FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z KIV/UIR

Automatická klasifikace dialogových aktů

Dominik Zappe
(A20B0279P)

Květen 2022

Obsah

1 Zadání

Ve zvoleném programovacím jazyce navrhněte a implementujte program, který umožní v komiksovém dialogu klasifikovat věty (nebo jejich části) do tříd podle jejich obsahu, např. rozkaz, otázka zjišťovací (wh–question), odpověď, apod. Tyto věty odpovídají tzv. dialogovým aktům a mají důležitou roli pro řízení dialogu, protože určují funkci věty v dialogu. Například funkce otázky je žádost o nějakou informaci, naproti tomu, funkcí sdělení je poskytnutí požadované informace. Při řešení budou splněny následující podmínky:

- Datová sada viz https://drive.google.com/drive/folders/1ZsEPcSh0MlFU-9iQrib-8eWI1bCv9ygp?usp=sharing obsahuje:
  - Testovací (Test) a trénovací (Train) množiny.
  - Trénovací množinu anotují sami studenti podle anotační příručky.
- Pro trénování implementovaných algoritmů bude NUTNÉ vybrané dokumenty ručně označkovat. Každý student ručně anotuje 40 vybraných komiksů – termín *15. 4. 2022*. Za dodržení termínu obdrží student **bonus 10b**.
- Přiřazení konkrétních komiksů jednotlivým studentům spolu s návodem na anotaci a příklady bude uloženo spolu s daty na výše uvedené adrese.
- Implementujte alespoň tři různé algoritmy (z přednášek i vlastní) pro tvorbu příznaků reprezentující textový dokument.
- Implementujte alespoň dva různé klasifikační algoritmy (klasifikace s učitelem):
  - Naivní Bayesův klasifikátor
  - klasifikátor dle vlastní volby
- Funkčnost programu bude následující:
  - Spuštění s parametry:
    název_klasifikátoru
    soubor_se_seznamem_klasifikačních_tříd,
    trénovací_množina, *testovací_množina*,
    parametrizační_algoritmus, *klasifikační_algoritmus*,
    název_modelu

    Program natrénuje klasifikátor na dané trénovací množině, použije zadaný parametrizační a klasifikační algoritmus, zároveň vyhodnotí úspěšnost klasifikace a natrénovaný model uloží do souboru pro pozdější použití (např. s GUI).

– Spuštění s jedním parametrem:
název_klasifikátoru

název_modelu

Program se spustí s jednoduchým GUI a uloženým klasifikačním modelem. Program umožní klasifikovat věty (dialogové akty) napsané v GUI pomocí klávesnice (resp. překopírované ze schránky).

- Ohodnoťte kvalitu klasifikátoru na dodaných datech, použijte metriku přesnost (accuracy), kde jako správnou klasifikaci uvažujte takovou, kde se klasifikovaná třída nachází mezi anotovanými. Otestujte všechny konfigurace klasifikátorů (tedy celkem 6 výsledků).

  Poznámky:

  – Pro implementaci parametrizačních / klasifikačních algoritmů není možné používat hotové knihovní funkce!

  – Pro vlastní implementaci není potřeba čekat na dokončení anotace. Pro průběžné testování můžete použit testovací korpus (rozdělit na trénovací a testovací množinu).

  – Další informace, např. dokumentace nebo forma odevzdávání jsou k dispozici na CW pod záložkou *Samostatná práce*.

  Bonusové úkoly:

  – Vyzkoušejte již nějakou hotovou implementaci klasifikátoru (scikit–learn, Weka, apod.) a výsledky srovnejte s Vaší implementací (až 10b navíc).

  – Vyzkoušejte shlukování (klasifikaci bez učitele, např. k–means) a výsledky porovnejte s výsledky klasifikace s učitelem (až 10b navíc).

  – Implementujte navíc klasifikační algoritmus založený na neuronové síti typu MLP s využitím knihoven Keras a Tensorflow (až 10b navíc).

  – Vyzkoušejte klasifikaci anglických dokumentů, korpus na vyžádání (až 20b navíc).

2 Analýza problému

Hlavním úkolem semestrální práce je vyřešit problém parametrizace a poté klasifikace. Pod pojmem parametrizace se skrývá převod textových řetězců na číselné vektory. Jednotlivé složky těchto vektorů by měly být příznaky. Tedy parametrizováním se myslí tvorba příznaků a příznakových vektorů. Klasifikace pak pracuje s výše zmíněnými příznakovými vektory a vyhne se tak práci s řetězci.

2.1 Parametrizace

V zadání se hovoří o třech parametrizačních algoritmech pro tvorbu příznaků. Vybrány byly následující algoritmy: **Bag of Words (BoW)**, **Term Frequency (TF)** a **Term Frequency – Inverse Document Frequency (TF–IDF)**.

2.1.1 Bag of Words

Algoritmus **Bag of Words** spočívá v tom, že v paměti je batůžek plný známých slov. Věta se rozloží na jednotlivá slova a vytvoří se vektor nul, který má dimenzi stejnou jako je počet známých slov v batůžku. Pro každé slovo z věty se následovně provede kontrola, jestli se jedná o známé slovo. Pokud ne, nestane se nic a slovo se přeskočí, pokud ano, tak se číslo na jednoznačně určeném indexu vektoru inkrementuje o jedna. Tímto přístupem nezáleží na pořadí slov v původní větě, neboť se kontext slov nijak nebere v potaz.

2.1.2 Term Frequency

Term Frequency je podobný algoritmu **Bag of Words**. Jedná se o vektor, kde každý příznak naznačuje relativní četnost slova ve větě. Tvorba vektoru je totožná s výše vysvětlenou metodou, pouhou změnou je, že se na konci celý vektor vydělí počtem slov ve větě.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

$f_{t,d}$ značí počet kolikrát se slovo t vyskytuje ve větě d. Jmenoval je součet všech slov ve větě d.

2.1.3 Term Frequency – Inverse Document Frequency

TF–IDF algoritmus se skládá ze dvou částí – **TF** a **IDF**. **Term frequency** již bylo vysvětleno výše. **Inverse document frequency** je míra toho, jak moc je dané slovo informačně přínosné – tedy jestli se slovo vyskytuje ve větách často, nebo jestli je vzácné. Vypočítá se jako logaritmus z podílu dvou čísel, kde v čitateli je počet všech vět a ve jmenovateli je počet vět obsahující sledované slovo. Výsledný vektor příznaků je spočítán jako jednoduchý součin příznaků **TF** a vypočítaných odpovídajících **IDF**.

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in D\}|}$$

N značí počet všech vět. Jmenoval značí počet vět, ve kterých se vyskytuje slovo t.

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

2.2 Klasifikace

Dle zadání je vyžadována implementace **Naivního Bayesova klasifikátoru**. Druhý klasifikátor má být dle vlastní volby, byl tedy zvolen **klasifikátor K–nejbližších sousedů**.

2.2.1 Naivní Bayes

Bayesův klasifikátor je založený na **Bayesově větě**.

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{p(\mathbf{x})}$$

Pomocí tohoto vztahu je možné převést posteriorní pravděpodobnost třídy za podmínky, že je na vstupu klasifikátoru prvek $\mathbf{x}$, na součin jednodušeji získatelných pravděpodobností – apriorní pravděpodobnosti třídy, což je poměr četnosti výskytu třídy vůči všem výskytům, a pravděpodobnosti $\mathbf{x}$ za předpokladu, že patří do dané třídy. Pravděpodobnost prvku $\mathbf{x}$ se může zanedbat, neboť je pro všechny třídy stejná.

***Naivní* Bayesův klasifikátor** se nazývá naivní, neboť pro jeho funkčnost se předpokládá podmíněná nezávislost jednotlivých složek vektoru $\mathbf{x}$ při platnosti hypotézy C_k.

Prvek $\mathbf{x}$ představuje vektor příznaků, vzoreček lze tedy přepsat pro jednotlivé složky do podoby:

$$p(C_k|\mathbf{x}) = p(C_k) \prod_{i=0}^{N} p(x_i|C_k)$$

Po napočítání všech pravděpodobností $p(C_k|\mathbf{x})$ se vybere taková třída C_k, pro kterou platí, že její vypočítaná pravděpodobnost je maximální, tedy je nejpravděpodobnější, že prvek **x** patří do dané třídy.

2.2.2 K–nejblížších sousedů

Nejbližší soused je jedním z nejjednodušších přístupů u mnoha algoritmů. Při klasifikaci příznakových vektorů se pouze spočítá vzdálenost zadaného vektoru ke všem známým vektorům. Vybere se minimální vzdálenost, tedy nejbližší soused, a nově zadaný vektor pouze převezme třídu jeho nejbližšího souseda.

Klasifikátor **K–nejbližších sousedů** neuvažuje pouze jednoho nejbližšího souseda, ale bere v potaz **K** sousedů. Tedy z napočítaných vzdáleností se vybírá **K** minim. Z principu by **K** mělo být liché, neboť u sudých čísel by mohlo docházet k nerozhodnutelným situacím.

Výše byla zmíněná vzdálenost vektorů, ta se dá spočítat různými metrikami. Nejznámější je **Euklidovská vzdálenost**, ta uvažuje na vstupu dva vektory stejné dimenze. Funguje následovně – vypočítá se odmocnina ze součtu kvadrátů rozdílů jednotlivých složek vektorů.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Jinou možností může být např. **Kosinová podobnost**. Spočítá se následovně:

$$d(p,q) = \frac{p \cdot q}{||p|| \cdot ||q||} = \frac{\sum_{i=1}^{n} p_i \times q_i}{\sqrt{\sum_{i=1}^{n}(p_i)^2} \times \sqrt{\sum_{i=1}^{n}(q_i)^2}}$$

3 Návrh a popis řešení

Celá aplikace je napsána v Pythonu. Obsahuje pět spustitelných souborů: **main.py**, **scikitlearn.py**, **kmeans.py**, **mlp.py** a **conllparser.py**.

3.1 Tvorba příznaků

Základním objektem je obalovací třída **Bag**. Ta slouží jako přepravka slovníku se slovy, ke kterým je přiřazený index. Ten odpovídá pozici, kde má být příznak ve vektoru. Dále tato třída disponuje jednoduchou funkcí na přidání slova do batohu.

Tvorba příznakového vektoru pomocí **Bag of Words** je tak velmi jednoduchá. Vstupní věta se rozdělí na jednotlivá slova, u každého slova je zkontrolován jeho výskyt v batohu. Pokud se jedná o známé slovo, tak v batohu na klíči slova se nachází index do vektoru, kam se má přičíst jednička.

Vytvoření příznakového vektoru uživajícího **Term Frequency** je prakticky identické s výše popsanou metodou pomocí **Bag of Words**. Pouhou změnou je, že se neinkrementuje číslo ve vektoru o jedna, ale o relativní číslo vůči počtu slov ve větě.

TF–IDF metoda je nejsložitější, ale polovinu získá z výše popsané **TF** metody. Je zapotřebí získat **IDF** hodnoty. Jsou vypočítány výše zmíněným vzorečkem, pouhá změna ve vzorečku je přičtení jedničky k čitateli i jmenovateli, aby se tak zamezilo jak dělení nulou, tak výpočtu logaritmu nuly. Obě operace jsou nedefinované a vedou k chybě. Po napočítání **TF** i **IDF** už zbývá jen udělat součin.

3.2 Klasifikátory

Byla vytvořena třída **NaiveBayes**, která má dvě základní funkce – natrénování a klasifikaci.

Trénovací funkce si nejdříve napočte apriorní pravděpodobnosti jednotlivých tříd. Nasčítá se četnost jednotlivých tříd, ta se pak vydělí celkovým počtem vět v trénovacích datech. Podmíněná pravděpodobnost pro každou třídu je vypočítaná tak, že se nasčítají všechny vektory, které mají spadat do dané třídy. Tento vektor součtů jednotlivých složek je nakonec vydělen součtem všech složek. K čitateli je navíc přičtena jednička a ke jmenovateli počet všech slov v batohu – **Laplaceovo vyhlazování**. Tím je model natrénovaný a připraven klasifikovat nově příchozí vektory.

Posteriorní pravděpodobnost vektoru ze vstupu se spočítá pomocí výše vysvětlených vzorců jako součin apriorní a podmíněných pravděpodobností. Apriorní pravděpodobnost bohužel vykazuje špatné výsledky pro daná data, neboť přibližně 50 % všech trénovacích dat spadá do jedné třídy – **INFORM**. Pro lepší výsledky klasifikátoru tedy není apriorní pravděpodobnost užívána. Pro ulehčení výpočtu navíc nejsou počítané součiny pravděpodobností, ale součty logaritmů pravděpodobností, převedl se tak problém součinu na problém součtu, což je pro počítače podstatně rychlejší. Z napočtených pravděpodobností se zvolí maximum a je tak klasifikováno do nejpravděpodobnější třídy.

Dále byla vytvořena třída **KNN**, která mimo dvě základní funkce, natrénování a klasifikaci, má ještě další dvě své statické funkce – napočítání **Euklidovské** a **Kosinové** vzdálenosti.

Trénování klasifikátoru **KNN** prakticky neexistuje, neboť natrénování vlastně spočívá v uložení si všech trénovacích vektorů do paměti.

Klasifikace napočítá vzdálenost vektoru ze vstupu s každým vektorem z trénovací množiny. Vybere se **K** minim a z nich se vybere nejpočetnější třída.

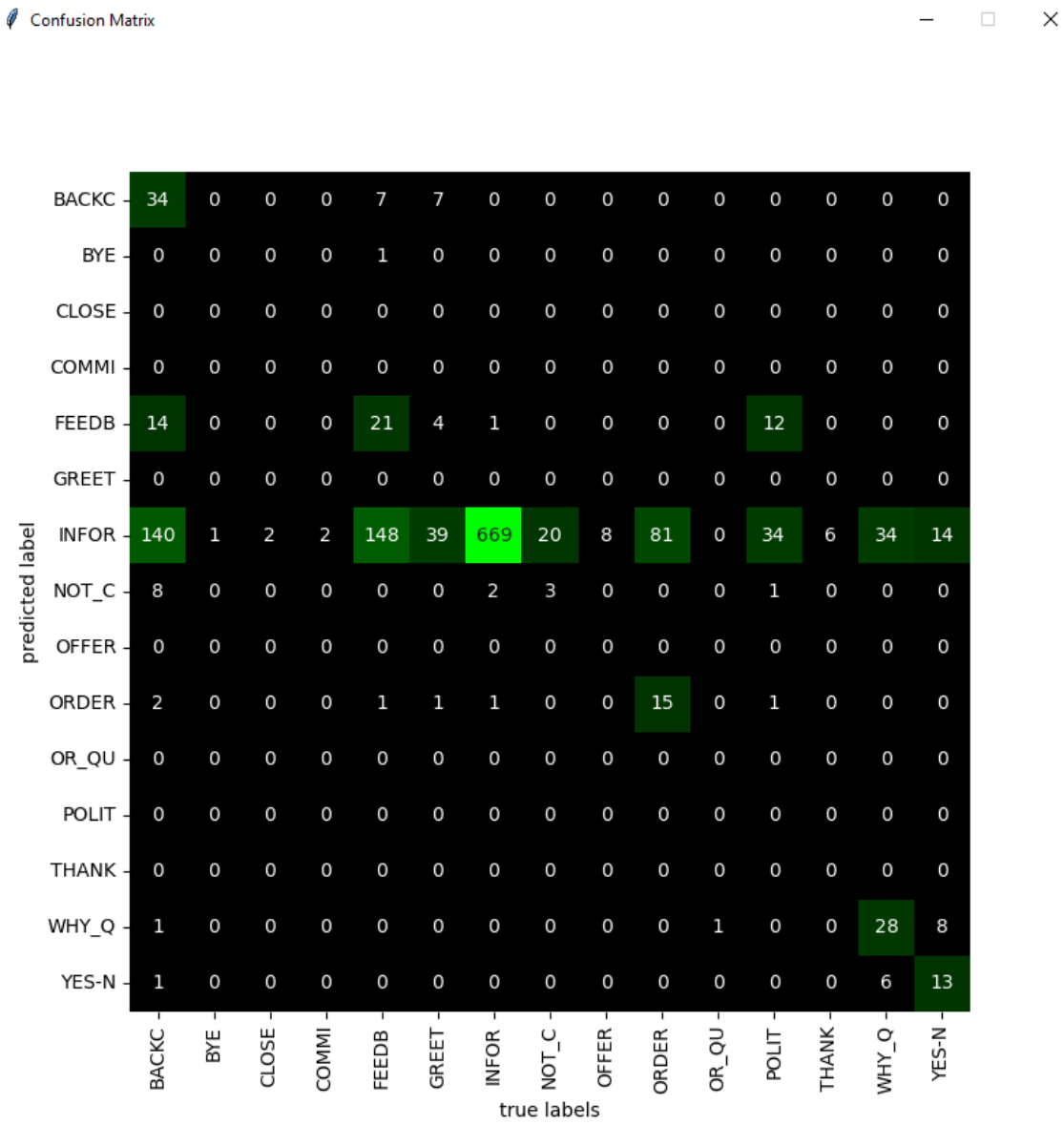
Klasifikátory byly zpracovány objektovým přístupem hlavně kvůli nutnosti ukládání a načítání natrénovaných modelů. Tato funkčnost je zajištěna knihovnou **pickle**. Uložení probíhá jednoduše, vytvoří se soubor s příponou .pickle, který obsahuje natrénovaný objekt klasifikátoru. Načtení modelu je stejné, jen se soubor nevytváří, ale čte.

3.3 GUI

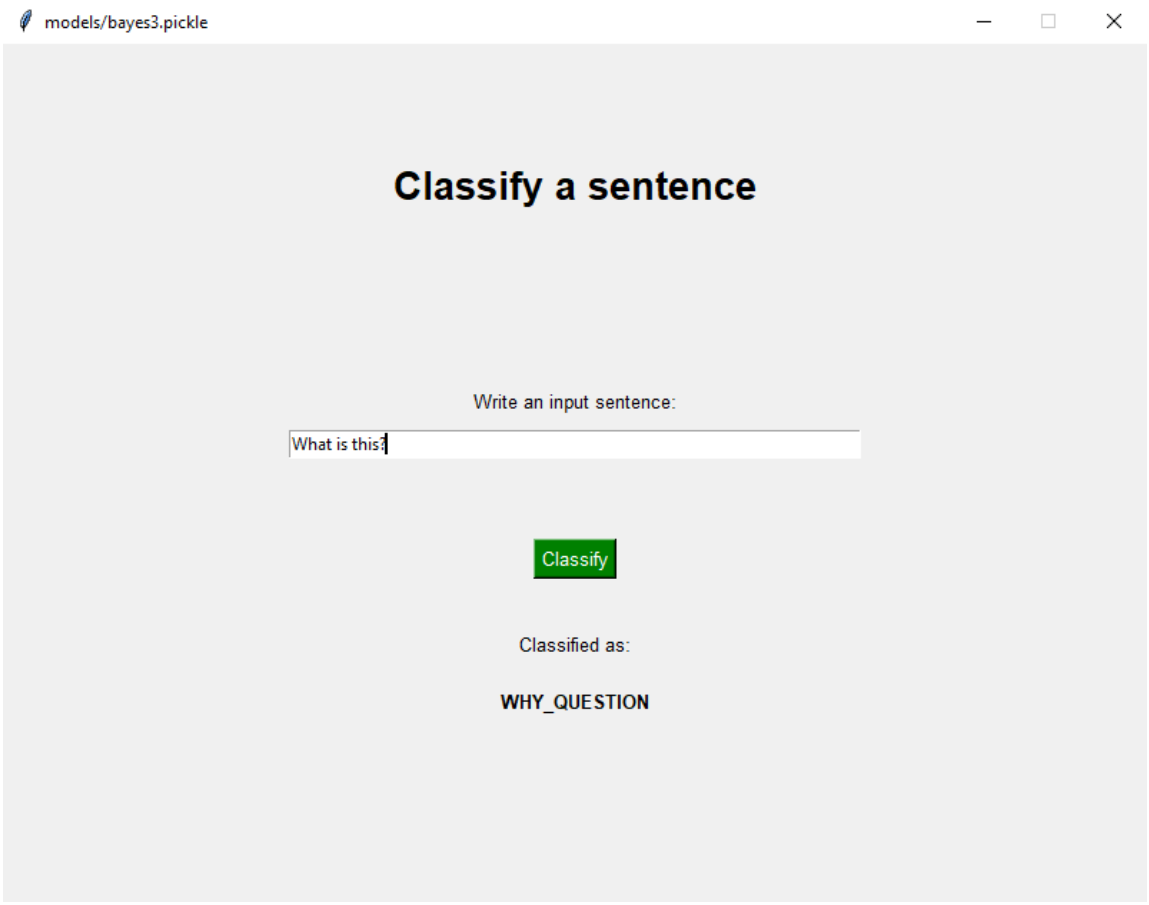
Grafické uživatelské rozhraní si řeší každá spustitelná část aplikace samostatně. Na veškerou vizualizaci byla využita knihovna **Tkinter**.

Při trénování modelu se po natrénování zobrazí okno s obrázkem **matice záměn** (viz Obr. 1). Na této matici je dobře vidět, co si klasifikátor s čím plete a co mu naopak jde klasifikovat dobře bez chyb. Jak již bylo zmíněno, vstupní data obsahují nejvíce vět třídy *INFORM* a ostatní třídy jsou méně zastoupené – lze tedy předpokládat, že obecně v matici bude řádek třídy *INFORM* více vyplněný.

Při spouštění natrénovaného modelu se záměrem klasifikovat se ve vizualizačním okně zobrazí textové vstupní pole, do kterého je možné psát. Po odeslání požadavku na klasifikaci je věta převedena na příznakový vektor. Tento vektor je klasifikátorem zpracován a zpět do okna je vypsán výstup – klasifikovaná třída (viz Obr. 2).



Obr. 1: Příklad matice záměn (jedná se o Naivního Bayese s příznaky TF–IDF)



Obr. 2: Příklad grafického uživatelského rozhraní (Naivní Bayesův klasifikátor)

3.4 Bonusové úkoly

Prvním bonusovým úkolem je vyzkoušení již hotové implementace. Byla zvolena *scikit–learn* knihovna. Práce s ní byla velmi jednoduchá. Z nabízených **Naivních Bayesovských klasifikátorů** byl zvolen **MultinomialNB**. Pro tvorbu příznaků bylo užito objektů **CountVectorizer** a **TfidfTransformer**. Načtené věty byly převedeny na přiznakové vektory zapomocí funkcí **fit_transform()**. Dále nad objektem klasifikátoru byly volány funkce **fit()** a **predict()**. Tento klasifikátor má horší *přesnost*, než vlastní implementace – zřejmě kvůli apriorní pravděpodobnosti třídy a kvůli povaze trénovacích dat.

Druhým bonusovým úkolem je vyzkoušení shlukování. Byl proto zvolen algoritmus **K–means**. Klasifikátor na principu **K–means** vlastně funguje podobně jako klasifikátor **KNN**. Rozdíl je v tom, že při trénování **K–means** klasifikátoru se spočítají **středy** tříd a za klasifikovanou třídu je považována ta nejbližší – ve smyslu nejbližšího souseda ze středů tříd. Jsou implementovány dvě možné vzdálenosti – **Euklidovská** a **Kosinová** – je možné si mezi nimi přepínat. Díky šikovné volbě počátečních středů je zpětně možné podle čísla shluku určit třídu. Šikovnou volbou je myšleno, že střed každé třídy je náhodně zvolená věta odpovídající dané třídě.

Třetím bonusovým úkolem je klasifikační algoritmus založený na neuronové síti typu **MLP**. Byla využita knihovna **Tensorflow**, konkrétněji **Keras**. Je nutné předzpracovat data do formátu, který tato knihovna požaduje na vstupech. Po předzpracování byl vytvořen model se čtyřmi vrstvami – první vstupní, dvě skryté a poslední výstupní. Model lze parametrizovat z příkazové řádky – měnitelné parametry jsou: *počty neuronů ve skrytých vrstvách*, *učící konstanta*, *počet epoch* a *velikost balíčku*.

Posledním bonusovým úkolem je vyzkoušení klasifikace anglických dokumentů. Na vyžádání byla obdržena trénovací i testovací data ve formátu **.conll**. Kvůli tomuto formátu byl vytvořen pátý spustitelný soubor – *parser*. Po převedení dat do požadovaného formátu je možné data poslat do jakéhokoliv klasifikátoru. Při vlastní implementaci však nebylo počítáno s možnou nadměrnou velikostí dat, a tak není možné provést klasifikaci na vlastní implementaci. Scikit implementace si s velkými daty hravě poradí a klasifikace funguje. Data jsou opět zanesena velkým množstvím jedné třídy, takže se opět nedosahuje moc velké přesnosti – konkrétně 55.189 %.

4 Zhodnocení klasfikátorů

Po natrénování každého z klasifikátorů se do konzole vypíše **přesnost** klasifikátoru v procentech a zobrazí se **matice záměn** v samostatném okně.

Přesnost je počítána jednoduchým poměrem správně klasifikovaných vůči všem testovaným vektorům.

Matice záměn je importnována z *Scikit-learn* balíčku. Navíc je matice upravena, aby ořezávala popisky os, neboť nějaké byly příliš dlouhé. Každý popisek má nejvíce pět písmen. Dále byla matice upravena, aby byla zelená a měla "logaritmickou"škálu barevnosti. Vše je vidět na Obr. 1.

4.1 Základní klasifikátory

Následující tabulka 1 zobrazuje přesnosti pro kombinace klasifikátorů a příznaků ze základního zadání na standardních datech ze zadání.

Tabulka 1: Přesnosti kombinací klasifikačních a parametrizačních algoritmů

	Bag of Words	TF	TF–IDF
Naivní Bayes	56.106 %	53.520 %	56.250 %
5–Nejbližších Sousedů (Euklid)	48.348 %	1.724 %	10.991 %
5–Nejbližších Sousedů (Kosinus)	52.083 %	51.868 %	45.331 %

Z tabulky 1 je možné usoudit, že **Naivní Bayesův** klasifikátor si vede obecně lépe než **KNN** klasifikátor. Nejpřesnější kombinací na daných datech se jeví *Naivní Bayes* s příznakovou metodou *TF-IDF*.

Dále lze z tabulky 1 vypozorvat neúspěšnost **Euklidovské vzdálenosti** vůči **Kosinové podobnosti**. Pro vícedimenzionální vektory je obecně *Euklidovská vzdálenost* méně přesná s rostoucími dimenzemi, obzvláště jedná-li se o "řídké"vektory – vektory obsahující převážně nuly. Na takovýchto vektorech by obecně měla *Kosinová podobnost* fungovat lépe, u *Euklidovské vzdálenosti* roste chyba s kvadrátem rozdílu, kde jeden člen tohoto rozdílu je nulový.

4.2 Bonusové úkoly

Následující tabulka 2 zobrazuje přesnosti klasifikátorů z bonusových úkolů na standardních datech ze zadání. Tedy **scikit–learn** implementaci, **K–means** shlukování a **neuronovou síť**.

Tabulka 2: Přesnosti klasifikátorů z bonusových úkolů

	Přesnost
Scikit–learn Naivní Bayes	49.066 %
Scikit–learn K–means – BoW	5 ± 1 %
Scikit–learn K–means – TF	5 ± 1 %
Scikit–learn K–means – TF–IDF	5 ± 1 %
Tensorflow, Keras – BoW	52 ± 2 %
Tensorflow, Keras – TF	51 ± 2 %
Tensorflow, Keras – TF–IDF	49 ± 2 %

Jak je z tabulky 2 vidět, *Scikit–learn* implementace naivního Bayesova klasifikátoru dosahuje horších výsledků než vlastní implementace. Nejspíše to bude způsobené tím, že ve vlastní implementaci se nepočítá s apriorní pravděpodobností třídy, neboť je známa zanesenost dat třídou **INFORM**. Když se do výpočtu připočítá apriorní pravděpodobnost, klasifikátor pak má sklony klasifikovat více vět do třídy **INFORM**, samozřejmě nesprávně.

Dále je z tabulky 2 zřejmé, že *K–means* si nevede dobře s žádnými příznakovými vektory. Je to opět dáno povahou dat – obsahují nadměrné množství vět třídy **INFORM**, které mají velký rozptyl – jakožto vektory v prostoru. *K–means* se snaží vytvářet kompaktní shluky, a tak se věty, které mají být ve třídě **INFORM**, dostanou kamkoliv jinam díky jejich různorodosti. Navíc si lze v tabulce 2 všimnout, že hodnoty přesnosti mají rozptyl. Přesnost je totiž ovlivněna počáteční volbou středů a maximálním počtem iterací.

Nakonec lze z tabulky 2 vypozorovat i přesnost neuronové sítě. Ta je přibližně stejná, jako u běžných klasifikátorů, což je dle očekávání vzhledem k povaze dat. Obdobně jako u *K–means* přesnost je uvedena s rozptylem, neboť u neuronové sítě závisí na všech vstupních parametrech. Průměrně se s dobrými parametry dosahuje hodnot přibližně totožným těm z tabulky 2.

5 Uživatelská příručka

Aplikace obsahuje celkem *pět* spustitelných souborů s příponou **.py** – **main.py**, **scikitlearn.py**, **kmeans.py**, **mlp.py** a **conllparser.py**. Každou z těchto aplikací lze spustit z příkazové řádky pomocí příkazu

python ⟨nazev_souboru.py⟩

Když jsou soubory spuštěny takto bez argumentů příkazové řádky, každý z programů vypíše svůj seznam požadovaných argumentů – viz. obrázky 3, 4, 5, 6 a 7.

```
C:\Users\SpeekeR\Documents\zzz>python main.py

Invalid number of parameters!

Expected use case 1 (training model): main.py <list_of_classes.txt> <training_data.txt> <test_data.txt> <featur
es_algorithm> <classifier_algorithm> <model_name>
<list_of_classes.txt> - file with list of classes
<training_data.txt> - file with training data
<test_data.txt> - file with test data
<features_algorithm> - algorithm for features extraction (1, 2 or 3) (1 = bag of words, 2 = tf, 3 = tf-idf)
<classifier_algorithm> - algorithm for classifier (1 or 2) (1 = Naive Bayes, 2 = KNN)
<model_name> - name of the model (without extension)

Expected use case 2 (using model): main.py <model_name>
<model_name> - name of the model (without extension)
```

Obr. 3: Spuštění *main.py* bez parametrů

```
C:\Users\SpeekeR\Documents\zzz>python scikitlearn.py

Invalid number of parameters!

Expected use case 1 (training model): scikitlearn.py <training_data.txt> <test_data.txt> <model_name>
<training_data.txt> - file with training data
<test_data.txt> - file with test data
<model_name> - name of the model (without extension)

Expected use case 2 (using model): scikitlearn.py <model_name>
<model_name> - name of the model (without extension)
```

Obr. 4: Spuštění *scikitlearn.py* bez parametrů

```
C:\Users\SpeekeR\Documents\zzz>python kmeans.py

Invalid number of parameters!

Expected use case 1 (training model): kmeans.py <list_of_classes.txt> <training_data.txt> <test_data.txt> <feat
ures_algorithm> <max_iter> <model_name>
<list_of_classes.txt> - file with list of classes
<training_data.txt> - file with training data
<test_data.txt> - file with test data
<features_algorithm> - algorithm for features extraction (1, 2 or 3) (1 = bag of words, 2 = tf, 3 = tf-idf)
<max_iter> - maximum number of iterations for K-means algorithm
<model_name> - name of the model (without extension)

Expected use case 2 (using model): kmeans.py <model_name>
<model_name> - name of the model (without extension)
```

Obr. 5: Spuštění *kmeans.py* bez parametrů

```
C:\Users\SpeekeR\Documents\zzz>python mlp.py

Invalid number of parameters!

Expected use case 1 (training model): mlp.py <list_of_classes.txt> <training_data.txt> <test_data.txt> <feature
s_algorithm> <hidden_units_1> <hidden_units_2> <number_of_epochs> <batch_size> <learning_rate> <model_name>
<list_of_classes.txt> - file with list of classes
<training_data.txt> - file with training data
<test_data.txt> - file with test data
<features_algorithm> - algorithm for features extraction (1, 2 or 3) (1 = bag of words, 2 = tf, 3 = tf-idf)
<hidden_units_1> - number of hidden units in the first hidden layer
<hidden_units_2> - number of hidden units in the second hidden layer
<number_of_epochs> - number of epochs
<batch_size> - batch size
<learning_rate> - learning rate
<model_name> - name of the model (without extension)

Expected use case 2 (using model): mlp.py <model_name>
<model_name> - name of the model (without extension)
```

Obr. 6: Spuštění *mlp.py* bez parametrů

```
C:\Users\SpeekeR\Documents\zzz>python conllparser.py

Invalid number of parameters!

Expected use case 1 (training data): conllparser.py <input.conll> <output.txt> <classes.txt>
<input.conll> - path to input file in conll format
<output.txt> - path to output file in txt format
<classes.txt> - path to output file with list of classes

Expected use case 2 (test data): conllparser.py <input.conll> <output.txt>
<input.conll> - path to input file in conll format
<output.txt> - path to output file in txt format
```

Obr. 7: Spuštění *conllparser.py* bez parametrů

Jak je na obrázcích 3 až 7 vidět, parametry jsou anglicky detailně popsané přímo v příkazové řádce.

Po správném zadání parametrů by měl proběhnout jeden ze dvou scénářů:

1. **Trénování** modelu

   Dojde k natrénování modelu na dodaných trénovacích datech. Dále se v konzoli vypíše anglicky hláška o *přesnosti* daného modelu v procentech – měřeno na dodaných testovacích datech. Spolu s výpisem se otevře samostatné okno obsahující obrázek *matice záměn* (viz Obr. 1).

2. **Klasifikace** pomocí modelu

   Otevře se samostatné okno obsahující textové pole a tlačítko (viz Obr. 2). Do textového pole se očekává zapsání věty, která má být klasifikována. Po stisknutí tlačítka (nebo klávesy *Enter*) se odešle požadavek na klasifikaci a vybraný předtrénovaný model klasifikuje větu. Výsledek klasifikace – klasifikovaná třída – se zobrazí v grafickém uživatelském rozhraní – v okně.

6 Závěr

Úspěšnost všech klasifikátorů se pohybuje okolo 50 %, což je způsobeno povahou dat, ve kterých je 15 tříd, ale zastoupení je okolo 55 % ve prospěch jedné třídy – *INFORM*.

Navzdory své jednoduchosti nejlepší přesnosti dosahuje **Naivní Bayesův** klasifikátor s příznakovými metodami **Bag of Words** nebo **TF–IDF**. **Naivní Bayes** vyšel nejlépe díky tomu, že nebyly brány v potaz apriorní pravděpodobnosti tříd.

Případné vylepšení práce by mohlo spočívat v paměťové úspoře, neboť vlastní implementace si nebyla schopná poradit s dodaným korpusem anglických dokumentů.

Reference

1. Král, P. (n.d.). *Index of / pkral/UIR.* Retrieved May 3, 2022, dostupné na: https://home.zcu.cz/ pkral/uir/

2. Great Learning Team. (2022, March 22). *An introduction to bag of words (bow): What is bag of words?* GreatLearning Blog: Free Resources what Matters to shape your Career! Retrieved May 3, 2022, dostupné na: https://www.mygreatlearning.com/blog/bag-of-words/

3. *A friendly guide to NLP: Bag-of-words with python example.* Analytics Vidhya. (2021, August 19). Retrieved May 3, 2022, dostupné na: https://www.analyticsvidhya.com/blog/2021/08/a-friendly-guide-to-nlp-bag-of-words-with-python-example/

4. *Understanding TF-IDF for Machine Learning.* Capital One. (n.d.). Retrieved May 3, 2022, dostupné na: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

5. Machine Learning & My Music. (2019, September 10). *Naive Bayes: Text Classification Example* [Video]. YouTube. dostupné na: https://youtu.be/mqYa0LaA9WI