Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Automatická identifikace revizí textových dokumentů

Plzeň 2017 Filip Kupilík

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 30. června 2017

Filip Kupilík

Poděkování

Rád bych poděkoval Ing. Miloslavu Konopíkovi, Ph.D. za trpělivost, ochotu, pomoc a cenné rady poskytnuté při vedení práce.

Abstract

The goal of the thesis is to design, create and test an algorithm which identifies the revisions of test documents. The first part of the thesis is focused on analysing current approaches to document searching and a identification of highly similar documents (near-duplicates). The second part deals with a design and an implementation of a new algorithm. The efficiency of the algorithm is verified on a set of test documents. The tests results are compared with the results of the experiments which were done with the selected existing algorithms.

Abstrakt

Cílem této práce je navrhnout, vytvořit a otestovat algoritmus pro identifikaci revizí v množině textových dokumentů. V první části práce jsou zmapovány současné přístupy ve vyhledávání dokumentů a popsány stávající algoritmy pro identifikaci podobných dokumentů. Druhá část se zabývá návrhem a implementací algoritmu zaměřeného na detekci revizí, jehož úspěšnost je ověřena na vytvořené kolekci testovacích dokumentů. Výsledky získané z provedených experimentů jsou porovnány s výsledky vybraných stávajících algoritmů.

Obsah

1 Úvod **1**

2 Indexace a podobnost textových dokumentů **2**

2.1 Postup indexace . . . 2

2.1.1 Tokenizace . . . 2

2.1.2 Odstranění nevýznamových slov . . . 3

2.1.3 Normalizace . . . 3

2.1.4 Invertovaný seznam . . . 4

2.2 Reprezentace textových dokumentů . . . 4

2.2.1 Bag of words . . . 4

2.2.2 Booleovský model . . . 4

2.2.3 Vektorový model . . . 6

2.2.4 Latent Semantic Analysis . . . 6

2.2.5 Latent Dirichlet Allocation . . . 7

2.3 Váhování termů . . . 8

2.3.1 Binární váha . . . 8

2.3.2 Frekvence termu . . . 8

2.3.3 Inverzní frekvence dokumentu . . . 9

2.3.4 tf-idf . . . 9

2.4 Porovnávání dokumentů . . . 10

2.4.1 Euklidovská vzdálenost . . . 11

2.4.2 Kosinová podobnost . . . 11

2.4.3 Jaccardův index . . . 13

2.4.4 Kullback-Leibler divergence . . . 13

2.5 Vyhodnocení výsledků . . . 13

2.5.1 Preciznost . . . 13

2.5.2 Úplnost . . . 14

2.5.3 Přesnost . . . 14

2.5.4 F-míra . . . 15

3 Detekce téměř duplicitních dokumentů **16**

3.1 Shingling . . . 16

4 Rozpoznávání pojmenovaných entit **18**

4.1 Nástroje pro anglický jazyk . . . 18

4.2 Nástroj pro český jazyk . . . 19

5 Apache Lucene **20**
5.1 Podobnost v Lucene 20
5.1.1 Lucene scoring formula 20
5.2 More like this 22
5.3 Apache Tika 22

6 Revize **23**
6.1 Definice 23
6.2 Příklady revizí 24

7 Testovací množina dokumentů **26**

8 Implementace stávajících algoritmů **27**
8.1 Analýza výběru algoritmů 27
8.2 Použité technologie 27
8.3 Vytvoření indexu 27
8.4 TF-IDF a kosinová vzdálenost 28
8.5 K-shingling 29
8.6 More like this 29
8.7 Průběh experimentů 30
8.8 Výsledek experimentů 31

9 Implementace vlastního algoritmu **33**
9.1 Analýza a návrh řešení 33
9.2 Popis řešení 33
9.3 Výsledky testování 34

10 Závěr **37**

Literatura **38**

Seznam příloh **41**

A Uživatelská příručka **42**
A.1 Spuštění programu 42

B Obsah CD **43**

C Třídy českých jazykových modelů projektu NameTag **44**

D Ukázky zdrojových kódů **47**
D.1 Specifikace vlastností pole pro udržování textu 47
D.2 Získání ováhovaného vektoru z dokumentu 47

D.3 Vytvoření vlastního analyzéru na generování shinglů 48
D.4 Zjištění KL divergence mezi dvěma ohodnocenými dokumenty 49

1 Úvod

Ve velkých kolekcích textových dokumentů může postupným upravováním nějakého z nich vzniknout několik nových s podobným obsahem, z nichž ovšem jen jeden je aktuální a správný a všechny ostatní obsahují neaktuální a chybné informace. Tyto různé verze dokumentů nazýváme revize. Abychom předešli potenciálnímu problému, který by mohl nastat v případě, že by uživatel pracoval s dokumentem, který by obsahoval nepravdivé informace, je třeba tyto revize najít a nejlépe z kolekce odstranit.

Cílem této práce je zmapovat současné techniky a přístupy pro zpracování a vyhledávání podobných dokumentů. Dále bude vytvořena množina testovacích dokumentů s revizemi, na které budou otestovány některé ze stávajících algoritmů pro vyhledávání podobných dokumentů.

Na základě výsledků experimentů bude navržen a implementován algoritmus pro vyhledávání revizí. Úspěšnost vytvořeného algoritmu bude rovněž otestována a výsledky porovnány s hodnotami dosaženými vybranými existujícími metodami.

2 Indexace a podobnost textových dokumentů

Indexace je obecný pojem, pojmenovávající proces zpracování dat, který urychluje vyhledávání v množině textových dokumentů. Můžeme si snadno představit případ, kdy bychom chtěli ve velké kolekci textových dokumentů najít pouze ty, ve kterých se vyskytuje vybrané slovo. Jedním ze způsobů, jak tuto informaci získat, je projít všechny dokumenty jeden po druhém a u každého zjišťovat, zda se v něm hledané slovo vyskytuje. Takto se dá velmi jednoduchým způsobem dojít k výsledku, ovšem problém nastává při použití v opravdu velkých kolekcích, jelikož doba prohledávání všech dokumentů se lineárně navyšuje s počtem souborů v kolekci[20]. S výkonem moderních počítačů a ve skromných kolekcích se ovšem stále jedná o efektivní proces vyhledávání informací (*information retrieval* - informatický obor).

Řešením, jak se vyhnout zdlouhavému procesu postupného prohledávání pro každý dotaz, je dokumenty před vytvořením dotazu předzpracovat, neboli zaindexovat.

2.1 Postup indexace

Indexační algoritmus prochází zpracovávaný dokument, rozpoznává jednotlivá slova a každému z nich se snaží přiřadit odpovídající hodnotu, jak velkou mírou vypovídají o daném dokumentu. Každý dokument obsahuje různá slova, některá velmi dobře vypovídají o obsahu dokumentu, a v dalších dokumentech se například tolikrát vyskytovat nebudou. Ovšem obsahuje i taková slova, která nám o tématu, jímž se dokument zabývá, mnoho nepoví. Tato běžná slova se obvykle vyskytují ve velkém množství dokumentů v kolekci[20].

2.1.1 Tokenizace

Tokenizace je ta část, při které dochází k rozpoznávání slov a slovních spojení v dokumentu. Rozdělení textu na tokeny, nebo dále také termy, v našem jazyce je pro počítač jednoduchou záležitostí, problémy nastávají při použití např. zkratek, kdy stroj musí rozpoznat, zda se jedná o tečku oddělující věty, nebo zda jde právě o zkratku[23].

2.1.2 Odstranění nevýznamových slov

Nevýznamová slova, někdy také tzv. stop slova, jako například spojky, předložky nebo citoslovce jsou pro zapisování do indexu zbytečná, jelikož nic neříkají o významu textu, a tak je třeba je v textu odhalit a odstranit z indexace. Stop slova se detekují porovnáváním se slovníky nevýznamových slov. Slovníky stop slov se u každého světového jazyku liší.

2.1.3 Normalizace

V této fázi indexace je text rozdělen na jednotlivé termy, mezi kterými nejsou žádná stop slova.

Ještě předtím, než se tato slova zapíšou do indexu, je zapotřebí je upravit, normalizovat. Chceme totiž, aby algoritmus našel ke slovu v dotazu *hudba* shodu se slovy *hudby*, *Hudbu*, *hudbe* ale i například se slovem *hudební*. Normalizace se dělí na následující kroky.

Převedení na malá písmena - nejdříve se termy převedou na malá písmena. Když se takto převedou slova i v dotazu, algoritmus spolehlivě najde shodu i se slovy, která začínají větu, nebo například mezi *Czech* a *czech*. Problémem při tomto převádění je to, že hodně nejenom českých jmen nebo názvů společností vychází z obyčejných slov, tudíž se najde shoda mezi panem *Černým* a *černým* dnem[20].

Stematizace - tato technika se snaží ve slovech nalézt jejich kořen. To umožňuje detekovat shodu slova v základním tvaru v dotazu a stejného slova ovšem v množném čísle, nebo třeba ve tvaru přídavného jména. Systém by bez použití stemmingu takovouto shodu nenašel. Hledá kořen slov odstraněním přípon a předpon. Příkladem je extrakce slova *connect* ze slov *connected*, *connecting*, a *connection*. Stematizace má také druhotný efekt, jímž je redukce velikosti indexu, jelikož se indexují pouze tyto kořeny. Stematizace v některých případech zaostává, jelikož například v češtině dochází v různých tvarech slova ke změně v kořeni[12].

Lemmatizace - podobná operace jako stematizace, ale nehledá kořen, nýbrž lemmu, čili základní tvar slova. Dosahuje toho díky morfologické analýze. Ta se neprovádí nad jednotlivými slovy, ale nad celými větami, nebo nad většími částmi textu, aby se dal přesně určit význam daného termu[15]. Lemmatizátor musí mít k dispozici slovník, ve kterém může vyhledat danou základní formu slova. Pár příkladů aplikací stematizace a lemmatizace v anglickém jazyce je ukázáno v tabulce 2.1.

I když stematizace v některých případech nenalezne přesný základní tvar slova, nebo spojí do společného významu termy, které spolu obsahově nesou-

Původní tvar slova	Stematizace	Lemmatizace
regulated	regul	regulate
activations	act/activ	activation
actor	act	actor

Tabulka 2.1: Porovnání stematizace a lemmatizace

visí, tak její implementace je snazší, algoritmus je rychlejší a efektivita bývá vyšší než v případu lemmatizace[19].

2.1.4 Invertovaný seznam

Sestavení invertovaného seznamu (*inverted index*) je základní myšlenkou procesu indexace. Je to datová struktura, která umožňuje efektivní full-textové vyhledávání. V invertovaném seznamu se ukládají unikátní termy a ke každému z nich seznam ID dokumentů, ve kterých se vyskytují. Je často používaný indexačními nástroji.

2.2 Reprezentace textových dokumentů

V další fázi je třeba převést dokumenty do nějakého tvaru, abychom je mohli mezi sebou, například z hlediska podobnosti, porovnávat.

2.2.1 Bag of words

Bag of words je jednoduchá reprezentace textových dokumentů, ve které je dokument uchováván jen jako neuspořádaná sekvence termů vyskytujících se v něm. Pozice slov se nezohledňují, pořadí termů v dokumentu je ignorováno[15]. V tomto modelu je dokument s textem *Jan je vyšší než Petr* totožný s dokumentem *Petr je vyšší než Jan*[18]. Ovšem je patrné, že dokumenty, které obsahují stejná slova, mají podobný obsah, čehož využívají algoritmy pro hledání podobných dokumentů. Bag of words model slouží velmi často jako základ pro další typy reprezentace textových dokumentů.

2.2.2 Booleovský model

Booleovský model vytváří pro kolekci dokumentů matici výskytů, do které ukládá pro všechna slova, zda se v jednotlivých dokumentech nacházejí, či nikoliv. Používá pouze hodnoty booleovské algebry, 1 pro termy, které daný dokument obsahují, a 0 pro opačnou hodnotu. Uživatel poté může při vytváření dotazů používat různě komplikované booleovské výrazy. Ovšem ze

Tabulka 2.2: Ukázková matice výskytů

	d1	d2	d3	d4	d5	d6
auto	0	1	0	1	1	0
porucha	0	1	1	0	0	1
motor	1	1	1	1	1	1
dodávka	1	0	1	0	0	1
benzín	0	0	1	1	0	0
diesel	1	1	0	0	1	0

struktury tohoto modelu vychází několik nedostatků. Jelikož vyhledávání v tomto modelu je založeno jen na binárních rozhodovacích kritériích, tak dokumenty rozděluje jen jako dotazu odpovídající nebo neodpovídající. Model nepodporuje ohodnocování termů, tudíž ani seřazování výsledků vyhledávání podle relevantnosti[12]. Dále pro běžného, neznalého uživatele může být práce s booleovskými operátory těžká, tudíž i sestavení dotazu pomocí boolevské sémantiky náročné. Navíc při práci s velkými kolekcemi nabývá matice výskytů obrovských rozměrů a je řídká. I přes všechny tyto nedostatky je booleovský model stále velmi často používaný. Některé systémy pracující s velkým množstvím dokumentů při vyhodnocování dotazu nejdříve projdou celou matici výskytů v booleovském modelu, naleznou relevantní dokumenty a teprve až poté tyto dokumenty ohodnotí. Základ tohoto přístupu rozšiřuje populární vektorový model.

Na příkladu je ukázán postup vyhodnocování dotazů pomocí booleovské logiky. Tabulka číslo 2.2 zastupuje matici výskytů booleovského modelu, kde řádky jsou jednotlivé termy v modelu a sloupce reprezentují dokumenty. Uživatel, který bude chtít najít auto bez poruchy s benzínovým nebo dieselovým motorem, zadá dotaz ve tvaru:

$$\text{auto} \wedge ((\text{motor} \wedge \neg\text{porucha}) \wedge (\text{benzín} \vee \text{diesel})).$$

Booleovský model najde výskyty jednotlivých termů a převede dotaz na následující tvar:

$$010110 \wedge ((111111 \wedge 100110) \wedge (001100 \vee 110010))$$

a vyhodnotí jej, čímž dostane výsledek

$$000110,$$

jenž znamená, že model vrátí jako relevantní dokumenty číslo 4 a 5.

2.2.3 Vektorový model

Ve *vektorovém modelu*(vector space model) je dokument reprezentován jako vektor ve vícerozměrném prostoru[16]. Tento pojem byl poprvé definován v [22]. Model je, stejně jako booleovský, založen na přístupu bag-of-words. Jestliže m je počet různých termů ve všech dokumentech v kolekci, tak vektor, reprezentující dokument, bude mít právě m dimenzí. Na dané souřadnici bude mít hodnotu 0, jestliže se daný term v dokumentu vyskytuje, nebo jinou kladnou hodnotu, kterou určí váhovací algoritmus, jestliže dokument toto slovo neobsahuje. Všechny dokumenty v kolekci jsou ukládány do vektoru ve stejném prostoru, ve kterém termy tvoří osy, tudíž mají stejou dimenzi a lze mezi nimi provádět operace lineární algebry jako například počítat vzájemnou vzdálenost nebo velikost vektorů. Tento model umožňuje seřazování výsledků vyhledávání podle relevantnosti, což odpovídá moderním nárokům na vyhledávací systémy.

Tyto vektory bývají modelovány do term-document matice, která má rozměry MxN, kde M je počet dokumentů v kolekci a N je množství různých termů v kolekci. Řádky matice odpovídají termům a sloupce dokumentům. Na libovolné souřadnici lze pak zjistit, jakou váhou je ohodnocen term v dokumentu[20].

Jednou z výhod vektorového modelu je možnost pracovat s dotazy jako s vektory daného prostoru. Jelikož dotaz bereme jako bag-of-words, můžeme na něj hledět jako na krátký dokument a ohodnotit jej stejně jako ostatní dokumenty v prostoru. Tento přístup poté dovoluje mezi dotazem a dokumenty vypočítávat míry podobnosti, kterým se věnuje kapitola 3.

Vektorový model se nijak nezaobírá sémantikou termů, nijak nezohledňuje synonyma a neumí detekovat vícevýznamová slova.

Na vektorovém modelu je založených několik metod pro reprezentaci dokumentů, kterým se budu věnovat dále.

2.2.4 Latent Semantic Analysis

Term-dokument matice nabývá při praktickém použití velkých rozměrů a často je řídká, obsahuje velké množství nul. Lineární algebra poskytuje operace, jenž dokáží snížit dimenzi této matice.

Použití latent semantic analysis, nebo také latent semantic indexing nám umožňuje v textech najít pro počítač skryté vazby mezi termy jako například synonyma nebo různé významy stejných slov. Slovo latentní znamená skryté a sémantika je nauka o významu slov. LSA předpokládá, že významově podobná slova se budou vyskytovat ve stejné části textu. LSA používá SVD(*Singular value decomposition - singulární rozklad*). SVD je maticová

metoda z lineární algebry, která přerovnává a hodnotí dimenze ve vektorovém prostoru[27]. Při velkém počtu dokumentů nastává problém, protože vektorový model má obrovské množství dimenzí a je nutno využít nějakou z metod lineární algebry, která počet dimenzí dokáže snížit. Právě SVD dokáže zredukovat počet dimenzí při zachování prostorové podobnosti termů[16].

Nevýhodou LSA je, že když do indexu přidává nový dokument, musí se LSA reprezentace přepočítat pro celou term-document matici.

SVD rozdělí matici vektorů na následující tři matice:

$$\underset{m \times n}{A} = \underset{m \times m}{U} \underset{m \times n}{\Sigma} \underset{n \times n}{V^T}$$

Obrázek 2.1: Singulární rozklad matice A

kdy m je počet různých termů, n je počet dokumentů, U je ortogonální matice, kde sloupce definují levé singulární vektory matice A, V je také ortogonální matice a její sloupce definují pravé vektory vlastních čísel matice A. Matice Σ je diagonální maticí obsahující vlastní čísla sestupně uspořádané na diagonále[16]. SVD ještě můžeme zadat počet dimenzí, na který chceme vstupní matici zredukovat. Tento parametr k definuje počet shluků podobných si dokumentů, který se pomocí SVD vytvoří. Nyní můžeme získat redukovanou matici dokumentů i termů.

Redukovanou matici termů dostaneme ze vztahu:

$$D_k = V_k \Sigma_k \tag{2.1}$$

Redukovanou matici dokumentů získáme ze vztahu:

$$T_k = U_k \Sigma_k \tag{2.2}$$

Stejným způsobem aproximujeme dotaz, což nám umožní porovnávat mezi sebou vektory dotazu i dokumentů.

2.2.5 Latent Dirichlet Allocation

Latent Dirichlet Allocation(dále LDA) je generativní pravděpodobnostní model množiny textových dokumentů. Jeho základní myšlenkou je, že do-

kument je reprezentovaný směsicí témat a termy jsou k těmto tématům přiřazovány[13].

LDA se využívá mezi více obory, kromě zpracování přirozeného jazyka se také používá například v počítačovém vidění, kdy na obrázku dokáže rozpoznat objekty[26].

2.3 Váhování termů

Díky vektorové reprezentaci dokumentů můžeme nadále pracovat s jejich jednotlivými termy. Některá slova v textu nejsou pro význam dokumentu důležité, ovšem dají se najít takové termy, které vypovídají o obsahu textu větší mírou. Zároveň stejný term ve dvou dokumentech zabývajícími se naprosto odlišnými tématy může v každém z nich hrát naprosto jinou roli v rámci daného kontextu. Například slovo *programování* by mělo mít přiřazenou velikou váhu v dokumentu zabývajícím se historií a vývojem *programování* a menší váhu v pracovní nabídce, kde je *programování* uvedeno jako jeden z požadavků na uchazeče. Ohodnovací algoritmy se snaží ohodnotit všechny termy v dokumentu podle jejich důležitosti.

Právě díky ováhovaným termům pak počítač rozpozná, jaký dokument je více relevantní dotazu, který obsahuje slovo vyskytující se ve více textech.

2.3.1 Binární váha

Nejjednodušším způsobem je pouze ohodnocení, zda dokument daný term obsahuje, nebo neobsahuje. Tuto váhu používá booleovský model a přiřazuje 1, když se slovo v textu jednou, nebo víckrát vyskytuje, a 0 v opačném případě.

Tato technika ovšem nijak neumožňuje seřazování výsledků podle relevantnosti.

2.3.2 Frekvence termu

Frekvence termu (*term frequency - tf*)se opírá o myšlenku, že slova, která se v textu vyskytují častěji, jsou více významná. Ovšem toto tvrzení není zcela pravdivé. Nejčastěji vyskytovaná slova jsou stop slova, která už dříve vyřadil filtr a další frekventovaná slova, která se často vyskytují ve všech dokumentech, nám toho moc o významu neprozradí. Obvykle největšími nosiči významu bývají slova, která se v dokumentu objevují v menším počtu[18].

Složka *tf* vyjadřuje frekvenci výskytu termu v dokumentu. Používá se v různých interpretacích:

- První způsob nijak hodnotu neupravuje, term dostane hodnotu frekvence výskytu slova t v dokumentu d.

- Často nastává případ, že delší dokumenty jsou nadhodnocovány, neboť termy se v nich přirozeně vyskytují častěji. První normalizací, která se snaží toto eliminovat, je vydělení frekvence termu počtem všech slov v dokumentu.

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}} \tag{2.3}$$

  V rovnici číslo 2.3 f_{ik} je počet výskytů termu k v dokumentu D_i a jmenovatel zlomku značí počet všech slov v textu. Ovšem tento princip normalizuje délku jen do určité míry[14].

- Dalším druhem normalizace je dělení největším výskytem.

$$tf_{ik} = \frac{f_{ik}}{max_j f_{jk}} \tag{2.4}$$

  Tato rovnice ukazuje dělení frekvence termu největším počtem výskytů jakéhokoliv slova v dokumentu. Díky tomu nejfrekventovanější slovo v dokumentu j získá frekvenci 1 a ostatní termy budou mít podíly na intervalu (0,1)[18].

2.3.3 Inverzní frekvence dokumentu

Inverzní frekvence dokumentu (*inverse document frequency - idf*) zkoumá výskyt termu v celé kolekci prohledávaných dokumentů. Idea této míry je taková, že slovo, které se objevuje často ve všech dokumentech, nebude mít pro význam zásadní roli. Definujme df_t jako document frequency, neboli jednotku vyjadřující počet dokumentů obsahujících term t. Při N počtu dokumentů v kolekci získáváme rovnici pro výpočet inverznií frekvence dokumentů [20].

Nejčastěji se používá v následujícím tvaru.

$$idf_t = \log \frac{N}{df_t} \tag{2.5}$$

2.3.4 tf-idf

Kombinací dvou předchozích mír, frekvence termu a inverzní frekvence termu, vzniká velmi populární TF-IDF.

Dokumenty	dopolende	odpoledne	déšť	oblačno	slunce
1	0.25	0.25	0.25	0.25	0
2	0.25	0.25	0	0.25	0.25
3	0.25	0.25	0	0	0.5

Tabulka 2.3: Ukázka frekvence termu

	dopolende	odpoledne	déšť	oblačno	slunce
idf	0	0	1.0986	0.4055	0.4055

Tabulka 2.4: Ukázka inverzní frekvence dokumentu

$$tf\text{-}idf_{td} = tf_{td} \times idf_t = \frac{f_{td}}{\sum_{j=1}^{t} f_{jd}} \log \frac{N}{df_t}. \tag{2.6}$$

TF-IDF bude nabývat velkých hodnot pro termy, které se vyskytují často, ovšem v malém počtu dokumentů. Menší hodnoty bude dosahovat u těch, které se vyskytují v dokumentu několikrát nebo se nacházejí ve více dokumentech. A nejmenší váhu přiřadí algoritmus těm slovům, které se vyskytují téměř ve všech dokumentech[20].

Pro ilustraci demonstruji postup TF-IDF ohodnocování na příkladu. Mějme 3 ukázkové dokumenty s krátkými předpověďmi počasí.

1. dokument - *Dopoledne oblačno, odpoledne déšť.*
2. dokument - *Dopoledne oblačno, odpoledne slunce.*
3. dokument - *Dopoledne slunce, odpoledne slunce.*

V první tabulce číslo 2.3 je zobrazeno ohodnocení pomocí frekvence termu normalizované délkou dokumentu.

V tabulce číslo 2.4 je ukázáno přiřazení inverzní frekvence dokumentu. Pro slova *dopoledne* a *odpoledne* je tato váha nulová, jelikož se vyskytují ve všech textech, a tudíž logaritmus jedné je nulový.

V poslední tabulce číslo 2.5 je ukázána výsledná *tf-idf* váha všech termů v jednotlivých dokumentech.

2.4 Porovnávání dokumentů

Předpokladem k tomu, abychom mohli zjistit podobnost mezi dvěma dokumenty, je jejich zpracování do vektorových reprezentací. V tomto případě lze použít různé metriky na výpočet jejich podobnosti dosahujících různých úspěšností. V následující části shrneme nejpoužívanější z nich.

Dokumenty	dopolende	odpoledne	déšť	oblačno	slunce
1	0	0	0.2747	0.1037	0
2	0	0	0	0.1037	0.1037
3	0	0	0	0	0.2027

Tabulka 2.5: Ukázka tf-idf míry

2.4.1 Euklidovská vzdálenost

Jelikož dokumenty představují ve vektorovém prostoru body, a ty z nich, které by se daly klasifikovat jako podobné, se vyskytují blízko sebe, tak se zdá jako dobrá úvaha použít vzdálenost pro vyjádření jejich vzájemné podobnosti. Standardní míra, která se používá pro počítání vzdálenosti ve vektorovém prostoru, je euklidovská vzdálenost.

Je definovaná takto:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2} \tag{2.7}$$

kde x a y jsou vektory dokumentů, N je počet dimenzí prostoru a x_i a y_i jsou souřadnice.

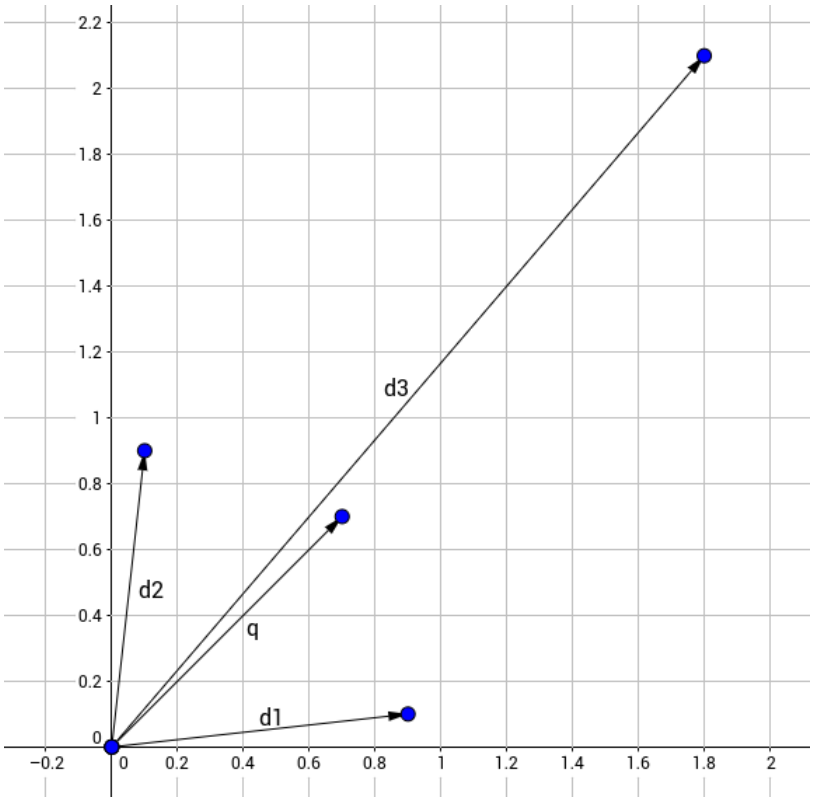
Nevýhoda použití této metriky je zobrazená na obrázku číslo 2.2. V grafu obě osy reprezentují odlišné termy a vektory *d1, d2* a *d3* jsou dokumenty. Vektor q je dotaz. Vidíme, že dokument *d1* je více relevantní se slovem, jež znázorňuje osa x, kdežto dokument *d2* se slovem osy y. Dokument *d3* leží na podobném rozmezí mezi oběma termy jako dotaz q, tudíž by měl být označen jako nejrelevantnější. Jelikož ale euklidovská vzdálenost nenormalizuje délku dokumentů, je vzdálenost mezi *d3* a q větší než mezi dotazem a ostatními dokumenty. Z tohoto pohledu se nabízí použití úhlu jako lepší řešení.

2.4.2 Kosinová podobnost

Kosinová podobnost určuje úhel mezi dvěma dokumenty, dokumentem a dotazem nebo mezi dvěma dotazy.

Kosinová podobnost mezi dokumentem d a dotazem q se vypočte následovně:

$$cos(\theta) = \frac{q \cdot d}{|q| \cdot |d|} = \frac{\sum_{i=1}^{N} q_i \cdot d_i}{\sqrt{\sum_{i=1}^{N} q_i^2}\sqrt{\sum_{i=1}^{N} d_i^2}} \tag{2.8}$$



Obrázek 2.2: Euklidovská vzdálenost

	d1	d2	d3
d1	1	0,24483	0
d2	0,24483	1	0,707106
d3	0	0,707106	1

Tabulka 2.6: Kosinová podobnost mezi dokumenty

N je dimenze vektoru.

Vydělením obou prvků rovnice jejich velikostmi vzniknou jednotkové vektory, díky čemuž nebudou nadhodnocovány delší dokumenty.

Jestliže bude vypočítaný úhel nulový a jelikož $\cos 0° = 1$, tyto dva dokumenty budou téměř totožné. V opačném případě, kdy se úhel bude blížit 90°, jsou odlišné.

V tabulce číslo 2.4.2 jsou vyjádřené kosinové podobnosti mezi dokumenty definovanými v kapitole o váhování termů.

2.4.3 Jaccardův index

Jednou z jednodušších metrik, jak vyjádřit podobnost mezi dvěma dokumenty, je Jaccardův index.

Předpokladem vyjádření Jaccardova indexu je reprezentace dokumentů jako množin, nikoliv jako vektorů. Množiny používají model bag-of-words, čili to jsou neuspořádané kolekce slov, takže platí *a,b* = *a,b*[21].

Index vyjadřuje překrytí dvou množin a má následující tvar:

$$JS(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.9}$$

2.4.4 Kullback-Leibler divergence

I když se pro tuto metriku často používá i název vzdálenost, tak se přímo o vzdálenost nejedná, jelikož na rozdíl od ostatních vzdálenostních metrik není symetrická, tudíž $D_{KL}(P||Q)$ se nerovná $D_{KL}(Q||P)$.

Ve strojovém učení se KL-divergence označuje míra, kolik nové informace obsahuje P oproti Q. Samotný vzorec KL-divergence má následující tvar:

$$D_{KL}(P||Q)) = \sum_i P(i) \log \frac{P(i)}{Q(i)}. \tag{2.10}$$

KL divergence nabývá hodnot z intervalu $\langle 0,1 \rangle$, přičemž při dosažení nulové hodnoty jsou distribuce stejné a hodnoty blízké 1 ukazují, že distribuce jsou velmi rozdílné. Jednou z podmínek vyjádření KL divergence je jestliže $Q(i)=0$, pak i $P(i)=0$ pro každé i. Druhou podmínkou je, že sumy obou distribucí se musejí rovnat jedné[2].

2.5 Vyhodnocení výsledků

Při testování algoritmů budeme chtít nějak ohodnotit, s jakou úspěšností vyhledává dokumenty. Zda nalezené dokumenty odpovídají zadanému dotazu či ne. Pro vyjádření efektivity vyhledávacího systému se používají následující veličiny.

2.5.1 Preciznost

Preciznost (precision) definuje, jak velká část z obdržených dokumentů je relevantní. Tato hodnota se pohybuje v intervalu <0,1>, přičemž hodnota 1 znamená, že všechny obdržené dokumenty jsou relevantní, ovšem neříká

	revelantní	**nerelevantní**
obdržené	true positives (tp)	false positives (fp)
neobdržené	false negatives (fn)	true negatives (tn)

Tabulka 2.7: Vyhodnocování výsledků

nám nic o tom, zda byly systémem vráceny všechny relevantní dokumenty z kolekce[20].

Schéma pro výpočet přesnosti je:

$$preciznost = \frac{počet\ obdržených\ relevantních\ dokumentů}{počet\ obdržených\ dokumentů} \tag{2.11}$$

2.5.2 Úplnost

Úplnost (recall) určuje, jaké procento z relevantních dokumentů v kolekci dokumentů systém určil jako relevantní. Opět může nabývat hodnot v intervalu <0,1>, přičemž hodnota 1 určuje, že všechny relevantní dokumenty v kolekci byly systémem vráceny, ovšem nevíme, kolik nerelevantních dokumentů bylo vráceno s nimi[20].

Schéma pro výpočet úplnost je:

$$úplnost = \frac{počet\ obdržených\ relevantních\ dokumentů}{počet\ relevantních\ dokumentů} \tag{2.12}$$

Obě tyto hodnoty se zvlášť moc nepoužívají, navíc si velmi často odporují. Většinou mezi nimi platí nepřímá úměra, čím vyšší přesnosti chceme dosáhnout, tím nižší bude výsledný recall a obráceně.

2.5.3 Přesnost

Všechny výsledky nalezené vyhledávacím systémem lze podle úspěšnosti rozdělit do 4 skupin, které definuje tabulka číslo 2.5.3.

Podle této tabulky lze vyjádřit precision a úplnost dalším tvarem:

$$Preciznost = \frac{tp}{tp + fp} \tag{2.13}$$

$$Úplnost = \frac{tp}{tp + fn} \tag{2.14}$$

Další možností, jak vyhodnotit úspěšnost klasifikátoru je přesnost (accuracy). Ta vyjadřuje součet dokumentů, které klasifikátor označil správně vydělený počtem všech dokumentů. Má následující tvar:

$$P\check{r}esnost = \frac{(tp + tn)}{tp + fp + fn + tn} \tag{2.15}$$

Tato míra je užitečná, jelikož opravdová práce klasifikátoru je rozdělování dokumentů do dvou tříd, relevantní a nerelevantní. Ovšem moc se nepoužívá, jelikož v praxi je velkých kolekcích 99% dokumentů v nerelevantní kategorii a systém nastavený na maximalizaci accuracy může jednoduše označit všechny dokumenty jako nerelevantní a tím dosáhnout velké přesnosti. Ale uživatel vždy požaduje vrácení nějakých dokumentů i za cenu, že se v nich bude vyskytovat malé množství *false positives*[20].

2.5.4 F-míra

F-míra vytváří harmonický průměr mezi precizností a úplností a tím zároveň i rozumný kompromis mezi těmito dvěma hodnotami. F-míra se počítá podle následujícího vztahu:

$$F = \frac{(1 + \beta^2) \cdot preciznost \cdot úplnost}{(\beta^2 \cdot preciznost) + úplnost} \tag{2.16}$$

kde β je parametr v intervalu (0,∞) a ukazuje, které z hodnot bude přidělena větší váha. Při $\beta = 1$ mají preciznost i úplnost stejnou váhu. Hodnoty $\beta < 1$ zdůrazňují preciznost a hodnoty $\beta > 1$ zdůrazňují váhu úplnosti[20].

3 Detekce téměř duplicitních dokumentů

Problémem rozsáhlých kolekcí je, že obvykle obsahují velké množství duplicitních a téměř duplicitních dokumentů. Jejich udržování v kolekci zvětšuje velikost indexu a zpomaluje vyhledávání. Vyhledávací systém, který nedokáže tyto dokumenty rozpoznat, také obvykle není dobře hodnocen, jelikož uživatel nechce, aby vyhledávač vrátil na prvních dvaceti místech buď totožné dokumenty, nebo takové, které se liší jen v několika slovech. Tento rozdíl bývá často jen v časovém údaji, v souborovém formátu nebo například ve webových dokumentech v zobrazení reklamy. Pro vyhledávač je nezbytné tyto dokumenty najít, uchovat v indexu jen v jedné verzi a zbylé odstranit z indexu. Případně je možné při přidávání nového dokumentu do indexu zkontrolovat, zda to není duplikát nebo skoro duplikát k již zaindexovaném dokumentu.

Metodou na detekci téměř duplicitních dokumentů se v této práci zabývám z toho důvodu, že velké množství revizí vzniká jen malými úpravami daného dokumentu, tudíž je vzniklá revize téměř duplicitou upravovaného dokumentu.

Jednoduchým přístupem, jak tyto dokumenty najít, je spočítat kosinovou podobnost mezi všemi dvojicemi v kolekci, a jako téměř duplicitní detekovat takové, které přesáhnou nějakou předem definovanou hodnotu. Použití tohoto způsobu je ve velkých kolekcích neefektivní, jelikož takových dvojic je obrovské množství a zároveň zjištění kosinové podobnosti velkých vektorů je drahá operace.

3.1 Shingling

Model bag-of-words neuchovává o slovech žádnou poziční informaci. Zanedbává sousední termy, tudíž nevíme, v jakém kontextu bylo slovo použito. Při použití tohoto modelu může nastat situace, že dva dokumenty, jež mají téměř totožnou množinu slov, jsou obsahově naprosto odlišné.

Nejefektivnější cestou, jak reprezentovat dokumenty jako množiny za účelem detekce podobných dokumentů, je vytvoření z dokumentu k-gramy, neboli k-shingly. Dokumenty, které obsahují stejné věty nebo fráze, budou sdílet i velké množství těchto k-shinglů[18].

K-shingle je množina k po sobě jdoucích slov. Tudíž použití shinglů velikosti 1 má stejný efekt jako použití modelu bag-of-words[21].

Dokument 1: *Honza si včera koupil nové auto.*

Množina 2-shinglů z dokumentu 1 má tvar:{(Honza si), (si včera), (včera koupil), (koupil nové), (nové auto)}.

Množina 3-shinglů z dokumentu 1 má tvar:{(Honza si včera), (si včera koupil), (včera koupil nové), (koupil nové auto)}.

Pro vyjádření míry podobnosti množin k-shinglů se používá Jaccardův index. Na dokumenty s ohodnocenými shingly lze ovšem aplikovat i kosinovou podobnost.

Důležitou kategorií při vytváření k-shinglů z dokumentu je zvolení jeho velikosti. Při zvolení malého k se může stát, že dvojice dokumentů bude mít vysoký Jaccardův index, jelikož oba obsahují velké množství stejných k-shinglů, ovšem nemusí mít stejné věty nebo fráze. Správná volba k závisí na velikosti dokumentů v kolekci. Nejčastěji používanou hodnotou k je 5[18].

Nevýhodou generování k-shinglů z dokumentu jsou velké nároky na úložiště. Když používáme shingly o k slovech, tak každé slovo z dokumentu se vyskytuje právě v k-shinglech, což znamená, že musíme každé slovo uchovávat několikrát[17].

Spočítání Jaccardova indexu, obzvláště mezi velkými množinami, je často výpočetně velmi náročné.

4 Rozpoznávání pojmenovaných entit

Pojmenované entity jsou slova nebo uskupení slov, jež vyjadřují například jméno osoby, lokalitu, název společnosti, datum a mnoho dalších. Jedna z úloh počítačového zpracování jazyka je jejich detekce a klasifikace do zmiňovaných tříd. Rozpoznávání pojmenovaných entit se používá například v strojovém překladu, zpracování nestrukturovaných dokumentů a následovné vyhledávání v nich. Když dokážeme v nabídce cestovní kanceláře dobře detekovat stát a město, o kterém tato nabídka pojednává, usnadníme potom uživateli vyhledávání jen těch dovolených, které se odehrávají v daném státě a městě.

Pro rozpoznávání se používá několik metod například porovnávání s regulárními výrazy, vyhledávání v seznamu pojmenovaných entit (seznam křestních jmen, příjmení atd.), ovšem nejúspěšnější metodou se ukázalo strojové učení, kdy se počítač učí z velkého množství ručně anotovaných trénovacích dat. Úspěšnost klasifikátoru se nejčastěji hodnotí F-mírou[25].

Existuje několik nástrojů pro klasifikaci pojmenovaných entit. Všechny většinou pro rozpoznávání používají jazykový model natrénovaný na nějaké množině anotovaných dat.

4.1 Nástroje pro anglický jazyk

Asi nejobsáhlejším takovýmto nástrojem je *General Architecture for Text Engineering* neboli *GATE*, který vznikl již v roce 1995, a má vytvořené modely pro největší počet jazyků. Dalším populárním nástrojem je *Stanford Named Entity Recognition*, což je projekt napsaný v javě, jenž byl poprvé zveřejněn v roce 2006. Kromě Javy má implementace i v mnoha jiných programovacích jazycích. Nástroj obsahuje 3 modely pro anglický jazyk, které dokáží klasifikovat pojmenované entity do 3, 4 nebo 7 tříd. V modelu pro 7 tříd rozpoznává místo, jméno osoby, organizaci, měnu, datum, čas a číslo vyjádřené v procentech. Kromě anglického modelu má vytvořené modely pro němčinu, španělštinu a čínštinu.

4.2 Nástroj pro český jazyk

Nástroj s českým jazykovým modelem vznikl v roce 2014 na Matematicko-fyzikální fakultě Karlovy Univerzity v Praze. Je to open-source projekt, jmenuje se *NameTag* a jeho tvůrci jsou Milan a Jana Strakovi. Je distribuován jako samostatná aplikace nebo jako knihovna s natrénovanými jazykovými modely. Projekt je psaný v C++, ale obsahuje rozhraní i v ostatních programovacích jazycích.

Projekt obsahuje dva české jazykové modely. Prvním z nich je model označovaný jako *Czech Named Entity Corpus 1.1 Model*. Tento model byl trénován na *Czech Named Entity Corpus 1.1*, což je množina 5868 českých vět s 33662 ručně anotovanými pojmenovanými entitami klasifikovanými do dvoustupňové hierarchie 62 tříd zobrazené v příloze C na obrázku číslo C.1.

Nejnovější verze 1.1 modelu dosáhla na *Czech Named Entity Corpus 1.1* při dvoustupňové klasifikaci úspěšnosti 75.47% vyjádřené F-mírou a 78.51% při klasifikaci jen do tříd nejvyššího stupně.

Druhým modelem je *Czech Named Entity Corpus 2.0 Model*, který byl natrénován na *Czech Named Entity Corpus 2.0*, jenž obsahuje 8993 vět s ručně označenými 35220 pojmenovanými entitami roztříděnými do dvouúrovňové hierarchie 46 tříd zobrazené v příloze C na obrázku číslo C.2.

Aktuální verze 2.0 modelu klasifikovala do tříd obou úrovní entity z *Czech Named Entity Corpus 2.0* s úspěšností vyjádřenou F-mírou 75.38% a do tříd nejvyššího stupně s úspěšností 79.16%[24].

5 Apache Lucene

Apache Lucene je open-source vyhledávací software, jehož největším projektem je knihovna kompletně napsaná v Javě poskytující indexační a full-textové vyhledávací nástroje. Tato knihovna se jmenuje Lucene Core. První release knihovny byl zveřejněn v roce 1999 a napsal jej Doug Cutting. I když je napsaná v Javě, tak má rozhraní i v ostatních jazycích, například v C, C++, .NET, Perl nebo Python. Lucene Core knihovna je velmi populární, využívají ji portály Twitter, LinkedIn, Wikipedie, vyhledávač Elasticsearch nebo například vysokovýkonný vyhledávací server Solr. Jejími hlavními přednostmi jsou rychlé, optimalizovatelné indexování, nízké požadavky na paměť RAM a malá velikost výsledného indexu. Poskytuje seřazení nalezených výsledků podle relevantnosti (ohodnocování), podporuje Vector Space Model a není závislá na platformě[3].

Lucene Core používá invertovaný index. Základní stavební jednotkou Lucene indexu je dokument, jenž se skládá z jednoho, nebo více polí. Příkladem pole je třeba obsah souboru, jeho autor, datum vytvoření nebo různá další metadata. Lze u nich specifikovat různé parametry, například zda se má text v indexu uchovávat v původní formě, zda má být indexován nebo tokenizován.

Při indexaci se definuje analyzér, který popisuje použitý tokenizér, slovník stop-slov, použití stemmingu a další parametry, které určují, jak zpracovat vstupní text před začátkem samotné indexace. Lucene má již vytvořené standardní analyzéry pro většinu světových jazyků, které poskytují standardní tokenizéry, normalizaci a obsahují slovník stop-slov daného jazyku[1].

5.1 Podobnost v Lucene

Srdcem knihovny jsou její vyhledávací schopnosti. Při zadání dotazu Lucene najde všechny dokumenty, které obsahují termy specifikované v dotazu a až poté jsou z těchto dokumentů vytvořené vektorové modely a ohodnotí se pomocí TF-IDF. Lucene potom podobnost vypočítává modifikovanou verzí kosinové podobnosti, jenž se nazývá Lucene scoring formula.

5.1.1 Lucene scoring formula

Pro vypočítání podobnosti mezi dotazem q a dokumentem d má tato porovnávací funkce následující podobu:

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t\ in\ q} (tf(t\ in\ d) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d)) \quad (5.1)$$

kde

- **tf(t in d)** definuje frekvenci termu t v dokumentu d. U **tf(t in q)** vyjadřující frekvenci termu t v dotazu q se předpokládá hodnota 1.
- **idf(t)** ukazuje, v kolika dokumentech se vyskytuje term t.
- **coord(q,d)** zjišťuje, kolik termů z dotazu obsahuje zkoumaný dokument. Dokumenty, které obsahují více výrazů, z dotazu obdrží větší skóre.
- **queryNorm(q)** normalizuje vypočtené hodnoty podobnosti, abychom mohli mezi sebou porovnávat získané hodnoty od různých dotazů.

$$queryNorm(q) = \frac{1}{sumOfSquaredWeights^{1/4}} \quad (5.2)$$

$$sumOfSquaredWeights = q.getBoost()^2 \cdot \sum_{t\ in\ q} (idf(t) \cdot t.getBoost())^2 \quad (5.3)$$

- **tBoost** je zesílení významu termu t v dotazu q.
- **norm(t,d)** se skládá z několika složek:
  - **Field boost** - můžeme zesílit význam nějakého pole
  - **lengthNorm** - vypočítává se, když je dokument přidáván do indexu a způsobuje to, že kratší pole budou více přispívat do skóre[10].

$$norm(t,d) = lengthNorm \cdot \prod_{pole\ f\ v\ d\ pojmenované\ t} f.boost() \quad (5.4)$$

5.2 More like this

More like this je název algoritmu implementovaném v Lucene pro nalezení k danému dokumentu jemu podobné. More like this najde ve vyhledáváném dokumentu k počet termů s nejvyšší TF-IDF, tyto termy spojí operátorem OR a vytvoří z nich disjunktní dotaz. Maximální počet termů, které se budou z vyhledávaného dokumentu extrahovat lze nastavit, přičemž výchozí hodnota je 25. Tento dotaz se poté předá do indexového vyhledávače, který najde definovaný počet relevantních dokumentů. Tento vyhledávač pro porovnávání defaultně používá Lucene Scoring Formula[4].

5.3 Apache Tika

Tika je nástroj psaný v Javě, který detekuje a extrahuje metadata a text z více než tisíce různých souborových formátů. Tika je distribuována jako javovská knihovna, ukázková aplikace a ve formě serveru.

Klíčovou částí knihovny je rozhraní *Parser*. Implementuje ho velké množství tříd, kde každá má již v názvu definováno, z jakého typu souboru extrahuje text. Všechny tyto třídy implementují metodu *parse*, která ze vstupního proudu detekuje a extrahuje text a metadata[11].

Seznam všech podporovaných formátů se nachází na adrese[1].

[1] https://tika.apache.org/1.15/formats.html

6 Revize

Pro úspěšné klasifikování, zda je daný dokument revizí nějakého jiného, je důležité si nejdříve přesně definovat pojem revize.

6.1 Definice

Slovo revize se nejčastěji používá ve významu zkoumání nebo zjišťování správnosti něčeho[9], jako například revize jízdenek. V této práci se ovšem zabývám odlišným významem tohoto slova, jenž znázorňují definice z několika českých výkladových slovníků:

- **Slovník spisovného jazyka českého** - *přezkoumávání něčeho za účelem úpravy, změny, zlepšení ap.: r. zahraniční politiky; r. hranic; r. trestního řízení; r. trestu; jazyková r.; polygr. oprava technických chyb podle poslední korektury*[9].
- **Slovník současné češtiny LINGEA** - *přezkoumá(vá)ní za účelem změny, úpravy ap., revize hranic/stanov/zahraniční politiky, jazyková revize, Přišla revize, dopravní revize*[7].
- **ABZ.cz - slovník cizích slov** - *oprava, úprava, poupravení, editace, nová (revidovaná) verze něčeho, přehodnocení, přezkoumání, opětovné uvážení a následná změna něčeho (např. zákona)*[5].

Dále přiložím pár definic z anglických slovníků:

- **Macmillan dictionary** - *the process of changing, improving, or making additions to something such as a plan, law, or piece of writing*[8].
- **Cambridge dictionary** - *a change that is made to something, or the process of doing this*[6].

Všechny uvedené definice popisují pojem revize jako změnu nebo úpravu nějakého textu. To je velmi obecné označení, podle něhož bychom mohli nazvat revizemi obrovské množství různě upravených dokumentů. My si pro potřeby této práce musíme definovat tento pojem přesněji.

Dokument A je revizí dokumentu B, jestliže dokument A vznikl úpravou dokumentu B, která přinesla nějakou novou informaci nebo opravu, přičemž smysl dokumentu zůstal nezměněn, a dokument B tak obsahuje chybné nebo neaktuální informace.

Při znalosti této definice již víme konkrétněji co klasifikovat jako revizi a co ne. I když posouzení, zda se jedná o revizi, je stále velmi subjektivní. Existují případy, kdy zcela není pochyb o tom, jestli je jeden dokument revizí druhého nebo ne, ovšem dají se najít i případy, kdy je těžké nebo nemožné v daných podmínkách určit, zda je dokument revizí.

6.2 Příklady revizí

Příkladem revize může být již zmíněná novela zákona, kdy se mění nebo doplňuje jiný právní předpis. Tato změna není ovšem zásadní. Zákon pojednává stále o stejných věcech, ovšem v novele došlo například k opravení chyb, přizpůsobení na nové podmínky nebo k aktualizaci dat. V okamžiku vzniku novely je upravovaný zákon neaktuální, nemůže být nikde zveřejněn a nesmí podle něj nikdo jednat.

Další příklad jasné revize je dokument s volnými tématy bakalářských prací pro studenty v daném roce. Paní sekretářka postupně dostává od pracovníků školy vypisovaná témata a přidává je do dokumentu. Při každém přidání nových témat vznikne revize, jelikož jakákoliv předchozí varianta dokumentu je chybná. Ta samá situace nastane při mazání z volných témat, protože studenti si je budou postupně vybírat. Dostupný musí být jen ten dokument, který obsahuje pouze volná témata, aby si studenti nemohli vybírat již ze zabraných.

Tady je na místě vysvětlit rozdíl mezi verzí a revizí dokumentu. Nová verze dokumentu může vzniknout podobným způsobem jako revize, tudíž nějakou úpravou nebo přidáním části textu, ovšem v případě verze se upravuje tak, že nová verze obsahuje nějakou jinou informaci a zároveň původní dokument, ze kterého verze vychází, je stále platný a může se nějakým způsobem hodit. Dalším rozdílem je fakt, že při přečtení předchozí verze se čtenář narozdíl od zrevidovaného dokumentu, nedozví chybné, neplatné informace.

Případ, kdy by dokument neměl být klasifikován jako revize, může nastat například v nějaké advokátní kanceláři, která vytváří smlouvy na nákup zboží. Zákazníkům vytváří jednotlivé smlouvy podle nějaké šablony, tudíž dvě vytvořené nákupní smlouvy pro dva různé zákazníky budou téměř totožné, řekněme, že se můžou lišit v několika parametrech, nebo v pár odstavcích. V případě těchto dvou smluv se o revize nejedná, protože obě obsahují aktuální, a žádná z nich neobsahuje chybné informace, a oba jsou samostatně vystupující dokumenty. Pokud by došlo k úpravě a vzniku nového dokumentu v rámci jedné smlouvy, potom by se o revizi dané smlouvy jednalo.

Jako revize by neměl být klasifikován například sylabus univerzitního

předmětu pro různé školní roky. Je dost pravděpodobné, že když nedojde k žádným razantním změnám, tak budou sylaby stejných předmětů pro dva školní roky téměř totožné, ovšem o revize by se jednat nemělo z toho důvodu, že i když uvážíme například rozdíl jen v datech odevzdání semestrální práce, zápočtového testu a zkoušky, tak v obou dokumentech jsou minimálně tato data aktuální a zároveň se nedají klasifikovat jako zastaralá, a je třeba oba tyto dokumenty v kolekci zachovat, jelikož můžeme kdykoliv chtít zjistit, jaké byly termíny zkoušek v nějakém školním roce.

Příkladem dokumentů, u kterých je sporné určit, zda jsou revizemi, mohou být například dvě pozvánky na vědecké konference. Změnou data a místa konání konference spolu i s úpravou programu může vzniknout revize v situaci, ve které došlo ke změně místa pořádání. V opačném případě se jednat o revizi nemusí, protože pisatel při psaní pozvánky na další termín konference vycházel z tohoto dokumentu, čímž vznikla pozvánka podobná té pro minulý termín. V případě vytvoření nové pozvánky revize nevzniká.

Jako druhý příklad nejasného rozpoznání revizí uvedu dva totožné dokumenty, například nějaká prohlášení vedení firmy, která se liší jen v podpisu. Tyto odlišné podpisy mohou znamenat to, že byla v dokumentu podepsaná nesprávná osoba, tudíž došlo k opravě a tím ke vzniku revize, a nebo ve vedení firmy nastaly změny a dokument podepíše nový ředitel, čímž předchozí dokument ztrácí platnost, ovšem neobsahuje chybné informace a revize nevznikla.

V neposlední řadě bude pro člověka těžké rozpoznat, zda je dokument revize v případě, kdy proběhnou v dokumentu zásadní změny takové, že bude těžké posoudit, zda upravený dokument stále pojednává o stejné věci jako ten, ze kterého vychází. Úplně typickým příkladem vzniku revizí je situace, při které spisovatel začne psát úvod knihy. Psaní několikrát přeruší a při každém návratu k práci vždy připíše, upraví nebo odstraní část úvodu, čímž pokaždé vznikne revize. V případě, kdy by spisovatel přehodnotil svůj názor a celý úvod přepsal, by se samozřejmě o revizi také jednalo, ale bez znalosti toho, že došlo k tomuto přepisu, by čtenář při pohledu na dva odlišné úvody knihy nejspíše určil, že se v daném případě o revize nejedná.

7 Testovací množina dokumentů

Pro účely testování úspěšnosti stávajících algoritmů a vlastního algoritmu bylo nutné vytvořit množinu testovacích dokumentů. Od vedoucího práce jsem dostal kolekci rozličných dokumentů týkajících se převážně univerzitního života. Z nich jsem některé vybíral a vytvářel k nim revize. Snažil jsem se do této vlastní množiny testovacích dokumentů zahrnout všechny možné kombinace dokumentů s revizemi a také textů, jež žádné revize mít nemají. Vznikla množina 100 dokumentů, kdy ke každému z nich byl vytvořen nějaký počet revizí, a toto číslo se nejčastěji pohybuje od jedné do sedmi.

Pro určení toho, s jakou úspěšností algoritmus nalezl revize, bylo třeba všechny testovací dokumenty a k nim vytvořené revize nějakým způsobem označit. Platí, že dokumenty, které mají nějaké revize, obsahují na začátku jména dva identifikátory oddělené podtržítkem. První určuje ID revize a druhé jeho pořadové číslo. Každý dokument s nějakými revizemi má v metadatech položku s názvem *pocetRevizi*, kterou je nezbytné u každého dokumentu vědět pro změření úspěšnosti klasifikátoru. Při vyhodnocování výsledků vyhledávání jsou pak u nalezených dokumentů parsovány podle podtržítka jejich jména a u nalezeného dokumentu se porovnává, zda jeho ID souhlasí s ID vyhledávaného dokumentu. V případě shody se jedná o úspěšně detekovanou revizi.

Celá testovací kolekce se skládá z 996 dokumentů.

8 Implementace stávajících algoritmů

Jedním z bodů zadání práce je i vyzkoušení toho, jak obstojí stávající přístupy pro hledání podobných dokumentů při rozpoznávání revizí.

8.1 Analýza výběru algoritmů

Z důvodu, že ve velké většině případů vzniká revize jen menšími úpravami v dokumentu, není zjišťování jakékoliv sémantické informace z textu relevantní tématu. Z principů popsaných v teoretické části otestuji nejrozšířenější způsob vyjádření podobnosti dvou dokumentů v kolekci, čímž je kosinová vzálenost mezi vektory ohodnocenými TF-IDF. Jako druhý jme si pro implementaci vybrali k-shingling, protože to je velmi rozšířený a osvědčený způsob pro detekci duplikátů a téměř totožných dokumentů, a jako poslední z používaných algoritmů vyzkoušíme More like this.

8.2 Použité technologie

Pro implementaci všech stávajících algoritmů jsem použil programovací jazyk Java a knihovnu Lucene. Tuto knihovnu jsem si vybral z toho důvodu, že umožňuje vytvořit index a pracovat s ním, a zejména i proto, že obsahuje nástroje pro získání vektorů z dokumentů.

8.3 Vytvoření indexu

Prvním krokem pro získání vektorů z dokumentů v testovací množině je celou kolekci zaindexovat. Testovací dokumenty nejsou jen obyčejné textové soubory. Jsou to soubory různých textových formátů, nejčastěji s příponou *doc* nebo *docx*, tudíž je nutné z nich v první řadě pomocí nástroje Tika extrahovat text a metadata.

Klíčovou komponentou pro parsování dokumentu pomocí nástroje Tika je rozhraní *Parser* z balíku *org.apache.tika.parser*. Toto rozhraní iplementuje velké množství různých parserů pro rozličné druhy souborových formátů. Nejobecnějším parserem je *AutoDetectParser*, jelikož u ostatních parserů je

třeba vědět předem, jaký formát textového souboru se bude parsovat. U *AutoDetectParseru* tato povinnost odpadá, protože dokáže sám rozpoznat, o jaký formát se jedná. Zásadní metodou rozhraní *Parser* je *parse()*, jež má 4 parametry, kterými jsou *InputStream* souboru, *ContentHandler*, *Metadata* a *ParseContext*. Do *ContentHandleru* se naparsuje rozpoznaný text, *Metadata* je objekt, který udržuje metadata dokumentu a *ParseContext* umožňuje přizpůsobení parsovacího postupu.

Tika získá ze souboru text a metadata a ta již mohou být zaindexována. Indexace v Lucene probíhá takovým způsobem, že je prvně vytvořen dokument, který se skládá z polí, a ten je pomocí *IndexWriter* zapsaný do indexu. První pole, které má každý indexovaný dokument, definuje název dokumentu. Je to instance třídy *StringField*, která vložený text netokenizuje, ale udržuje jej v celém stavu, což je u názvu dokumentu žádoucí. Text dokumentu se předává do pole *text*, které je instancí třídy *Field*, jež je rodičem třídy *StringField*. Tuto obecnější třídu jsem použil z toho důvodu, že potřebuji přesněji definovat, s jakými vlastnostmi se bude pole do indexu zapisovat. Tyto vlastnosti se předávají do pole pomocí třídy *FieldType* a jsou zobrazeny na ukázce č. D.1.

Z takto specifikovaného pole lze poté při procházení indexu získat vektor termů, u kterých bude uchovávaná jejich frekvence v daném dokumentu spolu s inverzní frekvence dokumentu.

V případě, že *Parser* nalezl v souboru nějaká metadata, jsou uložena do jednotlivých polí pojmenovaných názvem metadat a vyplněna hodnotou.

Jak již bylo uvedeno, o samotné zapsání dokumentu do indexu a následné vytvoření indexu v paměti počítače se stará třída *IndexWriter*. *IndexWriter* potřebuje znát 2 parametry, jimiž jsou adresář, kde se vytvoří index, a *Analyzer*, který bude analyzovat text v polích.

8.4 TF-IDF a kosinová vzdálenost

Jak jsem již uvedl, v této práci je nerelevantní snažit se zjistit jakoukoliv sémantickou informaci z textu, tudíž jsem při indexaci použil *CzechAnalyzer* bez jakékoliv normalizace. Defaultní *CzechAnalyzer* implementovaný v Lucene pouze tokenizuje text a používá český slovník stop-slov, tudíž se hodí pro moje použití.

Pro zjištění kosinové podobnosti mezi dvěma zaindexovanými dokumenty je třeba z obou nejdříve získat vektory termů a ohodnotit je. Pak lze z těchto vektorů pomocí vzorce na výpočet kosinové vzdálenosti vyjádřit číslo určující podobnost těchto dvou dokumentů. V kosinové podobnosti se ve jmenovateli

používají normalizace obou zúčastněných vektorů. Tyto normalizace jsou vypočítány na začátku společně s ohodnocováním dokumentů, protože jsou pro jednotlivé dokumenty vždy stejné a bylo by nepraktické je počítat při každém porovnávání znovu. Z toho důvodu, že v čitateli vzorce je součin, tak nulový prvek v jednom z vektoru znamená nulu přidanou do sumy kosinové podobnosti. Nenulovou hodnotu získají pouze slova, která se vyskytují v obou vektorech, tudíž není třeba přepočítávat všechny vektory tak, aby měly stejnou dimenzi, ale stačí pouze testovat, zda se dané slovo nachází v obou vektorech.

Na ukázce číslo D.2 je demonstrován postup pro získání ováhovaného vektoru ze zaindexovaného dokumentu.

8.5 K-shingling

Při aplikaci shinglingu není text tokenizován na jednotlivá slova, ale na k-tice. Lucene neobsahuje samotný analyzér, který by generoval shingly. Má implementovaný *ShingleFilter*, který z příchozího proudu tokenů vytváří shingly definované délky. Je tedy třeba vytvořit si vlastní *Analyzer*, který bude pracovat s tímto filtrem. Jen poznamenám, že Lucene má *ShingleAnalyzerWrapper*, kterým se dá obalit jiný *Analyzer*, ovšem toto spojení, například při obalení *StandardAnalyzeru*, generuje jak jednotlivé tokeny, tak i shingly.

V ukázce číslo D.3 je vytvoření vlastního analyzéru pro tvorbu shinglů. *StandardFilter* jen předává tokeny přicházející z *StandardTokenizéru* do *ShingleFilteru*, který je poté skládá do k-tic. Velikost shinglů je definována posledními dvěma parametry, kdy prvním z nich je minimální velikost a druhým maximální. Výchozí hodnota je 2. Při zadání rozdílných hodnot, například minimální 2 a maximální 4, bude text rozsekán postupně na shingly velikosti 2, 3 a 4 a v indexu budou uchovány všechny.

K měření podobnosti množin shinglů se nejčastěji používá Jaccardův index, ovšem já otestuji i vážení shinglů s TF-IDF a poté vyjádření podobnosti kosinovou vzdáleností a porovnám výsledky všech experimentů. I když se nedá očekávat, že shingly větší velikosti, například 5, se těžko budou v daném dokumentu nebo v celé kolekci vyskytovat opakovaně.

8.6 More like this

More like this je jediný způsob v Lucene, jak porovnat dva dokumenty mezi sebou z hlediska podobnosti. Hlavním stavebním kamenem Lucene je vyhle-

dávání dokumentů relevatních nějakému dotazu.

Algoritmus reprezentuje třída *MoreLikeThis*. Její instanci lze definovat použitý *Analyzer* i pole dokumentů, která se budou porovnávat. Metoda *like()* s číslem dokumentu v indexu jako parametrem vygeneruje způsobem popsaným v teoretické části dotaz, ke kterému *IndexSearcher* vyhledá definovaný počet relevantních dokumentů.

8.7 Průběh experimentů

Možností, jak určovat revize pomocí zmíněných algoritmů, je detekovat jako revizi takový dokument, který při porovnávání dosáhl hodnoty přesahující nějaký daný práh. Cílem experimentů bylo zjistit pro každý stávající algoritmus ideální práh, tudíž ten, se kterým vyhledávání dosáhlo nejlepších výsledků a poté tento práh otestovat na množině testovacích dokumentů.

Při testování byla použita křížová (cross) validace a to tak, že množina 100 dokumentů, pro něž budou vyhledávány revize, byla rozdělena na 80 trénovacích a 20 testovacích. Na trénovacích dokumentech se postupně zkoušely různé práhy a měřila se jejich úspěšnost až nakonec byl vybrán práh s nejvyšší úspěšností a ten byl použit pro testování na testovacích dokumentech. Po uplynutí jednoho takového kroku bylo těchto 100 dokumentů v jiném pořadí opět rozděleno na 80 trénovacích a 20 testovacích, přičemž kroků bylo celkem provedeno 5. Celkovým výsledkem experimentu je průměr ze zjištěných 5 optimálních prahů a průměr pěti dosažených výsledků vyjádřených F-mírou. Před začátkem každého experimentu je všech 100 dokumentů náhodně promícháno.

U samotného vyhodnocování výsledků jsem musel ošetřit několik případů. Jedním z nich je ten, kdy jsem zkoušel vyhledávat revize pro dokumenty, které žádné nemají. Když algoritmus v tomto případě žádné revize nenalezne, tak se F-míra rovná jedné. A dále jsem musel zajistit, aby nedocházelo k dělení nulou, takže když se počet relevantních obdržených nebo počet obdržených dokumentů rovnal nule, v tomto případě se F-míra vyhodnotila jako nulová.

Jelikož vyjádření podobnosti dvou zvláště větších dokumentů je výpočetně poměrně náročné a s použitím cross validace na docela velké množině testovacích dokumentů trvají všechny experimenty dlouhou dobu. Kvůli urychlení experimentů jsem nejdříve v indexu vytvořil term-dokument matici booleanovského modelu a do zvláštního indexu uložil pro jednotlivé dokumenty ID dokumentů, které obsahují nějaké společné termy. Tímto se zamezí počítání podobnosti mezi dokumenty, které nemají žádná stejná slova.

	Kosinová vzdálenost a TF-IDF		Jaccardův index	
Použitá metoda	optimální práh	výsledná F-míra	optimální práh	výsledná F-míra
jednotlivé termy	0.78	0.784	X	X
2-shingling	0.7	0.838	0.5	0.783
3-shingling	0.7	0.859	0.5	0.832
4-shingling	0.7	0.859	0.5	0.842
5-shingling	0.7	0.878	0.5	0.846

Tabulka 8.1: Výsledky experimentů s kosinovou vzdáleností a Jaccardovým indexem

More like this	**optimální práh**	**výsledná F-míra**
jednotlivé termy	64.0	0.560
2-shingly	38.0	0.508
3-shingly	12.8	0.414
4-shingly	7.0	0.330
5-shingly	5.4	0.229

Tabulka 8.2: Výsledky experimentů s algoritmem More like this

U indexů se shingly jsem použil postup, že bude docházet k porovnávání pouze mezi dokumenty, které mají alespoň jeden stejný shingle. U jednotlivých termů tento princip nebyl úspěšný, protože velké množství dokumentů v kolekci sdílejí alespoň jedno slovo. Z provedených testů vyplynulo, že ke stejným výsledkům jako při porovnávání všech dokumentů se došlo při porovnávání jen těch dokumentů, které obsahují alespoň 20 stejných termů. Tímto postupem se mi podařilo rapidně snížit časy trvání experimentů, ovšem samotné zjištění informace, s jakými dokumenty porovnávat je výpočetně náročné, tudíž jsou vytvořené indexy společně se popsanými seznamy přiloženy k aplikaci. Toto řešení, čili uložení seznamů dokumentů k porovnání, jsem zvolil z toho důvodu, že všechny indexy se používají v několika experimentech, tudíž je dobré, mít tyto indexy vytvořené na začátku.

8.8 Výsledek experimentů

U přístupů využívajících k vyjádření podobnosti kosinovou vzdálenost nebo Jaccardův index bylo nalezení optimálního prahu jednoduché, jelikož výsledná hodnota je vždy z intervalu $\langle 0,1 \rangle$, kdežto More like this takovouto vlastností nedisponuje. Celkové výsledky experimentů s kosinovou vzdáleností a Jaccardovým indexem jsou zobrazeny v tabulce č. 8.1.

Výsledky experimentů s algoritmem More like this jsou zobrazeny v tabulce č. 8.2.

Z tabulky číslo 8.1 je patrné, že vysokých výsledků dosáhly obě porovnávací metody při porovnávání shinglů větších velikostí. Ukázalo se, že k-shingling je kvalitním algoritmem pro hledání velmi podobných dokumentů. Druhým poznatkem je, že ohodnocení TF-IDF a porovnávání pomocí kosinové vzdálenosti má smysl i pro použití k-shinglingu, jelikož takto získané výsledky jsou vyšší než samotné srovnávání množin Jaccardovým indexem.

Všechny experimenty prováděny se shingly byly výpočetně daleko více náročnější než experimenty s jednotlivými termy, jelikož s rostoucí velikostí shinglu roste i velikost slovníku indexu a také jeho celková velikost. Pro ilustraci složka s indexem 996 testovacích dokumentů s jednotlivými termy dosahuje velikosti 8,5 MB s celkovým počtem 54983 termů v indexu. Složka se shingly velikosti 2 je 19 MB velká a obsahuje celkem 339382 jednotlivých shinglů a adresář s indexem se shingly velikosti 5 dosahuje velikosti 43 MB a má 571691 termů.

9 Implementace vlastního algoritmu

V této části následuje analýza, návrh a popis zvoleného řešení.

9.1 Analýza a návrh řešení

Díky poměrně podrobnému rozboru revizí dokumentů v kapitole 6 již přesně víme, v jakých případech se jedná o revize a můžeme zkusit navrhnout vlastní řešení pro jejich rozpoznávání.

Jedním z problémů detekování revizí je, že i dokument, který byl oproti výchozímu velmi pozměněn, je jeho revizí. V takovýchto případech je nemají šanci algoritmy na hledání podobných dokumentů detekovat. S tímto úkolem by si ovšem mohla dobře poradit KL divergence, která vyjadřuje kolik informace obsahuje jeden dokument na rozdíl od druhého. Tato míra by měla být úspěšná ve zmíněných případech a zároveň by měla dokázat rozlišit, zda byly přidané nějaké nové informace, nebo se dokument zabývá odlišnými věcmi.

Z příkladů revizí uvedených v kapitole č. 6 vyplývá, že téměř vždy budou revizemi dokumenty, které byly upravovány, ale stále pojednávají o stejné osobě, akci, zákonu, nebo například se stále jedná o zápis z dané schůze vedení fakulty. Tyto dokumenty budou často mít v horní části uvedena nějaká data přesně charakterizující o jakou osobu, akci se jedná, nebo například informace o filmu, který dokument recenzuje. Když dokážu zjistit, že tyto informace zůstaly nezměněny a zbylý text dokumentu byl upraven, tak můžu s velkou jistotou tyto dokumenty označit jako revize. Ovšem mohou nastat i případy, kdy například oponent začne psát posudek bakalářské práce a při kontrole zjistí, že špatně zapsal jméno studenta a opraví jej, čímž vznikne revize, ale tento přístup ji nerozezná. Stejně se, myslím, jedná o dobrý přístup pro detekci revizí a dokážu jej v omezené míře realizovat pomocí nástroje na rozpoznávání pojmenovaných entit.

9.2 Popis řešení

Dle definice nabývá KL divergence pro množiny P a Q hodnot z intervalu $\langle 0,1 \rangle$ za splnění počátečních podmínek. První podmínkou je, že sumy P i Q

se rovnají 1 a pro každé i platí, že když Q(i)=0, pak P(i)=0, což při porovnávání dvou ohodnocených dokumentových vektorů nemusí platit. Druhá podmínka je porušena například v případě, kdy se slovo z množiny P vůbec nevyskytuje v množině Q. V tomto případě zároveň dochází ve vzorci k dělení nulou, což je třeba ošetřit. Jednou z variant ošetření je přidávat do sumy nulu, čímž se ale ve výsledku naprosto ztratí to, že daný term se v druhém dokumentu vůbec nevyskytuje. Druhou variantou je přiřazení do druhé hodnoty velmi malé číslo, které možná symbolicky vyjadřuje, že se daný term v druhé množině oproti první množině téměř neobjevuje. První podmínka je splněna normalizací jednotlivých prvků vektorů vydělením sumou a zaručuje získání pouze nezáporných hodnot. U více rozdílných dokumentů KL divergence nabývá hodnot větších než 1 z důvodu rozdílů frekvencí termů v množinách.

Na ukázce číslo D.4 je zobrazen postup zjištění KL divergence mezi dvěma ohodnocenými dokumenty.

Pro rozpoznávání pojmenovaných entit jsem použil český nástroj NameTag, protože testovací data jsou psaná v českém jazyce. Jelikož většina důležitých dat, jako například názvy dokumentů, data vzniku, jméno autora, definujících dokument se nacházejí na jeho úplném začátku a nebo konci, tak můj přístup hledá a porovnává entity v první třetině a poslední čtvrtině obou dokumentů. Pokud si odpovídají ve více než 90%, budou kvalifikovány jako revize. A v případě, kdy hodnota entity na stejném místě v druhém dokumentu nebude odpovídat hodnotě v prvním dokumentu, se algoritmus podívá, jestli se tato hodnota nenachází v okruhu 100 znaků v druhém dokumentu. Tímto eliminuji malé úpravy, které změní pozici entity v druhém dokumentu, ovšem nezmění její hodnotu. Prohledávám větší kus na začátku z toho důvodu, že právě více informací se vyskytuje na počátku, kdežto na konci dokumentu bývají často například jen podpisy a jejich data nebo místa. Je třeba porovnávat entity v obou kombinacích, jak mezi prvním a druhým, tak i mezi druhým a prvním. Když dojde k úspěšnému porovnání alespoň v jedné z těchto kombinací, tak algoritmus označí dokumenty jako revize. Entity jednotlivých dokumentů jsou nalezeny před začátkem rozpoznávání.

Následují experimenty s KL divergencí a kombinacemi rozpoznávání pojemnovaných entit s různými metodami.

9.3 Výsledky testování

KL divergence není symetrická, tudíž její hodnota mezi dokumentem 1 a dokumentem 2 je rozdílná oproti její hodnotě mezi dokumentem 2 a doku-

	Optimální práh	**Výsledná F-míra**
KL divergence s frekvencí termu	0.62	0.918
KL divergence s TF-IDF	0.6	0.938

Tabulka 9.1: Porovnání výsledků KL divergence s různými typy ohodnocení

KL divergence	**Optimální práh**	**Výsledná F-míra**
jednotlivé termy	0.6	0.938
2-shingly	1.1	0.935
3-shingly	2.2	0.892
4-shingly	2.7	0.885
5-shingly	3.6	0.893

Tabulka 9.2: Výsledky dosažené pomocí KL divergence a ohodnocením TF-IDF

mentem 1. Pro vyhodnocování úspěšnosti všech experimentů byla použita cross validace.

Z provedených pokusů vyplynulo, že nejlepších výsledků dosahuje KL divergence spolu s ohodnocením TF-IDF. Výsledky těchto dvou ohodnocení jsou v tabulce číslo 9.1.

V tabulce číslo 9.2 jsou zobrazeny výsledky získané porovnáváním KL divergencí. Podařilo se získat velmi kvalitní výsledky, dokonce hodnoty 0.938 při aplikaci KL divergence na jednotlivé termy ohodnocené TF-IDF.

Následují výsledky testování algoritmů s přidaným nástrojem na rozpoznávání pojmenovaných entit. V tabulce číslo 9.3 jsou zobrazeny výsledky dosažené porovnáváním kosinovou podobností. S přidáním rozpoznáváním entit se podařilo zvýšit úspěšnost rozpoznávání u jednotlivých termů ohodnocených TF-IDF. Se shingly se bohužel zlepšení dosáhnout nepodařilo.

Ani při porovnávání shinglů s KL divergencí a s ropoznáváním entit se nedosáhlo vyšší úspěšnosti rozpoznávání než při experimentech se stejnými

Kosinová vzdálenost s entitami	**Optimální práh**	**Výsledná F-míra**
jednotlivé termy	0.72	0.821
2-shingly	0.54	0.829
3-shingly	0.66	0.839
4-shingly	0.6	0.839
5-shingly	0.6	0.839

Tabulka 9.3: Výsledky kosinové podobnosti s ohodnocením TF-IDF a rozpoznáváním entit

KL divergence s entitami	**Optimální práh**	**Výsledná F-míra**
jednotlivé termy	0.66	0.861
2-shingly	1.1	0.878
3-shingly	3.4	0.862
4-shingly	3.6	0.870
5-shingly	4.4	0.878

Tabulka 9.4: Výsledky KL divergence s rozpoznáváním entit a shingly

metodami bez rozpoznávání entit. Získané výsledky KL divergencí s entitami a shingly jsou ukázány v tabulce číslo 9.4.

I když se ukázalo, že s porovnáváním entit revizí se mi nepodařilo dosáhnout lepších výsledků, tak jedním z poznatků z uskutečněných experimentů je, že se mi podařilo díky hledání odpovídajících pojmenovaných entit nejen na stejném místě, ale i v okolí, které je popsané v předcházející části, zvýšit průměrně úspěšnost vyhledávání o více než jednu desetinu.

Závěrem ze všech experimentů je, že použití KL divergence jako porovnávací míry je velmi dobrým přístupem pro detekci revizí. S její aplikací na jednotlivé termy s TF-IDF bylo při testování dosaženo nejvyšší úspěšnosti 0.938, což je velmi dobrý výsledek. Dalším výstupem je, že přidání porovnávání pojmenovaných entit na začátku a na konci dokumentů, dokáže mírně zvýšit úspěšnost vyhledávání s kosinovou podobností, ovšem vylepšit tímto způsobem výsledky KL divergence se této testovací množině nepodařilo. Celkový výsledek všech provedených testů je ovšem závislý na zvolené testovací množině a při aplikaci na rozpoznávání revizí například jen anotovaných dokumentů by algoritmy dosáhly jiných výsledků.

Nedostatkem zvoleného řešení je výpočetní náročnost všech experimentů. Možností urychlení výpočtů je hashování uchovávaných shinglů.

10 Závěr

V první části práce jsem zmapoval používané přístupy ke zpracování, indexaci a vyhledávání dokumentů, což mi umožnilo se seznámit se základními principy informatického oboru zvaného Information retrieval. Dále byly vytvořeny revize ke 100 dokumentům a získána testovací kolekce čítající 996 dokumentů. Poté byla provedena implementace některých ze stávajících algoritmů pro vyhledávání podobných dokumentů a jejich otestování na vytvořených testovacích datech. Nejlepším z nich se pro detekci revizí ukázal shingling s velikostí 5, s ohodnocením pomocí TF-IDF a vyjádřením podobnosti dokumentů kosinovou vzdáleností, který při aplikaci na 100 testovacích dokumentech metodou cross validace dosáhl nejvyšší úspěšnosti 0.878 vyjádřené F-mírou.

V poslední části byl navržen a implementován nový algoritmus pro detekci revizí. Pro jeho správný návrh bylo důležité přesně definovat, co je to revize. Pasáž, která se tímto zabývá a ve které byly uvedeny i konkrétní příklady revizí, považuji za jednu z nejdůležitějších v této práci. V této části byla implementována a testována KL divergence jako míra pro vyjádření podobnosti dvou dokumentů. Tato metrika se ukázala jako velmi úspěšná při detekci revizí, protože při její aplikaci na jednotlivé termy ohodnocené TF-IDF dosáhla úspěšnosti s metodou cross validace 0.938. Dále jsem použil český nástroj na rozpoznávání pojmenovaných entit a otestoval jsem jeho kombinace s existujícími metrikami. U KL divergence se mi nepodařilo dosáhnout vylepšení úspěšnosti, zejména i proto, že samotná KL divergence dosáhla velmi dobrých výsledků. Mírného zlepšení úspěšnosti jsem dosáhl při přidání rozpoznávání pojmenovaných k porovnávání dokumentů s kosinovou podobností a ohodnocení pomocí TF-IDF.

Byla vytvořena práce, která zkoumá možnosti použití principů rozpoznávání pojmenovaných entit pro detekci revizí. Může sloužit jako dobrý teoretický základ pro další experimenty v této oblasti. Při testování bylo dosaženo velmi dobrých výsledků, jejichž ověření na větším množství testovacích dokumentů by mohlo ukázat, zda by se dal tento algoritmus využívat v praxi.

Literatura

[1] *Package org.apache.lucene.analysis Description* [online]. 2016. [Online; accessed 10-May-2017s]. Dostupné z: https://lucene.apache.org/core/6_2_0/core/org/apache/lucene/analysis/package-summary.html.

[2] Kullback-Leibler Divergence Explained. Dostupné z: https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained. [Online; accessed 12-June-2017].

[3] Welcome to Apache Lucene, 2016. Dostupné z: https://lucene.apache.org/. [Online; accessed 21-December-2016].

[4] More Like This Query. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-mlt-query.html#query-dsl-mlt-query. [Online; accessed 25-May-2017].

[5] ABZ.cz: slovník cizích slov, . Dostupné z: http://slovnik-cizich-slov.abz.cz/web.php/slovo/revize-revise. [Online; accessed 10-June-2017].

[6] Cambridge dictionary - Meaning of "revision" in the English Dictionary, . Dostupné z: http://dictionary.cambridge.org/dictionary/english/revision. [Online; accessed 12-June-2017].

[7] LINGEA, . Dostupné z: http://www.nechybujte.cz/slovnik-soucasne-cestiny/revize? [Online; accessed 10-June-2017].

[8] Macmillan dictionary - revision - definition and synonyms, . Dostupné z: http://www.macmillandictionary.com/dictionary/british/revision. [Online; accessed 12-June-2017].

[9] Slovník spisovného jazyka českého, . Dostupné z: http://ssjc.ujc.cas.cz/search.php?hledej=Hledat&heslo=revize&sti=EMPTY&where=hesla&hsubstr=no. [Online; accessed 10-June-2017].

[10] Class Similarity. Dostupné z: https://lucene.apache.org/core/3_6_0/api/core/org/apache/lucene/search/Similarity.html. [Online; accessed 21-December-2016].

[11] Apache Tika - a content analysis toolkit. Dostupné z: http://tika.apache.org/. [Online; accessed 25-May-2017].

[12] Baeza-Yates, R. – Ribeiro-Neto, B. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., 1999. ISBN 020139829X.

[13] Blei, D. M. – Ng, A. Y. – Jordan, M. I. Latent Dirichlet Allocation. *Journal of Machine Learning Research*. January 2003, 3, 4-5, s. 993–1022. Dostupné z: http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf.

[14] Croft, B. – Metzler, D. – Strohman, T. *Search Engines: Information Retrieval in Practice 1st Edition*. Pearson, 2009. ISBN 0136072240.

[15] Kotíková, B. M. Aplikace metod předzpracování při dolování znalostí z textových dat. Master's thesis, Mendelova univerzita v Brně, Provozně ekonomická fakulta, 2014. Vedoucí diplomové práce doc. Ing. František Dařena, Ph.D.

[16] Krátký, M. Využití SVD pro indexování latentní sémantiky. *Department of Computer Science, VŠB-Technical University of Ostrava, Czech Republic*. 2002. Dostupné z: http://www.cs.vsb.cz/arg/techreports/lsi-svd_ma.pdf.

[17] Lecocq, D. *Near-Duplicate Detection* [online]. 2015. Dostupné z: https://moz.com/devblog/near-duplicate-detection/.

[18] Leskovec, J. – Rajaraman, A. – Ullman, J. D. *Mining of Massive Datasets*. Cambridge University Press, 2011. ISBN 9781139058452.

[19] Liu, H. et al. BioLemmatizer: a lemmatization tool for morphological processing of biomedical text. *Journal of Biomedical Semantics*. 2012.

[20] Manning, C. D. – Raghavan, P. – Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 978-0-521-86571-5.

[21] Phillips, J. M. *Jaccard Similarity and Shingling* [online]. 2013. Dostupné z: https://www.cs.utah.edu/~jeffp/teaching/cs5955/L4-Jaccard+Shingle.pdf.

[22] Salton, G. The SMART Retrieval System: Experiments in Automatic Document Processing. *Prentice Hall*. 1971.

[23] Schwarz, J. *Současný stav a trendy automatické indexace dokumentů:přehledová studie* [online]. Praha, 2003. [cit. 2016/12/16]. Dostupné z: http://full.nkp.cz/nkdb/docs/studie/MAIobsah.html.

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

[25] Straková, J. *ROZPOZNÁVÁNÍ POJMENOVANÝCH ENTIT* [online]. CzechEncy – Nový encyklopedický slovník češtiny online. [cit. 2017/6/15]. Dostupné z: `https://www.czechency.org/slovnik/ROZPOZN%C3%81V%C3%81N%C3%8D%20POJMENOVAN%C3%9DCH%20ENTIT`.

[26] Wang, X. – Grimson, E. *Spatial Latent Dirichlet Allocation* [online]. Computer Science and Artificial Intelligence Lab Massachusetts Institute of Technology, Cambridge, MA, 02139, USA, 2007. [cit. 2016/12/21]. Dostupné z: `http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf`.

[27] Wiemer-Hastings, P. *Latent Semantic Analysis* [online]. DePaul University School of Computer Science, Telecommunications, and Information Systems 243 South Wabash Avenue Chicago IL 60604, USA, 2004. [cit. 2016/12/16]. Dostupné z: `https://pdfs.semanticscholar.org/6055/27f92a5f8ebaae7405d7ae78ea03f830fb5a.pdf`.

Seznam příloh

Uživatelská příručka - Příloha A

Obsah CD - Příloha B

Třídy českých jazykových modelů projektu NameTag - Příloha C

Ukázky zdrojových kódů - Příloha D

A Uživatelská příručka

V práci bylo implementováno několik algoritmů a ty jsou k práci přiloženy jako aplikace ve formátu spustitelného *jar* souboru z toho důvodu, aby mohlo dojít k ověření provedených experimentů.

A.1 Spuštění programu

Program se spouští z příkazové řádky s jedním parametrem, kterým je název konfiguračního textového souboru definující formu, ve které se aplikace spustí. Konfigurační soubory mají danou strukturu, kdy na jednotlivých řádkách je vždy dvojice název parametru a hodnota parametru oddělené mezerou. Při spuštění programu je nutné pro použití knihovny na rozpoznávání pojmenovaných entit přidat do *java library path* sdílenou dynamickou knihovnu s příponou libnametag_java.so. Tyto knihovny jsou pro 64-bitové a 32-bitové linuxové distribuce přiloženy k programu na CD. Ukázkový příkaz pro spuštění experimentu s kosinovou vzdáleností spolu s roznáváním entit v 64-bitovém Linuxu vypadá takto.

java -Djava.library.path=libnametag_java.so -jar aplikace.jar experimenty-/experimentyKosinovaVzdalenostEntity/EntityKosJednotliveTermy.txt

K programu je přiložen skript s názvem *spusteni.sh*, po jehož spuštění se provedou všechny provedené experimenty. Spuštění se provede následujícím příkazem.

./spusteni.sh

Indexy a k nim vytvořené seznamy dokumentů se společnými termy jsou již vytvořené a přiložené k práci z toho důvodu, že nalezení právě seznamu dokumentů, které mají společné termy je časově velmi náročné. K případnému vytvoření vlastních indexů slouží skript *vytvoreniIndexu.sh*, po jehož spuštění se dokumenty ze složky *testovaciSoubory/vse* zaindexují po jednotlivých termech do složky *index/index* a poté po shinglech velikosti 2-5 do složek pojmenovaných podle velikosti shinglu. K těmto indexům se také vytvoří nové indexy s uloženými seznamy ID dokumentů, se kterými se budou porovnávat.

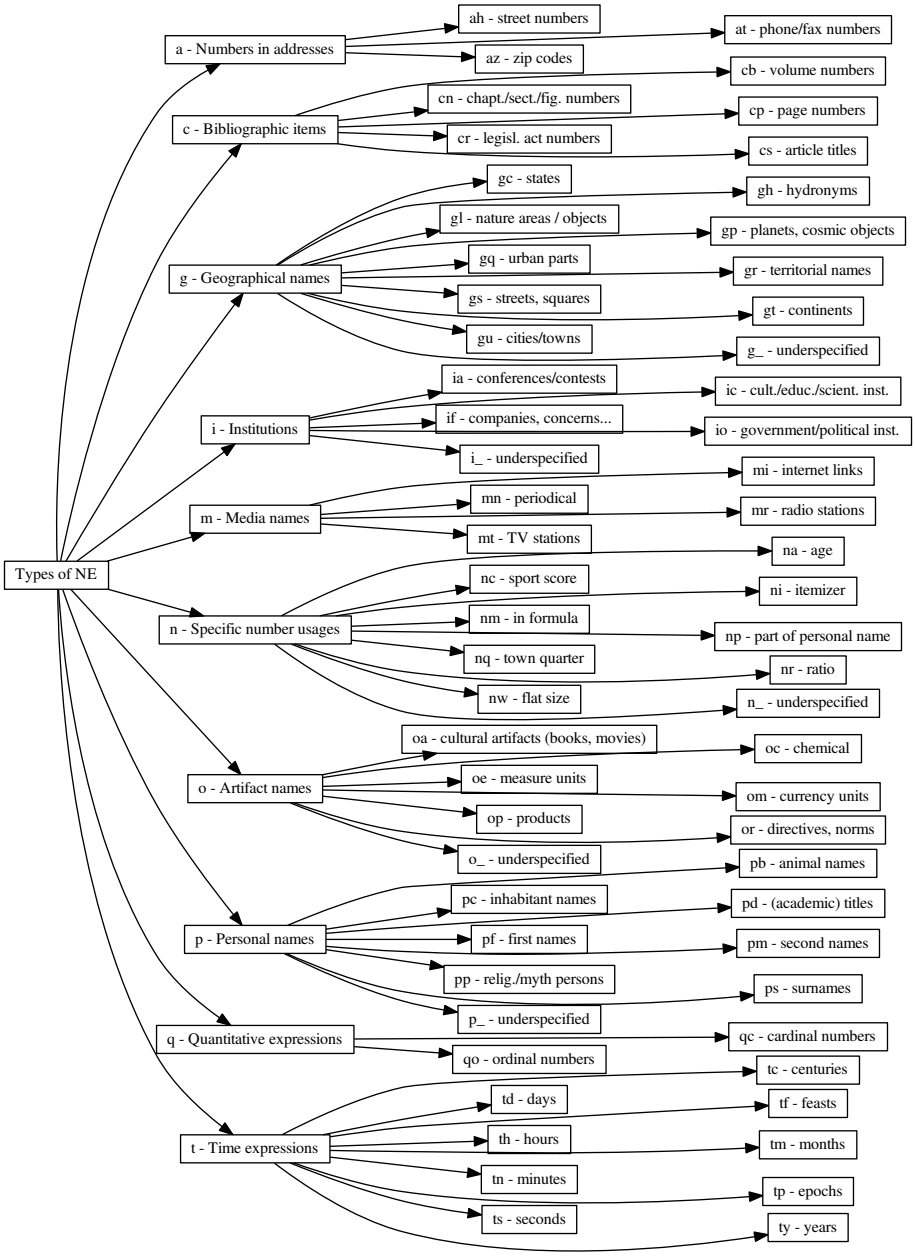
Oba tyto skripty byly vytvořeny pro 64-bitový Linux.

B Obsah CD

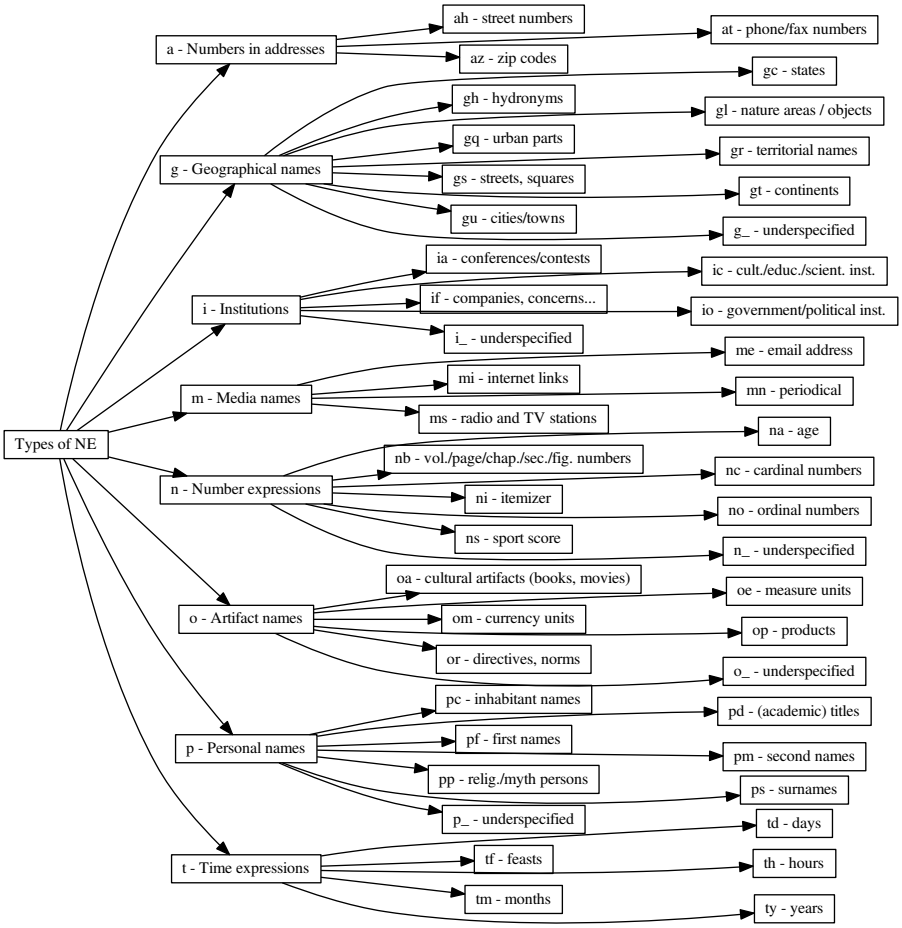
K práci je přiloženo CD, které obsahuje následující soubory.

- **bakalarskaPrace** - složka se zdrojovými LaTeXsoubory, která zároveň obsahuje i tuto práci ve formátu *pdf*.
- **experimenty** - složka s konfiguračními soubory pro spuštění experimentů.
- **index** - složka s vytvořenými indexy.
- **sdilenaKnihovna32** - obsahuje sdílenou knihovnu *libnametag_java.so* ve verzi pro použití na 32-bitovém Linuxu.
- **slovnik** - složka, která obsahuje natrénovaný jazykový model nutný pro rozpoznávání pojmenovaných entit.
- **src** - složka se zdrojovými kódy.
- **testovaciSoubory** - adresář obsahující testovací dokumenty.
- **vytvorIndex** - složka s konfiguračními soubory pro vytvoření indexů a také pro nalezení a zapsání do indexů pro každý dokument seznam dokumentů, se kterými se bude porovnávat.
- **aplikace.jar** - spustitelná aplikace.
- **README.txt** - uživatelská příručka aplikace.
- **spusteni.sh** - skript, po jehož spuštění se provedou všechny experimenty vyzkoušené v práci.
- **vytvoreniIndexu.sh** - skript, který po spuštění zaindexuje dokumenty ve složce *testovaciSoubory*.
- **libnametag_java.so** - sdílená knihovna, kterou je nutné pro experimenty s entitami přidat do *java build path*.

C Třídy českých jazykových modelů projektu NameTag



Obrázek C.1: 62 tříd dvoustupňové hirearchie modelu 1.1



Obrázek C.2: 46 tříd dvoustupňové hirearchie modelu 2.0

D Ukázky zdrojových kódů

D.1 Specifikace vlastností pole pro udržování textu

```
public static final FieldType TYPE_STORED = new FieldType();

  static {
    TYPE_STORED.setIndexOptions(IndexOptions.
        DOCS_AND_FREQS_AND_POSITIONS);
    TYPE_STORED.setTokenized(true);
    TYPE_STORED.setStoreTermVectors(true);
    TYPE_STORED.setStoreTermVectorPositions(true);
    TYPE_STORED.setStored(true);
    TYPE_STORED.freeze();
  }
```

Ukázka kódu D.1: Specifikace vlastností pole pro udržování textu

D.2 Získání ováhovaného vektoru z dokumentu

```
Map<String, Double> getVektor(IndexReader reader, int docId)
    throws IOException {
  Terms vector = reader.getTermVector(docId, "text");
  Map<String, Integer> dokumentFrekvence = new HashMap<String,
      Integer>();
  Map<String, Integer> termFrekvence = new HashMap<String,
      Integer>();
  Map<String, Double> tfIdfFrekvence = new HashMap<String,
      Double>();
  TermsEnum termsEnum = vector.iterator();
  BytesRef text = null;
  int pocetSlovVdokumentu = 0;
  while ((text = termsEnum.next()) != null) {
    String term = text.utf8ToString();
    pocetSlovVdokumentu += termsEnum.totalTermFreq();
    dokumentFrekvence.put(term, reader.docFreq(new Term("text"
        , term)));
    termFrekvence.put(term, (int) termsEnum.totalTermFreq());
  }
  for (String term : dokumentFrekvence.keySet()) {
```

```
        double tf = (double) termFrekvence.get(term) / (double)
            pocetSlovVdokumentu;
        double idf = Math.log((double)reader.maxDoc()/(double)
            dokumentFrekvence.get(term));
        double vaha = tf * idf;
        tfIdfFrekvence.put(term, vaha);
      }
      return tfIdfFrekvence;
    }
```

Ukázka kódu D.2: Získání ováhovaného vektoru z dokumentu

D.3 Vytvoření vlastního analyzéru na generování shinglů

```
public class ShingleAnalyzer extends Analyzer{

    int minShingle, maxShingle;

    public ShingleAnalyzer(int minShingle, int maxShingle) {
      this.minShingle = minShingle;
      this.maxShingle = maxShingle;
    }

    @Override
    protected TokenStreamComponents createComponents(String arg0)
        {
        StandardTokenizer source = new StandardTokenizer();
        CzechAnalyzer czech = new CzechAnalyzer();
        TokenStream tokenStream = new StopFilter(source, czech.
            getStopwordSet());
        ShingleFilter sf = new ShingleFilter(tokenStream,
            minShingle, maxShingle);
        sf.setOutputUnigrams(false);
        tokenStream = sf;
      return new TokenStreamComponents(source, sf);
    }
}
```

Ukázka kódu D.3: Vytvoření vlastního analyzéru na generování shinglů

D.4 Zjištění KL divergence mezi dvěma ohodnocenými dokumenty

```
public double vratMiru(Map<String, Double> v1, Map<String,
    Double> v2) {
  double sum1 = spoctiSumu(v1);
  double sum2 = spoctiSumu(v2);
  double vysledek = 0.0;
  for (String string : v1.keySet()) {
    double frekvence2 = 0.0;
    double frekvence1 = (double) v1.get(string) / sum1;
    if (v2.containsKey(string)) {
      frekvence2 = (double) v2.get(string) / sum2;
    }else{
      frekvence2 = 0.1 * Math.pow(10, -10);
    }
    double prirustek = frekvence1 * (Math.log(frekvence1 /
        frekvence2));
    vysledek += prirustek;
  }
  return vysledek;
}
```

Ukázka kódu D.4: Zjištění KL divergence mezi dvěma ohodnocenými dokumenty