VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POKROČILÉ METODY STROJOVÉHO UČENÍ PRO KLASIFIKACI TEXTU

ADVANCED MACHINE-LEARNING METHODS FOR TEXT CLASSIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN DOČEKAL**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. RNDr. SMRŽ PAVEL, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Dočekal Martin**
Obor: Informační technologie
Téma: **Pokročilé metody strojového učení pro klasifikaci textu**
Advanced Machine-Learning Methods for Text Classification
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s pokročilými metodami strojového učení používanými v oblasti zpracování přirozeného jazyka, např. fastText, Word2Vec a dalšími.
2. Prostudujte rozhraní dostupných nástrojů pro jazykové předzpracování českého textu se zaměřením na výkonnost a kvalitu výstupů.
3. Na základě získaných znalostí navrhněte a realizujte systém pro sémantickou klasifikaci obsahu knih a dalších zadaných dokumentů.
4. Vytvořte testovací sadu pro vyhodnocení přínosu jednotlivých částí systému a vyhodnoťte vytvořený systém pomocí standardních metrik.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá pokročilými metodami strojového učení pro klasifikaci textu. Metody jsou nejprve popsány a poté je na základě těchto metod vytvořen systém sloužící pro klasifikaci textových dokumentů. Systém poskytuje také nástroje pro předzpracování dokumentů a hodnocení klasifikátoru. Práce uvádí použití systému na úloze v reálných podmínkách.

Abstract

This thesis deals with advanced machine-learning methods for text classification. At first, these methods are described, and then text classification system is created based on these methods. The system also provides tools for document preprocessing and evaluation of classifier. The thesis describes the use of the system in a real-life task.

Klíčová slova

strojové učení, extrakce příznaků, Bag-of-words, TF-IDF, word2vec, doc2vec, hašování příznaků, k-nejbližších sousedů, Multinomiální naivní Bayes, Support Vector Machine, klasifikace, vyhodnocování klasifikátoru, předzpracování, slučování klasifikačních metod, vyvažovací algoritmy

Keywords

machine-learning, feature extraction, Bag-of-words, TF-IDF, word2vec, doc2vec, feature hashing, k-nearest neighbors, Multinomial Naive Bayes, Support Vector Machine, classification, evaluation of classifier, preprocessing, ensemble classification methods, balancing algorithms

Citace

DOČEKAL, Martin. *Pokročilé metody strojového učení pro klasifikaci textu.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Pokročilé metody strojového učení pro klasifikaci textu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Dočekal
16. května 2017

Poděkování

Rád bych poděkovat svému vedoucímu bakalářské práce Doc. RNDr. Pavlu Smrži, Ph.D., za podnětné rady a pomoc při tvorbě. Dále děkuji Moravské zemské knihovně, která mi poskytla testovací data.

Obsah

Kapitola 1

Úvod

Klasifikace textu je zkoumána již desetiletí a metody strojového učení se pro ni používají již také dlouho dobou. Za ten čas se z klasifikace textu stala obsáhlá disciplína, která zahrnuje spoustu starších i novějších metod. Tyto metody se v dnešní době, díky rostoucímu výkonu počítačů a lepší dostupnosti informací, používají na stále větší objemy dat.

Výzkum v dané oblasti stále pokračuje. Soustředí se na využití nových metod strojového učení a na specifické problémy klasifikace textu, jako je například problém velkého množství kategorií s nevyváženým zastoupením v trénovacích datech.

V této práci chci vyzkoušet moderní metody, které lze použít na velká data. Zjistit vliv předzpracování na dokumentech v českém jazyce, které byly převedeny z tištěné formy do elektronické pomocí OCR. Takovéto dokumenty běžně obsahují spoustu chyb. Také chci zjistit vliv strategií pro problémy s výrazně nevyváženou reprezentací kategorií v trénovací sadě. Toto vše se pokusím ověřit na dostatečně velkém vzorku dat a praktické úloze v reálných podmínkách.

Mimo výše uvedeného si tato bakalářská práce dává za cíl vytvořit nástroj, který bude schopen klasifikovat elektronické textové dokumenty. Nebude se jednat pouze o klasifikátor, ale o komplexní systém, který bude nabízet vícero nástrojů pro: výběr dat, předzpracování textu, extrakci příznaků, testování klasifikátoru a také samotnou klasifikaci. Uživatel si bude moci sám konfigurovat parametry systému a přizpůsobit si jej tak ke své konkrétní úloze.

Prvně budou popsány jednotlivé kroky, které je třeba pro klasifikaci učinit. Poté uvedeme jednotlivé techniky. Následně popíšeme jakým způsobem lze testovat úspěšnost klasifikátoru a pomocí jakých metrik může být úspěšnost klasifikátoru hodnocena. V práci bude uveden také návrh systému a popis jeho implementace. Na úplný závěr uvedeme experimenty s vytvořeným systémem.

Kapitola 2

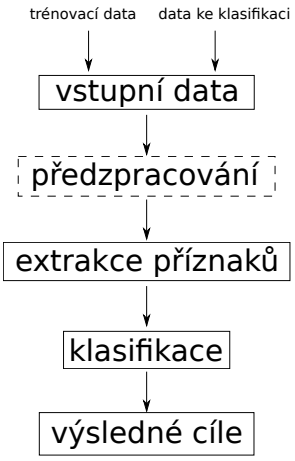
Klasifikace textových dokumentů

Nejprve, než-li se pustíme do samotné klasifikace dokumentů, si uvedeme obor umělé inteligence a strojového učení.

Česká terminologická databáze knihovnictví a informační vědy vykládá termín umělá inteligence takto: „Mezioborová vědní disciplína na pomezí kognitivních věd, kybernetiky a počítačové vědy, která zkoumá a modeluje inteligenci s cílem vyvinout software a hardware, který bude při řešení úloh používat postupy považované za projev lidské inteligence...“ [13]

Strojové učení je jednou z oblastí umělé inteligence. Dává si za cíl, aby byl počítačový program schopen měnit své chování na základě nových dat, aniž by byl přímo naprogramován. Mezi základní druhy algoritmů strojového učení lze zařadit:

- **učení bez učitele** - Systému poskytneme pouze vstupní data.
- **učení s učitelem** - K datům, která poskytneme systému připojíme i žádaný výstup. Tedy například u klasifikace bychom připojili značku.
- **zpětnovazebné učení** - Podle toho jak systém jedná dostává odměny nebo naopak tresty. Na základě těchto podmětů se učí.



Obrázek 2.1: Příklad fází klasifikátoru.

Práce se primárně zaměřuje na využití metod strojového učení s učitelem.

Při klasifikaci textových dokumentů chceme automaticky na základě trénovacích dat přiřadit danému dokumentu značku (např. kategorii). Samotné značky můžou být pouze dvě, kupříkladu pro případy, kdy se rozhodujeme, jestli se jedná o nevyžádanou zprávu či nikoliv. Také jich může býti více, v takovém případě mluvíme o vícetřídní klasifikaci[1]. Typický úkol pro tento druh klasifikace může být roztřídění knih do kategorií.

Klasifikace se typicky skládá z několika fází. Na obrázku 2.1 je znázorněn příklad možných fází.

Pracujeme se dvěma druhy vstupních dat. Jedny slouží pro natrénování klasifikátoru a druhý typ dat se používá pro samotnou klasifikaci. Nejprve je tedy nutné poskytnout

[1] Anglický termín: multiclass classification.

trénovací data, na kterých se klasifikátor učí. Poté jej můžeme použít pro data, která mají být klasifikována.

Jak data pro trénování tak samotná data pro klasifikaci procházejí znázorněnými kroky až na krok předzpracování, který není nutný (proto čárkovaně), ovšem je vhodný . Provádí se v něm například odstraňování stopslov. V této kapitole si předzpracování a další kroky více popíšeme. Ukážeme, co konkrétně se v jednotlivých krocích odehrává a jaké techniky se k tomu používají.

2.1 Předzpracování

Jedná se o nepovinný krok v klasifikátoru, který ale může mít značný vliv na úspěšnost klasifikace. Vstupem do této fáze jsou textová data, která se budeme snažit redukovat a pozměňovat. Uvedeme si typické představitele technik, které se pro tuto činnost používají.

Mimo těch, které budou uvedeny dále, lze pro vhodnější podobu vstupního textu (z hlediska klasifikace) oddělovat od slov znaky: čárky, tečky, dvojtečky a další, které se v běžném textu píší jako součást slova (neoddělené mezerou). Dále bychom mohli také převést znaky na pouze malá či velká písmena. Toto je vhodné hlavně u slov na začátku vět, ale už to nemusí být tak vhodné u názvů a jmen. V názvech/jménech se mohou totiž vyskytovat jinak běžná slova jako například název města Velké Meziříčí. Po převedení by pak byl smazán rozdíl mezi běžným významem tohoto sousloví a názvu města.

2.1.1 Odstranění stopslov

Dle [2] se za stopslovo dá označit slovo, které díky své vysoké frekvenci výskytu v textu ztrácí význam pro klasifikaci. Takováto slova nás tedy zpravidla nezajímají. V jednotlivých jazycích lze nalézt celou řadu slov, které tuto podmínku splňují. My si tedy můžeme vytvořit seznam slov a tato slova z textu, která se vyskytují v seznamu, odstranit. Pro češtinu by se v seznamu vyskytovala slova typu: ale, ani, co, však a další.

2.1.2 Výběr slovních druhů

Obdobně jako při odstraňování stopslov můžeme odstranit slova, která nepatří do námi zvolených slovních druhů. Můžeme například vybrat pouze podstatná jména a klasifikovat pouze na základě nich. Tímto způsobem lze značně zjednodušit vstupní množinu slov, ale také při výběru pouze nevhodných slovních druhů (spojky, předložky) můžeme ztratit příliš mnoho informací z původního textu.

2.1.3 Lemmatizace

Při lemmatizaci převádíme slova do jejich základního tvaru. Jelikož nemusí být žádoucí, abychom brali jako dvě naprosto rozdílná slova například: rychlejší a rychlý. Převedeme tedy všechna slova, do jejich základního tvaru, a tím docílíme toho, že budeme mít vždy dané slovo pouze v jednom tvaru.

2.2 Extrakce příznaků

Při extrakci příznaků se snažíme vytvořit popis dokumentu, který bude vhodnější pro zpracování pomocí počítače. Budeme hledat takovou reprezentaci vstupního dokumentu, kde

nebudeme muset pracovat s celým jeho textovým obsahem, ale budeme se jej snažit převést například do číselného vektoru. Hledáme takovou reprezentaci, která bude úsporná a zároveň informativní. Informativní v případě, kdy se snažíme klasifikovat do kategorií znamená, že dokumenty obsahují takové příznaky, na základě kterých lze odlišit jednotlivé kategorie.

Podíváme se na metody, které dokáží z množiny dokumentů vyrobit číselný vektor s fixním počtem dimenzí, který bude reprezentovat dokument, pro každý z dokumentů v množině.

2.2.1 Bag-of-words a Bag-of-n-grams

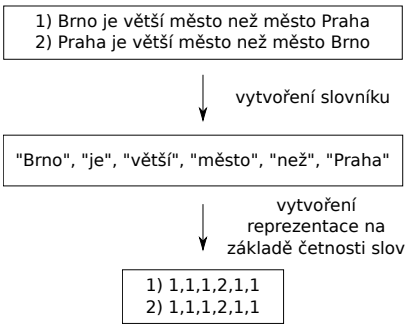
Informace pro tuto sekci byly získány z [16].

Nežli si vysvětlíme pojem Bag-of-words (někdy také jen BOW), tak nejdříve uvedeme pojem, který se k BOW úzce váže a to term frequency (četnost slova v dokumentu).

Term frequency je váha slova v dokumentu, která se vypočítá na základě počtu výskytu slova v daném dokumentu. Takovou váhu můžeme označit například takto: $\mathrm{tf}_{t,d}$. Kde spodní index t vyjadřuje slovo v dokumentu d.

Bag-of-words je poté multimnožina[1] takovýchto vah slov pro každý dokument. Můžeme se tedy na to podívat jako na číselnou reprezentaci dokumentu. Je nutné ovšem poznamenat, že nezohledňuje pořadí slov, ale jen jejich počet.

Uvedený příklad na obrázku 2.2 ukazuje jak dvě sémanticky odlišné věty dostanou stejnou výslednou reprezentaci. Mimo jiné obrázek nastiňuje postup, jakým je možné se k reprezentaci v podobě fixního číselného vektoru dostat.



Obrázek 2.2: Bag-of-words ukázka

Nicméně přes uvedený nedostatek se dá intuitivně očekávat, že podobné dokumenty, budou mít podobnou BOW reprezentaci. Je také vhodné zvážit odstranění stopslov a další techniky uvedené v sekci o předzpracování.

Nemusejí se ovšem používat pouze slova samotná, ale mohou se používat i n tice slov (n-gramy).

N-gramy jsou po sobě jdoucí sekvence n prvků (v našem případě slov). Speciálním případem n-gramu je unigram což je pouze jeden prvek/slovo. Dále používáme bigram (2 slova), trigram (3 slova) a vyšší se již běžně rozlišují pouze s číslovkou na začátku (např. 4-gram).

Pokud bychom za základní jednotku například chtěli vždy brát bigram a měli následující větu:

Netopýrek thajský je nejmenší savec.

Tak bereme všechny možné bigramy, tedy: „Netopýrek thajský“, „thajský je“, „je nejmenší“, „nejmenší savec“. Pro takovýto případ se používá namísto názvu Bag-of-words název Bag-of-n-grams. Z tohoto uvedeného příkladu si můžete povšimnout, že jsme již navázali informaci o pořadí slov, ale jen pro dvě sousedící slova. Zvyšováním velikosti n-gramu by se tato informace rozšiřovala, ale ztrácela by se jednotlivá slova.

[1] Anglicky se multimnožina kromě termínu multiset také někdy označuje jako bag.

2.2.2 TF-IDF

Informace pro tuto sekci byly získány z [16].

TF-IDF je zkratka odvozená od dvou anglických termínů term frequency a inverse document frequency. Pojem term frequency je vysvětlen v sekci výše. Zatím jsme jej popisovali, jako přímý počet výskytů, ale je možné i použití v podobě s logaritmem:

$$1 + log(tf_{t,d}) \tag{2.1}$$

Inverse document frequency je inverzní dokumentová četnost a je vyjádřena pro slovo t takto:

$$idf_t = log\frac{N}{df_t} \tag{2.2}$$

Kde N je počet všech dokumentů a df_t je počet dokumentů, které obsahují slovo t. Takovéto vyjádření nám umožní dostat velké idf_t pro méně četná slova a malé idf_t pro četná slova. Toto je vhodné, protože vzácné slovo napříč dokumenty může být relevantnější pro pozdější rozhodování klasifikátoru.

Jak TF, tak IDF mají vícero podob a my si zde uvádíme pouze některé z nich (více jich lze nalézt v [16]).

Pronásobením obou zmiňovaných četností (tf a idf) dostaneme výsledné TF-IDF pro slovo t v dokumentu d, jak ukazuje rovnice 2.3.

$$tf\text{-}idf_{t,d} = tf_{t,d} \cdot idf_t \tag{2.3}$$

Tímto způsobem tedy můžeme také získat váhy, kterými můžeme reprezentovat dokument podobně jako s Bag-of-words.

2.2.3 Hašování příznaků

Dle [17] se jedná o metodu, která na rozdíl od BOW, nepotřebuje předem utvářet slovník. Místo toho použije hašovací funkci na dané slovo. Tím získá číslo, které podělí počtem rozměrů výsledného vektoru a zbytek po dělení použije jako index do vektoru. Na daný index poté přičte jedničku.

Tímto způsobem můžeme získat vektorovou reprezentaci dokumentu. Jak se píše v odstavci výše, tak dělíme počtem rozměrů vektoru. To nám umožňuje nastavit si vlastní počet rozměrů výsledného vektoru, na rozdíl od BOW. Ovšem pokud zvolíme počet rozměrů příliš malý, bude často docházet ke kolizím.

2.2.4 word2vec

Zde si uvedeme techniku, která byla prezentována v článku [18]. Informace čerpané pro tuto část jsou převzaty z uvedeného článku a z [12].

Word2vec je technika, která se dokáže naučit (jedná se o učení bez učitele) vektorovou reprezentaci slov, s tím, že vektory sémanticky podobných slov si jsou blízké. Také je možné s těmito vektory dělat počty takovéhoto typu:

vektor(žena)+vektor(král)-vektor(muž)

Lze očekávat, že dostaneme vektor, který je blízký vektoru vyjadřující slovo královna.

Pro získávání vektorů se používají dva přístupy: Continuous Bag-of-Words (CBOW) a Skip-Gram model. Jedná se o podobné modely. CBOW se snaží predikovat slovo z jeho kontextu, kdežto Skip-Gram predikuje kontext na základě slova. Dle [12] je CBOW vhodnější pro menší datasety, kdežto Skip-Gram je vhodnější pro větší datasety.

Je ovšem nutné si uvědomit, že touto technikou dostáváme pouze vektorovou reprezentaci slov. Jenomže, jak již bylo uvedeno výše, tak hledáme reprezentaci dokumentů, kde každý dokument v našem datasetu bude reprezentován v podobě číselného vektoru s fixním počtem dimenzí.

Musíme tedy provést dodatečné operace nad těmito vektory. Mohou se použít metody, kdy se průměrují (případně i s váhou) vektorové reprezentace slov, které se v daném dokumentu vyskytují. Dostaneme tak vektor s fixním počtem dimenzí, kde počet dimenzí odpovídá původnímu počtu dimenzí u vektoru, který reprezentoval slovo.

Další cestou je Paragraph Vector popsaný v článku [15]. Paragraph Vector vychází z word2vec, ale jde dále než je reprezentace slova. Dokáže reprezentovat i věty, odstavce či celé dokumenty. Pomocí něj tedy získáme potřebnou vektorovou reprezentaci dokumentu.

2.3 Klasifikace a klasifikátory

Uvedené informace pro tuto sekci byly získány z [16].

Problém klasifikace je následující. Máme množinu cílů (cílem může být například kategorie) a množinu objektů (objektem v našem případě je textový dokument) a my se pokusíme určit, který cíl patří k danému objektu. Někdy může být požadavek i takový, že se má objektu přiřadit více cílů, této problematice se, ale práce dále nevěnuje.

Klasifikátor je funkce, která přiřadí objektům cíl. Může tak činit na základě předem vytvořených pravidel, či pomoci nějaké metody strojového učení.

My se budeme věnovat právě klasifikátorům, které využívají metod strojového učení, bude se vždy jednat o metody spadající pod metody učení s učitelem. Každý takovýto klasifikátor potřebuje trénovací množinu objektů, ze kterých získá potřebné znalosti pro určování cílů. Dokumenty v trénovací množině mají reprezentaci, která byla získána ve fázi extrakce příznaků. Jedná se tedy o příznakové vektory, které mají stejný počet dimenzí.[1] Ke každému takovému vektoru je přidán cíl pro trénování.

Testovací množina (množina objektů, které chceme klasifikovat) je také reprezentována vektory získanými z fáze extrakce příznaků, ale bez cílů, ty se chystáme pomocí klasifikátoru určit.

2.3.1 K nejbližších sousedů

Někdy také jen kNN z anglického k nearest neighbor, je klasifikace, která rozhoduje o příslušnosti objektu k danému cíli na základě k nejbližších objektů. Přičemž předpokládáme, že reprezentace podobných objektů (objektů se stejným cílem) pomocí n rozměrného vektoru jsou si blízké.

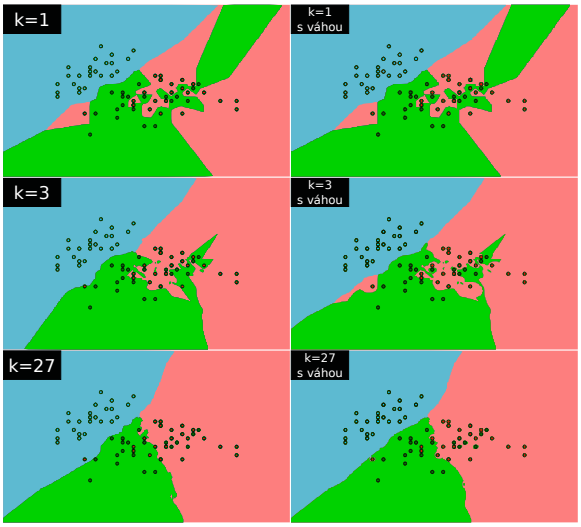
Písmeno k v názvu vystupuje jako parametr. Vezmeme-li k rovno jedné (tedy 1NN), tak v takovémto případě se rozhodujeme na základě jednoho nejbližšího souseda, tedy objektu který klasifikujeme přímo přiřadíme cíl tohoto nejbližšího souseda. Obecně řečeno nemusí

[1] Pro vysvětlení Multinomiálního naivního Bayesova klasifikátoru se ovšem vrátíme zpět k reprezentaci v podobě sekvence slov. Je ovšem možné použít i například binární vektor, kde 0/1 rozlišují zda-li se slovo ze slovníku v dokumentu vyskytuje. Popřípadě i vektor s počty slov (BOW).

být rozhodování na základě pouze jednoho nejbližšího souseda dobré. Tento soused může mít špatně přiřazený cíl nebo být příliš atypický.

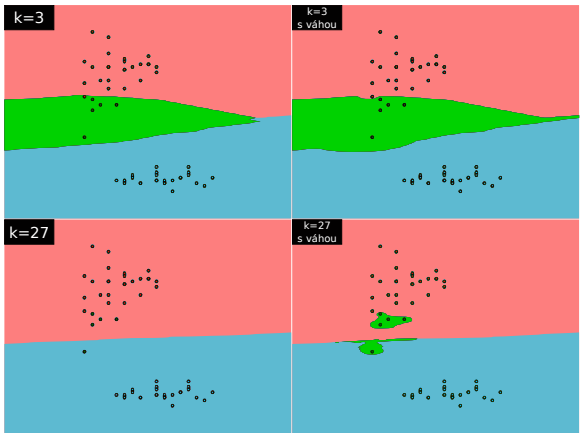
Proto je dobré používat k, které je větší než jedna a je liché. Liché protože o výsledném cíli pro klasifikovaný dokument rozhodujeme na základě většiny a lichým číslem minimalizujeme možnost kolize, která vzniká v případech, kdy alespoň dva cíle mají stejný počet objektů v množině k vybraných nejbližších sousedů. Dále je možné takovéto kolize minimalizovat výběrem k, které je prvočíslem větším než dva.

Doposud jsme předpokládali, že každý soused má stejnou váhu. Ovšem je možné sousedům přiřazovat rozdílné váhy a to na základě jejich vzdálenosti od klasifikovaného dokumentu. Bližším dokumentům přiřadíme váhu větší a vzdálenějším váhu menší. Takovýmto použitím vah můžeme opět zmenšit pravděpodobnost kolize.



Obrázek 2.3: Ukázka vlivu váhy (na základě vzdálenosti) a počtu sousedů na rozhodovací hranice klasifikátoru kNN.

Na obrázku 2.3 je možné vidět vliv parametru k a rozdíl mezi body s váhou a bez. Na tomto příkladu jsou objekty, které mají tři odlišné cíle. Objekty, které patří do cíle s přiřazenou modrou barvou, je možné bez problémů separovat pomyslnou linkou, ale červené a zelené nikoliv.Také je vidět jak se hranice se zvyšujícím se k začíná ucelovat.



Obrázek 2.4: Ukázka zastínění cíle u kNN.

Další problém, který může u kNN nastat je, že pokud zvolíme příliš velké k, tak cíle s menším počtem objektů budou zastíněny, jak ilustruje obrázek 2.4 u neváhované verze.

2.3.2 Multinomiální naivní Bayes

Multinomiální naivní Bayes, v angličtině také jen jako multinomial NB, je pravděpodobnostní metoda strojového učení.

Slovo naivní v názvu získal kvůli předpokladům nezávislosti mezi příznaky. Předpokládá že výskyt či absence slova v dokumentu je nezávislý na výskytu či absenci jakéhokoliv jiného slova. Dále také předpokládá polohovou nezávislost slov, tedy nezávislost na tom na jaké pozici se slovo vyskytuje.

Nejvhodnější cíl pro klasifikovaný objekt hledá na základě tohoto předpisu:

$$c_{map} = \arg\max_{c \in C} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c) \tag{2.4}$$

Kde:

- c_{map} - Je hledaný cíl pro dokument. Dolní index map je z maximum a posterior.
- C - Označuje množinu cílů.
- $\hat{P}(c)$ - Udává apriorní pravděpodobnost, že se dokument nachází v cíli c.
- $\hat{P}(t_k|c)$ - Je podmíněná pravděpodobnost, že se slovo t_k vyskytuje v dokumentu s cílem c. Lze i chápat jako míru toho kolik t_k přispívá k tomu, že c je ten správný cíl.
- t_1, t_2, ..., t_{n_d} - Jsou slova v dokument d, která máme vybraná ve slovníku. Počet těchto slov poté udává n_d.

Pokud si povšimnete, tak v předpisu se vyskytuje součin, který pro velká n_d může způsobit podtečení v plovoucí řádové čárce[1]. Proto se raději v praxi používá následující vyjádření s logaritmy:

$$c_{map} = \arg\max_{c \in C} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)] \tag{2.5}$$

Pravděpodobnosti $\hat{P}(c)$ a $\hat{P}(t_k|c)$ získáváme na základě trénovací množiny. Jakým způsobem je ovšem možné je z trénovací množiny získat? Pro $\hat{P}(c)$ spočítáme všechny dokumenty patřící k danému cíli (N_c) a podělíme počtem všech dokumentů (N). Vizte rovnici 2.6.

$$\hat{P}(c) = \frac{N_c}{N} \tag{2.6}$$

$\hat{P}(t_k|c)$ vyjádříme jako relativní četnost slova t v dokumentech patřících do cíle c:

$$\hat{P}(t_k|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}} \tag{2.7}$$

Kde T_{ct} je počet výskytů slova t v dokumentech s cílem c. V označuje slovník.

Zde však může nastat problém. Máme-li slovo, které se nevyskytuje ani v jednom dokumentu v dokumentech nějakého cíle, vyjde $\hat{P}(t_k|c)$ nula. Pokud vyjde nula bude i celý součin v 2.4 nulový. Tím ovšem potlačíme vliv ostatních příznaků (slov).

Pro vyřešení tohoto problému je lepší použít následující vyjádření:

$$\hat{P}(t_k|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} T_{ct'} + |V|} \tag{2.8}$$

2.3.3 Support Vector Machines

V této části se budeme zabývat lineárním Support Vector Machine, nebo také někdy jen SVM. SVM je klasifikátor, který hledá nejvzdálenější rozhodovací hranici mezi dvěma třídami dokumentů (třídu určuje cíl dokumentu). Tato hranice je v podobě nadroviny[2], kterou můžeme pomocí matematického aparátu vyjádřit jako body $\vec{x}$, které splňují:

$$\vec{w}^T \vec{x} + b = 0 \tag{2.9}$$

[1] Podtečení v plovoucí řádové čárce znamená, že číslo je tak malé, že jej počítač nemůže uložit do paměti. Kdy dojde k podtečení je dáno reprezentací konkrétního číselného datového typu.

[2] Ve 2D je nadrovinou přímka, pro 3D je to plocha.

Kde $\vec{w}$ je normálový vektor a b je konstanta.

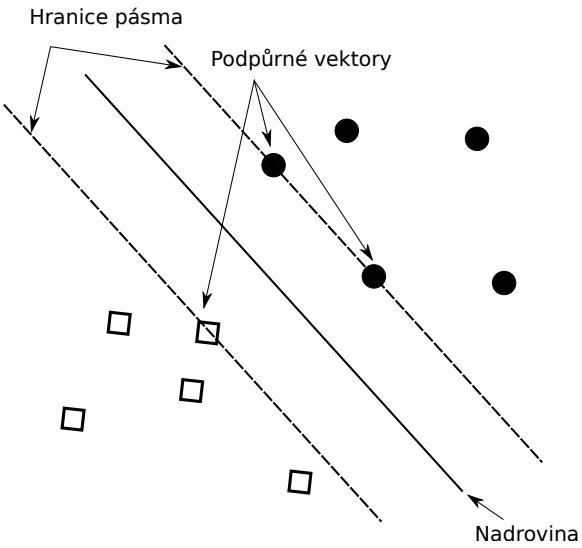
Nyní předpokládejme, že máme lineárně separovatelná trénovací data. Kde každý objekt je v podobě dvojice bodu a třídy, tedy *($\vec{x}_i$, y_i)*. Třídy kvůli dalšímu postupu budeme značit jako 1 a -1, protože máme pouze dvě třídy, tak si s tímto značením postačíme.

Klasifikátor lze pak vyjádřit takto:

$$f(\vec{x}) = sgn(\vec{w}^T\vec{x} + b) \tag{2.10}$$

Hodnoty funkce signum (1 a -1) nám rozdělí body do cílů s naším značením.

Jak ovšem nalézt $\vec{w}$ a b? Pro nalezení těchto hodnot je nutné vyřešit optimalizační úlohu, která zní: Najdi $\vec{w}$ a b že:



Obrázek 2.5: Ilustrace SVM.

- $\frac{1}{2}\vec{w}^T\vec{w}$ je minimalizované
- a pro všechny $\{(\mathrm{x_i},\mathrm{y_i})\}$, $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$

Pro lineárně neseparovatelné případy se používá soft margin klasifikace. Tu nebudeme v tomto textu popisovat (pro více informací vizte [16]) a zaměříme se spíše na rozšíření klasifikátoru pro data s více než dvěma cíli.

Protože nadrovina dělí daný n rozměrný prostor vždy na dvě části jsou SVM ze své podstaty pouze dvou třídní klasifikátor. V praxi se pro rozšíření na více třídní klasifikaci používají tyto přístupy:

- Vytvoříme one-versus-rest klasifikátory, také známé jako one-vs-all.
  - Tato strategie spočívá v tom, že natrénujeme pro každou třídu jeden klasifikátor a všechny ostatní třídy použijeme jako jednu sdruženou protitřídu. Za výslednou třídu testovaného dokumentu budeme brát tu třídu, kterou klasifikujeme s největší jistotou. Počet klasifikátorů, které budeme muset takto vytvořit je roven počtu tříd.
- Vytvoříme one-versus-one klasifikátory:
  - Pro všechny možné páry tříd je vytvořen klasifikátor. Výsledná třída je ta, kterou zvolilo nejvíce těchto klasifikátorů. Počet všech takovýchto klasifikátorů je $|C|(|C|-1)/2$, kde C je množina tříd.

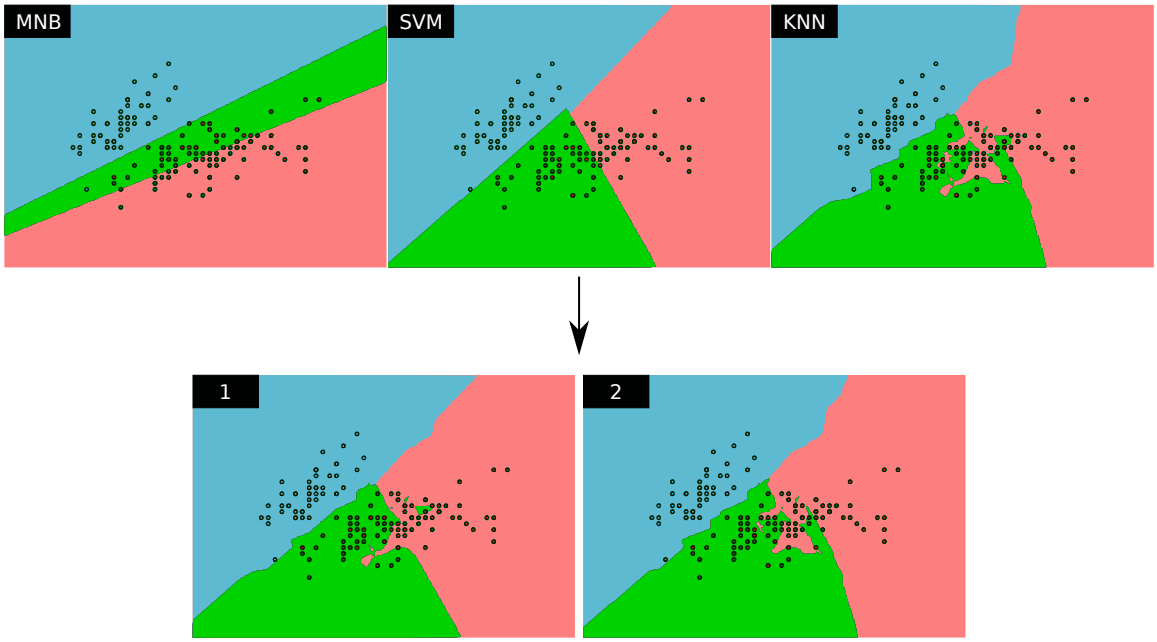
2.4 Slučování výsledků klasifikátorů

V této sekci si uvedeme jakým způsobem jsem řešil v klasifikačním systému, který bude popsán dále v této práci, slučování výsledků z více klasifikátorů do celkového výsledku. Každý klasifikátor má přiřazenou svoji váhu v_k. Čím větší v_k tím více pokládáme výsledky klasifikátoru za přesnější.

Prvně si představme, že máme klasifikátory, které nám pouze říkají výsledný cíl klasifikace a žádné další informace. Dostáváme tedy množinu predikovaných cílů C_p. Pro každý cíl c z C_p vypočítáme:

$$\sum_{k \in K_c} v_{kc} \tag{2.11}$$

Kde v_{kc} označuje váhu klasifikátoru k, který predikoval cíl c a K_c je množina klasifikátorů, které predikovaly cíl c. Sčítáme tedy váhy klasifikátorů, vždy v rámci cíle, který predikovaly. Za výsledný vybereme cíl, který má největší součet těchto vah. Má-li více cílů takovýto součet stejný, vybereme jeden z nich.



Obrázek 2.6: Ukázka sloučení výsledků tří klasifikátorů do jednoho. Obrázek s označením 1 ukazuje výsledek postupu bez váhování jistotami a obrázek označený 2 ukazuje s váhováním.

Nyní se posuneme ke klasifikátorům, které nám poskytují ke každému cíli z množiny všech cílů C míru jistoty s jakou patří objekt do daného cíle. Na takovýto výsledek se můžeme podívat jako na n-tici $(j_{k1}, j_{k2}, ..., j_{k|C|})$, která obsahuje jistoty s jakou přísluší objekt do daného cíle a to pro všechny cíle z C (cíle jsme si oindexovali čísly). Předpokládáme, že tyto jistoty jsou pro všechny klasifikátory ve stejných škálách řekněme tedy, že jsou v intervalu <0,1>. Budeme chtít vytvořit výslednou n-tici $(j_1, j_2, \ldots, j_{|C|})$, kde výsledná jistota pro cíl s indexem i je:

$$j_i = \frac{\sum_{k \in K} j_{ki} v_k}{\sum_{k \in K} v_k} \tag{2.12}$$

Kde K je množina klasifikátorů a j_{ki} je jistota jakou klasifikátor k přiřazuje cíli, kterému jsme dali index i. Váhu klasifikátoru označuje v_k. Jedná se tedy o vážený průměr jistot, který je váhován váhou klasifikátorů.

Za výsledný cíl vybereme takový, jehož výsledný vážený průměr je nejvyšší. Pokud je takových cílů více, vybereme opět jeden z nich.

2.5 Vyvažování trénovací sady

V reálném prostředí se často setkáváme s daty, která nejsou početně vyvážená. Máme tím namysli, že jsou velké rozdíly v počtech dokumentů u jednotlivých tříd. Tato nevyváženost se může promítnout do horších výsledků klasifikace.

My si tu představíme dvě základní techniky, pro které jsem čerpal informace z [3].

2.5.1 Náhodné podvzorkování

Náhodné podvzorkování (Random Under-Sampling) je první metodou, kterou si představíme. Tato metoda vyvažuje rozdíly ve třídách tím, že u majoritních tříd náhodně odstraňuje dokumenty.

Je nutné zmínit, že tímto způsobem můžeme potenciálně přicházet o podstatné informace. Snižujeme však objem trénovacích dat a tím urychlujeme proces trénování klasifikátoru.

2.5.2 Náhodné převzorkování

Náhodné převzorkování (Random Over-Sampling), na rozdíl od náhodného podvzorkování, neodstraňuje dokumenty z majoritních tříd, ale naopak klonuje náhodně vybrané dokumenty z minoritních tříd. Tímto vyvažuje rozdíly v počtech dokumentů u jednotlivých tříd.

Nedochází tu ke ztrátě dat. Zvyšujeme však pravděpodobnost, že se systém přeučí, a tedy není schopen dobře generalizovat.

2.6 Vyhodnocování úspěšnosti klasifikátoru

V této sekci se podíváme na to, jakým způsobem můžeme zjistit jak úspěšný je náš klasifikátor. Běžně klasifikátor naučíme na našich datech a použijeme jej ke klasifikaci. Při trénování si ovšem naše data rozdělíme na dvě množiny. Tyto množiny obsahují data, která byla předem označena (například ručně). Jednu množinu použijeme k natrénování klasifikátoru, tuto množinu nazýváme trénovací. Druhé množině necháme přiřadit cíle natrénovaným klasifikátorem, takovéto množině dokumentů říkáme testovací. Je žádoucí, aby trénovací množina a testovací množina byly disjunktní. Zajímá nás, jak si klasifikátor vede na datech, která nezná.

Samotné vyhodnocení úspěšnosti provádíme tak, že predikované cíle pro testovací množinu porovnáme s předem získanými cíly, přičemž tyto předem získané cíle považujeme za správné.

Většinou vyžadujeme nějaké kvantitativní vyjádření vyhodnocení úspěšnosti klasifikátoru. Uvedeme si zde metriky: správnost (accuracy), míra F_1, přesnost (precision) a úplnost (recall)[1]. Čerpané informace pro všechny tyto metriky jsou z [16]. Budeme si je pro názornost uvádět pro binární klasifikaci.

[1] České ekvivalenty se v literatuře vyskytují ve vícero překladech. Precision bývá překládáno jako: preciznost, důvěra, konfidence či přesnost. Překlady pro recall jsou například: pokrytí, výtěžnost nebo úplnost.

		pravé	
		pozitivní	negativní
predikované	pozitivní	skutečně pozitivní (tp)	falešně pozitivní (fp)
	negativní	falešně negativní (fn)	skutečně negativní (tn)

Tabulka 2.1: Kontingenční tabulka nebo také matice záměn pro binární případ.

2.6.1 Správnost

Správnost je asi nejintuitivnější metrikou. Vyjadřuje procento správně predikovaných cílů ze všech cílů v testovací množině. Je to tedy zlomek:

$$\text{správnost} = \frac{\text{správně predikované cíle}}{\text{všechny cíle}} \tag{2.13}$$

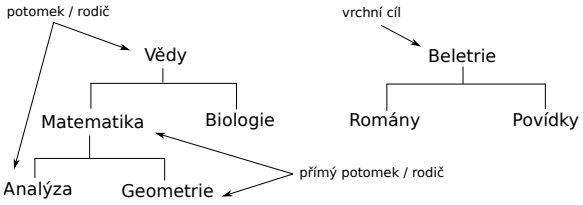
Vyjádření na základě tabulky 2.1:

$$\text{správnost} = \frac{tp + tn}{tp + tn + fp + fn} \tag{2.14}$$

V této práci budeme i pracovat s cíli, které jsou v rámci hierarchie. Například jak ukazuje obrázek 2.7. Považoval jsem tedy za vhodné zavést správnosti, které tuto hierarchii odráží. Jelikož je horší, pokud klasifikátor netrefí ani nejobecnější rozdělení, než když netrefí pouze specifickou kategorii.

Můžeme například považovat predikci, která odhadla cíl mající stejný vrchní cíl v hierachii s pravým cílem, za správnou.

U další možnosti bereme za správné ty cíle, které padnou do množiny všech rodičů (i prarodičů a dále, obdobně i u potomků) nebo potomků pravého cíle, popřípadě bereme za správný i přímo pravý cíl samotný.



Obrázek 2.7: Ilustrace hierarchie cílů.

Nakonec zavedeme správnost, která bere za správné pouze přímé potomky a rodiče pravého cíle. Stejně tak i pravý cíl samotný.

Pokud označíme správnost pracující s nejvrchnějším cílem jako A_T, správnost pracující se všemy potomky/rodiči jako A_F, správnost pracující pouze s přímými potomky/rodiči jako A_{DF} a klasickou správnost uvedenou dříve jako A. Tak platí následující:

$$A_T \geq A_F \geq A_{DF} \geq A \tag{2.15}$$

2.6.2 Přesnost, úplnost a míra $\mathbf{F_1}$

Přesnost se ptá na otázku: Jaký je podíl skutečně pozitivních ve všech cílech predikovaných jako pozitivních. Jedná se o:

$$\text{přesnost} = \frac{tp}{tp + fp} \tag{2.16}$$

Úplnost je podíl pozitivních cílů, které byly predikovány klasifikátorem, ve všech pozitívních cílech. Dostáváme zlomek:

$$\text{úplnost} = \frac{tp}{tp + fn} \tag{2.17}$$

Míra F_1 vychází z míry F. Míra F je váženým harmonickým průměrem přesnosti a úplnosti:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{2.18}$$

Kde P je přesnost a R je úplnost. Pokud zvolíme $\beta = 1$ získáme míru F_1, tedy:

$$F_{\beta=1} = F_1 = \frac{2PR}{P + R} \tag{2.19}$$

2.6.3 Křížová validace

Dle [22] při křížové validaci dělíme naši množinu označených dat na dvě množiny: trénovací a testovací. Na trénovacích datech natrénujeme klasifikátor a zjistíme jeho úspěšnost pomocí metrik na testovacích datech. Toto můžeme několikrát opakovat, ale pokaždé s jiným rozdělením dokumentů/objektů do trénovací a testovací množiny.

Kapitola 3

Požadavky na systém pro klasifikaci textu a jeho návrh

V této kapitole si vymezíme požadavky, které od systému pro klasifikaci textu očekáváme. Dále si uvedeme návrh systému, ve kterém budeme vycházet z těchto požadavků. Popis implementace tohoto systému bude uveden, až v kapitole 4.

3.1 Požadavky na systém

Zde si ve stručnosti uvedeme požadavky, které na výsledný systém klademe. Jedná se o tyto body:

- Schopnost pracovat nad plnými texty dokumentů a současně i nad metadaty, které jsou k textovým dokumentům připojeny.
- Výběr dat/metadat z poskytnuté datové množiny, a to na základě uživatelem zvolených kritérií.
- Předzpracování textu (se zaměřením na český jazyk).
- Možnost provedení fáze extrakce příznaků samostatně. Možnost uložit si tyto extrahované příznaky a až posléze je použít pro trénování klasifikátoru.
- Dovednost trénovat samotný klasifikátor a možnost uložit si jej pro pozdější použití.
- Schopnost na základě uloženého natrénovaného klasifikátoru predikovat cíle textových dokumentů.
- Integrace metod pro vyvažování reprezentace kategorií v trénovací sadě.
- Poskytnout rozhraní pro testování úspěšnosti klasifikátoru.

Snahou je vytvořit systém, který uživateli poskytne škálu nástrojů, které jsou vhodné pro klasifikaci dokumentů. Snažíme se tedy minimalizovat potřebu dalších nástrojů, jako by tomu bylo například v případě, kdyby systém podporoval pouze extrakci příznaků a samotnou klasifikaci.

3.2 Návrh

V této sekci je popsána fáze návrhu systému. Návrh je tvořen na vymezených požadavcích ze sekce 3.1.

Snahou bude navrhnout systém tak, aby umožňoval základní práci s minimální konfigurací a současně umožňoval uživateli, který má komplexnější požadavky, přizpůsobit si parametry systému ke svým účelům.

Pro snadnější udržovatelnost a rozšířitelnost bude systém implementován za pomocí objektů.

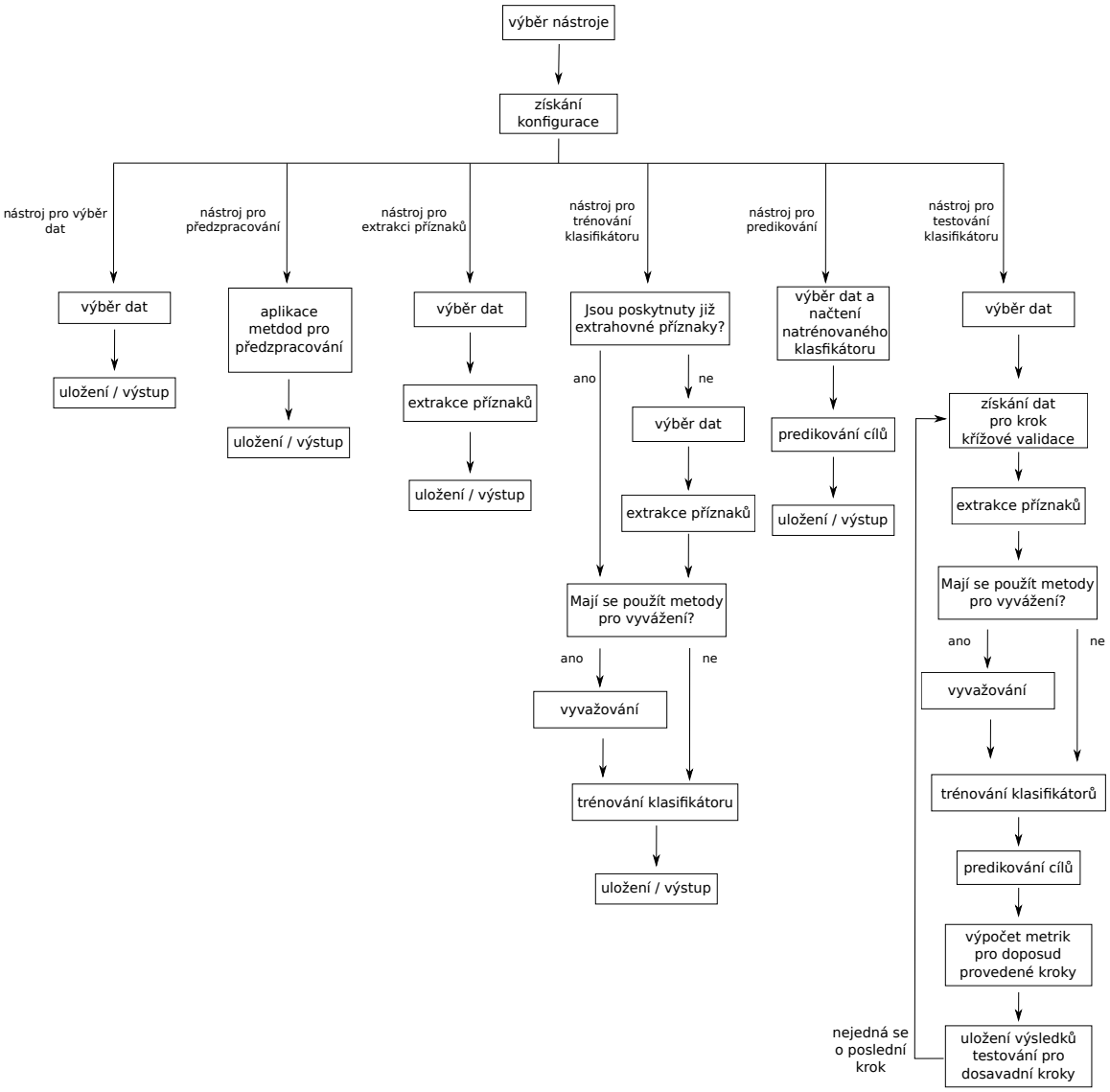
Ovládání systému je řešeno skrze klasické textové uživatelské rozhraní. Dodatečná nastavení budou obsahovat konfigurační soubory. Konfigurační soubor umožňuje lepší znovupoužitelnost a je vhodnější pro více parametrů než-li zadávat parametry přímo z příkazové řádky.

Dle specifikovaných požadavků navrhuji, aby systém nabízel uživateli tyto funkce/nástroje:

- Výběr dat
  - Uživatel si bude moci vytvořit, na základě definovaných parametrů, podmnožinu ze svých dat/metadat a uložit si ji pro další použití.
- Předzpracování
  - Úprava plného textu dokumentů.
  - Minimálně techniky, popsané v 2.1. Tedy: odstranění stopslov, výběr slovních druhů a lemmatizaci.
- Extrakce příznaků
  - Extrahování příznaků a jejich uložení ze vstupních dat/metadat. Tato vstupní data bude možné vybírat stejně jako u výběru dat. Tedy fáze výběru dat bude probíhat před extrahováním příznaků.
  - Pro extrakci bude podporovat minimálně: Bag-of-words (n-grams), TF-IDF a Paragraph Vector (word2vec). Pro více informací vizte 2.2.
- Trénování klasifikátoru
  - Trénování klasifikátoru může probíhat na již extrahovaných příznacích nebo přímo ze vstupních dat. Pokud se bude trénovat přímo na vstupních datech, provede se fáze výběru dat a extrakce příznaků, před natrénováním klasifikátoru. Natrénovaný klasifikátor bude možné uložit, pro pozdější použití.
  - Podporované klasifikátory: Support Vector Machines, K nejlbližších sousedů, Multinomiální naivní Bayes (vizte 2.3).
  - Schopnost vyvážení reprezentace kategorií v trénovací sadě.
- Predikování cílů
  - Na základě natrénovaného klasifikátoru provede predikci cílů. K predikovaným cílům bude možné zobrazovat dodatečná metadata dokumentu. Také podobně jako v předchozích i tady bude možné vybírat vstupní data pro klasifikaci.

- Testování
  - Tato funkce umožní testovat úspěšnost natrénovaného a uloženého klasifikátoru, kterou bude hodnotit na základě standardních metrik. Data, která budou použita pro trénování, lze selektovat stejně jako v minulých případech.
  - Bude tedy podporovat standardní metriky pro vyhodnocování úspěšnosti a také křížovou validaci (vizte 2.6).

Nástroje se budou vybírat pomocí prvního argumentu programu a za tímto argumentem budou následovat dodatečné argumenty, které budou dále (i s konfiguračním souborem) rozvíjet požadavky ke konkrétní činnosti.



Obrázek 3.1: Návrh systému vyjádřený blokovým schématem.

3.2.1 Konfigurace

Systém bude umožňovat nastavovat velké množství parametrů a není vhodné vše zadávat z prostředí příkazové konzole. Proto většina parametrů bude nastavována skrze konfigurační soubory. Systém bude moci používat dva konfigurační soubory.

Jeden s trvalým nastavením a druhý, který bude připojován pomocí argumentu z příkazové řádky. Tento druhý konfigurační soubor bude přepisovat hodnoty z trvalého konfiguračního souboru. Slouží ke konkrétnímu nastavení, které si žádá daná úloha. Pokud si uživatel vytvoří takovýto konfigurační soubor může jej i použít v budoucnu k nahlídnutí s jakými parametry systém spustil.

3.2.2 Dělení dat

Systém má pracovat jak s daty tak s metadaty dokumentů. Za data bude považovat systém plné texty dokumentů (neobsahující formátovací znaky a podobně). Metadata jsou pak data, která vnášejí k dokumentu dodatečné informace. Dále v experimentální části budeme používat jako metadata zejména název a klasifikaci v příbuzném systému (k našim kategoriím). Dalším příkladem metadat by mohl být například autor dokumentu.

Budeme předpokládat, že tato metadata jsou co se týče do velikosti o mnoho menší než plná data a vzhledem k tomuto předpokladu bude dále systém implementován.

3.2.3 Implementační jazyk a knihovny

Vzhledem k poměrně velké rozsáhlosti projektu jsem se rozhodl použít jako implementační jazyk Python. Python dovoluje psát kód rychle a také poměrně efektivně. Implementace například v C/C++ by sice dosahovala pravděpodobně větší efektivity, ale psaní takového systému by bylo mnohem časově náročnější.

Mimo již zmíněné důvody je Python vhodný i z jiného důvodu a tím je velké množství nástrojů pro oblast strojového učení.

Pro extrakci příznaků, klasifikátory a jejich testování navrhuji použít nástroj scikit-learn [20]. Bohužel tento nástroj nezahrnuje Paragraph Vector (word2vec vůbec). Pro tyto účely bude použit nástroj Gensim [21].

Kapitola 4

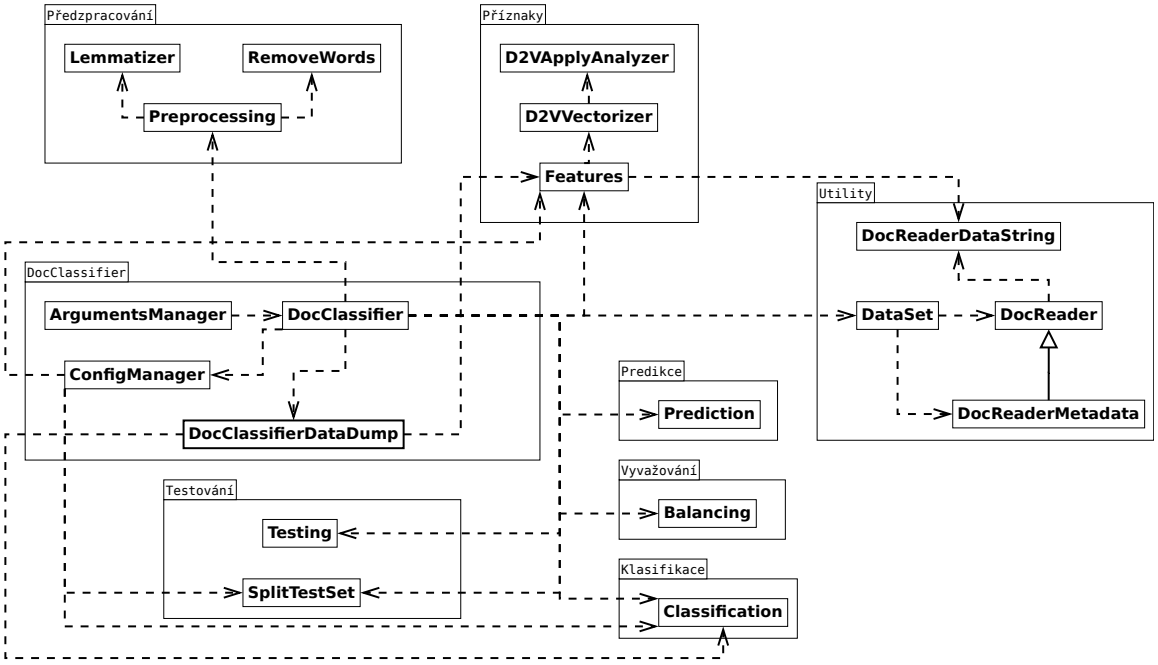
Implementace systému pro klasifikování dokumentů

V této kapitole provedeme popis implementace systému. Uvedeme si externí knihovny/nástroje, které byly v systému začleněny, a také, k čemu jsou používány. Popíšeme požadavky na formát vstupních dat a také formu výstupních dat. Budou zde uvedeny všechny části systému z pohledu programátora a také jejich popis.

4.1 Části systému

V této sekci si popíšeme systém z pohledu programátora. Uvedeme si, jak jednotlivé části systému spolu souvisí a jaké mají odpovědnosti.

Nejprve si uvedeme diagram 4.1, který nám graficky znázorňuje části systému, a to včetně všech význačných tříd a vztahů mezi těmito třídami.



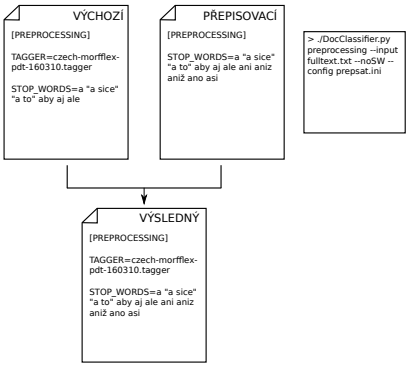
Obrázek 4.1: Zjednodušený diagram znázorňující třídy systému a závislosti mezi nimi.

Dále si popíšeme jednotlivé části systému, které jsou uvedeny v diagramu 4.1. Zaměříme se na odpovědnosti, které těmto částem přísluší. Budeme používat k osvětlení názvy tříd, které jsou v tomto diagramu uvedeny.

4.1.1 Konfigurace a argumenty programu

Argumenty zadané od uživatele zpracovává ArgumentsManager, který na základě nich spouští obslužnou metodu třídy DocClassifier pro konkrétní úkol.

Další konfigurace je načítána z konfiguračních souborů pomocí třídy ConfigManager. Nejprve je načtena konfigurace z výchozího konfiguračního souboru a poté, pokud je zadána v argumentu programu, tak i konfigurace z jiného konfiguračního souboru. Výchozí konfigurace je přepisována konfigurací, která je v konfiguračním souboru jehož cesta je zadána v argumentech programu. Samotný formát konfiguračních souborů odpovídá formátu, který zpracovává pythonovský modul configparser (.ini).



Obrázek 4.2: Ilustrace získání výsledné konfigurace.

Všechny tři třídy (ConfigManager, ArgumentsManager a DocClassifier) provádí validaci vstupní konfigurace. ConfigManager validuje konfigurační soubory. ArgumentsManager validuje argumenty programu z příkazové řádky. DocClassifier validuje sloučenou konfiguraci ze souborů a argumentů z příkazové řádky.

DocClassifier po získání konfigurace daný úkon vykoná s pomocí ostatních tříd.

4.1.2 Práce s daty a metadaty

Systém pro klasifikaci pracuje se dvěma druhy dat: data (plný text) a metadata. S tím, že metadata jsou pokládán za povinná a data nikoliv. Tento přístup nám umožní, provádět klasifikace pouze na základě metadat. Klasifikace pouze na základě plných textů není možná, protože pro natrénování klasifikátorů potřebujeme znát cíle dokumentů. V trénovacích datech je nutné tyto cíle uvést, a to právě mezi metadaty.

Data a metadata jsou uložena v samostatných souborech, jejichž formát je popsán v podkapitole 4.3.

U souboru, který obsahuje plné texty dokumentů, systém předpokládá, že bude dosahovat velkých datových objemů, kdežto soubor s metadaty ne. Je vhodné mít tyto dva druhy dat oddělené, protože pokud chceme pracovat pouze s metadaty, nemusíme procházet potenciálně velký soubor s daty, které nás momentálně nezajímají.

Z programátorského hlediska je ovšem vhodnější pracovat s těmito daty pohromadě. Chceme totiž přednostně pracovat s dokumentem jako celkem.

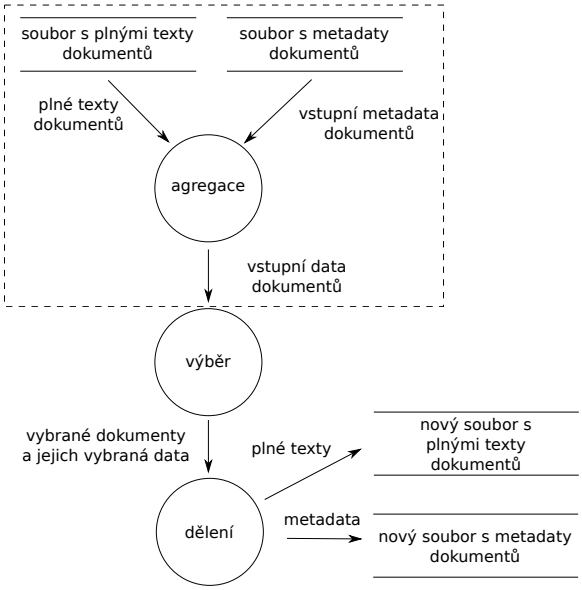
Třída DocReader čte tyto dva soubory a slučuje data a metadata dokumentů dohromady (čtení pouze metadat provádí DocReaderMetadata). Poskytuje také možnosti pro filtrování dokumentů. Mezi tyto možnosti patří:

- Výběr dokumentů, které mají ne/prázdná daná pole v metadatech.
- Výběr dokumentů, které mají prázdná daná pole v metadatech.
- Výběr metadatových položek splňující daný regulární výraz.

- Výběr dokumentů přiřazených k položce v poli, která se vyskytuje u alespoň definovaného minimálního počtu dokumentů.
- Omezení maximálního počtu dokumentů na položku v poli.

Mimo tyto filtry, také umožňuje výběr slov na základě jejich pozice z plného textu. Můžeme tedy například vybrat posledních 10 slov, z každého dokumentu. Také na základě pozice lze vybírat položky v metadatovém poli.

V neposlední řadě třída DocReader poskytuje možnost pro ušetření systémové paměti. Místo toho, aby se nahrály všechny plné texty pro vybrané dokumenty, tak najde pouze pozice v souboru odkud má daný dokument začít číst. Nemusíme tedy nutně uchovávat dlouhé řetězce textu. Místo toho vytvoří třídu DocReaderDataString a předá jí potřebné informace, aby věděla, jaký dokument a jakou jeho část má číst. DocReaderDataString až na vyžádání přečte určenou část dokumentu ze souboru. Tímto způsobem můžeme ušetřit paměť, ovšem tento proces je časově náročnější než prosté čtení textového řetězce.



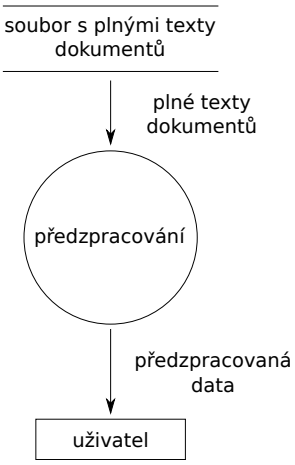
Obrázek 4.3: Ilustrace toku dat při selektování dokumentů a jejich následném ukládání. V systému je možná varianta i bez ukládání plných textů, či metadat. Část diagramu, která je po obvodu vyčárkována, bude dále v textu abstrahována do jednoho celku.

Třída DataSet používá DocReader/DocReaderMetadata a vytváří na základě získaných dat strukturu vhodnou pro extrakci příznaků a trénování/testování klasifikátoru. Lze s ní také zapsat vyfiltrovaná data do souborů a znovu je v budoucnu načíst. Jedná se o třídu, která obaluje činnosti pro práci se vstupními daty.

4.1.3 Předzpracování plných textů

Systém implementuje tyto nástroje sloužící pro předzpracování plného textu:

- Lemmatizace textu.
- Odstranění stopslov
- Překlad znaků do jejich nejbližší možné reprezentace v ascii
- Oddělení znaků ,.:;?! od slov.
- Výběr slov, která splňují podmínku na minimální/maximální počet znaků.
- Výběr slov dle slovního druhu.
- Převod všech znaků na malá/velká písmena.



Obrázek 4.4: Ilustrace toku dat při předzpracování dokumentů. Tento diagram je oproti ostatním v tomto bloku jednoduchý. Jeho hlavní podstatou je ukázat, že se zde nepracuje s metadaty a neprobíhá selekce dat.

Lemmatizace textu a výběr slov na základě slovních druhů je ve třídě Lemmatizer. Pro získávání základního tvaru slova a určování slovního druhu používá nástroj MorphoDiTa (více informací [19] a [24]). Pro správné fungování MorphoDiTy je nutné dodat taggery. Cestu k těmto souborům lze dodat v konfiguračním souboru, je tedy možné je časem měnit za novější varianty.

Za pomocí třídy RemoveWords lze odstraňovat stopslova, či slova nesplňující dané minimum/maximum počtu znaků. Množinu stopslov, lze měnit v konfiguračním souboru.

Zbylé techniky, tedy převod velikosti znaků, překlad znaků do ASCII a oddělení znaků ,.:;?! od slov, provádí přímo třída Preprocessing.

4.1.4 Extrakce příznaků

Popíšeme si jakým způsobem implementujeme extrakci příznaků. Systém podporuje více technik a je navržen s ohledem možného přidávání dalších.

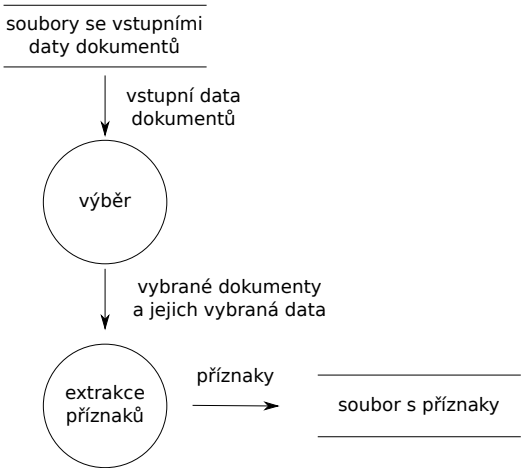
Nežli si uvedeme jaké techniky pro extrakci jsou aktuálně podporovány rozebereme si krok, který předchází samotné extrakci. Jedná se o krok, ve kterém tvoříme ze slov n-gramy.

Z fáze, kde jsme získali plné texty a k nim příslušná metadata, dostáváme z plných textů oddělená slova od sebe a z metadat dostáváme prvky v celku (nedochází k dělení na slova). Musíme tedy k datům a metadatům přistupovat odlišně. Na první pohled by se tento způsob mohl zdát zbytečný, ale dovoluje nám zařadit analyzátor tvořící n-gramy z metadatového prvku jako celku a ne v závislosti na počtu slov, což může být užitečné.

Kromě zmíněného analyzátoru, systém podporuje další analyzátory pro tvoření n-gramů u metadat i plného text. Velikost n-gramu je volitelná pro každý druh dat zvlášť (plný text, pole v metadatech). Nastavení se provádí v konfiguračním souboru.

Tvoření n-gramů a extrakce příznaků provádí třída Features. Pro svoji činnost využívá externí knihovny scikit-learn a třídy D2VVectorizer.

D2VVectorizer slouží jako obálka pro Doc2Vec z Gensimu. Doc2Vec je nástroj, který je založen na Paragraph Vectoru. Přestože tímto nástrojem dokážeme získat vektor, reprezentující daný dokument, tak Doc2Vec nezapadá do scikit-learn. Proto je obalen pomocí D2VVectorizeru, aby zapadl k dalším nástrojům pro vektorizaci, které jsou použity z scikit-learn.



Obrázek 4.5: Ilustrace toku dat při extrakci příznaků.

Další nástroje pro vektorizaci pochází z scikit-learn. Jako první uvedeme CountVectorizer, který odpovídá metodě Bag of Words. Pro TF-IDF systém podporuje TfidfVectorizer (je používán jako výchozí). Navíc oproti

návrhu jsem zařadil HashingVectorizer (používá hašování příznaků 2.2.3), který umožňuje nastavovat velikost výsledného vektoru (stejně tak u D2VVectorizeru). To u CountVectorizer a TfidfVectorizer nelze a může dojít k velkému nárůstu počtu dimenzí výsledného vektoru a s tím související navýšení paměťové náročnosti.

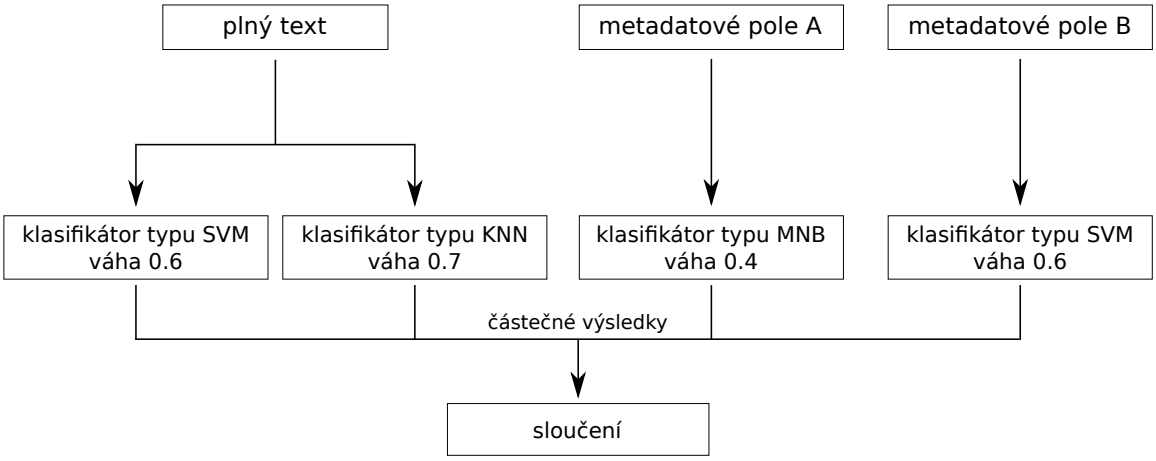
Systém umožňuje uživateli provést extrakci příznaků nezávisle na trénování samotného klasifikátoru. Díky tomu je možné z jedné sady extrahovaných příznaků trénovat více klasifikátorů, aniž by musel, každý z nich provést tuto fázi samostatně. Ovšem v implementovaném systému je možný i scénář, kdy uživatel nechce provádět tuto fázi samostatně a je možné ji uskutečnit jako součást trénování klasifikátoru.

4.1.5 Trénování klasifikátorů

Před samotnou klasifikací dokumentů je nutné natrénovat klasifikátor. Natrénovaný klasifikátor si uživatel uloží a použije jej pro klasifikaci. Při uživatelském výběru nástroje trénování klasifikátoru, systém neumožňuje použít okamžitě tento klasifikátor pro klasifikaci, ale musí být nejprve uložen a poté pomocí nástroje pro predikci cílů k dokumentům opět načten.

O záležitosti týkající se trénování klasifikátoru a predikování cílů dokumentů pomocí natrénovaného klasifikátoru se stará třída Classification.

V systému je umožněno trénovat jeden klasifikátor, či více klasifikátorů. Stejně tak je možné vybrat, která data/metadata budou používat jaký klasifikátor. Používáním je myšleno, že daný typ dat bude trénovat klasifikátor a posléze na něm bude tento druh dat klasifikován. Pro jeden druh dat je možné použít více klasifikátorů najednou. Každé kombinaci druhu dat a klasifikátoru, lze přiřadit váhu, s jakou vstupuje výsledek klasifikace do fáze, kdy se výsledky ze všech klasifikátorů slučují do finálních výsledků (více 2.4). Pro lepší orientaci vizte obrázek 4.6.



Obrázek 4.6: Ilustrace jakým způsobem lze přiřazovat klasifikátory k datům.

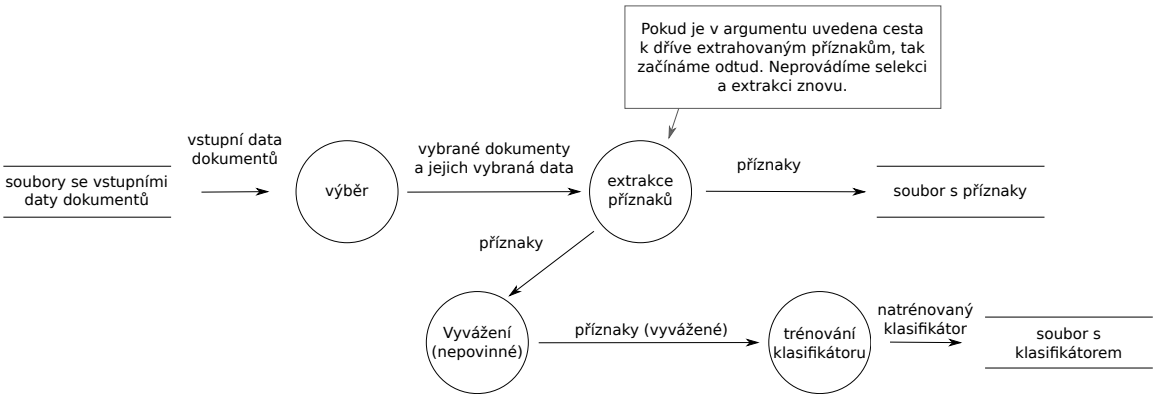
Teoreticky je možné přiřadit dva stejné klasifikátory k jedněm datům. Doporučuji ovšem spíše navýšit váhu daného klasifikátoru, přestože tuto kombinaci systém nezakazuje.

Uvedeme si výčet použitých klasifikátorů a zařadíme je do kontrastu s informaci v 2.3. Všechny zařazené klasifikátory jsou z scikit-learn. Jejich výčet je následující:

- Na bázi Support Vector Machines:
  - SVC s lineárním jádrem
  - LinearSVC
  - SGDClassifier
- Na bázi K nejbližších sousedů:
  - KNeighborsClassifier
- Na bázi Multinomiálního naivního Bayesova klasifikátoru:
  - MultinomialNB

Na bázi Support Vector Machines systém poskytuje tři klasifikátory: LinearSVC, SGDClassifier a SVC. Dle [8] a [9] je LinearSVC vhodnější/rychlejší než SVC pro větší datasety.

Jaká metoda pro slučování výsledků, z metod popsaných v 2.4, bude použita může uživatel určit v konfiguračním souboru. V základním nastavení je použita metoda, která pracuje s pravděpodobnostmi, do které kategorie dokument patří.



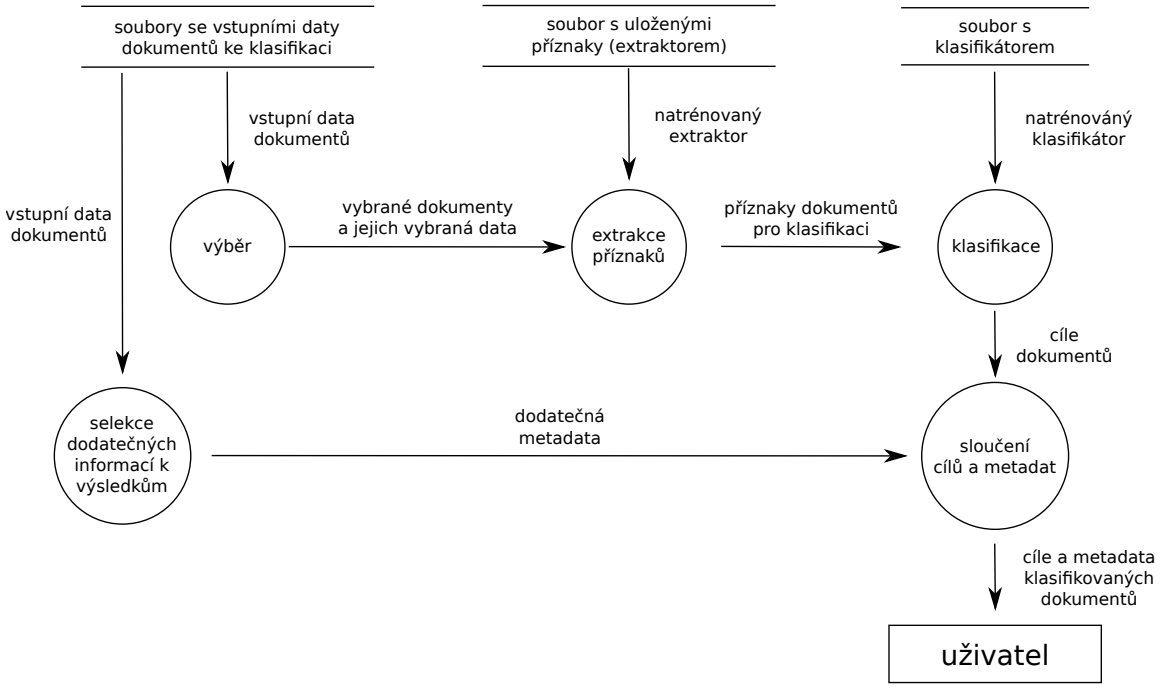
Obrázek 4.7: Ilustrace toku dat při trénování klasifikátoru. Povšimněte si, že pokud jsou zadány již extrahované příznaky, tak budou znovu uloženy, ale ovšem jako součást klasifikátoru.

4.1.6 Predikce

Systém umožňuje predikovat kategorie pomocí natrénovaného klasifikátoru (či více klasifikátorů) pro dokumenty. Je nutné předem natrénovat klasifikátor, uložit jej a poté předat opět systému. Ten jej použije pro klasifikaci vstupních dokumentů.

Lze získat až n cílů ke každému dokumentu seřazených podle toho, jak si je klasifikátor jistý, že daný dokument patří k danému cíli. Maximální n je dáno počtem prvků množiny cílů, které se vyskytovaly v trénovací množině.

Všechny použité klasifikátory, umožňují získat jednotlivé pravděpodobnosti ke každému cíli. Pokud však uživatel v konfiguračním souboru vypne získávání pravděpodobností z predikování, tak není možné vybrat n nejlepších cílů.



Obrázek 4.8: Ilustrace toku dat při klasifikování dokumentů. Diagram zobrazuje i tok dodatečných metadat (například název, id), která budou připojena k výsledkům.

Je také možné nechat si vypsat míru, jak moc si klasifikátor myslí, že dokument do daného cíle patří.

Samotnou predikci lze provést pomocí třídy Classification. Tato třída obsahuje metodu, která dokáže sama rozhodnout zda-li daná kombinace klasifikátorů umožňuje predikovat n cílů (usnadňuje rozšiřitelnost o další klasifikátory). Výsledky z klasifikace zpracuje třída Prediction. Zpracováním je myšleno zapsání dat do výstupního formátu. Umožňuje vypisovat k predikovaným cílům i další metadata dokumentu, která uživatel vybere. V praxi je tedy možné například k výsledkům přiřadit jméno dokumentu, či jiné informace.

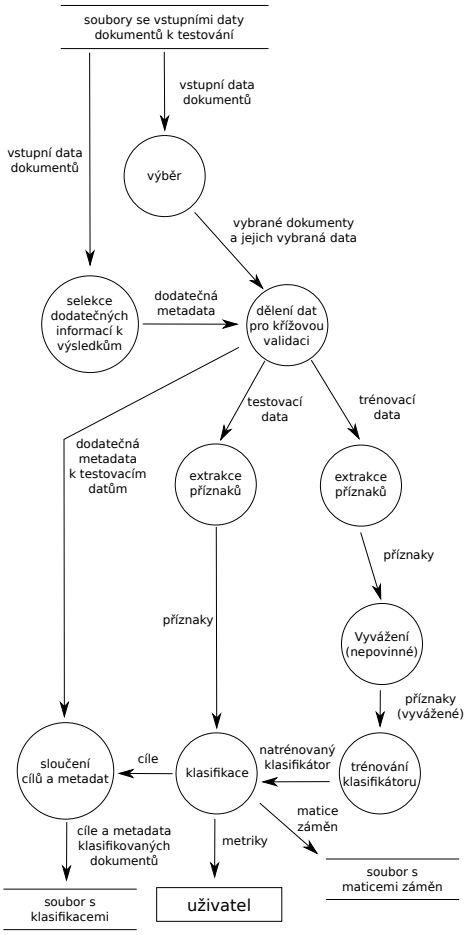
4.1.7 Testování klasifikátoru

Systém poskytuje nástroj pomocí, kterého lze testovat úspěšnost klasifikátoru. Nejedná se o testování klasifikátoru, který byl již natrénován a uložen. Jde spíše o testování klasifikátorů na trénovací/testovací množině dokumentů, kdy je klasifikátor na části dat natrénován a na druhé je testována jeho úspěšnost.

Testování se provádí s křížovou validací. Technik pro dělení dat do trénovací a testovací množiny je v systému více. Aktuální verze systému podporuje následující druhy získané z scikit-learn:

- LeaveOneOut
- KFold
- StratifiedKFold (zachovává procenta dokumentů v každé třídě)
- StratifiedShuffleSplit (Zachovává procenta dokumentů v každé třídě. Negarantuje, že každá část bude odlišná.)

Za každou iterací v průběhu křížové validace systém vypíše uživateli dosavadní zprůměrované výsledky. Testování může být někdy zdlouhavé a díky tomuto nemusíme čekat na provedení všech iterací, abychom si mohli udělat alespoň základní představu o tom jak úspěšný je náš klasifikátor.



Obrázek 4.9: Ilustrace toku dat při testování klasifikátoru. Zobrazen je jeden validační krok.

Metriky, které systém při testování na konci každého kroku vypíše, jsou shrnuty v následujícím seznamu:

- správnost
  - přímá shoda

    Za dobrý se považuje pouze přímá shoda cílů.
  - přímá rodina

    Bere za správné přímé potomky/rodiče pravého cíle a pravý cíl samotný.
  - rodina

    Za správné bereme i ty cíle, které padnou do množiny všech rodičů (i prarodičů a dále, obdobně i u potomků) nebo potomků pravého cíle, popřípadě bereme za správný i přímo pravý cíl samotný.
  - stejná nejvyšší úroveň

    Cíle mají stejný vrchní cíl v hierarchii.
- přesnost
- úplnost
- míra F_1

Přesnost, úplnost a míra F_1 jsou vypsány s mikro, makro a váženým průměrováním. Váhu určuje support dané třídy.

Uvedené správnosti, pokud je nastaveno predikovat n nejlepších cílů, jsou vypisovány postupně pro jednotlivé případy. Prvně se vypíší správnosti, kdy bereme za správný pouze nejlepší cíl. Dále bereme za správný jakýkoliv z dvou nejlepších cílů a tímto způsobem až do n.

Systém vždy uvede zprůměrované všechny uvedené metriky z jednotlivých validačních kroků, které doposud provedl.

Dále je pro každý cíl zvlášť vypsáno:

- přesnost
- úplnost
- míra F_1
- support (Určuje kolikrát se daný cíl vyskytuje v pravých cílech.)

- Počet dokumentů pro trénování.

Obdobně i zde se jedná o zprůměrované výsledky ze všech validačních kroků provedených před výpisem.

Systém také umožňuje vypisovat v jednotlivých krocích matici záměn a také výsledky predikcí. Výsledky predikcí se vypisují obdobně jako u 4.1.6. Systém navíc umožňuje přidat pole, kde bude uvedeno, zda-li se klasifikátor zmýlil nebo ne.

Všechny výše uvedené metriky jsou zpracovávány za pomocí třídy Testing. Pro vytváření trénovacích/testovacích množin slouží třída SplitTestSet.

Systém nabízí možnost natrénování klasifikátoru na všech datech a pak na těchto všech datech provést testování. Můžeme tímto způsobem zjistit, jak klasifikuje data, která zná. Lze takto nalézt třeba dokumenty, které by mohly být podezřelé ze špatně přidaného cíle. Například se mohlo stát, že byl dokument chybně manuálně označen.

4.1.8 Vyvažování nevyvážené trénovací sady

Systém poskytuje dvě metody pro vyvážení (náhodné podvzorkování, náhodné převzorkování), jejich princip je uveden v 2.5.

Možné je tyto metody i navzájem kombinovat. Pořadí v jakém se aplikují, se řídí pořadím metod zadaných v konfiguračním souboru.

O vyvažování se stará třída Balancing. Každá metoda vyvažování je reprezentována vlastní třídou. Takovéto třídy Balancing používá k vyvážení trénovací sady.

4.2 Použité nástroje a implementační jazyk

Tato sekce je zde zařazena, aby stručně seznámila čtenáře s použitými technologiemi. Některé byly již nastíněny v 3.2.3. Slouží také jako seznam externích nástrojů.

- Python
  - Použitým implementačním jazykem je Python 3 (konkrétně ve verzi 3.5.2). Dle [7] je Python univerzálním programovacím jazykem. Používá se v mnoha oborech a to včetně výuky programování.
  - Python má syntaxi, která přímo vnucuje programátorovi psaní formátovaného kódu. V jazyce jsou hojně používána anglická slova z běžné řeči. Svou syntaxí se jazyk snaží zvyšovat čitelnost kódu pro lidského čtenáře.
- scikit-learn
  - Jedná se open source nástroj pro strojové učení. Tento projekt jej aktuálně používá ve verzi 0.18.1. Dle [20] scikit-learn integruje široké množství nejmodernějších algoritmů strojového učení s učitelem či bez učitele. Zakládá si na snadném použití a výkonosti. Za použití jazyka python přináší strojové učení i pro ty, kteří nejsou specialisty v daném oboru.
- SciPy
  - Dle [10] je SciPy ekosystém open source softwaru pro matematiku, vědu a inženýrství. Systém jej nyní používá ve verzi 0.19.0.

- gensim
  - Tento nástroj používá aktuálně vytvořený systém ve verzi 1.0.1. V [21] je gensim charakterizován jako framework pro tématickou podobnost ve velkých korpusech.
- MorphoDiTa
  - Dle [19] je MorphoDiTa (Morphological Dictionary and Tagger) open source nástroj pro morfologickou analýzu přirozeného jazyka. V tomto projektu je používán pro lemmatizaci a určování slovních druhů slov.
- NumPy
  - Projekt používá NumPy ve verzi 1.12.1. Zdroj [5] uvádí, že NumPy je základním balíčkem pro vědecké výpočty v jazyce Python. Lze jej použít jako efektivní multidimenzionální kontejner pro obecná data, což umožňuje integraci s celou řadou databází.
- pandas
  - V [6] je pandas popisován jako open source knihovna poskytující vysoce výkonné, snadno použitelné datové struktury a nástroje pro datovou analýzu v jazyce python. V této práci je knihovna aktuálně používána ve verzi 0.19.2.
- Unidecode
  - Pro úplnost je zde uveden tento nástroj, který umožňuje překlad Unicode znaků do co nejbližší alternativy v ASCII. Je možné jej použít v rámci předzpracování vstupního textu. Používán je ve verzi 0.04.20.

4.3 Formát vstupních dat

Zde si popíšeme jaký formát vstupních dat systém očekává, jak vypadají výstupy a také si popíšeme jak se zadává konfigurace v konfiguračním souboru.

Pro některé vstupy/výstupy systém používá formát Comma-separated values (CSV), což v překladu v českém jazyce znamená: Hodnoty oddělené čárkami. Přesněji definovaný formát čtenář může nalézt v [23]. My jej zde nebudeme naplno definovat a spokojíme se s tím, že se jedná o formát kde:

- Každý záznam je oddělen koncem řádku.
- Hodnoty jsou oddělené čárkami.
- Pokud hodnota obsahuje čárku nebo znak konce řádku, tak je uzavřena do uvozovek.
- Pokud hodnota obsahuje uvozovky a je uzavřena do uvozovek, tak je uvozovka escapována pomocí předcházející uvozovky.

4.3.1 Data

Formát dat, nebo možná raději plných textů dokumentů, je prostý textový soubor kde je každý dokument na samostatném řádku. Slova v dokumentu jsou odděleny mezerou. Plný text dokumentu, který předkládáme systému obsahuje tedy pouze slova. Je nutné předtím odstranit případné konce řádků, formátovací znaky, či jiné hodnoty, které nejsou relevantní.

Protože neukládáme žádný identifikátor dokumentu, tak se souborem obsahujícím metadata existuje propojení na základě pozice řádků. To znamená, že například dokumentu na prvním řádku v souboru s plnými texty, přísluší metadat v metadatovém souboru v prvním záznamu. Všimněte si, že zde není uvedeno na prvním řádku, ale záznamu. CSV formát umožňuje mít záznam na více řádcích.

4.3.2 Metadata

Metadata poskytují informace o jiných datech. V našem případě se jedná o dokumenty (jejich plné texty). Metadata dokumentů jsou uložena v souboru s formátem CSV. Jakým způsobem jsou metadata propojena s plnými texty je popsáno v 4.3.1.

Metadatový soubor musí obsahovat CSV hlavičku, pro identifikaci polí. V konfiguračním soubor, poté uživatel nastavuje (popřípadě nechá výchozí hodnoty), které pole slouží jako cíl dokumentu. Tento cíl se pak použije pro natrénování klasifikátoru.

id	**název**	**kategorie**	**autor**	**klíčová slova**
123	R.U.R.	Beletrie->SCI-FI	Karel Čapek	robot$|$$drama
124	Kupec benátský	Beletrie->Komedie	William Shakespeare	Benátky$|$$drama

Tabulka 4.1: Ukázka metadat.

Tabulka 4.1 ukazuje příklad metadatového souboru. Na příkladu je vidět, že tabulka obsahuje položku id (jednoznačný identifikátor). Obecně řečeno je dobré, mít ve svém datasetu jednoznačný identifikátor dokumentu. Systém sice nenutí jeho použití, ale v případě, kdy je použito filtrování vstupních dat, tak již nesedí pozice vstupních dokumentů s pozicemi výsledků k danému dokumentu.

V tabulce si lze také povšimnout řetězce: ->. Jedná se o řetězec, který slouží k utváření hierarchie v cílech dokumentů. Tento řetězec je možné v konfiguračním souboru změnit na jiný.

Dalším řetězcem v příkladu je: $|$. Tento řetězec odděluje jednotlivé položky v poli. Opět jej lze změnit v konfiguračním souboru. V rámci metadatového pole je tedy možné mít více položek. Pokud je ovšem dané pole označeno jako cíl, tak je vybrána první položka. U jiných polí to však může být vhodné, jako na příkladu s klíčovými slovy. Zavedení položek v poli umožňuje tvořit n-gramy proměnlivé délky a to tak, že bereme položku jako celek.

4.3.3 Konfigurace

Konfigurační soubor používá strukturu, která se děli na sekce. Tyto sekce pak obsahují jednotlivé parametry. Sekce se uvozuje na samostatném řádku pomocí jejího názvu v hranatých závorkách []. Jednotlivé parametry jsou pak každý na samostatném řádku. Ve formátu:

NÁZEV=HODNOTA

Je povoleno uvádět více hodnotové parametry. K tomu se používá shell-like syntaxe, kdy jednotlivé hodnoty oddělujeme mezerou a pokud položka obsahuje mezeru tak celou položku dáme do uvozovek (lze uvozovky escapovat).

4.4 Výstupy systému

V této sekci si popíšeme v jaké podobě systém pro klasifikaci vypisuje výsledky a ukládá data pro další použití.

Každý nástroj v systému umožňuje ukládat informace o průběhu zpracování do souboru. Jedná se o takzvané logy či žurnály. Pomocí nich je možné sledovat v jaké fázi se nástroj nachází a také další informace týkající se aktuálního zpracování. Mezi tyto informace patří například: počet dokumentů, či velikost příznakového vektoru a další. Informace jsou vypisovány ve formátu:

datum a čas : druh : obsah

Následující seznam slouží pro objasnění výstupu z jednotlivých nástrojů:

- Výstupy z výběru dat
  - Po provedení výběru je možné uložení metadat ve formátu CSV do souboru jaký uživatel určí. Stejně tak je možné uložit plné texty dokumentů. Je zcela na uvážení uživatele, zda-li chce uložit oba druhy dat, či jen jeden z nich.
  - Mimo to je vypsán počet vybraných dokumentů a pokud je v konfiguračním souboru nastavený cíl dokumentu, tak vypíše i počty dokumentů pro dané cíle.
- Výstupy z předzpracování
  - Na standardní výstup jsou vypsány předzpracované dokumenty. Jeden dokument na řádek.
- Výstupy z extrakce příznaků
  - Z tohoto nástroje získáme a uložíme extraktor příznaků (celý objekt třídy Features je uložen), který bude použit v momentu, kdy budeme chtít klasifikovat neznámé dokumenty. Pomocí tohoto extraktoru z nich pak získáme příznaky, které použijeme pro klasifikátor.
  - Uložíme i samotné extrahované příznaky z trénovací množiny dokumentů.
  - Společně jsou ukládány, také data o použité konfiguraci a cílech.
  - Všechna tato data jsou ukládána samostatně do jednotlivých souborů, tedy samostatně je extraktor příznaků, příznaky a pak jeden soubor s konfigurací a cíli.
  - Ukládání realizuje DocClassifierDataDump, který ukládá přímo celé programové struktury. Používá k tomu joblib.dump z scikit-learn.
- Výstupy z trénování klasifikátoru
  - Tento nástroj ukládá všechna data jako u extrakce příznaků. Navíc je ještě ukládán natrénovaný klasifikátor (či více klasifikátorů), objekt třídy Classification, v samostatném souboru.

- Výstupy z predikce cílů k dokumentům
  - Na standardní výstup jsou ve formátu CSV vypisovány výsledky predikce. Jelikož systém pro klasifikaci umožňuje predikovat n nejlepších cílů pro dokument, je na výstupu až n cílů vypsaných každý ve svém samostatném poli. Název těchto polí je možné ovlivňovat nastavením v konfiguračním souboru.
  - V konfiguračním souboru, je také možné nastavit výpis hodnot, které značí míru s jakou si klasifikátor myslí, že daný dokument do dané kategorie patří. Tyto hodnoty, jsou opět vypisovány v samostatných polích s nastavitelným názvem.
- Výstupy z testování klasifikátoru
  - Je možné si z testování uložit predikované cíle z jednotlivých kroků křížové validace. Stejně tak je možné ukládat matici záměn.
  - Na standardní výstup jsou pak na konci každého kroku vypisovány statistiky zahrnující výsledky z doposud provedených kroků. Vypisuje metriky, které se vztahují pouze k jednotlivým kategoriím, a také metriky celkové.

4.5 Programová dokumentace

Tento text doplňuje programová dokumentace. K vytvoření programové dokumentace, byl použit nástroj Sphinx [11]. Dokumentace byla přímo vygenerována ze speciálního formátu komentářů ve zdrojovém textu programu. Nachází se na přiloženém elektronickém médiu.

Kapitola 5

Experimenty s klasifikátorem

V této kapitole provedeme několik experimentů s klasifikačním systémem. Budeme používat sadu s velkým objemem dat, řádově desítky GB. Jedná se o dokumenty (knihy), které byly převedeny z tištěné podoby do elektronické pomocí OCR. Nachází se tedy v textech dokumentů spoustu chyb vzniklých tímto typem převodu.

Mimo samotného plného textu obsahuje sada poměrně velké množství metadat. Mezi poskytnutá metadata patří například: název, autor, klíčová slova, MDT(mezinárodní desetinné třídění) či kategorie dokumentu. Kategorii dokumentu se budeme snažit pomocí systému získávat.

Neplatí však, že všechny typy dat jsou dostupné ke každému dokumentu. Budeme muset tedy dokumenty vybírat na základě dostupnosti daných dat, které budeme chtít použít pro trénování klasifikátoru a následnou klasifikaci.

Plné texty dokumentů budeme používat ve dvou podobách, lišících se předzpracováním. Vlastnosti a pojmenování, které budeme nadále používat, jsou uvedeny v následujícím seznamu:

- PLNÝ TEXT 1
  - Provedana lemmatizace.
  - Odstraněna stopslova.
  - Vybrány pouze slovní druhy: podstatná jména, přídavná jména a slovesa.
  - Odstraněna všechna slova s počtem znaků menším nebo rovným dvěma. Jak již bylo řečeno, tak plný text byl získán pomocí OCR. Tato metoda vnesla do textu spoustu nesmyslných znaků. Tímto krokem se snažíme takovéto znaky potlačit.
- PLNÝ TEXT 2
  - Tento plný text je pouze lemmatizován.

název	**počet dokumentů**	**průměrný počet slov na dokument**	**velikost v GB**
PLNÝ TEXT 1	118952	52034	51,6
PLNÝ TEXT 2	118952	94482	72,6

Tabulka 5.1: Statistiky souborů s plnými texty dokumentů.

Nejprve se budeme snažit provést klasifikaci pouze na základě fulltextu. Tuto část budeme moci porovnávat s nástrojem FastText [14]. Jedná se o nástroj od firmy Facebook, který je schopen provádět klasifikaci textových dokumentů.

V další části budeme přidávat do klasifikace metadata, abychom docílili zlepšení výsledku. V této části se již omezíme pouze na nástroj popsaný v této práci. FastText klasifikuje pouze na základě plných textů.

Testy budou prováděny na trénovací a testovací množině dat. Výsledky budou křížově validovány. Klasifikovat budeme dokumenty do kategorií, které jsou ve dvou úrovňové hierarchii. Použijeme metriky uvedené v 2.6. U správnosti se omezíme na pouze dva typy: na základě přesné shody a na základě stejné vrchní kategorie v hierarchii. Ostatní správnosti nejsou nutné, protože máme pouze dvě úrovně hierarchie.

V tabulkách s výsledky budeme používat následující značení a pojmy:

- D - Počet rozměrů příznakového vektoru.
- F_1 – míra F1
- P – přesnost
- R – úplnost
- A – Správnost na přímou shodu.
- A_T – Správnost na základě stejné vrchní kategorie v hierarchii.
- makro průměr – Metriky jsou počítány pro každou kategorii zvlášť a nakonec jsou zprůměrovány.
- vážený průměr – Metriky jsou počítány pro každou kategorii zvlášť a nakonec jsou zprůměrovány s váhou, která odpovídá počtu dokumentů označených v testovací množině, že patří do té konkrétní kategorie.

5.1 Klasifikace na základě plných textů

Zde se pokusíme klasifikovat dokumenty pouze na základě plného textu. Je třeba zmínit, že pracujeme s 292 kategoriemi, které jsou navíc velice nevyvážené co do počtu dokumentů. Nejmenší počet dokumentů u kategorie je roven 1 a největší 7977.

Pro účely testování není možné však kategorie s jednou položkou použít. Po odstraněni takovýchto kategorií dostáváme počet kategorií 283 a nové minimum dokumentů na kategorii jsou dva. Trénovací množina obsahuje 80% dokumentů a testovací množina obsahuje 20% dokumentů z celkového počtu dokumentů s přiřazenou kategorií (65611). Použil jsem metodu dělení, která zachovává procenta u jednotlivých kategorií.

Jak je možné vidět v tabulce 5.2, tak nejlépe si vedl HashingVectorizer (byl použit parametr non_negative=True). Doc2Vec si ovšem nevedl také špatně a budeme jej dále používat. Výstupy z Doc2Vec byly klasifikovány pomocí k nejbližších sousedů pro k rovno 3. Pro HashingVectorizer byl použit klasifikátor LinearSVC. LinearSVC byl vhodnější variantou než-li k nejbližších sousedů pro HashingVectorizer, naopak to platí pro Doc2Vec.

Snažil jsem se vylepšit výsledky FastTextu zvýšením počtu dimenzí příznakového vektoru, použitím dvojic slov a změnou ztrátové funkce. Jediné co pomohlo zlepšit výsledky bylo použití jiné ztrátové funkce (loss=hs).

	D	makro průměr			vážený průměr			správnost			
								1 nejlepší		2 nejlepší	
		F1	P	R	F1	P	R	A	AT	A	AT
FastText (loss=hs)	300	0,02	0,02	0,03	0,17	0,15	0,28	0,28	0,47	0,38	0,57
FastText (loss=hs)	500	0,2	0,02	0,03	0,17	0,15	0,28	0,28	0,47	0,38	0,57
FastText (loss=ns)	500	0,01	0,008	0,02	0,13	0,1	0,23	0,23	0,41	0,31	0,49
FastText (loss=hs, ngram=2)	500	0,01	0,01	0,02	0,09	0,09	0,19	0,19	0,38	0,27	0,44
Doc2Vec	500	0,38	0,45	0,36	0,56	0,57	0,56	0,56	0,71	0,68	0,82
HashingVectorizer	262144	0,52	0,59	0,5	0,68	0,69	0,69	0,69	0,80	0,82	0,9

Tabulka 5.2: Tabulka výsledků pro testování nad: PLNÝ TEXT 1. Všechny hodnoty uvedené v tabulce byly samostatně spočteny a až poté zaokrouhleny.

5.2 Klasifikace na základě plných textů a metadat

V této fázi se pokusíme výsledky zlepšit pomocí zařazení metadat. Metadata, která budeme používat jsou:

- Lemmatizované pole 245 (obsahuje název) [1].
- Zařazení v univerzálním klasifikačním systému MDT [4].

Ne všechny dokumenty s plným textem mají přiřazené MDT, avšak všechny s plným textem mají pole 245. Dokumentů s MDT je 101036. Z toho má přiřazenou kategorii 65452. Počet kategorií vhodných pro testování zůstal pro MDT stejný jako v případě plného textu (či pole 245).

Všechny druhy dat budou klasifikovány zvlášť a poté budou výsledky sloučeny pomocí metody z 2.4. Konkrétně se jedná o metodu, která zohledňuje míry jistoty. Váhy jednotlivých dat, jsou určeny dle správnosti jaké dosáhly při testování, pouze na základě daného druhu dat. Jedná se o správnost na přesnou shodu, kdy zohledňujeme pouze jednu predikovanou kategorii.

	makro průměr			vážený průměr			správnost			
							1 nejlepší		2 nejlepší	
	F1	P	R	F1	P	R	A	AT	A	AT
245	0,4	0,44	0,39	0,56	0,57	0,57	0,57	0,68	0,69	0,78
MDT	0,53	0,6	0,5	0,75	0,77	0,75	0,75	0,81	0,85	0,9
245 | MDT	0,52	0,61	0,48	0,75	0,75	0,76	0,76	0,82	0,85	0,9
PLNÝ TEXT 1 | 245	0,48	0,54	0,46	0,66	0,66	0,67	0,67	0,79	0,79	0,88
PLNÝ TEXT 1 | 245 | MDT	0,56	0,63	0,54	0,77	0,77	0,78	0,78	0,85	0,87	0,93
PLNÝ TEXT 2 | 245 | MDT	0,55	0,63	0,52	0,75	0,76	0,76	0,76	0,83	0,85	0,91

Tabulka 5.3: Tabulka znázorňuje výsledky pro plný text a metadata. Na metadata byl použit klasifikátor SVC a extraktor příznaků TfidfVectorizer. Pro plný text byl použit Doc2Vec a klasifikátor k (3) nejbližších sousedů. Všechny hodnoty uvedené v tabulce byly samostatně spočteny a až poté zaokrouhleny.

V tabulce 5.3 je vidět jak si vede plný text v kombinaci s metadaty. Metadata značně přispěla k úspěšnosti. Také je zde znázorněn vliv předzpracování u plných textů. U PLNÉHO TEXTU 1 dostáváme při stejném experimentu o něco málo lepší výsledky, než u PLNÉHO TEXTU 2, a to dokonce pracujeme s o hodně menším objemem dat.

Nyní si představíme experiment, kde budeme používat opět kombinaci dat, ale použijeme HashingVectorizer pro plný text a ještě aplikujeme metody pro vyvážení zastoupení kategorií. Použijeme náhodné podvzorkování a náhodné převzorkování. Těmito metodami nastavíme maximální možný počet dokumentů v kategorii na polovinu počtu dokumentů v nejpočetnější kategorii a minimální počet dokumentů v kategorii na 60 (30x nejméně početná kategorie).

	makro průměr			**vážený průměr**			**správnost**			
							1 nejlepší		**2 nejlepší**	
	F1	**P**	**R**	**F1**	**P**	**R**	**A**	**AT**	**A**	**AT**
plný text | 245	0,537	0,614	0,508	0,696	0,702	0,711	0,711	0,807	0,824	0,896
plný text | 245 | MDT	0,652	0,712	0,63	0,796	0,8	0,805	0,805	0,868	0,896	0,944
vyváženo (plný text | 245 | MDT)	0,67	0,716	0,657	0,805	0,808	0,813	0,813	0,874	0,903	0,949

Tabulka 5.4: Tabulka znázorňuje výsledky pro plný text a metadata. Na metadata byl použit klasifikátor linearSVC a extraktor příznaků TfidfVectorizer. Pro plný text byl použit HashingVectorizer a klasifikátor linearSVC. Všechny hodnoty uvedené v tabulce byly samostatně spočteny a až poté zaokrouhleny (tentokrát na 3 desetinná místa).

Hodnoty z tabulky 5.4 ukazují, že se podařilo zlepšit výsledky klasifikace, oproti výsledkům z tabulky 5.3, o několik málo procent. Nejlepší dosažený výsledek, jsme dostali při použití metod pro vyvážení zastoupení kategorií v našich trénovacích datech.

Jako poslední si uvedeme experiment, který pro natrénování a testování klasifikátoru použije stejnou množinu dat. Použijeme všechny dostupné dokumenty s přiřazenou kategorií. Pokusíme se zjistit, jak bude klasifikátor klasifikovat na datech, na kterých byl naučený.[1]

	makro průměr			**vážený průměr**			**správnost**	
	F1	**P**	**R**	**F1**	**P**	**R**	**A**	**AT**
vyváženo (plný text | 245 | MDT)	0,97	0,97	0,98	0,97	0,97	0,97	0,97	0,98

Tabulka 5.5: Tabulka znázorňuje výsledky pro test, kde trénovací a testovací množiny dokumentů jsou shodné. Nastavení je totožné jako v experimentu s tabulkou 5.4. Všechny hodnoty uvedené v tabulce byly samostatně spočteny a až poté zaokrouhleny.

Dle výsledků z tabulky 5.5 je vidět, že klasifikátor umí dobře rozřazovat dokumenty, na kterých byl naučen.

[1] Takovýto experiment, může sloužit pro odhalení špatně ručně označených dokumentů.

Kapitola 6

Závěr

V této práci jsem uvedl metody, které se v dnešních dnech používají pro klasifikaci textových dokumentů. Na základě těchto metod jsem poté navrhl systém pro klasifikaci dokumentů a implementoval jsem jej. Nástroj umožňuje jak samotnou klasifikaci, tak poskytuje prostředky pro: předzpracování textu, testování klasifikátoru, výběr dat a základní metody pro vyvažování nevyváženého datasetu.

Nástroj byl navrhnut tak, aby usnadňoval přidávání dalších funkcí. V budoucnu by bylo zejména vhodné jej obohatit o další metody pro vyvažování. Aktuálně jsou k dispozici pouze dvě základní metody. Dále by stálo za zvážení upravit konfiguraci nástroje. V konfiguračních souborech je spousta parametrů a možná by v tomto směru bylo vhodnější grafické uživatelské rozhraní.

Vytvořený systém jsem vyzkoušel v reálných podmínkách na obsáhlém datasetu s nevyváženými kategoriemi a texty obsahujícími chyby. Přesto se podařilo získat při testování uspokojivých výsledků.

Nástroj našel své praktické uplatnění a bude používán v rámci projektu NAKI CPK Moravskou zemskou knihovnou v Brně.

Literatura

[1] *245 - Title Statement (NR)*. [Online; navštíveno 11.05.2017].
URL https://www.loc.gov/marc/bibliographic/bd245.html

[2] *Centrum zpracování přirozeného jazyka FI MU, Stop List*. [Online; navštíveno 15.04.2017].
URL https://nlp.fi.muni.cz/cs/StopList

[3] *How to handle Imbalanced Classification Problems in machine learning?* [Online; navštíveno 10.05.2017].
URL https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/

[4] *MDT*. [Online; navštíveno 11.05.2017].
URL https://www.nkp.cz/o-knihovne/odborne-cinnosti/zpracovani-fondu/informativni-materialy/mdt-www

[5] *NumPy*. [Online; navštíveno 3.05.2017].
URL http://www.numpy.org/

[6] *pandas*. [Online; navštíveno 3.05.2017].
URL http://pandas.pydata.org/

[7] *Python, programovací jazyk*. [Online; navštíveno 3.05.2017].
URL https://python.cz/

[8] *scikit-learn LinearSVC*. [Online; navštíveno 28.04.2017].
URL http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[9] *scikit-learn SVC*. [Online; navštíveno 28.04.2017].
URL http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[10] *SciPy*. [Online; navštíveno 25.05.2017].
URL https://www.scipy.org/

[11] *Sphinx*. [Online; navštíveno 5.05.2017].
URL http://www.sphinx-doc.org/en/stable/

[12] *Vector Representations of Words*. [Online; navštíveno 19.04.2017].
URL https://www.tensorflow.org/tutorials/word2vec

[13] Helena, K.: *Česká terminologická databáze knihovnictví a informační vědy, výklad pojmu umělá inteligence.* [Online; navštíveno 31.03.2017].
URL
http://aleph.nkp.cz/F/?func=direct&doc_number=000000137&local_base=KTD

[14] Joulin, A.; Grave, E.; Bojanowski, P.; aj.: Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759*, 2016.

[15] Le, Q.; Mikolov, T.: Distributed Representations of Sentences and Documents. *ArXiv e-prints*, Květen 2014, 1405.4053.

[16] Manning, C. D.; Raghavan, P.; Schütze, H.: *Introduction to information retrieval.* Cambridge University Press, 2008, ISBN 0521865719.

[17] McGinnis, W.: *Useful Data Science: Feature Hashing.* [Online; navštíveno 12.05.2017].
URL http:
//www.kdnuggets.com/2016/01/useful-data-science-feature-hashing.html

[18] Mikolov, T.; Chen, K.; Corrado, G.; aj.: Distributed Representations of Words and Phrases and their Compositionality. *ArXiv e-prints*, Říjen 2013, 1310.4546.

[19] Milan Straka, J. S.: *MorphoDiTa.* [Online; navštíveno 27.04.2017].
URL http://ufal.mff.cuni.cz/morphodita

[20] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; aj.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, ročník 12, 2011: s. 2825–2830.

[21] Řehůřek, R.; Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta: ELRA, Květen 2010, s. 45–50,
http://is.muni.cz/publication/884893/en.

[22] Schneider, J.: *Cross Validation - Carnegie Mellon School of Computer Science.* [Online; navštíveno 24.04.2017].
URL http://www.cs.cmu.edu/~schneide/tut5/node42.html

[23] Shafranovich, Y.: Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180, RFC Editor, October 2005,
http://www.rfc-editor.org/rfc/rfc4180.txt.
URL http://www.rfc-editor.org/rfc/rfc4180.txt

[24] Straková, J.; Straka, M.; Hajič, J.: *Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In* Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, *Baltimore, Maryland: Association for Computational Linguistics, June 2014, s. 13–18.*
URL http://www.aclweb.org/anthology/P/P14/P14-5003.pdf