy analýzy webového obsahu (Prírodovedec č. 727, Vol. 1). Nitra: Univerzita Konštantína Filozofa v Nitre.

26. (N.d.). Dostupné na https://www.mendeley.com/reference-management/web-importer/uninstall-feedback/
27. Scopus - Document details - Improving aspect term extraction via span-level tag data augmentation. (n.d.). doi:10.1007/s10489-022-03558-5
28. Liu, B., Lin, T., & Li, M. (2023). Improving aspect term extraction via span-level tag data augmentation. Appl Intell, 53(3), 3207–3220. doi:10.1007/s10489-022-03558-5
29. Madukwe, K. J., Gao, X., & Xue, B. (2022). Token replacement-based data augmentation methods for hate speech detection. World Wide Web, 25(3), 1129–1150. doi:10.1007/s11280-022-01025-2
30. Saifan, A. A., & Obeidat, L. (2021). Feature location enhancement based on source code augmentation with synonyms of terms. Software: Practice and Experience, 51(2), 235–259. doi:10.1002/spe.2900
31. Marivate, V., & Sefara, T. (2020). Improving Short Text Classification Through Global Augmentation Methods. Machine Learning and Knowledge Extraction, 385–399. doi:10.1007/978-3-030-57321-8_21
32. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, September). Efficient Estimation of Word Representations in Vector Space. doi:10.48550/arXiv.1301.3781
33. Nazir, S., Asif, M., Sahi, S. A., Ahmad, S., Ghadi, Y. Y., & Aziz, M. H. (2022). Toward the Development of Large-Scale Word Embedding for Low-Resourced Language. IEEE Access, 10, 54091–54097. doi:10.1109/ACCESS.2022.3173259
34. Bird, S. (n.d.). Natural Language Processing with Python.
35. Bird, S., Klein, E., & Loper, E. (2009a). Natural language processing with Python (1st ed). Beijing ; Cambridge [Mass.]: O'Reilly.
36. Wei, J., & Zou, K. (2019, August). EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. doi:10.48550/arXiv.1901.11196
37. Řehurek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. 45–50. doi:10.13140/2.1.2393.1847
38. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013, October). Distributed Representations of Words and Phrases and their Compositionality. doi:10.48550/arXiv.1310.4546
39. Van Rossum, G. (1999). Python Library Reference. To Excel Inc.
40. Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1532–1543. doi:10.3115/v1/D14-116

41. Zhai, M., Tan, J., & Choi, J. (2016). Intrinsic and Extrinsic Evaluations of Word Embeddings. AAAI, 30(1). doi:10.1609/aaai.v30i1.9959
42. Shi, Y., Zheng, Y., Guo, K., Zhu, L., & Qu, Y. (2018, November). Intrinsic or Extrinsic Evaluation: An Overview of Word Embedding Evaluation. 2018 IEEE International Conference on Data Mining Workshops (ICDMW), 1255–1262. doi:10.1109/ICDMW.2018.00179
43. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2012). Scikit-learn: Machine Learning in Python. doi:10.48550/ARXIV.1201.0490
44. Verma, P. K., Agrawal, P., Amorim, I., & Prodan, R. (2021). WELFake: Word Embedding Over Linguistic Features for Fake News Detection. IEEE Trans. Comput. Soc. Syst., 8(4), 881–893. doi:10.1109/TCSS.2021.3068519
45. Risdal, M. (2016). Getting Real about Fake News. Kaggle. doi:10.34740/KAGGLE/DSV/911
46. Glenski, M., Sealy, W. I., Miller, K., & Arendt, D. (2021, November). Improving Synonym Recommendation Using Sentence Context. Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing, 74–78. doi:10.18653/v1/2021.sustainlp-1.9
47. Kobayashi, S. (2018, June). Contextual Augmentation: Data Augmentation by Words with Paradigmatic Relations. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), 452–457. doi:10.18653/v1/N18-2072
48. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Dostupné na http://arxiv.org/abs/1810.04805
49. Miller, G. A. (1995). WordNet: A lexical database for English. Commun. ACM, 38(11), 39–41. doi:10.1145/219717.219748
50. Beddiar, D. R., Jahan, M. S., & Oussalah, M. (2021). Data expansion using back translation and paraphrasing for hate speech detection. Online Social Networks and Media, 24, 100153. doi:10.1016/j.osnem.2021.100153

51. Xie, Q., Dai, Z., Hovy, E., Luong, M.-T., & Le, Q. V. (2020, November). Unsupervised Data Augmentation for Consistency Training. Dostupné na http://arxiv.org/abs/1904.12848
52. Haralabopoulos, G., Torres, M. T., Anagnostopoulos, I., & McAuley, D. (2021). Text data augmentations: Permutation, antonyms and negation. Expert Systems with Applications, 177, 114769. doi:10.1016/j.eswa.2021.114769
53. Pellicer, L. F. A. O., Ferreira, T. M., & Costa, A. H. R. (2023). Data augmentation techniques in natural language processing. Applied Soft Computing, 132, 109803. doi:10.1016/j.asoc.2022.109803
54. Wang, W. Y., & Yang, D. (2015, September). That's So Annoying!!!: A Lexical and Frame-Semantic Embedding Based Data Augmentation Approach to Automatic Categorization of Annoying Behaviors using #petpeeve Tweets. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2557–2563. doi:10.18653/v1/D15-1306
55. Wu, X., Lv, S., Zang, L., Han, J., & Hu, S. (2019). Conditional BERT Contextual Augmentation. In J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, … P. M. A. Sloot (Eds.), Computational Science – ICCS 2019 (pp. 84–95). doi:10.1007/978-3-030-22747-0_7

Vybrané kapitoly z dátovej vedy II

František Forgáč, Lívia Kelebercová, Valerii Volodymyrovych Popovych, Kristína Šteflovičová

Vydavateľ: Univerzita Konštantína Filozofa v Nitre

Edícia: Prírodovedec č. 809

Návrh obálky: Kristína Šteflovičová

Vydanie: prvé

Formát: A4

Rozsah: 224 strán

Rok vydania: 2023

ISBN 978-80-558-2020-0