VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MULTI-LABEL KLASIFIKACE TEXTOVÝCH DOKUMENTŮ

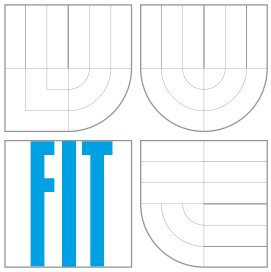
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR PRŮŠA

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MULTI-LABEL KLASIFIKACE TEXTOVÝCH DOKUMENTŮ

MULTI-LABEL CLASSIFICATION OF TEXT DOCUMENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR PRŮŠA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2012

Abstrakt

Diplomová práce se zabývá automatickou klasifikací textových dokumentů. Jsou zde vysvětleny základní pojmy a problémy dolování z textu. Práce vysvětluje pojem shlukování a ukazuje několik základních algoritmů shlukování. Je zde ukázáno i několik metod klasifikace a podrobně rozebrána vybraná metoda matrix regression. Dále byla navrhnuta a implementována aplikace používající ke klasifikace matrix regression. Provedené experimenty byly zaměřeny na normalizaci a prahování.

Abstract

The master's thesis deals with automatic classification of text document. It explains basic terms and problems of text mining. The thesis explains term clustering and shows some basic clustering algoritms. The thesis also shows some methods of classification and deals with matrix regression closely. Application using matrix regression for classification was designed and developed. Experiments were focused on normalization and thresholding.

Klíčová slova

Dolování z dat, dolování z textu, klasifikace, matrix regression, textový dokument, shlukování, strojové učení, prahování, normalizace, TF-IDF.

Keywords

Data mining, text mining, classification, matrix regression, text dokument, clustering, machine learning, thresholding, normalization, TF-IDF.

Citace

Petr Průša: Multi-label klasifikace textových dokumentů, diplomová práce, Brno, FIT VUT v Brně, 2012

Multi-label klasifikace textových dokumentů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka, Ph.D.

.....
Petr Průša
22. května 2012

Poděkování

Chtěl bych poděkovat vedoucímu Ing. Vladimíru Bartíkovi, Ph.D. za jeho odbornou pomoc při zpracování diplomové práce.

Obsah

Kapitola 1

Úvod

S rozvojem internetu jsme zahlceni velkým množstvím dokumentů, ze kterých je složité vybrat si článek odpovídající našemu zájmu. Proto se stále více pozornosti věnuje automatické klasifikaci textu. Cílem je, aby člověk nemusel procházet postupně jeden článek za druhým a zjišťovat, jestli je jeho obsah pro něj zajímavý či ne. Automatická klasifikace by měla umět roztřídit dokumenty do předem definovaných kategorií. Uživatel si následně zvolí určitou kategorii a může ihned procházet pro něj zajímavé materiály.

Multi-label klasifikace je klasifikace, při které může být jeden dokument zařazen do jedné i více tříd. Opakem je single-label klasifikace, kde dokument zařadíme pouze do jedné třídy. Dalším speciálním případem single-label klasifikace je binární klasifikace, kde se rozhodujeme pouze mezi dvěmi třídami.

Vyhodnocení multi-label klasifikace je podstatně složitější úkol než vyhodnocení single-label klasifikace. U single-label klasifikace pouze zjistíme, která kategorie vrací pro daný dokument nejlepší výsledek a tou ho označíme. Zatímco při multi-label klasifikaci se snažíme najít všechny třídy, do kterých dokument patří, a proto musíme být schopni určit hranici, kdy ještě dokument do třídy patří a kdy už ne. Ideální multi-label klasifikátor by tedy měl být schopný najít všechny kategorie relevantní pro daný dokument. V praxi zatím takový klasifikátor neexistuje a jeho vytvoření je velice naročné, ne-li nemožné.

Tato práce se zabývá multi-label klasifikací pomocí relativně nové metody matrix regression a navrhuje možná řešení vyhodnocení výsledků tohoto algoritmu.

Kapitola 2 nás uvádí do tématiky dolování z dat. Vysvětluje základní pojmy text miningu a jejich využití ve vztahu k dolování. Dozvíme se v ní také, jaké informace se snažíme z textu získávat. V kapitole 2 dále rozebereme, co je to kolekce dokumentů a jaké jsou základní rysy dokumentu, které používáme při text miningu. Je zde uvedena také typická architektura dolovacích systémů. Na konci kapitoly je popsáno shlukování a několik jeho metod, které se používají při dolování z textu.

Ve třetí kapitole se zaměřujeme na reprezentaci dokumentu, která je vhodná pro klasifikaci. Uvádíme v ní krátký popis reuters kolekcí, které budou využity jako vstupní data pro naše experimenty. Dále se zde věnujeme předzpracování dokumentu pomocí lematizace, stematizace a extrakce stop slov. V této kapitole se také seznámíme s pojmem vektor rysů, který se používá jako základní reprezentace dokumentu při klasifikaci a s ohodnocováním jeho prvků pomocí vah TF-IDF.

Čtvrtá část této práce pojednává o klasifikaci a jejím vztahu ke strojovému učení. Jsou v ní popsány vybrané metody klasifikace. Ve čtvrté části je rovněž rozebrána zvolená metoda matrix regression, která byla v rámci této práce implementována.

Pátá kapitola se věnuje samotné multi-label klasifikaci. Popisuje její vztah k single-label

klasifikaci a uvádí rozdíly mezi jejich vyhodnocováním. Je zde popsáno prahování a jedna jeho aplikace pro vyhodnocení klasifikace.

Kapitola 6 obsahuje zadání implementovaného programu. Jsou v ní uvedeny základní požadavky na aplikaci a je zde popsán formát vstupních dat, které budou zpracovávány. Diagram případů použití na konci této části přehledně zobrazuje funkční požadavky na naší aplikaci.

Sedmá část této práce se věnuje návrhu a popisu implementace naší aplikace. Popisujeme v ní námi vybranou architekturu MVC a pomocí diagramu tříd ukazujeme, jak budou jednotlivé vrstvy architektury implementovány na úrovni tříd. Jsou zde také okomentovány nejdůležitější metody jednotlivých typů objektů. Konec kapitoly se zaobírá ověřením funkčnosti naší aplikace.

V osmé kapitole rozebíráme detailněji váhy TF-IDF. Popisujeme data vybraná z reuters kolekcí pro naše experimenty. Následně zjišťujeme, jakých hodnot mohou váhy nabývat a pokoušíme se o jejich normalizaci různými způsoby. Pro získané hodnoty navrhujeme vhodné prahy a vyčíslujeme úspěšnost a chybovost klasifikace. V posledním experimentu se pokoušíme o prahování na základě nejvyšší váhy v kategorii.

V poslední kapitole se nachází závěrečné shrnutí této práce. Jsou zde zhodnoceny dosažené výsledky a navrhnut další postup, jak je do budoucna vylepšit.

Kapitola 2

Text mining

Text mining česky dolování z textu je obor zabývající se získáváním informací z textových dokumentů. Pracuje se zde s kolekcí dokumentů, ve kterých se snažíme najít užitečné informace pomocí identifikace a objevování zajímavých vzorů. Text mining má hodně společných rysů s data miningem. Pro text mining se používá podobná systémová architektura jako pro data mining. Je složena z předzpracování dat, algoritmů pro získání vzorů a nakonec z prezentace, například v podobě GUI, se kterým pracuje koncový uživatel. Hlavní rozdíl nastává ve fázi předzpracování, která u dat většinou spočívá v čištění a normalizaci, zatímco u textu musíme identifikovat a extrahovat rysy přirozeného jazyka, čímž získáme data s o něco lepší explicitní strukturou, avšak tato data stále ještě nevyhovují většině systémů pro dolování z dat.

2.1 Přehled znalostí získávaných z textu

Nejčastěji se dolování z textu používá pro tyto účely:

- Sumarizace textu – snaha získat abstrakt, který by co nejlépe charakterizoval zkoumaný dokument.
- Vyhledávání dokumentů – cílem je získat z dokumentu metadata, podle kterých by bylo možné najít podobné dokumenty. Takovými metadaty jsou např. nadpis, autor, klíčová slova.
- Kategorizace textu – roztřídění dokumentů do kategorií na základě podobnosti jejich obsahu. Podrobněji se tomuto tématu věnuje kapitola 4.
- Identifikace jazyka – zjišťujeme, v jakém jazyce je článek napsán.
- Zjištění autora – pokud je autor textu neznámý, lze ho zjistit na základě slovní zásoby, kterou používá.
- Identifikování klíčových slov – snaha najít slova charakterizující dokument.
- Vyhledávání strukturovaných dat – získání dat určitého formátu např. telefonní číslo, poštovní adresa, internetová adresa atd.
- Nalezení entit v dokumentu – hledáme ustálená slovní spojení. Příkladem mohou být názvy firem, Generals Motors, Dr. Pepper.

2.2 Kolekce dokumentů

Teoreticky kolekcí dokumentů můžeme rozumět jakékoliv seskupení textových dokumentů. Prakticky se při dolování z textu většinou zaměřujeme na velké kolekce, ze kterých získáváme zajímavé vzory. Kolekce dokumentů mohou být statické nebo dynamické. Dynamické se mění v čase přidáním nových dokumentů nebo úpravou současných. Statické kolekce se od inicializace nemění.

2.3 Dokument

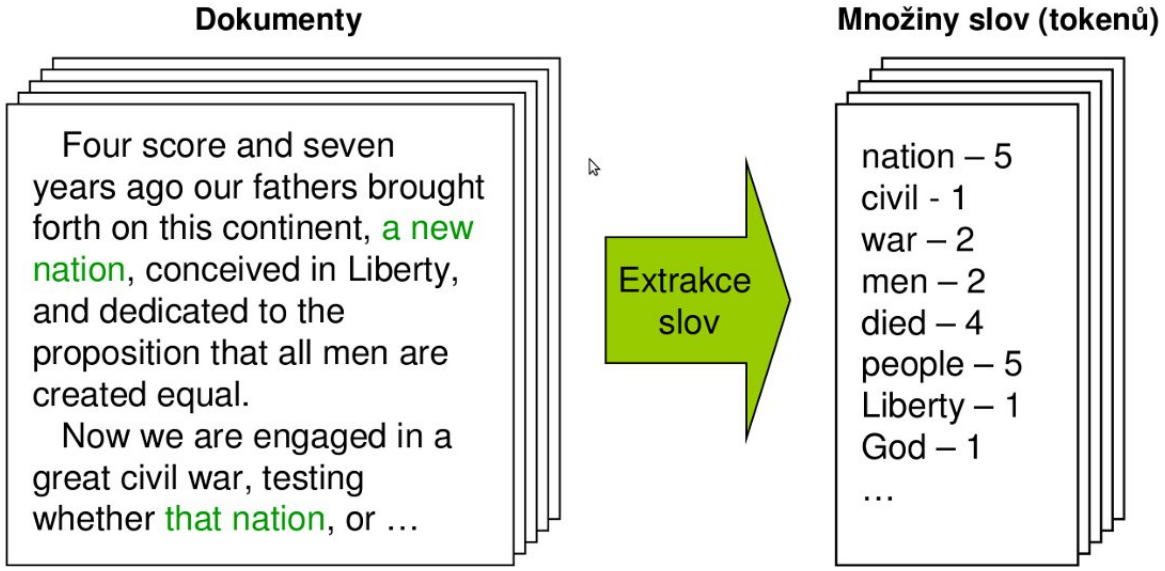
Dalším základním elementem v text miningu je dokument. Dokument můžeme neformálně definovat jako jednotku diskrétních textových dat v kolekci, která obvykle odpovídá skutečnému textovému dokumentu, e-mail, obchodní zpráva, novinový článek atd. Jeden dokument může ve stejný čas patřit do několika různých kolekcí. Textový dokument na první pohled vypadá jako nestrukturovaná data, ale bílé znaky, interpunkce, odstavce, zvýraznění a další prvky dávají dokumentu jistou strukturu. Navíc velká část sémantické a syntaktické struktury je implicitně ukryta v textovém obsahu.

Avšak dokument i přes svou implicitní strukturu je pro dolování nepoužitelný a je třeba získat jeho reprezentaci, ze které lze dolovat. Ta může být založena např. na jednotlivých slovech, slovních spojení, větách. Problémem je, že takováto reprezentace dává opravdu velkou množinu rysů, navíc některé elementy mohou mít více významů podle kontextu. Proto se dokument při předzpracování převádí na tzv. reprezentační model s explicitní strukturou, což je množina rysů charakterizující dokument jako celek.

Při výběru rysů musíme volit kompromis mezi dvěma cíli. Prvním cílem je získat takové rysy, které co nejvíce vystihují smysl dokumentu. To většinou vede k vybrání a extrakci více rysů. Druhým cílem je získání rysů, které jsou nejvíce výpočetně efektivní a praktické pro objevování vzorů. Těchto rysů je méně, ale jsou sémanticky významnější. Nejčastěji používané dokumentové rysy jsou:

- Znaky – zahrnují písmena, číslice, speciální znaky a mezery. Jsou základem pro stavbu dalších rysů, jako jsou slova, termy a koncepty. Jsou to však velice nízkoúrovňové rysy a s jejich použitím v text miningu se setkáváme zřídka.
- Slova – jsou základní úrovní sémantiky. Dokument může být reprezentován všemi slovy, která obsahuje. To je samozřejmě nepraktické z hlediska rozsáhlosti množiny rysů, proto se používají různé optimalizace, které odstraňují všechna z hlediska text miningu bezvýznamná slova.
- Termy – jsou slova nebo slovní spojení vybraná přímo z dokumentu. Reprezentace založená na termech se skládá z podmnožiny termů nalézajících se v dokumentu. Ke správnému nalezení termů se většinou používají slovníky. Tento druh reprezentace má vyšší sémantický význam než slovní nebo znaková reprezentace.
- Koncepty – mohou být sestaveny manuálně nebo pomocí metod předzpracování. Jedná se o slova a slovní spojení, která nejsou přímo obsažena v textu, avšak jsou pro dané téma relevantní.

Na obrázku 2.1 je zobrazen převod dokumentů na rysy. V tomto případě jsou rysy slova.



Obrázek 2.1: Ukázka extrakce slov z dokumentu (převzato z [1]).

2.4 Architektura systému pro dolování z textu

Typický systém pro dolování z textu se skládá z následujících částí:

- Předzpracování – připravuje data pro hlavní operace dolování. Převádí je z jejich původní podoby do reprezentace vhodné pro metody extrahující rysy.
- Jádro dolování – je základem text minigu. Obsahuje algoritmy pro objevování vzorů, analýzu trendů a získávání znalostí.
- Prezentace – zpravidla obsahuje GUI, které umožnuje prohlížení nalezených vzorů a kladení dotazů.
- Zpřesňující techniky – obsahují metody filtrující redundantní informace, provádějící čištění, řazení a optimalizaci.

Na obrázku 2.2 je zobrazena typická architektura systému pro dolování z textu.

Obrázek 2.2: Ukázka typické architektury systému pro dolování z textu (převzato z [7]).

2.5 Shlukování

Jednou ze základních metodik text miningu je shlukování. Jedná se o učení bez učitele, které rozdělí objekty do skupin nazývaných shluky. Myšlenkou shlukování je uspořádat data do shluků tak, aby v jednom shluku byla co nejvíce si podobná data a zároveň se lišila od dat v ostatních shlucích. Se shlukováním se můžeme setkat u nástrojů pro kompresi dat, segmentaci dat, zpracování obrazů, rozpoznání vzorů, zpracování dat a u mnoha dalších případů, včetně klasifikace textových dokumentů.

2.5.1 Kvalita shlukovacího algoritmu

Jedním z velkých problémů shlukování je určení kvality. Optimální algoritmus vytvoří takové shluky, jejichž objekty si jsou co nejvíce podobné a přitom jsou odlišné od objektů jiných shluků. Pro zjištění kvality takového algoritmu potřebujeme pracovat s nějakou hodnotou, která bude reprezentovat míru podobnosti mezi jednotlivými objekty. Tuto hodnotu podobnosti je nutné získat s ohledem na reprezentaci dokumentu. Nejčastěji používanou reprezentací je vektor rysů, a proto jsou většinou funkce pro určení podobnosti založeny na základě vzdálenosti vektorů.

Metriky podobnosti

Nejpopulárnější metrikou je Euklidovská vzdálenost:

$$D(x_i, x_j) = sqrt\sum_k (x_{ik} - x_{jk})^2, \tag{2.1}$$

která je speciálním případem Minkowského metriky pro $p = 2$:

$$D_p(x_i, x_j) = (\sum_k (x_{ik} - x_{jk})^p)^{1/p}. \tag{2.2}$$

Pro textové dokumenty se nejčastěji používá cosinova podobnost:

$$Sim(x_i, x_j) = (x'_i * x'_j) = \sum_k x'_{ik} * x'_{jk}. \tag{2.3}$$

2.5.2 Algoritmy shlukování

Existují různé způsoby dělení shlukovacích algoritmů. Jednou z možností je dělení na hierarchické a ploché algoritmy. Ploché algoritmy řadí objekty do vzájemně disjunktních jednoduchých shluků, naproti tomu hierarchické shlukování produkuje více vnořených shluků. Shlukování také může být *hard* nebo *soft*, kde *hard* znamená, že objekty mohou patřit pouze do jedné skupiny, zatímco u *soft* je možné zařadit jeden objekt do více shluků.

Optimalizace shlukovaní je velmi výpočetně náročná, proto se používají různé druhy aproximačních algoritmů. Mezi ně patří například *agglomerative* algoritmus, který na začátku umístí každý objekt do jednoho shluku a následně spojuje shluky, dokud není splněna podmínka pro zastavení. Další možností jsou dělící algoritmy, které naopak zařadí všechny objekty do jednoho shluku a ten pak následně dělí, dokud není splněna podmínka ukončení. Poslední možností jsou *shuffling* algoritmy, které iterativně přemísťují objekty mezi shluky.

2.5.3 Některé algoritmy shlukování

Základními algoritmy shlukování jsou k-means, EM-based pravděpodobnostní shlukování a hierarchický aglomerativní clustering (HAC), kterým se budeme více věnovat v následujícím textu.

Algoritmus k-means

Metoda k-means dělí kolekci vektorů $\{x_1, x_2, \ldots, x_n\}$ na množinu shluků $\{C_1, C2, \ldots, C_n\}$. Každý shluk je inicializován jedním semínkem, jehož hodnota je buď předem určena, anebo náhodně přiřazena. Pak se iterativně vypočítává centroid každého shluku podle vzorce:

$$M_i = |C_i|^{-1} \sum_{x \in C_i} x.$$

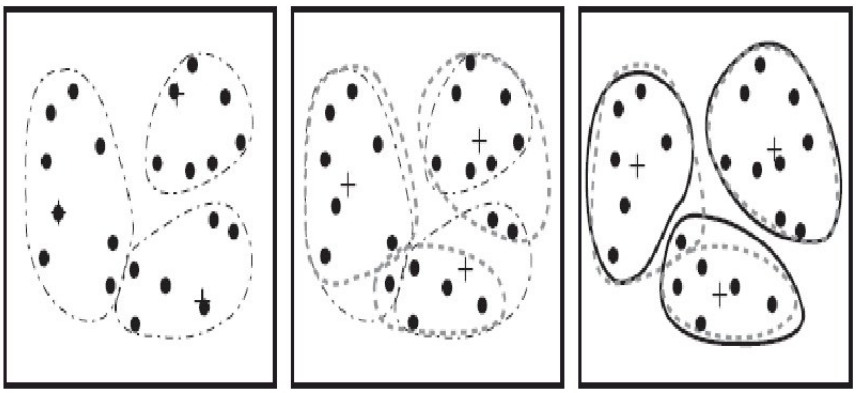
Následně se vektory přeřadí do shluků s nejbližším centriodem. Tento postup se iterativně opakuje, dokud přináší změny ve shlucích. K-means maximalizuje funkci kvality shlukování Q:

$$Q(C_1, C_2, \ldots, C_k) = \sum C_1 \sum_{x \in C_i} Sim(x - M_i).$$

Po každé iteraci měnící zařazení ve shlucích se hodnota funkce Q zvyšuje.

K-means algoritmus je často používán pro svou jednoduchost a efektivitu. Složitost jedné iterace je $O(kn)$ a počet nutných iterací bývá obvykle malý. Jeho nevýhodou je citlivost na kvalitě určení počátečních semínek, šum a odlehlé hodnoty. Také jím nelze nalézt shluky různých velikostí a nekonvexního tvaru.

Na obrázku 2.3 vidíme shluky vytvořené algoritmem k-means.



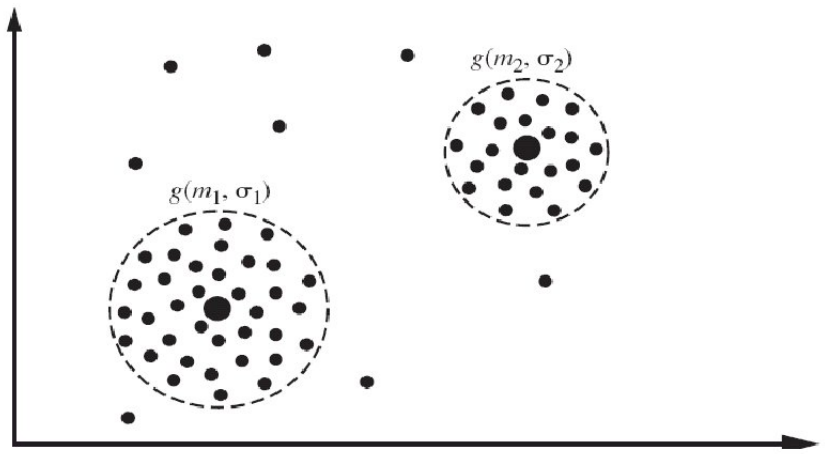
Obrázek 2.3: Ukázka algoritmu k-means (převzato z [3]).

Algoritmus pravděpodobnostního shlukování na základě maximalizace očekávání (EM-base probabilistic clustering algorithm)

Algoritmus pravděpodobnostního shlukování na základě maximalizace očekávání předpokládá, že objekty pro shlukování jsou složeny z k distribucí. Cílem je vypočítat pravděpodobnost $P(C_i|x)$, že daný objekt patří do shluku C. Účelem maximalizace očekávání je určit parametry distribuce ve zkoumaných datech.

Počáteční parametry distribucí jsou buď zadány externě, nebo náhodně zvoleny. Pak se vypočítá pravděpodobnost $P(C_i|x)$ pro všechny objekty x a objekty se přeznačí podle vypočtených pravděpodobností. Dalším krokem je přeodhadnutí parametrů distribuce tak, aby se maximalizovala věrohodnost označení objektů. Algoritmus končí, když každá další iterace poskytuje minimální změny.

Na obrázu 2.4 vidíme shluky určené algoritmem pravděpodobnostního shlukování na základě maximalizace očekávání.



Obrázek 2.4: Ukázka algoritmu pravděpodobnostního shlukování na základě maximalizace očekávání (převzato z [3]).

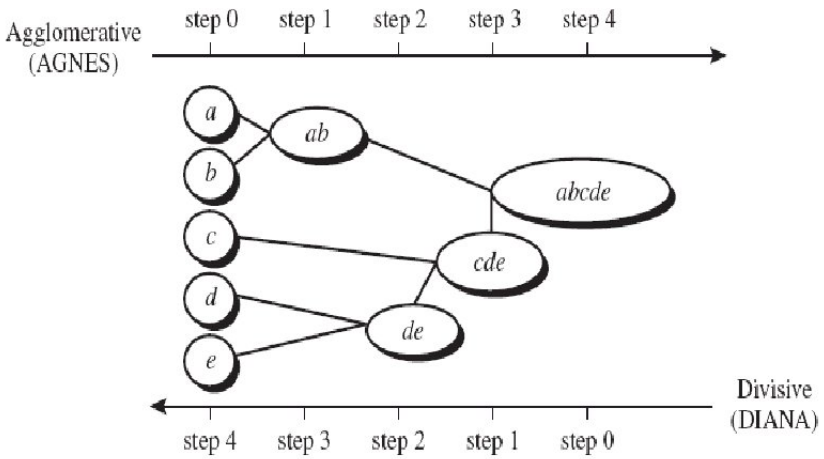
Hierarchické aglomerativní shlukování (HAC)

Hierarchické aglomerativní shlukování začíná tím, že každý objekt reprezentuje jeden shluk. Algoritmus postupně podle zadaných kritérií spojuje nejpodobnější shluky a skončí ve chvíli, kdy spojí všechny objekty do jednoho shluku. Výsledkem je binární strom, který znázorňuje proběhlé spojování shluků. Složitost HAC je $O(n^2 s)$, kde n je počet objektů a s je složitost výpočtu podobnosti.

Různé verze HAC se od sebe liší výpočtem podobnosti jednotlivých shluků. Může se jednat o single-link metody, kde se podobnost dvou shluků počítá jako maximální podobnost mezi dvěma objekty. Výsledkem pak bývají tenké dlouhé shluky. Opakem single-link je metoda complete-link, která vyjadřuje podobnost jako minimum podobnosti dvojic objektů. Jejím výsledkem bývají pevné kompaktní shluky. Complete-link metoda většinou dává lepší výsledky než single-link.

Na obrázku 2.5 vidíme postup hierarchického shlukování.

V této kapitole bylo čerpáno z [10], [3], [1] a [7].



Obrázek 2.5: Ukázka hierarchického shlukování (převzato z [3]).

Kapitola 3

Reprezentace dokumentu

Jak již bylo řečeno textový dokument ve své původní podobě není vhodný pro metody klasifikace. Proto při předzpracování dokument převedeme na tzv. vektor rysů, který je pro naše účely podstatně vhodnější. Nejběžnější praxí je použít každé slovo z předzpracovaného dokumentu jako rys. Potom vektor rysů má stejnou dimenzi, rovnající se počtu různých slov upraveného dokumentu.

3.1 Reuters kolekce

Velmi často používaným zdrojem dat pro klasifikaci jsou reuters kolekce. Ty byly původně sesbírány a roztříděny společnostmi Carnegie Group, Inc. a Reuters, Ltd. za účelem vývoje systému CONSTRUE pro kategorizaci textu a jsou volně šiřitelné pro vědecké účely. Obsah reuters kolekcí je ve formátu SGML.

Pomocí tagů SGML jsou u jednotlivých článků přidány informace užitečné pro klasifikaci. V kolekci jsou mimo jiné vyznačeny kategorie podle různých třídících kritérií, do kterých článek patří, např. podle zemí, organizací a pro nás nejdůležitějších témat článku. Jeden reuters soubor obsahuje více článků. Při předzpracování reuters dokumentu je tedy nutné získat z příslusných tagů jeho kategorii, nadpis a samotný obsah článku a odstranit veškeré SGML tagy.

Náš program bude testován na kolekci *Reuters-21578*. Ta obsahuje 21 souborů po tisíci dokumentech a jeden soubor s 578 dokuemty. Dohromady tedy máme k dispozici 21 578 článků. Avšak jisté množství dokumentů nemá přiřazené kategorie.

Kolekce obsahuje pět množin kategorií, kterými jsou *exchanges, orgs, people, place, topics*. Pro naši klasifikaci budeme používat kategorie z množiny *topics*, ve které jsou dokumenty roztříděny podle ekonomických témat. V množině *topics* je 135 možných kategorií, ale pouze 120 z nich má v kolekci alespoň jeden výskyt. Více jak 20 výskytů má pouze 57 kategorií z *topics*. Právě počet dokumentů v tématu má velký vliv na kvalitě klasifikace. Máme-li v kategorii větší množství dokumentů, jsme schopni pro ni získat lepší charakteristiku a tím zpřesnit kategorizaci.

Kategorie v množině *topics* jsou dále roztříděné do následujících pěti témat: *commodity, subject, ecnomic indicator, corporate, energy*. V tabulce 3.1 lze vidět několik zástupců každého z témat. Zatímco témata *subject* a *corporate* nemají ani pět zástupců, zbylá tři témata obsahují podstatně více kategorií, než je zobrazeno v tabulce 3.1.

Informace o reuters kolekcích byly čerpány z [8], [12].

commodity	grain	cocoa	corn	rubber
subject	foreign exchange	shipping	interest rates	
economic ind.	retail sales	trade	unemployment	money supply
corporate	earnings	acquisitons		
energy	crude oil	jet and keresone	gasoline	fuel oil

Tabulka 3.1: Ukázka kategorií v tématech.

3.2 Předzpracování dokumentu

Před klasifikací dokumentů je důležité provést jejich předzpracování. Předzpracování upraví dokument tak, aby v něm zbyla pouze klíčová slova, tj. slova, která nesou nějakou významovou informaci pro klasifikaci. Tato slova jsou dále upravena tzv. stematizací nebo lematizací, která hledá základní tvar slova. Z takto předzpracovaného dokumentu již můžeme vytvořit vektor rysů, který lze použít ke klasifikaci, nebo může být dále optimalizován pomocí redukce rysů.

Cílem předzpracování je tedy snížit paměťovou náročnost klasifikace a zvýšit její rychlost a přesnost.

3.2.1 Stematizace a lematizace

Pro snížení počtů klíčových slov se používá stematizace a lematizace. Obě metody mají stejný cíl a tím je najít jeden vhodný základ slova, na který by se převáděla všechna slova odvozená od daného kořene.

Stematizace zpravidla spočívá v tom, že ze stemovaného slova odstraňujeme předpony a přípony, které jsou typické pro zkoumaný jazyk. Výsledkem této aplikace je tzv. *stem*. *Stem* nemusí být nutně slovo daného jazyka nebo morfologický kořen slova. Typickým problémem stematizace je tzv. *over-stemming*, kdy je ze slova odstraněno více než pouhé koncovky. Opačným extrémem je *under-stemming*, což znamená, že ve *stemu* zůstala část koncovky. Pokud nemáme na stematizaci nějaké speciální požadavky, bohatě nám postačí, když pro slova se stejným základem dostaneme stejný stem.

Při lematizaci se naopak snažíme najít platný slovní základ, tzv. *lemma*, což bývá algoritmicky velice složité. Pro tento účel se nejčastěji používají slovníky, které obsahují *lemma* pro jednotlivé tvary slov.

Z pohledu dolování dat z textu, však většinou dáváme přednost stematizaci, která bývá implementačně jednodušší a postačí nám, když slova stejného významu budou vracet jeden *stem*. Někdy se stematizace využívá i pro nalezení *lemma*. V takovém případě využíváme tabulku, která převádí *stem* na *lemma*.

Existuje několik druhů stematizačních a lematizačních algoritmů, které se liší přesností, výkonností a principem, kterým základ slova získávají. Nyní si představíme několik vybraných algoritmů:

- Lookup algoritmy – jsou algoritmy, které k získání základu slova používají lookup tabulky. Výhody jsou jednoduchost a snadné ošetření výjimek v případech, kdy daná koncovka je součástí základu slova, ale odstraňováním přípon by byla eliminována. Nevýhodami bývá velikost tabulek, neboť zde musí být uvedeny všechny možné tvary slova. Ty se v praxi získávají přidáním nejčastějších předpon a přípon k danému základu slova. Pokud nově vzniklé slovo neexistuje, nebude se na něj z textu odkazovat

a na funkci algoritmu to nebude mít vliv. Dalším problémem je, že tabulka by měla pokrýt všechna slova jazyka, což je velice náročná úloha.

- Algoritmy odstraňující přípony(anglicky suffix-stripping algorithms) – jedná se o přístup, při kterém nepotřebujeme lookup tabulky. V tomto případě je algoritmus založen na seznamu pravidel, která se mají aplikovat na zpracovávané slovo. Pravidla bývají většinou následujícího tvaru: obsahuje-li slovo danou příponu, odstraň ji. Výhodou tohoto přístupu je jeho jednoduchost. Nevýhodou je, neschopnost poradit si s výjimkami, kdy je koncovka nebo její část součástí základu slova.

- Stochastické metody – snaží se určit kořen slova na základě pravděpodobnostního modelu. Nejdříve se na trénovací množině zjistí pravidla pro stematizaci a ta jsou následně využívána k nalezení *stemů* slov.

- Hybridní metody – v praxi se často používá více přístupů najednou. Klasickým příkladem může být stematizace pomocí odstraňování koncovek, která je rozšířena o tabulku s výjimkami. V případě, že se slovo nachází v tabulce, použije se jeho *stem* z tabulky. Jinak bude stematizováno eliminací koncovek.

3.2.2 Eliminace stop slov

Stop slova jsou slova, která nenesou žádnou významovou informaci pro klasifikaci. Zpravidla se odstraňují nejčastější slova jazyka, kterými jsou členy, předložky, spojky a často se vyskytující slovesa např. být, mít, chtít, která tvoří většinu vět, a proto nemají rozlišovací schopnost.

V praxi neexistuje žádný oficiální seznam stop slov, neboť určení stop slov záleží na úloze, pro kterou text předzpracováváme. Může se stát, že potřebujeme nalézt např. frázi „The Artist“ a je zcela nevhodné mít člen „the“ v seznamu stop slov, kam by jinak bezpochyby patřil.

Informace o předzpracování byly čerpány z [13], [4], [15].

3.3 Vektor rysů

Dokument budeme reprezentovat jako vektor dvojic (term, váha) následovně:

$$d_i = (w_{ij}, w_{i2}, \ldots, w_{it}), \tag{3.1}$$

kde $w_{ij} (i = 1, 2, \ldots, N$ a $j = 1, 2, \ldots, T)$ je váha termu j v dokumentu i. T (N) je celkový počet termů (dokumentů). Termům přiřadíme váhy podle vzorce 3.5.

Pro podobnost dokumentů budeme používat následující vzorec:

$$sim(d_i, d_j) = cos(d_i, d_j) = \frac{d_i * d_j}{||d_i||_2 \times ||d_j||_2} = \frac{\sum_{l=1}^{T} w_{il} \times w_{jl}}{\sqrt{\sum_{l=1}^{T} w_{i,l}^2} \times \sqrt{\sum_{l=1}^{T} w_{j,l}^2}} \tag{3.2}$$

3.4 Redukce počtu rysů

Používat všechna různá slova obsažená v dokumentu jako vektor rysů je značně neefektivní. U větších kolekcí se tak můžeme dostat ke stovkám až tisícům termů. A proto je vhodné zredukovat počet termů buď výběrem rysů, nebo jejich extrakcí.

Při výběru rysů odstraňujeme z množiny termů slova, která běžně neobsahují sémantiku dokumentu a nebyla eliminována při předzpracování.

Používají se i další metody výběru rysů. Jednou z nejjednodušších metod je výběr termů, které jsou v dokumentu nejfrekventovanější. Ukázalo se, že pouhých 10% nejfrekventovanějších rysů postačuje ke kvalitní klasifikaci.

Druhou metodou je extrakce rysů. Ta provádí transformaci původní množiny termů do jiné množiny termů, která může být menší a lépe charakterizující dokument, neboť počítá se synonymy, hononymy a víceznačností.

3.5 Váhování

Dalším krokem je přiřazení váhy jednotlivým rysům. To lze jednoduše provést tak, že term, který se vyskytuje v dokumentu, bude mít váhu 1 a 0 bude určena termům bez výskytu. Toto řešení však není příliš kvalitní a podstatně lepší je termům přiřadit váhy na základě počtu frekvence termu v dokumentu, v kategorii a v celé kolekci.

K tomu se nejčastěji používá TF-IDF schéma. Myšlenkou TF váhování je, že čím větší je frekvence termu v dokumentu, tím je term relevantnější. U jednoduché TF je váhou počet výskytů. To však nezahrnuje délku dokumentu, a proto je lepší použít normovanou TF:

$$TF(t,d) = 0{,}5 + \frac{0{,}5 * f(t,d)}{MaxFreq(d)}, \tag{3.3}$$

kde f je počet výskytů termu t v dokumentu d a MaxFreq(d) je frekvence nejčastěji se vyskytujícího termu v dokumentu.

Váha IDF zase vyjadřuje, že méně frekventované termy jsou více diskriminativní. Pro IDF pak platí vzorec:

$$IDF(t) = 1 + \log(\frac{n}{k}), \tag{3.4}$$

kde n je celkový počet dokumentů a k je počet dokumentů, ve kterých se vyskytuje term t. Výsledný vzorec pro váhování TF-IDF je:

$$weight(t,d) = TF(t,d) * IDF(t,d). \tag{3.5}$$

Informace o reprezentaci dokumentu a váhování byly čerpány z [1] a [7].

Kapitola 4

Klasifikace(kategorizace) textu

Cílem klasifikace textu je roztřídit textové dokumenty do předem daných kategorií. Aplikace klasifikace se využívají k třídění internetových stránek do kategorií, klasifikaci novinových článků, k rozpoznání spamů v emailu, třídění e-mailů atd.

Kategorizace textu je jednou z úloh umělé inteligence, ke které lze přistoupit dvěma základními způsoby. Prvním přístupem je znalostní inženýrství(*knowledge engineering*), kde jsou systému uživatelem zadány znalosti o jednotlivých kategoriích a pravidla pro třídění. Druhou metodou je strojové učení. Zde systému zadáme pouze několik již roztříděných dokumentů (trénovací data), ze kterých si systém sám získá informace, jež následně uplatňuje při klasifikaci. Lepších výsledků dosahuje metoda znalostního inženýrství, která je však pracnější než strojové učení. Na druhou stranu strojové učení momentálně prožívá značný rozvoj a rozdíly mezi těmito přístupy se pomalu začínají stírat.

4.1 Strojové učení a přístupy k němu

Základním přístupem, který je používán pro automatické sestavení klasifikátoru textu, je strojové učení. To využívá znalostí získaných z předtříděných trénovacích dat. Existují dva základní způsoby přístupu ke strojovému učení. Jedním je učení s učitelem (supervised learning), při kterém v trénovací fázi víme, zda dokument patří do dané kategorie nebo ne. Opakem tohoto přístupu je učení bez učitele (unsupervised learning), též označované jako shlukování (clustering), které je popsáno v kapitole 2.5.

Nyní se budeme věnovat vybraným základním přístupům ke klasifikaci textu pomocí strojového učení s učitelem.

4.1.1 Bayesovská klasifikace

Beyesovská klasifikace je založena na podmíněné pravděpodobnosti vyjádřené vztahem:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(x)}, \tag{4.1}$$

kde $P(C_i|x)$ je pravděpodobnost, že dokument X patří do kategorie C_i. $P(X)$ v uvedeném vzorci je konstantní pro všechny kategorie a je nutno dopočítat pouze $P(C_i|X)$. Pokud máme dokument reprezentován vektorem rysů $d = (w_1, w_2, \ldots, w_n)$ a předpokládáme, že jednotlivé souřadnice jsou na sobě nezávislé, můžeme použít vztah:

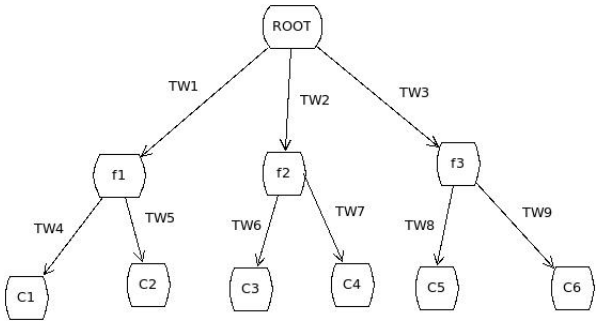
$$P(X|C_i) = \Pi_{k=1}^{n} P(X_k|C_i) \tag{4.2}$$

Výhodou tohoto přístupu je snadná implementace a dobré výsledky pro řadu úloh. Naopak největší nevýhodou je předpoklad, že jednotlivé souřadnice jsou na sobě nezávislé.

4.1.2 Rozhodovací strom a rozhodovací pravidla

Rozhodovací strom je stromová struktura, ve které rysy označují vnitřní uzly, na hranách jsou umístěny testy vah rysů a listové uzly obsahují výslednou kategorii. Pokud se po stromu pohybujeme od kořene k listu, sepisujeme jednotlivá rozhodovací pravidla umístěná na hranách přechodů a spojujeme tato pravidla spojkou „a“, tím získáváme rozhodovací pravidlo pro výslednou kategorii. Chceme-li sloučit několik kategorií do jedné, použijeme spojku „nebo“.

Obrázek 4.1 zobrazuje rozhodovací strom, kde $TW1, TW2, \ldots, TW9$ jsou testy vah rysů, $f1, f2, f3$ jsou rysy a $C1, C2, \ldots, C6$ jsou výsledné kategorie.



Obrázek 4.1: Ukázka rozhodovacího stromu.

4.1.3 Neuronové sítě

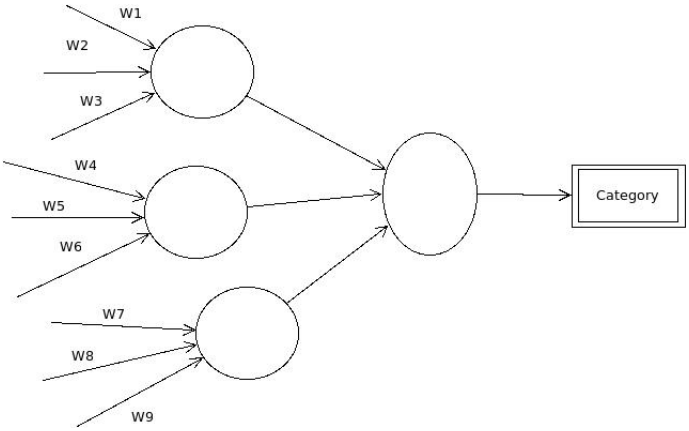
Jedná se o přístup, kdy se klasifikace provádí pomocí neuronové sítě, což je několik neuronů spojených za sebe. Typický neuron má několik vstupů, bias, aktivační funkci a výstup. Práce neuronu tedy vypadá tak, že se na něj přivedou vstupní hodnoty. Tyto hodnoty se vynásobí, sečtou a přidá se k nim bias. Potom před výstupem je ještě hodnota transformována pomocí aktivační funkce.

Neuronová síť je trénovaná metodou back-propagation. Pokud v trénovací fázi je zjištěna chyba, tak se tato chyba propaguje zpět přes neuronovou síť a upravují se příslušné koeficienty. To se opakuje tak dlouho, dokud není síť schopná správně reagovat na všechna data z trénovací množiny.

Na obrázku 4.2 je zobrazena neuronová síť, kde na vstupech neuronů jsou váhy rysů $W1, W2 \ldots W9$, výstupem je výsledná kategorie.

4.1.4 Klasifikace založena na k-nejbližším sousedství

Je tradiční metoda založená na tom, že nový dokument zařadíme do třídy(tříd), která je nejvíce zastoupena v k-nejbližších sousedech. U tohoto algoritmu v trénovací fázi data pouze shromažďujeme jako vektory dokumentů. Pokud přijde nový dokument, zjistíme jeho podobnost s dokumenty ostatních tříd podle vzorce 3.2. Potom pomocí prahovací hodnoty odfiltrujeme odpovídající kategorie a těmi dokument označíme.

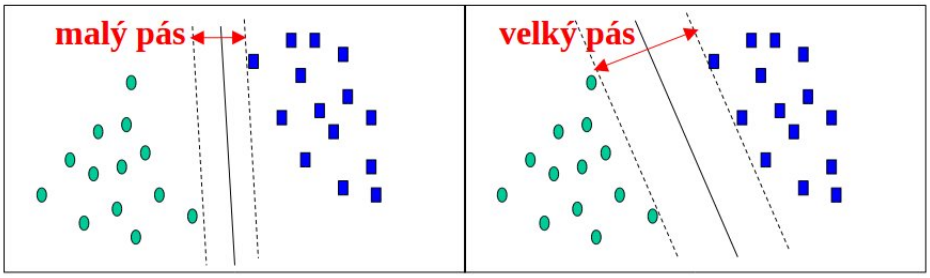


Obrázek 4.2: Ukázka neuronové sítě.

4.1.5 Support vector machine (SVM)

SVM je relativně nová, rychlá a efektivní metoda klasifikace textu. Metoda se snaží oddělit kladné a záporné instance tak, aby mezi nimi vznikl co největší pás. Tento pás je určitou tolerancí, která nesmí být překročena, aby dokument nebyl špatně zařazen. K určení oddělující nadroviny stačí relativně malé množství trénovacích dat, nazývaných support vector.

Na obrázku 4.3 vidíme znázornění metody SVM s ukázkou dobrého a špatného pásu.



Obrázek 4.3: Ukázka SVM (převzato z [9]).

Informace o klasifikaci byly čerpány z [7],[14],a [9].

4.2 Matrix regression

Matrix regreession je relativně nová metoda kategorizace založená na strojovém učení s učitelem. V trénovací fázi vytvoříme matici termů a kategorií a v testovací fázi podle této matice klasifikujeme nové dokumenty. Jádrem metody je matice charakterizující vztah mezi termem a kategorií. Algoritmus matrix regression bude implementován v rámci této práce.

4.2.1 Trénovací fáze

Nejprve získáme z trénovacích dokumentů množinu všech termů, které se v nich objevují. Každému dokumentu přiřadíme jeho vektor termů. Sjednocením vektorů všech dokumentů získáme množinu termů v trénovacích dokumentech. Tuto množinu označíme T:

$$T = \{t_1, t_2, \ldots, t_T\},$$

kde t_i je term a $|T|$ je počet všech termů v tréninkové kolekci.

Poté sestavíme seřazenou množinu všech kategorií z trénovací množiny. Seřazená množina vznikne sjednocením všech kategorií v předtříděných dokumentech. Množinu kategorií v trénovací kolekci označíme jako C:

$$C = \{c_1, c_2, \ldots, c_C\},$$

kde c_i je kategorie a $|C|$ je celkový počet kategorií v trénovací kolekci. V ideálním případě by $|C|$ bylo stejně velké jako počet všech možných kategorií. Avšak může dojít k situaci, kdy v trénovacích dokumentech některá kategorie nebude zastoupena. Z množin T a C pak sestavíme matici $W = (w_{ij})_{T \times C}$, ve které řádky odpovídají termům a sloupce kategoriím.

$$W = \begin{pmatrix} w_{11} & w_{12} & \ldots & w_{1C} \\ w_{21} & w_{22} & \ldots & w_{2C} \\ \vdots & \vdots & \vdots & \vdots \\ w_{T1} & w_{T2} & \ldots & w_{TC} \end{pmatrix} \tag{4.3}$$

Na začátku jsou všechny hodnoty v matici 0, tedy $w_{ij} = 0, \forall i, j$. Potom pro každý dokument z trénovací sady zjistíme kategorie, do kterých je zařazen. Následně pro každý term a kategorii identifikujeme příslušné pole w_{ij} v matici, které inkrementujeme o hodnotu $tf - idf$, což je váha termu.

Vytvořenou matici uložíme na disk a dále ji budeme používat jako model, podle kterého budeme klasifikovat další dokumenty. Pokud do trénovací kolekce přibyde nějaký dokument, není nutné celý model přepočítávat, ale stačí pouze zkontrolovat nové kategorie a nové termy v dokumentu a pro ně přidat nové sloupce(řádky). Poté již jen inkrementujeme dotknuté čítače a model je opět aktuální.

4.2.2 Testovací fáze

V druhé fázi nejprve načteme model a obě množiny vektorů T a C získané v trénovací fázi. Každý nový dokument je spojen s vektorem termů, označme jej T_d. Vytvoříme nový tzv. filtrovací vektor $F = (0)_{1 \times t}$. Jeho velikost odpovídá počtu termů T. Nechť $T' = T_d \cap T$. Pro každý prvek z T' nastavíme hodnotu odpovídajícího prvku z F na 1. Úlohou vektoru F je získat řádky trénovací matice, které odpovídají termům z klasifikovaného dokumentu. Tyto řádky jsou navrženy jako kategorie nového dokumentu. Výsledkem je vektor $W' = F \times W, W' = (w'_{ij})_{1 \times C})$ o velikosti $|C|$, kde prvky odpovídají vahám jednotlivých kategorií. Nedůležité řádky matice W byly vektorem F nastaveny na nula a důležité řádky byly vybrány.

Posledním krokem je vybrání správných kategorií pomocí prahovací hodnoty. Výsledné kategorie jsou ty, jejichž váha je větší než prahovací hodnota.

Informace o metodě matrix regression byly čerpány z [11];

Kapitola 5

Multi-label klasifikace a prahování

Pokud se snažíme dokument zařadit do více než jedné kategorie, pak hovoříme o multi-label klasifikaci. Hlavním problémem takovéto klasifikace je určení, která kategorie je ještě relevantní pro daný dokument a která už ne. K jejich odlišení se nejčastěji používá prahování.

5.1 Prahování

Prahování je funkce, která zobrazí množinu libovolných hodnot do množiny o určité četnosti. Zde si názorně ukážeme nejjednodušší prahování pro množinu o četnosti 2. V tomto případě je zpravidla výsledkem prahování prvek z množiny $s = \{0; 1\}$. Matematicky lze jednoduché prahování definovat takto:

$$p(x) = \begin{cases} 1, x \geq T \\ 0, x < T \end{cases},$$

kde p je prahovací funkce, která porovná prahovanou hodnotu x s prahem T a pokud je x větší jak T, přiradí mu hodnotu 1, jinak 0.

5.2 Multi-label klasifikace

Nejprve je třeba si říci, proč je volba správného prahu tak důležitá a problematická pro multi-label klasifikaci. U klasické klasifikace, kdy každému dokumentu přiřadíme pouze jednu kategorii, není prahování vůbec potřeba. Jednoduše zjistíme, která kategorie danému článku nejvíce vyhovuje a tu považujeme za výsledek.

Multi-label klasifikace je v tomto ohledu mnohem složitější. Cílem je vybrat všechny kategorie, které jsou pro daný dokument relevantní. Proto musíme najít takový práh, který bude oddělovat vhodné kategorie od těch, do kterých dokument nepatří. Bohužel předem nevíme, kolik tříd bude odpovídat našim datům. Kdyby bylo předem dáno, že výsledkem bude x kategorií, vzali bychom pouze x výsledků s nejvyššími vahami a problém by byl vyřešen.

Metod pro multi-label klasifikaci je opravdu celá řada. Dá se pro ni použít i většina algoritmů popsaných v kapitolách 4, 2.5 a samozřejmě i námi vybraná matrix-regression 4.2. Zatímco tyto metody jsou velice dobře definovány, zhodnocení jejich výsledků už tak jednoznačné nebývá. Samozřejmě bylo napsáno mnoho prací, které se zabývaly určením relevantnosti kategorií k dokumentům u multi-label klasifikace a bylo nalezeno i mnoho řešení. Avšak nalezené metodiky jsou značně individuální podle použitého algoritmu a dat,

na které byly aplikovány. Navíc metoda matrix regression je relativně nová a tak o jejím prahování moc informací nemáme. Bude tedy potřeba vhodný práh stanovit experimentálně. Pro inspiraci si představíme jednu jednoduchou metodu prahování, tak jak byla navržena v [2].

5.2.1 Metoda prahování pro multi-label klasifikaci založená na nejmenších rozdílech

Metoda byla původně navržena pro reklasifikaci dokumentů v tématických slovnících při shlukové analýze. S drobnými úpravami by bylo možné ji použít i pro naše účely.

- Pro každý dokument spočítáme rozdíly mezi maximální vahou TFIDF TI_{max} a k-tou vahou TDIDF pro všechny kategorie K a označíme je $diff_{k+k(i)}$. Tzn. rozdíl vah TFIDF mezi k-tou kategorií a TI_{max} je roven nule, zatímco pro ostatní kategorie je rozdíl vyšší nebo nižší podle toho, jak se jejich váha TFIDF bliží TI_{max}.
- Vybereme nejnižší hodnotu rozdílu větší jak nula v kategorii $K-1$ pro i-tý dokument a označíme ji $diff_{min}(i)$.
- Spočítáme průměr těchto nejnižších rozdílů pro všech N-dokumentů a označíme ho $diff_{med}$.
- Vybereme pouze hodnoty v rozmezí $diff_{k+k(i)} < diff_{med}$.
- Kategorie odpovídající vybraným hodnotám jsou nově přirazeny dokumentům. Dokumenty, které nemají žádnou kategorii v daném rozmezí zůstávají pouze v jedné kategorii s hodnotou TI_{max}.

Kritérium nejnižšího rozdílu samozřejmě také není ideální. Je sice výhodné pro fragmenty, kde TI_{max} je malé a druhá nejvyšší váha TFIDF je blízko TI_{max} ale pořád malá. Nevýhodné naopak je v případech, kdy je TI_{max} a druhá nejvyšší hodnota vysoká.

Jiným kritériem je kritérium druhé nejvyšší váhy TFIDF, kdy vybereme ještě kategorie patřící k druhé nejvyšší váze. Toto řešení vyhovuje u dokumentů s vysokou hodnotou druhé nejvyšší váhy a naopak není vhodné pro dokumenty s nízkými vahami.

Informace o prahování a multi-label klasifikaci byly čerpány z [2] a [6].

Kapitola 6

Zadání aplikace

Aplikace má být implementována v programovacím jazyce java. Má fungovat pod operačním systémem Microsoft Windows a Linux a má by poskytovat intuitivní uživatelské rozhraní.

Program má provádět multi-label klasifikaci pomocí metody matrix regression. Má sestavovat kolekci ze zadaného souboru nebo adresáře. Vytvořenou kolekci uloží do specifikovaného adresáře v binární podobě, pokud je to požadováno. Program musí nabízet náhled na kolekci a možnost její úpravy pomocí přidání nových dokumentů, do již existujících kategorií, nebo rovnou přidáním celé kategorie.

Program bude pomocí načtené kolekce klasifikovat testované dokumenty. Výsledek zobrazí prostřednictvím grafického uživatelského rozhraní. Výsledné zobrazení bude ve tvaru kategorie a do ní patřící dokumenty. Pro snadnější hledání kategorií pro jednotlivé dokumenty musí být program schopný vyfiltrovat kategorie podle názvu dokumentu.

6.1 Vstupní data

Vstupní kolekce může být zadána ve formátu reuters a to jako jediný soubor nebo adresář obsahující více reuters souborů.

Dalším formátem vstupní kolekce je adresář obsahující adresáře reprezentující kategorie. V adresářích kategorie budou umístěny textové soubory odpovídající dané kategorii. Při tomto formátu vstupu bude jako jméno kategorie brán název adresáře, jež jí reprezentuje a jako jméno dokumentu název textového souboru.

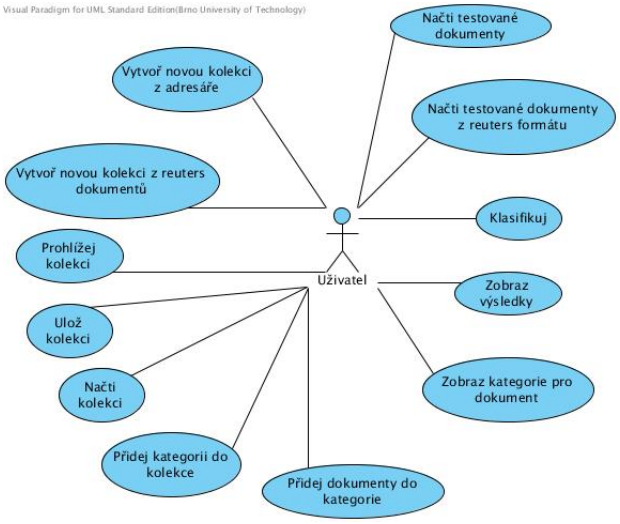
Posledním vstupním formátem může být kolekce již dříve vytvořená naším programem a uložená v binární podobě. Název kolekce pak bude shodný s názvem vstupního souboru.

Formáty testovaných dokumentů jsou prakticky stejné jako vstupní formáty kolekcí. Jediný rozdíl je u dokumentů, které nejsou reuters. Ty budou načítány z jednoho adresáře bez vnitřní struktury, případně může jít i o samotný textový soubor, chceme-li testovat jediný dokument.

6.2 Případy použití

Požadavky na cílový systém přehledně zobrazuje diagram případů použití 6.1, který si nyní blíže popíšeme.

- Uživatel – je jedinou rolí v systému, která má práva na všechny implementované operace.



Obrázek 6.1: Diagram případů použití.

- Vytvoř novou kolekci z aresáře – ze zadaného adresáře obsahujicího roztříděné dokumenty se vytvoří kolekce.
- Vytvoř novou kolekci z reuters dokumentů – z reuters souboru nebo adresáře s reuters dokumenty se vytvoří kolekce.
- Prohlížej kolekci – přehledně zobrazí aktuálně načtenou kolekci, kde bude patrné, jaké dokumenty kategorie obsahují.
- Ulož kolekci – uloží kolekci v binární podobě do zadaného umístění.
- Načti kolekci – načte kolekci v binární podobě ze zadaného umístění.
- Přidej kategorii do kolekce – přidá kategorii daného jména do kolekce.
- Přidej dokumenty do kategorie – přidá dokumenty z daného adresáře nebo souboru do kategorie.
- Načti testované dokumenty – načte dokumenty, které mají být klasifikovány, ze zadaného adresáře případně souboru.
- Načti testované dokumenty z reuters formátu – načte dokumenty ze zadaného umístění s reuters soubory.
- Klasifikuj – roztřídí dokumenty do kategorií podle aktuální kolekce.
- Zobraz výsledky – přehledně zobrazí, do kterých kategorií byly zařazeny dokumenty.
- Zobraz kategorie pro dokument – podle názvu dokumentu zobrazí všechny kategorie, do kterých daný článek patří.

Kapitola 7

Návrh a implementace

Při návrhu java aplikace s grafickým uživatelským rozhraním(dále jen GUI) je důležité zvolit si vhodnou architekturu, která nám umožní smysluplně pracovat s daty a reagovat na podněty od uživatele. Snažíme se především oddělit data od jejich samotného zobrazení. Logika aplikace by tedy měla být implementována pouze na úrovni dat nezávisle na jejich grafické reprezentaci.

Další pro implementaci nezbytnou částí je návrh tříd. K jeho realizaci využijeme diagram tříd, který nám zobrazí, jak jednotlivé objekty reprezentují skutečnost. Jaká nesou data, jaké jsou operace nad nimi implementovány a jak mezi sebou jednotlivé třídy komunikují.

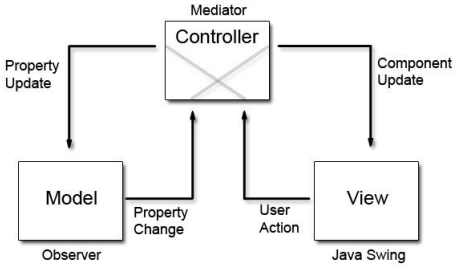
7.1 Architektura MVC

Nejběžnější architekturou pro oddělení dat od zobrazení je architektura MVC, která se skládá ze tří vrstev a to modelu, pohledu(view) a kontroléru(controller). Nyní si popíšeme jednotlivé vrstvy:

- View – je vrstva starající se o vnější prezentaci modelu. Zajišťuje komunikaci s uživatelem a jím vyvolané události předává kontroléru. Zároveň naslouchá událostem jdoucím z modelu a realizuje odpovídající změny v GUI.
- Controller – je vrsta obsahující kontroléry, které zajišťují komunikaci mezi GUI a daty. V klasické MVC architerktuře se kontrolér využívá pouze pro příjem událostí z pohledu a jejich distribuci do příslušného modelu.
- Model – je vrstva reprezentující data a operace nad nimi. Reaguje na události přicházející z kontroléru a realizuje příslušné změny v datech. Pokud je potřeba na základě změny dat provést nějaké akce v pohledu, pošle mu odpovídající událost.

Vrstva kontroléru tak od sebe odděluje zbylé dvě vrstvy. Obvykle se kontrolér používá k tomu, aby přijímal události z GUI a podle typu události zavolal příslušnou metodu v modelu. Model následně zpracuje data požadovaným způsobem a pošle pohledu data k zobrazení.

Samozřejmě existuje více variant MVC architektury. Pro tuto práci byla vybrána modifikovaná MVC architektura, tak jak ji prezentuje firma Oracle na svých stránkách viz. [5]. V této variantě je kontrolér umístěn přímo mezi model a pohled a veškerá komunikace mezi těmito vrstvami probíhá skrz kontrolér. Z toho plyne, že jak všechny pohledy, tak všechny modely musí být u kontroléru registrovány pro příjem událostí. Tento model tak zcela zrušil přímou komunikaci mezi pohledem a modelem.



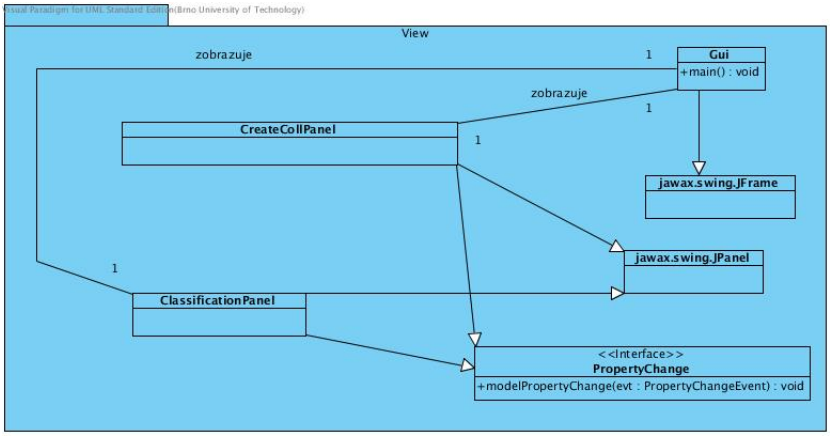
Obrázek 7.1: Diagram modifikované MVC architektury podle firmy Oracle(převzato z [5]).

7.2 Diagram tříd

Nyní si pomocí diagramu tříd názorně popíšeme, jak budou jednotlivé vrstvy MVC architektury implementovány. Vrstvy MVC jsou reprezentovány stejnojmennými balíky. Všechny diagramy v této sekci jsou zjednodušeny oproti skutečnosti z důvodu přehlednosti a úspory místa.

7.2.1 Pohled

Jak je patrné z obrázku 7.2 pohled se skladá z jednoho okna, ve kterém se střídavě zobrazují dva základní panely. Oba panely implementují rozhraní *PropertyChange*, jež obsahuje pouze jedinou metodu a tou je *modelPropertyChange*. Pomocí tohoto rozhraní se panely zaregistrují u kontroléru a metodou *modelPropertyChange* zjišťují, jaká událost přišla, zda patří danému panelu či nikoli a volají metody reagující na ni.



Obrázek 7.2: Zjednodušený diagram tříd pro pohled.

7.2.2 Kontrolér

Prostředníka mezi vrstvou modelu a pohledu dělá vrstva kontroléru. U námi vybrané architektury musí kontrolér umět přijímat události jak z modelu, tak i z pohledu a doručit

je cílové vrstvě. Náš kontrolér se skládá z jedné abstraktní třídy a další třídy, která ji implementuje. Grafické znázornění se nachází na obrázku 7.3.

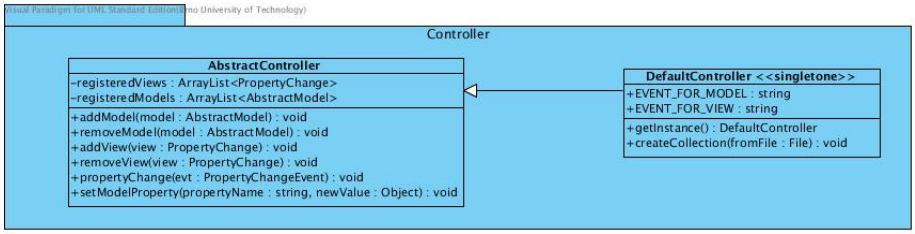
AbstractController

Třída *AbstractController* je obecná třída pro implementaci kontroléru. Slouží k registrování modelů a pohledů pro distribuci událostí. Samozřejmě také umí konkrétní objekty odstranit z distribučního seznamu.

AbstractController se chová malinko odlišně pro model a pro pohled. V případě, že příchozí událost je určena některému objektu z modelu, hledá *AbstractController* metodu odpovídající dané události mezi zaregistrovanými modely pomocí metody *setModelProperty*. Zatímco událost patřící pohledu je metodou *propertyChange* přeposlána do všech registrovaných pohledů a v objektu, kterému patří, je teprve volána odpovídající metoda.

DefaultController

Třída *DefaultController* pouze implementuje funkčnost třídy *AbstractController*. Jsou v ní nadefinovány konkrétní události pro model a pohled. Jelikož k třídě *DefaultController* budeme přistupovat poměrně často z mnoha různých objektů, byla navržena podle návrhového vzoru singletone a odkaz na ni lze získat kdekoliv v naší aplikaci.



Obrázek 7.3: Zjednodušený diagram tříd pro kontrolér.

7.2.3 Model

Vrstvou reprezentující data a implementující logiku aplikace je model. Ten musí umět naslouchat na události přicházející z vrstvy kontroléru a posílat své události do kontroléru. Komunikaci s vyšší vrstvou zde implementuje třída *AbstractModel*. Diagram modelu je zobrazen na obrázku 7.4.

MatReg

Nejdůležitější třídou modelu je třída *MatReg*, která reprezentuje celou kolekci. Je implementována podle návrhového vzoru singletone, takže je dosažitelná z kterékoli části programu. Zároveň to ale znamená, že existuje pouze jedna její instance. Náš program tedy vždy pracuje pouze s jednou kolekcí.

Třída *MatReg* je jakýmsi „mozkem“ aplikace. Z kontroléru přijímá většinu událostí vytvořených v GUI a buď je sama vyřizuje, nebo je distribuuje do dalších objektů. Pokud je potřeba změnit zobrazení dat v pohledu, vytvoří *MatReg* příslušnou událost a odešle ji do kontroléru. Aby mohla komunikovat s kontrolérem, dědí třídu *AbstractModel*.

Při pohledu na atributy, které *MatReg* obsahuje, si v prvé řadě všimněme proměnné *mat*, která je static a je nezbytná kvůli implementaci návrhového vzoru singletone, aby třída mohla vracet odkaz sama na sebe. Třída *MatReg* je dále odpovědná za vytváření objektů kategorií, které následně drží v proměnné *catList*. *CatList* je zde plněn pomocí metody *fillInCatList* volané z metody *createCollection*. Pokud je potřeba přidat kategorii do kolekce, postará se o to funkce *addCategoryToCollection*. Kolekce je do pohledu posílaná událostí vytvořenou v metodě *createCollectionTree*, kde je převedena do objektu typu *DefaultMutableTreeNode* pro reprezentaci v objektu *jTree* v GUI.

Váha TF-IDF je počítána na úrovni kolekce, proto se její určení provádí v objektu typu *MatReg*, konkrétně funkcemi *countIdf* a *countTfIdf*. Pro výpočet váhy IDF je důležitý atribut *allTerms*, který obsahuje všechny termy v kolekci a je nezbytný pro zjištění v kolika dokumentech se daný term vyskytl.

O uložení a načtení kolekce se starají metody *saveCollection* a *readCollection*, které implementují serializaci a deserializaci objektu matReg.

Category

Třídou reprezentující kategorie je třída *Category*, která udržuje trénovací dokumenty patřící do kategorie v HashMapě *docs*. Druhou její HashMapou je *terms*, která obsahuje všechny termy dané kategorie. *Terms* se využívá při samotné klasifikaci, neboť drží TF-IDF váhy všech termů v kategorii. Tedy slova z klasifikovaného dokumentu získávají váhy právě odtud. HashMapa *terms* je plněna funkcemi *unionDocs* a *updateCategoryTerms*, které jsou volány na právě zpracovaný dokument a sjednocují termy dokumentu s termy kategorie. Tyto funkce se také starají o počítání tf váhy termů pro kategorii.

Ve chvíli, kdy je načtena celá kolekce, je zavolána funkce *countTfIdf* ze třídy *MatReg*, která dopočítá váhy TF-IDF pro všechny termy v *terms*. V tuto chvíli *terms* reprezentuje jeden sloupec matice algoritmu matrix regression.

Poslední zajímavou metodou třídy *Category* je *addDocumentToCategory*. Jedná se o přetíženou metodu. Pokud je volána s parametrem *TrainingDocument*, tak pouze přidá načtený dokument do kategorie. V případě, že je volána s parametrem *ArrayList*, je jejím úkolem načíst dokument přidávaný do kategorie. Pokud kategorie neexistuje, pak je vytvořena.

TrainingDocument

Jeden dokument v kolekci je reprezentován třídou *trainingDocument*. Její úlohou je udržovat HashMapu termů *termList*, které se nacházejí v dokumentu. Po načtení celého trénovacího článku pomocí funkce *parseTerms* je metodou *countTf* spočítána váha TF všech jeho termů.

Term

Nejzákladnějším prvkem modelu je *Term*. Tato třída reprezentuje samotný term a drží četnosti výskytu termu a jeho váhy. Proměnné *countInDoc*, *countInCat* obsahují hodnotu, kolikrát se term vyskytl v dokumentu či kategorii. Zatímco atribut *countInMatreg* udává, v kolika různých dokumentech v kolekci se term vyskytl. Dále má term atributy, které nesou jeho váhy.

Všechny termy jsou drženy v instancích třídy *TrainingDocument* a dále už předávány pouze odkazem, což je umožněno výše popsaným rozložením atributů pro počty výskytů a váhy, kdy každá třída pracující s termem používá proměnnou určenou pro ni. Toto řešení by mělo pomoci uspořit paměť potřebnou pro běh programu.

TestDirectory

O načtení a klasifikaci testovaných dokumentů se stará třída *TestDirectory*, která dědí *AbstractModel*, aby mohla komunikovat s kontrolérem. *TestDirectory* získá testované dokumenty buď z reuters kolekce pomocí funkce *getTestDocumentsFromReuters*, nebo z adresářové struktury metodou *findTestDocuments*.

O třídění dokumentů se stará funkce *classify*, jejíž výsledek je převeden funkcí *createResultTree* do seznamu reprezentujícího výsledek, který je poslán událostí do pohledu. *TestDirectory* také implementuje na úrovni modelu filtrování kategorií, které patří zadanému dokumentu. K tomuto účelu slouží funkce *searchCategoriesForDocument*.

TestDocument

Jednoduchou třídou spravující klasifikovaný dokument je *TestDocument*. Ta obsahuje množinu řetězců, ve které jsou umístěny názvy termů, jež se objevily v dokumentu. O načtení množiny *terms* se stará funkce *parseTerms*.

Další důležitou proměnnou třídy *TestDocument* je *catArray*, což je pole typu double o velikosti počtu kategorií v kolekci. Do tohoto pole se ukládají váhy, které získá testovaný dokument pro danou kategorii. Pořadí vah v poli odpovídá pořadí kategorií v seznamu *catList* v objektu třídy *MatReg*. Kategorie, které jsou pro testovaný dokument navrženy, získáme funkcí *findCategories*, jež prochází *catArray* a vybírá všechny kategorie s vahou vyšší, než je zadaný práh.

PreprocessReuter

Předzpracování dokumentů implementuje třída *PreprocessReuter*, která obsahuje atributy *stopWords* a *stemmer*. Tyto dvě proměnné využívá funkce *advancePreprocess* implementující odstraňování stop slov a stemming. Další důležitou funkcí je *PreprocessSGM*, která zpracovává reuters kolekci tak, že z ní získá nadpis článku, jeho kategorie a jeho obsah a odstraní z ní SGML tagy.

Stemmer

Třída *Stemmer* implementuje stemming pomocí Portrova algoritmu.

ReuterCollection

Třída implementující přechod z reuters formátu do formátu získaného z adresářové struktury se jmenuje *ReuterCollection*. Je navržena podle návrhového vzoru objektový adaptér. Její metoda *parseTerms* načítá termy z reuters dokumentu a funkce *addDocumentToMatReg* se stará o přidání trénovacího dokumentu do správných kategorií, které načte ze seznamu *Categories*.

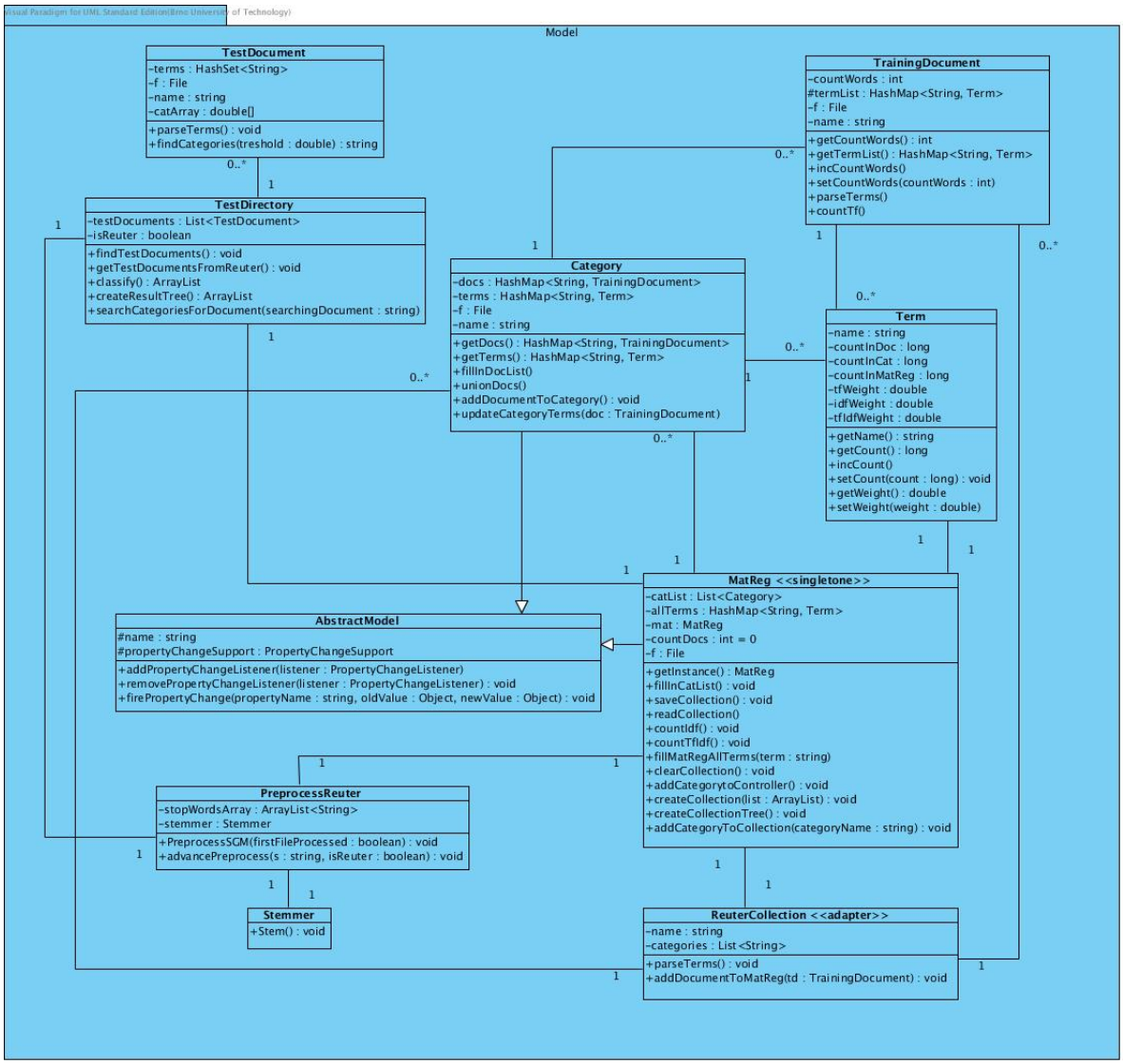
AbstractModel

Jak již bylo zmíněno, ke komunikaci s kontrolérem se používá implementace třídy *AbstractModel*. Ta implementuje přidávání a odebírání naslouchačů na události(metody *addPropertyChangeSupport* a *removePropertyChangeSupport*). Dále zprostředkovává posílání událostí pomocí metody *firePropertyChange*.

Výběr kontejnerů

Důležitou součástí návrhu je výběr kontejnerů, do kterých umístíme zpracovávaná data. Naše aplikace využívá HashMapy k držení velkého množství objektů, ve kterých často potřebujeme vyhledávat. Příkladem takových objektů můžou být objekty třídy *Term*. Výhodou HashMapy je přímý přístup k prvkům, tedy podstatně rychlejší vyhledávání, než je sekvenční procházení seznamu. Nevýhodou je větší paměťová náročnost tohoto přístupu. Avšak při načítání termů, kdy potřebujeme pokaždé zjistit, zda už daný term v kontejneru máme či ne, je přímý přístup HashMapy značně rychlejší a výhodnější vzhledem k počtu opakování takového vyhledávání.

Zatímco kontejner *ArrayList* byl využit například pro ukládání kategorií v třídě *MatReg*. V tomto případě víme, že kategorií nebude velké množství, takže sekvenční průchod nezabere příliš mnoho času a navíc při klasifikování jsme využili stálého pořadí prvků v seznamu.



Obrázek 7.4: Zjednodušený diagram tříd pro model.

7.3 Testování

Nedílnou součástí vývoje aplikace je testování. Náš program byl testován jak v průběhu jeho vytváření, tak po jeho dokončení, kdy bylo cílem ověřit na nějaké malé množině dokumentů, že opravdu dokáže rozlišit váhami články, které do kategorie patří a které ne.

Během implementace byla vždy ověřena funkčnost právě dokončené části programu. Vybrali jsme vhodná testovací data a pro ně zjistili správný výstup. Potom jsme ověřili, že aplikace vrací očekávané výsledky.

Podstatně složitější bylo ověření funkčnosti celého algoritmu. Pro zjištění, že výpočet vah v trénovací kolekci funguje správně, byla vytvořena miniaturní kolekce o přibližně deseti termech s různým počtem výskytů ve dvou kategoriích a čtyřech dokumentech. TF-IDF váhy byly ručně dopočítány a následně porovnány s váhami vypsanými aplikací.

7.3.1 Ověření klasifikace

Po jednoduchém testu trénovací fáze bylo potřeba ověřit celý algoritmus klasifikace. Pro tyto účely byla sestavena malá testovací kolekce, která obsahovala pouze pět náhodně zvolených kategorií(*grain, crude, acq, money-supply* a *money-fx*). Každá kategorie obsahovala pět trénovacích dokumentů.

Po sestavení trénovací kolekce byly náhodně vybrány dva dokumenty od každé kategorie zastoupené v námi vytvořené kolekci. Následně byly tyto dokumenty programem klasifikovány. Při tomto testu se jednalo pouze o nalezení jedné kategorie, do které dokument patří. Nešlo tedy o testování multi-label klasifikace, které bude předmětem experimentů viz. kapitola 8.

Tabulka 7.1 ukazuje váhy článků pro jednotlivé kategorie. Dokumenty byly pojmenovány podle kategorie, která je jim přirazena v reuters kolekci. Modře jsou vyznačeny váhy kategorií, do kterých by měl dokument patřit. Výsledné váhy jsou v intervalu $\langle 0{,}18; 1{,}76\rangle$.

Budeme-li výsledek testu hodnotit typickým kritériem single-label klasifikace, které říká, že kategorie s nejvyšší váhou je hledanou kategorií, zjistíme, že 6 dokumentů z 10 bylo klasifikováno správně. Úspěšnost klasifikátoru pro single-label klasifikaci by tak byla 60%.

Pokud se do tabulky 7.1 podíváme pozorněji, zjistíme, že dokumenty, které byly správně zařazeny měly pro svou kategorii zřetelně vyšší váhu než pro ostatní. Naopak u článku pod označením *grain2* vidíme, že o jeho špatné klasifikaci rozhodl rozdíl pouhých 0,09 oproti kategorii *crude*. Podobně malý rozdíl je i u špatné klasifikace dokumentu *money-supply2*, zde se jedná o pouhých 12 setin. Pro tyto dva dokumenty se tedy jedná o rozdíly skutečně minimální. Hlavní problém lze v tomto případě vidět především v malém množství dokumentů v trénovací kolekci, která tak neměla dostatečnou kvalitu pro bezpečné rozlišení všech kategorií. Navíc tyto problémové dokumenty měly poměrně nízké váhy oproti správně zařazeným dokumentům, což svědčí o nedostatečném pokrytí slov ze špatně zařazených článků v trénovací kolekci.

Největší nepřesnosti v našem testu však dosáhly dokumenty spadající do kategorie *money-fx*. To může být jistě způsobeno již zmíněnou nedostatečnou velikostí trénovací kolekce. Dalším faktorem mohlo být i to, že článek *money-fx1* byl v reuters kolekci označen také kategorií *gold*, tedy nejednalo se o dokument reprezentující pouze kategorii *money-fx*. Navíc tato kategorie patří do tématu *subject* a zkratka *money-fx* znamená *money/foreign exchange*(peníze/zahraniční burza), což u článků zabývajících se obchodem je dosti obecná kategorie.

Po předešlém rozboru získaných výsledků lze náš klasifikátor označit za funkční. Po-

kud bude použit s obsáhlejší trénovací kolekcí, měl by být ještě přesnější, čemuž budou napomáhat i podstatně vyšší váhy získané pro testované dokumenty a tak bude mít klasifikace lepší rozlišovací schopnost.

	grain	**crude**	**acq**	**money-supply**	**money-fx**
grain1	1,56	0,74	1,17	0,44	0,65
grain2	0,76	0,85	0,71	0,43	0,44
crude1	0,26	1,23	0,72	0,45	0,43
crude2	0,42	1,76	0,93	0,65	0,59
acq1	0,31	0,72	1,10	0,83	0,68
acq2	0,31	0,77	1,21	0,57	0,76
money-fx1	0,44	0,91	0,86	0,62	0,68
money-fx2	0,46	1,03	0,83	0,82	0,80
money-supply1	0,18	0,71	0,72	1,15	0,93
money-supply2	0,28	0,82	0,73	0,70	0,65

Tabulka 7.1: Výsledky ověření funkčnosti klasifikace. Modře označeny dokumenty, které mají být klasifikovány.

7.4 Ovládání aplikace

Nyní si stručně popíšeme ovládání vytvořené aplikace. Na obrázku 7.5 je zobrazeno GUI pro vytváření kolekcí. Menu v horní části aplikace slouží k přepínání mezi panelem pro vytváření kolekce a panelem pro klasifikaci.

Do textového pole *jméno kolekce* se zadává jméno kolekce, pod kterým bude kolekce uložena na disk. Vedle tohoto pole najdeme tlačítko *načíst jinou*, které nám umožní načtení dříve uložené kolekce z její binární podoby.

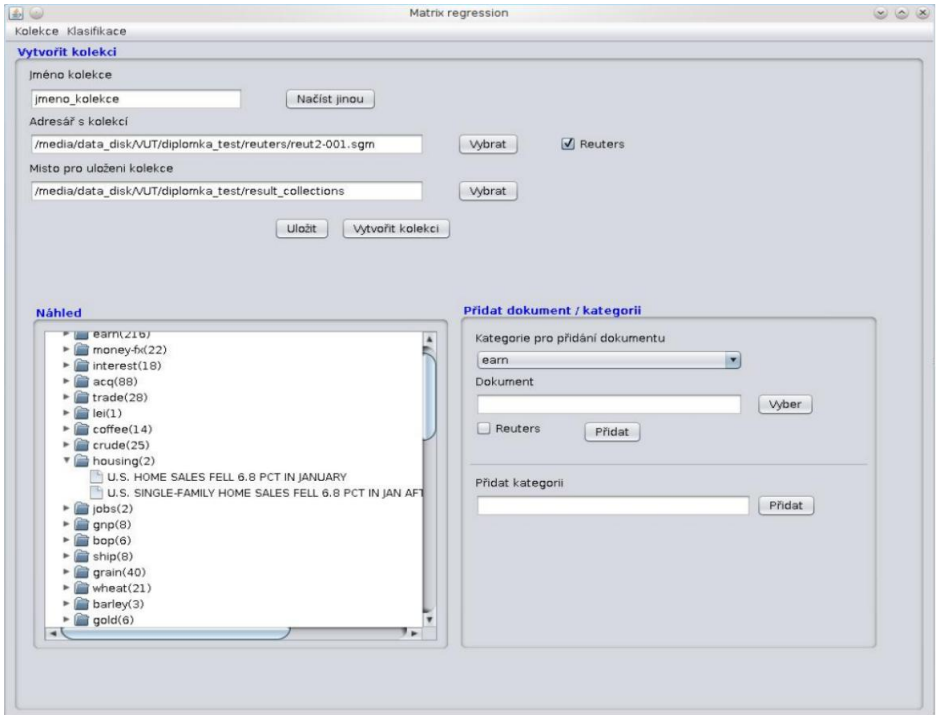
Výběr adresáře s kolekcí a místa pro uložení kolekce se provádí stisknutím tlačítka *Vybrat*, které je umístěno za příslušným textovým polem. Dále zde vidíme zaškrtávací políčko *Reuters*, které musíme označit, pokud zadaná kolekce je v reuters formátu.

Po vyplnění údajů o kolekci můžeme stisknout tlačítko *Vytvořit*, které nám vytvoří kolekci ze zadaného umístění. Pokud byla zadána cesta pro uložení kolekce, je kolekce v binární podobě automaticky uložena. Uložení kolekce je možné i později prostřednictvím tlačítka *Uložit*.

V panelu *Náhled* se nám po vytvoření nebo načtení kolekce zobrazí náhled na kolekci v podobě stromové struktury. Kliknutím na ikonku adresáře se zobrazí všechny dokumenty patřící do dané kategorie. Číslo za ikonkou adresáře vyjadřuje počet dokumentů v odpovídající kategorii.

Panel *Přidat dokument / kategorii* slouží k přidání dokumentu či kategorie do aktuální kolekce. Pokud chceme přidat kategorii, vyplníme příslušné textové pole názvem kategorie a klikneme na tlačítko *Přidat*. Potřebujeme-li přidat nový dokument do kategorie, vybereme kategorii ze seznamu *Kategorie pro přidání dokumentu*. Následně vybereme dokument pomocí tlačítka *Vyber* a v případě, že se jedná o dokument v reuters formátu, zaškrtneme políčko Reuters. Po kliknutí na příslušné tlačítko *Přidat* je dokument přidán do kategorie v kolekci.

Obrázek 7.6 zobrazuje panel klasifikace. Za nadpisem *Aktuální kolekce* vidíme jméno

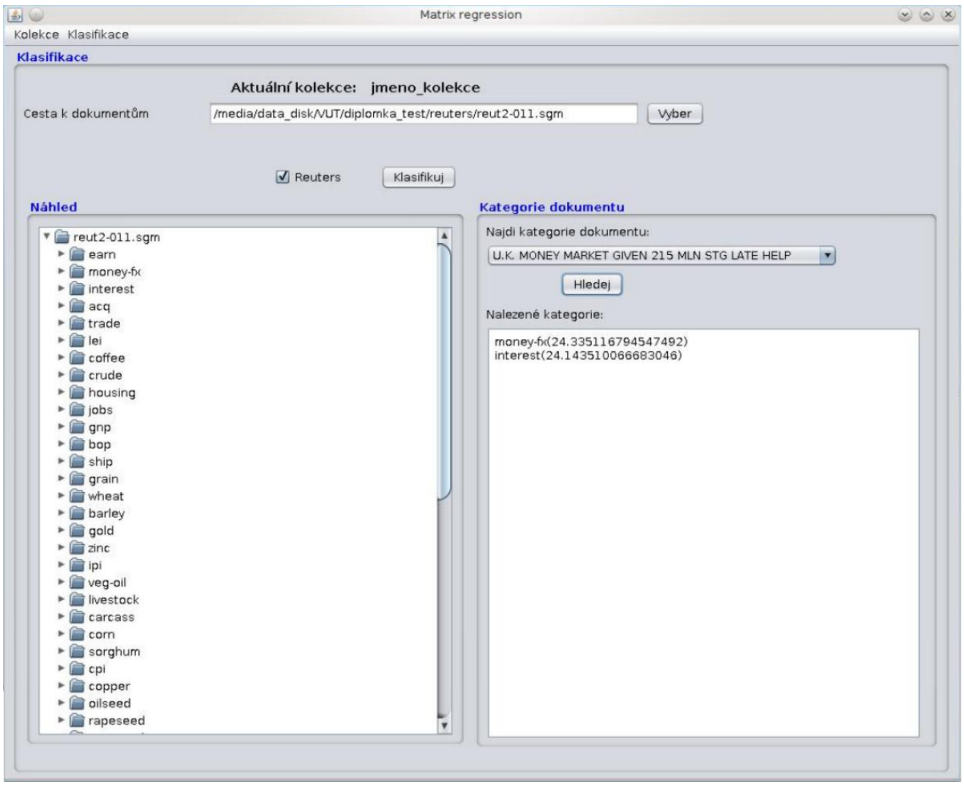


Obrázek 7.5: Uživatelské rozhraní pro vytvoření kolekce.

kolekce, vůči které bude klasifikace probíhat. Testovaná data zadáme kliknutím na tlačítko *Vyber*. Pokud jsou data ve formátu reuters je opět potřeba zatrhnout políčko Reuters. Následně už stačí pouze kliknout na tlačítko *Klasifikuj* a bude provedena klasifikace.

V panelu *Náhled* se nám zobrazí adresářová struktura reprezentující kategorie v dané kolekci. Po rozkliknutí ikonky adresáře vidíme dokumenty, patřící do příslušné kategorie.

Panel *Kategorie dokumentu* slouží k vyfiltrování kategorií pro dokument vybraný ze seznamu. Tlačítkem *Hledej* spustíme prohledávání výsledků klasifikace. Nalezené kategorie pro daný dokument jsou poté zobrazeny v textovém poli *Nalezené kategorie*.



Obrázek 7.6: Uživatelské rozhraní pro klasifikaci.

Kapitola 8

Experimenty

Další důležitou částí této práce jsou experimenty. V nich se budeme především snažit určit vhodný práh pro nalezení relevantních kategorií. Experimentování by také mělo ověřit funkčnost metody matrix regression.

8.1 Váhy TF-IDF

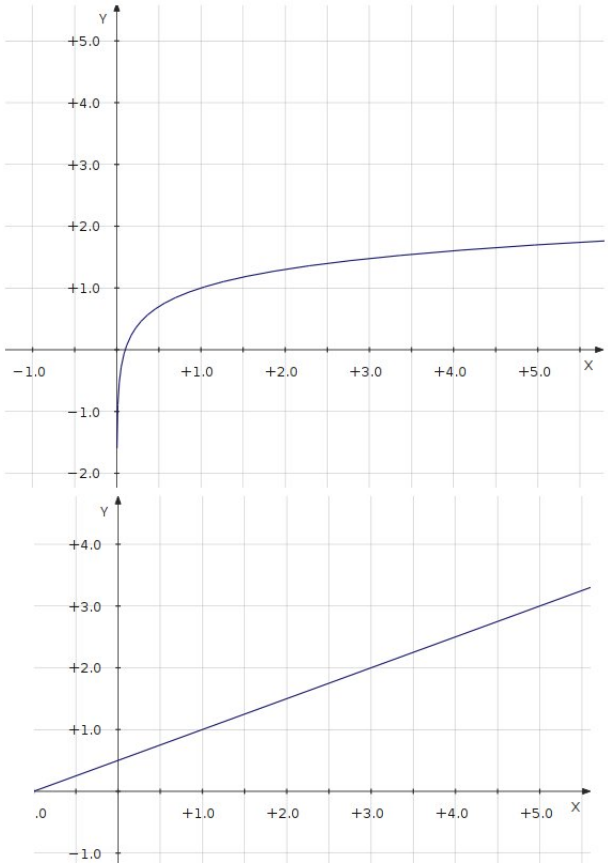
Abychom mohli správně určit práh, musíme se důkladně seznámit s váhami TF a IDF, které jsme si představili v kapitole 3. Potřebujeme nejprve zjistit, jakých hodnot může nabývat váha TF-IDF pro jeden term a následně z toho vyvodit možné hodnoty TF-IDF pro kategorie.

Váha TF definovaná ve vzorci 3.3, jejíž průběh je zobrazen dole na obrázku 8.1, je evidentně lineární funkcí, která může nabývat hodnot z intervalu $i = (-\infty; \infty)$. Avšak v našem případě bude její obor hodnot značně zmenšen, jelikož podíl frekvence termu f s frekvencí termu f_{max} nemůže být nikdy větší než 1. Dosadíme-li do vzorce za zlomek $\frac{f}{f_{max}}$ hodnotu 1, získame maximalní možnou váhu TF, která je taktéž rovna jedné. Minimální váha TF bude vždy větší než 0,5, jak je patrné ze vzorce a grafu této váhy. Pro TF tedy můžeme získat pouze hodnoty v intervalu $i = (0{,}5; 1\rangle$.

O něco málo složitější je situace u IDF váhy, jejíž graf vidíme na obrázku 8.1 nahoře a je dána rovnicí 3.4. Z grafu je patrné, že oborem hodnot této logaritmické funkce je interval $i = (-\infty; \infty)$. Nás však opět nebude zajímat celý definiční obor, ale pouze jeho část, kde platí, že $x \geq 1$, jelikož podíl celkového počtu dokumentů n s počtem dokumentů k, ve kterých se term vyskytuje, je vždy větší nebo roven jedné. Dosadíme-li 1 do vzorce 3.4, vyjde nám hodnota 1, která je minimální hodnotou, jakou může váha IDF nabývat. Z výše uvedeného vyplývá, že hodnota IDF je vždy z intervalu $i = \langle 1; \infty)$.

Z praktického hlediska však můžeme očekávat hodnoty IDF v intervalu $i = \langle 1; 3)$, protože počet dokumentů v našich kolekcích je v řádu stovek. Konec intervalu v hodnotě 3 tedy nebyl vybrán náhodně, ale jedná se o logaritmus z 1000.

Nyní, když víme, jakých hodnot budou váhy TF a IDF nabývat, lze snadno odvodit, že hodnoty TF-IDF by měly být v intervalu $i = (0{,}5; 3)$ pro kolekce o velikosti stovek dokumentů.



Obrázek 8.1: Grafy váhových funkcí(nahoře IDF, dole TF).

8.2 Normalizace váhy TF-IDF pro testovaný dokument

Algoritmus matrix regression je založen na tom, že každá kategorie si drží seznam termů v ní obsažených a jejich TF-IDF váhu. V tesovací fázi vybereme term z dokumentu a zjišťujeme, zda se nachází v právě zkoumané kategorii. Pokud ano, přičteme váhu termu v kategorii k TF-IDF zkoumaného dokumentu. Takto postupujeme pro všechny termy dokumentu, až dostaneme váhu, která by měla vypovídat o tom, jak moc je daná kategorie pro dokument relevantní.

V předchozí sekci jsme zjistili, že hodnota jednoho termu je v rozmezí 0,5 až přibližne 3. Váhu TF-IDF testovaného dokumentu pro danou kategorii získáme součtem vah všech termů obsažených jak v dokumentu, tak i v kategorii. Počet termů kategorie je neměnný, zatímco testované články mají různou délku. Zde samozřejmě vzniká problém, kdy dokument, který je delší a obsahuje stejný počet stejně relevantních slov pro zkoumanou kategorii jako kratší článek, získá vzhledem k součtu kladných vah jeho termů větší hodnotu TF-IDF.

Podobný problém budeme muset řešit i na úrovni kategorií. Pokud jsou kategorie různě velké, lze očekávat vyšší výsledné váhy pro obsáhlejší kategorie.

Vzniklé rozpory by měla odstranit vhodná normalizace váhy TF-IDF pro dokument. Hledáme tedy nějakou hodnotu, kterou bychom TF-IDF váhu dokumentu vydělili a tím získali hodnotu, která bude určovat relevantnost dokumentu ke kategorii bez ohledu na velikost kategorie a testovaného dokumentu.

V následujících třech experimentech se pokusíme empiricky zjistit vhodnou normalizaci, která by nám umožnila nastavit prahovací hodnotu bez ohledu na velikost kategorie. Ve čtvrtém experimentu se naopak pokusíme určit práh na základě velikosti hodnot TF-IDF pro kategorii.

8.3 Testovací data a trénovací kolekce pro experimenty

Jak bylo uvedeno v 3.1 velice často se pro klasifikaci textu využívají reuters kolekce. My jsme z nich pro naše experimenty vybrali deset testovacích dokumentů. Při výběru byla snaha nalézt dokumenty různých kategorií. Dalším požadavkem na data bylo zařazení každého článku do více kategorií v trénovací kolekci, což umožňuje testovat multi-label klasifikaci. Posledním kritériem výběru bylo, aby dokument měl alespoň několik řádků obsahu a tedy byl pro klasifikaci vhodný.

Náhodně vybraných deset dokumentů splňujících daná kritéria obsahuje 18 různých kategorií s různou četností v reuters kolekcích. Pro dokumenty budeme v experimentech používat označení *doc1 - doc10*. Kategorie vybraných článků lze nalézt v tabulce 8.1.

doc1	sugar	acq			
doc2	money-supply	money-fx	dlr		
doc3	gnp	trade			
doc4	grain	rice			
doc5	money-fx	dlr	yen	bop	gnp
doc6	crude	nat-gas			
doc7	grain	corn	rice		
doc8	trade	crude	nat-gas		
doc9	grain	corn	cornglutenfeed	meal-feed	
doc10	crude	fuel	jet		

Tabulka 8.1: Zastoupení kategorií ve vstupních datech.

Trénovací kolekce pro experimenty byla vytvořena ze souborů *reut2-005.sgm - reut2-015.sgm*. Kolekce obsahuje 10 000 dokumentů z různých kategorií. Některé dokumenty však nemají přiřazeny žádné kategorie, jiné jich mají naopak více.

Zajímavou charakteristikou kolekce vzhledem ke kvalitě klasifikace je počet dokumentů v jednotlivých kategoriích. Od počtu dokumentů v kategorii, na které se zde zaměříme nejvíce, se totiž odvíjí přesnost naučení se charakteristik dané kategorie. Samozřejmě nebudeme uvádět počty pro všechny kategorie, ale omezíme se pouze na ty, které obsahuje naše testovací sada dokumentů.

Druhým kritériem, z kterého by mohla být souzena kvalita naučení se klasifikátoru pro jistou kategorii, je počet různých termů v ní obsažený. Na první pohled by se mohlo zdát, že se jedná o stejnou charakteristiku jako v případě počtu dokumentů v kategorii. Ale vzhledem k tomu, že každý term může být v kategorii zastoupen pouze jednou a trénovací dokumenty jsou různě dlouhé, neplatí zde, že četnost termů v kategorii je přímo úměrná počtu dokumentů v kategorii. Jasným důkazem je nahlédnutí do tabulky 8.2, která zobrazuje počty dokumentů a slov pro kategorie z našich testovacích dokumentů, kde kategorie *crude* obsahuje 225 dokumentů a 4337 různých termů, zatímco kategorie *money-fx* má 378 dokumentů, ale pouze 4300 termů.

kategorie	dokumenty	termy	kategorie	dokumenty	termy
acq	1176	8598	**bop**	51	1396
sugar	89	2342	**crude**	225	4337
money-fx	378	4300	**nat-gas**	52	1906
mouney-supply	85	1474	**corn**	136	2860
dlr	101	1951	**cornglutenfeed**	0	0
gnp	68	1872	**meal-feed**	26	1175
trade	272	4143	**fuel**	10	385
grain	326	4480	**jet**	4	97
rice	32	1771	**yen**	35	1345

Tabulka 8.2: Počty dokumentů a termů v kategoriích, které jsou zastoupeny v testovacích datech.

Z tabulky 8.2 je patrné, že naše testovací data obsahují kategorie s výrazně odlišnými počty dokumentů a termů v trénovací kolekci, a proto by měla být vhodným vzorkem pro naše experimenty. Navíc námi vybraná množina obsahuje jak nejobsáhlejší kategorii *acq*, tak i kategorii s nulovým zastoupením v trénovací kolekci. Touto kategorií je *cornglutenfeed*, která neobsahuje žadný dokument a tudíž ani žádný term, je však obsažena v našich testovacích datech. Zde můžeme s jistotou říct, že tato kategorie nebude naším klasifikátorem nikdy rozpoznána, neboť její klasifikace nemohla být nacvičena v trénovací fázi. Hodnota vah dané kategorie bude tedy vždy 0, a proto ji nebudeme zahrnovat do výsledků.

8.4 Experiment 1: Normalizace délkou testovaného dokumentu

V našem prvním experimentu zjistíme výsledné váhy pro data ze sekce 8.3 a pokusíme se pro ně najít optimální prahovací hodnotu. Normalizace TF-IDF zde bude provedena pouze délkou testovaného dokumentu. Vliv obsáhlosti kategorie nebude nijak korigován.

V tabulce 8.3 vidíme výsledky prvního experimentu pro naše testovací data. Jsou zde zobrazeny pouze kategorie, které jsou v testovacích datech zastoupeny. Modré číslice v tabulce jsou hodnoty vah pro dokumenty, které patří do dané kategorie. Sloupce tabulky byly seřazeny podle obsáhlosti jednotlivých kategorií. Vlevo jsou největší kategorie a vpravo nejmenší.

Z tabulky 8.3 je dobře patrné, jaký vliv má na výslednou váhu testovaného dokumentu velikost kategorie. V levé části tabulky jsou výrazně vyšší hodnoty než v pravé části, což odpovídá seřazení od nejobsáhlejší kategorie po nejméně obsáhlou.

Z výsledných vah je jasné, že určení jedné prahovací hodnoty, která by rozlišila všechny dokumenty patřící do kategorie, je zcela nemožné. To potvrzuje i graf 8.2, který znázorňuje výsledky prvního experimentu. Modře jsou zobrazeny váhy dokumentů, které mají být klasifikovány a černě váhy dokumentů, které do kategorie nepatří. Z grafu jsme se pokusili určit práh, který by měl nejlepší poměr úspěšnosti klasifikace a chybovosti. Za prahovací hodnotu jsme vybrali váhu 26,56(červená rovnoběžka s vodorovnou osou). Pro tu jsme vypočítali úspěšnost klasifikace, tedy poměr mezi všemi dokumenty, které měly být klasifikovány(modrá barva na grafu a v tabulce) a dokumenty, které skutečně byly klasifikovány. Vyšla úspěšnost: $22/28 * 100 = 79\%$. Zde je však nutné ještě zohlednit chybovost, což je

	acq	fx	grain	trade	crude	corn	dlr	sugar	sup
doc1	119,54	68,99	62,83	65,70	44,07	52,75	24,41	21,39	7,77
doc2	98,84	93,99	67,22	62,85	50,06	58,54	26,56	15,59	19,61
doc3	105,29	106,13	54,20	82,57	54,30	46,14	39,64	13,59	11,36
doc4	117,35	82,06	113,89	87,52	79,91	102,36	30,20	23,31	9,58
doc5	121,27	117,97	96,67	95,45	64,07	88,89	40,62	19,74	15,12
doc6	114,89	54,38	68,54	45,47	74,49	57,89	12,33	14,99	11,76
doc7	111,82	69,92	109,18	71,04	55,23	89,86	18,39	21,14	10,42
doc8	100,93	74,89	71,08	91,01	65,38	57,60	25,49	16,65	6,88
doc9	104,67	70,08	100,45	74,51	52,14	94,33	25,32	19,33	7,84
doc10	94,61	50,03	50,80	50,64	69,23	39,95	16,62	12,52	6,32
	gnp	**n-g**	**bop**	**yen**	**rice**	**m-f**	**fuel**	**jet**	**cgf**
doc1	17,47	11,13	11,62	17,70	10,42	2,94	0,55	0,17	0
doc2	28,30	25,04	22,88	20,02	13,95	4,34	1,55	0,20	0
doc3	30,57	20,59	18,80	24,78	11,49	2,26	2,02	0,20	0
doc4	29,62	50,03	17,33	20,57	23,63	6,18	2,74	0,47	0
doc5	54,60	32,87	27,35	39,16	16,07	5,27	2,45	0,54	0
doc6	21,33	39,43	15,30	13,31	13,03	7,98	3,48	0,90	0
doc7	23,90	20,03	19,35	18,66	28,65	6,30	2,39	0,67	0
doc8	18,61	34,13	12,94	16,07	11,61	3,08	1,61	0,43	0
doc9	18,06	20,94	10,86	15,56	15,43	5,23	1,52	0,66	0
doc10	15,88	31,75	8,59	13,16	8,80	4,46	3,42	0,76	0

Tabulka 8.3: Tabulka vah pro experiment 1(*fx* je zkratka pro *money-fx*, *supply* zkratka pro *money-supply*, *cgf* označuje *corngluten feed*, *n-g* je *nat-gas* a *m-f* je *meel-feed*).

poměr mezi dokumenty, které byly klasifikovány a podle reuters kolekce neměly být, a všemi námi klasifikovanými dokumenty. Chybovost pro práh 26,56 je: $54/76 * 100 = 71\%$.

Výsledky prvního experimentu nám tedy neumožnily určit optimální prahovací hodnotu, ale lze z nich vyčíst funkčnost našeho klasifikátoru a také odhalit dokumenty, které jsou pro klasifikaci problémové.

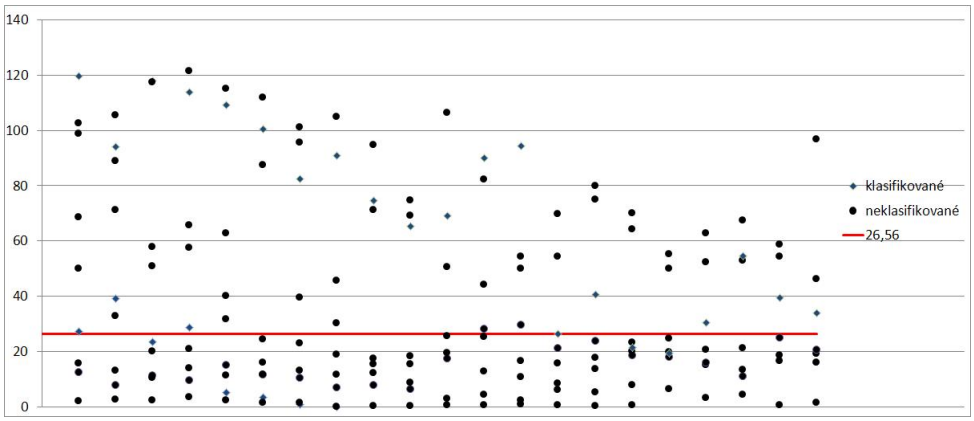
Obtížná se zdá být klasifikace dokumentu *doc4*, který má nejvyšší váhy ze všech dokumentů v kategoriích *crude, corn, sugar* a *nat-gas*. V reuters kolekci byl však označen pouze kategoriemi *grain* a *rice*. Příčínou těchto problémů je patrně malý obsah dokumentu, a proto ho nejsme schopni uspokojivě zařadit.

Podíváme-li se pozorně na modře vyznačené hodnoty v tabulce 8.3, zjistíme, že velice často se jedná o nejvyšší hodnoty ve sloupci nebo alespoň jedny z nejvyšších. Čím vyšší je hodnota v daném sloupci, tím by měl být dokument relevatnější pro danou kategorii. Náš klasifikátor má tedy dobrou rozlišovací schopnost uvnitř kategorie. Z výše uvedeného vyplývá, že např. *doc5* je ze všech dokumentů nejrelevantnější vzhledem ke kategorii *money-fx*, neboť má nejvyšší hodnotu ve sloupci. Což ale nutně neznamená, že dokument do této kategorie patří.

Nyní se pokusme určit úspěšnost klasifikátoru při odhalování nejrelevantnějšího dokumentu v kategorii. Za správnou klasifikaci budeme uvažovat situaci, kdy nejvyšší hodnota v rámci kategorie bude patřit dokumentu, který je touto kategorií označen v reuters kolekci. Z tabulky 8.3 lze zjistit, že 8-krát bylo toto kritérium splněno a 9-krát byl klasifikátor

neúspěšný. Jeho úspěšnost v rámci kategorie by tak byla jen 47%. Tato hodnota je však výrazně ovlivněna problémovým dokumentem *doc4*. Pokud bychom výsledky pro *doc4* zanedbali, zjistili bychom, že správná klasifikace nastala ve 13-ti případech a neúspěch byl zaznamenán pouze 4-krát. Úspěšnost při opomenutí *doc4* by se tak vyšplhala na 76%. Navíc ze čtyř neúspěšných případů nastaly tři u nejmenších kategorií v kolekci, což může být následek nedostatečného natrénování těchto kategorií z důvodu jejich malého rozsahu.

Na základě výsledků prvního experimentu jsme tedy schopni s drobnou optimalizací zjistit, který dokument je pro kategorii nejrelevatnější se 76% úspěšností. Máme však problém identifikovat, které kategorie jsou relavantní pro daný dokument. Dále víme, že dokument *doc4* není vhodný pro klasifikaci, a tak v dalších experimentech nemusíme brát ohled na jeho hodnoty.



Obrázek 8.2: Graf pro experiment 1. Černě hodnoty vah dokumentů nepatřících do kategorie. Modře hodnoty vah dokumentů patřících do kategorie. Červeně zobrazen zvolený práh.

8.5 Experiment 2: Normalizace délkou dokumentu a počtem termů v kategorii

Jelikož první experiment ukázal, že váhy dokumentů pro kategorie bude ještě potřeba upravit, aby byly porovnatelné mezi kategoriemi, vyzkoušíme nyní váhy normovat počtem termů v kategorii.

Výsledky našeho druhého experimentu jsou zobrazeny v tabulce 8.4. Hodnoty vah v tabulce jsou vynásobeny 1000-krát. Normalizace počtem termů v dokumentu a počtem termů v kategorii nám tedy dává hodnoty v řádech setin až tisícin. Vzhledem k tomu, že v průběhu výpočtu nebylo použito žádné zaokrouhlování a přesnost čísel byla stejná jako přesnost typu *double* v jazyce Java, nepřináší takto nízké hodnoty žádné zkreslení z pohledu klasifikace.

Už na první pohled je z tabulky 8.4 patrné, že tato normalizace nezvýhodňuje větší kategorie před menšími tak, jako tomu bylo v předchozím případě. Hodnoty dokumentů, které by měly být klasifikovány, jsou však vcelku ve velkém rozmezí $\langle 0{,}0045; 0{,}03298\rangle$ v porovnání s intervalem všech výsledných hodnot $\langle 0{,}00142; 0{,}03298\rangle$. To nasvědčuje tomu, že i tato normalizace nebude příliš vhodná pro následné prahování.

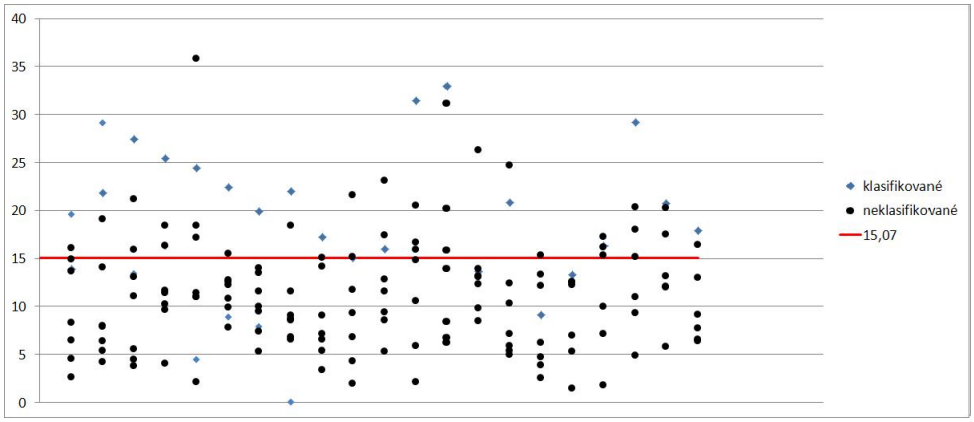
Úspešnost klasifikátoru uvnitř kategorie by měla zůstat stejná jako v případě experimentu 1, jelikož všechny váhy v jedné kategorii dělíme stejnou konstantou.

Na obrázku 8.3 jsou hodnoty z tabulky 8.4 zobrazeny graficky. Opět jsme se z grafu pokusili určit nejvhodnější práh. Jeho hodnotu jsme stanovili na 15,07, pro kterou nám vyšla úspěšnost klasifikace $20/28 * 100 = 71\%$ a chybovost je v tomto případě $32/52 * 100 = 62\%$. Vyšlé hodnoty jsou tedy v důsledku téměř shodné s hodnotami z prvního experimentu, kde jsme sice dosáhli větší úspěšnosti, ale také větší chybovosti.

V tabulce se opět jeví jako problémové tři nejmenší kategorie, jejichž váhy jsou výrazně nižší než u ostatních kategorií. Zajímavé jsou také relativně nízké váhy pro nejobsáhlejší kategorii *acq*, což by mohlo naznačovat znevýhodňování výrazně větších kategorií. To je patrně dáno tím, že testovací dokument obsahuje omezené množství termů, které přiřadí dokumentu váhu vzhledem ke kategorii a v kategorii tak zbyde mnoho termů, jejichž váha není započtena. Avšak výsledná váha je dělena všemi termy v kategorii. Další zvláštností jsou nízké hodnoty u kategorií *sugar* a *money-supply*. Přitom tyto kategorie se na první pohled nijak výrazně neliší od ostatních. To by mohlo nasvědčovat jisté nevyzpytatelnosti tohoto způsobu normování a tedy jeho nevhodnosti pro klasifikaci.

	acq	**fx**	**grain**	**trade**	**crude**	**corn**	**dlr**	**sugar**	**sup**
doc1	13,90	16,04	14,02	15,86	10,16	18,43	12,51	9,13	5,27
doc2	11,50	21,86	15,01	15,17	11,54	20,47	13,61	6,66	13,30
doc3	12,25	24,68	12,10	19,93	12,52	16,13	20,32	5,80	7,71
doc4	13,65	19,08	25,42	21,13	18,43	35,79	15,48	9,95	6,50
doc5	14,10	27,43	21,58	23,04	14,77	31,08	20,82	8,43	10,26
doc6	13,32	12,45	15,30	10,98	17,18	20,24	6,32	6,40	7,98
doc7	13,01	16,26	24,37	17,15	12,73	31,42	9,43	9,02	7,07
doc8	11,74	17,42	15,87	21,99	15,07	20,14	13,06	7,11	4,67
doc9	12,17	17,23	22,42	17,98	12,02	32,98	12,98	8,25	5,32
doc10	11,00	11,63	11,34	12,22	15,96	13,97	8,52	5,35	4,29
	gnp	**n-g**	**bop**	**yen**	**rice**	**m-f**	**fuel**	**jet**	**cgf**
doc1	9,33	5,84	8,32	13,16	5,88	2,50	1,42	1,72	0
doc2	15,12	13,14	16,39	14,88	7,88	3,77	4,04	2,08	0
doc3	16,33	10,81	13,46	18,42	6,49	1,92	5,26	2,11	0
doc4	15,82	26,25	12,41	15,30	13,34	5,26	7,13	4,85	0
doc5	29,17	17,44	19,59	29,11	9,07	4,49	6,36	5,55	0
doc6	11,40	20,69	10,96	9,89	7,36	6,79	9,04	9,28	0
doc7	12,77	10,51	13,86	13,88	16,18	5,36	6,21	6,90	0
doc8	9,94	17,91	9,27	11,95	6,55	2,62	4,18	4,41	0
doc9	9,65	10,99	7,78	11,57	8,71	4,45	3,36	6,75	0
doc10	8,49	16,66	6,15	9,78	4,97	3,80	8,88	7,88	0

Tabulka 8.4: Tabulka vah(*fx* je zkratka pro *money-fx*, *supply* zkratka pro *money-supply*, *cgf* označuje *corngltenfeed*, *n-g* je *nat-gas* a *m-f* je *meel-feed*). Všechny hodnoty vynásobeny 1000-krát.



Obrázek 8.3: Graf pro experiment 2. Černě hodnoty vah dokumentů nepatřících do kategorie. Modře hodnoty vah dokumentů patřících do kategorie. Červeně zobrazen zvolený práh.

8.6 Experiment 3: Normalizace délkou dokumentu a počtem dokumentů v kategorii

Vzhledem k tomu, že předchozí dvě normalizace se neukázaly býti vhodnými pro následné prahování, vyzkoušíme nyní vydělit váhu TF-IDF testovaného dokumentu počtem jeho termů a počtem dokumentů v kategorii.

Výsledky třetího experimentu jsou uvedeny v tabulce 8.5 a jsou vynásobeny stem, z důvodu lepší čitelnosti. Na první pohled vidíme velký rozptyl v hodnotách mezi kategoriemi, což svědčí o tom, že ani tento pokus nebyl příliš úspěšný v hledání optimální normalizace. Dokumenty, které měly být klasifikovány(v tabulce modré hodnoty), mají váhy v intervalu $\langle 0{,}1017; 1{,}1188\rangle$, což je značná část z intervalu všech hodnot $\langle 0{,}0418; 1{,}1188\rangle$, a proto bude obtížné najít vhodnou prahovací hodnotu.

Z tabulky 8.5 je patrné, že při tomto způsobu normalizace se již do výsledných vah příliš neodráží velikost kategorie. Snad jedinou výjimkou je kategorie *acq*, jejíž váhy jsou překvapivě nízké. Nižší váhy vidíme také u malých kategorií, ale v tomto případě bude vina spíše na straně jejich malé natrénovanosti než na straně použité normalizace. Zůstal zde však problém s velkým rozptylem hodnot mezi kategoriemi.

I třetí experiment by měl mít stejnou úspěšnost určení nejrelevatnějšího dokumentu pro kategorii, jako měly předchozí pokusy, neboť hodnota počtu dokumentů v kategorii je uvnitř kategorie neměnná.

Výsledky třetího experimentu jsou graficky znázorněny na obrázku 8.4. I zde jsme se z grafu pokusili určit nejvhodnější práh, kterým se zdá být hodnota 29,06, pro kterou je úspěšnost klasifikace $21/28 * 100 = 75\%$ a chybovost vyšla $48/69 * 100 = 70\%$. Opět jsme tedy získali velice podobné hodnoty jako v předchozích případech.

8.7 Experiment 4: Prahování podle nejvyšší váhy v kategorii

První tři experimenty byly zaměřeny na nalezení vhodné normalizace, která by dokázala značně rozdílné váhy kategorií převést do vah, které by nebyly tolik ovlivněny velikostí kategorie a bylo by je možné porovnat i mezi kategoriemi. Bohužel vhodnou normalizaci se nám

	acq	fx	grain	trade	crude	corn	dlr	sugar	sup
doc1	10,17	18,25	19,27	24,16	19,58	38,77	24,17	24,03	9,14
doc2	8,40	24,86	20,62	23,11	22,45	43,05	26,30	17,52	23,07
doc3	8,95	28,08	16,63	30,36	24,13	33,93	39,25	15,27	13,37
doc4	9,98	21,71	34,94	32,18	35,52	75,26	29,90	26,20	11,27
doc5	10,31	31,21	29,65	35,09	28,47	65,36	40,22	22,18	17,79
doc6	9,74	14,39	21,02	16,72	33,11	42,57	12,21	16,84	13,83
doc7	9,51	18,50	33,49	26,12	24,54	66,07	18,21	23,75	12,26
doc8	8,58	19,81	21,80	33,49	29,06	42,36	25,24	18,71	8,09
doc9	8,90	19,60	30,81	27,39	23,17	69,36	25,07	21,72	9,23
doc10	8,05	13,23	15,58	18,62	30,77	29,37	16,46	14,07	7,44
	gnp	**n-g**	**bop**	**yen**	**rice**	**m-f**	**fuel**	**jet**	**cgf**
doc1	25,70	21,40	22,79	50,58	32,56	11,30	5,45	4,18	0
doc2	41,62	48,15	44,86	57,20	43,59	17,06	15,55	5,03	0
doc3	44,96	39,61	36,86	70,79	35,92	8,68	20,25	5,11	0
doc4	43,55	96,21	33,97	58,78	73,85	23,77	27,44	11,77	0
doc5	80,30	63,21	53,62	111,88	50,21	20,28	24,49	13,46	0
doc6	31,37	75,82	30,00	38,02	40,71	30,70	34,80	22,51	0
doc7	35,14	38,51	37,95	53,32	89,52	24,23	23,92	16,73	0
doc8	27,37	65,64	25,37	45,92	36,27	11,84	16,09	10,69	0
doc9	26,56	40,27	21,30	44,46	48,21	20,11	15,23	16,38	0
doc10	23,36	61,07	16,84	37,59	27,51	17,15	34,20	19,10	0

Tabulka 8.5: Tabulka vah(*fx* je zkratka pro *money-fx*, *supply* zkratka pro *money-supply*, *cgf* označuje *corngltenfeed*, *n-g* je *nat-gas* a *m-f* je *meel-feed*). Všechny hodnoty vynásobeny 100-krát.

nepovedlo nalézt, proto se nyní pokusíme navrhnout jiný postup pro zařazení dokumentu do kategorie.

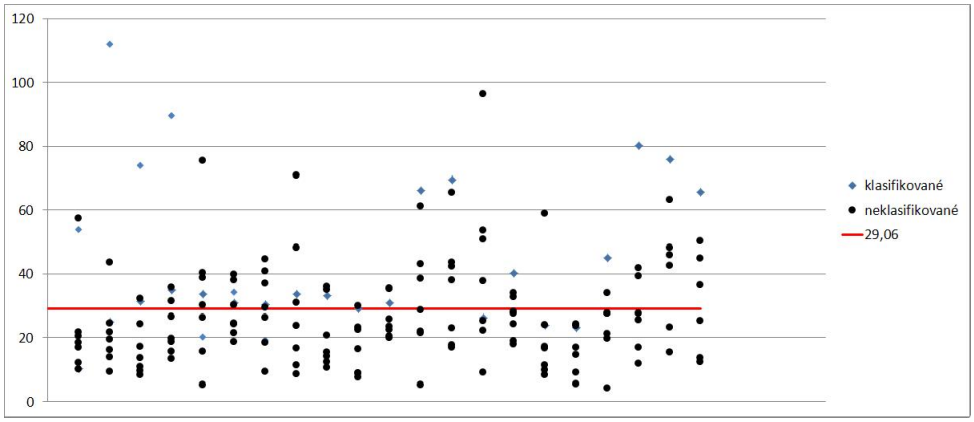
Jelikož dokumenty pro jednotlivé kategorie mají značně rozdílné váhy, mohli bychom se pokusit nalézt vhodnou prahovací hodnotu na základě nejvyšší váhy v kategorii. V experimentu 1 jsme ověřili, že nejrelevantnější dokumenty pro kategorii v ní mají nejvyšší váhy. Naším cílem tak nyní je najít konstantu k, pro kterou platí:

$$k = w * w_{max}, \tag{8.1}$$

kde w_{max} je maximální váha v kategorii a w je nejmenší váha relevantního dokumentu. Tedy pokud dokument dosáhne v kategorii váhy větší jak w, zařadíme ho do dané kategorie.

K určení konstanty k využijeme výsledky z prvního experimentu. Z tabulky 8.3 vybereme kategorie, které mají mít zařazeny alespoň dva dokumenty. Najdeme dokument s maximální váhou v kategorii a dokument s nejmenší váhou, který má být zařazen do kategorie a dosadíme do vzorce 8.1. Následující seznam ukazuje výpočet a vyšlé konstanty pro vhodné kategorie.

- *money-fx*: $k = 93{,}99/117{,}97 => k = 0{,}80$
- *grain*: $k = 100{,}45/113{,}85 => k = 0{,}88$
- *trade*: $k = 82{,}57/91{,}01 => k = 0{,}91$



Obrázek 8.4: Graf pro experiment 3. Černě hodnoty vah dokumentů nepatřících do kategorie. Modře hodnoty vah dokumentů patřících do kategorie. Červeně zobrazen zvolený práh.

- *crude*: $k = 65{,}38/74{,}49 => k = 0{,}88$
- *corn*: $k = 89{,}86/94{,}33 => k = 0{,}95$
- *dlr*: $k = 26{,}56/40{,}62 => k = 0{,}65$
- *gnp*: $k = 30{,}57/54{,}60 => k = 0{,}56$
- *nat-gas*: $k = 34{,}13/39{,}43 => k = 0{,}87$
- *rice*: $k = 23{,}63/28{,}65 => k = 0{,}82$

Pro zvolené kategorie vyšla konstanta k v intervalu $\langle 0{,}56 * w_{max}; w_{max} \rangle$. Pro náš experiment zvolíme za k medián z předchozích hodnot. Tím je hodnota $k = 0{,}87$. Nyní vyzkoušíme prahování pomocí konstanty k na našich výsledcích z experimentu 1.

Tabulka 8.6 zobrazuje, jak moc byla úspěšná klasifikace pomocí prahování konstantou $k = 0{,}87$. Sloupec *úspěch* znamená, kolik dokumetů bylo správně označeno, že patří do dané kategorie. Sloupec *neúspěch* je počet dokumentů, které měly být zařazeny do zkoumané kategorie, ale nebyly. Sloupec *navíc* ukazuje počet dokumentů, které byly nesprávně zařazeny do kategorie. Jedná se opět o test, který určoval, zda daný dokument je relevantní pro danou kategorii, či nikoli.

Narozdíl od předchozích pokusů se zde snažíme určit všechny dokumenty, které do kategorie patří. Vyhodnocení výsledků je proto mírně odlišné od předchozích experimentů. Úspěšnost klasifikace v rámci kategorie nyní spočítáme tak, že vezmeme počet úspěšných klasifikací dokumentu do kategorie a vydělíme jej počtem správných klasifikací podle reuters kolekce. Dohromady máme 28 správných klasifikací dokumentů do kategorií podle reuters kolekce(což jsou modře zvyrazněné hodnoty v tabulce 8.3 bez kategorie *corngluten feed*). Povedlo se nám úspěšně zařadit 18 dokumentů. Z toho dopočítáme, že úspěšnost nalezení všech dokumentů patřících do kategorie, je 64%. Musíme však ještě přihlédnout k počtu klasifikací, které zařadily dokument do kategorie, do níž nepatří. Jedná se celkem o 19 klasifikací. Proto vypočítáme ještě míru chybovosti, což bude počet klasifikací dokumentů, které klasifikovány být neměly, vydělený počtem všech provedených klasifikací. Míra chybovosti je: $19/28 * 100 = 51\%$.

	w_{max}	w	**úspěch**	**neúspěch**	**navíc**
acq	121,27	104,00	1	0	7
money-fx	117,97	102,63	1	1	1
grain	113,89	99,08	3	0	0
trade	91,01	79,18	2	0	2
crude	79,91	69,52	1	2	1
corn	102,36	89,05	2	0	1
dlr	40,62	35,34	1	1	1
sugar	23,31	20,28	1	0	2
money-supply	19,61	17,06	1	0	0
gnp	54,60	47,50	1	1	0
nat-gas	50,03	43,53	0	2	1
bop	27,35	23,79	1	0	0
yen	39,16	34,07	1	0	0
rice	28,65	24,93	1	1	0
meel-feed	7,98	6,94	0	1	1
fuel	3,48	3,03	1	0	1
jet	0,9	0,78	0	1	1
celkem			18	10	19

Tabulka 8.6: Souhrn výsledků experimentu 4.

Z výsledků vyplývá, že jsme schopni nalézt všechny dokumenty patřící do kategorie s úspěšností 64%, ale přibližně polovina klasifikovaných dokumentů bude do kategorií přiřazena navíc.

Nyní se podíváme jaké kategorie by dokumenty získaly pomocí této prahovací metody. V závorkách jsou uvedeny kategorie, do kterých by články měly patřit podle reuters kolekce.

- *doc1* – *acq, sugar*(*acq, sugar*)
- *doc2* – *money-supply*(*money-supply, money-fx, dlr*)
- *doc3* – *acq, money-fx, trade, dlr*(*gnp, trade*)
- *doc4* – *acq, grain, trade, crude, corn, sugar, nat-gas, rice*(*grain, rice*)
- *doc5* – *acq, money-fx, trade, dlr, gnp, bop, yen*(*money-fx, dlr, yen, bop, gnp*)
- *doc6* – *acq, crude, meal-feed, fuel, jet*(*crude, nat-gas*)
- *doc7* – *acq, grain, corn, sugar, rice*(*grain, corn, rice*)
- *doc8* – *trade*(*trade, crude, nat-gas*)
- *doc9* – *acq, grain, corn*(*grain, corn, meel-feed*)
- *doc10* – *fuel*(*crude, fuel, jet*)

V tabulce 8.7 vidíme přehledně výsledky klasifikace dokumentů pro prahovací konstantu 0,87. Sloupec *nalezené* obsahuje počet kategorií, které byly dokumentu správně přiřazeny.

Dokument	**nalezené**	**nenalezené**	**navíc**
doc1	2	0	0
doc2	1	2	0
doc3	1	1	3
doc4	2	0	6
doc5	5	0	2
doc6	1	1	4
doc7	3	0	2
doc8	1	2	0
doc9	2	1	1
doc10	1	2	0
celkem	19	9	18

Tabulka 8.7: Souhrn výsledků klasifikace dokumentů.

Sloupec *nenalezené* určuje, kolik kategorií nebylo pro dokument nalezeno. Sloupec *navíc* zobrazuje počet kategorií, které byly dokumentu přiřazeny navíc.

Vždy se povedlo určit alespoň jednu kategorii. V pěti případech se povedlo určit i druhou kategorii dokumentu. Velkým problémem je však počet klasifikací, které jsou navíc. Při zvoleném $k = 0{,}87$ je tento počet téměř stejný jako počet správných klasifikací. To svědčí o tom, že byla zvolena příliš nízká prahovací konstanta k.

Nyní se pokusíme vyčíslit úspěšnost klasifikátoru tak, jak jsme ji počítali v předchozím případě pro kategorie. Z deseti případů jsme v deseti našli alespoň jednu kategorii. Úspěšnost nalezení první kategorie je tedy 100%. Nejméně dvě kategorie se nám povedlo nalézt v polovině případů, úspěšnost nalezení druhé kategorie je tedy 50%. Tyto úspěšnosti jsou opravdu vysoké, ale je třeba k nim připočíst ještě chybovost, která je v tomto případě rovněž vysoká: $18/37 = 49\%$, tedy téměř každá druhá nalezená kategorie je navíc. Pro porovnání s předchozími experimenty zde ještě spočítáme úspěšnost pro určení všech kategorií, která ve čtvrtém experimentu je $19/28{*}100 = 68\%$. Chybovost je již dříve spočtených 49%. Z toho plyne, že se nám touto metodikou určení prahu povedlo snížít chybovost při zachování téměř stejné úspěšnosti jako v případě normalizace.

Metoda je velmi citlivá na extrémní hodnoty. Pokud dostaneme pro nějaký dokument váhu výrazně vyšší než pro zbytek dokumentů v kategorii, nejsme následně schopni do této kategorie klasifikovat jiný dokument. Příkladem jmenovaného problému je dokument *doc4*, který v našem testu značně ovlivnil klasifikování kategorií *corn* a zejména *nat-gas*. Dalším evidentním problémem je klasifikace do kategorie *acq*, která především díky své velikosti získává pro většinu dokumentů vysokou váhu. Opačný problém se vyskytuje u tří nejmenších kategorií, které nejsou dostatečně natrénované a algoritmus je tak spíše "tipuje"než určuje. I přes všechny zmíněné problémy jsme touto metodou získali nejlepší výsledky klasifikace.

Kapitola 9

Závěr

V teoretické části nás práce uvedla do problematiky multi-label klasifikace textových dokumentů. Seznámili jsme se s reprezentací textových dokumentů vhodnou pro text mining. Rozebrali jsme druhy rysů, kterými můžeme dokument charakterizovat. Představili jsme si typickou architekturu systémů pro dolování z dat. Dále zde byly rozebrány důležité pojmy shlukování a jeho základní algoritmy.

Ukázali jsme si reprezentaci dokumentu pomocí vektoru rysů. Zabývali jsme se získáváním a hodnocením jednotlivých rysů a redukcí počtu všech rysů. Dále byly rozebrány vybrané přístupy ke klasifikaci pomocí strojového učení. Podrobně byla uvedena metoda matrix regression, kterou jsme zvolili pro implementaci.

V praktické části práce jsme vytvořili zadání programu, který bude klasifikovat textové dokumenty pomocí algoritmu matrix regression. V návrhu jsme použili architekturu MVC. Její vrstvy byly podrobněji navrženy pomocí diagramu tříd. Podle návrhu byl následně naimplementován program, který jsme nejprve otestovali na malé kolekci, aby se ověřila funkčnost námi vytvořené klasifikace. Následně byla naše aplikace využita ke třem experimentům, jejichž cílem bylo najít vhodnou normalizaci výsledných vah, aby bylo možné určit jednu prahovací hodnotu pro všechny kategorie, která by oddělila relevantní třídy od tříd, do kterých dokument nepatří. Vzhledem k tomu, že vyzkoušené normalizace nedokázaly srovnat váhy kategorií tak, aby bylo možné nalézt prahovací hodnotu s dobrou úspěšností, byl proveden ještě čtvrtý experiment, který se pokusil o prahování pomocí nejvyšší váhy v kategorii.

Experimenty jsme potvrdili funkčnost naší implementace, jelikož algoritmus ve většině případů přiřadil nejvyšší váhu v kategorii dokumentu, který do ní skutečně patřil. Normalizace a následné prahování se ale ukázalo býti dosti obtížné.

Nejúspěšnějším experimentem na normalizaci podle spočítané úspěšnosti(79%) a chybovosti(71%) byl první experiment, ve kterém byla výsledná váha vydělena pouze počtem termů v testovaném dokumentu. To znamená, že klasifikace v tomto případě byla silně ovlivněna velikostí kategorie a úspěšně klasifikovány byly pouze dokumenty patřící do obsáhlejších kategorií.

Z normalizací, které měly odstranit znevýhodňování malých kategorií, dopadla o něco málo lépe ta, která byla použita v experimentu 2. Zde byly váhy děleny počtem termů v testovaném dokumentu a počtem termů v kategorii. Úspěšnost klasifikace v tomto případě byla 71% a chybovost vyšla 62%. Tato normalizace ukázala jistou míru nestability, poněvadž některým kategoriím přiřadila hodnoty podstatně nižší než jiným a tím prakticky znemožnila označení dokumentu těmito kategoriemi.

Ve třetím experimentu jsme váhu zkusili normovat počtem termů v dokumentu a počtem

dokumentů v kategorii. Tento pokus dopadl nejhůře. Úspěšnost sice vyšla 75% ale s chybovostí 70%. Dalším problémem byly značně nevyrovnané hodnoty mezi kategoriemi, které znemožňují lepší určení prahovací hodnoty.

Poslední experiment dosáhl nejlepších výsledků. Jeho cílem nebylo zarovnat váhy kategorií tak, aby měly stejnou vypovídající hodnotu, ale zjištění prahovací konstanty, která po vynásobení s nejvyšší váhou v kategorii oddělí dokumenty, které do kategorie patří od těch, co do ní nepatří. Tento postup dosáhl úspěšnosti 68% při chybovosti „pouhých“ 49%, což se jeví jako nejlepší výsledek ze všech čtyř experimentů. Problémem tohoto určení relevatních kategorií jsou extrémní nejvyšší hodnoty. Pokud nějaký dokument v kategorii získá výrazně vyšší hodnotu než ostatní, znemožní tím klasifikaci dalších dokumentů do této kategorie. Velkou výhodou je, že nemusíme hledat vhodnou normalizaci.

Z uvedených experimentů je patrné, jak složité je najít správné vyhodnocení výsledků pro multi-label klasifikace. Zde uváděná úspěšnost je hodnotou, v kolika procentech případů jsou odhaleny všechny kategorie dokumentu. Problémem však je, že s rostoucí úspěšností roste míra chybovosti, tedy počet dokumentů, které byly do kategorie přiřazeny navíc. Zde má zásadní vliv volba prahovací hodnoty. Samozřejmě pokud zvolíme nízký práh, získáme vysokou úspěšnost ale také vysokou chybovost. Naopak když máme vysoký práh, dostáváme nízkou úspěšnost i chybovost.

Při volbě prahu by měl být zohledněn především účel aplikace. Pokud potřebujeme získat všechny dokumenty patřící do kategorie a není pro nás kritické, že některý dokument v ní bude navíc, zvolíme vysoký práh. Jsme-li naopak v situaci, kdy potřebujeme roztřídit velké množství dokumentů a není pro nás limitující, že některé dokumenty nebudou do své kategorie zařazeny, určíme nízký práh.

Výsledky experimentů jsou samozřejmě ovlivněny námi zvolenou sadou testovacích dokumentů. Tedy uváděné úspěšnosti a chybovosti jsou pouze přibližné. Dále se v nich projevuje charakter reuters kolekce, která neobsahuje úplně ideální testovacími data. Nicméně byla použita pro naše experimenty z důvodu velkého počtu roztříděných dokumentů.

Předmětem dalších prací na základě našich výsledků by měly být další pokusy o nalezení vhodné normalizace, kterou naše experimenty neodhalily. Velký prostor pro zlepšení se také nachází u určení prahovací konstanty pro nejvyšší váhu v kategorii, kde bude potřeba omezit vliv extrémní nejvyšší váhy na klasifikaci. Bylo by také vhodné provést další experimenty na větší množině testovacích dat, neboť naše data byla značně ovlivněna krátkým dokumentem *doc4* a tři trénovací kategorie, měly v kolekci nízké zastoupení, tudíž naučení jejich klasifikace bylo značně omezené.

Zdá se, že samotný algoritmus matrix regression funguje velice dobře o čemž svědčí vysoká úspěšnost při hledání nejrelevantnějšího dokumentu uvnitř kategorie.

Literatura

[1] Bartík, V.: *Dolování z textu a na webu – přednáška z předmětu ZZN*. FIT VUT v Brně, 2010.

[2] Belasco, S.; Pavone, P.: Multi-class categorization based on cluster analysis and TFIDF [online]. Dostupné na URL: `http://lexicometrica.univ-paris3.fr/jadt/jadt2008/tocJADT2008.htm`, 2008 [cit. 2012-05-05].

[3] Burgetová, I.: *Shluková analýza – přednáška z předmětu ZZN*. FIT VUT v Brně, 2010.

[4] Chmelař, P.; Hellebrand, D.; Bártík, V.: Nalezení slovních kořenů v češtině [online]. Dostupné na URL: `ceur-ws.org/Vol-802/paper6.pdf`, 2011 [cit. 2012-10-05].

[5] Eckstein, R.: Java SE Application Design With MVC [online]. Dostupné na URL: `http://www.oracle.com/technetwork/articles/javase/mvc-136693.html`, 2007-03 [cit. 2012-05-01].

[6] Fan, R.-E.; Lin, C.-J.: A Study on Threshold selection for Multi-label Classification [online]. Dostupné na URL: `http://www.csie.ntu.edu.tw/ cjlin/papers/`, 2007-11-12 [cit. 2012-05-05].

[7] Feldman, R.; Sanger, J.: *The Text Mining Handbook*. Cambridge University Press, 2007, iSBN 0-521-83657-3.

[8] Lewis, D. D.: Reuters–21578 [online]. Dostupné na URL: `http://www.daviddlewis.com/resources/testcollections/reuters21578/`, 1997-09-26 [cit. 2012-04-28].

[9] Lukáš, R.: *Klasifikace a predikce – přednáška z předmětu ZZN*. FIT VUT v Brně, 2009.

[10] Manning, C. D.; Raghavan, P.; Schütze, H.: *Introduction to Information Retrieval*. Cambridge University Press, 2008, iSBN: 0521865719.

[11] Popa, I. S.; Zeitouni, K.; Gardirin, G.: Text Categorization for Multi-label Documents and many Categories [online]. Dostupné na URL: `http://dx.doi.org/10.1109/CBMS.2007.108`, 2007 [cit. 2011-12-05].

[12] Reuters: Reuters corpus [online]. Dostupné na URL: `http://about.reuters.com/researchandstandards/corpus`, 2000 [cit. 2012-04-28].

[13] Smirnov, I.: Overview of Stemming algorithms [online]. Dostupné na URL: `the-smirnovs.org/info/stemming.pdf`, 2008-12-03 [cit. 2012-05-10].

[14] Zendulka, J.; a kol.: *Získávání znalostí z databází ZZN – opora k předmětu ZZN*. FIT VUT v Brně, 2009.

[15] Zhu, X.: Basic text process [online]. Dostupné na URL: `http://pages.cs.wisc.edu/ jerryzhu/cs769/text_preprocessing.pdf`, 2010 [cit. 2012-10-05].