



Semestrální práce KIV/UIR

Klasifikace dokumentů

Pavel Třeštík
A17B0380P

21. května 2020

Obsah

1	Zadání	1
2	Analýza úlohy	3
2.1	Algoritmy pro tvorbu příznaků	3
2.1.1	Bag of Words	3
2.1.2	Bigram	4
2.1.3	TF-IDF	5
2.2	Klasifikační algoritmy	5
2.2.1	Naive Bayes	5
2.2.2	k-nearest neighbors	6
3	Implementace	7
3.1	Bag of Words	7
3.2	Bigram	7
3.3	TD-IDF	7
3.4	Naive Bayes	7
3.5	k-nearest neighbors	8
4	Uživatelská dokumentace	8
4.1	Konzolový mód	8
4.2	GUI mód	9
5	Shrnutí výsledků	9
6	Závěr	9

1 Zadání

Ve zvoleném programovacím jazyce navrhnete a implementujete program, který umožní klasifikovat textové dokumenty do tříd podle jejich obsahu, např. počasí, sport, politika, apod. Při řešení budou splněny následující podmínky:

- Použijte data z českého historického periodika Posel od Čerchova“, která jsou k dispozici na <https://drive.google.com/drive/folders/1mQbBNS43gWFRMHDYdSkQug47cuhPTsHJ?usp=sharing>. V původní podobě jsou data k dispozici na <http://www.portafontium.eu/periodical/posel-od-cerchova-1872?language=cs>.
- Pro vyhodnocení přesnosti implementovaných algoritmů bude NUTNÉ vybrané dokumenty ručně označovat. Každý student ručně anotuje 10 stran zadaného textu – termín 31.3.2020. Za dodržení termínu obdrží student bonus 10b.
- Přiřazení konkrétních textů jednotlivým studentům spolu s návodem na anotaci a příklady je uloženo spolu s daty na výše uvedené adrese, konkrétně:
 - 0 - vzorová složka (takhle by měl výsledek vypadat)
 - 1, 2, .. , 15, 101, 102, .. - data k anotaci
 - přiřazení souboru studentum.xlsx - určení, jaké soubory má jaký student anotovat. Až budete mít anotaci hotovou, doplňte sem informaci.
 - Anotační příručka - návod, jak články anotovat.
 - Klasifikace dokumentů - kategorie.xlsx - seznam kategorií k anotaci s příklady.
 - sem prace20.pdf - Zadání semestrální práce
- 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 dokumenty 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 vlastní implementaci není potřeba čekat na dokončení anotace. Pro průběžné testování můžete použít korpus současné češtiny, který je k dispozici na <http://ctdc.kiv.zcu.cz/> (uvažujte pouze první třídu dokumentu podle názvu, tedy např. dokument 05857 zdr ptr eur.txt náleží do třídy zdr – zdravotnictví). ”
- další informace, např. dokumentace nebo forma odevzdávání jsou k dispozici na CW pod záložkou Samostatná práce.

2 Analýza úlohy

2.1 Algoritmy pro tvorbu příznaků

Existuje řada metod získávání informací (modelů) z rozdílných typů dat. Pro tuto práci nás ale zajímají pouze jazykové pravděpodobnostní modely. Ty se řadí do několika tříd: unigram, n-gram, exponenciální, neuronové sítě a ostatní (takové, které nepřípadají do ani jedné z přechozích skupin). Ve skutečnosti ale některé třídy jsou založeny na jiné třídě. Na příklad unigram je ve skutečnosti n-gram pro jednotlivá slova. Mezi nejvýznačnější reprezentanty patří Bag of Words, Bigram, Trigram a Word2vec.

2.1.1 Bag of Words

Prvním z algoritmů, které jsem vybral je Bag of Words. Jedná se o unigram, tudíž každý text/ dokument je reprezentován jednotlivými slovy. Slova jsou ukládána do slovníku a jednotlivé klasifikační třídy/ dokumenty jsou reprezentovány vektory o délce slovníku. Hodnoty vektorů jsou reprezentací četností výskytu slov v textu/ dokumentu pro každou klasifikační třídu (nebo jednotlivé dokumenty). Toto ovšem vytváří problém, kdy při velké množině slov ve slovníku, se jich pouze malá část vyskytuje ve vektoru reprezentující klasifikační třídu/ dokument.

Příklad Bag of Words

Zdroj (Wikipedia): [url](#)

Texty

(1) John likes to watch movies. Mary likes movies too.

(2) Mary also likes to watch football games.

Rozdělení textů

BoW1 = {"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1};

BoW2 = {"Mary":1,"also":1,"likes":1,"to":1,"watch":1,"football":1,"games":1};

Slovník textů

BoW3 = {"John":1,"likes":3,"to":2,"watch":2,"movies":2,"Mary":2,"too":1,"also":1,"

Vektory reprezentující texty

(1) [1, 2, 1, 1, 2, 1, 1, 0, 0, 0]

(2) [0, 1, 1, 1, 0, 1, 0, 1, 1, 1]

2.1.2 Bigram

Druhým algoritmem pro tvorbu příznaků bude **Bigram**. Ačkoliv **Bag of Words** je ve skutečnosti **n-gram** pro $n = 1$ a **Bigram** je **n-gram** pro $n = 2$ je v těchto algoritmech značný rozdíl. **Bigram** páruje slova a počítá pravděpodobnost páru jako podmíněnou pravděpodobnost obou slov páru.

Párování slov je ukázáno na následujícím příkladu Ukázka 1 Pravděpodobnost výrazu je spočítána podle Vztahu 2.

Věta

Tato věta je příklad.

Bigramy

bigrams = {"S, Tato", "Tato, věta", "věta, je", "je, příklad",
"příklad, /S"}

S = začátek věty /S = konec věty

Obrázek 1: Příklad párování slov

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

Obrázek 2: Pravděpodobnost výrazu bigramu
Zdroj: <https://en.wikipedia.org/wiki/Bigram>

2.1.3 TF-IDF

Posledním vybraným algoritmem je TF-IDF. Na rozdíl od předchozích dvou vybraných algoritmů nevytváří příznaky z textu, ale upravuje již existující příznaky. Tudiž je využíván spolu s **Bag of Words**. Algoritmus ohodnotí jednotlivá slova váhou místo jejich četností. Čím častěji se slovo v textech/dokumentech vyskytuje, tím nižší váhu má po TF-IDF. Tento algoritmus svým způsobem "vytváří" takzvané **stop words**. To jsou slova, které v jazyce pro klasifikaci nemají žádný význam (na příklad pro Český jazyk slova jako: a, pro, nebo...). Přidat url na příklad a jak počítat??

2.2 Klasifikační algoritmy

Výběr klasifikačního algoritmu značně závisí na účelu našeho projektu. Z mnoha klasifikátorů jako je **Naive Bayes**, **Support Vector Machine (SVM)**, **Lineární regrese**, **Neural Networks** a dalších, potřebujeme vybrat podle potřeb. Nejdůležitějšími vlastnostmi klasifikátoru při výběru jsou jeho přesnost (accuracy), složitost a rychlost.

2.2.1 Naive Bayes

Je jeden z nejrychlejších a nejjednodušších klasifikátorů. Jeho nedostatkem je, že potřebuje poměrně přesná trénovací data, aby jeho klasifikace byly co možná nejpřesnější.

Klasifikátor vybere nejpravděpodobnější třídu podle následujícího vztahu (viz Obrázek 3).

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} p(C_k) \prod_{i=1}^n p(x_i | C_k).$$

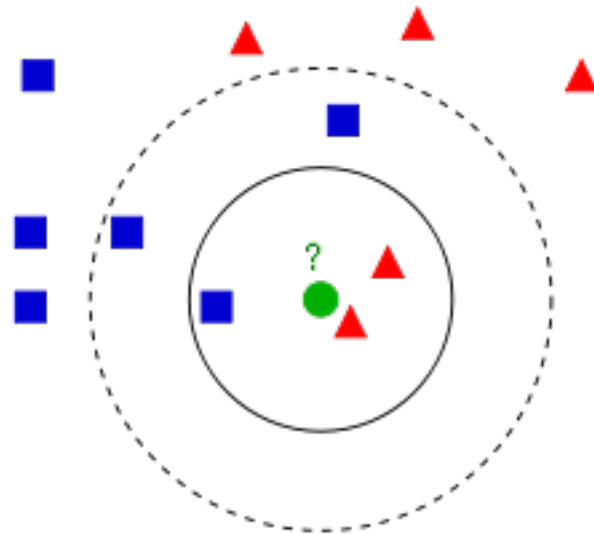
Obrázek 3: Rovnice vybrání třídy

Zdroj: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

2.2.2 k-nearest neighbors

Převede klasifikovaný text/ dokument na vektor a spočítá vzdálenost ke každému trénovacímu textu/ dokumentu. Poté přiřadí klasifikovanému textu třídu, která má nejvíce zástupců v určité blízkosti. Tento algoritmus je velmi jednoduchý a poměrně rychlý. Nevýhodou je přesnost, která pokud trénovací soubory nemají jednoznačně určenou třídu, není velmi vysoká.

Příklad klasifikace zelené tečky na Obrázku 4. Zelená tečka bude klasifikována jako červený trojúhelník, protože v okolí jsou 2 červené trojúhelníky a pouze 1 modrý čtverec.



Obrázek 4: Příklad klasifikace knn

Zdroj:

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

3 Implementace

Pro tento projekt jsem zvolil jazyk Python. Důvodem bylo, že pro umělou inteligenci je pro Python množství knihoven. Naneštěstí jsem se k využití žádné z knihoven nedostal.

3.1 Bag of Words

Původně jsem tento model implementoval použitím teoretického způsobu. Neboli vytvořil jsem slovník a podle toho reprezentoval třídy jako vektory. Toto ale trvalo přibližně 15 vteřin.

Nakonec jsem začal jednotlivé třídy reprezentovat pomocí Python struktury **dictionary**. Tato struktura proces vytváření vektorů tříd urychlila na méně než vteřinu, ale není třeba dodržovat původní model a slovník není vůbec potřeba. Výhoda absence slovníku je, že každá třída nemá značné množství nul za slova, která se ve třídě nevyskytují.

3.2 Bigram

Také reprezentován pomocí struktury **dictionary**. Vytvářen velmi podobně jako Bag of Words, ale klíče slovníku jsou 2 slova oddělená mezerou.

Má změna oproti teoretickému způsobu algoritmu je v počítání pravděpodobnosti. Stejně jako u Bag of Words, jsou počítány pouze četnosti jednotlivých bigramů v textu. Pravděpodobnost je potom v klasifikátoru počítána jako $\text{počet četností} / \text{počet bigramů ve třídě}$, což není správný způsob počítání pravděpodobnosti bigramu.

3.3 TD-IDF

Provádí výpočty přesně jak má. Implementován tak, aby počítal nad strukturou **dictionary**.

3.4 Naive Bayes

Také pracuje nad strukturou **dictionary**. Z důvodu velkého počtu slov ve třídách, čímž vznikaly pravděpodobnosti s příliš mnoha desetinnými místy pracuje v logaritmovém prostoru. Místo násobení pravděpodobností, sčítá logaritmy pravděpodobností. Také používá tak zvaný **Laplace smoothing**,

díky čemuž i slova, která se ve třídě nevyskytují mají malý vliv na klasifikaci místo nulového.

3.5 k-nearest neighbors

Pro tento algoritmus bylo třeba upravit příznakové metody, aby místo s jedním slovníkem definující třídu, vytvářely list souborů definující danou třídu.

Díky této úpravě může algoritmus fungovat přesně jak je popsán v analýze úlohy.

4 Uživatelská dokumentace

Program se spouští z konzole. Program je spustitelný na verzi Python 2.x. Aby se rozlišilo jestli se program spouští v příkazové řádce nebo v GUI má 2 rozdílné zpracování argumentů. Proto je nutno specifikovat, který použít formou argumentu.

4.1 Konzolový mód

Program se spustí následujícím způsobem: **python main.py Console [-h] classes_name_file train_files test_files irs classifier model_name**

- **Console** – argument specifikující, které zpracování argumentů použít.
- **[-h]** – dobrovolný argument, při jehož použití se vypíše nápověda k ostatním argumentům.
- **classes_name_file** – soubor s listem klasifikovaných tříd. Jeden popis třídy na každé řádce.
- **train_files** – adresář obsahující soubory, ze kterých se vytvoří model klasifikátoru.
- **test_files** – adresář obsahující soubory na kterých se otestuje model klasifikátoru.
- **irs** – způsob vytváření příznaků:
 - **bow** – použije Bag of Words

- **bi** – použije Bigram
- **bowtfidf** – použije Bag of Words + TF-IDF
- **classifier** – který klasifikátor se použije s modelem:
 - **nb** – použije Naive Bayes
 - **knn** – použije k-nearest neighbors
- **model_name** – název souboru, do kterého se model uloží.

4.2 GUI mód

`python main.py main.py GUI [-h] model_name`

- **GUI** – argument specifikující, které zpracování argumentů použít.
- **[-h]** – dobrovolný argument, při jehož použití se vypíše nápověda k ostatním argumentům.
- **model_name** – název souboru, ze kterého se načte model, který se použije pro klasifikaci textu v grafickém rozhraní.

5 Shrnutí výsledků

Všechny kombinace jsou zahrnuty v následující tabulce (viz Tabulka 1).

Příznaková metoda	Klasifikátor	Potřebný čas (v sec)	Přesnost
BoW	Naive Bayes		%
Bigram	Naive Bayes		%
Bow + TF-IDF	Naive Bayes		%
BoW	k-nn		%
Bigram	k-nn		%
Bow + TF-IDF	k-nn		%

Tabulka 1: Tabulka výsledků

6 Závěr