****

**KIV/UIR – Semestrální práce**

**Automatická klasifikace dialogových aktů**

**Autor:** David Šavel**Doba řešení:** 20 hodin

Obsah

[1. Popis problému 3](#_Toc103260834)

[2. Analýza problému 3](#_Toc103260835)

[2.1. Reprezentace dat 3](#_Toc103260836)

[3. Algoritmy pro tvorbu příznaků 3](#_Toc103260837)

[4. Návrh řešení 5](#_Toc103260838)

[4.1. Načtení dat 5](#_Toc103260839)

[4.2. Tvorba příznaků 5](#_Toc103260840)

[4.3. Klasifikátory 5](#_Toc103260841)

[4.4. Výsledky klasifikace 5](#_Toc103260842)

[5. Popis řešení 5](#_Toc103260843)

[5.1. Čtení dat 5](#_Toc103260844)

[5.2. Příznaky 6](#_Toc103260845)

[5.3. Klasifikátory 6](#_Toc103260846)

[5.3.1. Naive-Bayes 6](#_Toc103260847)

[5.3.2. K-NN 7](#_Toc103260848)

[6. Uživatelská dokumentace 7](#_Toc103260849)

[7. Závěr 8](#_Toc103260850)

# Popis problému

Cílem semestrální práce je napsat program, který se na základě trénovacích dat naučí co nejlépe klasifikovat komiksové věty a hlášky do předem stanovených tříd. Prvním dílčím úkolem je načíst vstupní data a vhodně je číselně reprezentovat za pomocí tří různých algoritmů. Dále je třeba naimplementovat dva klasifikační algoritmy, které se na trénovacích datech naučí klasifikovat testovací data do tříd podle příznaků. Jeden z klasifikační algoritmů musí být Naivní Bayesův klasifikátor. Nakonec je třeba pro každou kombinaci příznakového a klasifikačního algoritmu vypočítat přesnost s jakou data klasifikoval.

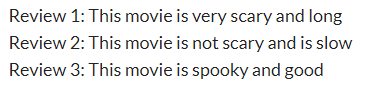
# Analýza problému

## Reprezentace dat

Vstupní data, jak trénovací, tak testovací je třeba vhodně, a hlavně stejným způsobem reprezentovat, aby se například dala počítat vzdálenost dvou vět nebo porovnat jejich zařazení. Nejideálnějším řešením je určitě reprezentovat větu jako pole řetězců, ovšem musí se myslet na to, jak správně z textu tyto řetězce dostat. Oddělit řetězce podle mezer by v tomto případě nebylo nejmoudřejší, protože   
by v řetězcích zůstávala interpunkční znaménka a program by kvůli tomu vyhodnocoval řetězce jako odlišné, proto je nejlepší interpunkční znaménka odstranit co nejvíce to půjde ještě před spuštěním algoritmu.

# Algoritmy pro tvorbu příznaků

Vybrat vhodné algoritmy pro tvorbu příznaků z textového dokumentu. Většina nejznámějších algoritmů využívá k tvorbě příznakového vektoru slovník, což je seznam všech různých slov, která   
se v celém textovém dokumentu vyskytují. Věta je pak reprezentována vůči celému slovníku, tedy vektorem o délce slovníku. Nejznámější algoritmy jsou Bag of Words a TF-IDF. Bag of Words vyjadřuje kolikrát se každé slovo ze slovníku ve větě vyskytuje.

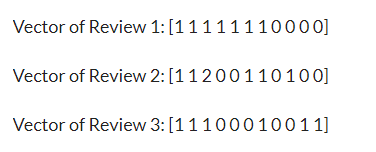


Obrázek 1 - ukázkové věty

Obsah obrázku stůl

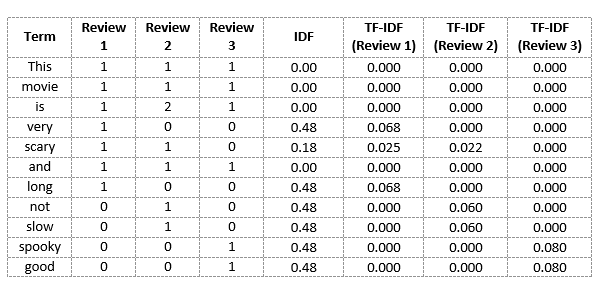
Popis byl vytvořen automaticky

Obrázek 2 - ohodnocení ukázkových vět



Obrázek 3 - příznakové vektory ukázkových vět

Druhý zmíněný algoritmus je o něco složitější. IDF je pro každé slovo ve slovníku stejné. Jedná   
se o logaritmus z podílu počtu celkových vět a počtu vět s daným slovem. TF se pak počítá pro každou větu zvlášť a je rovno podílu násobnosti daného slova ve větě a celkovému počtu slov ve větě. U slov, které se ve větě nevyskytují bude celková hodnota nulová stejně jako u Bag of Words.



Obrázek 4 - TF-IDF ukázkových vět

Zdroj:

Analytics Vidhya, 2020, https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/

# Návrh řešení

## Načtení dat

Vstupní textové soubory se budou načítat po řádcích a v jiných metodách se poté budou parsovat   
na pole řetězců. Stejnou metodou se budou parsovat trénovací i testovací data, aby z nich byla dostávána jednotlivá slova stejně. Nejprve se věta rozdělí na řetězce podle mezer a z jednotlivých řetězců pak budou odřezávány interpunkční znaménka. Pro reprezentaci vět bude vytvořena samostatná třída, ve které bude uloženo pole řetězců, příznakový vektor a typ věty.

## Tvorba příznaků

Pro tvorbu příznaků jsem zvolil algoritmy Bag of Words, TF-IDF a jako poslední Document frequency, která pro každé slovo ze slovníku vyjadřuje v kolik větách celkem se vyskytuje. Aby hlavní část kódu nebyla moc rozvětvená je vhodné, aby třídy, které reprezentují příznakový algoritmus, implementovali stejné rozhraní. Všechny algoritmy mají společnou vlastnost, že z pole řetězců vytvoří příznakový vektor o délce slovníku.

## Klasifikátory

První klasifikátor je Naivní Bayesův klasifikátor a jako druhý jsem zvolil K-NN. U N-Bayese   
je při trénování nutno vytvořit klasifikační třídy a uložit do nich příslušné věty a spočítat pravděpodobnosti pro každou klasifikační třídu. Pravděpodobnost je spočítaná hodnota pro každý prvek ve slovníku, značící pravděpodobnost, že věta s daným slovem bude v této třídě.   
Oba klasifikátory by opět měli implementovat stejné rozhraní, jejich společnou vlastností je, že pro každý prvek z testovacích dat určí, do jaké patří klasifikační třídy. K-NN na rozdíl od Bayese nemusí   
při klasifikaci mít rozdělená trénovací data do klasifikačních tříd.

## Výsledky klasifikace

Pro každý větný celek z testovacích dat se výsledek uloží jako atribut ve třídě reprezentující věty.   
Pro všechny testovací data se spočítá počet správně klasifikovaných a z toho se vypočítá procentuální úspěšnost.

# Popis řešení

Implementováno v jazyce Java, verze 11.0.8

## Čtení dat

Ze všech vstupních souborů se načítají data po řádcích do pole řetězců vlastní metodou *readFile(String fileName)*. Testovací i trénovací data pak budou uložena v poli instancí třídy Sentence. V konstruktoru této třídy se pro danou větu ze řádku ze souboru udělá pole řetězců a jednotlivé řetězce se rovnou oříznou o interpunkční znamínka. To se děla ve *while* cyklu, který kontroluje konec a pak začátek řetězce a pokud tam najde interpunkční znaménko, tak řetězec zkrátí. Pokud je řetězec tvořen pouze interpunkčními znaménky tak se neukládá do výsledného pole reprezentujícího větu. Výjimkou   
je otazník, který je ukládán jako samostatný řetězec, protože mi přijde jako důležitá část tázací věty.

## Příznaky

Ještě před klasifikací a tvorbou příznaků se metodou *makeVocabulary()* vytvoří slovník, tedy pole všech slov vyskytujících se v trénovacích datech. Poté se zavolá metoda *feature()*, která trénovacím datům vytvoří příznakové vektory. V této metodě se ve switchi rozhone podle vstupního parametru, který příznakový algoritmus bude použit. Typem proměnné příznakového algoritmu je rozhraní *Feature*, které implementují všechny třídy příznakových algoritmů. Nad všemi se dá zavolat metoda *feature()*, která ke všem trénovacím datům vytvoří příznakový vektor. V případě algoritmu TF-IDF se nejprve spočítá IDF pro všechny prvky slovníku a poté se až se počítá TF, pro každou větu zvlášť.

## Klasifikátory

Který klasifikátor bude použit se stejně jako u příznakových algoritmů vybere ve switch struktuře podle vstupního parametru. Typem proměnné klasifikátoru je rozhraní *Classificator*, které implementují všechny třídy reprezentující klasifikátor.

### Naive-Bayes

Naivní Bayesův klasifikátor si po zavolání metody *classify()* roztřídí trénovací data do klasifikačních tříd podle jejich typu. Klasifikační třídy mají opět pro ukládání dat vlastní třídu *ClassificationClass*.   
Do *ArrayListu* se v *ClassificationClass* ukládají všechny věty, které do klasifikační třídy patří.   
Po roztřízení trénovacích dat se volá metoda *train()*, která spočítá pro každou klasifikační třídu pravděpodobnosti výskytu prvků ze slovníku. Tyto výpočty se provádí ve třídě *ClassificationClass,*   
ve třídě *N\_Bayes* se tyto metody pouze volají. Klasifikace věty pak probíhá tak, že se pro každou klasifikační třídu spočítá pravděpodobnost, že do ní věta patří, a z těchto pravděpodobností se vybere ta největší a do této třídy se věta zařadí.

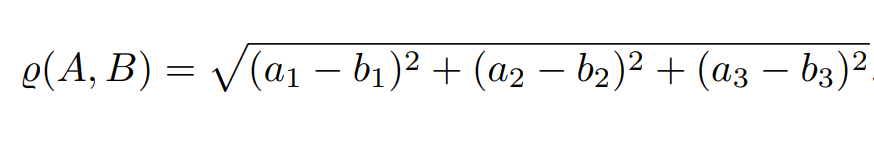


Obrázek 5 -vzorec pro výpočet pravděpodobnosti

Pravděpodobnost se počítá tak že se vynásobí pravděpodobnosti jednotlivých slov, že patří do dané třídy, a výsledek se vynásobí podílem počtu všech prvků co patří do klasifikační třídy a celkovým počtem prvků v trénovacích datech.

### K-NN

V případě, že se používá tento klasifikátor, je třeba ještě před voláním metody *classify()*, vytvořit příznakové vektory testovacím datům, aby se dala počítat vzdálenost prvku z testovacích dat a prvku z trénovacích dat. To se dělá zavoláním metody *feature()*. Správný příznakový algoritmus   
už je nastaven, akorát se jako parametr místo trénovacích dat dosadí ty testovací. Klasifikace   
pak probíhá tak, že se ke klasifikovanému prvku najde K nejbližších prvků z trénovacích dat. Nejbližší   
je myšleno na základě eukleidovské vzdálenosti vektorů.



Obrázek 6 - vzorec pro výpočet Euklidovské vzdálenosti 2 vektorů

K je konstanta pevně nastavená ve třídě K-NN, ale nesmí být příliš vysoká, protože některé klasifikační třídy mají hodně málo prvků. Hledání nejbližších sousedů probíhá tak, že se prvních K prvků trénovací množiny uloží do pole momentálně nejbližších sousedů, a poté se prochází zbytek prvků trénovací množiny. Vždy se vypočítá prvek ze skupiny momentálně nejbližších, který je nejdál,   
a momentálně procházený prvek se porovnává s tím nejvzdálenějším a pokud je blíž, tak ho nahradí   
a nejvzdálenější prvek se vypočítá znovu. Prvek se pak klasifikuje do třídy, která má mezi nejbližšími sousedy nejpočetnější zastoupení.

# Uživatelská dokumentace

Program se spouští celkem se 6 vstupními parametry, pokud chce uživatel vytvořit nový modul anebo s jedním parametrem, pokud modul již existuje a je umístěn do stejné složky.

## Spuštění se 6 parametry

* Soubor se seznamem klasifikačních tříd – každá třída má svůj název na samostatném řádku.
* Soubor s trénovacími daty – každá věta na vlastním řádku + na začátku řádku uvedena klasifikační třída, do které věta patří
* Soubor s testovacími daty – ve stejném formátu jako trénovací data
* Název parametrizačního algoritmu – povolené parametry: bow, df, tf\_idf
* Název klasifikačního algoritmu – povolené parametry: bayes, k\_nn
* Název výstupního modelu – libovolný název výstupního textového souboru, včetně .txt

## Spuštění s jedním parametrem

Uživateli se otevře jednoduché GUI. Do textového pole lze napsat vlastní text, který bude po stisknutí tlačítka *klasifikuj* zařazen do klasifikační třídy. Výsledek se vypíše do dolního pole. Je zde ještě tlačítko pro vymazání obsahu textového pole a nápisy informující o příznakovém algoritmu a klasifikátoru modulu.

# Závěr

V rámci semestrální práce byl vytvořen program, který se na trénovacích datech naučí klasifikovat části dialogů z komiksů. Vstupními parametry jsou trénovací a testovací soubory, název příznakového algoritmu, název klasifikačního algoritmu, seznam klasifikačních tříd a název modulu. Program klasifikuje testovací data a poté podle skutečnosti ověří svoji přesnost. Výsledky pro jednotlivé konfigurace byly následující:

|  |  |  |
| --- | --- | --- |
|  | N\_Bayes | K-NN |
| TF-IDF | 55,03 % | 48,34 % |
| Bag of Words | 52.87 % | 32,62 % |
| Document Frequency | 48,35 % | 47,13 % |

Přesněji klasifikoval N-Bayesův klasifikátor. Klasifikátor K-NN je ovlivněn tím, jaký zvolen parametr K, protože pokud se zvolí příliš vysoký, bude většinou prvek zařazen do třídy s největším početním zastoupením. Tento klasifikátor by byl přesnější, pokud by v trénovacích datech byl rovnoměrný počet prvků ze všech klasifikačních tříd. Mezi příznakovými metodami byli výsledky nejpřesnější při použití TF-IDF algoritmu. Funkce klasifikátoru je samozřejmě hodně ovlivněna i trénovacími daty.