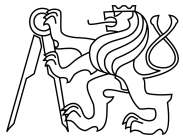
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Automatické pojmenovávání skupin slov
Student:	Michal Štefančík
Vedoucí:	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2016/17

Pokyny pro vypracování

1) Proveďte rešerši postupů, jakými lze lingvisticky zatřiďovat slova do skupin významově podobných slov.
2) Na základě předchozí rešerše vyberte vhodné postupy, kterými lze přiřadit skupině slov společný název.
3) Vybrané postupy diskutujte s vedoucím práce a implementujte ve vhodném programovacím jazyku.
4) Implementované postupy ověřte na reálných datech. Výběr testovacích dat konzultujte s vedoucím práce.

Seznam odborné literatury

[1] HEDDEN, Heather. The accidental taxonomist. Medford, N.J.: Information Today, c2010, xxix, 442 p. ISBN 15-738-7397-7.
[2] Camina, Steven. A Comparison of Taxonomy Generation Techniques Using Bibliometric Methods: Applied to Research Strategy Formulation. EECS Thesis, Massachusetts Institute of Technology, 2010.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 15. prosince 2014

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalárska práca

Automatické pojmenovávání skupin slov

Michal Štefančík

Vedúci práce: doc. RNDr. Ing. Marcel Jiřina, Ph.D.

8. januára 2017

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať.

Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, a za akýmkoľvek účelom (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené. Každá osoba, ktorá využije vyššie uvedenú licenciu, sa však zaväzuje priradiť každému dielu, ktoré vznikne (čo i len čiastočne) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného či zpracovaním Diela (vrátane prekladu), licenciu aspoň vo vyššie uvedenom rozsahu a zároveň sa zaväzuje sprístupniť zdrojový kód takého diela aspoň zrovnateľným spôsobom a v zrovnateľnom rozsahu ako je zprístupnený zdrojový kód Diela.

V Praze 8. januára 2017 .....

Odkaz na túto prácu

Štefančík, Michal. *Automatické pojmenovávání skupin slov*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cieľom práce je navrhnúť a implementovať algoritmus, ktorý zhluku sémanticky podobných slov priradí spoločný názov. V algoritme sa používajú metódy využívajúce externé zdroje vedomostí, a to konkrétne slovník WordNet. Ako testovacie dáta slúži databáza Wikipédie. Algoritmus je testovaný na presnosť výsledkov.

Klíčová slova text mining, lda, ontológia, Wikipédia, WordNet, tématické modelovanie, sémantická podobnosť

Abstract

The purpose of this research is design and implement algorithm to assign label to cluster of words with semantic similar meaning. Algorithm is using external knowledge bases, namely WordNet dictionary. Database of Wikipedia is used as testing data. Algorithms are tested for accuracy of results.

Keywords text mining, lda, ontology, Wikipedia, WordNet, topic modeling, semantic similarity

Obsah

Zoznam obrázkov

Úvod

Organizácia dát začala s rapídnym narastaním výpočtovej techniky postupne získavať na dôležitosti. S rozdeľovaním objektov a vedomostí začal už v staroveku Aristoteles, no nikdy tomuto problému ľudia nevenovali toľko pozornosti ako posledných pár desaťročí. Je to z toho dôvodu, že množstvo informácií je dnes už také veľké, že uspokojivo kategorizovať, vyhľadávať v nich alebo z nich získavať informácie nie je v ľudských môžnostiach.
Týmto problémom ako jedným z mnohých sa zaoberá vedná disciplína, nazývaná data mining, čo sa dá preložiť aj ako hĺbková analýza dát. Jej využitie sa často objavuje v bankovníctve, biofarmaceutickom priemysle či bezpečnostných technológiách, ale aj vo všetkých odvetviach, kde je potrebné spracovávať dáta aby mohli byť použité pri procese rozhodovania.
Už konkrétnejšou podmnožinou data miningu je text mining. Tento pojem sa dá vysvetliť ako proces dolovania vysoko kvalitných informácií z textu. Text mining napríklad zahŕňa lexikálne analýzy pre skúmanie frekvenčného rozdelenia slov, prediktívne analýzy a v neposlednom rade aj spracovanie prirodzeného jazyka (NLP).
Jedným z modelov používaných v text miningu je tématické modelovanie, ktoré v spojitosti s NLP znamená vytvorenie štatistického modelu, ktorý má za úlohu odhaliť abstraktné tématické oblasti v súbore dokumentov. Typickým využitím tejto skupiny metód je odhaľovanie sémantickej informácie zapuzdrenej v texte.
Ako príklady využitia tématického modelovania môžu slúžiť napríklad automatické tagovanie jednotlivých dokumentov vo veľkom súbore textov, automatické priradenie názvu bezmennému dokumentu, automatické generovanie taxonomickej štruktúry pre kategorizáciu dokumentov alebo aj zhluková analýza dokumentov.
Táto práca sa venuje návrhu algoritmu pre hľadanie spoločnej sémantickej informácie práve v takých zhlukoch, ktoré boli vytvorené v nejakej predchádzajúcej analýze, či už zhlukovej analýze slov podľa frekvencie (tzv. word clouds), alebo iných zložitejších analýzach (LSA, LDA).

Za predpokladu, že vstupom do algoritmu má byť iba zhluk slov a nie súbor dokumentov, je potrebné na ohodnotenie zhluku využitie externého zdroja vedomostí. Práca rozoberá metódy využívajúce ontológie, a to konkrétne často používanú lexikálnu databázu WordNet Princetonskej univerzity.
Prvá časť práce sa venuje popisu teoretického základu text miningu, reprezentácie textových dokumentov, merania sémantickej podobnosti slov. V druhej časti sa nachádza návrh algoritmu vyhodnocovania kandidátov pre pomenovanie zhluku slov a návrh algoritmu generujúceho testovacie dáta. V tretej časti práce sa nachádza popis implementácie daných algoritmov. Posledný cieľ práce je tieto algoritmy overiť na reálnych dátach a porovnať ich z hľadiska presnosti.

Kapitola 1

Teoretický aparát

Keďže má človek od prírody v mozgu vytvorenú schopnosť reflexívnej abstrakcie, dokáže veľmi ľahko a rýchlo priradiť skupine objektov, ktoré majú spoločné vlastnosti nejaký názov. Tento problém je pre človeka ľahko riešiteľný, avšak algoritmicky vyriešiť tento problém je náročnejšie. Zvolenie správnej značky je subjektívne, teda u každého človeka môže byť zvolená značka pre rovnaký zhluk rozdielna a napriek tomu ju obaja môžu považovať za správnu.
Čo je však jednoznačné je to, že pokiaľ sú objekty spájané podľa určitej spoločnej vlastnosti do skupiny, tak potom táto skupina s určitosťou nesie nejakú spoločnú informáciu a práve tú máme za úlohu odhaliť.
Štandardné zhlukovacie algoritmy, či algoritmy zložitejších analýz nepriraďujú skupinám slov žiadnu značku. Tým pádom sa dá predpokladať, že tento problém značkovania je nezávislý na konkrétnom zhlukovacom algoritme, avšak kvalita vygenerovaných zhlukov, teda miera sémantického šumu (slov, ktoré nenesú žiadnu alebo od väčšiny odlišnú sémantickú informáciu) ovplyvňuje kvalitu výsledkov. Tým pádom je dôležité zvoliť správnu metódu pre generovanie testovacích dát, aby výsledné testovanie algoritmu nebolo ovplyvnené slabou kvalitou zhlukov.
Aby sme popísali tento problém je doležité si určiť aké vlastnosti by mala mať táto značka. Značka musí byť opisná a ľudsky čitateľná. Správna značka je taká, ktorá vystihne celý zhluk a nejakým spôsobom ho sumarizuje. Ďalšia dôležitá vlastnosť, ktorú je potreba spomenúť je, že podobne ako zhlukovací algoritmus nevytvára iba zhluky objektov, ktoré majú spoločné vlastnosti, ale snaží sa aj aby bola čo najväčšia rozdielnosť medzi skupinami, [3] tak značka zhluku nielen že musí sumárne vystihnúť zhluk, ale aj ho jasne odlíšiť od ostatných zhlukov.
Dôležitým faktorom je aj, že značka by mala mať určité dominantné vlastnosti, aby mohla správne vystihnúť celú skupinu, na rozdiel od hľadania najbližšieho prvku k danému zhluku (voľba najcentrálnejšieho slova v zhluku). Najvhodnejšie riešenie je pozerať sa na tento problém ako na problém ohodnotenia deskriptora (angl. deskriptor ranking problem). [4]

Ten môžeme rozdeliť na tri podproblémy, a to: výber kandidátov na označenie zhluku slov, ohodnotenie kandidátov a zoradenie podľa podľa relevancie a výber prvých n-najvhodnejších kandidátov.

1.1 Reprezentácia textových dokumentov

1.1.1 Vector space model

Je to jeden z najpoužívanejších modelov pre reprezentáciu textových dokumentov. Základ tohto modelu tkvie v lineárnej algebre. Slová sú reprezentované ako body v n-rozmernom priestore, pričom každá dimenzia modelu odpovedá práve nejakému dokumentu. [2]

$$\begin{array}{c} \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & \dots & d_n \end{array} \\ \begin{pmatrix} 1 & 0 & \dots & 1 \\ 0 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} \end{array}$$

Poznáme viac pristupov k tomu, aby sme označili prítomnosť slova v jednotlivých dokumentoch. Jednou z možností je binárna reprezentácia. V prípade, že sa slovo v dokumente nachádza je reprezentované 1 a ak sa nenachádza tak 0. Táto reprezentácia je jednoduchá a často dostačujúca. Avšak nenesie doležitú informáciu, a to počet výskytov slova v texte. Avšak pre svoju minimalistickosť a jednoduchosť je niekedy používaná. Iná možnosť je udržiavať si počet výskytov slova v dokumente.

1.1.2 Metodika hodnotenia relevancie

Predchádzajúce reprezentácie májú tú nevýhodu, že častým slovám, ktoré sa vyskytujú vo všetkých textoch prikladajú väčšiu váhu, aj keď tieto slová vôbec nemusia niesť sémantickú informáciu. Ak budeme predpokladať tvrdenie, že doležité slová sú tie, ktoré sa vyskytujú často v jednom dokumente, avšak v iných dokumentoch sa často nevyskytujú, tento problém vyriešime.
Reprezentácia založená na tejto myšlienke sa nazýva TF-IDF. Táto váhová metrika je vlastne súčinom dvoch metrík, a to frekvencie slov v dokumente, ktorá reprezentuje dôležitosť slova v dokumente a inverznej dokumentovej frekvencie reprezentujúcej dôležitosť slova v kolekcii dokumentov. [1]

$$tfidf_{w_i,d_j} = tf_{w_i,d_j} \cdot idf_{w_i} \tag{1.1}$$

$$idf_{w_i} = \log \frac{|D|}{|\{d \in D : w_i \in d|} \tag{1.2}$$

Zložka $tf_{i,j}$ vyjadruje frekvenciu výskytu slova v dokumente, normalizovanú počtom slov v danom dokumente. Normalizuje sa z dôvodu nadhodnocovania slov v dlhých dokumentoch. Jedná sa teda o pomer počtu výskytov slova ku počtu všetkých slov v danom dokumente.
Zložka idf_i označuje dôležitosť slova. Je to logaritmicky škálovaný pomer počtu všetkých dokumentov v korpuse $|D|$ a počtu dokumentov, v ktorých sa dané slovo objavuje aspoň raz.

1.1.3 Meranie podobnosti slov

Keďže slová vo vektorovom priestore dokumentov sú reprezentované vektormi, problém merania sémantickej podobnosti jednotlivých slov môžeme považovať za problém merania podobností daných vektorov. Poznáme viac matematických nástrojov ako určiť veľkosť podobnosti medzi dvoma nenulovými vektormi.
Najpoužívanejší spôsob takéhoto merania sa nazýva kosínusová podobnosť. Spočíva v meraní kosínu uhlu medzi dvoma porovnávanými vektormi v n-rozmernom priestore. Keď je uhol medzi nimi malý dá sa povedať, že vektory sú si podobné a naopak, pokiaľ sa uhol blíži pravému uhlu, dá sa povedať, že sú si málo podobné.
Vzorec pre výpočet kosínusovej podobnosti prebraný z [2] kde $w_a \cdot w_b$ je skalárny súčin vektorov reprezentujúci slová, ktoré porovnávame a $||w_a||||w_b||$ je súčin absolútnych veľkostí oboch vektorov.

$$\cos(\theta) = \frac{w_a \cdot w_b}{\|w_a\|\|w_b\|} \tag{1.3}$$

Vo všeobecnosti môže kosínusová podobnosť nadobúdať hodnoty v intervale <-1,1>. Pri použití v text miningu pracujeme iba s kladnými hodnotami zložiek vektorov (počet výskytov slova v dokumente nemôže byť záporný), preto táto podobnosť nadobúda hodnoty v intervale <0,1>.

Tento problém merania podobnosti slov reprezentovaných jednotlivými vektormi matice vektorového priestoru dokumentov sa dá presunúť aj na riešenie problému merania podobnosti jednotlivých dokumentov. Pokiaľ budeme vykonávať merania na transponovanej matici tohto priestoru, jednotlivé vektory budú reprezentovať dokumenty a môžeme na nich aplikovať kosínusovú podobnosť.

Existuje viac možností ako merať podobnosť týchto vektorov. Medzi najpoužívanejšie metriky patrí: euklidovská vzdialenosť, Minkowského metrika a podobne. [1]
Nedá sa jasne povevedať, ktorý zo spôsobov merania podobnosti je absolútne najlepší. Porovnanie každej z nich záleží od problému, na ktorý je meranie aplikované. Napríklad, čo sa týka práve kosínusovej podobnosti, pokiaľ vy-

hodnotí v trasnponovanej matici dva dokumenty ako podobné je veľká šanca, že majú spoločnú tému, avšak má sklony k falošným negatívnym výsledkom. Je to z dôvodu výskytu synoným v dokumentoch, t.j. dva dokumenty môžu obsahovať zapuzdrenú spoločnú tému, avšak sa o nej bavia inými slovami prostredníctvom synoným.
Ak odhliadneme od modelu vektorového priestoru dokumentov, existujú aj iné metódy určovania podobnosti slov zvané topologická podobnosť. Tieto metódy využívajú ontológie a spočívajú v meraní najkratšej cesty medzi dvoma slovami.
Väčšina metód využíva hľadanie najnižšieho spoločného predchodcu angl. lowest common subsumer, ktorým rozumieme najnižší uzol v lexikálnej ontológii s najkratšou cestou od oboch porovnávaných slov. Na základe dĺžok najkratších ciest porovnávaných slov a tohto predchodcu od koreňa ontológie sa potom odvodzujú rôznymi metódami sématnické podobnosti slov. Medzi tieto metódy patrí: Wu & Palmer's Measure, Leakcock & Chodorow's Measure, Li's Measure, Resnik's Measure alebo Jiang's Measure a dalšie. [17]

1.2 Možnosti zníženia dimenzii

Pri použití vector space modelu máme častokrát problémy spôsobené veľkosťou priestoru. Príčinou toho je fakt, že s pridaním dalšej dimenzie rastie veľkosť priestoru exponenciálne. Tieto problémy nastávajú pri analyzovaní veľkých priestorov obsahujúcich tisíce dimenzií. Vykonávanie zložitejších výpočtov na takomto veľkom priestore ako napr. SVD je časovo náročné. Odhliadnuc od časovej náročnosti spracovania takéhoto priestoru, tieto reprezentácie obsahujú mnoho šumu, t.j. slov, ktoré nám nedávajú žiadnu sémantickú informáciu.
K zníženiu dimenzií v priestore existuje viac prístupov. Súhrn všetkých problémov súvisiacich s príliš veľkým počtom dimenzii sa nazýva kliatba dimenzionality, angl. curse of dimensionality.

1.2.1 Selekcia príznakov

Jeden z prístupov zníženia počtu dimenzií sa nazýva selekcia príznakov, angl. feature selection. Je to proces odstraňovania takých objektov z kolekcie, ktoré sú irelevantné alebo redundantné.
[1] Redundantnosť znamená, že výskyt objektu v danej kolekcii už nijak nezvyšuje mieru informácie, ktorú nám objekt dáva. Takéto objekty zbytočne zvyšujú počet dimenzií priestoru, avšak nezvyšujú množstvo informácie. Irelevantné objekty nenesú v sebe žiadnu informáciu. To sa týka takzvaných neplnovýznamových slov, čo sú napríklad spojky, zámená, predložky a pod. Touto metódou znižujeme nielen dobu behu algoritmu nad kolekciou, ale aj presnosť výsledkov, teda redukovanie šumu.
Medzi selekciu príznakov patria metódy ako stopwords deletion, čo znamená

odstránenie slov z textov za pomoci slovníka. Tento slovník obsahuje slová, o ktorých určite vieme, že nenesú žiadnu sémantickú informáciu.
Ďalšou metódou výberu príznakov je využitie štatistických metód na odstránenie slov, ktoré nesú len veľmi málo sémantickej informácie. Podľa [2] a [1] medzi tieto techninky patrí chi-squared test, meranie miery spoločnej informácie.
Existujú aj dalšie, viac agresívne metódy, a to odstraňovanie slov podľa percentuálneho výskytu v kolekcii dokumentov. Zvolíme si hranicu (napr. 15%), ktorú musí výskyt slova prekročiť na to, aby bolo brané do úvahy. Táto metóda sa používa pri naozaj veľkých súboroch dokumentov obsahujúcich státisíce slov, kde je potrebné rapídne zmenšiť počet spracovávaných slov. Tak isto sa touto meódou strácajú aj niektoré slová nesúce informáciu a naopak neodstraňujú sa ňou neplnovýznamové a často opakujúce sa slová.

1.2.2 Extrakcia príznakov

Ďalším prístupom redukcie počtu dimenzií je extrakcia príznakov. Je to proces, ktorý kolekciu objektov redukuje na menšiu kolekciu spájaním takých objektov, ktoré sú považované za redundantné. [1]
Ako príklad môžu slúžiť rôzne pády jedného podstatného mena, ktoré môžeme považovať za jeden objekt, pretože obe slová nesú rovnakú sémantickú informáciu napriek tomu, že ich textová reprezentácia je rozdielna. Alebo další príklad nastáva pri synonymii slov, kde dve rôzne textové reprezentácie slova nesú úplne rovnaký význam. Tieto rozdielne reprezentácie zvyšujú počet dimenzií, ale nezvyšujú množstvo nesenej informácie.
Medzi techniky extrakcie príznakov patrí stemming, čo je proces redukovania slov na koreň slova. Nevýhodou stemmingu je to, že spracovávané slovo častokrát deformuje a tým sťažuje jeho dalšie spracovávanie. Dokonca takáto deformácia slova môže ľahko zmeniť jeho význam. Ako príklad môže slúžiť práve slovo stemming, ktoré by táto metóda zmenila na stem, ktoré však má úplne iný význam.
Ďalšou technikou je lematizácia, čo je proces nahradzovania slov lemmou pre dané slovo. Lemma je jednoznačný slovníkový výraz určitej množiny slov s rovnakým významom. Na rozdiel od stemmingu nejde len o spájanie slov s rovnakým koreňom, ale o spájanie slov na základe slovníka bez vedomostí či ich textová forma zdieľa nejakú informáciu. Stemming je rýchlejšia metóda extrakcie príznakov ako lematizácia, no jej použitie sprevádza mnoho problémov a na rozdiel od lematizácie nerieši problém synonymie.

1.2.3 NP chunking

Táto metóda znamená proces extrakcie neprekrývajúcich sa fráz podstatných mien z textu (angl. noun phrase). Pri použití tejto metódy sme schopní z textu získavať nielen slová, ale aj celé slovné spojenia. Táto metóda je užitočná, pretože zachováva spojenia slov, ktoré osamote strácajú svoj sémantický význam. Ako dobrý príklad slúži meno a priezvisko. Navyše táto metóda filtruje všetky slová, ktoré nie sú podstatnými menami, tým pádom aj neplnovýznamové slová (spojky, zámená, členy a pod.) a takisto aj slová s malou sémantickou informáciou ako napr. slovesá, ktoré sa zvyknú vyskytovať v akomkoľvek kontexte. Metóda NP chunkingu sa skladá z dvoch krokov. Prvým krokom je tzv. POS tagging. Jedná sa o metódu, ktorá označí dané slovo vo vete jeho gramatickou kategóriou. Rovnako je schopná určiť, či sa jedná o množné alebo jednotné číslo, čas či rod. Všetko závisí od konkrétneho systému značkovania.
V druhom kroku je na danej vete, na ktorej je prevádzaný NP chunking spustený parser prirodzeného jazyka, ktorý má za úlohu zistiť gramatickú štruktúru vety, teda rozdeliť vetu na parsovací strom. [18] Je schopná rozdeliť vetu na skupiny slov, ktoré majú medzi sebou vzťahy a takisto aj odhaliť, ktoré z fráz sú predmetom, resp. prísudkom. Pre vyfiltrovanie slovných spojení podstatných mien stačí nájsť najnižšie uzly NP a v nich vyfiltrovať všetky podstatné mená.
Existuje aj mnoho iných prístupov k NP chunkingu väčšinou využívajúcich trénovacie dáta a následné vytvorenie NP s určitou pravdepodobnosťou. [19]

1.3 Metódy generovania zhlukov slov

1.3.1 Zhluková analýza

Zhluková analýza nie je konkrétny algoritmus, ale je to skupina metód používaných na triedenie dát. Je to najbežnejšia forma učenia bez učiteľa, čo znamená, že v tomto prístupe nie je vyžadovaná ľudská interakcia. [2] Jej úlohou je rozdeľovať dáta podľa nejakej vlastnosti do skupín zvané zhluky. Základným prístupom rozdelenia je, že sa vykonáva tak, aby prvky v jednom zhluku mali voči sebe čo najviac podobné vlastnosti a zhluky medzi sebou mali čo najviac rozdielne vlastnosti. [1]
Na porovnanie s problémom klasifikácie, kde sú skupiny vopred definované a chceme vedieť do ktorej skupiny objekt patrí, zhluková analyzá sa snaží roztriediť objekty podľa vzťahov medzi nimi, a teda skupiny nie sú predom dané. Klasifikácia je forma učenia s učiteľom, na rozdiel od zhlukovej analýzy.
Momentálne je známych mnoho metód a algoritmov, teda dá sa povedať, že tento problém je hlboko preskúmaný. Nedá sa však povedať, ktorý algoritmus zhlukovej analýzy je jednoznačne najlepší. Výber jednotlivých algoritmov veľmi úzko súvisí s dátami, ktoré chceme rozdeliť.
Medzi najznámejšie zhlukovacie algoritmy patrí hierarchické zhlukovanie, k-

means, exceptation-maximization, k-medoids a mnoho dalších. [1]

1.3.2 Latentná sémantická analýza

Latentná sémantická analýza (alebo nazývaná aj latentné sémantické indexovanie) je matematická technika, ktorá analyzuje vzťahy medzi slovami a dokumentami. Je takisto aj metódou pre objavovanie skrytých konceptov vo vektorovom priestore dokumentov [15]. Cieľom metódy nie je popísať koncepty slovne, ale reprezentovať dokumenty a slová jednotným spôsobom, aby sme boli schopní odhaliť sémantickú podobnosť slova k ostatným slovám, dokumentu k ostatným dokumentom a takisto aj dokumentu ku slovu.
Označme si mxn maticu vektorového priestoru dokumentov A. Všimnime si, že ak máme $B = A^T \cdot A$ vzniká matica podobností dokumentu k dokumentu a naopak ak $C = A \cdot A^T$ maticu podobností slova ku slovu. Očividne sú matice B a C štvorcové a symetrické (mxm a nxn). Metóda využíva matematický nástroj zvaný SVD (singular value decomposition), ktorá rozkladá maticu vektorového priestoru dokumentov na tri ďalšie matice. Ak vykonáme SVD s využitím matíc B a C dostávame

$$A = U\Sigma V^T \tag{1.4}$$

kde U je matica eigenvektorov matice B a V je matica eigenvektorov C. Σ je diagonálna matica singulárnych hodnôt druhých mocnín eigenhodnôt matice B.
V programovej reprezentácii sa veľká riedka matica zmenila na tri tabuľky, čo je veľkou výhodou. Navyše singulárne hodnoty na diagonále matice Σ sú zoradené zostupne a niektoré z hodnôt sú blízke nule. Tieto hodnoty môžeme nahradiť nulou, pretože ich absencia nemá signifikantný vplyv na výsledky a tým sme schopní zmenšiť maticu Σ. Takýmto spôsobom sme schopní nájsť určitý počet najviac signifikantých konceptov.

Táto metóda je schopná odhaliť sémantické vzťahy, ktoré inak zostávajú skryté. Skúma nielen vzťahy slov, ktoré sa nachádzajú spolu v dokumentoch, ale aj vzťah medzi slovami, ktoré sa v spoločnom dokumente neobjavili ani raz. Majme dokumenty:
d_1 : roľník sadí
d_2 : na poli sa sadí
d_3 : roľník orie pole
d_4 : orať je činnosť
d_5 : táto činnosť je dôležitá

hľadáme sémanticky najpodobnejšie dokumenty k slovám *orať pole*. Očividne najrelevantnejší dokument je d3, pretože obsahuje obe slová. Trocha menej relevantné dokumenty budú d2 a d4 pretože obsahujú iba jedno z hľadaných slov. Čo sa však týka dokumentov d1 a d5 ako ľudia vieme, že dokument d1 má väčší významový vzťah než d5. A práve takýto nepriamy vzťah

je LSA schopná odhaliť. Slovo *roľník* sa vyskytuje spolu so slovami *orať pole* a zároveň slovo *sadí* sa vyskytuje spolu so slovom *pole*. LSA prideľuje dokumentu d1 väčšiu relevanciu ako d5, pretože sa v ňom nachádzajú dve slová, ktoré sa nachádzali spolu v kontexte so zadanými slovami a v dokumente d5 je iba jedno slovo, ktoré sa nachádzalo spolu v kontexte so zadanými slovami. Keďže SVD je výpočetne časovo náročná, používajú sa aj iné metódy k identifikácii množiny konceptov. Môžeme uviesť napríklad pLSA (probabilistic latent semantic analysis). Táto metóda je založená na pravepodobnostnom prístupe a má výrazne lepšie výsledky ako LSA.

1.3.3 Latentná Dirichletova alokácia

LDA je relatívne nová štatistická metóda navrhnutá D. Bleiom z Kalifornskej Univerzity Berkley v roku 2003. Momentálne sa jedná o najpopulárnejšiu techniku NLP.
LDA je štatistická metóda k automatickému zisťovaniu tém pre danú skupinu dokumentov. Ako vstup berie súbor dokumentov a vopred určený počet tém, ktoré je potreba zistiť. Ako výstup dostávame zoznam slov a ich relevancie k danej téme a zoznam dokumentov a ich relevancie k danej téme [20].
Ako príklad môžu znova slúžiť dokumenty z predchádzajúcej kapitoly. Typickým výstupom z LDA by bolo potom (hodnoty sú ilustráčné):
t_1 : orať 30% pole 30% roľník 15% sadí 15% ...
t_2 : činnosť 30% dôležitá 20% orať 15%...

d_1 : 100% t_1
d_2 : 100% t_1
d_3 : 85% t_1 a 15% t_2
d_4 : 50% t_1 a 50% t_2
d_5 : 10% t_1 a 90% t_2

Téma v poňatí LDA je abstraktný pojem. Nemyslí sa ňou konkrétny termín, je to len spôsob ako reprezentovať vzťahy medzi slovami a dokumentami. Výstupom behu LDA sú teda dve tabuľky (väčšinou označované ako Φ a Θ). Prvá obsahuje podobnosti tém a slov a druhá obsahuje podobnosti dokumentov a tém.
Beh LDA pozostáva z troch krokov. Prvým je voľba počtu tém. Keďže tento počet úzko súvisí s daným súborom dát, nedá sa určiť všeobecný počet. Na určenie počtu môže užívateľ vychádzať z predchádzajúcej analýzy alebo ďalšou bežne používanou možnosťou voľby počtu je metóda pokus omyl. Na začiatku sa určí očakávaný počet tém a jeho počet môže byť ľahko spresnený pohľadom na výsledky behu metódy.
Druhý krok je randomizovaný. LDA priradí všetky slová z dokumentov náhodným témam. Toto priradenie nám už dáva reprezentácie tém a pravdepo-

dobnostné rozdelenie pre všetky témy. Pochopiteľne zatiaľ neodpovedá skutočnosti.
Krok číslo tri je iteratívny. Algoritmus v každom kroku počíta správnosť priradenia dokumentov a slov k témam. Prechádza každé slovo v každom dokumente a upravuje pravdepodobnosť priradenia na základe dvoch kritérií - pravdepodbnosť správneho priradenia témy k dokumentu $P(t|d)$ a pravdepodobnosť správneho priradenia slova k téme $P(w|t)$. Následne preradíme slovo w na novú tému s pravdepodobnosťou $P(t|d) \cdot P(w|t)$. Laicky povedané predpokladáme, že všetky ostatné slová témy sú priradené správne a pomocou nich spresňujeme priradenie akutalizovaného slova. Tento proces opakujeme pre každé slovo v každom dokumente kolekcie určitý počet iterácií, kým nedospejeme k relatívne stabilnému štádiu, kde sa priradenia slov a dokumentov k témam prestávajú meniť.

1.4 Značkovanie zhlukov

Pri probléme značkovania zhlukov si musíme definovať, čo má byť vstupom do algoritmu. Máme niekoľko možností takéhoto vstupu.
Prvou z nich je kolekcia dokumentov. Pokiaľ je tento súbor dokumentov už hierarchicky usporiadaný, sme schopní spracovávať podmnožiny dokumentov, ktoré spadajú do spoločnej sémantickej kategórie a vytvárať už označené zhluky. Pokiaľ táto kolekcia dokumentov nie je usporiadaná, ponúka sa možnosť hierarchického zhlukovania dokumentov a následná generácia kategorizačnej štruktúry [4]. Táto varianta je v praxi nevyužiteľná, avšak použiteľná na tvorbu testovacích alebo trénovacích dát.
Druhou je veľký súbor zhlukov, ktoré vyplynuli z predchádzajúcej analýzy textov. Na tieto zhluky je možné nahliadať ako na dokumenty a vykonávať nad nimi analýzy spomenuté v predchádzajúcej podkapitole (LSA, LDA).
Treťou z možností je tá, že na vstupe je samotný zhluk spolu s mierou relevancie jednotlivých slov. Pochopiteľne samotné dáta obsahujúce zhluk nestačia na jeho ohodnotenie, preto je potrebné použitie externého zdroja vedomostí, a to najčastejšie ontológii. Práca sa ďalej venuje testovaniu metód tretej varianty.

1.4.1 Metódy založené na váhe

Ako prvá naivná metóda môže byť predstavené značkovanie zhluku pomocou slova, ktoré má v zhluku najväčšiu váhu. Čo si môžeme predstaviť pod pojmom váha záleží na konkrétnom modeli, ktorý sme použili ku generovaniu zhluku.
Voľbou slova blízko ťažiska zhluku, teda prevrátenou vzdialenosťou od ťažiska volíme slovo, ktoré má najväčšiu sémantickú podobnosť s ostatnými slovami. Tieto slová častokrát nemusia vystihovať zhluk ako celok. Ako príklad

môže slúžiť zhluková analýza textov pojednávajúcich o rôznych zvieratách, ale hlavne o psoch. Veľmi ľahko sa potom môže stať, že zhluk slov so sémantickým významom zvieratá bude mať najbližšie pri ťažisku slovo pes, ktoré však význam skupiny ako celku neodhalí.

1.4.2 Metódy používajúce externé zdroje vedomostí

Jeden z možných zdrojov takýchto informácií je ontológia. Ontológia je zdroj vedomostí obsahujúci definíciu pojmov a definíciu vzťahov medzi pojmami. [6] Je to orientovaný acyklický graf, ktorý má dobre definovanú child-parent štruktúru. Obsahuje skupiny slov, ktoré sú už zaradené do spoločnej pomenovanej kategórie. Rovnako má tento graf vyznačený jeden uzol ako koreň ontológie.
Takéto ontológie sú už vytvorené a na internete ľahko dostupné. Vytvárané sú plne pomocou ľudskej interakcie, tým pádom v nich neexistuje šum a slová sú takmer 100% dobre priradené.
Takéto ontológie obsahujú hyperonymické vzťahy. Hyperonymum je nadradený pojem. Podľa [8] je hyperonymum slovo v širšom mienení určujúce kategóriu slova s viac špecifickým významom. Na rozdiel od synonymie hyperonymia nie je symetrická. Uvediem príklad: slovo jablko má hyperonymum slovo ovocie, avšak nie každé ovocie je jablko. Zatiaľ čo synonymum slova jablko je plod jablone, a každy plod jablone je jablko.

Podobným vzťahom je aj vzťah celok-časť, ktorý sa nazýva meronymický. Meronymia má blízky vzťah k hyperonymii. Podľa [8] výraz značí časť niečoho, ktorá odkazuje na celok. Ako príklad môže slúžiť: slovo motor má ako meronymum slovo auto, avšak v žiadnom prípade sa nedá povedať, že slovo auto je zároveň aj motor. Pri hyperonymii možeme s istotou povedať, že ovocie je zároveň aj jablko, a jablko je ovocie.

Jedným z príkladov takejto voľne dostupnej ontológie je aj slovník WordNet, ktorý obsahuje hyperonymické hierarchie slov a aj meronymické vzťahy. Tento slovník je často aktualizovaný a momentálne je vydávaný vo verzii 3.0.

Riešenie problému značkovania za pomoci ontológie sa skladá z troch častí. Prvou z nich je voľba kandidátov. Tá v tejto metóde závisí na strome hyperoným daného slova tzv. hyperonymickej hierarchii. Ako kandidátov na ohodnotenie zhluku volíme všetky prvky z hyperonymickej hierarchie každého slova zhluku a všetkých jeho významov.
Keďže berieme do úvahy aj hyperonymické hierarchie všetkých homoným slova, riešime tým aj problém nejednoznačnosti slov. Tento prístup však znižuje presnosť metódy, pretože za kandidátov volíme aj mnoho nesprávnych slov t.j. používame hyperonymické hierarchie významov slova, ktoré nemajú sémantický vzťah s celým zhlukom. Pokiaľ sú vstupom algoritmu iba jedno-

duché textové reprezentácie slov, tento problém je neriešitelný. Pokiaľ by sme využili nejakú z metód WSD (word sense disambiguation) [21] pred krokom generovania zhlukov, algoritmus by dosahoval kvalitnejšie výsledky.
Druhým krokom je ohodnotenie relevancie daného kandidáta. V metóde používanej v tejto práci podobne ako v [5] sa používa hodnotenie veľmi podobné miere relevantnosti tfidf. Skladá sa z dvoch zložiek, ktoroými je podobne ako v tfidf normalizovaná miera frekvencie výskytu slova. Namiesto invertovanej dokumentovej frekvencie ako miera dôležitosti slova sa však berie miera sémantickej informácie meraná topologickou podobnosťou spomínanou vyššie. Pokiaľ si označíme kandidáta C, mieru frekvencie $Tf(C)$ a množstvo sématickej informácie $I(C)$, výsledné skóre ohodnotenia kandidáta sa dá vypočítať takto.

$$score(C) = Tf(C) \cdot I(C) \tag{1.5}$$

Aby sme boli schopní vyjadriť percentuálnu podobnosť kandidáta k zhluku, obe miery musia byť položené v intervale <0,1>. Aby miera frekvencie výskytu kandidáta spadala do tohto intervalu je počet výskytov kandidáta v hierarchiách potrebné normalizovať počtom všetkých prvkov zhluku (a všetkých jeho významov), t.j. počtom všetkých používaných hyperonymických stromov. Avšak je potrebné si uvedomiť, že v hyperonymickej hierarchii môže existovať viac rôznych ciest ku koreňu. Je to pochopiteľne preto, že ontológia je orientovaný graf. V tom prípade aby sme zasadili túto mieru frekvencie do intervalu <0,1> je nutnosťou počítať každého kandidáta pre každý strom iba raz.
Čo sa týka druhej zložky rovnice počítajúcej skóre kandidáta, je potrebné zaviesť si jasné pravidlo určujúce mieru informácie. Budeme vychádzať z tvrdenia, že čím je slovo viac všeobecné (obsahuje menej sémantickej informácie) nachádza sa bližšie pri koreni ontológie. Najvodnejší kandidát by mal mať teda najväčšiu vzdialenosť od koreňa ontológie. V [5] sa na výpočet miery množstva informácie používa sigmoida normalizovaná konštantami -0.5 a 2 tak, aby spadala do intervalu <0,1>. Pričom parameter d_C je vzdialenosť kandidáta od koreňa ontológie a konštanta c riadi strmosť funkcie. Ako najvhodnejšia hodnota konštanty c sa ukázala hodnota 0.125 vo vzťahu s ontológiou WordNet [5].

$$I(C) = 2 \cdot \left(\frac{1}{1 + e^{-d_C \cdot c}} - 0.5 \right) \tag{1.6}$$

Treťou fázou algoritmu je voľba n-prvých kandidátov s najvačším dosiahnutým ohodnotením. Voľba tohto počtu závisí od toho, čo užívateľ potrebuje.

Kapitola 2

Návrh

2.1 Návrh algoritmu využívajúceho ontológiu

Algoritmus má dva vstupy, prvým je zhluk, ktorý chceme označkovať a druhým vstupom je ontológia obsahujúca hyperonymá pre daný jazyk zhluku. Algoritmus využívajúci ontológiu spočíva v dvoch krokoch, ako to vidno na obrázku 2.1.
Prvým krokom je voľba kandidátov. Pre textovú reprezentáciu slova v ontológii nájdeme všetky jeho možné významy a pre všetky významy spustíme funkciu *selectCandidates* s počiatočným parametrom $depth = 0$. Táto rekurzívna funkcia pridá do tabuľky kandidátov celú hyperonymickú hierarchiu pre dané slovo. Funkcia rovnako meria vzdialenosť od koreňa hierarchie.
Pre beh funkcie predpokladáme dve štruktúry, a to: množinu *openCandidates* kde si udržujeme identifikačné čísla už navštívených uzlov, aby bolo zabezpečené, že k žiadnemu uzlu z hierarchie nepristupujeme dvakrát, keďže v ontológii môže existovať viac ciest ku koreňu, ako bolo spomenuté vyššie.
Druhou štruktúrou je tabuľka kandidátov spolu so stĺpcami, kde sa ukladajú vzdialenosti kandidáta od koreňa ontológie a počtami jeho výskytov.

```
function selectCandidates(word, depth)
    openCandidates.add(word)
    H ← getHypernyms(word)
    for i ∈ {1, ..., |H|} do
        if openCandidates.contains(H_i) then
            depth ← selectCandidates(H_i, depth)
        end if
    end for
    addToCandidatesTable(word, depth)
     return ++depth
end function
```

Parametrom word do funkcie *selectCandidates* je identifikačné číslo významu slova a všetkých jeho synoným tzv. synset. Funkcia getHypernyms vracia ako návratovú hodnotu všetky hyperonymá zadaného synsetu vo forme poľa synsetov. Nie je teda potreba riešiť synonymiu ani ďalšie významy získaných hyperoným v množine H.
Funkcia *addToCandidatesTable* kontroluje výskyt daného synsetu v tabuľke kandidátov. Pokiaľ sa v tabuľke synset nachádza, zväčší počet jeho výskytov o jedna a uloží do tabuľky vzdialenosť od koreňa *depth*.
Druhá fáza algoritmu spočíva vo vyhodnotení kandidátov. Pre každý prvok tabuľky kandidátov sa vykoná ohodnocujúca funkcia, ktorej vzorec bol spomenutý v kapitole 1.4.2 (vzorce 1.5 a 1.6).
Dôležitým krokom je pri návrhu algoritmu nájdenie takých kandidátov, ktorí majú v sebe zapuzdrenú aspoň nejakú sémantickú informáciu. Príliš všeobecné slová (entity, object, artifact a pod.) nehovoria nič o sémantickej kategórii zhluku, avšak dosahujú vysokú frekvenciu, pretože každá hyperonymická hierarchia končí v týchto všeobecných slovách. Pozorovaním štruktúry WordNetu som zvolil, že sa vyhodnocujú len kandidáti s hĺbkou vyššou ako 5. Dramaticky to zvýšilo úspešnosť algoritmu viz. kapitola Testovanie.
Kandidát je po vyhodnotení pridaný do prioritnej fronty. Po vyhodnotení všetkých kandidátov sa z prioritnej fronty vyberie n-prvých kandidátov podľa nastavenia algoritmu.

2.2 Návrh metódy generujúcej testovacie dáta

Cieľom tejto metódy je generovať zhluky, o ktorých vieme, že majú správne pridelenú tému. Metóda spočíva vo využití hierarchicky usporiadanej množiny dokumentov. Podstata metódy tkvie v rozdeľovaní množiny dokumentov na podmnožiny patriace do spoločného tématického celku.
Kategorizačná štruktúra takejto množiny je tvorená užívateľmi a dokumenty sú jednotlivým uzlom priraďované užívateľom. Preto je vysoko pravdepodobné, že zvolené dokumenty sú si naozaj sémanticky podobné a patria do spoločného tématického celku.
Príkladom takéhoto korpusu dokumentov je Wikipédia. Štruktúra kategórií Wikipédie sa skladá z troch častí, kde prvou je graf kategórií reprezentovaný zoznamom kategórií a k nim prilehajúcim podkategóriám. Každý vrchol grafu kategorizačnej štruktúry obsahuje určitý počet dokumentov, teda druhou časťou je zoznam kategórií a im priliehajúce názvy článkov. Treťou časťou je zoznam názvov článkov a im priliehajúci obsah.
Metóda prebieha v nasledujúcich krokoch. Prvým je získanie názvov článkov pre danú kategóriu. Keďže kategorizačná štruktúra Wikipédie nie je strom, ale graf obsahujúci spätné hrany, pri rekurzívnom prehľadávaní môže nastať cyklenie algoritmu, a preto je nevhodné. Vhodné je však prehľadávanie do šírky, pri ktorom sú ignorované spätné hrany.

Po odobratí vrcholu z fronty využívanej BFS je každý dokument, ktorý vrchol obsahoval, pridaný do množiny (články sa často opakujú aj pre podkategórie). Beh BFS je ukončený v momente, keď veľkosť množiny dosiahne maximálny povolený počet dokumentov. V tejto fáze sa odstráni aj prípadna reflexia kategorizačnej štruktúry.
Jedným z problémov tejto metódy je zvoliť taký počet podkategórii pre danú kategóriu, aby výsledná skupina dokumentov bola dostatočne veľká a vzniknuté zhluky z nej analyzované boli kvalitné a zároveň aby dosahovala výpočetne zvládateľnú veľkosť. Bohužiaľ nedá sa presne povedať pri akom minimálnom počte dokumentov dosiahnu metódy pre generovanie tém kvalitné výsledky, pretože kvalita vzniknutých skupín veľmi úzko súvisí s dĺžkami dokumentov a slovami, ktoré obsahujú. Opačný problém nastáva s určením maximálneho počtu dokumentov. Jedno z riešení je použiť metódu pokus-omyl, spúšťať algoritmus na rôznych minimálnych a maximálnych hodnotách a sledovať výsledky a dobu behu algoritmu. V implementácii algoritmu je použitá hodnota maxima 1200 článkov a doba behu výpočtu jedného zhluku dosahuje v priemere 5 až 10 sekúnd (Intel® Core™ i5 CPU 2.67GHzx4).
Druhým krokom metódy je využitie algoritmov na hľadanie tém, konkrétne LDA. Zvolí sa pevná veľkosť zhluku, a to v implementácii konkrétne 20 tém a pre daný korpus následne vygeneruje zhluk. Kedže každý z uzlov kategorizačnej štruktúry má priradený unikátny názov, výstupom z algoritmu je ohodnotený zhluk tém.

2.3 Návrh programových tried ohodnocovacieho algoritmu

Pri programovom riešení je potrebné vyriešiť tri úlohy: načítanie vygenerovaných dát z disku, vytvorenie triedy zaobaľujúcej algoritmus značkovania zhlukov a triedy zaobaľujúcej spôsob vyhodnotenia presnosti algoritmu.

2.3.1 Trieda ClusterLabeling

Ako prvú vec pri návrhu triedy ClusterLabeling je vhodné zvoliť si formu vstupných dát. Vstupné dáta su reprezentované zoznamom reťazcov a ich váhových koeficientov. Ako formát je zvolené CSV. Jeden zhluk slov je uložený v jednom súbore, pričom názvu súboru je automatický priradená tématická značka vyplývajúca z predchádzajúcej analýzy (nasledujúca tabuľka).

The Beatles

Slovo	**Váha**
beatles	;0.02621769438903131

song ;0.026173265149260033
album ;0.024484954037951458
band ;0.006979833568067816
john lennon ;0.006224536491956085
recording ;0.005691385614700746
lennon ;0.00533595169653052
paul mccartney ;0.004758371579503902

Jedným z argumentov vstupu do algoritmu je cesta ku zložke, kde sa jednotlivé csv tabuľky obsahujúce zhluky nachádzajú. Trieda prechádza všetky súbory v zadanej zložke. Pokiaľ csv súbor obsahuje iba jeden stĺpec alebo druhý stĺpec nie je parsovateľný na desatinné číslo, do úvahy sa berie váha pre dané slovo 1.0.

2.3.2 Trieda ClusterLabeler

Táto trieda zaobaľuje algoritmus značkovania zhlukov. Ako parameter konštruktora sa zadáva už slovník WordNet, ktorý bol nahraný do pamäte v triede volajúcej túto triedu. Nahrávanie slovníka do pamäte musí byť vykonané už pred konštrukciou tejto triedy, pretože ho využíva aj nasledujúca trieda Evaluation a slovník by musel byť nahrávaný do pamäte dvakrát. Parametrom konštruktora je aj hodnota konštatny *c* spomenutej vyššie.
Trieda obsahuje 5 triednych premenných, a to hodnoty konštanty c (Double), odkaz na slovník uložený v pamäti, počet spracovávaných hyperonymických stromov, množinu (Set<SynsetID>) pre ukladanie otvorených kandidátov a mapu kandidátov pre ohodnotenie zhluku (Map<SynsetID, Candidate>).
Trieda Candidate je privátnou triedou triedy ClusterLabeler. Obsahuje vzdialenosť od koreňa, váhový koeficient pôvodného slova zhluku, z ktorého bola hyperonymická hierarchia generovaná a počet výskytov. Pre dalšie experimenty trieda ukladá aj inverznú hĺbku (vzdialenosť od listu). Trieda obsahuje metódu evaluate(), ktorá vracia desatinnú hodnotu ako výsledok vzorca 1.5.

2.3.3 Trieda Evaluation

Trieda Evaluation je trieda zaobaľujúca algoritmy pre výpočet presnosti algoritmu. Tieto algoritmy budú dalej spomenuté v kapitole 4. Doležité je si uvedomiť, prečo je vyhodnotenie vykonávané priamo pri ohodnocovaní zhlukov. Pokiaľ by sme ukladali medzivýsledky na disk a pokúšali sa vyhodnotiť ich kvalitu v inom programe, mali by sme iba textové reprezentácie kandidátov. Opäť by sme museli riešiť problém viacerých významov kandidátov, čo by sťažovalo vyhodnocovanie. Pokiaľ však výpočty prebehnú v rámci jednej inštancie triedy ClusterLabeling, dovoľuje nám to posielať do triedy evaluation

Synsety a problém nejednoznačnosti slov je vyriešený.
Trieda obsahuje jednu triednu premennú, a to odkaz na nahratý slovník v pamäti. Rovnako obsahuje aj privátnu metódu populateNgram, ktorá bude spomenutá neskôr v kapitole 4.

2.3.4 Trieda Pair

Trieda Pair zabaľuje pár Synset a jej váhový koeficient, a teda uľahčuje implementáciu.

2.4 Návrh programových tried generátora testovacích dát

Pri programovom riešení generovania testovacích dát je potrebné vyriešiť viac úloh. Je potrebné navrhnúť triedy schopné uspokojivo načítať súbory kategorizačnej štruktúry do pamate, obsah vrcholov v grafe kategorizačnej štruktúry a vyriešiť problém nahrávania textov dokumentov do pamäte v konštantnom čase. Ďalej je potrebné navrhnúť triedu zaobaľujúcu beh algoritmu BFS, triedu zaobaľujúcu beh metódy LDA a triedu zabezpečujúcu spôsob výstupu výsledkov z algoritmu.

2.4.1 Trieda CreateLabeledClusters

Táto trieda zabezpečuje správne nahranie argumentov programu. Vstupom do programu sú cesty k CSV súborom grafu kategórií (categoriesFile), obsahu kategórií (articleCategoriesFile), predspracovaného zoznamu dokumentov (preprocessedDocumentsFile), cesty k zložke, kde majú byť výsledky ukladané, separátor polí formátu CSV. Posledným argumentom je číslo udávajúce, od ktorej kategórie v zozname kategórií sa má začať výpočet. Tento argument sa udáva z dôvodu možného prerušenia behu algoritmu. O čo konkrétne ide môžeme ukázať na príklade. Spustili sme algoritmus pre generovanie skupín od indexu 0. Po určitom čase sa počítač prehrial a reštartoval, tým pádom sa algoritmus prerušil. Po opätovnom spustení počítača zistíme, koľko zhlukov slov algoritmus vytvoril a môžeme ho opätovne spustiť s daným číslom, a to samozrejme za predpokladu behu algoritmu na rovnakých vstupoch.

2.4.2 Trieda GraphReader

Trieda GraphReader zabezpečuje nahrávanie kategorizačnej štruktúry do pamäte. Ako parametre konštruktora berie separátor polí CSV a cestu k da-

nému súboru. Obsahuje tri triedne premenné, a to categoryDictionary (HashMap<String, Integer>), categoryEdges (ArrayList<ArrayList<Integer> >) a categoryNames (ArrayList<String>). Štruktúra categoryNames obsahuje zoznam textových reprezentácií názvov kategórií. Jednotlivé kategórie identifikuje poradím v poli. CategoryEdges obsahuje zoznamy identifikačných čísel všetkých podkategórii danej kategórie.
Metóda run() spustí nahrávanie, ktoré prebieha nasledovne: každý riadok vstupného súboru rozdeľuje pomocou štandardnej metódy split() a na údaje volá privátnu metódu addPair(). V tejto privátnej metóde sa práve používa tretia dôležitá štruktúra, a to categoryDictionary. Táto mapa spája textovú reprezentáciu s identifikačným číslom danej kategórie. V metóde addPair() kontrolujeme, či sa dané reťazce už nachádzajú v slovníku a ak nie, pridáme ich tam a zároveň vytvoríme nový údaj v zozname podkategórií.
Po prejdení všetkých riadkov súboru sa následne spustí privátna metóda convertDictionaries(), ktorá transformuje mapu categoryDictionary na zoznam názvov kategórií, indexovaný poradím.

2.4.3 Trieda GraphNodeContentReader

Táto trieda zabezpečuje spôsob nahrávania obsahu vrcholov grafu do štruktúry. Formát je veľmi podobný ako v predchádzajúcom prípade, teda zoznam párov kategória a názov článku.
Spôsob nahrávania do pamäte je veľmi podobný ako v predchádzajúcom prípade. Pribúdajú nám tri triedne premenné, a to zoznam mien článkov articleNames (ArrayList<String>), zoznam článkov pre dané kategórie articleEdges (ArrayList<ArrayList<Integer> >) a slovník názvov kategórií articleDictionary (HashMap<String, Integer>). Postup nahrávania je veľmi podobný ako v predchádzajúcom prípade, metóda run() prechádza CSV súbor, parsuje hodnoty a potom v metóde addArticle() sa kontroluje, či sa daný článok nachádza v slovníku. Pokiaľ nie, pridá sa do slovníka. Následne po načítaní všetkých riadkov súboru sa slovník konvertuje na zoznam názvov článkov articleNames.

2.4.4 Trieda ArticlesReader

Pri implementácii tejto metódy je jedným z hlavných problémov vyriešiť nahrávanie článkov do pamäte v konštantnom čase vzhľadom na to, že súbor s predspracovanými článkami má cca 1.3Gb a nahrávať celý súbor do pamäte neprichádza do úvahy. Navyše už predchádzajúce štruktúry, ktoré vyžadujú nahranie do pamäte sú pamäťovo náročné.
Tento problém rieši trieda ArticlesReader. Štruktúra vstupného súboru je opäť CSV s dvoma stĺpcami, pričom prvý stĺpec je názov článku a druhý text. Táto trieda prechádza celý súbor textov článkov a hľadá newline znaky. Využíva

triedu SeekableByteChannel (java.nio.channel), ktorá načítava ntice bytov. Na začiatku čítania riadku odhalí názov článku oddelený separátorom CSV hodnôt, ten uloží do mapy offsets (HasmhMap<String, Long>) ako kľúč spolu s hodnotou bytového offsetu zalomenia separátora polí od začiatku súboru.
Táto metóda sa ukázala ako časovo najefektívnejšia v porovnaní s použitím metód getLine a následného použitia metódy split() alebo načítavania po znakoch tried BufferedReader alebo Scanner. Využíva takisto triedu ByteBuffer (java.nio.ByteBuffer) s pevnou veľkosťou bloku 8192 bytov. Ako problém môže nastať deformácia non ascii znakov, avšak to je riešené konverziou bytového bloku na UTF-8 kódovanie, ktoré odpovedá kódovaniu vstupných súborov.
Verejná metóda getArticleText(String name) potom hľadá v HashMape daný názov dokumentu a príslušný byte offset začiatku textu dokumentu a pomocou triedy RandomAccessFile (java.io.RandomAccessFile) načítava text, kým nenarazí na newline.

2.4.5 Trieda GraphDivider

Trieda GraphDivider zaobaľuje beh BFS algorimu. Ako parametre konštruktora vstupujú štruktúry vytvorené v predchádzajúcich krokoch a to categoryNames, articleNames, categoryEdges, articleEdges spolu s hodnotou maximálneho počtu získavaných dokumentov.
V konštruktore sa potom vytvorí pole bytov veľkosti zoznamu kategórii, kde sa ukladá, či je daný uzol fresh (0), open (1) alebo closed (2). Jediná verejná metóda je getArticleSubset(int node) berúca ako parameter identifikačné číslo danej kategórie, pre ktorú hľadáme dokumenty a dokumenty jej podkategórii. Pre ňu je spúšťaný algoritmus BFS ako bolo popísané vyššie.

2.4.6 Trieda SelectTopWords

Výstupom knižnice metódy LDA sú dve tabuľky phi a theta. Prvá tabuľka obsahuje hodnoty podobností slov a jednotlivých tém a druhá tabuľka obsahuje podobnosti dokumentov a tém. Trieda SelectTopWords zabezpečuje výstup aký je vyžadovaný pre beh programu ClusterLabeling. Ako parametre konštruktora berie trieda tabuľku phi (double[][]), zoznam slov (ArrayList<String>), maximálny počet najpodobnejších slov (int) a inštanciu triedy BufferedWriter pre súbor kde zapisujeme výsledky.
Vo verejnej metóde printResulToFile() máme pevne určený počet tém (veľkosť výsledného zhluku), čo je aj veĺkosť stĺpcov tabuľky. Z tabuľky vyberieme slová, ktoré majú najvyššie hodnoty podobnosti a výsledný zoznam potom zoradíme a uložíme do súboru. Ako názov súboru slúži hodnota zoznamu mien kategórií categoryNames.

Kapitola 3

Implementácia

3.1 Voľba ontológie

Ako vhodnú ontológiu som zvolil slovník WordNet. Tento slovník spája slová na množiny synoným a obsahuje množstvo sémantických a lexikálnych vzťahov. WordNet takisto obsahuje aj krátke definície pre každé slovo alebo jeho použitie vo vete, ale to v tejto implementácii nie je dôležité.
Slovník WordNet bol vytvorený Princetonskou univerzitou v roku 1985 v Laboratóriu kognitívnych vied. [9]. Tento slovník obsahuje slová anglického jazyka, ale od jeho vzniku vzniklo mnoho projektov v iných jazykoch vrátane češtiny alebo slovenčiny. Pre ostatné databázy sa používa názov sémantická sieť typu WordNet.
Takýchto sémantických sietí existuje viacero. Za spomenutie stojí určite BabelNet [22], ktorá je výhodná práve svojou obsažnosťou. Obsahuje 70 miliónov sémantických vzťahov a 3 milióny synsetov. Iným dobrým príkladom sémantickej siete je Open Multilingual WordNet [23], ktorá spája väčšinu kvalitných jazykových mutácii. Spája sémantické siete 20 rôznych jazykov a umožnuje tiež online vyhľadávanie a stiahnutie dát.
Medzi dalšie známe sémantické siete patrí Mimida Project, MultiWordNet a sieť vytvorená organizáciou Global WordNet Association. Implementácia algoritmu sa orientuje iba na anglický jazyk, čo robí použitie viacjazyčných WordNetov zbytočným. Sieť BabelNet je generovaná automaticky spájaním rôznych iných databáz, zatiaľ čo WordNet je vytváraný ručne lingvistami. Z toho dôvodu bolo očakávané, že bude dosahovať najlepšie výsledky.
Licencia WordNetu umožňuje použitie databázy vo výskume alebo na komerčné účely pod podmienkou použitia správnej citácie. [10] [11] Iné odvodeniny WordNetu majú rôzne licencie, niektoré sú dostupné voľne, niektoré sú pod uzavretou licenciou.
Ďalším krokom bola voľba knižnice pre WordNet API. Užitočným sa ukázal byť článok [28], ktorý porovnáva Java knižnice pre prístup k WordNetu. Medzi najznámejšie patrí extJWNL, JWordNet, WNJN a mnohé dalšie. V

implementácii je použitá knižnica MIT Java WordNet Interface (JWI). Je to malá, rýchla a užitočná knižnica, podporujúca najnovšiu verziu WordNetu. Umožňuje nahranie slovníka do pamäte pre rýchlejšie vyhľadávanie a je tiež relatívne dobre dokumentovaná.

3.2 Príprava korpusu dokumentov

Wikipédia ponúka kópie svojho obsahu zadarmo na internete. Jej obsah je licencovaný pod licenciou Creative Commons. [12] Umožnuje kopírovanie a distribúciu obsahu a takisto aj jeho modifikáciu.
Wikipédia, ktorá je neustále aktualizovaná a jej články sa menia, ponúka aktuálne zálohy svojho obsahu, zálohované približne každý mesiac na svojich stránkach. Aktuálne zálohy nájdeme na [13]. Tieto zálohy sú vygenerované z MySql databázy a sú vo formáte sql (tabuľka stránok a tabuľka kategórií) alebo vo formáte XML (text článkov).
Dumpy databázy Wikipédie vo frome sql sú obrovské a plné neštrukturovaných dát. Takisto obsahujú aj informácie o užívateľoch, komentáre, výhonky, články o samotnej Wikipédii (kategórie Wikiprojects a pod.). To sťažuje prácu s nimi a dalšie výpočty.
Pre potreby NLP a analýz článkov Wikipédie vznikol projekt zvaný DBpedia. Za projektom stoja dve univerzity: Slobodná berlínska univerzita a Leipzigova univerzita [24] v spolupráci so spoločnosťou OpenLink Software. Cieľom tohto projektu je extrahovať štrukturované informácie z Wikipédie za účelom uľahčiť analýzy nad nimi vykonávané a zároveň umožniť, aby boli tieto informácie ľahko prístupné z webu.
Máme niekoľko možností ako získavať informácie z DBpedie. Získavať články priamo z http alebo cez sql je nevhodné, pretože dostaneme neštrukturované dáta a ich kvalita nie je zaručená ani obsahovo, ani syntakticky, navyše pri opätovnom spracovaní článku by sme ho museli znovu sťahovať z internetu. Výhodou DBpedie je, že ponúka štrukturované dáta na svojej stránke. Najvýhodnejší postup je stiahnutie týchto dát na disk a ich následné konvertovanie do tvaru, ktorý je najvhodnejší pre túto implementáciu.
Pre našu implementáciu sú dôležité súbory katégorií, článkov a kategórií a súbor abstraktov článkov (prvého odstavca). Tieto súbory sú v SKOS formáte podľa štandardu W3C. Trieda pre generovanie testovacích dát je navrhnutá tak, že vyžaduje formát CSV, pretože to uľahčuje implementáciu.
Stiahnutie, dekompresiu a následnu konverziu do formátu CSV zabezpečuje bashovský skript *exe/download−dbpedia.sh*. Prvá časť skritpu pomocou wget a bunzip2 stiahne a dekomresuje anglické verzie potrebných súborov. Druhá časť skritpu s využitím nástroja awk odstráni z SKOS súborov irelevantné informácie, odstráni komentáre (riadky začínajúce s #) a prázdne riadky. Tento skript zabezpečuje aj to, že vo všetkých názvoch kategórii, článkov a abs-

traktov dokumentov je každá bodkočiarka nahradená čiarkou z dôvodu, že bodkočiarka sa v implementácii používa ako CSV separátor polí a je teda zaručené, že všetky vygenerované CSV súbory budú mať presne 2 stĺpce.
Navyše v súbore *skos_categories_en.ttl* je viac typov riadkov. Sú označené druhým stĺpcom a nachádzajú sa tam 4 typy údajov (prefLabel, broader, type, related). Pre túto implementáciu je potrebný iba jeden údaj a to broader, teda podkategória. Tento skript v poslednej časti tiež vyhadzuje ostatné nepotrebné riadky.

3.2.1 Predspracovanie abstraktov článkov

Dôležitým krokom je predspracovanie abstraktov angl. preprocessing. Tento krok musí byť vykonaný osobitne a este pred spustením behu algoritmu generujúceho testovacie dáta, keďže sa niektoré abstrakty spracúvajú viackrát. Vykonávať ho pred každým behom LDA na skupine dokumentov by bolo zbytočne časovo náročné.
Knižníc na platforme Java, ktoré sú schopné riešiť NLP problémy je veľké množstvo. Medzi najznámejšie patrí LingPipe, openNLP, Apache UIMA, GATE a FrameNet.
Pre potreby tejto implementácie som zvolil knižnicu Stanford CoreNLP [25]. Obsahuje širokú škálu nástrojov spracovania prirodzeného jazyka. Táto voľba bola vykonaná z dôvodu, že knižnica obsahuje kvalitný POS tagger a gramatický parser viet. Jej použitie je veľmi jednoduché. Anotátory je možné spustiť na čistý text napríklad len pomocou dvoch riadkov kódu.
Postup predspracovania je nasledovný. Ako prvá prebieha tokenizácia textu a následne, sentence tokenizácia. Po tomto kroku prebehne POS tagovanie. Priradí každému slovu gramatickú kategóriu, do ktorej patrí. Následne prebehne gramatické parsovanie, ktoré pre každú vetu vytvorí strom. Posledným spúšťaným anotátorom je lematizácia. Stemming je nevhodný, pretože deformuje slová, čo by stažovalo vyhľadávanie v slovníku WordNet.

3.2.2 NP chunking dokumentov

Pre každú vetu jedného dokumentu stanfordský parser vygeneruje syntaktický strom. Príklad behu knižnice môžeme vidieť na online deme [29].
Stanfordská NLP knižnica má implementované veľmi efektívne a užitočne stromové regulárne výrazy pre filtrovanie uzlov takéhoto syntaktického stromu. Popis týchto stromových regulárnych výrazov je veľmi dobre vysvetlený v dokumentácii knižnice [26].
Pre riešenie problému nájdenia najmenších neprekrývajúcich sa slovných spojení podstatných mien je potrebné hľadať v syntaktickom strome NP frázy. Keďže chceme rozdeliť frázy na čo najmenšie časti, musíme hľadať také uzly,

ktoré sú najďalej od koreňa stromu, resp. najbližšie pri listoch. Na získanie práve takýchto uzlov majú tieto regulárne výrazy vytvorený operátor $< \#$. Keďže chceme filtrovať uzly iba nad podstatnými menami, v implementácii sa používa tento regulárny výraz:

$$NP(< \#NNP| < \#NNS| < \#NN| < \#NNPS)$$

Tento výraz vyfiltruje všetky NP uzly, ktoré sú prvým spoločným predchodcom hľadaných NP fráz. V druhom kroku zo získaných častí stromu vyfiltrujeme všetky odpovedajúce listy výrazom:

$$NNP|NNS|NN|NNPS.$$

Kedže stanfordská knižnica neumožnuje zo získaných listov získať lemmy, pretože tie sa získavajú zadaním pôvodného tokenu a nie uzlu syntaktického stromu, v implementácii sa využíva mapa, ktorá obsahuje ako kľúče listy daného stromu a ako hodnoty poradie slova vo vete. Nasledné vyfiltrované frázy sú lemmatizované pomocou týchto uložených hodnôt a pôvodnej tokenizovanej vety.

Vysvetlenie označenia použitých POS tagov

Značka	**Význam**
NP	Noun phrase
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural

3.2.3 Implementácia LDA

Na implementáciu LDA je využitá knižnica JGibbLDA [27], ktorú vytvorili Xuan-Hieu Phan z Tohokskej univerzity a Cam-Tu Nguyen z Vietnamskej národnej univerzity v Hanoi. Využíva Gibbsove samplovanie a je vlastne prepisom knižnice GibbLDA++, ktorá implementuje LDA v jazyku C++.
Táto knižnica je vydaná pod GNU Pulic Licence a umožnuje modifikáciu a využitie pre akýkoľvek účel, dokonca aj pre účel použitia v komerčnom produkte.
Knižnica umožňuje iba analýzu a nahrávanie dokumentov priamo z disku. To je pri tejto implementácii nevhodné, keďže už tokenizované dokumenty potrebujeme posielať do knižnice priamo. Preto sú zdrojové kódy tried knižnice Model a Estimator zlúčené do jednej triedy (v mojej implementácii je to trieda LDA). Vynechanie nahrávania vstupných dát a naopak ich opätovného ukladania na

disk umožnuje z knižnice vynechať triedy Constants, LDA, Pair, Conversion a Inferencer a LDACmdOption. Triedy Dictionary, Document, LDADataset sa využívajú v implementácii ako pôvodné, prípadne s malými zmenami. Trieda Inferencer je v implementácii nahradená vlastnou triedou SelectTopWords.

Kapitola 4

Testovanie

4.1 Testovacia metóda

Drvivá väčšina názvov kategórii Wikipédie tvoria slovné spojenia. Jedným z problémov pri testovaní bolo určenie percentuálnej úspešnosti určenia zhluku. Tento problém sa rieši rozdelením slovného spojenia ohodnotenia zhluku na menšie časti nasledujúcou metódou. Ak má kategória Wikipédie názov Adverse_effects_of_drugs, následne sa v slovníku WordNet vyhľadajú všetky slovné spojenia, ktoré spojenie môže generovať a to: Adverse, effects, of, drugs, adverse_effects, efects_of, of_drugs, atď. Pochopiteľne iba niekoľko zo získaných slovných spojení sa nachádza v slovníku WordNet. Pre tieto slová a slovné spojenia sú nájdené a uložené synsety všetkých ich významov do dvojrozmerného poľa. Pokiaľ sa ani jedno zo získaných slovných spojení v slovníku nenachádza, algoritmus vôbec neberie testovaný zhluk do úvahy.
Následne algoritmus pre každé zvolené ohodnotenie zhluku prechádza všetky získané synsety pre dané získané slovné spojenie a pokiaľ sa ohodnotenia zhluku zhodujú aspoň v jednom význame, počet zhôd sa zvýši o 1. Algoritmus skontroluje, v koľkých získaných synsetoch sa ohodnotenia zhluku zhodli a vypočíta percentuálnu úspešnosť ohodnotenia ako pomer všetkých nájdených slovných spojení k počtu ohodnotení zhluku, ktoré sa zhodli aspoň v jednom význame daného slovného spojenia. Metóda *split*() rozdelí vstupný reťazec na pole reťazcov pôvodne oddelené podčiarkovníkom. Metóda *append*() pripojí reťazec na koniec reťazca ngram. Metóda *clear*() nahradí obsah reťazca prázdnym reťazcom. Premenná results je výsledný zoznam získaných reťazcov.

```
function populateNgram(String: input)
    S ← input.split(_)
    for i ∈ {2, ..., |S|} do
        for j ∈ {1, ..., |S| − i} do
            for k ∈ {1, ..., i − 1} do
                ngram.append(S[j + k] + _)
```

```
            end for
            ngram.append(str[j + i − 1])
            result.add(ngram)
            ngram.clear()
        end for
    end for
    result.add(S)
return result
end function
```

4.2 Výsledky testovania

Algoritmus ohodnocovania zhlukov za pomoci ontológie bol testovaný na dvoch setoch dát. Prvý set dát obsahuje 14 656 vytvorených zhlukov o veľkosti 20 slov pre každý zhluk. Tieto zhluky boli vytvorené pomocou kategórií Wikipédie a metódy LDA. Pre každý zhluk bola vypočítaná percentuálna úspešnosť ohodnotenia a následne je vypočítaný aritmetický priemer všetkých úspešností priradení. Algoritmus na tomto sete dát dosiahol 21% úspešnosť. Druhý set dát je prebraný z webových stránok pre výučbu angličtiny [30]. Obsahuje 115 zhlukov, ktoré majú v priemere 113 slov. Tieto zhluky sú do tématickej kategórie zaradzované človekom a je isté, že naozaj patria do správnej kategórie. Na tomto sete dát dosiahol algoritmus 28% úspešnosť.

4.3 Diskusia výsledkov

Pri pohľade na výsledky behu algoritmu je jasne vidieť, v ktorých prípadoch algoritmus zlyháva. Pokiaľ všetky slová zhluku patria rovnakej sémantickej kategórii, algoritmus dosahuje vyššiu presnosť. Ak zluk obsahuje slová, ktoré majú akýkoľvek sémantický vzťah s danou oblasťou, ale zároveň patria do inej kategórie, algoritmus ťažko určuje správny výsledok. Ako príklad môže slúžiť zhluk: adder, anole, basilisk, bearded_dragon, black_racer, collared_lizard, na ktorom algoritmus správne určil, že sa jedná o plazy. Ako ďalší príklad môže slúžiť zhluk: episode, simpsons, homer, fox_network, season, bart, united_states, simpsons_season, show, pri ktorom kvalitu výsledkov znižovali slová ako fox_network (stanica, ktorá vytvára daný seriál), mená Homer a Bart (mená postáv seriálu, ktoré by osamote algoritmus ohodnotil ako tv_characters) a aj fakt, že zhluk obsahoval správneho kandidáta, a to slovo Simpsons. V praxi sa stretávame práve s ohodnocovaním takto sémanticky málo koherentných zhlukov, čo robí tento algoritmus nepoužiteľným. Riešením by bolo využitie inej ontológie, resp. menej špecifických vzťahov ako hyperonymia, alebo generovanie vlastnej ontológie (podobne ako boli generované testovácie dáta).

Záver

Úlohou mojej bakalárskej práce bolo navrhnúť a implementovať algoritmus, ktorý skupine slov priradí spoločný názov. Teoretická časť práce sa venuje teoretickému podkladu pre meranie sémantických vzťahov medzi slovami a zhlukovej analýze. Takisto sa venuje aj prehľadu rôznych metód pre automatické pomenovanie skupín slov.
V návrhovej časti boli navrhnuté dva algoritmy, ktoré by mohli slúžiť na pomenovanie skupín slov. Jeden z algoritmov vytvára pomenované zhluky slov za pomoci kategorizačnej štruktúry Wikipédie a abstraktov jej článkov.
Implementačná časť obsahuje prehľad existujúcich ontológii, voľbu nástrojov a knižníc, voľbu nástrojov pre predspracovanie textu a proces vytvárania testovacích dát.
V experimentálnej časti sa testuje algoritmus na presnosť výsledkov. Táto kapitola ukazuje, že algoritmus využívajúci ontológiu nedosahuje očakávané výsledky.
Presnosť algoritmu by sa dala zvýšiť predchádzajúcou analýzou, zjednoznačnením slov, využitím inej ontológie alebo iných menej špecifických vzťahov ako hyperonymických.

Literatúra

[1] FELDMAN, Ronen a SANGER, James. *The text mining handbook: advanced approaches in analyzing unstructured data*[online]. New York: Cambridge University Press, 2007. 410 s. [cit. 2017-01-08]. ISBN 9780521836579. Dostupné z: http://www.roelsbeestenboel.nl/text.pdf

[2] MANNING, Christopher D., RAGHAVAN, Prabhakar a SCHÜTZE, Hinrich. *Introduction to information retrieval*[online]. New York: Cambridge University Press, 2008. 482 s. [cit. 2017-01-08]. ISBN 0521865719. Dostupné z: http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html

[3] RIMARČÍK, Marián. *Štatistika pre prax.* Košice: M. Rimarčík, 2007. 200 s. ISBN 978-80-969813-1-1.

[4] TREERATPITUK, Pucktada a CALLAN, Jamie. Automatically labeling hierarchical clusters. In: *Proceedings of the 2006 International Conference on Digital government research*[online]. California: Digital Government Society of North America, 2006. s. 167-176. [cit. 2017-01-08]. Dostupné z: http://dl.acm.org/citation.cfm?id=1146650

[5] TSENG, Yuen-Hsien. Generic title labeling for clustered documents. In: *Expert Systems with Applications*[online]. Vol. 37. Taiwan: National Taiwan Normal University, 2010. s. 145-157. [cit. 2017-01-08]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0957417409007167

[6] HEDDEN, Heather. *Accidental taxonomist*[online]. New Jersey: Information Today, 2010. 442 s. [cit. 2017-01-08]. Dostupné z: https://www.scribd.com/doc/207323408/The-Accidental-Taxonomist

[7] SLIMANI, Thabet. Description and Evaluation of Semantic Similarity Measures Approaches: Evaluation of Semantic Term and Gene Similarity Measures. In: *International Journal of Computer Applications*[online]. Vol.

80. 2013. s. 25-33. [cit. 2017-01-08]. Dostupné z: http://arxiv.org/pdf/1310.8059.pdf

[8] HORNBY, Albert Sydney. *Oxford advanced learner's dictionary of current English.* Oxford: Oxford University Press, 2010. 1796 s. 8th edition. ISBN 978-0-19-479902-7.

[9] MILLER, G. A., BECKWITH, R., FELLBAUM, C. D., GROSS, D., MILLER, K. 1990. *WordNet: An online lexical database.* Int. J. Lexicograph. 3, 4, pp. 235–244.

[10] MILLER, G. A.(1995). *WordNet: A Lexical Database for English. Communications of the ACM.* Vol. 38, No. 11: 39-41.

[11] FELLBAUM, Christiane (1998, ed.) *WordNet: An Electronic Lexical Database.* Cambridge, MA: MIT Press.

[12] Wikipedia: the free encyclopedia. *Database download*[online]. San Francisco (CA): Wikimedia Foundation, 2001- . [cit. 2017-01-08]. Dostupné z: http://en.wikipedia.org/wiki/Wikipedia:Database_download

[13] The Wikimedia Foundation. *Wikimedia* [online]. [cit. 2017-01-08]. Dostupné z: https://dumps.wikimedia.org/enwiki/.

[14] MARTINEZ, Sergio, VALLS, Aida, SÁNCHEZ, David. Semantically-grounded construction of centroids for datasets with textual attributes. In: *Knowledge-Based Systems*[online]. Vol.35(4). 2012. s. 160-172. [cit. 2017-01-08]. DOI: 10.1093/obo/9780199756810-0003. Dostupné z: http://crises2-deim.urv.cat/docs/publications/journals/693.pdf

[15] LANDAUER, Thomas K., FOLTZ, Peter W. a LAHAM, Darrell. An Introduction to Latent Semantic Analysis. In: *Discourse Processes*[online]. Vol.25(2-3). 1998. s. 259-284. [cit. 2017-01-08]. DOI: 10.1080/01638539809545028. Dostupné z: lsa.colorado.edu/papers/dp1.LSAintro.pdf

[16] MOOR, Bart L.R. De. On the structure and geometry of the product singular value decomposition. In: *Linear Algebra and its Applications*[online]. Vol.168. 1992. s. 95-136. [cit. 2017-01-08]. Dostupné z: http://www.sciencedirect.com/science/article/pii/0024379592902900

[17] MENG, Lingling, HUANG, Runqing a GU, Junzhong.A Review of Semantic Similarity Measures in WordNet. In: *International Journal of Hybrid Information Technology*[online]. Vol.6. 2013. s. 1-12. [cit. 2017-01-03]. Dostupné z: http://www.cartagena99.com/recursos/alumnos/ejercicios/Article1.pdf

[18] DE MARNEFFE, Marie-Catherine, MACCARTNEY, Bill a MANNING, Christopher D. *Generating Typed Dependency Parses from Phrase Structure Parses*[online]. 2006. 6 s. [cit. 2017-01-08]. Dostupné z: http://nlp.stanford.edu/pubs/LREC06_dependencies.pdf

[19] VEENSTRA, Jorn, BUCHHOLZ, Sabine. *Fast NP Chunking Using Memory-Based Learning Techniques*[online]. Netherlands: Computational Linguistics at Tilburg University. 1998. 9 s. [cit. 2017-01-08]. Dostupné z: ilk.uvt.nl/~ilk/papers/NPChunking.ps

[20] BLEI, David M., NG, Andrew Y., JORDAN, Michael I. LAFFERTY, John, ed. Latent Dirichlet Allocation *Journal of Machine Learning Research*[online]. Vol.3. 2003. s. 993–1022. [cit. 2017-01-08] DOI:10.1162/jmlr.2003.3.4-5.993. Dostupné z: http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf

[21] NAVIGLI, Roberto. Word sense disambiguation: A survey. *ACM Comput. Surv. 41, 2, Article 10*[online]. 2009. 69 s. [cit. 2017-01-08]. DOI: https://doi.org/10.1145/1459352.1459355. Dostupné z: http://dl.acm.org/citation.cfm?id=1459355

[22] NAVIGLI, Roberto a PONZETTO, Simone Paolo. *BabelNet*[online]. University of Rome "La Sapienza": European Research Council (ERC). 2015. [cit. 2017-01-05]. Dostupné z: http://babelnet.org

[23] BOND, Francis. *Open Multilingual WordNet*[online]. Singapore: Nanyang Technological University. 2015. [cit. 2017-01-05]. Dostupné z: http://compling.hss.ntu.edu.sg/omw/

[24] BIZER, Christian, LEHMANN, Jens, KOBILAROV, Georgi, AUER, Soren, BECKER, Christian, CYGNIAK, Richard, HELLMANN, Sebastian. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*[online]. Vol.7(3). 2009. s. 154–165. [cit. 2017-01-08]. Dostupné z: http://www.wiwiss.fu-berlin.de/en/fachbereich/bwl/pwo/bizer/research/publications/Bizer-etal-DBpedia-CrystallizationPoint-JWS-Preprint.pdf

[25] MANNING, Christopher D., SURDEANU, Mihai, BAUER, John, FINKEL, Jenny, BETHARD, Steven J. a MCCLOSKY, David. The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*[online]. 2014. s. 55-60. [cit. 2017-01-08]. Dostupné z: http://stanfordnlp.github.io/CoreNLP/

[26] *TregexPattern. Stanford JavaNLP API Documentation*[online]. California: Stanford University. [cit. 2017-01-05]. Dostupné z:

http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/trees/tregex/TregexPattern.html

[27] *JGibbLDA*. Sourceforge.net [online]. 2008 [cit. 2017-01-05]. Dostupné z: http://jgibblda.sourceforge.net/

[28] FINLAYSON, Mark Adam.*Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation*[online].MIT. [cit. 2017-01-08]. Dostupné z: https://ai2-s2-pdfs.s3.amazonaws.com/0043/0c80d58d7bcbfe723be49946e01f03e0de2e.pdf

[29] *Stanford CoreNLP*[online]. [cit. 2017-01-08]. Dostupné z: http://corenlp.run/

[30] Enchanted learning. *Word Lists by Theme: Wordbanks*[online]. [cit. 2017-01-08]. Dostupné z: http://www.enchantedlearning.com/wordlist/

Dodatok **A**

Seznam použitých zkratek

LSA Latent semantic analysis

LSI Latent semantic indexing

SVD Singular value decomposition

XML Extensible markup language

pLSA probabilistic Latent semantic analysis

LDA Latent Dirichlet allocation

LCS Lowest common subsumer

TF Term frequency

IDF Inverted document frequency

NP Noun phrase

POS Part of speech

WSD Word sense disambiguation

BFS Breadth first search

CSV Comma separated values

URL Unifrom Resouce Locator

JWI Java WordNet Interface

SKOS Simple Knowledge Organization System

Dodatok B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` .....adresár so spustiteľnou formou implementácie
  - `clusterlabeling` .....
    - `clusterLabeling.jar` .....Spustiteľná froma implementácie značkovania zhlukov
    - `lib` ..Zložka obsahujúca knižnice nutné pre spustenie značkovania zhlukov
  - `testing-data-generation.sh` .....Skript pre spustenie generovania testovacích dát
  - `preprocessing-abstracts.sh` .Skript pre spustenie predspracovania dokumentov
  - `long_abstracts_en.csv` .....Tabuľka s abstraktami dokumentov
  - `run-clusterlabeling.sh` .....Skript pre spustenie algoritmu pre značkovanie zhlukov
  - `preprocessing` .....
    - `preprocessing.jar` ..... Spustiteľná forma predspracovania dokumentov
    - `lib` .....Zložka obsahujúca knižnice nutné pre spustenie predspracovania dokumentov
  - `abstracts-npchunks.csv` . Tabuľka s predspracovanými abstraktami dokumentov
  - `skos_categories_en.csv` .... CSV tabuľka kategorizačnej štruktúry Wikipédie
  - `article_categories_en.csv` ...CSV tabuľka článkov kategorizačnej štruktúry Wikipédia
  - `results.csv` .Výsledky behu algoritmu značkovania zhlukov množiny A
  - `homogenresults.csv` . Výsledky behu algoritmu značkovania zhlukov množiny B
  - `WordNet-3.0` .....Zložka obsahujúca slovník WordNet
  - `download-dbpedia.sh` .....Skript pre stiahnutie súborov DBpedie
  - `results` .....Zložka obsahujúca testovacie data A
  - `homogenresults` .....Zložka obsahujúca testovacie data B
  - `testingdatageneration` .....
    - `testingDataGeneration.jar` ..... Spustiteľná forma generovania testovacích dát
    - `lib` ..Zložka obsahujúca knižnice nutné pre spustenie generovania testovacích dát
- `src` .....
  - `corpusGeneration` .....NetBeans projekt obsahujúci zdrojové kódy generovania testovacích dát
  - `preprocessing` ..... NetBeans projekt obsahujúci zdrojové kódy predspracovania dokumentov
  - `clusterLabeling` ..... NetBeans projekt obsahujúci zdrojové kódy algoritmu značkovania zhlukov
  - `thesis` ..... Zdrojová forma práce vo formáte LATEX
- `text` .....Text práce
  - `thesis.pdf` .....Text práce vo formáte PDF
  - `thesis.ps` .....Text práce vo formáte PS