2010

Technická univerzita v Košiciach

Ján Paralič

Karol Furdík

Gabriel Tutoky

Peter Bednár

Martin Sarnovský

Peter Butka

František Babič

DOLOVANIE ZNALOSTÍ Z TEXTOV

Košice

doc. Ing. Ján Paralič, PhD.
Katedra kybernetiky a umelej inteligencie
Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach
Jan.Paralic@tuke.sk

Vydané s podporou Agentúry pre výskum a vývoj, na základe zmluvy
č. RPEU-0011-06

Lektoroval: RNDr. Michal Laclavik, PhD.

ISBN 978-80-89284-62-7

Obsah

Predhovor

Explicitné znalosti sa vo všeobecnosti najčastejšie reprezentujú pomocou textových, textovo-grafických, respektíve multimediálnych dokumentov. Pri tejto reprezentácii sa využíva skutočnosť, že prirodzený jazyk je najefektívnejším univerzálnym prostriedkom komunikácie medzi ľuďmi. Text, čiže grafická podoba prirodzeného jazyka, je takto médiom sprostredkúvajúcim prenos informácií a znalostí od autora k recipientom. Naviac, textová reprezentácia dovoľuje trvalé (perzistentné) uchovávanie komunikovaných informácií a znalostí.

V ešte pomerne nedávnej minulosti sa textové informácie zhromažďovali a uchovávali predovšetkým v klasickej „papierovej“ forme, napr. ako knihy, tlačené periodiká, kartotéky, archívne záznamy, a podobne. V súčasnosti sa však prakticky vo všetkých oblastiach ľudskej činnosti uprednostňuje elektronická forma záznamu textových údajov. Príkladom sú podnikové informačné systémy, elektronické zbierky zákonov a predpisov, elektronické knižnice a archívy a samozrejme, predovšetkým rozsiahly priestor webu. Transformácia do elektronickej podoby je výrazne podporovaná aj v tých oblastiach, kde sa textové dokumenty tradične vytvárali a uchovávali v „papierovej“ forme – možno napríklad spomenúť budovanie elektronických archívov historických knižníc, elektronizáciu zdravotníctva (*eHealth*) alebo verejnej správy (*eGovernment*).

Dôvodov, pre ktoré sa uprednostňuje elektronická reprezentácia textových dokumentov, je viacero. Predovšetkým je to efektivita a zníženie nákladov na archiváciu, znovupoužiteľnosť a prístupnosť širokej škále recipientov. Dôležitou výhodou elektronickej reprezentácie textových údajov je možnosť využiť sofistikované softvérové nástroje a prostriedky regulujúce prístup k takýmto dokumentom. Výrazne sa zjednodušuje a zrýchľuje vyhľadávanie v textoch, efektívne sa dajú texty analyzovať, vzájomne porovnávať, kategorizovať, štruktúrovať a integrovať s inými typmi údajov do rozsiahlych informačných systémov. Tieto úlohy komplexne rieši dolovanie znalostí z textov napr. pomocou metód klasifikácie (kategorizácie), zhlukovania, extrakcie informácií, ale aj iných.

Aj keď literárnych zdrojov o dolovaní znalostí z textov v anglickom jazyku v posledných rokoch pribúda, sú charakteristické značným rozptylom uhlu svojho pohľadu. Okrem toho publikácia o tejto problematike v slovenskom jazyku podľa nášho vedomia doteraz nebola vydaná. Keďže sa problematike dolovania znalostí v textoch venujeme na Fakulte elektrotechniky a informatiky, Technickej univerzity v Košiciach už niekoľko rokov, a to tak vo výskume, ako aj v pedagogickej činnosti, rozhodli sme sa preto vyplniť medzeru, ktorú v tejto oblasti vidíme.

Cieľom tejto monografie je na jednej strane predstaviť pomerne ucelený pohľad na komplexnú problematiku dolovania znalostí z kolekcií textových dokumentov, priblížiť niektoré výsledky nášho výskumu v tejto oblasti

za posledné roky[1] a ponúknuť aj praktické návody na riešenie konkrétnych úloh s využitím nástrojov, ktoré boli vyvinuté na Katedre kybernetiky a umelej inteligencie a v Centre pre informačné technológie, FEI, TU v Košiciach.

Dúfame, že túto publikáciu ocenia nielen naši študenti, najmä v inžinierskych študijných programoch Hospodárska informatika, Umelá inteligencia a Kybernetika na FEI, TU v Košiciach, ale aj širšia vedecká a odborná komunita na Slovensku a prípadne aj v Čechách.

Spomínané softvérové nástroje, najmä knižnica JBowl a k nej vytvorené webové rozhranie môže poslúžiť tak študentom, ako aj programátorom v širšom okolí, keďže je k dispozícii pod otvorenou licenciou. Popis spomínaných softvérových nástrojov a niekoľkých ukážkových úloh možno nájsť v druhej, prakticky zameranej časti tejto monografie, ako aj na web stránke, ktorá by sa na rozdiel od tejto publikácia mala ďalej vyvíjať v čase a poskytovať priebežne nové informácie a výsledky experimentov či už našich kolegov a študentov, alebo ako dúfame, postupne aj výsledky výskumníkov a študentov z iných univerzít na Slovensku.

Táto publikácia vyšla s podporou APVV projektu RPEU-0011-06.

Naše poďakovanie na tomto mieste patrí aj recenzentovi za starostlivé prečítanie rukopisu, opravu mnohých formálnych a aj vecných chýb a za cenné pripomienky a námety, ktoré obohatili obsah predkladaného textu.

Košice, marec 2010 autori

[1] Najmä tieto projekty: APVV projekty RPEU-0011-06 PoZnaŤ - riešený v rokoch 2007-2009 a APVV-0391-06 SEMCO-WS (2007-2009), 6FP-IST-27490 projekt KP-Lab (2006-2011), Slovensko-nemecký projekt Dolovanie znalostí v textoch pre extrakciu metadát a sémantické vyhľadávanie (2005-2006), VEGA projekty č. 1/4074/07 (2007-2009) a č. 1/1060/04 (2004-2006), Projekt MZ SR Dolovanie genomických dát o vývinových defektoch (2008-2010) a Rakúsko-slovenské projekty Analýza textov (2001-2002) a Vyhodnotenie metód zhlukovania (1999-2000).

Časť A (teoretická časť)

1 Úvod

V prvej kapitole budú vysvetlené základné pojmy, najmä definícia procesu objavovania znalostí vo všeobecnosti, a potom so zameraním sa na objvovanie znalostí v textových kolekciách. Nasleduje stručná charakteristika jednotlivých krokov tohto procesu, ktoré sú potom podrobne vysvetľované v nasledujúcich kapitolách.

1.1 Proces objavovania znalostí

Objavovanie znalostí je ***proces*** (semi-)automatickej extrakcie znalostí z rôznych typov dát (napr. z databáz – KDD: *Knowledge Discovery in Databases* [Paralič 2003], z textov – KDT: *Knowledge Discovery in Texts*, alebo často označované len dolovanie znalostí z textov – TM: *Text Mining*).

Extrahované znalosti pritom musia byť:

- platné (v štatistickom zmysle)
- doposiaľ neznáme a
- potenciálne užitočné (pre dané použitie, danú aplikáciu).

To, že objavovanie znalostí je *proces*, znamená, že sa skladá z viacerých krokov. Jednotlivé kroky tohto procesu budú stručne popísané v ďalšej časti. Aj keď ich postupnosť je pevne daná, realizácia samotného procesu už nie je len jednoduchý automatický prechod jednotlivými deterministickými krokmi, ale má dva podstatné atribúty.

Tento proces spravidla nie je možné plne automatizovať, resp. jeho plnou automatizáciou len ťažko dosiahneme optimálne výsledky v podobe skutočne cenných znalostí, ktoré môžu priniesť reálny prínos. Z tohto dôvodu je účelná, ba priam nevyhnutná *asistencia človeka*, ktorý rozhoduje o výbere vhodných operácií predspracovania, algoritmov a ich parametrov v rámci jednotlivých krokov procesu objavovania znalostí. Naviac len človek môže s konečnou platnosťou rozhodnúť, ktorá z objavených znalostí je natoľko nová a užitočná, aby sa dala s úspechom aplikovať v skúmanej aplikácii.

Z vyššie uvedeného vyplýva, že tento proces je v rámci svojich jednotlivých krokov nedeterministický, s mnohými možnými alternatívnymi rozhodnutiami v každom zo svojich krokov. Výber alternatívnej operácie, algoritmu, alebo zmena jeho parametrov potom nutne vedú k iným výstupom, čo následne ovplyvňuje ďalšie kroky. Preto v rámci procesu objavovania znalostí často dochádza k *iteráciám* v rámci jedného alebo viacerých jeho krokov s cieľom dosiahnuť čo najlepší výsledok. Proces objavovania znalostí z databáz je preto ***iteratívny*** a ***interaktívny***.

1.2 Proces objavovania znalostí z textov

Proces objavovania znalostí v množine textových dokumentov (*knowledge discovery in texts* – KDT, alebo často nazývaný aj *text mining*, resp. dolovanie z textov) sa v dôsledku inherentnej neštruktúrovanosti a neurčitosti jazyka javí ako omnoho zložitejší než proces objavovania znalostí v databázach. Pozrime sa teraz veľmi stručne na vývoj chápania konceptu

dolovania v textoch.

Jednou z prvých publikácií, ktorá sa snažila presnejšie vymedziť pojem dolovania znalostí z textov je [Hearst99]. Autorka sa v nej snaží vymedziť tento pojem v kontexte iných známych disciplín, ako napr. vyhľadávanie informácií (*information retrieval*), ktoré smeruje k vyhľadávaniu už známych informácií (minimálne autorom nájdených textov), a teda nespĺňa požiadavku novosti v zmysle vyššie uvedenej definície objavovania znalostí. Kľúčovou vlastnosťou je pre autorku novosť hľadaných vzorov v textových kolekciách.

Úlohy ktoré rieši výpočtová lingvistika založená na štatistickej analýze veľkých korpusov dokumentov, vedú k nájdeniu rôznych typov vzorov (väčšinou slúžiacich na riešenie vybraných úloh analýzy prirodzeného jazyka ako značkovanie textu, desambiguácia a pod.) sú podľa autorky analogické klasickému dolovaniu v dátach. Kategorizáciu textov chápe ako dolovanie v textoch len za určitých špeciálnych okolností (napr. identifikácia nových tém v článkoch, porovnávanie distribúcie kategórií v rôznych skupinách dokumentov s cieľom nájsť netypické vlastnosti, odlišnosti). Pre autorku je dolovanie v textoch najviac podobné exploračnej analýze dát, kedy dochádza k používateľom riadenej analýze za pomoci interakcie s výpočtovými nástrojmi na získavanie zaujímavých vzorov v textových kolekciách.

Autori v [Kroeze03] kriticky prehodnocujú niektoré závery z [Hearst99] a navrhujú mierne odlišné členenie dolovania v textoch. Pod tzv. *klasickým dolovaním v textoch* chápu napr. kategorizáciu textov, zhlukovanie textov, extrakciu lexikálnych a syntaktických znakov, acociácií medzi termami, extrakciu liniek medzi entitami v rámci textu a medzi rôznymi textami, ale napr. aj sumarizáciu textov. *Inteligentným dolovaním v textoch* potom nazývajú interakciu výskumníka a počítačového nástroja, ako aj použitie metód umelej inteligencie s cieľom vytvárať znalosti o okolitom svete na základe odvodených lingvistických znakov a ďalších typov vzorov. Napr. odráža novo identifikovaná téma v prúde dokumentov realitu? Aké stratégie skúmania implikujú nájdené prepojenia, sú naozaj relevantné? Aké obchodné rozhodnutia môžno na ich základe učiniť, a pod.

Dan Sullivan v aplikačne zameranej knihe [Sullivan 2001] definuje dolovanie znalostí z textov dvoma spôsobmi. Podľa prvého ide o akékoľvek operácie zamerané na získavanie a analýzu textov z externých zdrojov pre účely podnikovej inteligencie. Druhá definícia je zhoduje so všobecnou definíciou objavovania znalostí uvedenou na začiatku tejto kapitoly, teda "objavovanie predtým neznámych znalostí v textoch". Dan Sullivan potom uvádza viaceré podnikové aplikácie dolovania znalostí z textov a celý proces riadi metodológiou CRISP-DM[2] (Cross Industry Standard Process for Data Mining).

Veľmi podrobne sa problematike dolovania v textoch venuje kniha [Feldman, Sanger 2007], v ktorej autori definujú tento pojem dosť široko ako „*znalostne intenzívny proces v ktorom používateľ priebežne interaguje s kolekciou dokumentov za pomoci analytických nástrojov*". Pričom analogicky ako u dolovania v dátach aj pri dolovaní v textoch dochádza k extrakcii užitočných

[2] http://www.crisp-dm.org/

informácií z dátových zdrojov pomocou identifikácie a následnej analýzy zaujímavých vzorov. Feldman a Sanger ale kladú dôraz na analýzu prepojení (liniek) medzi informáciami obsiahnutými v dokumente, resp. kolekcii dokumentov. Kategorizácia, zhlukovanie a extrakcia informácií sú tu chápané najmä ako nástroje na predspracovanie textových dokumentov. Analýza liniek, z nej vyplývajúca grafová reprezentácia a v nej používané metódy analýzy (napr. rôzne typy analýzy sociálnych sietí) je nesporne veľmi zaujímavým predmetom skúmania, avšak podľa nášho názoru výrazne presahuje oblasť dolovania znalostí z textov. Analýza rôznych typov sietí je obzvlášť aktuálna v prostredí webu a rôznych zdrojov a aplikácií na ňom.

My sa prikláňame k chápaniu dolovania v textoch podľa základnej definície pre objavovanie znalostí, t.j. ako interaktívny a iteratívny proces získavania platných, pre danú aplikáciu užitočných a doposiaľ neznámych znalostí. Pritom používané algoritmy získavajú platné vzory rôznych typov, ktorých interpretácia a posúdenie užitočnosti je ponechané na človeka. Preto viacerými autormi zdôrazňovaná interakcia používateľa s analytickým systémom je naozaj nevyhnutná pre úspech celého procesu.

Objavovanie znalostí z textov stavia na viacerých vedných oblastiach. Jednou z najdôležitejších z nich je vyhľadávanie informácií [Laclavík et al. 2007], ktoré sa uplatňuje napr. V procese výberu cieľovej kolekcie textových dokumentov. Ale aj naopak, vhodná kategorizácia kolekcie textových dokumentov môže výrazne napomôcť kvalite procesu vyhľadávania informácii v ňom [Paralič, Koštiaľ 2003].

Ďalším príkladom veľmi dôležitej veeckej oblasti na ktorej dolovanie znalostí z textov stavia je výpočtová lingvistika, ktorá sa uplatňuje najmä vo faze predspracovania textových dokumentov a pri extrakcia informácií z nich (viď. kapitola 2, prípadne [Furdík 2003]).

Ako tretiu z významných vedeckých oblastí relevantných pre dolovanie znalostíá z textov spomenieme strojové učenie [Machová 2002]. Mnohé algoritmy pre kategorizáciu a zhlukovanie textov pochádzajú práve z tejto oblasti. Niektoré možnosti ich využitia ukázala nedávno publikovaná práca [Machová 2009].

Okrem spomínaných vedeckých oblastí významnou mierou ovplyvňuje dolovanie znalostí z textov aj dostupné technológie pre implementáciu systémov dolovania z textov. Autori tejto knihy pritom úspešne využívajú vlastný softvérový nástroj, tzv. knižnicu JBowl (*Java Bag-of-words Library*), ktorá poskytuje objektový model a rozhrania (API) pre vytváranie aplikácií získavania znalostí a dolovania v textoch. Knižnica JBowl obsahuje prostriedky na správu, indexáciu a manipuláciu s textovými dokumentmi, predovšetkým na komplexnú štatistickú analýzu textu vrátane podpory spracovania prirodzeného jazyka. Súčasťou knižnice sú implementácie viacerých algoritmov kontrolovaného a nekontrolovaného strojového učenia s voliteľnými vstupnými parametrami a metódami na vyhodnocovanie kvality modelov dolovania v textoch. Knižnica JBowl je podrobne popísaná v druhej časti tejto knihy (príloha B.1) a je voľne dostupná pre výukové a výskumné účely.

Z technologických vplyvov spomenieme ešte jeden. V poslednej dobe je to

napr. službovo orientovaný prístup [Habala et al. 2009], ktorý umožňuje celý process realizovať distribuovane, pričom napr. jednotlivé operácie predspracovania, resp. tvorby modelov, alebo ich vizualizáciu môžu byť realizované ako samostatné webové alebo gridové služby [Sarnovský et al. 2009]. O tomto prístupe pojednáva podrobnejšie kapitola 6 tejto knihy.

1.3 Stručný popis jednotlivých krokov

V princípe je možné proces dolovania znalostí z textových kolekcií priamo namapovať na proces objavovania znalostí. Pre jednotlivé kroky tohto procesu to znamená nasledovné.

Pochopenie aplikačnej domény – používateľ musí identifikovať kľúčové koncepty aplikačnej domény a stanoviť cieľ procesu KDT.

Získanie relevantnej množiny dokumentov – texty je potrebné starostlivo vybrať tak, aby pokrývali celú aplikačnú oblasť. Dokumenty sa získavajú buď z existujúcich zdrojov nástrojmi pre získavanie informácií (*information retrieval*) alebo manuálne, prípadne kombinovaným spôsobom.

Predspracovanie dát (podrobnejšie viď. nasledujúca kapitola **Error! Reference source not found.**) – v tomto prípade je ešte omnoho náročnejšie a závislé od použitého jazyka. Zahŕňa obvykle rôzne techniky spracovania prirodzeného jazyka (lexikálna, syntaktická a sémantická analýza, ale aj iné). V procese predspracovania dochádza obvykle k redukcii (odstránenie nevýznamových slov, prevedenie rôznych tvarov toho istého slova na kmeň a podobne) a následne k uloženiu množiny dokumentov v *internej forme reprezentácie* (napr. na báze vektorového modelu).

Dolovanie v textoch – tu ide o samotnú aplikáciu zvoleného algoritmu dolovania v textoch, napr. klasifikácia, resp. kategorizácia textov (podrobnejšie viď. kapitola **Error! Reference source not found.**), zhlukovanie (podrobnejšie viď. kapitola **Error! Reference source not found.**), alebo aj extrakciu informácií (podrobnejšie viď. kapitola 5). Extrakciu informácií možno však v niektorých prípadoch chápať skôr ako úlohu predspracovania. Okrem toho možno nájsť v druhej časti tejto knihy príklady dvoch konkrétnych aplikácií, jedna využívajúca kategorizáciu textov (príloha B.2) a druhá využívajúca extrakciu informácií (príloha B.3).

Vizualizácia a interpretácia výsledkov – vizualizáciu poskytujú technické nástroje na vhodnú vizuálnu reprezentáciu nájdených vzorov, resp. modelov odvodených v predchádzajúcom kroku procesu KDT, ale samotnú interpretáciu výsledkov musí urobiť človek. V tejto publikácii sa otázkam vizualizácie nájdených vzorov venujeme iba okrajovo. Čitateľov, ktorých zaujíma práve táto časť procesu objavovania znalostí v textových kolekciách, môže nájsť mnoho zaujímavých informácií na túto tému napr. v už spomínanej knihe [Feldman, Sanger 2007].

Dolovanie znalostí z textov môže byť užitočné prakticky v každej aplikačnej oblasti, kde sa vyskytuje veľké množstvo textových dát. Autori tejto knihy úspešne aplikovali prístupy dolovania v textoch v celom rade národných a medzinárodných projektov s rôznymi typmi pilotných aplikácií, napr. elektronická verejná správa [Paralič, Bednár 2003], alebo posledne oblasť

kolaboratívnej tvorby nových znalostí v procesoch učenia, resp. práce [Furdík et al. 2010]. Základné informácie o týchto ale aj ďalších úspešných projektoch, kde sme dolovanie znalostí z textov využívali možno nájsť v prílohe (konkrétne v časti B.1.3).

O významnosti aplikácií (najmä pre oblasť biomedicíny a genetiky) svedčí napr. aj existencia národného cetra pre dolovanie z textov (*National Centre for Text Mining - NaCTeM*)[3], ktoré je prevádzkované Univerzitou v Manchestri v spolupráci s Univerzitou Tokio.

[3] http://www.nactem.ac.uk/

2 Predspracovanie textových dát

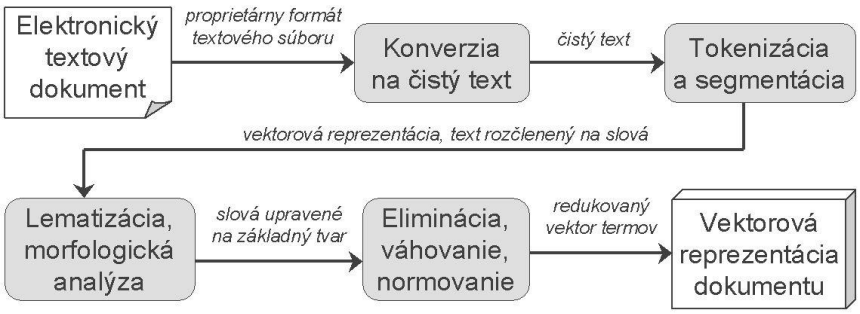
Predpokladom pre aplikovanie metód dolovania znalostí je transformácia textových dokumentov na reprezentačnú štruktúru (viac o základných modeloch na reprezentáciu textov je uvedené nižšie, v časti 2.7) vhodnú pre príslušné klasifikačné či zhlukovacie algoritmy[4]. Tieto sú najčastejšie založené na viacrozmernej analýze, ktorá na vstupe predpokladá údajovú m–rozmernú maticu pozorovaní na n objektoch [Řezanková a kol. 2007]. Objektmi sú v tomto prípade jednotlivé textové dokumenty v skúmanom súbore (*korpuse*). Pozorovania sú váhy jednotlivých kľúčových slov (resp. *termov*; angl. *keywords*, resp. *terms*) v texte každého z dokumentov. Vyjadrujú hodnoty týchto charakteristických znakov, resp. premenných, pozorovaných na textových dokumentoch.

$$\mathbf{F}(N \times M) = \begin{pmatrix} \bar{d}_1 \\ \bar{d}_2 \\ \dots \\ \bar{d}_N \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1M} \\ w_{21} & w_{22} & \dots & w_{2M} \\ \dots & \dots & \dots & \dots \\ w_{N1} & w_{N2} & \dots & w_{NM} \end{pmatrix}$$

Obr. 2.1 Matica dokument–term

V takejto vstupnej matici, označovanej tiež ako *matica dokument–term* (viď. Obr. 2.1) je jeden textový dokument reprezentovaný ako vektor váh odvodených primárne od početností výskytov jednotlivých termov (podrobnejšie o výpočte váh v časti 2.5) – kľúčových slov, ktoré dostatočne reprezentatívne vyjadrujú obsah daného dokumentu a vymedzujú ho voči iným dokumentom zo skúmaného korpusu. Z tohto hľadiska možno považovať túto vektorovú reprezentáciu dokumentu aj za *príznakový popis*, kde početnosti výskytov termov v texte dokumentu sú príznakmi, t.j. charakteristickými vlastnosťami analyzovaného dokumentu.

Získanie príznakových popisov pre všetky textové dokumenty zo skúmaného korpusu sa súhrnne označuje ako *predspracovanie textových údajov*. Táto prípravná fáza aplikácie metód dolovania znalostí pozostáva zo sekvencie čiastkových krokov znázornených na Obr. 2.2.



Obr. 2.2 Schéma predspracovania textových údajov

[4] Základné modely na reprezentáciu textových dokumentov sú popísané v časti 2.7. Algoritmom klasifikácie a zhlukovania textov sú venované kapitoly 3 a 4.

Na vstupe sa predpokladajú elektronické dokumenty v rôznych proprietárnych formátoch. Prvým krokom je odstránenie redundantných formátovacích znakov a konverzia dokumentu na tzv. „čistý text“ (angl. *plain text*). Tento text sa ďalej delí na elementárne textové jednotky, tzv. *tokeny* (angl. *tokens*). Následne sa v texte identifikujú slová, tzv. *lexikálne jednotky*, pre ktoré sa určí príslušný základný tvar (*lema*) a morfologické kategórie. Napokon sa odstránia neplnovýznamové slová, t.j. tie, pri ktorých sa predpokladá malý prínos k vyjadreniu celkového obsahu dokumentu. Zostávajúce plnovýznamové slová, ohodnotené vhodnou váhovou funkciou, potom tvoria hľadanú vektorovú reprezentáciu vstupného dokumentu.

Pre zníženie výpočtovej náročnosti a zvýšenie efektívnosti algoritmov dolovania znalostí z textov je potrebné, aby bol rozmer vektora príznakov čo najmenší. Na to sa používajú rôzne matematické metódy, kombinované s heuristickými metódami založenými na lingvistickej analýze textu. Tieto metódy spomedzi všetkých termov obsiahnutých v dokumente vyberajú tie, ktoré čo najlepšie charakterizujú daný dokument a čo najviac ho odlišujú od všetkých ostatných dokumentov.

V nasledujúcich častiach podrobnejšie popíšeme jednotlivé kroky predspracovania, predstavíme najčastejšie používané modely reprezentácie textových dokumentov a priblížime spôsoby redukcie príznakového priestoru termov.

2.1 Konverzia elektronických dokumentov na textový formát

Prvým krokom predspracovania dokumentov obsahujúcich textovú informáciu je jej získanie v podobe čistého textu.

Čistý text. Čistý text možno definovať ako sekvenciu alfanumerických, interpunkčných, oddeľovacích grafických znakov a niektorých špeciálnych symbolov (napr. %, &, *, a pod.). Alfanumerické znaky majú fonetickú a lexikálnu hodnotu, sú teda priamymi nositeľmi obsahu textu. Oddeľovacie a interpunkčné znaky (napr. medzera, tabelátor, bodka, čiarka, pomlčka, zátvorky, lomka, atď.) sa používajú na členenie textu. Čistý text sa dá získať z formátovaného elektronického dokumentu tak, že sa z neho odstránia všetky typografické značky a netextové informácie, ako sú napríklad označenia veľkosti a typu písma, obrázky a grafika, vzorce, tabuľky, grafy, a podobne.

V internetovej komunikácii, napr. na webe alebo pri elektronickej pošte, sa používa štandard MIME (*Multipurpose Internet Mail Extensions*). Typ obsahu je možné nastaviť v hlavičke dokumentu pomocou atribútu *Content-type*. Internetový dokument obsahujúci čistý text sa označuje konštrukciou:

```
Content-Type: text/plain
```

Typy obsahu internetových dokumentov definuje organizácia IANA (*Internet Assigned Numbers Authority*, http://www.iana.org/). Aktuálny zoznam typov obsahu možno nájsť na http://www.iana.org/assignments/media-types/ [IANA–MIME 2007].

Dokumenty sú k dispozícii obyčajne **v rôznych formátoch**, napr. v RTF, MS Word, PDF, HTML, a v ďalších. Každý z formátov je špecifický a rôznym spôsobom podporuje konverziu do čisto textového formátu. Tab. 2.1 uvádza prehľad niektorých najčastejšie podporovaných formátov spolu s ich stručným popisom a nástrojmi na extrakciu čistého textu v platforme Java.

Tab. 2.1 Najčastejšie formáty elektorických dokumentov, nástroje na získanie čistého textu z týchto formátov.

Formát: **.DOC**, **.XLS**	
Popis:	Proprietárny binárny formát. Vlastník licencie: Microsoft Corporation Formáty balíka Microsoft Office 97-2003, MS Word, Excel
Konverzia na čistý text:	Apache POI, http://poi.apache.org/
Formát: **.DOCX** (Office Open XML Document)	
Popis:	Proprietárny textový formát, štandardizovaný od r. 2008 (normy ECMA-376 a ISO/IEC 29500) Vlastník licencie: Microsoft Corporation Formát balíka Microsoft Office 2007, MS Word
Konverzia na čistý text:	POI-OpenXML4J, http://poi.apache.org/oxml4j/index.html Keďže .DOCX formát zodpovedá komprimovanému .ZIP súboru obsahujúcemu XML, možno použiť Java mechanizmy na rozbalenie a parsovanie. Napr. na http://computercranium.com/programming/java/howto-extract-text-from-docx-file-word-2007. Ďalšou možnosťou je konverzia do .DOC formátu (viď napr. http://www.docx2doc.com) a následné použitie príslušných nástrojov pre konverziu.
Formát: **.RTF** (Rich Text Format)	
Popis:	Proprietárny textový formát. Vlastník licencie: Microsoft Corporation
Konverzia na čistý text:	Knižnica javax.swing.text.rtf.*.
Formát: **.PDF** (Portable Document Format)	
Popis:	Otvorený textový formát od 1.7.2008, kedy bola verzia PDF 1.7 publikovaná ako norma ISO/IEC 32000-1:2008 Vlastník licencie: Adobe Systems

Konverzia na čistý text:	Viacero Java knižníc a nástrojov, napr.: Apache PDFBox, http://incubator.apache.org/pdfbox/, jPDFText, http://www.qoppa.com/pdftext/jptindex.html, a ďalšie (väčšinou komerčné) nástroje
Formát: **.ODT** (OpenDocument)	
Popis:	Otvorený textový formát, založený na XML. Od r. 2006 ISO štandard (norma ISO/IEC 26300:2006). Od 1.7.2008, kedy bola verzia PDF 1.7 publikovaná ako norma ISO/IEC 32000-1:2008 Vlastník licencie: Sun Microsystems, OASIS
Konverzia na čistý text:	Viacero Java knižníc a nástrojov, napr.: OpenOffice.org API projekt, http://api.openoffice.org JODConverter, http://sourceforge.net/projects/jodconverter/
Formát: **.PS** (PostScript)	
Popis:	Otvorený textový formát – programovací (skriptovací) jazyk, ktorý po interpretovaní emuluje tlačenú formu dokumentu. Vlastník licencie: Adobe Systems
Konverzia na čistý text:	Konverzia do PDF (napr. pomocou nástroja Ghostscript, http://pages.cs.wisc.edu/~ghost/. Následne použitie nástrojov pre konverziu PDF. Tiež sa dá použiť trieda PSFormatter, viď kód napr. na http://www.javafaq.nu/java-example-code-959.html.
Formát: **.TEX**	
Popis:	Otvorený textový formát. TeX Users Group, http://www.tug.org
Konverzia na čistý text:	JavaTeX, http://sourceforge.net/projects/javatex/
Formát: **.HTML** (HyperText Markup Language)	
Popis:	Otvorený textový formát pre web dokumenty. Od r. 2000 ISO štandard (norma ISO/IEC 15445:2000). HTML 4.01, http://www.w3.org/TR/1999/REC-html401-19991224/

Konverzia na čistý text:	Viacero Java knižníc a nástrojov, napr: Knižnica javax.swing.text.html.*; HTML Parser, http://htmlparser.sourceforge.net; JTidy, http://jtidy.sourceforge.net; NekoHTML, http://nekohtml.sourceforge.net.
Formát: **.XML** (eXtensible Markup Language)	
Popis:	Otvorený textový formát pre štruktúrovaný obsah. XML 1.0, http://www.w3.org/TR/2008/REC-xml-20081126/
Konverzia na čistý text:	Xerces parser (DOM/SAX), súčasť distribúcie JDK 1.5, http://xerces.apache.org/xerces-j/

Pri získavaní čistého textu a prípadnej ďalšej manipulácii s ním má zásadný význam jeho **kódovanie** (angl. *encoding*). Štandardom a základným kódovaním pre angličtinu je ASCII (*American Standard Code for Information Interchange*). Obsahuje definície 128 znakov – 33 riadiacich znakov, 94 znakov pre tlač a znak medzery. ASCII znaky sú kódované na 7 bitoch.

Na základe ASCII štandardu boli vytvorené verzie pre rôzne európske jazyky. Kódovanie sa rozšírilo na 8 bitov, čo dovoľovalo definovať 256 znakov. Prvých 128 znakov je v týchto kódovaniach povinne zhodných s pôvodným kódovaním ASCII, ďalších 128 pozícií je obsadzovaných znakmi špecifickými pre ten-ktorý jazyk. Týmto spôsobom vzniklo pomerne veľké množstvo rôznych kódovacích schém, a to aj niekoľko rôznych schém pre jeden jazyk [IANA–ChS 2007]. Pre slovenčinu a češtinu boli na základe ASCII vytvorené kódovania *kód Kamenických* (tiež *KEYBCS2*), *PC Latin 2*, *ISO Latin 2* (tiež *ISO 8859-2*), *KOI-8 CS2*, *Win-1250* a *MacOS CE*.

Existencia mnohých kódovaní odvodených od ASCII spôsobovala veľké problémy. Nevyhnutné boli nástroje na transformáciu z jedného kódovania do iného v rámci jedného jazyka. Nebolo možné vytvárať viacjazyčné dokumenty. Naviac, mimoeurópske, najmä východoázijské jazyky (čínština, japončina, kórejčina) nebolo možné na 256 znakoch zakódovať vôbec. Preto sa vytvorili kódovacie systémy na iných princípoch ako ASCII, ktoré uvedené nedostatky odstraňovali. Dnes je najrozšírenejším štandardom kódovanie na základe kódovacej tabuľky *Unicode*, navrhnuté pomocou dvojbajtového kódovania znakov. To dovoľuje vytvárať sady s 65 536 znakmi. Štandard Unicode rozdeľuje všetky možné znaky do sedemnástich dvojbajtových rovín, čo umožňuje definovať až 1 114 112 (= 17 × 216) znakov. Prvá verzia štandardu vznikla v októbri 1991, v súčasnosti je najnovšia verzia Unicode 5.1 z roku 2008 [Unicode5.1 2008]. Obsahuje 100 713 znakov a symbolov zo 75 rôznych jazykov.

Kvôli spätnej kompatibilite s kódovaním ASCII a pod vplyvom niektorých ďalších problémov s praktickým používaním Unicode sa vytvorilo kódovanie UTF (*Unicode Transformation Format*) s premenlivou bitovou dĺžkou. UTF kóduje prvých 128 znakov zhodne s ASCII tabuľkou, čím sa zabezpečila spätná kompatibilita. Odlišujú sa až ďalšie znaky, ktoré sa kódujú viac ako

ôsmimi bitmi. Azda najpoužívanejším je kódovanie *UTF-8*, ktoré je popísané v štandarde ISO 10646-1:2000. Formát UTF-8 kóduje na prvom bajte (t.j. 8 bitoch) znaky ASCII, na druhom až šiestom bajte kóduje ďalšie znaky tabuľky Unicode. Pre slovenskú abecedu stačí pre znaky bez diakritiky jeden bajt a pre znaky s diakritikou dva bajty.

V internetových dokumentoch (e-mail, web) sa používa kódovanie UTF-8, avšak aj staršie typy kódovaní odvodených od ASCII. Dokument má v svojej hlavičke deklarované použité kódovanie pomocou atribútu *charset* v parametri *content*. Nasledujúce zápisy ukazujú príklady nastavenia kódovania v HTML dokumentoch:

Kódovanie ISO-8859-2:

```
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-2">
```

Kódovanie ISO UTF-8:

```
<meta http-equiv="content-type" content="text/html;
charset=utf-8" />
```

Pri konverzii na čistý text sa odporúča používať kódovanie UTF-8. Vývojové prostredia jazyka Java toto kódovanie priamo podporujú, preto pre texty v UTF-8 nie je potrebná žiadna ďalšia konverzia. Avšak pre texty v kódovaniach odvodených od ASCII je potrebné vykonať príslušnú transformáciu, a to pomocou triedy *Locale* a ďalších objektov pre internacionalizáciu [Java–Intl 2008].

Pri získavaní čistého textu, predovšetkým pri extrakcii z bohatších formátov (PDF, MS Word, HTML), sa často strácajú relevantné a dôležité informácie o obsahu, štruktúre textu a o dokumente ako celku. Viaceré z týchto informácií môžu byť v ďalšom procese dolovania z textov veľmi užitočné a je preto dôležité ich istým spôsobom zachovať. Jednou z možností je archivácia pôvodného dokumentu spolu s referenciou na príslušný čistý text. Ďalším spôsobom je kódovanie dôležitých údajov v modifikovanom formáte čistého textu, napríklad pomocou XML notácie – týmto spôsobom možno definovať hlavičku s globálnymi informáciami o pôvodnom dokumente, čistý text členiť do kapitol a odsekov, zachovať označenia nadpisov a zvýraznených častí textu, a podobne.

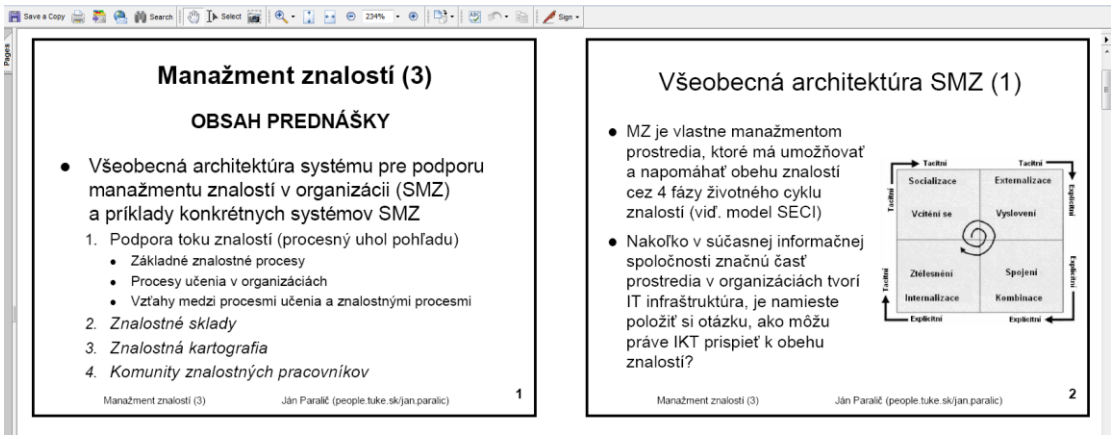
Údaje, ktoré sa dajú odvodiť z pôvodného elektronického dokumentu, avšak sa strácajú pri jednoduchej konverzii na čistý text, označujeme termínom **metainformácie**. Tieto je možné a aj vhodné v etape konverzie na čistý text z pôvodného dokumentu získať a v príslušnom formáte (napr. v XML) uložiť pre ich využitie v ďalších fázach procesu dolovania z textov.

Metainformácie môžu byť napríklad:

- *Bibliografické údaje* o textoch, napríklad autor, dátum publikovania, vydavateľ, zdroj alebo umiestnenie dokumentu, typ dokumentu, veľkosť elektronického súboru, a podobne.

- *Metaúdaje*, anotácie vložené autorom do dokumentu, ktoré dodatočne charakterizujú obsah textu (napr. *Properties* v MS Word formáte).
- *Informácie o štruktúre textu*, napríklad členenie na kapitoly, odseky, nadpisy a podobne. Možno sem zaradiť aj typografické informácie – zlom riadkov a stránok, zvýraznenie rôznymi typmi písma, odsadenie, odrážky, a ďalšie.

Na Obr. 2.3 je príklad elektronického dokumentu v PDF formáte [Paralič 2008]. Zdrojový kód tohto dokumentu je interpretovaný a zobrazený aplikáciou Adobe Reader (http://get.adobe.com/reader/).



Obr. 2.3 Elektronický dokument v PDF formáte

Čistý text získaný z PDF dokumentu z Obr. 2.3 je prezentovaný na Obr. 2.4. Pri tejto transformácii sa stratilo rozlíšenie názvov od textu a členenie odrážok. Texty z grafického obrázku sa nepodarilo extrahovať.

```
Manažment znalostí (3-4)
OBSAH PREDNÁŠKY
Všeobecná architektúra systému pre podporu manažmentu
znalostí v organizácii (SMZ) a príklady konkrétnych
systémov SMZ
1. Podpora toku znalostí (procesný uhol pohľadu)
Základné znalostné procesy
Procesy učenia v organizáciách
Vzťahy medzi procesmi učenia a znalostnými procesmi
2. Znalostné sklady
Požiadavky na znalostný sklad
Typy znalostných skladov (znalostné podkrovie, znalostná
špongia, znalostné vydavateľstvo, znalostná pumpa)
Vnútorná organizácia znalostného skladu
3. Znalostná kartografia
4. Komunity znalostných pracovníkov
Manažment znalostí (3-4) Ján Paralič
(people.tuke.sk/jan.paralic) 1

Všeobecná architektúra SMZ (1)
MZ je vlastne manažmentom prostredia, ktoré má umožňovať
a napomáhať obehu znalostí cez 4 fázy životného cyklu
```

```
znalostí (viď. model SECI)
Nakoľko v súčasnej informačnej spoločnosti značnú časť
prostredia v organizáciách tvorí
IT infraštruktúra, je namieste položiť si otázku, ako môžu
práve IKT prispieť k obehu
znalostí?
Manažment znalostí (3-4) Ján Paralič
(people.tuke.sk/jan.paralic) 2
```

Obr. 2.4 Čistý text extrahovaný z elektronického dokumentu

Na Obr. 2.5 je opäť čistý text daného PDF dokumentu, ktorý je však ohodnotený metainformáciami vo formáte XML. Hlavička obsahuje údaje o autorovi, mieste uloženia, veľkosti a formáte pôvodného dokumentu, o type a zameraní obsahu textu. Vlastný text je členený XML značkami. Vyznačený je nadpis a podnadpis, rozdelenie odrážok (použité sú značky HTML) a poznámka v dolnej časti textu, v ktorej je identifikované číslo strany a webový odkaz.

```
<header>
    <author>Ján Paralič</author>
    <source>
         <location>http://people.tuke.sk/jan.paralic/
         prezentacie/MZ/MZ3.pdf</location>
         <size>2MB</size>
         <format>PDF</format>
    </source>
    <type>Prezentácia k prednáškam</type>
    <topic>Manažment znalostí</topic>
</header>
<plain text>
    <title>Manažment znalostí (3)</title>
    <sub_title>OBSAH PREDNÁŠKY</sub_title>
    <ul>
         <li>Všeobecná architektúra systému pre podporu
         manažmentu znalostí v organizácii (SMZ) a príklady
         konkrétnych systémov SMZ
         <ul>
              <li>1. Podpora toku znalostí (procesný uhol
              pohľadu)
              <ul>
                   <li>Základné znalostné procesy</li>
                   <li>Procesy učenia v organizáciách</li>
                   <li>Vzťahy medzi procesmi učenia
                   a znalostnými procesmi</li>
              </ul>
              </li>
              <li>2. Znalostné sklady</li>
              <li>3. Znalostná kartografia</li>
              <li>4. Komunity znalostných pracovníkov</li>
         </ul>
         </li>
    </ul>
<footnote>Manažment znalostí (3-4) Ján Paralič
(<a>people.tuke.sk/jan.paralic</a>)
<page_num>1</page_num>
```

```
</footnote>

... tu by mal nasledovať výpis druhej časti textu,
ten však kvôli úspornosti vynechávame.

</plain text>
```

Obr. 2.5 Čistý text v XML notácii, doplnený o metainformácie

2.2 Tokenizácia a segmentácia

Segmentácia, členenie na slová (angl. *parsing*) a následné značkovanie (tiež *tokenizácia*, angl. *tokenization*) sú inicializačné textové operácie, pri ktorých sa vo vstupnom čistom texte dokumentu identifikujú základné lexikálne textové jednotky – slová, slovné spojenia a frázy, vety, odseky, a pod.

Vo fáze segmentácie sa text delí na najmenšie prípustné sekvencie znakov, tzv. *elementárne textové jednotky*, ktorými sú:

- súvislé reťazce alfanumerických znakov, oddelené medzerami alebo znakmi interpunkcie,
- jednotlivé znaky interpunkcie.

Napríklad fragment textu:

```
1. Podpora toku znalostí (procesný uhol pohľadu)
```

sa rozdelí na desať elementárnych textových jednotiek takto:

```
[1][.][Podpora][toku][znalostí][(][procesný][uhol][pohľadu][)]
```

V nasledujúcej fáze tokenizácie sa elementárne textové jednotky konvertujú na tzv. *lexikálne jednotky* – značky, resp. tokeny (angl. *tokens*). Tokeny možno definovať ako systémom rozpoznané a akceptované skupiny znakov s kolektívnym významom[5], ktoré obyčajne zodpovedajú konkrétnemu slovníkovému záznamu. Identifikujú sa pomocou vopred zostavených slovníkov prípustných tvarov slov v kombinácii s rôznymi pravidlovými systémami[6].

Pri tokenizácii sa združujú postupnosti identifikovaných elementárnych textových jednotiek tak, aby výsledkom bol prípustný slovníkový tvar určitého slova. Napríklad textové jednotky `[1]` a `[.]` sa dajú spojiť do jedného tokenu `[1.]`, ktorý zodpovedá slovníkovému záznamu radovej číslovky „prvý“. Podobne sa môžu spájať sekvencie elementárnych textových

[5] Podľa definície uvedenej v [Garabík a kol. 2004], token je „arbitrárna jednotka textu, ktorá rozširuje lingvistický význam pojmu slovo“.

[6] Dôsledkom toho je, že proces tokenizácie je závislý od konkrétneho jazyka analyzovaných textov. Pre automatizovanú tokenizáciu je potrebné heuristickými algoritmami zisťovať jazyk týchto textov [Zeman 2009].

jednotiek, ktoré vyjadrujú zložitejšie číselné výrazy (napr. „3,14“, „1 984“, „25 °C“, „8-krát“, „18,36 EUR “, „Win98“), ustálené zložené pomenovania a názvy („Spišská Nová Ves“, „Technická univerzita“, „viac-menej“), skratky („angl.“, „a pod.“), adresy elektronickej pošty, URL adresy webovských stránok a ďalšie špeciálne formáty.

O čosi problematickejšie je spájanie voľnejších zložených tvarov, napríklad „v rámci“, „a teda“, prípadne aj „tok znalostí“, „uhol pohľadu“, a podobne. V týchto prípadoch, ktoré nezodpovedajú štandardným slovníkovým záznamom, je azda vhodnejšie reprezentovať každú zo zložiek výrazu osobitným tokenom a prípadné spájanie vykonať až pri morfologickej alebo syntaktickej analýze. Konkrétny postup však závisí od stavby použitého slovníka a pravidlového systému.

Naopak, pri aglutinovaných podobách slov „oňho“, „akoby“, a podobne, by bolo možné uvažovať o rozdelení týchto elementárnych textových jednotiek do viacerých tokenov. Avšak takýto postup sa neodporúča, pretože v ďalšom spracovaní (najmä pri morfologickej a syntaktickej analýze) by mohlo dôjsť k dezinterpretácii [Garabík a kol. 2004].

Vzhľadom na využitie tokenizácie na predspracovanie a následné dolovanie znalostí z textov pomocou klasifikačných a zhlukovacích algoritmov je vhodné zohľadniť isté javy a špeciálne tvary lexikálnych jednotiek a riešiť ich pomocou adekvátneho návrhu systému tokenizačných pravidiel. Sú to najmä:

- *Čísla*. Čísla a zmiešané alfanumerické reťazce väčšinou nie sú dobrými termami vo vektorovej reprezentácii dokumentov, a to kvôli ich neurčitosti. Niekedy je však potrebné ich ako jednoznačné termy identifikovať. Ide napríklad o telefónne čísla, PSČ, ŠPZ a podobne. Termami môžu byť aj čísla nachádzajúce sa v technických dokumentáciách, čísla kapitol a rôzne označenia („vitamín B12“, „tlačiareň HP 2000“, „projekt 100“ atď.).

  Ak sa má slovo alebo slovné spojenie neskôr stať termom, musí sa najprv vo fáze tokentizácie identifikovať ako token[7]. Je preto vhodné ako tokeny identifikovať čísla a číselné výrazy s čo najširším textovým okolím so známym formátom (dátum, PSČ, ŠPZ, telefónne čísla). Možným riešením je napríklad tokenizácia tých reťazcov, ktoré čísla obsahujú, no nezačínajú nimi, prípadne je možné vyjadriť akceptované reťazce pomocou formálnej gramatiky alebo vymenovaním v slovníku (napr. dátum *11. september* môže byť všeobecným dátumom, ale aj termom s veľmi konkrétnym významom).

  Čísla, ktoré sa nepodarí tokenizovať s textovým okolím alebo pomocou niektorého z akceptovaných formátov, sa reprezentujú ako samostatné tokeny. Tieto, vzhľadom na ich neurčitosť, je potrebné

[7] Azda je užitočné na tomto mieste zdôrazniť *rozdiel medzi tokenom a termom*. Token je časť textu so špecifikovaným slovníkovým významom a s určenou pozíciou. Text sa teda počas tokenizácie spojito člení na jednotlivé tokeny. Term je kľúčové slovo, ktoré je súčasťou vektorovej reprezentácie textového dokumentu. Termy sa vytvárajú z tokenov v procese lematizácie, jazykovej analýzy a eliminácie stop-slov. Z toho vyplýva, že nie každý token sa transformuje na term, avšak každý term je vytvorený z jedného alebo viacerých tokenov .

vo fáze eliminácie stop-slov vyradiť zo zoznamu kandidátov na termy (porov. časť 2.4).

- *Zložené slová*. Je otázkou, či slová so spojovníkom (napr. „Jean-Claude“, „hi-fi“, „MS-DOS“ a pod.) pri tokenizácii spojiť, alebo rozdeliť do viacerých tokenov. Odpoveď nie je jednoznačná – spojenie do jedného tokenu môže byť problematické napríklad pri tvaroch typu „Prešov-Košice“, kde bol v texte chybne použitý spojovník namiesto pomlčky. Možným riešením je preskúmať, či sa textové jednotky pred a za spojovníkom nachádzajú v slovníku prípustných tvarov – ak áno, potom treba zložený tvar rozdeliť a zložky reprezentovať ako samostatné tokeny.
- *Interpunkcia*. Väčšinou sú interpunkčné znaky interpretované ako oddeľovače, no niekedy nastáva podobný problém ako pri zložených slovách (napr. „verzia 1.32“, „1.44 MB“, „PS/1“, „abc@email.com“, atď). Potrebné je nastaviť systém tokenizačných pravidiel tak, aby dokázal identifikovať aspoň základné typy zložených tvarov obsahujúcich interpunkčné tvary – adresy elektronickej pošty a webovských stránok, desatinné čísla, a podobne.
- *Veľkosť písma*. Znaky sa často v procese tokenizácie konvertujú buď na veľké, alebo na malé písmo, čo však môže niekedy znamenať stratu sémantickej informácie. Naopak, veľké písmená vo všeobecnosti nemajú absolútnu sémantickú platnosť (napr. rozhodne neplatí, že vždy indikujú začiatok vety). Odporúčaným postupom je zachovať informáciu o skutočnom tvare tokenu v texte, vrátane veľkých a malých písmen, ako jeden z parametrov tokenu.

Vo všeobecnosti však pre proces tokenizácie platí, že všetky typy tokenov musia byť identifikovateľné a odlíšiteľné od ostatných.

Tokenizácia, spolu s ďalšími etapami predspracovania, sa uplatňuje nielen pri úlohách dolovania znalostí z textov, ale aj v oblasti vyhľadávania informácií (angl. *information retrieval*) a v korpusovej lingvistike. Aplikácie v týchto výskumných oblastiach používajú spôsoby reprezentácie tokenu, ktoré sa dajú s výhodou použiť aj pri predspracovaní textu pre úlohy získavania znalostí. V korpusovej lingvistike [Rychlý 2000] sa pri tokenizácii definuje prvok *pozícia* (niekedy tiež *L-word*), ktorý zodpovedá vyššie uvedenej definícii tokenu a identifikuje najmenšiu zaznamenateľnú textovú jednotku. Texty v korpuse sú reprezentované ako *postupnosti pozícií*, kde na každej pozícii je jedno slovo, číslo, znak interpunkčný znak, atď. Neurčitosti pri značkovaní interpunkcie, zložených slov, skratiek, číselných údajov a podobne sa najčastejšie riešia postupom, pri ktorom sa do samostatných pozícií text rozčlení na maximálny možný počet samostatných pozícií (t.j. bez spájania zložených slov, číselných výrazov atď.). Takto sa na samostatnej pozícii značkujú alfanumerické reťazce oddelené akoukoľvek medzerou, a tiež rozhrania alfabetického a iného znaku, azda s výnimkou konečnej pevne definovanej množiny ustálených „nedeliteľných“ spojení [Hajič 2001]. Tento prístup je výhodný, pretože tvary rozdelené tokenizáciou sa dajú neskôr spojiť v ďalších fázach analýzy textu.

Softvérové nástroje na segmentáciu a tokenizáciu zahŕňajú pomerne širokú škálu pravidlovo orientovaných systémov. V týchto nástrojoch používateľ pomocu formálnej gramatiky definuje pravidlá a ohraničenia na identifikáciu žiadaných tvarov reťazcov v texte. Nástroj potom vygeneruje vykonateľný alebo zdrojový kód, ktorý je priamo použiteľný na segmentáciu a tokenizáciu podľa zadaných pravidiel. Medzi takéto nástroje patria napríklad:

- *JavaCC* (https://javacc.dev.java.net), známy a často používaný generátor textových analyzátorov pre platformu Java,
- *SableCC* (http://sablecc.org), objektovo orientovaný systém na generovanie textových analyzátorov,
- *JFlex* (http://www.jflex.de), pravidlový generátor textových analyzátorov,
- *FLEX* (http://flex.sourceforge.net), generátor textových analyzátorov pre platformu C.

Hotové riešenie na segmentáciu a tokenizáciu anglických textov ponúka vyhľadávací systém *Apache Lucene* (http://lucene.apache.org). K dispozícii sú štyri typy nástrojov [Hatcher, Gospodnetić 2004]:

- *Whitespace Analyzer* – jednoduchý textový analyzátor, ktorý delí vstupný text na tokeny ohraničené prázdnymi znakmi:

```
[1.][Podpora][toku][znalostí][(procesný][uhol][pohľadu)]
```

- *Simple Analyzer* – textový analyzátor, ktorý robí konverziu na malé písmená a delí vstupný text na tokeny ohraničené prázdnymi znakmi a znakmi interpunkcie:

```
[1][podpora][toku][znalostí][procesný][uhol][pohľadu]
```

- *Stop Analyzer* – textový analyzátor, ktorý robí konverziu na malé písmená a eliminuje stop-slová (pre angličtinu):

```
[1][podpora][toku][znalostí][procesný][uhol][pohľadu]
```

  resp. pre anglický text:

```
The quick brown fox jumps over the lazy dog.
```

  bude výsledok tokenizácie:

```
[quick][brown][fox][jumps][over][lazy][dog]
```

- *Snowball Analyzer* – textový analyzátor, ktorý aplikuje algoritmus na izoláciu koreňa (*stemming* - pre angličtinu):

```
[the][quick][brown][fox][jump][over][the][lazy][dog]
```

Vo všeobecnosti, takmer bez ohľadu na použitý nástroj, je výsledkom tokenizácie súbor identifikovaných a ohodnotených tokenov. Pre ďalšie

spracovanie býva súbor tokenov najčastejšie vyjadrený vo formáte XML [Hajič 2001]. Tokenizovaný fragment textu z príkladu na Obr. 2.4 bude mať tvar:

```
<f type=num mod=conjoin pos=1>1.</f>
<f mod=firstcapital pos=4>podpora</f>
<f pos=12>toku</f>
<f pos=17>znalostí</f>
<f type=spec_opparent pos=26>(</f>
<f pos=27>procesný</f>
<f pos=36>uhol</f>
<f pos=41>pohľadu</f>
<f type=spec_clparent pos=48>)</f>
```

Značka `<f>` označuje lexikálnu formu (*word form*), ktorej atribútmi sú pozícia, typ tokenu, údaj o modifikácii (napr. konverzia veľkého začiatočného písmena, zloženie z viacerých elementárnych jednotiek), prípadne ďalšie parametre.

Token je teda reprezentovaný pozíciou výskytu v texte, svojím pôvodným tvarom v texte a modifikovaným tvarom. Niekedy sa do XML formátu tokenizovaného textu pridávajú aj údaje o začiatku a konci vety [Hajič 2001]. V neskoršom procese lematizácie a morfologickej analýzy sa tokenu priraďujú atribúty *lema* a *tag* [Garabík a kol. 2004].

2.3 Lematizácia a morfologická analýza

Keďže v texte sa jednotlivé slová vyskytujú v rôznych morfologických tvaroch (pádoch, osobách, číslach, atď.) , je nutné ich prevádzať na základné tvary, tzv. *lemy* [Furdík 2003; Forróová a kol. 2003]. Podľa [Forróová a kol. 2003] je lema základným, "slovníkovým" tvarom tokenu, čiže v tomto zmysle je totožná s lexikálnou jednotkou (napr. *pekná* → *pekný*, *prípadov* → *prípad*, atď.). Pri substantívach a adjektívach je to prvý pád jednotného čísla, pri slovesách neurčitok. Proces, ktorý z tvaru slova v texte určí základný tvar (najčastejšie odstránením slovotvorných, pádových a iných predpôn a prípon), sa nazýva **lematizácia** [Rychlý 2000; Garabík a kol. 2004].

Špeciálnou formou lematizácie je **izolácia koreňa slova** (angl. *stemming*), pri ktorej sa označkované slovo na danej pozícii nahrádza svojím kmeňovým základom, čiže sa konvertuje na základný tvar. Zo slova (tokenu) sa odstraňujú slovotvorné, pádové a iné predpony a prípony tak, že ostáva iba základ (koreň) slova, ktorý sa identifikuje ako term (t.j. kľúčové slovo):

```
{clos-ing , clos-es , clos-er , clos-e} → clos+0
```

Predpokladá sa pritom, že základný tvar má rovnaký význam ako každý z gramatických tvarov[8]. Pomocou izolácie koreňa slova dochádza k významnej redukcii počtu termov. Táto redukcia sa vykonáva pomocou [Schwarz 2003]:

[8] To však, najmä pri jazykoch s bohatou morfológiou, nemusí byť pravda. Svedčia o tom príklady gramatických tvarov *per-o* / *per-a*, *vi-ť* / *vi-la*, a mnohé ďalšie.

- *slovníka koreňov* – výhodou je minimálna chybovosť metódy, no nevýhodou je rozsiahlosť slovníka a jeho prípadné obmedzenie na špecifický odbor. Významným obmedzením je tiež skutočnosť, že slovník je prakticky vždy neúplný. Nenachádzajú sa v ňom vlastné mená a názvy, ktoré nesú dôležité informácie o obsahu textu a mali by byť zahrnuté medzi termy. Rozpoznávanie vlastných mien a názvov rieši samostatný smer výskumu, tzv. Named Entity Recognition [Cucerzan, Yarowsky 1999] (viď aj v časti 5.3.1).
- *odstránením afixov* t.j. sufixov (prípon) a prefixov (predpôn) – ide o metódu, kde algoritmus je schopný odstraňovať afixy na základe známeho a vopred definovaného zoznamu sufixov a prefixov alebo na základe pravidiel, podľa ktorých sú konkrétne afixy generované.
- *štatistickou metódou* – na základe variety po sebe nasledujúcich grafém (znakov) v slovách, kde sa pomocou frekvencie jednotlivých zhlukov grafém stanovuje, či ide o prefix alebo o sufix. Táto metóda je nezávislá na konkrétnom jazyku textu.

Pravidlové a slovníkové algoritmy na izoláciu koreňa sú silne závislé na použitom jazyku. V angličtine, kde je izolácia koreňa pomerne jednoduchou záležitosťou, je najpoužívanejším *Porterov algoritmus* [Porter 1980]. Je založený na odstraňovaní prípon, pričom využíva pevný zoznam prípon a niektoré ďalšie pravidlá morfológie anglického jazyka. Inicializačné operácie algoritmu zahŕňajú definície znakov pre spoluhlásky `c` a samohlásky `v`, zavedenie premennej `m`, ktorá vyjadruje tzv. „mieru slova“ (*measure of a word* – počet samohláskových skupín v slove) a definície pravidel v tvare:

```
(podmienka) S1 -> S2
```

Podmienková časť pravidla obsahuje logický výraz, pomocou ktorého sa testuje, napríklad, či tvar končí na *-s*, či tvar obsahuje samohlásku, či tvar končí na dvojitú spoluhlásku, a podobne. Vstupom pre Porterov algoritmus sú tokeny anglických slov, výstupom sú určené korene týchto slov (angl. *stems*).

Algoritmus sa následne vykonáva v piatich krokoch:

Krok 1a - odstránenie prípon *-s* a *-es*:

```
SSES -> SS   caresses -> caress
IES  -> I    ponies   -> poni
S    ->      cats     -> cat
```

Krok 1b - odstránenie prípon *-d*, *-ed* a *-ing*:

```
(m>0) EED -> EE   agreed    -> agree
(*v*) ED  ->      plastered -> plaster
(*v*) ING ->      motoring  -> motor
```

Krok 1c - zmena prípony *-y* na *-i*:

```
(*v*) Y -> I      happy -> happi
```

Krok 2 - zmena prípon:

```
(m>0) ATIONAL ->  ATE    relational     ->  relate
(m>0) TIONAL  ->  TION   conditional    ->  condition
(m>0) ENCI    ->  ENCE   valenci        ->  valence
(m>0) ANCI    ->  ANCE   hesitanci      ->  hesitance
(m>0) IZER    ->  IZE    digitizer      ->  digitize
(m>0) ABLI    ->  ABLE   conformabli    ->  conformable
(m>0) ALLI    ->  AL     radicalli      ->  radical
(m>0) ENTLI   ->  ENT    differentli    ->  different
(m>0) ELI     ->  E      vileli       - >  vile
(m>0) OUSLI   ->  OUS    analogousli    ->  analogous
(m>0) IZATION ->  IZE    vietnamization ->  vietnamize
(m>0) ATION   ->  ATE    predication    ->  predicate
(m>0) ATOR    ->  ATE    operator       ->  operate
(m>0) ALISM   ->  AL     feudalism      ->  feudal
(m>0) IVENESS ->  IVE    decisiveness   ->  decisive
(m>0) FULNESS ->  FUL    hopefulness    ->  hopeful
(m>0) OUSNESS ->  OUS    callousness    ->  callous
(m>0) ALITI   ->  AL     formaliti      ->  formal
(m>0) IVITI   ->  IVE    sensitiviti    ->  sensitive
(m>0) BILITI  ->  BLE    sensibiliti    ->  sensible
```

Krok 3 - zmena alebo odstránenie prípon:

```
(m>0) ICATE ->  IC       triplicate  ->  triplic
(m>0) ATIVE ->           formative   ->  form
(m>0) ALIZE ->  AL       formalize   ->  formal
(m>0) ICITI ->  IC       electriciti ->  electric
(m>0) ICAL  ->  IC       electrical  ->  electric
(m>0) FUL   ->           hopeful     ->  hope
(m>0) NESS  ->           goodness    ->  good
```

Krok 4 - odstránenie prípon:

```
(m>1) AL    ->    revival     ->  reviv
(m>1) ANCE  ->    allowance   ->  allow
(m>1) ENCE  ->    inference   ->  infer
(m>1) ER    ->    airliner    ->  airlin
(m>1) IC    ->    gyroscopic  ->  gyroscop
(m>1) ABLE  ->    adjustable  ->  adjust
(m>1) IBLE  ->    defensible  ->  defens
(m>1) ANT   ->    irritant    ->  irrit
(m>1) EMENT ->    replacement ->  replac
(m>1) MENT  ->    adjustment  ->  adjust
(m>1) ENT   ->    dependent   ->  depend
(m>1 and (*S or *T)) ION   ->
                  adoption    ->  adopt
(m>1) OU    ->    homologou   ->  homolog
(m>1) ISM   ->    communism   ->  commun
(m>1) ATE   ->    activate    ->  activ
(m>1) ITI   ->    angulariti  ->  angular
(m>1) OUS   ->    homologous  ->  homolog
(m>1) IVE   ->    effective   ->  effect
(m>1) IZE   ->    bowdlerize  ->  bowdler
```

Krok 5a - úprava koreňov, odstránenie *-e*:

```
(m>1) E ->             probate -> probat
(m=1 and not *o) E ->  cease   -> ceas
```

Krok 5b - úprava koreňov, zmena *-ll* na *-l*:

```
(m>1 and *d and *L) LL -> L     controll -> control
```

Viac informácií o Porterovom algoritme, vrátane hotových implementácií v najpoužívanejších programových prostrediach (Java, ANSI C, C#, Perl, atď.), možno nájsť napríklad na http://tartarus.org/~martin/PorterStemmer/.

Ďalšími používanými nástrojmi sú napríklad *S-stemmer*, *Lovinsov stemmer*, a ďalšie [Sparck-Jones, Willett 1997]. Obyčajne sú tieto algoritmy pomerne jednoduché, rýchle a pre angličtinu aj dostatočne efektívne a presné.

V slovenčine a ďalších flektívnych jazykoch je situácia komplikovanejšia, pravidlá morfológie sú podstatne zložitejšie a izolácia koreňa sa rieši morfologickou analýzou a komplexným lingvistickým prístupom [Furdík 2003]. Výsledky bývajú presnejšie, avšak algoritmus je zložitejší a výrazne výpočtovo náročnejší.

Problémy, s ktorými je potrebné vysporiadať sa pri lematizácii v slovenčine, spôsobujú častné nejednoznačnosti a rôzne typy nepravidelností jazyka, predovšetkým *tvarová homonymia* [Páleš 1994]. Napríklad token *mier* nemá jednoznačnú lemu – môže to byť sloveso *mieriť*, aj substantíva *mier* a *miera*. Je potrebné zistiť okolie (kontext) tokenu v texte, odvodiť slovný druh a morfologické kategórie tokenu, a až potom sa dá presne určiť lema. Podobne nejednoznačné je určenie lemy pri tokenoch *mať* (substantívum *matka* vs. sloveso *mať*), *veď* (sloveso *viesť* vs. častica *veď*), *otcov* (substantívum *otec* vs. privlastňovacie adjektívum *otcov*), a mnohé ďalšie.

Na lematizáciu a morfologickú analýzu textov v slovenčine bolo vytvorených niekoľko nástrojov [Laclavík a kol. 2007], napríklad:

- *Minimálny automat na separáciu koreňov* [Páleš 1994] – pravidlový systém, do istej miery podobný Porterovmu algoritmu. Vyhodnocovaním zhody začiatkov a zakončení reťazca tokenu s vopred danou štruktúrou predpôn a prípon sa generujú hypotézy o rozdelení na koreň a axify (prípony a predpony). Algoritmus sa správa ako nedeterministický konečný automat, pretože vytvorené hypotézy je často potrebné overiť ďalšou morfologickou analýzou.
- *Lematizátor L. Galamboša* [Galamboš 2001], navrhnutý všeobecne pre slovanské jazyky, s implementáciou pre poľský jazyk. Lematizátor je integrovaný do systému na vyhľadávanie informácií Egothor (http://www.egothor.org).
- *Tvaroslovník* – lematizátor slovenského jazyka, vytvorený na UPJŠ v Košiciach v rámci projektu NAZOU [Krajči a kol. 2009]. Na základe vopred definovaných vzorov prípon sa pre vstupný token generuje lema, ktorá sa následne kvôli presnosti overuje v slovníku.

- *Morfologický analyzátor Slovenského národného korpusu*, ktorý kombinuje pravidlá na elimináciu axifov s rozsiahlym slovníkom paradigiem substantív, adjektív a slovies. Analyzátor sa využíva na automatizovanú anotáciu textov v Slovenskom národnom korpuse [Garabík a kol. 2004].

Po segmentácii a tokenizácii, popísanej v predchádzajúcej časti 2.2, pokračuje predspracovanie textu na úrovni slov morfologickou analýzou, v rámci ktorej sa vykonáva morfologická analýza tokenu, izolácia koreňa a lematizácia[9]. Slovám identifikovaným v texte počas tokenizácie sa priradí príslušný základný tvar – lema. Zároveň sa slová ohodnocujú príslušnými morfologickými kategóriami, na základe ktorých sa tokenom priradia gramatické značky – tagy. Značka určuje predovšetkým slovný druh, a potom, v závislosti od slovného druhu, aj ďalšie kategórie ako rod, číslo, pád, osobu, atď.

Reprezentácia morfologických tagov je daná použitým súborom značiek a spôsobom ich zápisu. Súbory značiek pre češtinu, tzv. *tagsety*, vznikali pri tvorbe a anotácii lingvistických korpusov na pracoviskách v Prahe a v Brne [Rychlý 2000; Hajič 2001]. Pražský systém značiek je pozičný, kde každý z typov gramatických značiek má definovanú vlastnú pevnú pozíciu v reťazci 15 znakov. Brnenský spôsob značkovania je kódovací, typy značiek sú vyjadrené postupnosťou dohodnutých kódov atribútov pozostávajúcich vždy z dvoch znakov. Prvý znak (malé písmeno) reprezentuje morfologický atribút a druhý znak (veľké písmeno alebo číslica) hodnotu atribútu.

Nasledujúci výpis ukazuje príklad morfologického značkovania českej vety „*Na okně seděla kočka, byl horký letní den.*“ pomocou pražského a brnenského systému gramatických značiek [Rychlý 2000]:

```
Token    Lemma     pražské značkovanie       brnenské
značkovanie
Na       na        RR-------------           k7c6
okně     okno      NNNS6-----A----           klgNnSc6
seděla   sedět     VpQW---XR-AA---           k5eAp3nStMmPaI
kočka    kočka     NNFS1-----A----           klgFnSc1
,                  Z:-------------
byl      být       VpYS---XR-AA---           k5eAp1nStMmPaI
horký    horký     AAIS1----1A----           k2eAgInSc1d1
letní    letní     AAIS1----1A----           k2eAgInSc1d1
den      den       NNIS1-----A----           klgInSc1
.                  Z:-------------
```

[9] Pri predspracovaní anglických textov sa lematizácia a morfologická analýza nahrádza procesom identifikácie slovných druhov (angl. *Part-of-speech tagging*, *POS tagging*). Angličtina má, v porovnaní s flektívnymi jazykmi (slovenčina, čeština) morfologický systém pomerne chudobný, lemy sú identifikované už v procese separácie koreňov. Pre riešenie nejednoznačností vyplývajúcich zo slovnodruhovej homonymie je však v angličtine potrebné analyzovať text na úrovni syntaxe. Slovné druhy pre jednotlivé tokeny sa určujú na základe poradia a vzájomných vzťahov slov vo vetách analyzovaného textu.

Význam značiek použitých v príklade pražského značkovania:

1. pozícia – slovný druh: N – substantívum, A – adjektívum, V – sloveso, R - predložka, Z – interpunkcia, hranica vety.
2. pozícia – bližšie určenie slovného druhu: N – obyčajné substantívum, A – obyčajné adjektívum, p – slovesný tvar minulého príčastia aktívneho, R – obyčajná predložka, : - všeobecná interpunkcia.
3. pozícia – menný rod: I – mužský rod neživotný, Y – mužský rod, N – stredý rod, F – ženský rod, Q – ženský rod singuláru alebo stredný rod plurálu.
4. pozícia – číslo: S – singulár, W – singulár pre ženský rod a plurál pre stredný rod.
5. pozícia – pád: 1 – nominatív, 6 – lokatív.
8. pozícia – osoba: X – ľubovoľná osoba (1/2/3).
9. pozícia – čas: R – minulý čas.
10. pozícia – stupňovanie: 1 – 1. stupeň.
11. pozícia – negácia: A – afirmatív (bez predpony *ne*-).
12. pozícia – aktívum / pasívum: A – aktívum.

Význam značiek použitých v príklade brnenského značkovania:

1. slovný druh: k1 – substantívum, k2 – adjektívum, k5 – sloveso, k7 – predložka.
2. rod: gI – mužský neživotný, gF – ženský, gN – stredný.
3. číslo: nS – jednotné (singulár).
4. pád: c1 – nominatív, c6 – lokatív.
5. osoba: p1 – 1. osoba.
6. čas: tM – minulý.
7. spôsob: mP – particípium (neurčitý slovesný tvar, príčastie).
8. vid: aI – imperfektum (nedokonavé).
9. stupňovanie: d1 – nominatív.

Tagset pre slovenčinu bol vyvinutý na pôde JÚĽŠ v Bratislave, ako súčasť projektu Slovenského národného korpusu (http://korpus.juls.savba.sk). Koncepčne slovenský morfologický tagset vychádza z brnenského kódového značkovania, líši sa však spôsob zápisu aj množina atribútov tagu. Každý z atribútov tagu je kódovaný iba na jednej pozícii, a to písmenom latinskej abecedy, číslicou alebo matematickým symbolom. Súbor týchto atribútov tvorí jeden tag (značku) a je priradený k jednému tokenu a leme. Počet znakov je variabilný, ale ich poradie v tagu je záväzné [Garabík a kol. 2004].

Každý tag sa skladá z dvoch častí. Prvá časť určuje morfologické a gramatické kategórie tokenu, druhá (nepovinná) časť zaraďuje token

do určitých špeciálnych skupín (vlastné mená, defektné zápisy)[10].

Prvá časť tagu sa začína vždy znakom pre slovnú triedu. Slovenský tagset rozlišuje 19 slovných tried, z ktorých 10 zodpovedá tradičným slovným druhom a 9 obsahuje špecifické jazykové prvky: príčastia, reflexívne zámená *sa*/*si*, podmienková morféma *by*, interpunkcia, skratky a značky, neurčiteľný slovný druh, neslovný element, citátový výraz, číslice. Ďalšie atribúty tagu sa dopĺňajú morfologickými kategóriami v závislosti od slovnej triedy.

Morfologickú analýzu, ktorá týmto spôsobom označkuje tokenizovaný text, možno vo všeobecnosti formálne popísať ako matematickú funkciu, ktorá postupnosti tokenov priraďuje množinu možných výsledkov zložených z dvojíc *<lema l, značka t>*:

$$MorfAn(f) \rightarrow \{<l,t>;\ l \in L,\ t \in T\},$$

kde $f \in A^+$ je slovný tvar tokenu zložený zo znakov abecedy A analyzovaného jazyka, L je množina identifikácií lem a T je množina značiek používaných pre tagovanie. Výstup z morfologickej analýzy sa dá zapísať do formátu XML rozširujúceho špecifikáciu jednotlivých tokenov o značku `<l>` pre lemu a značku `<t>` pre token. XML výstup pre fragment vety z príkladu na Obr. 2.4, analyzovaný pomocou slovenského tagsetu, bude mať tvar:

```
<f>1.</f>        <l>prvý</l>      <t>0</t>
<f>podpora</f>   <l>podpora</l>   <t>SSfs1</t>
<f>toku</f>      <l>tok</l>       <t>SSis2</t>
<f>znalostí</f>  <l>znalosť</l>   <t>SSfp2</t>
<f>(</f>         <l>(</l>         <t>#</t>
<f>procesný</f>  <l>procesný</l>  <t>AAis1x</t>
<f>uhol</f>      <l>uhol</l>      <t>SSis1</t>
<f>pohľadu</f>   <l>pohľad</l>    <t>SSis2</t>
<f>)</f>         <l>)</l>         <t>#</t>
```

Význam značiek tagu `<t>`, použitých v príklade slovenského značkovania:

1. slovná treda: S – substantívum, A – adjektívum, 0 – číslica, # – neslovný element.
2. typ paradigmy (pri menných slovných druhoch): S – substantívna paradigma, A – adjektívna paradigma.
3. rod: i – mužský neživotný, f – ženský.
4. číslo: s – jednotné (singulár), p – množné (plurál).
5. pád: 1 – nominatív, 2 – genitív.
6. stupňovanie: x – pozitív (irelevantný stupeň).

Morfologické analyzátory, pomocou ktorých sa vykonáva proces lematizácie a tagovania, sú zväčša automatické a pracujú buď ako konečné stavové automaty, alebo ako tzv. sekvenčné stroje (konečné prevodníky). Súbory

[10] V lingvistickom korpuse sa skúmajú a zaznamenávajú údaje o (ne-)spisovnosti slov a výrazov (napr. *neni*, *do Košicoch*, *za prvé*, *postavím sa do rady*, *prádlo*). Defektné výrazy sa v korpuse neopravujú, ale zachovávajú sa v pôvodnom tvare. Pre aplikácie dolovania znalostí z textov údaj o defektnosti výrazu nemá veľké opodstatnenie. Avšak údaj o zaradení tokenu medzi vlastné mená je dôležitý a je vhodné ho do tagu zaznamenať.

značiek a pravidlá bývajú uložené v špecializovanom morfologickom slovníku. Výsledky takejto analýzy však nie sú jednoznačné, čo vyplýva už z povahy samotných morfologických jazykových javov [Furdík 2003; Páleš 1994]. Často býva jeden token morfologicky ohodnotený viacerými značkami, z ktorých správna však môže byť iba jedna, pretože analyzovaný text je už konkrétnou realizáciou jazykového systému. Po prvotnej morfologickej analýze teda nevyhnutne nasleduje proces *zjednoznačnenia* (angl. *disambiguation*), ktorý už môže, na rozdiel od samotnej morfologickej analýzy, využiť aj kontext, v ktorom sa slovo nachádza. Na zjednoznačnenie sa používajú štatistické metódy, založené na strojovom učení [Hajič 2001]. Východiskom je vopred *ručne* označkovaný korpus, pre flektívne jazyky zhruba v rozsahu 1,5 milióna slov (čo zodpovedá chybovosti cca 5%), ktorý slúži ako trénovacia množina textov. Potom sa môže použiť *metóda podmienených pravdepodobností* spočítavajúca relatívne početnosti po sebe nasledujúcich morfologických značiek pre tokeny z textov v trénovacej množine. Tým sa vytvorí pravdepodobnostný klasifikátor, ktorý je schopný do určitej miery (resp. s istou tolerovateľnou chybou) odstrániť nejednoznačnosti v ostatných morfologicky anotovaných textoch korpusu. Na úplné a korektné odstránenie nejednoznačností je však potrebné použiť lingvistický prístup k analýze textu (viď v časti 2.6).

2.4 Eliminácia neplnovýznamových slov

Všetky identifikované a lematizované tokeny sú kandidátmi pre termy vektorového modelu textového dokumentu. Termy, čiže kľúčové slová, by mali čo najpresnejšie vyjadrovať obsah daného dokumentu. Zároveň je pre efektívnosť ďalšieho spracovania pomocou klasifikačných a zhlukovacích algoritmov výhodné minimalizovať počet termov popisujúcich jednotlivé dokumenty. Tieto dve skutočnosti sú dôvodmi pre to, aby sa z ďalšieho spracovania vylúčili tokeny s malým príspevkom k celkovému obsahu textu.

Vo všeobecnosti sa predpokladá, že hlavnými nositeľmi obsahu textu sú plnovýznamové slová, predovšetkým substantíva a adjektíva (najmä v prívlastkových konštrukciách typu *tok znalostí*, *uhol pohľadu*, *znalostný sklad*, a podobne). Lematizované tokeny, ktoré vznikli z týchto slov, sú vhodné na reprezentáciu obsahu dokumentov a majú sa transformovať na termy.

Naopak, minimálny a zanedbateľný prínos k obsahu textu sa predpokladá pri neplnovýznamových slovách – spojkách, predložkách, zámenách, časticiach, rôznych netextových elementoch a sčasti aj pri číslicach a číslovkách. Tieto takzvané *stop-slová* (angl. *stop-words*, niekedy tiež *noise-words*) sa obyčajne v texte vyskytujú s vyššou frekvenciou, pričom však k celkovému obsahu textu prispievajú iba malým informačným ziskom. Tokeny vzniknuté z týchto slov je vhodné vylúčiť zo zoznamu termov.

Typické stop-slová pre angličtinu sú napríklad:

```
a, about, above, after, again, an, and, any, are, be,
before, behind, been, both, brief, can, come, did, didn't,
down, during, each, else, et, etc, except, far, for, from,
get, go, got, had, half, has, have, he, hello, his, how,
in, into, it, just, last, let, low, me, most, much, my,
```

```
near, no, none, not, now, of, off, on, our, out, own, past,
per, rather, recent, say, see, self, she, so, soon, such,
take, than, that, the, their, then, there, they, this, to,
too, try, under, up, upon, us, use, via, want, was, we,
were, what, when, who, why, would, yes, yet, ...
```

Pre slovenčinu sú stop-slová napríklad:

```
a, aby, aj, ako, ale, alebo, ani, áno, asi, bez, by, byť,
cez, čo, či, dnes, do, ďalší, ešte, ho, i, iba, ja, je,
jeho, jej, k, kam, každý, kde, kto, ktorý, ku, mať, môcť,
môj, my, na, nad, nie, niet, než, nič, nový, o, od, on, po,
pod, podľa, práve, prečo, pred, preto, potom, pri, prvý, s,
sa, si, so, späť, svoj, tak, takže, teda, ten, tento, to,
toto, tu, tuto, tvoj, ty, u, už, v, váš, viac, však,
všetko, vy, z, za, že, ...
```

Štandardizovaný a všeobecne akceptovaný zoznam stop-slov neexistuje, pretože zaradenie toho-ktorého slova medzi stop-slová je častokrát závislé od aplikácie alebo domény, ktorej sa analyzované texty týkajú. Napriek tomu existujú voľne prístupné zoznamy stop-slov pre niektoré jazyky, napríklad:

- angličtina: http://www.link-assistant.com/seo-stop-words.html
- nemčina: http://www.ranks.nl/stopwords/german.html
- čeština: http://seo-servis.cz/libs/stopwords.txt.cz
- poľština: http://pl.wikipedia.org/wiki/Wikipedia:Stopwords

Pre slovenčinu, podľa našich vedomostí, nie je všeobecný zoznam stop-slov voľne k dispozícii.

Eliminácia stop-slov sa uplatňuje pri predspracovaní textov pre algoritmy dolovania znalostí, a tiež pre systémy vyhľadávania informácií, kde zoznam stop-slov býva vopred definovaný a implementovaný ako súčasť vyhľadávacieho stroja [Slawski 2008]. Odstraňovanie stop-slov sa nepoužíva v korpusovej lingvistike, ktorá skúma jazykové javy v celej ich komplexnosti. Neplnovýznamové slová majú dôležitú úlohu najmä na úrovni syntaxe.

Obyčajne sa pri vyhľadávaní informácií aj pri dolovaní znalostí stop-slová odstraňujú dvoma základnými spôsobmi. Prvý pôsob používa vopred ručne zostavený zoznam neplnovýznamových slov, tzv. *negatívny slovník* (angl. *stop-words list*). Z tokenizovaného textu sa odstránia slová nachádzajúce sa v negatívnom slovníku. Efektivita tohto prístupu závisí od úplnosti zoznamu a jeho nevýhodou je to, že je závislý na použitom jazyku.

Druhý spôsob je automatický, odstraňuje z textu slová, ktoré sa v celom korpuse analyzovaných dokumentov vyskytujú s príliš veľkou (a aj s príliš malou) frekvenciou. Podľa [Salton, Wong, Yang 1975] majú dostatočne silnú rozlišovaciu schopnosť tie slová, ktorých frekvencia výskytu v celom korpuse patrí do intervalu $<\frac{N}{100},\frac{N}{10}>$, kde N je počet dokumentov. Tento spôsob je použiteľný pre väčšinu jazykov, v praxi však vykazuje slabšie výsledky ako prvý prístup. Optimálne výsledky sa dosahujú kombináciou obidvoch prístupov.

Predpokladajme, že zoznam stop-slov (vytvorený vopred ako negatívny slovník alebo určený na základe štatistickej analýzy výskytov slov v korpuse dokumentov) obsahuje, okrem iných, aj tvary číslic a netextových elementov:

```
[1] [1.] [2] [2.] [.] [,] [(] [)] [/] [*] ...
```

Potom vo fragmente vety z príkladu na Obr. 2.4 sa elimináciou stop-slov odstránia tokeny `<f>1.</f>`, `<f>(</f>` a `<f>)</f>`. Výsledný vektor termov je daný lemami zostávajúcich tokenov a bude mať tvar:

```
[podpora] [tok] [znalosť] [procesný] [uhol] [pohľad]
```

Vylúčením stop-slov sa dosahuje zníženie počtu termov reprezentujúcich dokumenty, čím sa redukuje príznakový priestor a zefektívňuje sa činnosť algoritmov klasifikácie, zhlukovania alebo vyhľadávania informácií. Nevýhodou je strata určitej časti informácií z textu, napríklad ak sa odstráni chybne detekované stop-slovo (t.j. slovo, ktoré v danom kontexte nie je stop-slovom). Tiež celkom neplatí, že stop-slová nenesú žiadnu informáciu o obsahu textu. Neplnovýznamové (pomocné) slová sa zúčastňujú na jazykovom prejave a často v ňom plnia dôležitú funkciu – napríklad modifikujú význam okolitých plnovýznamových kontextovo zapojených slov. Čiastočne sa táto druhá nevýhoda dá eliminovať použitím vhodného frázového slovníka, v úplnosti sa však tento problém rieši komplexnou syntakticko–sémantickou analýzou (viď v časti 2.6).

2.5 Váhovanie a normovanie termov

Identifikované a lematizované termy v reprezentácii dokumentu je potrebné ohodnotiť dodatočnými charakteristikami vyjadrujúcimi dôležitosť toho, ktorého termu v rámci daného dokumentu aj v rámci korpusu ako celku. Spôsob tohto ohodnotenia je daný použitým modelom pre reprezentáciu textových dokumentov (viac o modeloch ďalej, v časti 2.7). Techniky *váhovania a normovania termov* zabezpečujú takéto dodatočné ohodnotenie a dovoľujú tým zvýšiť efektívnosť procesu získavania znalostí z textov, t.j. klasifikáciu, zhlukovanie, vyhľadávanie a extrakciu informácií.

Pod **váhovaním** rozumieme úpravu frekvencie termov každého dokumentu ktorý sa nachádza v korpuse dokumentov. Toto váhovanie podľa [Dumais 1991] prebieha v dvoch základných rovinách. Prvou rovinou je váhovanie na základe počtu výskytov v samotnom dokumente – *lokálne váhovanie* $L(k_t, d_i)$ a druhou rovinou je *globálne váhovanie* $G(k_t)$. Váhovaná frekvencia termu k_t v dokumente d_i je potom súčinom lokálnej váhy a globálnej váhy, teda:

$$w_{it} = L(k_t, d_i) \times G(k_t)$$

Lokálna váha je špecifická pre term aj dokument. Vo všeobecnosti pre dva dokumenty d_x a d_y, $d_x \neq d_y$, platí, že aj $L(k_t, d_x) \neq L(k_t, d_y)$. Tri najpoužívanejšie lokálne váhovania sú:

- $L(k_t, d_i) = \begin{cases} 1 & \text{ak sa daný term v dokumente vyskytuje;} \\ 0 & \text{inak, čo je tzv. binárne váhovanie.} \end{cases}$
- $L(k_t, d_i) = tf_{it} = \frac{freq_{it}}{\max freq_{it}}$ je normovaný počet výskytov termu k_t v dokumente d_i ($\max freq_{it}$ je frekvencia najčastejšie sa vyskytujúceho termu k_t v dokumente d_i)

- $L(k_t, d_i) = log_2(tf_{it} + 1)$ – term je reprezentovaný logaritmom výskytu, čo tlmí veľké rozdiely vo frekvenciách

Druhou rovinou je *globálne váhovanie* $G(k_t)$, ktoré určuje, aký významný je term v celom korpuse dokumentov. Globálna váha termu $G(k_t)$ je pre každý dokument rovnaká. Štyri bežne používané globále váhovania sú: *Norm*, *GfIdf*, *Idf*, a *Entrópia* (resp. *1 – Entropy*), kde:

$$G(k_t) = Norm(t) = \frac{1}{\sqrt{\sum_{i=1}^{N} tf_{it}}} \tag{2.1}$$

$$G(k_t) = GfIdf(t) = \frac{gf_t}{df_t} \tag{2.2}$$

$$G(k_t) = Idf(t) = \log_2\left(\frac{N}{df_t}\right) \tag{2.3}$$

$$G(k_t) = 1 - Entropy(i) = 1 - \sum_{i=1}^{N} \frac{P_{it} \log_2(P_{it})}{\log_2(N)}, \text{ kde } P_{it} = \frac{tf_{it}}{gf_t} \tag{2.4}$$

pričom:

- N je počet dokumentov v množine spracovávaných dokumentov.
- tf_{it} je termová frekvencia, čo je počet výskytov termu k_t v dokumente d_i,
- df_t je dokumentová frekvencia, čo je počet dokumentov, v ktorých sa term k_t vyskytuje,
- gf_t je globálna frekvencia, teda počet výskytov termu k_t v celom korpuse dokumentov,
- P_{it} je pomer počtu výskytov termu k_t v dokumente d_i k počtu výskytov tohto termu v celom korpuse dokumentov.

Tieto globálne váhy zmenšujú dôležitosť tých termov, ktoré sa vyskytujú vo väčšej časti dokumentov. Podľa [Dumais 1991] sú pritom *GfIdf*, *Idf* a *Entrópia* kvalitatívne na približne rovnakej úrovni. Váhovanie *Norm* (normovanie veľkosti vektora termu) tým, že kladie na rovnakú úroveň termy vyskytujúce sa vo veľkom počte dokumentov s termami s vysokou frekvenciou v jednom dokumente, je kvalitatívne horšie.

2.6 Lingvistický prístup k analýze textu

Premisou vyššie popísaného prístupu k predspracovaniu textov a výberu termov je téza, že „*text hovorí o tom, o čom sú slová v tomto texte*“, respektíve, že obsah textu je zložený z plnovýznamových slov nachádzajúcich sa v tomto texte. Tento predpoklad je do určitej miery správny – obsah textu nemôže byť úplne odlišný od významov slov, ktoré tento text tvoria. Avšak obsah textu je zložený z významov jednotlivých slov

v kontextových vzťahoch, je teda vždy viac, ako iba sumár typických významov týchto slov (podobne ako dom nie je iba „kopa tehál“) [Furdík 1998]. Ak sa pri predspracovaní nezohľadnia aj kontextové vzťahy v jazykovom prejave, môže to negatívne ovplyvniť kvalitu a úspešnosť algoritmov dolovania znalostí z textov a vyhľadávania informácií.

V texte v prirodzenom jazyku sa dajú identifikovať nasledujúce javy, ktoré odrážajú kontextovú zapojenosť jazykových prvkov a sú dôležité pre výber termov reprezentujúcich obsah textu:

- *Zámená, kontextové odkazy*. V texte sa pomerne často vyskytujú konštrukcie obsahujúce odkazy na objekty z predchádzajúceho alebo nasledujúceho kontextu. Tieto odkazy sa väčšinou, hoci nie výlučne, realizujú pomocou zámen. Napríklad vo vete „*Znalosti programátorov by vo firme ostali aj po ich odchode*.“ zámeno *ich* odkazuje na *programátorov*. To však znamená, že vektor termov pre túto vetu by namiesto lemy zámena *ich*, teda tvaru *on*, mal obsahovať dvakrát substantívum *programátor*. Priradiť zámenu príslušnú frázu alebo objekt, na ktorý sa toto zámeno odkazuje, však vyžaduje komplexnú syntaktickú a sémantickú analýzu textu.
- *Synonymia*. Slová s odlišnou formou a rovnakým alebo podobným významom (napr. *kniha*, *publikácia*) by sa mali reprezentovať tým istým termom. Na identifikáciu podobnosti významov nepostačuje skúmanie na úrovni morfológie ani syntaxe, potrebná je slovotvorná, sémantická a pragmatická analýza.
- *Homonymia, polysémia*. Slová s náhodne totožnou formou a s rôznym významom (napr. *jazyk* [reč] vs. *jazyk* [v ústach] vs. *jazyk* [na topánke], *akcia* [činnosť] vs. *akcia* [cenný papier], *kurz* [jazykový] vs. *kurz* [meny]) by sa mali podľa kontextu rozlíšiť a reprezentovať vzájomne rôznymi termami. Na rozlíšenie polysémických slov treba uskutočniť syntaktickú a sémantickú analýzu textu.
- *Frázy, viacslovné ustálené pomenovania*. Slovné spojenia so špecifickým významom, napríklad *Technická univerzita v Košiciach*, *manažment znalostí*, *expertný systém*, a podobne, majú charakter samostatných termínov. Ich celkový význam je iný ako významy slov, z ktorých sa tieto termíny skladajú. Bolo by teda vhodné, aby sa ustálené viacslovné pomenovania reprezentovali jedným viacslovným termom. Identifikácia viacslovných pomenovaní v texte, vrátane príslušných gramatických modifikácií, predpokladá integrovanú morfologickú a syntaktickú analýzu.
- *Porozumenie významu textu*. Ak by analyzovaný text obsahoval tvrdenie „*Tento text sa netýka manažmentu znalostí*“, tak by reprezentácia obsahu tohto textu nemala zahŕňať termy *manažment*, *znalosti*, resp. *manažment znalostí*. Takýto stupeň porozumenia významu textu, ak sa vôbec v súčasnosti dá dosiahnuť, by si vyžadoval úplnú sémantickú a pragmatickú analýzu, s využitím znalostí mimojazykových skutočností.

Naviac, pri predspracovaní textu vo fázach tokenizácie a najmä lematizácie vznikajú nejednoznačnosti (viď v časti 2.3), ktoré nie je možné riešiť bez

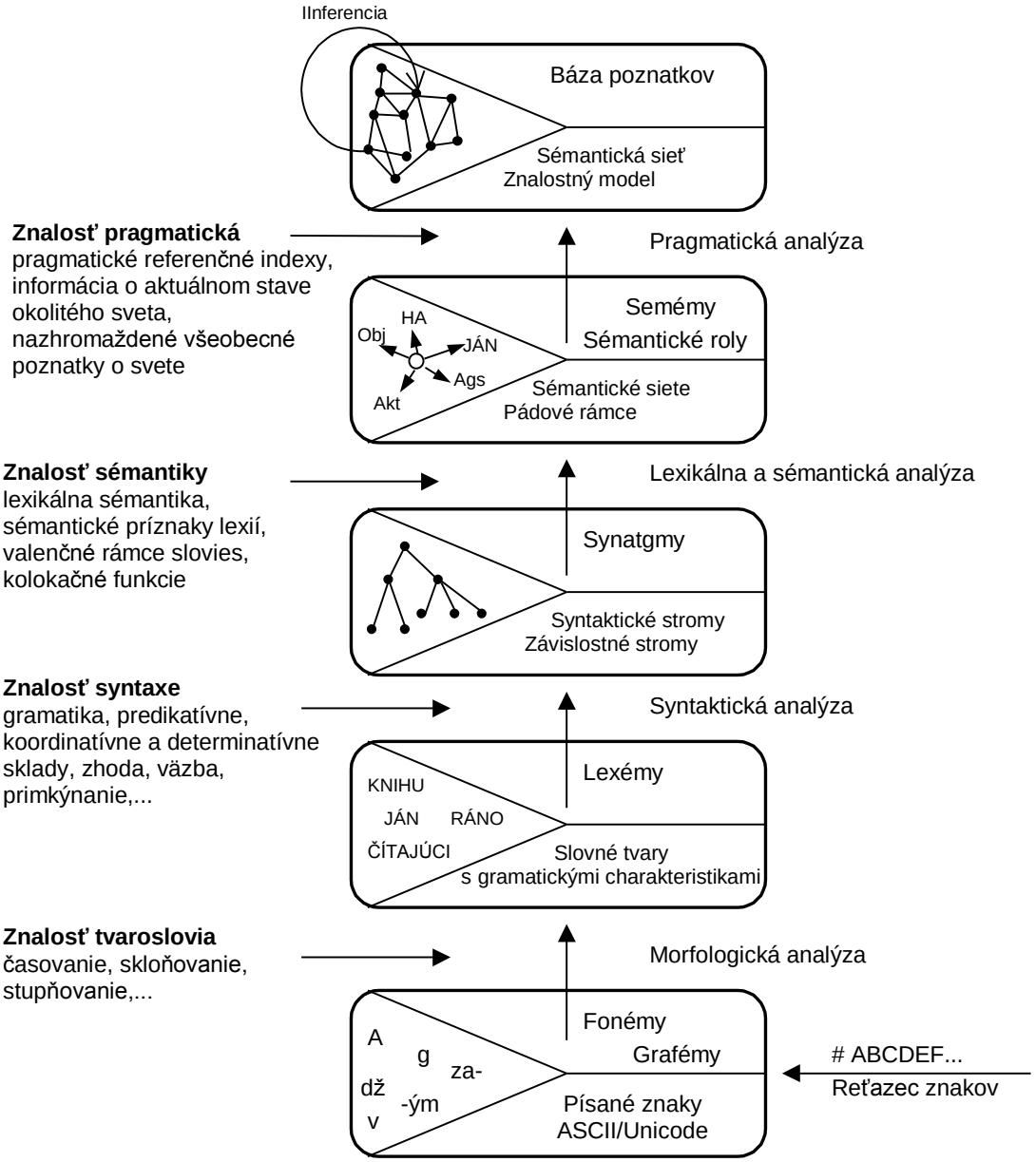
zohľadnenia kontextu a analýzy syntaktických, prípadne aj sémantických vzťahov. Majme napríklad text obsahujúci vetu „*Cieľom štúdie je preskúmať účinnosť navrhovaných mier.*“. Nejednoznačnosť vzniká pri pokuse lematizovať token *mier* (možné lemy: *mier*, *miera*, *mieriť*). Na jednoznačné rozhodnutie, ktorá z možných lem je správna, je potrebné zistiť syntaktickú štruktúru danej vety, z nej odvodiť prípustné gramatické kategórie vetných členov a na ich základe určiť správnu lemu. Slovo *mier* je v danej vete prívlastok, ktorý je rozvitý ďalším zhodným prívlastkom *navrhovaných*. Pre tvar *navrhovaných* sa určia gramatické kategórie – adjektívum, genitív množného čísla. Pravidlo o zhode medzi nadradeným vetným členom a zhodným prívlastkom predpisuje, že tvar *mier* má byť tiež genitív množného čísla – túto podmienku spĺňa lema *miera*, ktorá sa vyberie ako správna.

Riešiť tieto jazykové javy a nejednoznačnosti môže pomôcť *lingvistický prístup* k analýze textu. V texte skúmaného dokumentu sa identifikujú *jazykové jednotky* (t.j. tokeny ako prvky zložitého jazykového systému), ktoré synergickým efektom tvoria význam a obsah textu [Furdík 2001]. Jazykovou jednotkou môže byť niektorý z elementov podľa rozdelenia na jednotlivé jazykové roviny – fonéma, morféma, lexéma, syntagma, sémantický príznak, a podobne [Páleš 1994; Furdík 2001]. Ambíciou tohto prístupu je komplexná analýza textu na základe niektorej jazykovednej, lingvistickej teórie.

Proces predspracovania textu pomocou lingvistického prístupu pozostáva z čiastkových analýz vykonávaných podľa jednotlivých jazykových rovín – analýzy *fonologickej*, *morfologickej*, *lexikálnej*, *syntaktickej*, *sémantickej* a *pragmatickej*. Výsledkom je hĺbková reprezentácia textu, v ktorej sú zaznamenané všetky z hľadiska obsahu a jazykovej stavby relevantné údaje – sú jednoznačne identifikované jazykové jednotky ako tokeny vo vzájomných vzťahoch, sú nastavené ich parametre a sú vytvorené väzby medzi tokenmi na základe ich významu a funkcie v texte. Opačný proces, pri ktorom sa z hĺbkových (sémantických) štruktúr generujú lineárne reťazce jazyka, sa nazýva *jazyková syntéza*.

Proces jazykovej analýzy textu sa obyčajne vykonáva v sekvencii, postupnosti od najnižšej, fonologickej roviny, až po analýzu na rovine sémantickej a pragmatickej. Výsledky analýzy na nižšej úrovni tvoria pritom vstup pre analýzu na úrovni o jednu rovinu vyššej. Takáto postupná jazyková analýza vykonávaná na báze lingvistických zákonitostí sa podľa [Páleš 1994] nazýva *sekvenčná stratifikačná jazyková analýza*. Jej model, upravený pre reprezentáciu obsahu textov v systémoch pre vyhľadávanie informácií, je zobrazený na Obr. 2.6. Keďže ide o problematiku prekračujúcu rámec a ciele tejto publikácie, uvádzame len zjednodušenú grafickú prezentáciu celého prístupu so stručnými textovými vysvetleniami priamo v prezentovanej schéme na Obr. 2.6.

Fonologická analýza sa obyčajne pri predspracovaní písaných textov vynecháva, alebo sa redukuje na zjednodušenú sústavu pravidiel spájania foném, respektíve znakov. Tieto pravidlá sa potom využívajú počas morfologickej analýzy. Príkladom takýchto pravidiel sú podmienky pre striedanie samohlások, spoluhlások a hláskových skupín v Porterovom algoritme na izoláciu koreňa (viď v časti 2.3).

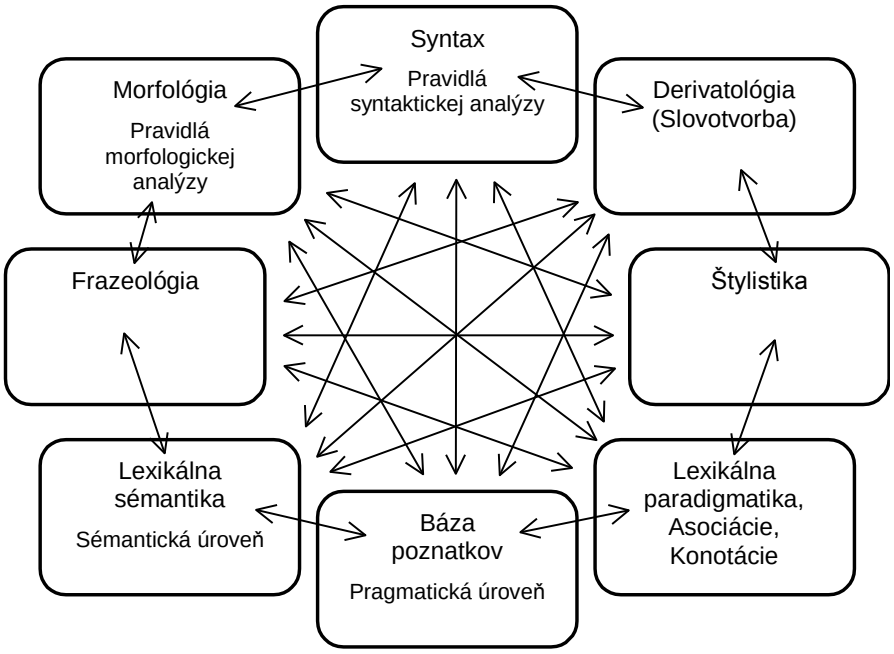


Obr. 2.6 Sekvenčný stratifikačný model jazykovej analýzy (upravené podľa [Páleš 1994]).

Na druhej strane sa často pragmatická analýza buď zanedbáva, alebo sa spája so sémantickou analýzou. Táto sémanticko-pragmatická analýza, ktorá je pre veľkú mieru zložitosti najproblematickejšia v celom procese lingvistickej jazykovej analýzy a je najviac závislá od rôznych jazykovedných či iných teórií významu, tvorí plynulý prechod k metódam znalostného prístupu reprezentácie obsahu textu [Furdík 2003].

Pri sekvenčnom modeli vznikajú na každej z úrovní analýzy nejednoznačnosti, ktoré sa riešia v interakcii s modulmi iných úrovní. Pri sekvenčnom spracovaní však tieto interakcie nie sú efektívne a spôsobujú nárast výpočtovej zložitosti [Páleš 1994, Furdík 2003]. Riešením môže byť modifikáca sekvenčného modelu analýzy na *paralelný model*, kde čiastkové procesy môžu vzájomne voľne komunikovať, odovzdávať si informácie

a verifikovať výsledky analýz na jednotlivých stupňoch. To umožňuje okamžitú spätnú korekciu medzivýsledkov konzultáciou s inými úrovňami. Schéma takéhoto paralelného modelu jazykovej analýzy je na Obr. 2.7.



Obr. 2.7 Paralelný model jazykovej analýzy (upravené podľa [Páleš 1994]).

Okrem toho, že paralelný model je schopný riešiť nejednoznačnosti jednotlivých úrovní analýzy, je aj efektívnejší a rýchlejší ako sekvenčný model. Dôvodom je to, že nesprávne alternatívy sa konzultáciou s inou úrovňou odhalia oveľa skôr a systém sa nemusí zaťažovať ich ďalším zbytočným spracovávaním. Naviac sa paralelnou činnosťou dosahuje efekt súčinnosti, tzv. *synergický efekt* [Páleš 1994]. Znamená to, že účinnosť a výkonnosť systému prevyšuje sumu účinkov jeho jednotlivých častí samých osebe. Napríklad pre slovenčinu a ďalšie flektívne jazyky je typický zložitý a komplikovaný morfologický systém, preto morfologická analýza je obyčajne robustná a výpočtovo náročná. Avšak v kombinácii so syntaktickými pravidlami sa proces analýzy výrazne zefektívni a zrýchli, pretože sa zníži miera neurčitosti pri generovaní alternatívnych riešení.

Na záver tejto časti je potrebné zdôrazniť, že lingvistický prístup sa pri predspracovaní uplatňuje iba zriedka, a v celej svojej komplexnosti doteraz prakticky realizovaný azda ani nebol. Dôvodom je zložitosť dôsledného popisu jazykovedného systému, nejednotnosť a nejednoznačnosť rôznych lingvistických teórií, a napokon aj vysoká výpočtová náročnosť systémov aplikujúcich lingvistickú analýzu textu. Častejšie bývajú parciálne doplnenia tradičných postupov tokenizácie a lematizácie o určité prvky morfologickej, syntaktickej, a niekedy aj sémantickej analýzy.

Ojedinelým a v mnohých smeroch inšpiratívnym pokusom o vytvorenie skutočne komplexného systému na báze lingvistického prístupu bol parafrázovač SAPFO [Páleš 1994]. Tento systém, vytvorený v jazyku Prolog, však nebol testovaný na rozsiahlejšom materiáli skutočných textov

a nepresiahol úroveň výskumného prototypového riešenia. Naopak, skupinu komerčných lingvistických modulov – aplikácií na lematizáciu, jazykovú korektúru, slovník synoným, a ďalšie ponúka vo svojom portfóliu spoločnosť Forma, s.r.o. (http://www.forma.sk). Avšak tieto zaiste užitočné, avšak iba čiastkové riešenia nemožno dosť dobre považovať za ucelený systém lingvistickej analýzy textu. O takýto ucelený systém, založený na lingvistickom prístupe, sa snaží projekt Slovenského národného korpusu (http://korpus.juls.savba.sk, [Garabík a kol. 2004]). Doteraz však bola vyriešená iba morfologická analýza a anotácia zdrojov [Garabík 2007], analýza na úrovni povrchovej syntaxe, hĺbkovej syntaxe a sémantiky sa postupne realizuje. Zaujímavý alternatívny prístup, ktorý redukuje zložitosť lingvistických pravidiel štatistickým a heuristickým porovnávaním voči rozsiahlemu súboru textových údajov, bol aplikovaný v projekte NAZOU (http://nazou.fiit.stuba.sk, [Krajči a kol. 2009]). Výsledky tohto výskumného projektu a ďalších podobných aktivít môžu byť dobrým základom pre budúcu realizáciu prakticky použiteľného systému na predspracovanie textov pomocou lingvistického prístupu.

2.7 Modely reprezentácie textových dokumentov

Modelom reprezentácie textových dokumentov nazývame takú formalizáciu dokumentu, resp. jeho textu v prirodzenom jazyku, ktorá nejakým spôsobom vyjadruje obsah tohto textu a zároveň umožňuje ich efektívne spracovanie metódami dolovania v textoch, predovšetkým algoritmami klasifikácie a zhlukovania (viď kapitoly 3 a 4). Významnou vlastnosťou a predpokladom aplikovateľnosti modelov reprezentácie textov je možnosť matematicky porovnávať texty podľa ich vzájomnej podobnosti, príbuznosti ich obsahov [Furdík 2003].

Východiskom konštrukcie rôznych modelov reprezentácie textových dokumentov pre úlohy získavania znalostí z textov je *matica dokument-term* (Obr. 2.1 v kap. 2). Texty z dokumentov v analyzovanom korpuse sa predspracujú postupmi popísanými v predchádzajúcich častiach 2.1 – 2.3, čiže konverziou na čistý text, segmentáciou, tokenizáciou, lematizáciou, morfologickou a prípadne aj komplexnou jazykovou analýzou. Pre každý z dokumentov sa získa zoznam (vektor) lematizovaných termov, ktoré vyjadrujú obsah daného dokumentu. Problémy pri tejto reprezentácii môže spôsobovať potenciálne veľký počet termov, ktorý negatívne ovplyvňuje efektivitu zhlukovacích a klasifikačných algoritmov, a tiež nerovnomerné zastúpenie termov vzhľadom na ich príspevok k celkovému obsahu dokumentu resp. kolekcie. Prvý problém, teda zníženie počtu termov v reprezentácii dokumentu, sa rieši odstránením neplnovýznamových slov (pozri v časti 2.4) a štatisticko-algebraickými metódami popísanými ďalej v časti 2.8. Druhý problém, čiže vhodné proporcionálne ohodnotenie termov podľa ich príspevku k obsahu dokumentu a kolekcii, odstraňuje práve použitie niektorého z modelov reprezentácie textových dokumentov. Pritom sa využívajú techniky váhovania a normovania termov, ktorých základné princípy boli popísané v časti 2.5.

V nasledujúcich podkapitolách 2.7.1 – 2.7.4 predstavíme štyri základné modely reprezentácie textových dokumentov, ktoré sa najčastejšie využívajú

pri úlohách získavania znalostí pomocou algoritmov klasifikácie a zhlukovania.

Podobná forma reprezentácie sa používa aj pri úlohe vyhľadávania informácií, kde je cieľom vhodným spôsobom uchovať informácie o tých jednotkách textu (termoch), ktoré majú byť čo najrýchlejšie vyhľadateľné. Z praktických dôvodov nie je reprezentácia pomocou matice dokument-term vhodná pre vyhľadávanie informácií, pretože je neefektívna z hľadiska uloženia v pamäti a prístupu k termom. Matica dokument-term je vo všeobecnosti riedka matica, obsahujúca veľa núl na pozíciách termov, ktoré sa nenachádzajú v textoch niektorých dokumentov z korpusu. Preto sa pre úlohu vyhľadávania informácií používa upravená matica dokument-term, ktorá pre každý z termov uchováva iba zoznam tých dokumentov, v ktorých sa daný term vyskytuje. Takáto reprezentácia textu sa nazýva **invertovaný index** (alebo skrátene **index**), proces transformácie dokumentu (textu) do tejto reprezentácie sa označuje ako **indexácia**.

Index obsahuje informácie o indexovaných termoch (vyhľadateľných jednotkách textu), o ich príslušnosti k jednotlivým dokumentom a prípadne aj o pozíciách ich výskytov v korpuse indexovaných dokumentov. Ide o vhodnú údajovú štruktúru, ktorá umožňuje rýchle získanie odpovede na informačný dopyt zadaný do systému pre vyhľadávanie informácií.

Z indexu je možné následne priamočiaro vygenerovať model pre reprezentáciu dokumentov, ktorý je vhodný na spracovanie pri úlohách dolovania v textoch. Nie je to však podmienkou, modely pre reprezentáciu textových dokumentov pre účely dolovania v textoch je možné vytvoriť aj priamo ich spracovaním, nie je nutné vytvárať najprv ich index.

Základné modely reprezentácie textových dokumentov pre dolovanie v textoch, ktoré sú bližšie popísané v nasledujúcich podkapitolách, používajú takéto označenia:

- K je množina príznakových popisov všetkých dokumentov z daného korpusu (indexových termov).
- M je počet všetkých termov v systéme K.
- k_t je t–ty term z množiny všetkých termov použitých v systéme K. Teda platí: $K = \{k_1, k_2, \ldots, k_M\}$, $t = 1, 2 .., M$.
- Nech d_i je i–ty dokument z korpusu dokumentov a číslo $w_{it} \geq 0$ je váha priradená dokumentu d_i a termu k_t, pričom $w_{it} = 0$ práve vtedy, keď sa term k_t v dokumente d_i nevyskytuje. Táto váha vyjadruje dôležitosť termu v dokumente.
- Každému dokumentu je priradený vektor váh $d_i = \{w_{i1}, w_{i2},..., w_{iM}\}$, nazývaný aj *lexikálny profil* dokumentu.
- $g_t(d_i) = w_{it}$ je funkcia vracajúca príslušnú váhu termu k_t pre dokument d_i.

2.7.1 Boolovský model

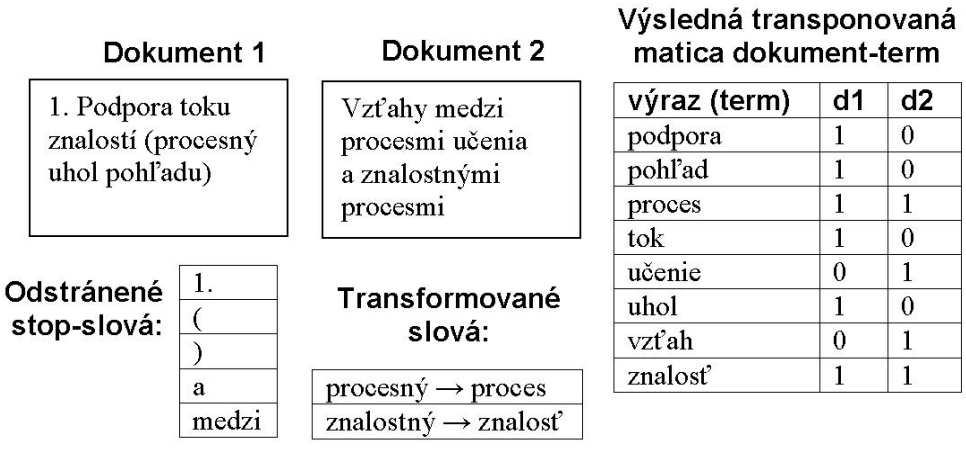
Tento model využíva na reprezentáciu textových dokumentov boolovskú logiku príslušnosti termov k dokumentom. Uvažuje sa iba prítomnosť resp.

neprítomnosť termu v dokumente. Váha t–teho termu v i–tom dokumente w_{it} nadobúda len boolovské hodnoty 0 (ak sa term v danom dokumente nevyskytuje) alebo 1 (ak sa term v dokumente vyskytuje aspoň raz), t.j. formálne: $\forall\, i, t: w_{it} \in \{0, 1\}$.

Dokument je v tomto modeli reprezentovaný ako množina indexovaných termov, hovoríme preto o *množinovo-teoretickom* prístupe. Jeho výhodou je jednoduchosť reprezentácie dokumentu, a tiež aj dopytu (v prípade jeho využitia na účely vyhľadávania informácií), ktorý je používateľom zadávaný v zloženom tvare a skladá sa z termov pospájaných logickými spojkami AND, OR, aleno NOT. Jednoduché je tiež vyhodnotenie podobnosti dokumentu k dopytu, ktorý buď spĺňa podmienku podobnosti, alebo nie.

Jednoduchosť boolovského modelu však prináša aj pomerne podstatné nevýhody. Takto reprezentované dokumenty nemožno zoradiť podľa stupňa podobnosti ku dopytu pri vyhľadávaní informácií. Problematickou je tiež reprezentácia rozsiahlych dokumentov, pri ktorých vo všeobecnosti narastá počet termov (príznakov). Tým sa v konečnom dôsledku veľké dokumenty stávajú podobnými ku väčšine dopytov, napriek ich odlišnému významu.

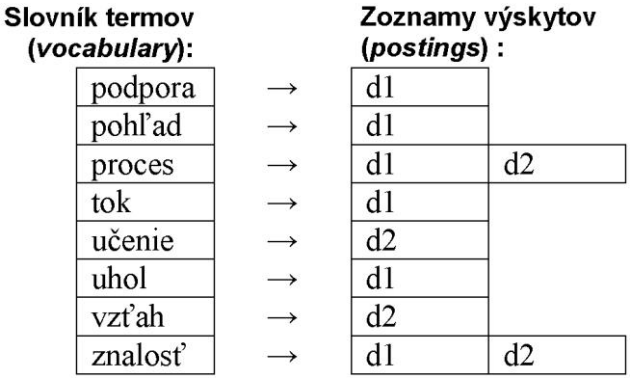
Príklad reprezentácie dvoch krátkych dokumentov v boolovskom modeli je zobrazený na Obr. 2.8. Priestor termov bol zmenšený odstránením piatich neplnovýznamových slov a morfologickou transformáciou dvoch odvodených prídavných mien (desubstantívnych adjektív) na tvary príslušných podstatných mien: *procesný → proces*, *znalostný → znalosť*. Vo výslednej transponovanej matici dokument-term stĺpce predstavujú dokumenty d_1 a d_2, na riadkoch sú jednotlivé termy. Matica obsahuje hodnoty 0 pri termoch, ktoré sa v dokumente nevyskytujú, a hodnoty 1 v riadkoch zodpovedajúcich tým termom, ktoré sa v danom dokumente vyskytujú aspoň raz.



Obr. 2.8 Reprezentácia dokumentov v boolovskom modeli

V takejto reprezentácii zodpovedá každému termu vektor binárnych hodnôt s dĺžkou rovnajúcou sa počtu dokumentov v analyzovanom korpuse. So vzrastajúcim počtom dokumentov tým výrazne rastú nároky na pamäťový priestor. Najmä pre úlohu vyhľadávania informácií je tento spôsob reprezentácie nevhodný a nahrádza sa invertovaným indexom. Pre každý term sa uchová iba zoznam tých dokumentov, v ktorých sa tento term

vyskytuje. Invertovaný index pre vyššie uvedený príklad je na Obr. 2.9.



Obr. 2.9 Reprezentácia dokumentov v boolovskom modeli

Boolovský model priamo podporuje vyhľadávanie dokumentov na základe dopytov konštruovaných pomocou hľadaných termov a logických spojok AND, OR, NOT. Pri reprezentácii binárnou maticou dokument-term vyhľadávanie prebieha ako bitový súčin reprezentácií termov z dopytu. Napríklad pre dopyt:

```
podpora AND znalosť AND NOT vzťah
```

je výsledok daný binárnym súčinom reprezentácií termov takto:

```
10 AND 11 AND NOT 01 = 10 AND 11 AND 10 = 10
```

Danej podmienke vyhovuje dokument d1, ktorý sa v tomto prípade vráti ako výsledok vyhľadávania.

Pri použití invertovaného indexu sa v procese vyhľadávania najprv získajú zoznamy výskytov pre termy z dopytu, ktoré sa potom zlúčia podľa zadaných logických spojok. Ak dopyt obsahuje iba spojky AND a NOT, postup je priamočiary – pre dopyt z predchádzajúceho príkladu je výsledkom:

```
d1 AND d1 AND NOT d2 = d1
```

Pre optimalizáciu zložitejších konjunktívnych dotazov (t.j. väčšie množstvo termov pospájaných spojkou AND) v rozsiahlych korpusoch dokumentov sa používa algoritmus založený na skúmaní frekvencie výskytu termov v celom priestore korpusu. Uprednostňujú sa zoznamy výskytov tých termov, ktoré majú najnižší počet výskytov [Manning, Raghavan, Schütze 2008]. Disjunktívne dotazy (t.j. termy pospájané spojkou OR) sa vyhodnocujú osobitne, ako samostatné termy, pričom sa zlúčia invertované zoznamy elementárnych termov tvoriacich dotaz.

Pre úlohy klasifikácie a zhlukovania je dôležité vedieť vyhodnotiť vzájomnú podobnosť dvoch dokumentov. Boolovská reprezentácia dokumentu, čiže binárny vektor príslušnosti termov, zodpovedá konjunktívnemu dotazu pri vyhľadávaní. Dokument teda možno reprezentovať ako konštrukciu jemu

príslušných termov pospájaných logickými spojkami AND. Reprezentácia dokumentu d_1 z príkladu na Obr. 2.8 bude teda mať tvar:

d_1 = podpora AND pohľad AND proces AND tok AND uhol AND znalosť

Podobnosť dokumentu d_1 s d_2, sa vyhodnotí binárnym súčinom reprezentácií termov alebo zlúčením:

sim(d_1,d_2) = *podpora* AND *znalosť*

alebo v binárnom vyjadrení:

sim(d_1,d_2) = 00100001

Kvantitatívne vyjadrenie podobnosti je však problematické. Istým indikátorom by mohol byť pomer počtu spoločných termov ku celkovému počtu termov v zlúčenej reprezentácii oboch dokumentov, čo je v tomto prípade $2 / 8 = 0{,}25$ (t.j. 25%). Avšak toto číselné vyjadrenie nezohľadňuje, ktoré konkrétne termy sa podieľali na zistenej miere podobnosti (napr. rovnakú podobnosť s d_1 by mal dokument obsahujúci termy *podpora* a *pohľad*), neberie sa do úvahy počet výskytov termov v textoch dokumentov, poradie termov, ich kontext a prípadná vzájomná súvislosť (lineárna nezávislosť termov je podmienkou aplikácie boolovského modelu).

Napriek výhodným vlastnostiam, ktorými sú jednoduchosť a jasný formalizmus, nie je boolovský model príliš vhodný pre aplikácie úloh vyhľadávania alebo získavania znalostí z textov. Medzi hlavné nevýhody tohto modelu patrí nutnosť presnej zhody výskytu termov v dopyte a v hľadanom dokumente. Pri vyhľadávaní informácií je dôsledkom získanie príliš veľkého (pri otázkach typu OR) alebo príliš malého (pri AND) počtu dokumentov v odpovedi, čo je nevhodné najmä z používateľského hľadiska. Nevýhodná je aj skutočnosť, že dokumenty nemožno korektne usporiadať podľa stupňa relevancie k dopytu, resp. podľa vzájomnej podobnosti ich obsahov, a to najmä preto, že sa neberie sa do úvahy frekvencia výskytu jednotlivých termov v dokumentoch. Niektoré z týchto nevýhod odstraňujú modely popísané v nasledujúcich častiach, predovšetkým vektorový model (časť 2.7.3).

2.7.2 Pravdepodobnostný model

Pravdepodobnostný model reprezentácie textových dokumentov bol pôvodne navrhnutý pre úlohu vyhľadávania informácií ako rozšírenie boolovského modelu. Cieľom bolo jednak odstrániť nevýhodu získavania dokumentov iba na základe presného porovnávania termov, a tiež umožniť usporiadanie vyhľadaných dokumentov podľa ich relevancie k dopytu. Keďže v systémoch na vyhľadávanie informácií sa dopyt reprezentuje rovnakým spôsobom ako vyhľadávaný dokument, princíp pravdepodobnostného modelu je možné využiť aj pre klasifikačné a zhlukovacie úlohy získavania znalostí z textov.

Reprezentácia dokumentov pri pravdepodobnostnom modeli ostáva, rovnako ako pri boolovskom modeli, binárna, avšak výpočet vzájomnej podobnosti

dvoch textov sa vykonáva v iteračnom procese na základe odhadu pravdepodobnosti ich obsahovej blízkosti, pričom sa berie do úvahy poradie termov v reprezentácii dokumentov. Podobnosť dvoch textov sa teda podľa tohto modelu vyjadrí nie binárnou hodnotou (rovnaký / odlišný), ale percentuálne (t.j. napr. dokumenty d_a a d_b sú si vzájomne obsahovo podobné na 84%).

Stratégia pravdepodobnostného modelu pri vyhľadávaní informácií spočíva vo výpočte pravdepodobnosti, že dokument patrí do množiny dokumentov z ideálnej množiny dokumentov v odpovedi na dopyt. Dokument aj dopyt sú reprezentované binárnymi vektormi d_j a q[11]. Pravdepodobnostný model [Fuhr 1992] používa podobnosť definovanú vzťahom:

$$sim(d_j, q) = \frac{p(R / q, \bar{d}_j)}{p(\bar{R} / q, \bar{d}_j)} \tag{2.5}$$

kde R je množina relevantných dokumentov, a $\bar{R}$ je množina dokumentov nerelevantných k dopytu. Podobnosť dokumentu k dopytu (resp. vzájomná podobnosť dvoch dokumentov) je teda daná pomerom pravdepodobnosti toho, že dokument d_j je relevantný k otázke, ku pravdepodobnosti toho, že dokument d_j k otázke relevantný nie je.

Za predpokladu lineárnej nezávislosti indexových termov v boolovských reprezentáciách dopytu q aj dokumentu d_j potom možno s použitím Bayesovho pravidla odvodiť:

$$sim(d_j, q) = \frac{p(R / \bar{d}_j)}{p(\bar{R} / \bar{d}_j)} = \frac{p(\bar{d}_j / R) \cdot p(R)}{p(/ \bar{d}_j / \bar{R}) \cdot p(\bar{R})} \approx \frac{p(\bar{d}_j / R)}{p(\bar{d}_j / \bar{R})}, \tag{2.6}$$

z čoho ďalej plynie:

$$sim(d_j, q) = \frac{\left(\prod_{g_i(\bar{d}_j)=1} p(k_i / R)\right) \cdot \left(\prod_{g_i(\bar{d}_j)=0} p(\bar{k}_i / R)\right)}{\left(\prod_{g_i(\bar{d}_j)=1} p(k_i / \bar{R})\right) \cdot \left(\prod_{g_i(\bar{d}_j)=0} p(\bar{k}_i / \bar{R})\right)} \tag{2.7}$$

kde

- $p(k_i / R)$ je pravdepodobnosť výskytu termu k_i v dokumente náhodne vybratom z R,
- $p(\bar{k}_i / R)$je pravdepodobnosť, že sa term k_i nevyskytuje v dokumente náhodne vybratom z R.

Ďalším odvodením sa získa podobnosť:

$$sim(d_j, q) \approx \sum_{i=1}^{t} w_{i,q} \cdot w_{i,j} \cdot \left(\log \frac{p(k_i / R)}{1 - p(k_i / R)} + \log \frac{1 - p(k_i / \bar{R})}{p(k_i / \bar{R})} \right) \tag{2.8}$$

[11] Vzhľadom na ekvivalenciu reprezentácie dopytu a dokumentu pri vyhľadávaní informácií možno v nasledujúcich vzťahoch dopyt *q* zameniť za dokument d_i. Následne sa dajú vzťahy pravdepodobnostného modelu priamo použiť pre úlohy klasifikácie a zhlukovania v textoch.

kde $w_{i,j} \in \{0,1\}$ a $w_{i,q} \in \{0,1\}$ sú váhy, t.j. indikátory existencie alebo absencie jednotlivých termov v boolovskej reprezentácii dokumentu a dopytu.

Keďže však množina R dokumentov podobných k dopytu na začiatku procesu nie je známa, je nutná inicializácia uvedených pravdepodobností. Jedným so spôsobov je na začiatku deklarovať:

- $p(k_i / R) = 0{,}5$, čo vyjadruje predpoklad, že výskyt všetkých termov k_i v dokumentoch z R je rovnako pravdepodobný, a tiež
- $p(k_i / \overline{R}) = \frac{n_i}{N}$, t.j. že distribúcia termu k_i mimo dokumentov z R je zhodná s jeho distribúciou v celej množine dokumentov. N je celkový počet dokumentov v analyzovanom korpuse, n_i je počet dokumentov ktoré vo svojej reprezentácii obsahujú term k_i.

Takto vytvorený model vráti počiatočnú množinu relevantných dokumentov. Táto sa potom iteratívne ďalej spresňuje. Nech V je množina dokumentov vrátených v prvej iterácii ako odpoveď na q a V_i je podmnožina takých vrátených dokumentov z V, ktoré obsahujú term k_i. Potom sú možné tieto alternatívne vyjadrenia pravdepodobností:

- $$p(k_i / R) = \frac{V_i}{V} \approx \frac{V_i + 0{,}5}{V+1} \approx \frac{V_i + \frac{n_i}{N}}{V+1} \quad (2.9)$$
- $$p(k_i / \overline{R}) = \frac{n_i - V_i}{N - V} \approx \frac{n_i - V_i + 0{,}5}{N - V + 1} \approx \frac{n_i - V_i + \frac{n_i}{N}}{N - V + 1} \quad (2.10)$$

Tento proces môže pokračovať v iteráciách automaticky, aj bez asistencie človeka, alebo s jeho asistenciou tak, že človek vyberie z odpovede systému množinu *V.*

Výstupom pravdepodobnostného modelu sú dokumenty usporiadané podľa pravdepodobnosti, miery relevancie k zadanému dopytu. Týmto sa výrazne zvyšuje praktická použiteľnosť boolovskej reprezentácie najmä pre vyhľadávanie informácií. Usporiadanie podľa miery relevancie je tiež dôležité pre úlohy klasifikácie a zhlukovania, kde sa dá aplikovať na výpočet vzájomnej podobnosti dvoch textových dokumentov reprezentovaných binárnym vektorom príslušnosti indexových termov. Pravdepodobnostný model má však aj nevýhody, medzi ktoré patrí najmä nutnosť počiatočného odhadu niektorých pravdepodobností, skutočnosť, že sa neberie do úvahy frekvencia výskytu jednotlivých termov v dokumentoch, a tiež predpoklad nezávislosti indexových termov.

2.7.3 Vektorový model

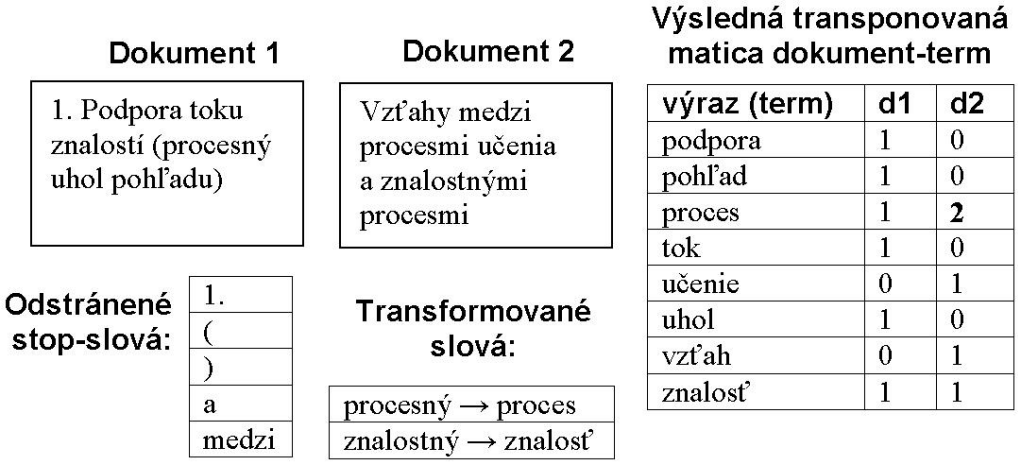
Vektorový model (V*ector Space Model*, VSM), spomínaný už vyššie (pozri napr. v častiach 2, 2.2, 2.4), je reprezentácia dokumentov pomocou štatistickej informácie popisujúcej každý dokument z analyzovaného korpusu ako *vektor termov* [Luhn 1953; Salton, Wong, Yang 1975]. Ide o najčastejšie používaný model reprezentácie textových dokumentov pri úlohách

vyhľadávania informácií aj pri získavaní znalostí z textov. Oproti boolovskému modelu, ktorý používa binárne ohodnotenie prítomnosti resp. absencie termu v texte dokumentu, vektorový model zohľadňuje relatívnu dôležitosť termu v danom dokumente a vyjadruje ju príslušnou váhou, najčastejšie reprezentovanou reálnym číslom.

Formálne možno zapísať, že dokument d_i je vo vektorovom modeli reprezentovaný vektorom $\bar{d}_i = (w_{i1}, w_{i2}, ..., w_{iM})$, kde w_{it} je váha, resp. dôležitosť termu k_t v dokumente d_i [Salton, Buckley 1988]. Súbor, resp. korpus N dokumentov sa pomocou vektorového modelu dá reprezentovať už vyššie uvedenou maticou dokument-term $\mathbf{F}(N \times M)$, ktorá má tvar:

$$\mathbf{F}(N \times M) = \begin{pmatrix} \bar{d}_1 \\ \bar{d}_2 \\ \dots \\ \bar{d}_N \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1M} \\ w_{21} & w_{22} & \dots & w_{2M} \\ \dots & \dots & \dots & \dots \\ w_{N1} & w_{N2} & \dots & w_{NM} \end{pmatrix} \qquad (2\text{–}11)$$

Najjednoduchším a aj nezriedka používaným spôsobom vyjadrenia váh termov je **frekvencia**, čiže *počet výskytov* daného termu v dokumente. Tento spôsob reprezentácie je založený na predpoklade, že čím častejšie sa term v danom texte nachádza, tým viac tento term prispieva k celkovému obsahu textu. Inak povedané, často sa vyskytujúce slová sú pre obsah textu dôležitejšie ako tie slová, ktoré sa v danom texte vyskytujú iba zriedka.



Obr. 2.10 Reprezentácia dokumentov vo vektorovom modeli

Na Obr. 2.10 je prezentovaný rovnaký príklad, aký bol použitý pri boolovskom modeli, ibaže v matici dokument-term sa tentokrát použili vyjadrenia váh termov podľa počtu ich výskytov v textoch. Term *proces* sa v dokumente d_2 vyskytuje dvakrát, preto váha tohto termu v dokumente d_2 má hodnotu 2.

Reprezentácia dokumentov pomocou VSM nezohľadňuje poradie slov v texte a ani ich kontext (t.j. okolie, spoluvyskytujúce sa slová). Termy sú reprezentované a spracovávané ako izolované a samostatné elementy, ktorých jedinou relevantnou charakteristikou je ich váha, ktorá zodpovedná frekvencii, počtu výskytov daného termu v texte dokumentu. Takýto spôsob reprezentácie sa označuje ako *bag-of-words model* [Harris 1954].

Samotný počet výskytov termu v texte však nie je najvhodnejším indikátorom váhy, t.j. relevantnosti, dôležitosti termu vzhľadom na obsah dokumentu. Je možné predpokladať, že pre dokument s 10-timi výskytmi nejakého termu bude tento term viac relevantný ako pre dokument s jedným výskytom tohto termu, avšak určite nebude 10-krát relevantnejší. Faktom je, že relevancia termu nerastie proporcionálne (lineárne) s frekvenciou jeho výskytu texte. Preto sa v aplikačnej praxi váha w_{it} termu k_t v indexe vektorového modelu nevyjadruje priamo frekvenciou tf_{it}, ale logaritmom tejto frekvencie:

$$w_{i,t} = \begin{cases} 1 + \log_{10} tf_{i,t}, & ak\ tf_{i,t} > 0 \\ 0, & \text{ináč} \end{cases} \qquad (2\text{–}12)$$

Okrem frekvencie výskytu termu v danom dokumente je dôležitá aj jeho frekvencia v celom skúmanom korpuse dokumentov. Zohľadnenie tejto globálnej frekvencie výskytu termu v celom korpuse je založené na pozorovaní, že zriedkavé termy nesú viac obsahovej informácie, než termy často sa vyskytujúce v mnohých dokumentoch. Pre matematické vyjadrenie tohto poznatku sa zavedie tzv. **dokumentová frekvencia** df_t termu k_t, ktorá je určená počtom takých dokumentov v korpuse, v ktorých sa nachádza term k_t. Dokumentová frekvencia df_t je nepriamo úmerná informatívnosti daného termu, čiže jeho príspevku na odlíšenie obsahu niektorého z dokumentov od ostatných. Platí tiež, že dokumentová frekvencia nie je nikdy väčšia než počet dokumentov v korpuse, čiže $df_t \leq N$.

Na základe df_t a celkového počtu dokumentov v korpuse N možno, podobne ako pri úprave frekvencie termu logaritmickou mierkou, definovať tzv. **inverznú dokumentovú frekvenciu** idf_t:

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right) \qquad (2\text{–}13)$$

Hodnota idf sa takto dá vypočítať pre každý z indexových termov korpusu. Pre príklad z Obr. 2.10 budú váhy w_{it} počítané logaritmom frekvencie termov, dokumentové a inverzné dokumentové frekvencie pre jednotlivé termy nadobúdať hodnoty:

Tab. 2.2 Dokukmentové a inverzné dokumentové frekvencie pre príklad z Obr. 2.10

výraz (term)	$w_{it} = 1 + log_{10} tf_{it}$		df_t	idf_t
	d1	**d2**		
podpora	1	0	1	0,3
pohľad	1	0	1	0,3
proces	1	0,3	2	0
tok	1	0	1	0,3
učenie	0	1	1	0,3
uhol	1	0	1	0,3
vzťah	0	1	1	0,3
znalosť	1	1	2	0

Kombinácia frekvencie termu v jednom dokumente a rozloženia výskytov tohto termu naprieč celou kolekciou dokumentov sa označuje ako ***tf-idf* schéma**. Je to najlepšia známa váhovacia schéma pre vyhľadávanie informácií [Salton, Buckley 1988] aj pre úlohy získavania znalostí z textov pomocou klasifikácie a zhlukovania. Váha w_{it} termu k_t je pomocou *tf-idf* schémy vyjadrená vzťahom $w_{it} = tf_{it} \times idf_t$, čiže:

$$w_{i,t} = \left(1 + \log tf_{i,t}\right) \times \log_{10}\left(\frac{N}{df_t}\right) \qquad (2\text{–}14)$$

Okrem tejto základnej *tf-idf* schémy sa používa viacero ďalších odvodených typov schémy s odlišným spôsobom výpočtu frekvencie termov, inverznej dokumentovej frekvencie, aj normalizácie – zjednotenia dĺžky vektora termov. Prehľad rôznych variantov *tf-idf* schémy v kombinácii s normalizáciou na dĺžku vektora (porov. v časti 2.5) je prezentovaný v Tab. 2.3. V ľavom stĺpci tabuľky sú uvedené kódové označenia pre najčastejšie používané možnosti faktorov zahrnutých do *tf-idf* váhovania – frekvencia termu: {b, n, m, a}, dokumentová frekvencia: {n, t, p}, normalizovanie: {n, c}. Medzi najčastejšie používané schémy patrí binárne váhovanie $w_{it} \in \{0, 1\}$ (kóodové označenie *bnn*), frekvencia termov tf_{it} *(nnn)*, základná *tf-idf* schéma (*ntn/ntc*) a normalizovaná schéma *atn/atc*.

Tab. 2.3 Faktory *tf-idf* schémy

frekvencia termu		
b	1,0	binárne váhovanie
n	tf_{it}	frekvencia termu
m	$\frac{tf_{it}}{\max tf_{it}}$	normalizovaná frekvencia termu
a	$0{,}5 + 0{,}5\frac{tf_{it}}{\max tf_{it}}$	normalizovaná frekvencia termu (normalizácia maximálnou hodnotou, interval 0,5 - 1,0)
dokumentová frekvencia		
n	1,0	bez inverznej dokumentovej frekvencie
t	$\log\frac{N}{df_t}$	inverzná dokumentová frekvencia
p	$\log\frac{N - df_t}{df_t}$	pravdepodobnostná inverzná dokumentová frekvencia
normalizovanie		
n	1,0	bez normalizácie
c	$\frac{1}{\sqrt{\sum_{t=1}^{M} w_{it}^{\,2}}}$	normalizácia na 1 dĺžku vektora

Reprezentácia pomocou VSM dovoľuje rôznym spôsobom porovnávať a vyhodnocovať obsahovú blízkosť, podobnosť dvoch dokumentov. Majme napríklad dva dokumenty d a q (pričom pri vyhľadávaní informácií môže byť q dopyt, je však reprezentovaný rovnakým spôsobom ako dokument d). Podobnosť d a q je potom daná súčtom váh tých termov, ktoré sa vyskytujú v reprezentáciách oboch dokumentov. Pri použití *tf-idf* schémy má vzťah pre podobnosť d a q tvar:

$$sim(q,d) = \sum_{t \in q \cap d} tf_t \times idf_{t,d} \qquad (2–15)$$

Ďalšie metriky, založené na porovnávaní vzdialenosti vektorov, sú napríklad:

- Euklidovská vzdialenosť: $sim(d_i, d_j) = \sum_{i=1}^{n} \sqrt{(a_i - b_i)^2}$
- L-metrika (Manhattan): $sim(d_i, d_j) = \sum_{i=1}^{n} |a_i - b_i|$
- Snp metrika: $sim(d_i, d_j) = \max_i \{|a_i - b_i|\}$
- Sokalova metrika: $sim(d_i, d_j) = \frac{1}{n} \sum_{i=1}^{n} \sqrt{(a_i - b_i)^2}$

Všetky tieto metriky predpokladajú rovnakú dĺžku vektorov, medzi ktorými sa určuje podobnosť. V uvedených vzťahoch sú d_i a d_j dva dokumenty reprezentované vektormi $d_i = (a_1, ..., a_n)$ a $d_j = (b_1, ..., b_n)$, kde a_x a b_x sú frekvencie výskytov termov (slov) popisujúcich tieto dokumenty. Konštanta n je počet termov, čiže dĺžka vektorov d_i a d_j.

Dokumenty, a pri vyhľadávaní informácií aj dopyty, možno vo vektorovom modeli reprezentovať vektormi v M-rozmernom priestore (kde M je počet všetkých termov v korpuse). Vzájomná obsahová podobnosť dokumentov, resp. pri vyhľadávaní podobnosť dokumentov a dopytu, sa potom dá vyjadriť ako „blízkosť“ príslušných vektorov reprezentujúcich dokumenty alebo dopyt. Vzhľadom na potenciálne rôznu veľkosť dokumentov, a tým daný veľmi rôznorodý počet indexových termov v ich reprezentáciách, nie je výhodné blízkosť vektorov vyjadrovať napríklad pomocou Euklidovskej vzdialenosti, pretože táto má veľkú hodnotu pre vektory rôznej dĺžky. Z rovnakého dôvodu nie je pre korpus pozostávajúci z príliš rôznorodých dokumentov vhodné používať súčin alebo súčet zložiek vektorov, napríklad vyššie uvedený vzťah pre výpočet podobnosti podľa *tf-idf* schémy. Najvýhodnejším spôsobom na vyjadrenie blízkosti vektorov vo VSM je vyčíslenie a porovnávanie *uhla medzi vektormi* dvoch dokumentov, resp. medzi vektorom dokumentu a vektorom dopytu.

Najčastejšie používanou metrikou na porovnávanie uhla dvoch vektorov vo vektorovom modeli je tzv. **kosínusová miera podobnosti** [Manning, Raghavan, Schütze 2008]. Pri tejto metrike sa využíva skutočnosť, že kosínus je monotónne klesajúca funkcia na intervale $<0^o, 180^o>$. Vo všeobecnosti platí, že vektor môže byť normalizovaný predelením všetkých jeho zložiek jeho dĺžkou:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2} \qquad (2–16)$$

Výsledkom normalizácie je, že vektor má jednotkovú dĺžku a spadá teda do intervalu $<0^o, 180^o>$. Pre dva vektory reprezentujúce dokumenty alebo dopyty je ich kosínusová vzdialenosť rovná skalárnemu súčinu zložiek týchto vektorov, pričom súčin je normalizovaný na dĺžku oboch vektorov:

$$\cos\left(\vec{q},\vec{d}\right)=\frac{\vec{q}\bullet\vec{d}}{\left|\vec{q}\right|\left|\vec{d}\right|}=\frac{\vec{q}}{\left|\vec{q}\right|}\bullet\frac{\vec{d}}{\vec{d}}=\frac{\sum_{i=1}^{|T|}q_i d_i}{\sqrt{\sum_{i=1}^{|T|}q_i^2}\sqrt{\sum_{i=1}^{|T|}d_i^2}} \qquad (2–17)$$

Za najviac podobné (identické) sa považujú dokumenty reprezentované vektormi s tým istým uhlom, teda s koreláciou $\cos(q, d) = 1$. Variantom tejto metódy je *pravdepodobnostný model kosínusovej korelácie* [Goldszmidt, Sahami 1998], v ktorom sa podobnosť, resp. prekrytie dvoch dokumentov vypočítava odhadom pravdepodobnosti výskytu každého termu v každom z dokumentov a vynásobením týchto odhadov.

Vektorový model reprezentácie textových dokumentov je najčastejšie využívaným modelom pre úlohy vyhľadávania informácií aj získavania znalostí z textov. Medzi jeho výhody patrí schéma váženia termov podľa frekvencie výskytu, ktorá zvyšuje výkonnosť vyhľadávania a vzájomného porovnávania obsahovej podobnosti dokumentov. Vektorový model umožňuje vyhľadať aj také dokumenty, ktoré iba čiastočne vyhovujú dopytu, a následne usporiadať vyhľadané dokumenty podľa stupňa relevancie, obsahovej blízkosti. Nevýhodou VSM je predpoklad nezávislosti indexových termov, v praxi sa však v textových dokumentoch vyskytujú iba lokálne závislosti malých skupín termov.

2.7.4 Model distribuovanej sémantiky

Model distribuovanej sémantiky (*Distributional Semantic model, DSM*) je zovšeobecnením vektorového modelu. Podobne ako vektorový model, aj DSM nahrádza termy vo vektorovej reprezentácii obsahu (kľúčovými) slovami textu. Vychádza sa pritom z predpokladu, že existuje silná korelácia medzi slovom a jeho textovým okolím, *kontextom*. **Kontext** je v tomto prípade definovaný ako zoznam slov, s ktorými sa dané slovo spoločne objavilo v rámci určenej textovej jednotky. Textovou jednotkou môže byť veta, odsek, kapitola, resp. akákoľvek zadefinovaná časť textu.

Nech je textovou jednotkou veta a nech text obsahuje napríklad nasledujúce vety:

1. *X môžete predať prostredníctvom obchodníka z cennými papiermi.*
2. *Zákon ukladá vyhlásiť povinnú ponuku na kúpu X od všetkých akcionárov.*
3. *Valné zhromaždenie schválilo ukončenie obchodovania X na burze.*

Potom množinou kontextov slova X je zoznam všetkých (plnovýznamových) slov, s ktorými sa X spoločne objavilo v textovej jednotke, čiže vo vete:

```
{{predať, obchodník, cenný papier}, {zákon, vyhlásiť,
ponuka, kúpa, akcionár}, {valné zhromaždenie, obchodovanie,
burza}}
```

Na základe tejto množiny kontextov by X malo byť identifikované ako *akcie*.

DSM predpokladá, že dve slová sú sémanticky (obsahovo) zhodné vtedy, ak sa zhodujú kontexty, v ktorých sa tieto slová vyskytli [Besancon, Rajman, Chappelier 1998]. Na úrovni dokumentov to znamená, že dva dokumenty sú si natoľko obsahovo podobné, nakoľko sa zhodujú kontexty slov v ich vektorovej reprezentácii.

Nech je množina kontextov slova definovaná množinou *spolu-výskytov*, teda množinou slov, s ktorými sa dané slovo spoločne objavilo v určenej textovej jednotke. Frekvencia spolu-výskytu dvoch slov je definovaná ako počet spoločných výskytov týchto dvoch slov v jednotke textu. Pre každé slovo je definovaný jeho profil spolu-výskytov ako vektor frekvencií spolu-výskytov daného slova a každého iného slova, s ktorým sa dané slovo spoločne vyskytlo v jednotke textu.

Pre každý profil spolu-výskytov daného slova je určený vektor frekvencií spolu-výskytov medzi týmto slovom a každým elementom prioritne určenej množiny slov, ktorá tvorí skupinu určujúcich znakov tohto profilu spolu-výskytov daného slova. Ak je P veľkosť skupiny určujúcich znakov, potom profil spolu-výskytov slova k_t môže byť zapísaný ako vektor $\bar{c}_t = (c_{t1},...,c_{tP})$. Pre množinu M slov sa dá zostrojiť matica spolu-výskytov $\mathbf{C}(M \times P)$, v ktorej každý riadok bude reprezentovať profil spolu-výskytu jedného slova:

$$\mathbf{C}(M \times P) = \begin{pmatrix} \bar{c}_1 \\ \bar{c}_2 \\ \dots \\ \bar{c}_M \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1P} \\ c_{21} & c_{22} & \dots & c_{2P} \\ \dots & \dots & \dots & \dots \\ c_{M1} & c_{M2} & \dots & c_{MP} \end{pmatrix} \qquad (2–18)$$

Reprezentácia celého dokumentu d_i je potom definovaná ako vážený priemer profilov spolu–výskytov slov, ktoré dokument popisujú:

$$\bar{d}_i = \sum_{t=1}^{M} w_{it} \cdot \bar{c}_t \qquad (2–19)$$

Váha w_{it} priradená každému profilu spolu-výskytov $\bar{c}_t$ je zhodná s váhou, ktorá bola priradená slovu k_t pre dokument d_i v modeli VSM. Celá kolekcia dokumentov je potom reprezentovaná maticou $\mathbf{D}$:

$$\mathbf{D}(N \times P) = \mathbf{F}(N \times M) \times \mathbf{C}(M \times P) \qquad (2–20)$$

Dôležitou výhodou modelu DSM je tzv. **redukcia príznakového priestoru** (porov. nasledujúcu kapitolu 2.8), teda zníženie dimenzie vektorovej reprezentácie dokumentov. Rozmer P v matici $\mathbf{D}$ je totiž daný veľkosťou skupiny určujúcich znakov, ktorá je určite menšia ako počet M všetkých reprezentovaných slov v modeli VSM[12]. Napriek tomu model DSM berie

[12] Určujúce znaky sú vopred (napr. administrátorom) vybrané slová, ktoré jednak vhodne charakterizujú texty z korpusu, zároveň ich však aj vzájomne obsahovo odlišujú. Práve výber vhodných slov do množiny určujúcich znakov je problémom tejto metódy, pretože výrazne ovplyvňuje kvalitu indexácie a následného vyhľadávania. Jedným z možných prístupov je

do úvahy rovnaký počet slov ako VSM, pretože všetky slová sú reprezentované prostredníctvom váženého priemeru profilov svojich spolu-výskytov.

Priemerný dokument je reprezentovaný 2000 až 5000 slovami [Koštiaľ 2002]. Keďže výpočet podobnosti sa vykonáva cez každé z týchto slov, pre kolekcie obsahujúce tisíce dokumentov je mimoriadne dôležité znížiť výpočtovú náročnosť a tým zvýšiť rýchlosť porovnávania vektorových profilov dokumentov. Model DSM je jednou z možností riešenia tohto problému, pri vhodnom výbere množiny určujúcich znakov je dobrým kompromisom medzi stratou informácie a rýchlosťou.

Aby sa v indexe nestratila priama informácia o slovách vyskytujúcich sa v textoch dokumentov, dá sa použiť **hybridná reprezentácia** kombinujúca modely VSM a DSM. Ak $\mathbf{F}'$ bude taká vážená matica frekvencií výskytov slov podľa VSM, že bude obmedzená iba na tie slová, ktoré sú zároveň určujúcimi znakmi, potom hybridná reprezentácia dokumentov bude mať tvar:

$$\mathbf{D} = (1 - \alpha)\mathbf{F}' + \alpha\mathbf{FC} \qquad (2\text{–}21)$$

kde α je parameter hybridizácie, pre ktorý platí $0 \leq \alpha \leq 1$.

2.8 Redukcia príznakového priestoru

Pre vektorovú reprezentáciu dokumentov je charakteristický veľký počet dimenzií, čo spôsobuje neefektívnosť pri učení a obmedzuje použitie niektorých klasifikačných alebo zhlukovacích metód. Navyše, medzi termami zahrnutými do vektorovej reprezentácie je veľa irelevantných termov, ktoré predstavujú napríklad pri induktívnom učení klasifikátorov šum a spôsobujú náchylnosť k preučeniu. Pri dolovaní v textoch je teda vhodné z uvedených dôvodov použiť metódy redukcie príznakového priestoru.

Metódy redukcie je možné rozdeliť podľa vytvoreného redukovaného priestoru K_r na metódy *selekcie termov* pri ktorých je redukovaný priestor K_r podmnožinou pôvodného priestoru K a metódy *extrakcie termov*, pri ktorých sú príznaky redukovaného priestoru vytvorené kombináciou alebo transformáciou pôvodných termov (výsledné príznaky nemusia byť interpretovateľné ako termy).

Najjednoduchším spôsobom selekcie termov je vynechanie polovýznamových a funkčných slov (napríklad častíc, pomocných slovies, zámen, spojok atď., ktoré môžu byť zahrnuté medzi stop slová – viď v časti 2.4). K redukcii taktiež dochádza pri prevedení ich jednotlivých tvarov na koreň (angl. *stemming*), alebo na základný tvar (lematizácia).

Ďalšie techniky redukcie sú už viazané na určitý typ úloh dolovania v textoch. Napr. pri selekcii termov *pre kategorizáciu textov* sa najčastejšie používa heuristika, pri ktorej sa každý term ohodnotí podľa relevancie určenej na základe štatistík odhadnutých na trénovacích dátach [Yang, Pedersen 1997]. Výsledný príznakový priestor je potom tvorený r najlepšie

výber na základe frekvencie [Koštiaľ 2002], prípadne sa dajú použiť niektoré metódy jazykovej analýzy.

ohodnotenými termami, pričom počet dimenzií r je možné určiť napr. krížovou validáciou. Medzi funkcie, ktoré sa najčastejšie používajú na ohodnotenie termov patrí napr. frekvencia termov tf_{it} *(*na rozdiel od ďalších troch sa táto dá použiť kedykoľvek, nielen pri kategorizácii textov*)*, informačný zisk:

$$IG(w_t, c_j) = H(A+C, N) - \left(\frac{A+B}{N}\right) H(A, A+B) - \left(\frac{C+D}{N}\right) H(C, C+D) \quad (2.22)$$

χ^2 štatistika:

$$\chi^2(w_t, c_j) = \frac{N(AD-BC)^2}{(A+C)(B+D)(A+B)(C+D)} \quad (2.23)$$

a pomerová pravdepodobnostná funkcia:

$$OR(w_t, c_j) = \frac{(A+1/2)(D+1/2)}{(B+1/2)(C+1/2)}, \quad (2.24)$$

kde $H(n,m) = -(n/m \log(n/m))$ je entropia, $N = A+B+C+D$ je celkový počet dokumentov, A je počet dokumentov z triedy c_j v ktorých sa vyskytol term w_t, B je počet dokumentov v ktorých sa vyskytol term w_t a ktoré nie sú z triedy z triedy c_j, C je počet dokumentov z triedy c_j v ktorých sa nevyskytol term w_t, a D je počet dokumentov ktoré nie sú z triedy c_j a v ktorých sa nevyskytol term w_t.

Metódy extrakcie termov sú založené na mapovaní priestoru termov K do nového, menej rozmerného priestoru R. Medzi základné metódy extrakcie termov patrí zhlukovanie termov a *latentné sémantické indexovanie (Latent Semantic Indexing, LSI)*.

Pri extrakcii termov pomocou zhlukovania sa nahradia jednotlivé termy zhlukmi, ktoré vzniknú spojením viacerých termov s podobnou distribúciou kategórií [Baker, McCallum 1998]. Pri LSI [Deerwester, Dumais, Landauer, Furnas, Harshman 1990] (podrobnejšie viď. nasledujúca časť 2.8.1) je mapovanie z priestoru K do R založené na dekompozícii matice vektorov $\bar{d}_i \in \mathbf{D}$. Na rozdiel od selekcie a zhlukovania termov, vytvorené dimenzie už nie sú jednoducho interpretovateľné ako termy (dimenzie v redukovanom priestore sú lineárnou kombináciou pôvodných termov).

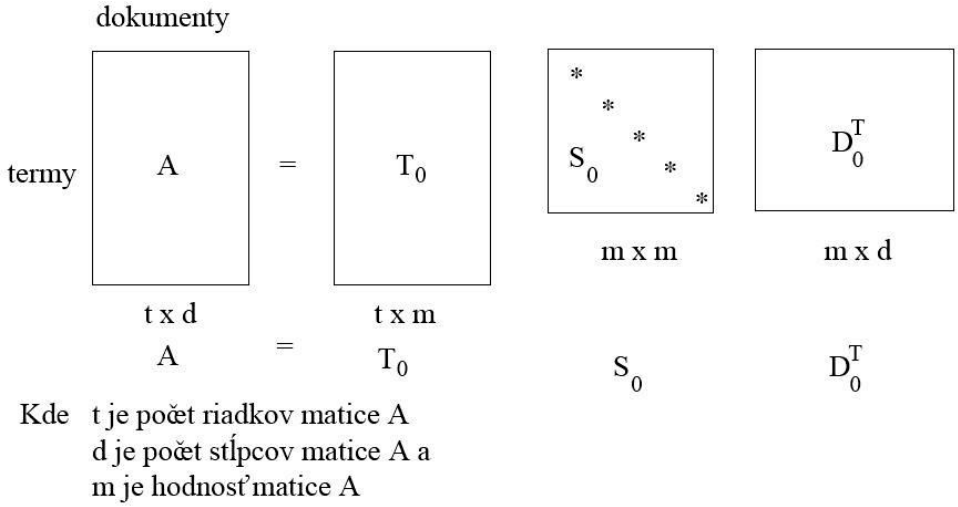
2.8.1 LSI model

Latent Semantic Indexing (LSI) model [Deerwester, Dumais, Landauer, Furnas, Harshman 1990] vychádza z maticovej reprezentácie termov (slov) a dokumentov, ktorú vytvára vektorový model. Na túto maticu sa aplikuje výpočet *Singular Value Decomposition* (SVD). Podobne ako model DSM, aj LSI redukuje rozmer vektorov reprezentujúcich dokumenty. Používa však na to práve algoritmus SVD, čiže nie je potrebné ručne navrhovať množinu špeciálnych určujúcich znakov na riadenie indexácie.

Nech $\mathbf{A}$ je matica term-dokument (transponovaná matica $\mathbf{F}$ uvedená vyššie), ktorá má M riadkov (každý riadok pre jeden term) a N stĺpcov (každý stĺpec pre jeden dokument). Potom dekompozícia $\mathbf{A}$ pomocou SVD bude mať tvar (viď. Obr. 2.1):

$$SVD(\mathbf{A}) = \mathbf{T}_0\mathbf{S}_0\mathbf{D}_0^{\mathrm{T}}, \qquad (2–25)$$

kde $\mathbf{T}_0(M \times K)$ je matica ortonormálnych stĺpcov, ktoré sa označujú ako ľavé singulárne vektory, $\mathbf{S}_0(M \times M)$ je diagonálna matica pozitívnych singulárnych hodnôt usporiadaná zostupne a $\mathbf{D}_0(M \times K)$ je matica ortonormálnych stĺpcov, ktoré sa zvyčajne nazývajú pravé singulárne vektory. Hodnota m je hodnosť matice **A**.



Obr. 2.11 Schéma dekompozície singulárnych hodnôt

Takýto rozklad pomocou diagonálnej matice singulárnych hodnôt $\mathbf{S}_0$ dovoľuje vykonať operácie, ktoré aproximujú pôvodnú maticu **A** a tým výrazne zredukujú priestor príznakov. *k* najmenších hodnôt singulárnych čísel v matici $\mathbf{S}_0$ sa eliminuje. Ich vynulovaním, spolu s príslušnými stĺpcami matice $\mathbf{T}_0$ a riadkami matice $\mathbf{D}_0^{\mathrm{T}}$ ostáva dimenzia priestoru rádovo 10-krát menšia [Koštiaľ 2002]. Po spätnom vynásobení je výsledkom aproximovaná matica $\mathbf{A}_{\mathrm{apr}}$, ktorá je optimálna v zmysle najmenších štvorcov:

$$\mathbf{A} \approx \mathbf{A}_{\mathrm{apr}} = \mathbf{T}_{\mathrm{k}}\mathbf{S}_{\mathrm{k}}\mathbf{D}_{\mathrm{k}}^{\mathrm{T}}$$

Parameter *k*, spravidla menší ako je hodnosť pôvodnej matice **A**, sa stanovuje tak, aby aproximovaná matica dostatočne pokrývala celú štruktúru termov, aby však pritom boli zanedbané všetky nepodstatné detaily v tejto štruktúre.

Ak sa priestor dokumentov zmení, napríklad ak doň pribudnú nové dokumenty alebo nové termy, potom je potrebné aktualizovať maticu $\mathbf{A}_{\mathrm{apr}}$ tak, aby zodpovedala novým údajom. Matica sa rozšíri o *p* nových dokumentov a *r* nových termov, pričom je zároveň nevyhnutné prepočítať a upraviť váhy tak, aby sa zachovala ortogonalita priestoru termov a dokumentov. Jednou z možností je kompletné prepočítanie SVD, čo je však časovo a priestorovo pomerne náročné. Metóda vloženia (*folding-in*) využíva existujúcu dekompozíciu matice $\mathbf{A}_{\mathrm{apr}}$. Každý nový dokument je reprezentovaný ako vážený priemer vektorov tých termov, ktoré obsahuje a každý nový term je reprezentovaný ako vážený priemer vektorov termov tých dokumentov, v ktorých sa nachádza. Vektory nových dokumentov sa vložia ako nové stĺpce do matice $\mathbf{D}_{\mathrm{k}}$ a vektory nových termov sa podobne pripoja k matici $\mathbf{T}_{\mathrm{k}}$. Táto metóda je rýchlejšia ako úplné prepočítanie SVD, jej nevýhodou je však

to, že sa nevykonávajú žiadne korekcie na udržanie ortogonality priestoru, čo sa pri väčšom počte nových dokumentov môže prejaviť v skreslených výsledkoch. Tieto a ďalšie známe metódy aktualizácie LSI modelu sú detailne popísané v [O'Brien 1994].

Redukcia príznakového priestoru metódou LSI má výhody aj nevýhody, ktoré možno zhrnúť nasledovne.

Výhody LSI:

- *Kvalita* - najmä vtedy, ak sú dokumenty v kolekcii dostatočne obsahovo rôznorodé, t.j. ak sú z pomerne širokej domény.
- *Rýchlosť* pri práci s týmto modelom - zrýchlenie výpočtov oproti VSM modelu je až 3,5-násobné.
- Nevýhody LSI:
- *Pamäťová náročnosť*. Veľké priestorové nároky patria medzi najväčšie nevýhody tejto metódy a prejavujú sa nielen pri dekompozícii, ale aj pri vyhľadávaní alebo klasifikácii.

Výpočtová náročnosť. Časovo náročná je iba dekompozícia matice term-dokument, ktorá je však pre nemennú kolekciu dokumentov jednorázovou procedúrou. Samotné použitie dekompozície je potom už z časového hľadiska oproti VSM veľmi výhodné.

3 Kategorizácia textov

3.1 Základné pojmy

Úlohou kategorizácie textov je nájsť aproximáciu neznámej funkcie $\Phi : D \times C \rightarrow \{true, false\}$, kde D je množina textových dokumentov a $C = \{c_1, ..., c_{|C|}\}$ je množina preddefinovaných *kategórií*.

Funkcia Φ nadobúda pre $\langle d_i, c_j \rangle$ hodnotu *true* ak dokument d_i patrí do kategórie c_j. Hľadaná funkcia $\hat{\Phi} : D \times C \rightarrow \{true, false\}$ ktorá aproximuje Φ sa označuje ako tzv. *klasifikátor*[13].

Definícia kategorizácie textov ďalej vychádza z nasledujúcich predpokladov:

- kategórie sú nominálne označenia a nie je dostupná žiadna (deklaratívna ani procedurálna) znalosť o ich význame
- kategorizácia musí byť založená iba na znalostiach extrahovaných z textu dokumentov[14].

Takáto definícia je teda všeobecná a nevyžaduje dostupnosť iných zdrojov, ktoré by bolo náročné vytvoriť. Tieto obmedzenia však nemusia byť dodržané pri operačných podmienkach, kedy je možné použiť akékoľvek znalosti, ktoré by mohli proces kategorizácie zefektívniť.

V závislosti na aplikácii je možné obmedziť počet kategórií pre ktoré nadobúda funkcia Φ hodnotu *true* pre daný dokument d_i. V prípade, kedy môže byť dokument $d_i \in D$ zaradený do práve jednej kategórie $c_j \in C$, ide o tzv. klasifikáciu do jednej triedy a množina C sa označuje ako množina *neprekrývajúcich sa* tried. Prípad, kedy môže byť každý dokument zaradený do ľubovoľného počtu $k = 0, ..., |C|$ kategórií z množiny C sa označuje ako viacnásobná klasifikácia a C sa označuje ako množina *prekrývajúcich sa* tried.

Zvláštnym prípadom je binárna klasifikácia, kedy môže byť dokument zaradený do jednej z dvoch tried. Riešenie binárneho prípadu je všeobecnejšie, pretože algoritmy pre binárnu klasifikáciu môžu byť použité aj pri viacnásobnej klasifikácii. Za predpokladu, že jednotlivé triedy sú navzájom nezávislé (tzn. pre každú triedu c_j, c_k $j \neq k$ je hodnota funkcie $\Phi(d_i, c_j)$ nezávislá od hodnoty $\Phi(d_i, c_k)$), problém viacnásobnej klasifikácie do viacerých tried je možné dekomponovať na problém $|C|$ nezávislých binárnych klasifikácií do tried $\{c_j, \bar{c}_j\}$ pre $j = 0, ..., |C|$. Klasifikátorom pre kategóriu c_j je potom označená funkcia $\hat{\Phi}_j : D \rightarrow \{true, false\}$, ktorá aproximuje neznámu funkciu $\Phi_j : D \rightarrow \{true, false\}$.

Automatická kategorizácia textov vyžaduje aby $\hat{\Phi}$ bola boolovskou funkciou, pretože je potrebné jednoznačne rozhodnúť či dokument patrí do danej kategórie. Pri semiautomatickej kategorizácii postačuje, aby systém zoradil

[13] resp. model alebo hypotéza

[14] tzn. že metadáta (napr. typ dokumentu, dátum publikovania atď.) nemusia byť dostupné

kategórie $C = \{c_1, ..., c_{|C|}\}$ podľa reálneho ohodnotenia (angl. *rank*), ktoré vyjadruje do akej miery má byť dokument zaradený do danej kategórie. Výsledná klasifikácia je ponechaná na experta, ktorému stačí prejsť pri rozhodovaní iba skrátený zoznam najlepšie ohodnotených kategórií. Neznáma funkcia Φ je potom definovaná ako $\Phi : D \times C \rightarrow R$, resp. je aproximovaná klasifikátorom $\hat{\Phi} : D \times C \rightarrow R$. Systémy pre semi–automatickú "interaktívnu" kategorizáciu sa používajú v kritických aplikáciách, kde sa predpokladá, že efektívnosť automatickej kategorizácie bude nižšia než efektívnosť experta [Larkey, Croft 1996]. Ide napr. o prípady, kedy je kvalita trénovacích dát nízka, resp. kedy nemožno s dostatočnou spoľahlivosťou zaručiť, že trénovacie dokumenty budú reprezentatívnou vzorkou kategorizovaných dokumentov.

Klasifikátor $\hat{\Phi}$ je možné použiť dvoma spôsobmi. Pre daný dokument d_i môže klasifikátor nájsť všetky kategórie $c_j \in C$ do ktorých má byť dokument zaradený (tzv. klasifikácia *orientovaná podľa dokumentov*), resp. je možné pre danú triedu $c_j \in C$ nájsť všetky relevantné dokumenty $d_i \in D$ (klasifikácia *orientovaná podľa kategórií*). Toto rozdelenie metód je potrebné, pretože množina D alebo C nemusí byť úplne známa na začiatku vytvárania klasifikátora. Klasifikácia orientovaná podľa dokumentov je teda použitá napr. v prípade ak jednotlivé dokumenty z D nie sú k dispozícii v rovnakom čase (napr. pri filtrovaní textov). Klasifikácia orientovaná podľa kategórií je potrebná, ak je do množiny C pridaná nová kategória $c_{|C|+1}$ potom, čo už boli niektoré dokumenty klasifikované podľa C [Larkey 1999]. Aj keď niektoré metódy je možné použiť iba v jednom prípade, väčšinu používaných metód je možné použiť obidvoma spôsobmi.

3.2 Aplikácie kategorizácie textov

Nasledujúce podkapitoly uvádzajú niektoré základné aplikácie kategorizácie textov. Medzi ďalšie aplikácie, ktoré nie sú uvedené, patrí kategorizovanie nahovorených textov v kombinácii s rozpoznávaním reči, klasifikovanie multimediálnych dokumentov podľa textových titulkov, alebo identifikovanie autora a žánru literárnych textov. Kategorizáciu textov je ďalej možné využiť aj pri lingvistickom spracovaní prirodzeného jazyka (NLP) napr. pre dezambiguáciu významu slov v texte, morfologické značkovanie, alebo identifikovanie fráz.

3.2.1 Automatické indexovanie pre vyhľadávanie informácií

Pri používaní informačných (a knižničných) systémov sa zistilo, že vyhľadávanie informácií môže byť efektívnejšie ak sa pre každý dokument vytvorí index zložený z metadát (napr. informácií o autorovi) a tzv. *indexových termov* [Maron 1961; Meadow 1992].

Pri manuálnom vyváraní indexu musí najprv indexátor určiť hlavný význam textu, ktorý potom reprezentuje množinou indexových termov (slov, alebo fráz). Indexovanie jedného dokumentu je zvyčajne časovo obmedzené, takže nie je možné určiť význam podľa celého textu. Z tohto dôvodu zvyčajne indexátor určí význam tzv. *skenovaním* textu, ktoré je založené na percepčných procesoch zameraných na aktuálny obsah dokumentu

a konceptuálnych procesoch, ktoré sú zamerané na globálne znalosti, ktoré nie sú v dokumente priamo zahrnuté (napr. znalosti implikované autorom, alebo doménové znalosti indexátora) [Cooper 1969; Masson 1982; Farrow 1991].

Po stanovení hlavného obsahu dokumentu musí indexátor zvoliť indexové termy ktoré budú dokument reprezentovať. Keďže index je vytváraný pre potreby vyhľadávania, indexátor musí brať do úvahy aj informačné potreby užívateľov. Je však obtiažne zvoliť "najlepšie" termy zo všetkých možných termov charakterizujúcich obsah textu. Viacero štúdií dokazuje nízku zhodu medzi termami priradenými tomu istému dokumentu rôznymi indexátormi [Cooper 1969; Tarr, Borko 1974]. Navyše, jeden indexový term môže mať viacero významov v rôznych kontextoch, čo spolu s nekonzistenciou indexovania vedie k nízkej presnosti a úplnosti vyhľadávania (spôsob ich výpočtu je popísaný nižšie, v časti 3.3.1).

Konzistenciu indexovania dokumentov medzi indexátormi je možné zvýšiť použitím tzv. *kontrolovaných slovníkov*. Kontrolované slovníky definujú zoznam možných indexových termov a vzťahy medzi jednotlivými termami pre danú doménu. Pre rýchle vyhľadávanie sú záznamy indexových termov hierarchicky štruktúrované (napr. v podobe tezauru). Aby sa zredukovala nejednoznačnosť, každý záznam v slovníku môže obsahovať vymedzenie použitia spolu s odkazmi na súvisiace termy. Použitie kontrolovaných slovníkov však neodstraňuje problém výberu termov, pretože je potrebné neustále aktualizovať slovník podľa obsahu indexovaných dokumentov. Aj keď táto činnosť môže byť nákladná, výhodou by malo byť efektívnejšie vyhľadávanie informácií.

Manuálne indexovanie môže byť časovo náročné. Navyše, pre zabezpečenie konzistencie musia byť indexátori trénovaní vo výbere termov zo slovníka špecifického pre danú doménu. Zvyšujúce sa nároky na množstvo indexovaných dokumentov a konzistenciu indexovania viedli k vývoju automatických (resp. semi–automatických) metód indexovania. Automatické indexovanie dokumentov podľa kontrolovaného slovníka je možné považovať za aplikáciu kategorizácie textov, kde jednotlivým kategóriám zodpovedajú indexové termy. Zvyčajne automatické indexovanie dokumentov vyžaduje viacnásobnú klasifikáciu do viacerých tried (keďže význam dokumentu môže byť reprezentovaný viacerými termami). Dokumenty sú indexované postupne, tak ako sú publikované, takže sa používa klasifikácia orientovaná podľa dokumentov. Je možné využiť aj klasifikáciu orientovanú podľa kategórií, pretože po rozšírení slovníka je potrebné preindexovať dokumenty podľa nových termov [Larkey 1999].

Klasifikáciu textov je možné využiť aj na indexovanie webových stránok. Pri vyhľadávaní na internete môže byť pre užívateľa jednoduchšie prechádzať hierarchický katalóg stránok, ktorý mu umožní obmedziť svoje ďalšie vyhľadávanie iba na určitú tematickú kategóriu. Výhodou automatického kategorizovania stránok je možnosť klasifikovať veľkú časť webu, ktorú by nebolo možné indexovať manuálne. Kategorizácia webových stránok sa ďalej vyznačuje nasledujúcimi vlastnosťami:

- *hierarchická štruktúra kategórií*: umožňuje napr. dekompozíciu klasifikačného problému na jednoduchšie podproblémy, ktoré zodpovedajú klasifikácii na jednotlivých úrovniach hierarchie [Ruiz, Srinivasan 2002].
- *hypertextové prepojenia*: odkazy medzi stránkami môžu poskytnúť dodatočné informácie o vzájomnej relevancii a obsahu stránok, ktoré môžu byť využité pri klasifikácií [Yang a kol. 2002].

3.2.2 Filtrovanie textov

Filtrovanie textov je činnosť pri ktorej sa klasifikujú dokumenty asynchrónne odosielané cez informačný kanál medzi producentom informácií a spotrebiteľom. Charakteristickým príkladom sú novinové správy, ktoré sú odosielané tlačovou agentúrou jednotlivým redakciám [Hayes, Weinstein 1990]. V tomto prípade by mal systém pre filtrovanie textov zamedziť odosielaniu nevyžiadaných správ (napr. všetkých správ ktoré nie sú o ekonomike, pre ekonomické noviny). Filtrovanie textov môže byť považované za kategorizáciu textov do dvoch disjunktných tried podľa relevantnosti pre daného spotrebiteľa. Systém môže dodatočne zaradiť relevantné správy do tematických kategórií, takže úloha filtrovania textov môže byť rozšírená na klasifikáciu do viacerých tried. Podobne je možné vytvoriť filter e-mailových správ, ktorý bude filtrovať nevyžiadané správy (*junk mail*) a dodatočne klasifikuje poštu podľa oblasti záujmu užívateľa [Androutsopoulos 2000; Carreras, Màrquez 2001].

Zvláštnym prípadom filtrovania textov je sledovanie udalostí (*event tracking*) podľa označených správ [Yang a kol. 2000]. Úloha sledovania udalostí je definovaná ako automatické prideľovanie predefinovaných označení udalostí dokumentom, ktoré sú chronologicky prezentované systému. Na rozdiel od bežnej tematickej definície kategórií, udalosti sú viazané na určité miesto a čas[15]. Systém musí rozlíšiť jednotlivé udalosti, bez ohľadu na to či patria do jednej tematickej kategórie, alebo nie. Pre definíciu sledovania udalostí ďalej platia nasledujúce obmedzenia:

každá udalosť je definovaná množinou pozitívnych inštancií (dokumentov) ktoré sú manuálne označené pred začiatkom sledovania. Nie sú prístupné žiadne ďalšie znalosti.

okrem malého počtu vopred označených pozitívnych inštancií, systém nemá žiadnu spätnú väzbu od užívateľa o relevancii sledovaných dokumentov.

3.3 Metódy strojového učenia pre automatickú kategorizáciu textov

Jedny z prvých metód použitých pri vytváraní klasifikátorov pre automatickú kategorizáciu textov boli metódy založené na postupoch *znalostného inžinierstva* [Hayes, Weinstein 1990]. Pri znalostnom prístupe je klasifikátor tvorený expertným systémom založeným napr. na rozhodovacích pravidlách,

[15] Napr. "nehoda letu EgyptAir-990" je udalosť, ale nie tematickou kategóriou, naopak "letecké nešťastia" sú kategóriou, ale nie udalosťou.

ktoré sú manuálne vytvorené pre jednotlivé kategórie na základe doménových znalostí o klasifikovaných dokumentoch.

Najväčšou nevýhodou znalostného prístupu je nákladný proces získavania znalostí. Pravidlá expertného systému vytvára znalostný inžinier za pomoci doménového experta, ktorý vie zaradiť dokumenty z danej domény do predefinovaných kategórií. Obidvaja experti musia spolupracovať pri modifikácií (alebo znovu vytvorení) systému vždy, keď dôjde ku zmene kategórií (domény).

Alternatívny spôsob vytvárania klasifikátorov je založený na využití metód strojového učenia [Mitchell 1996; Sebastiani 2002]. Na rozdiel od znalostného prístupu je klasifikátor vytvorený automaticky, induktívnym procesom, na základe charakteristík ktoré sú extrahované z množiny manuálne klasifikovaných dokumentov. Podľa definície strojového učenia ide o tzv. *kontrolované učenie*, pretože proces vytvárania klasifikátora je "kontrolovaný" znalosťou kategórií, ktoré boli priradené jednotlivým dokumentom.

Učenie klasifikátora vyžaduje počiatočnú množinu dokumentov $\Omega = \{d_1, ..., d_{|\Omega|}\}$ ktorá je manuálne klasifikovaná expertom do kategórií z množiny $C = \{c_1, ..., c_{|C|}\}$ (tzn., že hodnota funkcie $\Phi: \Omega \times C \rightarrow \{true, false\}$ je známa pre každý dokument $d_i \in \Omega$ a kategóriu $c_j \in C$). Aby bolo možné vyhodnotiť efektívnosť výsledného klasifikátora, množina Ω je rozdelená pred začiatkom učenia na dve podmnožiny (ktoré nemusia byť rovnako veľké):

- *trénovaciu množinu* $D = \{d_1, ..., d_{|D|}\}$, 7na základe ktorej sa určia charakteristiky dokumentov jednotlivých kategórií pri induktívnom učení klasifikátora $\hat{\Phi}$ a na
- *testovaciu množinu* $T = \{d_{|D|+1}, ..., d_{|\Omega|}\}$, ktorá slúži na otestovania efektívnosti vytvoreného klasifikátora.

Keďže učenie je založené na charakteristikách trénovacích dokumentov, odhad efektívnosti podľa dokumentov z množiny D by bol nereálne optimistický. Z tohto dôvodu nesmú byť testovacie dokumenty použité žiadnym spôsobom pri učení.

Pri učení je niekedy vhodné množinu D ďalej rozdeliť na trénovaciu množinu $\{d_1, ..., d_{|D|-|V|}\}$ a *validačnú množinu* $V = \{d_{|D|-|V|+1}, ..., d_{|D|}\}$, ktorú je možné využiť pri optimalizácii parametrov klasifikátora.

3.3.1 Spôsoby vyhodnocovania efektívnosti klasifikátorov

Vyhodnotenie efektívnosti klasifikátora je založené na zhode medzi predikciou $\hat{\Phi}(d_i, c_j)$ a skutočnou hodnotou $\Phi(d_i, c_j)$ určenou pre všetky dokumenty $d_i \in T$ (resp. $d_i \in V$). Kvantitatívne je možné efektívnosť vyhodnotiť podľa *presnosti* a *návratnosti* (podobne ako pri vyhodnocovaní metód pre vyhľadávanie informácií).

Pre klasifikáciu dokumentov z triedy c_j je presnosť π_j definovaná ako podmienená pravdepodobnosť $\mathrm{P}(\Phi(d_i, c_j) = true \mid \hat{\Phi}(d_i, c_j) = true)$.

Analogicky, návratnosť ρ_j je definovaná ako pravdepodobnosť $P(\hat{\Phi}(d_i, c_j) = true \mid \Phi(d_i, c_j) = true)$. Pravdepodobnosti π_j a ρ_j je možné odhadnúť podľa kontingenčnej tabuľky pre klasifikáciu testovacích dokumentov (Tab. 3.1) ako:

$$\pi_j = \frac{TP_j}{TP_j + FP_j} \tag{3.1}$$

$$\rho_j = \frac{TP_j}{TP_j + FN_j} \tag{3.2}$$

kde TP_j a TN_j (FP_j a FN_j) je počet správne (nesprávne) predikovaných pozitívnych a negatívnych príkladov triedy c_j.

Tab. 3.1 Kontingenčná tabuľka pre kategóriu c_j

	$\Phi(d_i, c_j) = true$	$\Phi(d_i, c_j) = false$
$\hat{\Phi}(d_i, c_j) = true$	TP_j	FP_j
$\hat{\Phi}(d_i, c_j) = false$	TN_j	FN_j

Celkovú presnosť a návratnosť pre všetky triedy c_j ($j = 1 .. |C|$) je možné určiť dvoma spôsobmi: tzv. *mikro-spriemerňovaním*

$$\pi^{\mu} = \frac{TP}{TP + FP} = \frac{\sum_{j=1}^{|C|} TP_j}{\sum_{j=1}^{|C|} (TP_j + FP_j)} \tag{3.3}$$

$$\rho^{\mu} = \frac{TP}{TP + FN} = \frac{\sum_{j=1}^{|C|} TP_j}{\sum_{j=1}^{|C|} (TP_j + FN_j)} \tag{3.4}$$

alebo makro-spriemerňovaním:

$$\pi^{M} = \frac{\sum_{j=1}^{|C|} \pi_j}{|C|} \tag{3.5}$$

$$\rho^{M} = \frac{\sum_{j=1}^{|C|} \rho_j}{|C|} \tag{3.6}$$

Použitie konkrétneho spôsobu vyhodnocovania závisí na aplikácii. Mikro-spriemerňovanie je viac ovplyvnené klasifikáciou často sa vyskytujúcich tried a naopak makro-spriemerňovanie lepšie odráža presnosť klasifikátora na kategóriách s malým počtom pozitívnych príkladov.

Presnosť a návratnosť môžu byť skombinované do jednej miery efektívnosti napr. podľa funkcie:

$$F_{\beta} = \frac{(\beta^2 + 1)\pi\rho}{\beta^2\pi + \rho} \tag{3.7}$$

kde parameter β vyjadruje do akej miery je hodnota funkcie F_β ovplyvnená hodnotou π a ρ [van Rijsbergen 1979]. Najčastejšie sa používa funkcia F_1, ktorá kombinuje presnosť a návratnosť s rovnakou váhou, t.j.

$$F_1 = F = \frac{2\pi\rho}{\pi + \rho} \tag{3.8}$$

Pri nedostatku trénovacích dát[16] je možné odhadnúť efektívnosť tzv. *krížovou validáciou* pri ktorej sa Ω rozdelí na k testovacích množín $T_1, ..., T_k$. Pre každú z týchto množín T_i, $i = 1, ..., k$, sa naučí klasifikátor $\hat{\Phi}_i$ na trénovacej množine $\Omega - T_i$ a následne otestuje na množine T_i. Výsledný odhad efektívnosti sa potom určí spriemernením výsledkov testovania klasifikátorov $\hat{\Phi}_i$, $i = 1, ..., k$ na zodpovedajúcich testovacích množinách. Krížovú validáciu je možné využiť aj pri optimalizovaní parametrov namiesto validačnej množiny.

3.3.2 Prehľad metód pre klasifikáciu textov

3.3.2.1 Naivný Bayesov Klasifikátor

Naivný Bayesov klasifikátor patrí medzi pravdepodobnostné modely, ktoré klasifikujú nové dokumenty na základe podmienenej pravdepodobnosti $\mathrm{P}(c_j \mid d)$, tj. pravdepodobnosti, že dokument d má byť zaradený do kategórie c_j. Pri učení klasifikátora je na vstupe trénovacia množina dokumentov $D = \{d_1, ..., d_{|D|}\}$, ktoré sú klasifikované do množiny tried $C = \{c_1, ..., c_{|C|}\}$. Každý z dokumentov d_i je reprezentovaný ako postupnosť termov[17], ktoré sa vyskytli v dokumente $d_i = \langle t_{i,1}, ..., t_{i,|d|} \rangle$, kde $|d|$ je dĺžka dokumentu d_i.

Na základe trénovacej množiny D nie je možné určiť odhad podmienenej pravdepodobnosti $\mathrm{P}(c_j \mid d)$ pre neznámy dokument d priamo. Je však možné priamo určiť pravdepodobnosť kategórie $\mathrm{P}(c_j)$ a pravdepodobnosť výskytu termu t v dokumente za predpokladu že dokument patrí do kategórie c_j, t.j. $\mathrm{P}(t \mid c_j)$. Naivný Bayesov klasifikátor využíva odhad týchto pravdepodobností určených podľa trénovacej množiny dokumentov pre určenie pravdepodobnosti $\mathrm{P}(c_j \mid d)$. V prvom kroku je potrebné skombinovať pravdepodobnosti $\mathrm{P}(t \mid c_j)$ jednotlivých termov vyskytujúcich sa v dokumente tak, aby sa získal odhad pravdepodobnosti $\mathrm{P}(d \mid c_j)$ pre celý dokument. Za predpokladu, že každý term sa vyskytuje v dokumente štatisticky nezávisle od ostatných termov je možné určiť $\mathrm{P}(d \mid c_j)$ ako súčin:

$$\mathrm{P}(d \mid c_j) = \prod_{i=1}^{|d|} \mathrm{P}(t_i \mid c_j) \tag{3.9}$$

[16] kedy nie je možné zvoliť dostatočne reprezentatívnu testovaciu množinu

[17] Nezohľadňuje sa poradie teremov, ale každý term sa môže vyskytovať viac krát.

Pravdepodobnosť $P(c_j \mid d)$ je potom určená z $P(d \mid c_j)$ podľa Bayesovho pravidla:

$$P(c_j \mid d) = \frac{P(c_j)P(d \mid c_j)}{P(d)} \tag{3.10}$$

Ak má byť dokument klasifikovaný iba do jednej triedy, na základe Bayesovho pravidla (3.10 sa vyberie trieda, pre ktorú je pravdepodobnosť $P(c_j \mid d)$ maximálna. Pravdepodobnosť $P(d)$ je možné v tomto prípade vynechať, keďže neovplyvňuje rozhodovanie podľa maximálnej hodnoty $P(c_j \mid d)$. Pre viacnásobnú klasifikáciu je dokument zaradený do triedy c_j ak pravdepodobnosť $P(c_j \mid d)$ prekročí zvolenú prahovú hodnotu, napr. $0{,}5$. Ďalším spôsob je nahradiť viacnásobnú klasifikáciu do $|C|$ tried je učením $|C|$ binárnych klasifikátorov, ktoré klasifikujú dokumenty do dvoch tried:

$+1 \rightarrow$ dokument patrí do c_j resp.

$-1 \rightarrow$ dokument nepatrí do cj, pričom $P(+1 \mid d) = 1 - P(-1 \mid d)$.

Pravdepodobnosť $P(c_j)$ je možné odhadnúť na základe trénovacej množiny D ako „počet dokumentov z D patriacich do triedy c_j“ delený “celkovým počtom dokumentov v D“, t.j.

$$P(c_j) = \frac{|\{d \in c_j \mid d \in D\}|}{|D|} \tag{3.11}$$

Pravdepodobnosť $P(t \mid c_j)$ je určená ako „počet výskytov termu t vo všetkých dokumentoch z triedy c_j“ delený “celkovým počtom výskytov termu vo všetkých dokumentoch z D”. Ak sa niektorý term t z dokumentu d nevyskytol ani raz v trénovacích dokumentoch patriacich do triedy c_j, uvedený odhad pravdepodobnosti $P(t \mid c_j)$ by sa rovnal 0, čo by viedlo k $P(c_j \mid d) = 0$. Dokument by teda nebol zaradený do c_j aj napriek tomu, že by boli pravdepodobnosti $P(t \mid c_j)$ ostatných termov veľké. Preto je počet výskytov zvýšený o 1 a pravdepodobnosť je normalizovaná voči celkovému počtu výskytov vrátane pridaných počtov, t.j.:

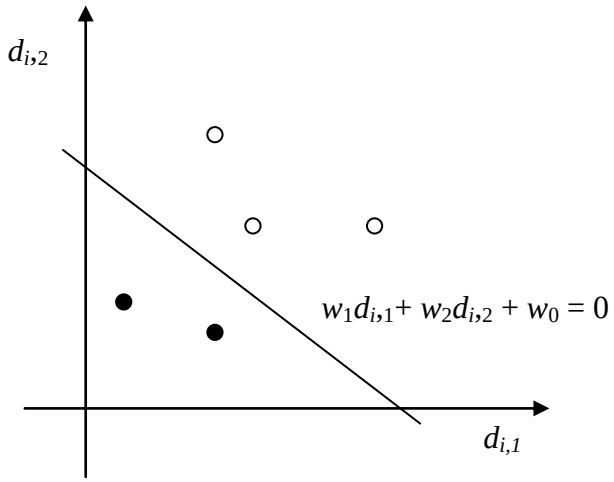
$$P(t \mid c_j) = \frac{1 + \sum_{d \in c_j \mid d \in D} n_d}{|V| + \sum_{d \in D} n_d} \tag{3.12}$$

kde $|V|$ je počet všetkých termov, ktoré sa vyskytli v trénovacej množine, a n_d je počet výskytov termu t v dokumente d.

Okrem postupnosti termov je možné dokument reprezentovať aj ako binárny vektor $d_i = (b_{i,1}, \ldots, b_{i,|V|})$, kde $b_{i,t} = 1$ ak sa term t v dokumente vyskytol, alebo 0 ak sa nevyskytol (viď. časť 2.7.3 o vektorovej reprezentácii dokumentov, schéma *bnn*). Pre binárnu reprezentáciu je potom $P(t \mid c_j)$ vypočítané ako „počet dokumentov z triedy c_j, v ktorých sa vyskytol term t“ delený “celkovým počtom dokumentov z triedy c_j” (pravdepodobnosť $P(c_j)$ je určená podľa vzťahu 3.11). V praxi však dosahuje binárna verzia horšie výsledky pre klasifikáciu textov, čo je odôvodňované tým, že nezohľadňuje frekvenciu výskytu termu v dokumente.

3.3.2.2 Lineárne klasifikátory

Lineárne klasifikátory vychádzajú z geometrickej interpretácie klasifikačného problému, pri ktorej sú dokumenty reprezentované ako vektory v M–rozmernom priestore dokumentov, t.j. $d_i \in \mathbb{R}^M = (d_{i,1}, ..., d_{i,M})$, kde K je množina termov ($M = |K|$). Každá dimenzia tohto priestoru zodpovedá jednému termu. Hodnoty zložiek vektora $d_{i,j}$ môžu byť binárne (t.j. 0 – term sa nevyskytol v dokumente, resp. 1 – ak sa vyskytol) resp. môžu byť odvodené z frekvencie výskytu termu v dokumente, alebo kombinácii výskytu termu v dokumente a výskytu v celej trénovacej množine (opäť pozri časť 2.7.3 o vektorovej reprezentácii dokumentov).



Obr. 3.1 Lineárny klasifikátor v 2-d priestore.

Lineárne klasifikátory sú reprezentované ako hyperplocha (t.j. priamka v dvojrozmernom priestore, rovina v trojrozmernom priestore, atď.), ktorá je definovaná rovnicou:

$$\sum_{j=1}^{|V|} d_{i,j} w_j + w_0 = 0 \qquad (3.13)$$

Dokumenty sú klasifikované do dvoch tried označených $+1$ a -1, podľa toho na ktorej strane hyperplochy sa nachádza vektor klasifikovaného dokumentu (pre klasifikáciu do viacerých tried je potrebné dekomponovať úlohu na učenie viacerých binárnych klasifikátorov). Rozhodovaciu funkciu je teda možné zapísať v tvare:

$$f(d_i) = \sum_{j=1}^{|V|} d_{i,j} w_j + w_0 \qquad (3.14)$$

Dokument je zaradený do triedy $+1$ ak $f(d_i) > 0$, resp. do triedy -1 inak. Pri učení lineárnych klasifikátorov je teda potrebné odhadnúť hodnoty parametrov $w = (w_1, ..., w_M)$ a w_0, tak aby sa minimalizovala chyba klasifikácie na trénovacích dátach. Užitočnou vlastnosťou lineárnych klasifikátorov je to, že je ich možné ľahko interpretovať. Parametre klasifikátora $w_1, ..., w_M$ zodpovedajú jednotlivým termom. Ak je "váha" w_t termu t kladná, term t "hlasuje" za zaradenie dokumentu do triedy $+1$, ak je menšia než 0, "hlasuje" proti (za predpokladu že všetky zložky vektora dokumentu sú kladné tak ako

je to v prípade binárnych vektorov alebo vektorov s frekvenciami termov). Termy pre ktoré sa $w_t = 0$, nemajú na klasifikáciu žiaden vplyv (pri klasifikácií dokumentov má zvyčajne iba malá časť termov nenulové váhy, t.j. klasifikácia závisí iba na malej podmnožine z množiny všetkých termov, ktoré sa vyskytli v celej trénovacej množine dokumentov).). Jednotlivé algoritmy pre učenie lineárnych klasifikátorov sa odlišujú spôsobom ako sa vypočítajú parametre $w = (w_1, ..., w_M)$ a w_0.

Perceptron

Algoritmus perceptróna [Rosenblatt 1988; Ng a kol. 1997] sa pokúša vytvoriť deliacu hyperplochu minimalizovaním vzdialenosti medzi chybne klasifikovanými dokumentmi d_i a medzi hyperplochou určenou $f(d_i) = 0$. Učenie perceptróna prebieha inkrementálne *stochastickým gradientovým zostupom*. Pri učení sú trénovacie dokumenty $d_i \in D$ prezentované algoritmu postupne v náhodnom poradí, pričom parametre w a w_0 sú zmenené pri každom výskyte chybne klasifikovaného dokumentu podľa:

$$\begin{pmatrix} \boldsymbol{w} \\ w_0 \end{pmatrix} \leftarrow \begin{pmatrix} \boldsymbol{w} \\ w_0 \end{pmatrix} + \gamma \begin{pmatrix} y_i d_i \\ y_i \end{pmatrix} \tag{3.15}$$

kde parameter $\gamma > 0$ je rýchlosť učenia, ktorá udáva ako veľmi ovplyvní váhy chybne klasifikovaný dokument v každom kroku učenia. Na začiatku učenia sú váhy w a w_0 inicializované náhodne.

Winnow algoritmus

Podobne ako Perceptrón je Winnow algoritmus inkrementálny. Na začiatku sú všetky váhy $w = (w_1, ..., w_M)$ inicializované na rovnakú kladnú hodnotu (napr. $1/M$). Ak je vstupný dokument d_i chybne klasifikovaný podľa aktuálnych váh, váhy sa zmenia podľa nasledujúceho pravidla:

- ak dokument d_i patrí do triedy $+1$, váhy všetkých termov, ktoré sa vyskytli v dokumente d_i sa zvýšia vynásobením konštantou $\alpha > 1$.
- ak dokument d_i patrí do triedy -1, váhy všetkých termov, ktoré sa vyskytli v dokumente d_i sa znížia vynásobením konštantou $0 < \beta < 1$,

kde parametre α a β určujú rýchlosť učenia podobne ako parameter γ pri Perceptróne. Takýmto spôsobom sa získa vektor váh w, ktorého všetky zložky sú kladné (t.j. výskyt termu môže „hlasovať len za“ zaradenie do triedy $+1$). Rozšírené verzia algoritmu umožňuje aj záporné váhy, pričom sa pri učení udržiavajú dve váhy $w_{j,+}$ a $w_{j,-}$ pre každý term, pričom výsledná váha pre term j sa určí ako $w_j = w_{j,+} - w_{j,-}$. Podobne ako pri základnej verzii algoritmu sú zmenené váhy všetkých termov, ktoré sa vyskytli v dokumente, ktorý je na vstupe a je chybne klasifikovaný podľa aktuálnych váh. Váhy sa menia podľa nasledujúceho pravidla:

- ak dokument d_i patrí do triedy $+1$, kladné váhy $w_{j,+}$ sa zvýšia vynásobením konštantou α a záporné sa znížia $w_{j,-}$ vynásobením β
- ak dokument d_i patrí do triedy -1, kladné váhy $w_{j,+}$ sa znížia vynásobením konštantou β a záporné sa zvíšia $w_{j,-}$ vynásobením α.

V oboch prípadoch je potrebné určiť prahovú hodnotu w_0 napr. testovaním

rôznych hodnôt krížovou validáciou.

3.3.2.3 K–najbližších susedov

Učenie klasifikátorov založených na pravidle k–najbližších susedov je naozaj jednoduché – pri učení sa len odpamätajú všetky trénovacie príklady z D (preto sa niekedy tieto algoritmy označujú ako „lazy“). Pri klasifikácii nového dokumentu sa podľa metriky alebo funkcie podobnosti určí k najbližších (najpodobnejších) dokumentov z D a klasifikátor zaradí nový dokument do triedy určenej podľa klasifikácie susedov.

Najdôležitejšou časťou algoritmu je metrika na meranie vzdialenosti vo vektorovom priestore dokumentov, resp. funkcia podobnosti. Keďže dokumenty sú reprezentované ako reálne vektory, je možné použiť Euklidovskú metriku, ale významovej podobnosti dokumentov lepšie zodpovedá kosínusová funkcia podobnosti. Je možné použiť aj ďalšie podobné funkcie navrhnuté pre vyhľadávanie informácií, kde funkcia podobnosti určuje skóre dokumentu voči dopytu.

V prípade, že má byť príklad klasifikovaný iba do jednej triedy, vyberie sa najčastejšie sa vyskytujúca trieda. Okrem toho môže byť „hlasovanie“ jednotlivých susedov vážené podľa ich vzdialenosti voči klasifikovanému dokumentu, t.j. trieda najbližšieho suseda má najväčšiu váhu atď. Ak je potrebné klasifikovať dokumenty do viacerých tried, určí sa celkové skóre (suma pre všetkých susedov) pre každú triedu c_j a ak je skóre väčšie ako zvolená prahová hodnota, príklad sa zaradí do c_j.

Hodnota parametra k sa určuje empiricky, testovaním na validačnej množine príkladov, resp. pomocou *krížovej validácie*. Pri krížovej validácii sa trénovacia množina D rozdelí na n podmnožín $D_1, ..., D_n$, najprv sa naučí klasifikátor na dátach z množín $D_1 ..., D_{n-1}$ a otestuje sa D_n, potom na $D_1 ..., D_{n-2}$, D_n a otestuje sa D_{n-1}, atď. Takto sa postupne otestujú všetky podmnožiny D_i, $i = 1, ..., n$.

Najväčšou nevýhodou algoritmov založených na pravidle k–najbližších susedov je ich veľká pamäťová a časová náročnosť pri klasifikácii, keďže je potrebné uchovať a vypočítať vzdialenosť voči všetkým trénovacím príkladom. Jedným zo spôsobov ako túto náročnosť znížiť je použitie redukčných algoritmov [Wilson, Martinez 2000]. Úlohou redukčných algoritmov je vybrať podmnožinu trénovacích príkladov $S \subset D$ tak, aby sa čo najviac zachovala presnosť klasifikácie podľa S oproti klasifikácii podľa všetkých trénovacích príkladov. V niektorých prípadoch sa môže presnosť klasifikácie zvýšiť, pretože redukčné algoritmy môžu odstrániť z trénovacej množiny šum, alebo vyhladiť hraničné plochy oddeľujúce jednotlivé triedy.

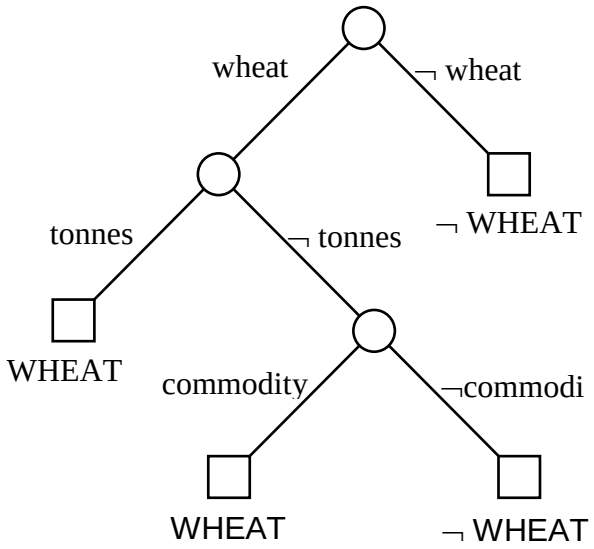
Pri redukcii je možné postupovať od prázdnej množiny S do ktorej sa postupne (*inkrementálne*) pridajú trénovacie príklady z D vybrané na základe zvoleného kritéria. Pri inkrementálnych metódach môže byť rovnaké kritérium použité aj pri výbere ďalších dokumentov, ktoré sa vyskytli neskôr po skončení učenia na trénovacej množine. Ďalšou výhodou oproti neinkrementálnym algoritmom je menšia pamäťová a časová náročnosť pri učení, keďže algoritmus môže pri pridávaní nových inštancií ignorovať vynechané príklady. Nevýhodou inkrementálnych metód je ich závislosť

na poradí v akom sú prezentované jednotlivé dokumenty. Inkrementálne metódy môžu byť taktiež citlivé na šum, hlavne pri rozhodovaní na začiatku učenia, kedy ešte nemajú dostatočné množstvo informácií o učených kategóriách.

Dekrementálne algoritmy využívajú pri redukcii opačný postup, pri ktorom sa z množiny $S = D$ postupne odstraňujú vybrané trénovacie príklady. Aj v tomto prípade je poradie odstránených príkladov dôležité. Algoritmus má však k dispozícii všetky trénovacie príklady, takže môže v každom kroku lepšie rozhodnúť, ktorý príklad bude odstránený. Aj keď dekrementálne algoritmy majú väčšie pamäťové a časové nároky pri učení, často vedú k lepšej redukcii uložených príkladov a teda k zodpovedajúcemu zrýchleniu pri klasifikovaní nových dokumentov.

3.3.2.4 Rozhodovacie stromy a pravidlá

Príklad rozhodovacieho stromu je uvedený na Obr. 3.2. Každý nelistový uzol stromu je označený testom, ktorý rozdeľuje dokumenty podľa výskytu jedného termu. Listové uzly sú označené priradením triedy. Klasifikácia prebieha rekurzívne od koreňového uzla, zvolením vetvy podľa testu až pokiaľ sa nedosiahne listový uzol, t.j. pre uvený príklad, dokumenty sú zaradené do triedy WHEAT ak obsahujú term „wheat" a zároveň obsahujú term „tonnes" alebo „commodity". Rozhodovacie stromy sú učené metódou "zhora nadol". Počiatočný strom je tvorený iba jedným uzlom, ktorý pokrýva všetky trénovacie dokumenty. Ak nastane prípad, že všetky príklady pokryté daným uzlom majú rovnakú triedu, delenie množiny dokumentov nie je potrebné a uzol sa označí ako listový. Inak algoritmus priradí uzlu logický test, ktorý rozdelí príklady na disjunktné podmnožiny, pre ktoré sa vytvoria nové uzly stromu. Celý proces sa potom rekurzívne opakuje na nových potomkoch, až pokiaľ sa nedosiahne úplné oddelenie príkladov jednotlivých tried.



Obr. 3.2 Príklad rozhodovacieho stromu.

Výber testu pre nelistové uzly je založený na hodnotiacej funkcii, ktorá zodpovedá "variabilite" tried v podmnožinách vytvorených po rozdelení uzla.

Pri delení sa otestujú všetky možné testy (pre všetky termy $w_t \in K$ a všetky hodnoty θ_t) a vyberie sa test, ktorý vedie k čo najväčšej redukcii hodnotiacej funkcie (tzn. k čo najväčšiemu počtu príkladov z jednej triedy v každej vytvorenej podmnožine). Ako hodnotiacu funkciu pre výber testov je možné použiť napr. *Gini* index (3.16), alebo entropiu (3.17):

$$G(T) = 1 - \sum_{j=1}^{|C|} \mathrm{P}(c_j \mid T)^2 \tag{3.16}$$

$$H(T) = -\sum_{j=1}^{|C|} \mathrm{P}(c_j \mid T) \log_2 \mathrm{P}(c_j \mid T) \tag{3.17}$$

kde $\mathrm{P}(c_j \mid T)$ je pravdepodobnosť, že príklad patriaci uzlu (množine T) bude mať triedu c_j, ktorá je určená ako „počet dokumentov z triedy c_j patriacich do T“ delený “celkovým počtom príkladov v T“ .

Uvedená metóda učenia stromov vytvorí strom, ktorý správne klasifikuje všetky trénovacie príklady (okrem situácie, keď sa v trénovacej množine vyskytujú navzájom si protirečiace príklady). Pri nedostatočnom počte trénovacích príkladov, resp. ak sa v dátach vyskytuje šum, učenie perfektného stromu môže viesť k preučeniu, t.j. nulovej chybe na trénovacích dátach, ale veľkej chybe na testovacích dátach. Aby sa zabránilo preučeniu, vygenerovaný strom sa zjednoduší tzv. orezávaním pri ktorom sa odstránia vetvy, ktoré sú špecifické iba pre danú trénovaciu množinu príkladov.

Pri orezávaní sa otestuje nahradenie každého nelistového podstromu jednoduchým listovým uzlom, ktorému je priradená trieda ktorá sa najčastejšie vyskytuje medzi pokrytými príkladmi. Zjednodušenie sa príjme ak sa po orezaní nezhorší odhad skutočnej chyby klasifikácie, ktorú je možné odhadnúť napr. testovaním na validačnej množine príkladov odlišnej od trénovacej množiny (algoritmus Reduced Error Prunning – REP). Nevýhodou tejto metódy je že vyžaduje dodatočné dáta pre validáciu.

Ďalšou možnou reprezentáciou tried pri kategorizácii textov je zoznam rozhodovacích pravidiel v tvare:

if ⟨*výraz*⟩ **then** c_j (3.18)

kde c_j je označenie triedy, do ktorej bude dokument zaradený ak spĺňa podmienku výrazu. Podmienka je vyjadrená ako konjunkcia testov, ktoré ohraničujú výskyt jedného termu podobne ako pri nelistových uzloch rozhodovacích stromov. Každý rozhodovací strom môže byť nahradený zoznamov rozhodovacích pravidiel, pričom sa vytvorí jedno pravidlo pre každý listový uzol. Podmienka pravidla vznikne spojením všetkých testov nelistových uzlov na ceste od koreňa k danému listu. Zoznam pravidiel generovaný z rozhodovacieho stromu pokrýva celý priestor dokumentov, t.j. každý dokument bude spĺňať podmienku práve jedného pravidla.

Vo všeobecnosti však dokument nemusí byť pokrytý žiadnym pravidlom zoznamu resp. jeden dokument môže byť pokrytý viacerými pravidlami. Tieto konflikty je potrebné vyriešiť napr. usporiadaním pravidiel podľa pravdepodobnosti triedy c_j a klasifikovaním nepokrytých príkladov do najpravdepodobnejšej triedy.

Pri učení rozhodovacích pravidiel sa algoritmus snaží nájsť pravidlo, ktoré by pokrývalo čo najviac príkladov z jednej triedy c_j a zároveň čo najmenej príkladov z ostatných tried [Clark, Niblett 1989; Quinlan 1996; Apté a kol. 1994]. Výraz pravidla sa postupne špecifikuje pridaním podmienok $d_{i,t} \leq \theta_t$, $d_{i,t} > \theta_t$ až pokiaľ nepokrýva iba pozitívne príklady z triedy c_j. Po nájdení "najlepšieho" výrazu algoritmus pridá do zoznamu pravidiel pre triedu c_j nové pravidlo a odstráni z trénovacej množiny všetky pokryté pozitívne príklady. Potom pokračuje v učení ďalších pravidiel na zostávajúcich trénovacích dátach. Pri generovaní výrazu pravidla sa v každom kroku otestujú všetky možnosti a vyberie sa špecifikácia, ktorá minimalizuje variabilitu tried podľa zvolenej hodnotiacej funkcie, podobne ako pre rozhodovacie stromy napr. podľa entropie [Quinlan 1996].

Aby sa zabránilo preučeniu, pri indukcii pravidiel sa používa rovnaká stratégia orezávania ako pri generovaní rozhodovacích stromov. Základný algoritmus REP pre orezávanie pravidiel je nasledujúci:

```
Algoritmus:


Náhodne rozdeľ trénovacie príklady na trénovaciu
a validačnú množinu (napr. 2/3 pre trénovaciu množinu).

Nauč zoznam pravidiel, ktorý perfektne klasifikuje
trénovaciu množinu.

Zjednoduš vygenerovaný zoznam použitím orezávacích operácií
(odstránenie pravidla zo zoznamu, alebo vynechanie
podmienok z výrazu pravidla). Zmena sa príjme ak operácia
nezvýši chybu celého zoznamu na validačnej množine.

Ak sa neprijme zmena žiadnej operácie, vráť výsledný zoznam
pravidiel.
```

Na rozdiel od rozhodovacích stromoch, kde orezanie vetvy nemá žiaden vplyv na susedné vetvy stromu, pri metóde "zhora nadol" ovplyvní orezanie pravidla generovanie všetkých nasledujúcich pravidiel.

Pri orezávaní sa pravidlo zovšeobecňuje, čo znamená že bude pokrývať viac pozitívnych príkladov spolu s niekoľkými negatívnymi príkladmi, ktoré boli identifikované ako špecifické pre danú trénovaciu množinu. Tieto príklady by mali byť odstránené z trénovacej množiny aby neovplyvnili generovanie nasledujúcich pravidiel. Nevýhodou je, že počiatočná fáza algoritmu REP (krok 2) nevie rozhodnúť ako budú pravidlá výsledne orezané (a teda nevie, ktoré príklady by mali byť navyše odstránené z trénovacej množiny).

Ďalšou nevýhodou algoritmu REP sú problémy spôsobené rozdelením príkladov pri nedostatku trénovacích dát. Po rozdelení sa nenaučia (alebo sa naučia nesprávne) pravidlá, ktorých príklady sa nevyskytujú v trénovacej množine. Naopak po naučení musia mať pravidlá dostatočný počet pokrytých príkladov vo validačnej množine, pretože inak by sa mohli orezať aj správne naučené pravidlá.

3.3.2.5 Boosting

Boosting je všeobecná metóda určená na zlepšenie klasifikácie daného algoritmu [Shapire a kol. 1998]. Pre prípad klasifikácie do dvoch tried vytvorí daný algoritmus na základe trénovacej množiny dokumentov K klasifikátor $H: K \rightarrow \{-1, 1\}$. Algoritmus boostingu vytvára modifikáciou trénovacej množiny dát postupnosť klasifikátorov $H_m: K \rightarrow \{-1, 1\}$, $m = 1, ..., M$, ktoré potom kombinuje do výsledného klasifikátora. Predikcia výsledného klasifikátora je daná váženou kombináciou predikcií jednotlivých základných klasifikátorov:

$$H(d_i) = sign\left(\sum_{m=1}^{M} \alpha_m H_m(d_i)\right) \tag{3.19}$$

Parametre α_m, $m = 1, ..., M$ sú určené algoritmom boostingu tak, aby klasifikátory s lepšou presnosťou ovplyvnili výslednú predikciu viac. Presnosť základných klasifikátorov H_m môže byť len o niečo vyššia než presnosť náhodnej klasifikácie. Klasifikátory H_m sú preto označené ako tzv. slabé (weak) klasifikátory.

Trénovacia množina dát sa modifikuje pomocou distribúcie váh priradených jednotlivým dokumentom $d_i \in K$. Pri učení prvého klasifikátora postupnosti sú váhy pre každý príklad nastavené uniformne. Pre každú nasledujúcu iteráciu $m = 2, ..., M$ sa zvýšia váhy príkladov, ktoré boli nesprávne klasifikované predchádzajúcim klasifikátorom $H_m - 1$, a váhy správne klasifikovaných príkladov sa znížia. Takto sa učenie nasledujúceho klasifikátora postupne sústreďuje na príklady nesprávne klasifikované predchádzajúcimi klasifikátormi.

Upravená verzia algoritmu využíva spojitú predikciu klasifikátora $H: K \rightarrow R$ [Shapire, Singer 1999]. H klasifikuje dokumenty podľa rozhodovacej funkcie $\text{sign}[H(d_i)]$, pričom $|H(d_i)|$ zodpovedá spoľahlivosti predikcie. Riadiaca procedúra algoritmu je nasledovná:

Algoritmus:

Inicializuj distribúciu váh $w_1(i) = 1 / |K|, i = 1, ..., |K|$
pre $m = 1, ..., M$
Volaním základného algoritmu nauč klasifikátor $H_m: K \rightarrow R$
podľa aktuálnej distribúcie váh $w_m(i)$.
Urči parameter α_m.
Zmeň distribúciu váh podľa pravidla:

$$w_{m+1(i)} = \frac{w_{m(i)} \exp(-\alpha_m y_i H_m(d_i))}{Z_m}$$

, vkde Z_m je normalizačná konštanta určená tak, aby $\sum_{i=1}^{|D|} w_{m+1(i)} = 1$.

Výstupom je rozhodovacia funkcia výsledného klasifikátora:

$$H(d_i) = sign\left(\sum_{m=1}^{M} \alpha_m H_m(d_i)\right) \tag{3.20}$$

Ako základný algoritmus pre boosting sa najčastejšie používajú rozhodovacie stromy ale je možné použiť aj ďalšie algoritmy ako napr. naivný bayesov klasifikátor alebo neurónové siete. Pre kategorizáciu textov sa najčastejšie používajú veľmi jednoduché klasifikátory, ktoré klasifikujú príklady do dvoch tried na základe výskytu jedného termu.

4 Zhlukovanie textových dokumentov

4.1 Základné pojmy

Zhlukovanie predstavuje problém nekontrolovaného učenia, pri ktorom sú objekty zaraďované do skupín, ktoré nazývame zhluky. V úlohách kategorizácie máme k dispozícii kolekciu vopred zatriedených trénovacích príkladov, pričom úlohou je vybudovať model, ktorý na základe takýchto popisov bude schopný kategorizovať aj nové, nezatriedené objekty. V prípade zhlukovania máme za úlohu vytvoriť zhluky navzájom podobných objektov zo vstupnej kolekcie bez vopred známej informácie o príslušnosti k nejakým triedam.

Zhlukovanie je rozdelenie vstupných dát do skupín navzájom si podobných objektov. Každá skupina (zhluk) teda obsahuje objekty čo najpodobnejšie ostatným v skupine a čo najodlišnejšie od objektov v ostatných skupinách. Z pohľadu strojového učenia zhluky odpovedajú skrytým vzorom v dátach, hľadanie zhlukov je nekontrolované učenie a výsledkom je určitý dátový koncept. Podrobný prehľad zhlukovacích metód je možné nájsť v [Berkhin 2002] a [Jain a kol. 1999].

Vzhľadom k charakteru popisovaných zhlukovacích algoritmov sa v rámci tejto kapitoly budeme držať nasledovného popisu. Nech model skúmanej množiny textových dokumentov pozostávajúcej z *N* objektov (dokumentov) je založený na vektorovej reprezentácii, počet termov (slov) použitých ako atribúty (príznaky) je *M*. Dokument d_i je teda popísaný jeho charakteristickým vektorom $\bar{d}_i = (w_{i1}, w_{i2}, ..., w_{iM})$, kde kde w_{it} je váha, resp. dôležitosť termu z indexom t (t=1, ..., M) v dokumente d_i, i=1, ..., N (podrobnejšie o vektorovej reprezentácii, viď. časť 2.7.3). Dátová množina vstupujúca do procesu zhlukovania preto zodpovedá objekt-atribútovej matici veľkosti $N \times M$. Základným cieľom zhlukovania je potom priradiť jednotlivé objekty (dokumenty) do konečného počtu *k* podmnožín – zhlukov. V závislosti od typu algoritmu je *k* konštantou alebo parametrom výsledného modelu. Množiny zhlukov sú najčastejšie (opäť v závislosti od typu algoritmu) disjunktné a ich zjednotenie dáva celú vstupnú množinu dokumentov.

Zhlukovanie textov je plne automatický proces, ktorý rozdelí súbor dokumentov do skupín. Dokumenty v každej skupine si sú v určitom (zvolenom) smere podobné. Pokiaľ je pre rozlíšenie použitý obsah dokumentu, potom rôzne skupiny korešpondujú s rôznymi okruhmi a témami obsiahnutými v tomto súbore dokumentov. Na zhlukovanie môžeme preto nazerať takisto ako na spôsob, ako zistiť čo daný súbor dokumentov obsahuje. K identifikácii okruhov/tém používajú nástroje na zhlukovanie slová, ktoré sú typické v dokumentoch danej skupiny.

Pre vektorovú reprezentáciu dokumentov je charakteristický veľký počet dimenzií, čo spôsobuje neefektívnosť pri učení a obmedzuje použitie niektorých metód. Navyše medzi termami zahrnutými do vektorovej reprezentácie je veľa irelevantných termov, ktoré predstavujú šum pri učení a nepriaznivo ovplyvňujú interpretáciu a využitie výsledkov, preto sa často pristupuje k redukcii príznakového priestoru metódami popísanými vyššie,

v časti 2.8.

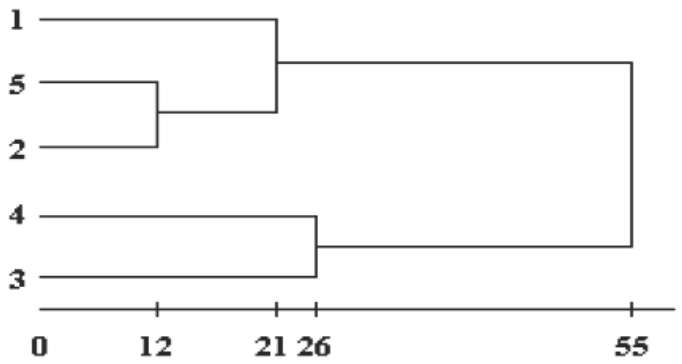
4.2 Prehľad metód pre zhlukovanie textov

Kategorizácia zhlukovacích algoritmov nie je jednoznačná, niektoré skupiny sa prekrývajú. Jedno z možných rozdelení je nasledujúce:

- hierarchické metódy
- aglomeratívne algoritmy
- divízne algoritmy
- rozdeľujúce metódy
- k-stredové metódy
- pravdepodobnostné zhlukovanie
- algoritmy založené na hustote
- metódy založené na mriežke
- metódy založené na riešení ohraničení
- biologicky inšpirované metódy
- algoritmy využívajúce neurónové siete (napr. samoorganizujúce sa mapy)
- evolučné metódy

4.2.1 Hierarchické zhlukovanie

Hierarchické metódy budujú hierarchiu zhlukov – tzv. *dendrogram* (príklad dendogramu viď. Obr. 4.1). Ide o stromový diagram, ktorý znázorňuje postupné zhlukovanie jednak jednotlivých objektov, jednak zhlukov vytvorených v predchádzajúcich krokoch. Obr. 4.1 predstavuje horizontálnu podobu dendrogramu, pri ktorej sú objekty uvedené na ose *Y*. Čím sú si dva objekty podobnejšie, tým skôr dochádza k ich spojeniu, ktoré vyjadruje vzdialenosť hladiny ich spojenia, ktorá je uvedená na ose *X*.



Obr. 4.1 Príklad zobrazenia hierarchie zhlukov v podobe dendrogramu.

Aglomeratívne zhlukovanie začína s množinou práve N zhlukov (t.j. každý objekt predstavuje samostatný zhluk) a v každom ďalšom kroku ďalej rekurzívne spája dokopy vždy dva alebo viacero najpodobnejších zhlukov. Pri aglomeratívnom zhlukovaní sa zmenšuje počet zhlukov v každej iterácii vždy o 1, a to aj vtedy, ak na rovnakej zhlukovacej hladine malo viac dvojíc zhlukov rovnaký koeficient podobnosti. V takom prípade sa zlúči len jedna dvojica. Na odstránenie tohto nedostatku slúži tzv. definitná hierarchická aglomeratívna procedúra, ktorá v prípade rovnakého koeficientu podobnosti viacerých dvojíc zhlukov zjednotí všetky takéto dvojice do súvislých zhlukov. Proces končí, ak je splnená ukončovacia podmienka, teda ak sa vytvorí rozklad $(N-1)$ taký, že všetky objekty sú zahrnuté v jednom zhluku.

Divízne metódy vytvárajú hierarchický systém zhlukov postupným delením existujúcich zhlukov. Začína sa jedným zhlukom, ktorý obsahuje celú množinu vstupných objektov. Tento zhluk sa rekurzívne delí dovtedy, pokiaľ nie sú všetky zhluky jednoprvkové.

Hierarchické zhlukovanie má flexibilnú úroveň zjemnenia pohľadu na hierarchiu, čo je jeho výhodou. Medzi jeho ďalšie výhody patrí jednoduchosť aplikácie rôznych foriem vzdialenostných metrík alebo podobnostných funkcií ako aj aplikovateľnosť na rôzne typy atribútov. Nevýhodou je vágnosť ukončovacieho kritéria, ako aj fakt, že väčšina hierarchických algoritmov neprehľadáva už skonštruované zhluky (za účelom ich zlepšenia).

Základným prostriedkom pre spájanie alebo rozklad zhlukov je analýza vzdialeností medzi aktuálnymi zhlukmi. Táto spočíva vo vytvorení matice nepodobnosti (vzdialeností), v ktorej sú aktuálne zhluky párovo porovnané vzhľadom k nami definovanej spojovacej (linkovacej) metrike. Znamená to, že miera nepodobnosti (zvyčajne vzdialenosť) zhlukov sa počíta pre každý pár objektov (pričom prvý je z jedného a druhý z iného zhluku). Typ tejto metriky má vplyv na to, ktoré zhluky sa v ďalšom kroku spoja (rozdelia). V zásade existujú dva hlavné typy spoiovacích metrík – medzizhlukové a geometrické. V prvom prípade sú najčastejšie používané:

- *Spriemerňujúca metóda* – vzdialenosť medzi zhlukmi je počítaná ako aritmetický priemer vzdialeností všetkých objektov zhlukov (pričom berieme iba páry, kde prvý objekt patrí prvému zhluku, a druhý objekt druhému zhluku)
- *Metóda najbližšieho suseda* – vzdialenosť zhlukov je daná minimálnou nájdenou vzdialenosťou objektov medzi oboma zhlukmi
- *Metóda najvzdialenejšieho suseda* – vzdialenosť zhlukov je daná maximálnou nájdenou vzdialenosťou objektov medzi oboma zhlukmi

Medzi geometrické spojovacie prístupy patria napríklad:

- *Centroidná metóda* – vzdialenosť medzi zhlukmi je počítaná ako euklidovská vzdialenosť medzi ich centroidmi (vektory aritmetických priemerov hodnôt atribútov všetkých objektov zhluku)
- *Mediánová metóda* – vzdialenosť zhlukov je počítaná podobne ako v prípade centroidov, avšak berieme do úvahy vzdialenosť medzi mediánmi zhlukov

- *Minimálne-variančná metóda* – vzdialenosť zhlukov je daná súčtom štvorcov odchýlok hodnôt atribútov objektov v zhluku voči ich zhlukovému priemeru, pričom cieľom je získať minimálnu hodnotu tejto variancie.

4.2.2 *K*–stredové metódy zhlukovania

Algoritmy K–means a K–medoids rozdeľujú dáta do podpriestorov. Kvôli výpočtovej náročnosti pri použití všetkých možných podpriestorov pracujú mnohé z nich v režime iteratívnej optimalizácie. Tým sa myslia rôzne relokačné schémy, ktoré iteratívne prideľujú body do k zhlukov. Na rozdiel od hierarchických algoritmov, relokačné algoritmy zhluky postupne vylepšujú. Pri použití vhodných dát je tak možné zhotoviť vysoko kvalitné zhluky. Iteratívne optimalizačné rozdeľovacie algoritmy delíme na k–medoids a k–means metódy na základe toho, ako sú reprezentanti množín zhotovení.

K–medoids používa pre reprezentáciu zhluku najvhodnejší dátový bod (podľa zvoleného kritéria) v príslušnom zhluku. Pri takejto reprezentácii je výhodou, že nezávisí na type atribútu a výber medoidov prebieha na základe umiestnenia prevládajúceho bodu v zhluku. Po určení medoidov sa zhluky definujú ako podpriestory bodov (body sú blízke k príslušným medoidom) a objektívna funkcia je definovaná ako priemerná vzdialenosť alebo miera nepodobnosti medzi bodom a medoidom zhluku.

K–means je vo všeobecnosti zrejme najpopulárnejší zhlukovací nástroj, ktorý sa používa vo vedeckých aj priemyselných aplikáciách. Rovnako ako v predchádzajúcom prípade aj v tomto prístupe pôjde o zatriedenie vstupných objektov do zhlukov, ktorých počet je vopred známy. Každý zo zhlukov je v tomto prípade reprezentovaný centroidom - priemer bodov daného zhluku. V počiatočnom kroku sú náhodne vybrané centroidy, ktoré budú reprezentovať jednotlivé zhluky. Vo všeobecnosti sa volí radšej voľba centroidov tak, aby boli čo najvzdialenejšie od ostatných centroidov. V nasledujúcom kroku ku všetkým objektom zo vstupnej množiny potom algoritmus priradí k nim najbližší centroid. Po priradení všetkých objektov k jednotlivým centroidom (zhlukom) algoritmus pre získané zhluky vypočíta nové priemerné hodnoty – nové centroidy. Tento krok opakujeme dovtedy, až kým sa už ďalej centroidy nemenia. Inými slovami ide teda o minimalizáciu účelovej funkcie, ktorá vyjadruje sumu rozdielov medzi bodmi a centroidom, vyjadrenú pomocou príslušnej mierky. Napríklad L_2 norma, ktorá je základom účelovej funkcie (suma štvorcov chýb medzi bodmi a prislúchajúcimi centroidmi, t.j. štvorec euklidovskej vzdialenosti) je rovná celkovému vnútro-zhlukovému rozdielu:

$$E(C) = \sum_{j=1:k} \sum_{x_i \in C_j} \left\| x_i - c_j \right\|^2 \qquad (4.1)$$

```
Algoritmus:

Zvoliť k objektov v priestore, ktoré reprezentujú prvotné
centroidy.
Priraď každému objektu z vstupnej množiny centroid.
Ak sú už všetky objekty priradené, prepočítaj pozíciu
centroidov.
Opakuj kroky 2 a 3 až pokiaľ sa pozícia centroidov ďalej
nemení.
```

Medzi výhody algoritmu k–means patrí jeho jednoduchosť, priamosť a to, že je založený na pevných základoch analýzy premenných. Algoritmus k-means má taktiež svoje nevýhody, medzi ktoré patrí:

- výsledok silne závisí od inicializačných odhadov centroidov,
- vypočítané lokálne optimum je (zvyčajne) vzdialené od globálneho,
- nejasné určenie správneho počtu zhlukov k,
- proces je citlivý na „outliers“ (body úplne mimo všetkých centier, často šum),
- algoritmus má problémy so škálovateľnosťou,
- sú použiteľné iba numerické atribúty,
- výsledné zhluky môžu byť nevyvážené (napríklad prázdne).

Metóda k–medoids je príbuznou metódou k metóde k–means. Rozdiel medzi týmito dvomi prístupmi spočíva v tom, že vzdialenosť medzi objektmi a zhlukmi nie je určovaná vzhľadom na priemernú hodnotu zhluku (centroid), ale vzhľadom na hodnotu objektu, ktorý je najbližšie k priemernej hodnote (medián, medoid). Algoritmus k–medoids je potom analogický s algoritmom k–means. Na začiatku sa podobne ako pri k–means inicializuje k náhodných objektov (k medoidov), ktoré reprezentujú k zhlukov. Medoid je konkrétny objekt zhluku, pre ktorý je súčet vzdialeností ostatných objektov v rámci zhluku minimálny. Ostatné objekty zo vstupnej množiny priradzujeme k jednotlivým zhlukom na základe najmenších vzdialeností. Ak je daný objekt najbližšie k medoidu v rámci svojho zhluku, v tomto zhluku ho ponecháme, ak nie, objekt bude zaradený do toho zhluku, ku ktorého medoidu má najbližšie. V ďalšom kroku pre takto získané zhluky algoritmus priradí nové medoidy. Celý proces iteruje dovtedy, kým klesá hodnota funkcie:

$$f = \sum_{i=1}^{n} D(x_i, m_{g,i}) \tag{4.2}$$

kde m je medoid v zhluku g, ku ktorému je priradený i–tý objekt.

```
Algoritmus:
Náhodne vyber k z n vstupných objektov - medoidy
Asociuj každý objekt zo vstupnej dátovej množiny ku
najbližšiemu medoidu. ("najbližší" podľa definovanej
metriky)
Pre každý medoid m
Pre každý iný objekt o
Vymeň  m za o a vypočítaj sumu vzdialeností konfigurácie
Zvoľ konfiguráciu s minimálnou hodnotou
Opakuj kroky 2 až 5 až pokiaľ sa medoidy nemenia.
```

Výhoda oproti k–means spočíva v menšej citlivosti tejto metódy na rušivé a nevhodné údaje.

Rozšírenia k–means a k–medoids metód spočívajú najmä v zlepšení ich škálovateľnosti, zmenšení citlivosti na problémy s vysokou dimenzionalitou a použitie rôznych prístupov k reprezentácii zhlukov.

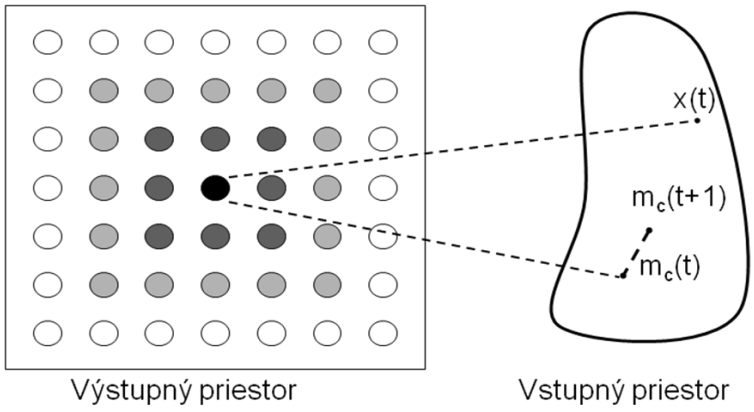
Pri použití algoritmu Kmeans je potrebné zistiť, pre akú hodnotu k je výsledok zhlukovania najlepší. Jednou z metód ako je možné takéto vyhodnocovanie vykonať je pomocou porovnávania kvadratických odchýlok od centier zhlukov, t.j. porovnávaním výpočtov súčtu druhých mocnín rozdielov medzi vektorom centra a vektorom dokumentu, cez všetky centrá, dokumenty a atribúty. Víťazným modelom je model s najnižšou hodnotou takejto kvadratickej odchýlky.

4.2.3 Samoorganizujúce sa mapy (SOM)

Samoorganizujúce sa mapy (Self-Organizing Maps) [Kohonen 1995] sú jedným zo základných modelov umelých neurónových sietí založených na princípe nekontrolovaného učenia, umožňujúci v rámci vizualizácie dát mapovať zvyčajne vysoko–rozmerný príznakový priestor do usporiadaného nízkorozmerného priestoru – mapy. Sieť pozostáva z určitého počtu neurónov. Každému z neurónov je priradený n rozmerný váhový vektor m_i, pričom vektor váh má ten istý rozmer ako vstupné vzorky x.

Špecifickou črtou SOM je realizácia zobrazenia zachovávajúceho topológiu. Za týmto účelom sú neuróny zoradené v pravidelnej štruktúre (mriežke). Takéto usporiadanie predstavuje výstupný priestor, v ktorom vzdialenosť dvoch neurónov je obyčajne euklidovskou vzdialenosťou vektorov ich súradníc v uvažovanej štruktúre. Zobrazenie ktoré vznikne po natrénovaní SOM má tú vlastnosť, že ľubovoľné dva vzory blízke vo vstupnom priestore evokujú v sieti odozvy na neurónoch, ktoré sú si tiež fyzicky blízke.

Proces trénovania tzv. Kohonenovej mapy (SOM) sa dá popísať ako adaptácia váh neurónov na základe vstupnej vzorky. Každá iterácia učenia mapy začína náhodným výberom jednej vstupnej vzorky $x(t)$. Táto vzorka tvorí vstup pre SOM a pre každý neurón i sa vypočíta jeho aktivácia $m_i(t)$. Najčastejšie sa na výpočet aktivácie neurónu používa Euklidovská vzdialenosť medzi váhovým vektorom a vstupným vektorom. Neurón s najnižšou hodnotou aktivácie (teda najbližší ku vzorke) sa označí ako víťaz c, t.j. pre jeho váhový vektor platí:



Obr. 4.2 Znázornenie procesu trénovania mapy SOM

$$m_c(t) = \min_i \left\| x(t) - m_i(t) \right\|. \tag{4.3}$$

Napokon, vektor váh víťaza, ako aj váhové vektory neurónov v blízkosti víťaza sú adaptované. Táto adaptácia je realizovaná ako postupné znižovanie rozdielu jednotlivých zložiek vstupného vektora a váhového vektora neurónu, teda:

$$m_i(t+1) = m_i(t) + \alpha(t) \cdot h_{ci}(t) \cdot \left[x(t) - m_i(t) \right]. \tag{4.4}$$

Z geometrického hľadiska sa váhový vektor víťazného neurónu posunie smerom k vstupnému vektoru. Hodnota, o ktorú sa daná váha posunie, je daná koeficientom učenia α, ktorý sa po každej iterácii zmenšuje. Počet vektorov ktoré sú adaptované je určený funkciou susednosti h_{ci} (viď. Obr. 4.2, kde je táto funkcia znázornená odtieňmi sivej farby). Počet neurónov, ktoré sa adaptujú na základe funkcie susednosti sa časom taktiež znižuje. Tento posun váhového vektora má za následok, že Euklidovská vzdialenosť medzi týmito vektormi klesá a teda váhový vektor neurónu sa čoraz viac približuje ku vstupnému vektoru. Tento neurón má potom väčšiu pravdepodobnosť vyhrať (byť víťaz) pri ďalšom výskyte tohto vstupného vektora na vstupe. Dôsledok adaptácie (zmeny) nielen víťaza, ale aj určitého počtu neurónov v okolí víťaza, vedie k priestorovému zhlukovaniu podobných vstupných vzoriek do susedných častí samoorganizujúcej sa mapy – k topológiu zachovávajúcemu mapovaniu.

Ako výsledok procesu učenia sú podobné vstupné dáta namapované na susedné regióny (neuróny) na mape. V prípade textových dokumentov budú teda dokumenty (popísané príznakovým vektorom), rozmiestnené topograficky podľa svojej podobnosti v dvojrozmernej mape (čím podobnejšie vektory dokumentov, tým bližšie budú umiestnené na mape neurónov).

Ku klasickému algoritmu Kohonenovej mapy vzniklo pochopiteľne viacero rozšírení, najmä z dôrazom na riešenie problémov, ktoré sa môžu pri klasickej SOM prejaviť, ako napr.:

- štruktúra, počet neurónov a dimenzia mapy musia byť dopredu pevne dané,

- možnosť vzniku „mŕtvych“ neurónov – sú to v podstate centrá zhlukov s nulovou pravdepodobnosťou výskytu vstupných vektorov,
- diskrétnosť a „rovnomernosť“ projekcie – konkrétny vstupný vektor sa premietne na jediný víťazný neurón, ktorého súradnice môžu byť len diskrétne hodnoty.

Jedným z rozšírení tradičnej SOM je Growing Grid (Growing SOM) [Fritzke 1995]. Ide o inkrementálne zväčšovanie mapy, ktoré umožňuje dynamicky rozširovať mapu podľa potrieb vstupného príznakového priestoru (vstupných dát). Je to teda prípad, kedy je možné pridávať neuróny, avšak musí byť splnená podmienka, že štruktúra usporiadania ostáva obdĺžniková – v tvare pravidelnej mriežky. Z toho vyplýva nutnosť pridávať nie jeden neurón, ale celé „pásy“ neurónov v dvojrozmernej štvorcovej mriežke, t.j. riadky resp. stĺpce neurónov.

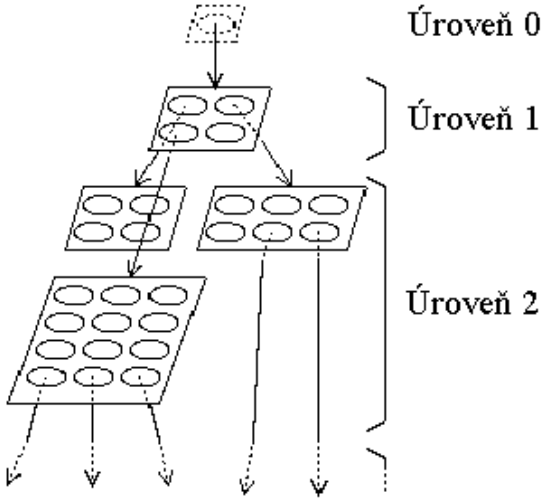
Ďalším typom rozšírenia SOM je zavedenie štruktúry máp namiesto jedinej mapy. Hlavnou myšlienkou hierarchickej mapy (HFM z anglického hierarchical feature map) je použitie hierarchického prístupu viacerých úrovní máp, kde každá úroveň pozostáva z určitého počtu nezávislých máp typu SOM. Jedna mapa je použitá na prvej úrovni hierarchie. Pre každý neurón na tejto mape je pridaný SOM na ďalšej úrovni hierarchie. Tento princíp sa opakuje do ľubovoľnej hĺbky hierarchie. Proces učenia takejto hierarchie začína učením mapy na najvrchnejšej úrovni. Táto mapa sa učí štandardným učiacim procesom. Keď je už prvá mapa stabilná, proces učenia začína na mapách druhej úrovne (vrstvy). Tu sa každá mapa preúča iba s tými príkladmi (vstupnými vektormi), ktoré boli priradené danému neurónu na mape vyššej úrovne.

Pre popis, spracovanie a organizáciu textových dokumentov sa tieto dva algoritmy hodia veľmi dobre, ale ich pozitívne vlastnosti sa ešte umocňujú v kombinovanom prístupe založenom na oboch týchto rozšíreniach SOM. Ich kombináciou dostávame algoritmus GHSOM - Growing Hierarchical SOM [Dittenbach a kol. 2000]. To vedie k vytvoreniu nového typu štruktúry, ktorá sa rozširuje oboma smermi. Najprv sa mapa rozširuje horizontálne (ako Growing Grid), po ukončení tohto procesu sa vybrané neuróny rozširujú vo vertikálnom smere, t.j. na novú úroveň hierarchie (ako HFM). Podrobnejšie je tento algoritmus popísaný v nasledujúcej časti 4.2.3.1.

4.2.3.1 Algoritmus GHSOM

Algoritmus GHSOM (Growing Hierarchical Self Organizing Map) [Dittenbach a kol. 2000] umožňuje dynamické rozširovanie máp (viď. Obr. 4.3), a to:

- *hierarchicky* – podľa distribúcie dát, čo umožňuje hierarchickú dekompozíciu a navigáciu v podmapách a
- *horizontálne* – veľkosť mapy sa mení tak, aby sa prispôsobila požiadavkám vstupného priestoru.



Obr. 4.3 Znázornenie procesu tvorby hierarchickej samoorganizujúcej sa mapy (GHSOM)

Algoritmus GHSOM má nasledovné kroky:

1. Najskôr sa vyráta celková odchýlka vektora vstupných dát na vrstve nultej úrovne. Tejto jednotke sa priradí váhový vektor $m_0 = [\eta_{01}, \eta_{02}, \ldots \eta_{0n}]^T$, ktorého zložky sú priemerom zložiek všetkých vstupných dát (vektorov). Stredná kvadratická chyba nultej vrstvy sa vyráta podľa vzťahu: $mqe_0 = \frac{1}{d} \cdot \|m_0 - x\|$, kde d je počet vstupných dát x.
2. Trénovanie GHSOM začína vrstvou 1. úrovne. Táto mapa je zvyčajne relatívne malá. Každému neurónu *i* sa priradí *n*-rozmerný váhový vektor $m_i = [\eta_{i1}, \eta_{i2}, \ldots \eta_{in}]^T, m_i \in \Re^n$, ktorý je inicializovaný náhodnými hodnotami. Váhový vektor má rovnaký rozmer ako vstupné vektory.
3. Učenie samoorganizujúcej sa mapy prebieha presne tak, ako u klasického SOM (popísané v časti 4.2.3). Ide vlastne o súperenie medzi neurónmi o to, ktorý najlepšie reprezentuje vstupný vektor (konkurenčné učenie). Neurón s váhovým vektorom, ktorý je najbližšie prezentovanému vektoru na vstupe vyhráva. Váhový vektor víťaza ako aj neuróny v jeho blízkosti sa adaptujú tak, aby sa čo najviac podobali na vstupný vektor. Miera adaptácie je riadená učiacim parametrom α, ktorý v čase klesá. Počet susedných neurónov, ktoré sa adaptujú spolu s víťazom tiež klesá v čase (na začiatku učenia sa adaptuje veľa susedných neurónov a na konci už iba víťaz). To, ktoré neuróny sa adaptujú určuje funkcia susednosti *hci*, ktorá je založená na vzdialenosti neurónov k víťazovi meranej v dvojrozmernej mriežke vytvorenej neurónovou sieťou. Kombináciou týchto princípov vzniká učiace pravidlo:
   $m_i(t+1) = m_i(t) + \alpha(t) \cdot h_{ci}(t) \cdot [x(t) - m_i(t)]$, kde x je aktuálna vstupná vzorka a c je víťaz v iterácii t.
4. Po určitom počte iterácií (daný parametrom λ) sa vyráta

stredná kvadratická chyba mapy podľa vzťahu: $MQE_m = \frac{1}{u} \cdot \sum_i mqe_i$ kde u je počet neurónov i v mape m, mqe_i je stredná kvadratická chyba neurónu i v mape m, t.j $mqe_i = \frac{1}{d} \cdot \|m_i - x\|$, kde d je počet vektorov x namapovaných na i-ty neurón. Každá vrstva GHSOM je zodpovedná za vysvetlenie určitej časti odchýlky vstupných dát z predchádzajúcej vrstvy. To sa dosiahne pridávaním neurónov do mapy na každej vrstve až pokiaľ sa nedosiahne vhodná veľkosť mapy. Mapy na každej úrovni rastú pokiaľ odchýlka neurónu predchádzajúcej vrstvy nie je zredukovaná aspoň na τ_m percent. Teda čím menší parameter τ_m, tým väčšia bude výsledná mapa. Pokiaľ platí pre vrstvu m podmienka: $MQE_m \geq \tau_m \cdot mqe_0$, pridáva sa k mape nový riadok alebo nový stĺpec neurónov. Vloží sa tak, aby susedil s chybovým neurónom e (neurón, ktorý má najväčšiu chybu). To, či sa pridá nový riadok alebo nový stĺpec závisí od toho, kde sa nachádza neurón, ktorý je najmenej podobný s chybovým neurónom. Miera podobnosti sa určuje vo vstupnom priestore. Hodnota vektorov váh nových neurónov sa nastaví na priemer hodnôt váh ich susedov. Po vložení sa učiaci parameter α a funkcia susednosti h_{ci} nastaví na počiatočnú hodnotu a učiaci proces pokračuje ďalej.

5. Akonáhle skončí učenie mapy prvej úrovne, teda: $MQE_m < \tau_m \cdot mqe_0$ zisťuje sa, ktoré neuróny je potrebné expandovať do ďalšej mapy. Sú to neuróny, ktoré majú veľkú kvadratickú chybu v porovnaní s kvadratickou chybou mapy nultej úrovne (mqe_0). Na určenie jemnosti (granularity) rozlíšenia vstupných dát v koncových mapách sa pritom používa parameter τ_u. Každý neurón i, vyhovujúci podmienke $mqe_i > \tau_u \cdot mqe_0$, bude následne expandovaný.
6. Učiaci proces pokračuje ďalej pre každú novovytvorenú mapu tak isto, ako v predchádzajúcich krokoch. Jediným rozdielom je, že do mapy vstupuje už iba časť vstupných dát prislúchajúci danému neurónu nadradenej mapy a chyba mqe_0, ktorá určuje rast mapy je nahradená chybou nadradeného neurónu (teda podmapa sa snaží minimalizovať chybu nadradeného neurónu).
7. Učiaci proces GHSOM končí, keď už žiadny neurón nie je potrebné expandovať, resp. sa dosiahne definovaná maximálna hĺbka hierarchie. Parameter τ_u určuje celkovú úroveň detailu (jemnosti) rozlíšenia vstupných dát. Čím menší je tento parameter, tým viac hierarchických úrovní sa vytvorí. Parameter τ_m určuje úroveň detailu rozlíšenia vstupných dát na konkrétnej mape. Čím je parameter τ_m menší, tým väčšie mapy vzniknú na jednotlivých úrovniach (a hierarchia bude „plytšia").

4.2.3.2 Popis zhlukov pomocou algoritmu LabelSOM

Algoritmus LabelSOM je prístup patriaci ku (všeobecne) centroidným modelom snažiaci sa určiť tie zložky centroidného vektora, ktoré sú najdôležitejšie pri mapovaní vstupných vektorov na dané centrum. To sa deje určením veľkosti prispenia každej zložky vektora voči celkovej Euklidovskej vzdialenosti medzi vstupným vektorom a váhovým vektorom centra.

Metóda LabelSOM [Rauber 1999] je založená na pozorovaní, že po naučení SOM-u sa zložky vektora váh daného neurónu čo najviac podobajú odpovedajúcim zložkám vstupných vektorov, ktoré boli namapované na tento jeden neurón, ako aj na niektoré zložky vektorov susediacich neurónov. Zložky vektora, ktoré majú približne tú istú hodnotu v rámci celej množiny vstupných vektorov namapovaných na konkrétny neurón, popisujú a pomenúvajú vlastne príznaky všetkých vstupných dát tohto neurónu. Preto môžeme tieto zložky vektora váh považovať za popis daného neurónu.

Kvadratická odchýlka pre všetky jednotlivé príznaky slúži ako miera pre ich relevantnosť ako názvy tried. Vyberú sa tie zložky vektora, ktorých hodnota chyby je blízka nule. Kvadratická odchýlka vektora sa vypočíta pre každý neurón i ako priemerná vzdialenosť medzi zložkami vektorov všetkých vstupných dát namapovaných na neurón i a zložkami vektora váh tohto neurónu. Aplikácia tohto prístupu v našom prípade je jednoduchá, príklady určitého konceptu ("neurónu") sú priamo jemu prislúchajúce objekty a váhový vektor konceptu (centroidu, "neurónu") je aritmetický priemer váhových vektorov daných objektov konceptu. Nech C_i je množina objektov x_j neurónu i. Vypočítaním vzdialeností pre všetky zložky vektora k cez všetky vektory x_j patriacich do C_i dostaneme kvadratickú chybu vektora q_i pre všetky koncepty i:

$$q_{ik} = \sum_{x_j \in C_i} \sqrt{\left(m_{ik} - x_{jk}\right)^2}, k = 1, ..., n \qquad (4.5)$$

Vybraním tých zložiek vektora, ktorých hodnota chyby je blízka nule sa získa zoznam atribútov, ktoré sa nachádzajú vo veľkom počte vstupných vektorov (dokumentov) namapovaných na tento neurón, a preto popisujú charakteristiky dát pre daný zhluk. Tieto atribúty slúžia ako názvy resp. popisy (*labels*) jednotlivých zhlukov (konceptov).

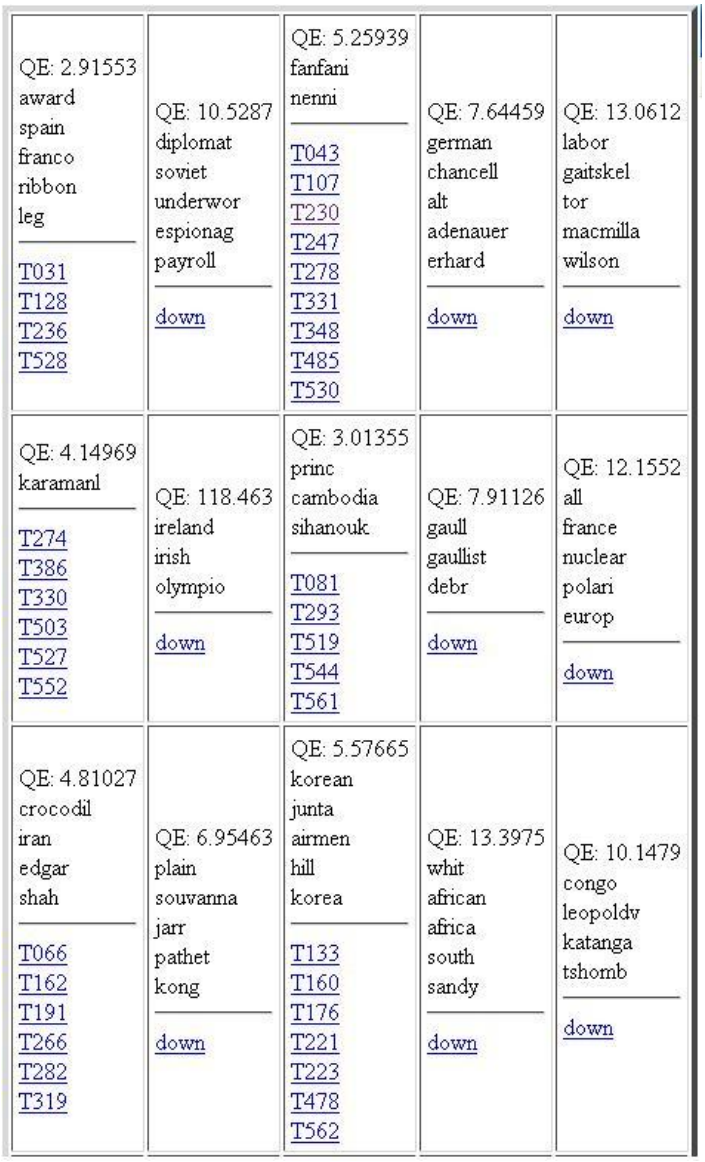
Tento postup však pri textových dokumentoch naráža na problémy. Kvôli veľkej dimenzionalite príznakového priestoru a charakteristike reprezentácie príznakových vektorov dokumentov sa zvyčajne vyskytuje veľký počet zložiek vstupného vektora, ktorých hodnota je nulová (existuje veľa termov - slov, ktoré nie sú obsiahnuté v danej skupine dokumentov). Tieto termy majú obvykle hodnotu kvadratickej chyby rovnú nule a preto by boli vybrané ako charakteristické termy. Dôsledkom by bolo, že neurón by bol prakticky popísaný len termami, ktoré sa v daných dokumentoch ani nevyskytujú (samozrejme vzhľadom na daný zhluk). Úlohou analýzy textov je však popísať zhluky pomocou príznakov, ktoré sú prítomné v danom zhluku. Riešenie je určiť tie zložky váhového vektora, ktoré na jednej strane preukazujú približne tie isté hodnoty pre všetky objekty špecifického konceptu a na druhej strane majú vysokú celkovú hodnotu váhového vektora, čo indikuje ich dôležitosť. Na dosiahnutie toho je potrebné definovať prah t, aby

boli vybrané iba atribúty s veľmi nízkou hodnotou kvadratickej chyby, ktoré vykazujú hodnotu váhového vektora nad prahom t. V tomto prípade je t vlastne minimálna hodnota (dôležitosť) atribútu potrebná na to, aby bol vybraný pre popis zhluku. Zhluky sú potom prehľadne popísané charakterizujúcimi slovami, ktoré z konceptov vyextrahoval LabelSOM.

Výsledkom aplikácie GHSOM a LabelSOM algoritmov na skúmanú množinu dokumentov je hierarchická štruktúra máp, kde každý neurón je popísaný napríklad nasledovne (viď. Obr. 4.4):

- stredná kvadratická chyba pre dokumenty namapované na daný neurón (QE),
- zoznam charakteristických termov (Obr. 4.4 prezrádza aplikáciu Porterovho algoritmu na stemming v procese predspracovania dokumentov),
- linky na konkrétne súbory (ak ide o listový uzol), resp. linka na podmapu (ak existuje mapa nižšej úrovne pre daný uzol).

Zároveň každá mapa, ktorá má svojho „rodiča“ (nadradenú mapu), obsahuje na začiatku aj linku na jej zobrazenie. Takto sú všetky mapy prehľadávateľné a je možné ich používať ako vizualizačný nástroj pre oboznámenie sa s dokumentmi a pod. Nájdené zhluky môžu byť samozrejme prospešné aj pre ďalšie analýzy.



Obr. 4.4 Príklad výrezu GHSOM mapy vytvorenej z kolekcie článkov z časopisu Times[18]

[18] http://www.ifs.tuwien.ac.at/~andi/somlib/data/time60/ghsom/time_orient2_1_1_0_0.html

5 Extrakcia informácií z textov

5.1 Úvod do extrakcie informácií

V júli 2009 sa na webe v rámci vrcholových (*ang. Top-level*) domén nachádzalo približne 110,2 milióna aktívnych dokumentov[19]. Tieto neustále narastajúce počty meniacich sa textových zdrojov nutne vytvorili priestor pre vznik nových disciplín a metód určitým spôsobom analyzujúcich tak vysoký počet rozdistribuovaných a neštruktúrovaných informácií. Textové webové zdroje v prirodzenom jazyku otvárajú priestor pre extrahovanie zaujímavých informácií, ktoré by manuálnym spracovaním bolo veľmi ťažké získať.

Napríklad obchodná spoločnosť predávajúca svoj produkt cez webový obchod potrebuje zistiť mienku zákazníkov na niektorý z produktov tak, aby vedela konkrétne zacieliť svoju reklamnú kampaň. Ak má firma predávajúca literatúru v ponuke napr. 10 produktov, ustrážiť postoj zákazníkov nie je až taká zložitá úloha. Netrpezlivejší človek by však veľmi rýchlo stratil optimizmus, ak by sa na zozname predávaných artiklov nachádzalo desaťtisíc titulov a on práve potrebuje ustriehnuť negatívne postoje zákazníkov o knihách v rámci kategórie literatúry o modernom umení za obdobie posledných troch mesiacov. Čo to znamená? To znamená, že na vstupe máme kolekciu dokumentov v prirodzenom jazyku pokrývajúcich dôležité neštruktúrované informácie, a našim cieľom je z daného korpusu dát použitím vhodných metód vytvoriť štruktúrovaný výstup, ktorý ponúkne hotové výsledky pre rozhodovanie, alebo vytvorí novú štruktúrovanú databázu pre ďalšie použitie napr. pri dolovaní z dát.

V literatúre [Cunningham 2004; Feldman, Sanger 2007; Sarawagi 2007] sa uvádza viac definícií, ktoré sú vo výskumnej komunite v rámci dolovania textov viac alebo menej akceptované, no našim zámerom je uviesť takú, ktorá je konzistentná a poskytuje dostatočné prostriedky abstrakcie pre popis čo najväčšieho počtu úloh. Definíciu uvádzame podľa [Moens 2006]: *Extrakcia informácií (IE[20]) je identifikácia a následná klasifikácia špecifickej informácie nájdenej v neštruktúrovaných zdrojoch (ako napr. v textových zdrojoch v prirodzenom jazyku) do sémantických kategórií tak, aby sa daná informácia stala vhodnejšou pre ďalšie spracovanie.*

Vo všeobecnosti systémy pre extrakciu informácií sú použiteľné v prípade, že sú splnené nasledovné podmienky [Feldman, Sanger 2007]:

- Extrahované informácia je v texte špecifikovaná explicitne a nie je nutné žiadne ďalšie odvodzovanie.
- Na sumarizovanie relevantných častí dokumentu postačuje malý počet šablón.
- Požadovaná informácia je v texte vyjadrená relatívne lokálne.

[19] http://www.domaintools.com/internet-statistics/
[20] Information Extraction

Podobne ako systémy na získavanie (vyhľadávanie) informácií (IR[21]), systémy na extrakciu informácií ponúkajú užívateľovi potrebné informácie. Zatiaľ čo IR systémy identifikujú podmnožinu dokumentov z veľkého množstva textových dokumentov, IE systémy identifikujú podmnožinu informácií v rámci jedného dokumentu. Táto podmnožina nemusí nutne popisovať jadro obsahu daného dokumentu, ako to je pri IR systémoch. Zameriava sa na to, aby daná podmnožina informácií reprezentovala špecifické inštancie nájdene v texte, ktoré sú definované užívateľom.

Cieľom systémov na extrakciu informácií je nájdenie konkrétnych *objektov* (taktiež nazývaných *entít*), ich *vlastností* a zmysluplných *vzťahov* vyplývajúcich z obsahu dokumetu. Pod vzťahmi chápame *fakty* alebo *udalosti* popisujúce nájdene objekty. Napríklad, nájdený objekt v texte môže byť nejaká konkrétna spoločnosť, ktorej vlastnosťou je jej typ, napr. farmateutická spoločnosť. Možnou udalosťou je začatie spoločného úsilia spolu s inými spoločnosťami o vytvorenie nového lieku. Príklad faktu v tomto kontexte môže byť znalosť, že dané ochorenie je spôsobené konkretnými látkami. Fakty tak predstavujú statické a zvyčajne nemeniace sa informácie, pričom udalosti sú oveľa viac dyamickejšie a spravidla majú priradený nejaký časový údaj.

5.2 Metódy používané na extrakciu informácií

Vo všeobecnosti existujú dva základné delenia metód používaných na extrakciu informácií [Sarawagi 2007]. Prvým z nich je rozdelenie na metódy *znalostného inžinierstva* a *automaticky učiacich sa systémov* [Appelt, Israel 1999; Feldman 2007]. Druhé delenie člení metódy na *pravidlovo založené metódy* a *štatistické metódy*.

5.2.1 Metódy znalostného inžinierstva a automaticky učiacich sa systémov

Pri metódach *znalostného inžinierstva* ide o manuálne vytváranie gramatík (zvyčajne pravidiel) použitých ako komponenty IE systému. Na ich tvorbu je potrebný „*znalostný inžinier*“ a programátor. Požiadavkou na znalostného inžiniera je to, aby bol expertom v danej problematike a dokázal explicitne vyjadriť a formalizovať vlastné znalosti. Vyžaduje sa tu častokrát ľudská intuicia čo je významým faktorom ovplyvňujúcim výkonnosť systému. Navyše samotné vytvorenie systému je časovo náročné a zmena pravidiel býva zložitá. Výhodou tohto prístupu je však rýchlosť systému a lepší výkon oproti druhému spôsobu.

Pri *automatický učiacich sa systémoch* ide o vytvorenie pravidiel pomocou metód strojového učenia. V tomto prípade nie je potreba doménového experta ako v prechádzajúcom prípade, avšak tento prístup si vyžaduje veľké množstvo manuálne označených trénovacích dát. Na dáta sú tak aplikované vybrané metody strojového učenia, ktoré automaticky generujú model zabezpečujúci extrakciu informácií. V prípade zmeny systému je však potreba preznačkovania trénovacích dát.

[21] Information Retrieval

5.2.2 Pravidlovo založené a štatistické metódy

Pravidlovo založené metódy (pravidlá) sú založené na pevne daných vlastnostiach (predpokladom sú kvalitné a nezašumené dáta) na základe ktorých sú generované. Tvorba pravidiel je jednoduchá a rovnako ľahko sú aj interpretovateľné. Použitie pravidiel je výhodné v uzatvorených doménach, kde je potrebná a zároveň aj dostupná ľudská angažovanosť, príkladom je napr. doména medicíny.

Druhý spôsob, *štatistické metódy* sú oveľa robustnejšími metódami v zašumených neštruktúrovaných dátach. Sú tak sú vhodnejšie pre použitie v otvorených doménach ako je napr. extrakcia informácií z blogov a pod.

5.2.2.1 Pravidlá

Mnoho reálnych úloh extrakcie informácií môžu byť zvyčajne riešené pomocou pravidiel, ktoré sú manuálne (znalostným inžinierstvom) alebo automaticky (učiacími sa systémami) vygenerované. Pravidlá sú predovšetkým vhodné pre špecifické a dobre definované úlohy ako sú napr. extrakcia telefónnych resp. poštových smerovacích čísel z web stránok alebo e-mailov. Taktiež pravidlové systémy sú rýchle a dobre optimalizovateľné. Typický pravidlový systém pozostáva z dvoch častí: zoznamu pravidiel a množiny politík pre riadenie aktivácie pravidiel.

Forma a reprezentácia pravidiel

Pravidlá pre rozpoznanie samostatných entít pozostávajú z troch vzorov:

- Voliteľný vzor zachytavajúci kontext pred začiatkom danej entity
- Vzor porovnávajúci tokeny v rámci entity
- Voliteľný vzor zachytavajúci kontext po ukončení danej entity

Príkladom je identifikácia mena osoby, napr. „Ing. Juraj Hruška“. V príklade je potrebné detekovať titul pred menom na základe zoznamu titulov uložených v slovníku (obasuje celý zoznam titulov: „prof“, „doc“, „Ing“, „Mgr“, „Bc“ a ďalšie). Po detekcií titulu pred menom nasleduje identifikácia mena, čo znamená dve za sebou idúce slova s veľkým počiatočným písmenom, príklad pravidla je možné zapisať nasledovne:

```
({SlovníkovýZáznam = titul} {ReťazecZnakov = ".”}
{PravopisnýTyp = SlovoSVeľkýmPočiatočnýmPísmenom}{2}) →
Meno Osoby,
```

kde `{2}` znamená, že pravidlo uvedené pred týmto zápisom bude za sebou vykonané toľko krát, aká hodnota je uvedená v zátvorkach.

Iným príkladom môže byť identifikácia názvu spoločnosti v anglickom jazyku. Pre dva typy zápisov „*The XYZ Corp.*“ a „*ABC Ltd.*“ je možné vytvoriť následujúce pravidlo:

```
({ReťazecZnakov = "The”}? {PravopisnýTyp =
SlovoVeľkýmiPísmenami} {PravopisnýTyp =
SlovoSVeľkýmPočiatočnýmPísmenom, Slovníkový Záznam =
KoncovkaSpoločnosti}) → Názov spoločnosti
```

V tomto prípade reťazec „*The*“ je voliteľný a spoločný zápis pravopisného typu a slovníkového záznamu v tretej podmienke vyjadruje to, že musia byť splnené oba predpoklady.

Organizácia kolekcie pravidiel

Typický pravidlový systém pozostáva z veľkej kolekcie pravidiel, a tak sa častokrát stáva, že niekoľko pravidiel pokrýva tú istú entitu. Vznikajú tak konfliktné akcie a prekrývanie pravidiel. Na zamedzenie takýchto problémov je potrebné zabezpečiť kontrolu nad aktiváciou pravidiel. Keďže často krát sú pravidlá navrhované znalostnými inžiniermi, kontrola aktivácie pravidiel je tak často založená na rôznych heuristikách. Tu však existujú zaužívané spôsoby predchádzania uvedených problémov:

- *Neusporiadaná kolekcia pravidiel* – je to veľmi jednoduchá metóda, kde sú aktivované pravidlá nezávisle na sebe. Ich konflikty a prekrývanie je riešene iba pomocou jednoduchých pravidiel: pravidlá označujúce dĺhšiu časť textu sú preferované; prekrývajúce pravidlá sú vzájomne spojené do väčšieho celku.
- *Usporiadaná kolekcia pravidiel* – táto metóda využíva usporiadanie pravidiel podľa priorít. V prípade konflitnej situácie je tak preferované pravidlo s vyššou prioritou.

Automatická tvorba pravidiel

Ako bolo spomenuté vyššie, pravidlá sú zvyčajne generovaná manuálne znalostnými inžiniermi. Okrem toho je však možné pravidla generovať automaticky pomocou metód strojového učenia. Tu je požiadavka trénovacích dát, v ktorých sú jednotlivé entity označené. Aplikovaním učiacého algoritmu na takéto dáta sú získané pravidlá, príklad činnosti algoritmu je nasledovný:

```
Algoritmus:

R – množina pravidiel, na žiatku je prázdna.
Pokial existuje entita e v texte, ktorá nie je pokrytá
pravidlom z R.
Vytvor nové pravidlo pre e.
Pridaj nové pravidlo do R.
Po ukončení orež redundantné pravidlá
```

Podľa metód strojového učenia [Machová 2002] existujú dve základné prístupy k tvorbe pravidiel. Prístupy označujú smer formalizácie pravidiel:

- *Zdola nahor* – v tomto prípade sa vytvárajú maximálne špecifické pravidlá tak, aby ich presnosť bola 100%, avšak ich pokrytie je slabé. Ďalším chodom algoritmu sú špecifiké pravidlá nahradzované všobecnejšími, čím výrazne rasie pokrytie avšak za cenu zhoršenia presnosti.
- *Zhora nadol* – ide o opačný prístup ako v prechádzajúcom prípade. Tu sú vytvorené najprv čo najviac všeobecné pravidlá, ktorých pokrytie je 100%, avšak ich presnosť je slabá. Iteráciou procesu

tvorby pravidiel sú všeobecné pravidlá nahradzované špecifickejšími, čím rastie presnosť, ale klesá pokrytie.

5.2.2.2 Štatistické metódy

Štatistické metódy extrakcie informácií menia problém extrakcie na problém dekompozície neštrúkturovaného textu na častí a následnoho značenia rôznych častí dokumentu. Najviac používanými formami dekompozície textu sú *tokenovo-úrovňové metódy*, ktoré rozdeľujú text na tzv. tokeny. Rozdelenie je vykonávané na základe množiny znakov, ktorej znaky rozdeľujú text na tokeny, sú to napr. medzery, čiarky, bodky a pod. Ďalej, vo fáze značenia je každému tokenu priradená značka popisujúca daný token. Druhým typom metód sú *segmentovo-úrovňové metódy* rozdeľujúce text na väčšie časti ako sú tokeny, na tzv. chunky. Na rozdelenie textu sa využívajú metódy na spracovanie prirodzeného jazyka, a tak detekované chunky môžú pozostávať aj z viacslovných entít. Ďalšími metódami sú metódy založené napr. na *gramatike jazyka*.

Tokenovo-úrovňové metódy

Úlohou metód tohto typu je priradenie značky každému tokenu v texte. Predpokladajme postupnosť tokénov $x = x_1,...,x_n$, ktorej každému tokenu x_i je priradená značka z množiny Y. Množina Y obsahuje zoznam typov jednotlivých entít, napr. {Organizácia, Mesto, Konferencia, Rok, InýTyp}. Pre umožnenie označenia viacslovných entít je možné ich značenie ako "Entita_Začiatok", "Entita_Trvanie", "Entita_Koniec", čo je známe ako BCEO[22] značenie. Výstupom tejto metódy je postuponosť označení jednotlivých tokenov $y - y_1,...,y_n$.

Uveďme si príklad, majme vetu, rozdelenú na tokeny, Tab. 5.1.

Tab. 5.1 Rozdelenie vety na tokeny.

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
x	*Technická*	*univerzita*	*v*	*Košiciach*	*organizovala*	*SAMI*	*2009*	*v*	*Herľanoch*
y	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

Označenie jednotlivých tokenov vo vete uvedenej v Tab. 5.1 bude podľa Y takéto:

```
Organizácia_Začiatok, Organizácia_Trvanie,
Oranizácia_Trvanie, Organizácia_Koniec, InýTyp, Konferencia
Rok, InýTyp, Mesto
```

Týmto je tak získana postupnosť označení tokenov y.

[22] Skratka z anglických slov; B = Begin, C = Continue, E = End, O = Other

Segmentovo-úrovňové metódy

Výtupomo tohto typu metód je postupnosť segmentov definujúcich jednotlivé entity na miesto postupnosti označení tokenov ako v predchádzajúcej metóde. Formálne to môžeme zapísať, že pre vstup dĺžky n je postupnosť segmentov $s_1,...,s_p$, kde prvý segment začína na $1.$ pozícií a posledný segment končí na $n.$ pozícií. Každý segment s_j má pozíciu začiatku l_j, pozíciu konca u_j a označenie y_j z Y. Príklad uvedený v predchádzajúcej metóde tak bude mať nasledujúci tvar, viď Tab. 5.2 nižšie.

Tab. 5.2 Rozdelenie vety na segmenty.

i	1	2	3	4	5	6	7	8	9
x	Technická univerzita v Košiciach				organizovala	SAMI 2009		v	Herľanoch
(l_j, u_j, y_j)	*1, 4, Organizácia*				*5, 5, InýTyp*	*6, 7, Konferencia*		*8, 8, InýTyp*	*9, 9, Mesto*

5.3 Najčastejšie úlohy riešené v rámci extrakcie informácií

Riešenie výskumných úloh a implementácia konkrétnych systémov adresujúcich potreby praxe sa hýbe okolo riešenia niekoľkých kategórií úloh, ktoré by sme mohli spoločne pomenovať *podúlohami* problematiky extrakcie informácií. Medzi tieto podúlohy patria nižšie uvedené úlohy, z ktorých prvé dve sú podrobnejšie popísané v nasledujúcich dvoch kapitolách:

- **Rozpoznávanie pomenovaných entít** – *named entity recognition (NER)*. Cieľom je nájdenie a identifikácia pre používateľa zaujímavých (pomenovaných) objektov v texte, viď. 5.3.1.
- **Extrakcia relácií** – *coreference resolution (CO)*. Ide o nájdenie a identifikovanie relácií (vzťahov) medzi pomenovanými objektmi nájdenými pomocou predchádzajúcej úlohy, viď. 5.3.2.
- **Šablónové prvky.** Úlohou šablónových prvkov – *template elements (TE)* je nájdenie a pridanie jednoduchých popisných informácií k objektom extrahovaným pomocu NER.
- **Šablónové relácie.** Úlohou šablonových relácií – *template relations (TR)* je nájdenie a vyjadrenie doménovo nezávyslých relácií medzi objektmi s prihliadnutím na identifikáciu objektov pomocou TE.
- **Scenárové šablóny.** Scenárové šablóny – *scenarious templates (ST)* vyjadrujú doménovo a úlohovo špecifické objekty a relácie.

5.3.1 Rozpoznávanie pomenovaných objektov

Prvou z podúloh je rozpoznávanie pomenovaných objektov (NER[23]), ktorej cieľom je identifikácia pre používateľa zaujímavých pomenovaných objektov vo vstupných textoch (väčšinou v prirodzenom jazyku). Tu môže ísť o mená osôb, názvy organizácií, názvy miest alebo geografických lokalít, alebo aj rôzne typy peňažných alebo numerických výrazov. Univerzálne je akceptovaných niekoľko hlavných kategórií objektov, [ACE 2004] definuje týchto sedem: *osoby*, *organizácie*, *zariadenia* (obmedzené na budovy a nehnuteľný majetok), *miesta* (lokality), *geo–politické územné celky* (GPE), *dopravné prostriedky* a *zbrane*. Ďalšími, často vyskytujucími sa kategoriami sú: časové a dátumové údaje, e-mailové adresy a číselné vyjadrenia (paňazí, percént, váh a pod). Inými môžu byť doménovo-špecifické objekty, napr. názvy liekov či chorôb alebo aj bibliografické referencie a pod.

Príkladom NER je identifikácia názvov všetkých firiem a spoločností z preddefinovanej množiny dokumentov, ktorá by mala zahŕňať aj identifikáciu takých firiem, ktoré neboli predtým používateľovi známe. Navyše je dobré, ak je možné špecifikovať systému, aby identifikoval firmy len určitého typu, napr. len slovenské firmy alebo firmy zaoberajúce sa IT technológiami.

Ďalšou z možných úloh IE systémov, ktorá je v princípe zhodná s identifikáciou objektov je tzv. *značkovanie*. Tu napríklad môžu byť v texte identifikované a označené rôzne, vyššie spomenuté, typy objektov. Výstupom takýchto systémov je označkovaný text, ktorý môže slúžiť pri neskoršom spracovaní ako vstupné dáta, resp. môže byť použitý na indexovacie účely pomocou IR systémov[24]. Iným príkladom je priame zobrazenie spracovávaného textu a označenie, či zvýraznenie relevantných informácií, ktoré môžu pomôcť pri ďalšej analýze dokumentu.

Zástupcom posledného z vyššie uvedených príkladov je nástroj pre rozpoznávanie entít v medicínskych textoch (pozri aplikáciu v prílohe B.3). Tu je cieľom rozpoznanie výrazov z oblasti medicíny ako sú názvy génov, proteínov či názvy chorôb alebo častí ľudského tela a pod. Cieľom takéhoto rozpoznávania je to, že správne rozpoznané výrazy umožňujú extrakciu ďalších informácií na vyššej úrovní, ako je napríklad rozpoznávanie vzťahov medzi entitami. V príklade ide o doménovo špecifický problém bez dostatočného zdroja trénovacích dát, a tak je možné použíť iba prístup znalostného inžinierstva.

Tu treba mať na pamäti, že ide o obrovské množstvo termínov (a ich vzájomných kombinácií), z čoho vyplýva, že nie je možné vytvoriť kompletný slovník, ktorý by obsahoval všetky potrebné výrazy. Navyše, keďže jeden koncept, napr. názov choroby, môžeby byť vyjadrený viacerými spôsobmi (viď. príklad nižšie), nie je možné použiť iba jednoduché vyhľadávanie termínov. Sú tak vytvorené jazykové pravidlá pre porovnávanie termínov v texte s termínmi uloženými v slovníku. Problémom je, že neexistuje jednoznačné prepojenie medzi konceptmi a termínmi, ktoré predstavujú textové vyjadrenie daných konceptov, t.j. existuje variabilnosť a dvojznačnosť

[23] Named Entitiy Recognition
[24] napr. internetové vyhľadávače

termínov.

Variabilnosť termínov – je to schopnosť prirodzeného jazyka vyjadriť jeden koncept viacerými termínmi, tzv. synonymami. V oblasti medicíny to môže byť vážny problém, kde je napríklad mnoho synoným pre vyjadrenie proteínov, enzýmov či génov. Nie je nezvyčajné, že jeden koncept je možné vyjadriť aj šiestimi rôznymi termínmi. Problém variabilnosti termínov je možné riešiť použitím synonymických slovníkov.

Dvojznačnosť termínov – nastáva vtedy, keď jeden termín je použitý pre vyjadrenie viacerých konceptov, tzv. homonymá. Význam daného termínu je tak úzko závislý od kontextu v ktorom sa nachádza.

Ďalším problémom, prečo je vyhľadávanie entít v texte náročnou úlohou je ten fakt, že jednotlivé termíny (jedno alebo viacslovné) popisujúce jeden koncept môžu byť zapísané rozličným spôsobom, obzvlášť v doméne medicínskych textov, uvedieme si príklad.

Majme koncept s menom ***A-23187*** ktorý predstavuje konkrétne antibiotikum. Daný koncept je však možné vyjadriť nasledujúcimi termínmi:

- *A-23187*
- *A23187*
- *Antibiotic A23187*
- *A23187, Antibiotic*
- *a iné variácie*

Iným príkladom je použitie rôznej terminológie pre rovnaké koncepty:

- *Floor of Mouth*
- *Floor, Mouth*
- *Floors, Mouth*
- *Mouth Floor*
- *Mouth Floors*

5.3.2 Extrakcia relácií

Všeobecne v úlohách extrakcie informácií z textov nie je postačujúce iba nájdenie pomenovaných objektov, ale je potrebne nájdenie a identifikovanie relácií (vzťahov) medzi pármi týchto objektov. *Extrakcia relácií*, v angličtine *coreference resolution (CO)*, je tak druhou podúlohou problematiky extrakcie informácií identifikujúcou relácie medzi objektmi.

Pokial v predchádzajúcej kapitole bolo uvedených sedem hlavných kategórií objektov definovaných v [ACE 2004], tak podobne sú definované aj najčastejšie vyskytujúce sa relácie, ktoré vyjadrujú: *lokalizáciu*, *blízkosť*, *časť*, *postavenie* resp. *funkciu* a *sociálnosť*. Napríklad relácia *lokalizácie* popisuje adresu/sídlo osoby alebo organizacie, relácia *postavenia* zas popisuje vzťahy medzi osobami a organizaciami a pod.

Vezmime si napríklad fragment textu: „*Technická Univerzita v Košiciach sa pričlenením Leteckej fakulty stala najväčšou univerzitou východoslovenského*

regiónu". Prepisom textu z príkladu do jednoduchých oznamovacích viet môžeme získať jednoznačný pohľad na relácie (v tomto prípade ide o binárne relácie) medzi entitami: Fakulta A *jeSúčasťou* univerzity B (vyjadrenie pre reláciu postavenia/funkcie). Univerzita B *pôsobíVo* východoslovenskom regióne (vyjadrenie pre reláciu lokalizácie).

Uvedený príklad popisuje binárne relácie, ktorých identifikáciu je možné vykonať tromi možnými spôsobmi:

- V prvom prípade sú už identifikované objekty z textu, pričom je predložený fixný pár objektov pre ktorý je hľadaná jedna alebo viacero väzieb.
- Druhý spôsob uvažuje o tom, že je daná relácia r a nejaký objekt o. Cieľom je tak nájdenie všetkých objektov v texte, ktoré majú reláciu r s objektom o.
- Tretí spôsob predpokladá veľký korpus neštruktúrovaných dát (textov), kde nemôžeme predpokladať označené objekty, a tak je daná iba relácia r. Úlohou je tak nájdenie všetkych párov objektov, medzi ktorými je táto relácia r.

Identifikácia relácií medzi daným párom objektov

Nech je daná množina relácií R, pár objektov a úloha, ktorej cieľom je identifikácia všetkých relácií z R medzi všetkými označenými objektmi v texte. V zásade pre extrakciu informácií platí predpoklad, že dva uvažované objekty (o ktorých predpokladame, že sú vo vzájomnej relácií) sú vo vzájomnej blízkosti v texte, zvyčajne v jednej vete alebo jej časti. Základnú identifikáciu relácie je tak možné definovať následujúcim spôsobom. Uvažujme úryvok textu x a dva označené objekty e_1 a e_2 v x, pričom y je množina relácií, ktoré existujú medzi e_1 a e_2. Nájdenie množiny y je možné pomocou viacerými spôsobmi.

Poloha tokénov. Tokény ktoré sú v okolí a medzi dvoma objektmi často vedú k extrakcií relácie. V uvedenom príklade (nižšie) je relácia lokalizácie umiestnená medzi dvoma identifikovanými objektmi (sú označené tagmi v ostrých zátvorkach).

> `<Organizácia>` *Technická univerzita v Košiciach* `</Organizácia>`
> *pôsobí vo* `<Lokalizácia>` *východoslovenskom regióne* `</Lokalizácia>`.

Part of Speech tagy. Part of Speech tagy (POS) majú významnú úlohu pri extrakcií relácií. Pomocou týchto tagov sú priradené jednotlivé slovné druhy k slovám, a tak je identifikácia relácií jednodúchšou a presnejšou. Relácie medzi objektmi, ktoré sú zvyčajne podstatné mená resp. menné frázy sú tak detekované na základe slovies.

> `<Lokalizácia>` *Technická univerzita v Košiciach* `</Lokalizácia>`
> *organizovala* `<Conference>` *SAMI* `</Conference>` *v roku 2009*.

V uvedenom príklade slovo „organizovala" je označené ako sloveso, čo zabezpečuje spoľahlivú extrakciu relácie.

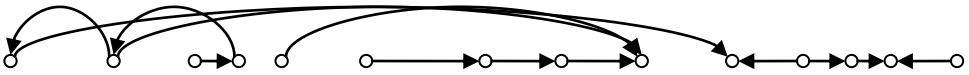
Syntaktická analýza. Syntaktická analýza vety umožňuje identifikovať významné typy fráz ako predmetové, podmetové a slovesné frázy. Tým je

umožnená tvorba syntaktickej stromovej štruktúry, čo prináša nové a cennejšie informácie pre identifikáciu relácií než POS. Príkladom je veta:

> <Lokalizácia> *Herľany* </Lokalizácia>*, vzdialené 40* km *od* <Lokalizácia> *Technickej univerzity v Košiciach* </Lokalizácia>, *hostili* <Conference> *SAMI* </Conference> *v roku 2009*.

Relácia „hostili“ medzi „Technická univerzita v Košiciach“ a „SAMI“ by mohla byť v uvedenom príklade preferovaná, avšak správnou je relácia medzi objektmi „Herľany“ a „SAMI“. V tomto prípade syntaktická analýza identifikuje „Herľany“ ako podmet, ktorý je Y rovnocenne spojený prísudkom „hostili“ popisujúcim predmet „SAMI“.

Závislostný graf. Využíva sa namiesto syntaktickej stromovej štruktúry, ktorá v prípadoch analýzy celej vety môže byť veľmi zložitou. Veta z príkladu vyššie je tak zobrazená na obrázku nižšie, kde sú znázornené všetky závislosti medzi slovami.



Obr. 5.1 Závislostný graf vety.

5.4 Iné úlohy riešené v rámci extrakcie informácií

5.4.1 Extrakcia a dolovanie názorov

Existuje mnoho aplikácií kedy nestačí extrahovať z textov len názvy konkrétnych objektov, ale čoraz viac – hlavne v oblasti e-Commerce – narastá potreba zistiť dôležité vlastnosti popisovaných objektov. Tu vznikol priestor pre vyformovanie sa nového smeru – *dolovania názorov* zastrešujúceho metódy extrakcie názorov ľudí na konkrétne vlastnosti popisovaných objektov. Ako zdroj dát pre extrakciu slúžia predovšetkým diskusie v internetových obchodoch k predávaným produktom, blogy recenzujúce produkty alebo služby, filmové databázy na webe ponúkajúce možnosť komentovania zaradených filmov, atď.

Podľa toho ako sa niektorí odborníci [Ding a kol. 2008] pozerajú na súčasný stav textových informácií publikovaných predovšetkým na Webe, je ich možné tieto informácie rozdeliť do dvoch hlavných skupín – *na fakty a názory*. Čo sa týka faktov, tie môžeme považovať za „objektívne tvrdenia o entitách a udalostiach vo svete“. Na druhej strane „názory sú subjektívne tvrdenia o entitách alebo udalostiach odrážajúce ľudské dojmy alebo vnímania“.

Myšlienka dolovania názorov ako jednej z podoblastí extrakcie informácií nie je v odbornej komunite až taká nová. Prvýkrát bola idea subjektivity nastolená takmer pred 15 rokmi [Wiebe 1994] a odvtedy sa medzi odborníkmi používa niekoľko navzájom zamieňaných pojmov označujúcich tú istú vec. Podľa aktuálne riešených problémov v rámci komunity môžeme

sledovať dva rôzne smery výskumu – *klasifikáciu dojmu* a *dolovanie názorov založené na vlastnostiach*.

5.4.1.1 Klasifikácia dojmu

Klasifikácia dojmu je klasifikačný problém, ktorý klasifikuje textové dokumenty do najčastejšie troch kategórií – pozitívnej, neutrálnej a negatívnej. Táto kategorizácia sa môže vykonať na úrovni viet alebo celých dokumentov.

Čo sa týka klasifikácie na úrovni celých dokumentov výskumníci vymysleli niekoľko prístupov k určeniu výstupnej triedy – sémantickej orientácie analyzovaného textu. V práci [Turney 2001] je prezentovaný jednoduchý algoritmus nekontrolovaného učenia pre klasifikáciu recenzií do kategórií *odporúčam* a *neodporúčam*. Celková sémantická orientácia textu je vypočítaná ako súčet čiastkových predikovaných orientácií viet obsahujúcich prídavné mená a príslovky.

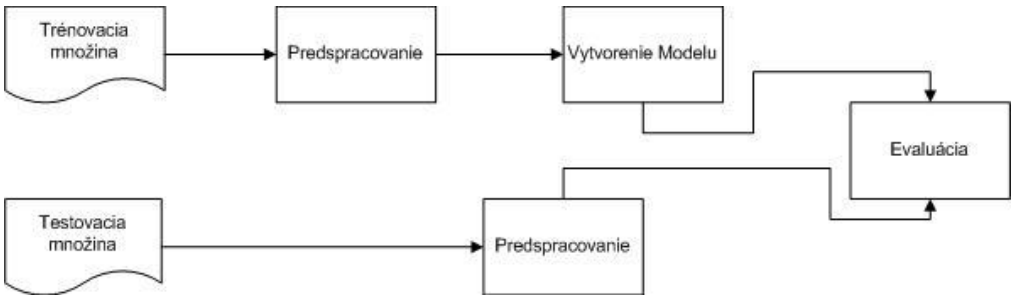
Klasifikovanie dojmov na úrovni viet ponúka vyššiu variabilitu čo do možností extrahovania dojmov orientovaných na konkrétne detaily o entitách vyjadrené v jednotlivých vetách. Napríklad v práci [Hatzivassiloglou 2000] je pri extrakcii vidno rozlišovanie medzi vetami prezentujúcimi názory a vetami faktického charakteru.

5.4.1.2 Dolovanie názorov zamerané na vlastnosti

Podľa [Liu 2008] každá entita môže byť popísaná konečnou množinou vlastností (napr. spotreba auta, frekvencia CPU, trvanlivosť mlieka, atď.) ktoré sama zahŕňa. Každá vlastnosť môže byť ďalej popísaná konečným počtom slov alebo fráz, ktoré sú synonymami. Ak sa pozrieme na dokument, ktorý je zdrojom pre vyjadrenie názorov, môžeme pozorovať skutočnosť, že osoba, ktorá zaujala patričný názor, sa vyjadruje ku konkrétnym vlastnostiam analyzovaného produktu. Pre každú vlastnosť on alebo ona vyberá z konečnej množiny synoným slovo alebo frázu pre popísanie danej vlastnosti a takto k nej vyjadruje svoj pozitívny, negatívny alebo neutrálny postoj – názor.

6 Službovo–orientovaný pohľad na dolovanie znalostí z textov

Reprezentácia pomocou služieb umožňuje nazerať na proces dolovania v textoch ako na pracovný tok služieb. Jednotlivé etapy procesu dolovania v textoch môžeme v prípade vhodnej implementácie (napr. ako webové, alebo gridové služby) vyjadriť ako procesy. Tieto procesy definujú výsledný pracovný tok. Zjednodušená a zovšeobecnená schéma typického pracovného toku pri takejto aplikácii je znázornená na obrázku nižšie (Obr. 6.1). Jednotlivé stavebné bloky sú v tomto prípade realizované službami (v niektorých prípadoch aj viacerými, komponovaným službami).



Obr. 6.1 Dolovanie znalostí z textov ako pracovný tok služieb

Problém definovania, sémantického popisu a kompozície služieb pre takto definované pracovné toky je podrobne popísaný v dizertačnej práci Martina Sarnovského [Sarnovský 2009]. V tejto publikácii sa zameriame len na jeden aspekt, ktorý takéto chápanie procesu dolovania v textoch prináša, a síce možnosť ich distribuovaného spracovania.

Spoločným znakom takéhoto typu úloh na kolekciách textových dokumentov je totiž fakt, že algoritmy používané na ich riešenie sú časovo a výpočtovo náročné. Narastajúci objem dát predstavuje veľký problém, pretože pri využití klasických výpočtových prostriedkov môže riešenie takýchto úloh trvať neúnosne dlho.

Vhodným riešením tohto problému sa javí byť distribúcia algoritmov a dát s využitím SOA – službovo orientovanej architektúry (napr. webové, alebo gridové služby). Ich výhoda spočíva v tom, že pre ich nasadenie je možné využiť už existujúce výpočtové kapacity a infraštruktúru (napr. Internet). Technológie pre distribuované objavovanie znalostí sú obzvlášť vhodné pre aplikácie, ktoré zvyčajne spracúvajú veľké objemy dát alebo roztrúsené dátové zdroje (digitálne knižnice, vedecké simulácie, obchodné dáta, ...), ktoré nie je možné analyzovať v prijateľnom čase na klasických počítačoch.

6.1 Distribuované dolovanie v textoch

Jednou z možností ako technicky realizovať distribuované dolovanie v textoch je využiť na tento účel výpočtový Grid. To je distribuovaná architektúra, ktorá v sebe integruje distribuované a paralelné počítanie, teda predstavuje vhodný nástroj pre distribuovanú obrovského množstva analýzu

dát [Foster, Kesselman 1999; Foster, Kesselman 2004]. Grid je podľa viacerých definícií infraštruktúra určená najmä pre zdieľanie heterogénnych zdrojov, bez prítomnosti centralizovaného riadenia. Infraštruktúra to má byť spoľahlivá, bezpečná, štandardizovaná, flexibilná, koordinovaná. Viaceré snahy o praktickú implementáciu takejto infraštruktúry však narážali na problémy spojené s nekompatibilitou a heterogenitou používaných protokolov, či jazykov. Táto situácia prirodzene viedla k návrhu gridovej infraštruktúry, ktorá by bola orientovaná na služby a bola prakticky postavená na webových službách. Takéto služby dolovania znalostí by potom umožnili podnikom a spoločnostiam distribuovať výpočtovo náročnú analýzu dát na veľký počet vzdialených zdrojov (počítačov, klastrov, superpočítačov). To prirodzene vedie aj ku vzniku nových algoritmov a techník, ktoré umožňujú analyzovať dáta tam, kde sú uložené, čo je v kontraste s klasickými metódami, kde je potrebné presunúť dáta na centrálne miesto, kde sa vykonáva ich analýza (čo v prípade objemných dát môže klásť neúnosné nároky na počítačovú sieť).

V kontexte dolovania z textov existuje viacero možností ako využiť paralelné resp. distribuované počítanie na zníženie časovej náročnosti celého procesu. V procese dolovania z textov sa stretávame čoraz viac so sekvenčnými úlohami, ktoré vzhľadom na objem niektorých dátových kolekcií vyžadujú dlhý čas na spracovanie. Využitím služieb ktoré grid poskytuje je možné určité časti týchto úloh paralelizovať a výrazne tak skrátiť priebeh samotného procesu dolovania.

Distribuované indexovanie je možné najmä využiť v oblasti systémov vyhľadávania informácií. Výskum v oblasti distribuovaného indexovania sa zameriava najmä na zefektívnenie indexovania, teda minimalizáciu zaťaženia serverov a udržiavanie relatívne malých veľkostí indexov. Existujú modely distribuovaného vyhľadávania informácií, najčastejšie založené na viacerých nezávislých serveroch indexujúcich lokálne kolekcie dokumentov. Popri nich beží jeden hlavný server, ktorý presmeruje požiadavky užívateľov na ten-ktorý lokálny server. Tento prístup ale predpokladá, že jednotlivé lokálne kolekcie dokumentov tvoria dokopy celý korpus.

Vo fáze budovania samotného modelu (klasifikačného, alebo zhlukovacieho) je nutné zohľadniť aj spôsob funkcie konkrétneho algoritmu. Potom je možné prístupy k distribúcii rozdeliť do dvoch skupín s ohľadom aj na štruktúru samotných algoritmov.

- *Dekompozícia riadená dátami* - tento spôsob vychádza priamo z povahy niektorých algoritmov. Vychádzame z predpokladu, že pri určitých druhoch algoritmov je postačujúca možnosť rozdeliť vstupnú dátovú množinu T na i navzájom disjunktných podmnožín T_i. Rozdeľujeme takto priamo vstupnú trénovaciu množinu na podmnožiny, na ktorých natrénujeme zodpovedajúce čiastkové modely pre jednotlivé podmnožiny. Pre vytvorenie výsledného modelu je potrebné navrhnúť mechanizmus spájania výsledného modelu. Medzi algoritmy, ktoré sú prirodzene vhodné pre takýto typ distribúcie patrí metóda *k*-najbližších susedov (a všeobecne všetky metódy učenia založeného na inštanciách), alebo metódy Support Vector Machines.

- *Dekompozícia riadená modelom* – tento spôsob dekompozície vychádza z predpokladu zachovania vstupnej dátovej množiny a modifikácie samotných algoritmov. V tomto prípade sa jedná o dekompozíciu samotnej tvorby modelu. Miesto a spôsob samotnej dekompozície je závislý sa konkrétnom uvažovanom algoritme. Vo všeobecnosti sa jedná o dekompozíciu na podúlohy v rámci samotnej tvorby modelu tak, aby jednotlivé podúlohy boli riešiteľné nezávisle na sebe. Do tejto skupiny spadá niekoľko typov algoritmov, napr. algoritmy indukcie klasifikačných stromov, alebo klasifikačných pravidiel, zhlukovacie algoritmy založené na samoorganizujúcich sa mapách, alebo združené klasifikátory (bagging, boosting).

6.1.1 Distribuovaná kategorizácia textov

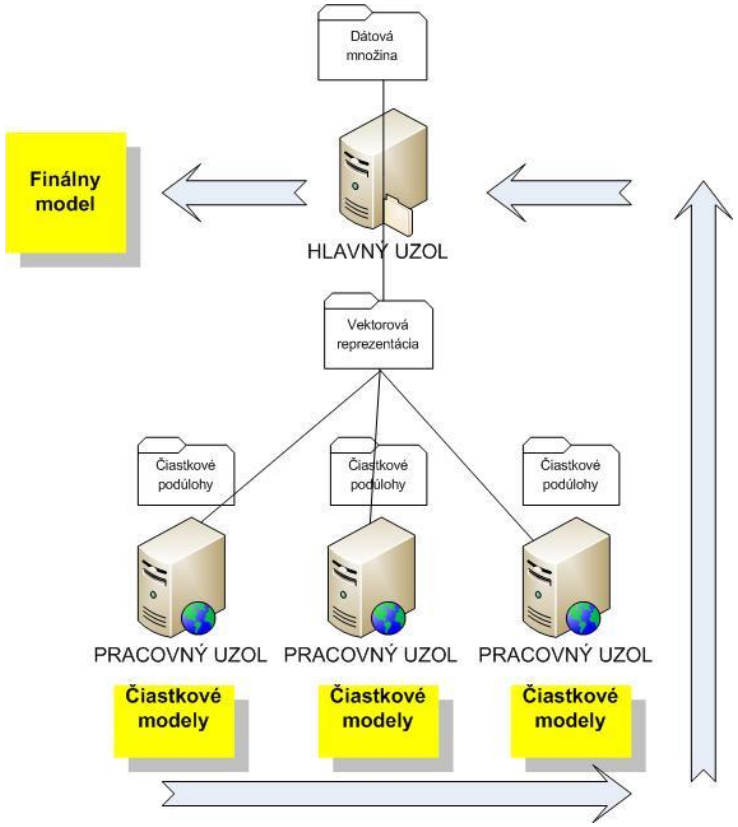
Budovanie rozhodovacích stromov sa pri úlohách dolovania v textoch odlišuje od dolovania v dátach v jednom aspekte. Pri štandardných kolekciách textových dokumentov sa stretávame so skutočnosťou, že na rozdiel od klasických úloh dolovania v dátach môže jeden dokument patriť súčasne do viacerých kategórií (tzv. multi-label klasifikácia). Konštrukcia jediného rozhodovacieho stromu, ktorý by vedel klasifikovať dokumenty do viacerých kategórií v takomto prípade nie je možná. Riešenie tohto problému v úlohách dolovania v textoch spočíva v konštrukcii binárnych rozhodovacích stromov pre každú kategóriu zvlášť.

Väčšinou sú takto binárne stromy budované sekvenčne, čo vedie pri dátových kolekciách s veľkým množstvom kategórií k neúmernému predĺženiu času konštrukcie modelu. Konštrukcia binárnych stromov pre jednotlivé kategórie je nezávislý proces, ktorý je možné ľahko a efektívne paralelizovať.

Navrhnutá metóda buduje multi-label rozhodovací strom distribuovane, rozdelením jednotlivých podúloh na pracovné uzly, tzv. *worker nodes* [Jančiak a kol. 2006a], [Jančiak a kol. 2006b]. Pracovné uzly sú pracovné stanice prepojené sieťou a ich úlohou je iba spustiť konkrétnu inštanciu služby na daných dátach. Druhým typom uzla použitom v tomto prístupe je tzv. *master*, alebo hlavný uzol, na ktorom beží samotná služba a ktorý jednotlivé čiastkové podúlohy priradzuje na pracovné uzly.

- Na začiatku hlavný uzol spracuje dátovú množinu do vektorovej reprezentácie. Na hlavnom uzle prebehne proces predspracovania dát, spolu s vytvorením vektorovej reprezentácie dátovej množiny. Táto sa potom stáva vstupom do služby realizujúcej konštrukciu samotného klasifikačného modelu. Zároveň hlavný uzol začne prideľovať jednotlivé podúlohy (úlohy vybudovania parciálnych binárnych klasifikátorov) na jednotlivé pracovné uzly.
- Na jednotlivých pracovných uzloch prebieha vytvorenie čiastkových modelov - sú spustené inštancie služby tvorby klasifikačného modelu nad prislúchajúcimi dátami z vektorovej reprezentácie. Výsledkom je množina binárnych klasifikačných modelov pre jednotlivé kategórie.
- Binárne klasifikátory sú potom poslané naspäť na hlavný uzol, kde bola služba spustená. Po vytvorení a poslaní všetkých binárnych

klasifikátorov sú potom na hlavnom uzle spojené do výsledného klasifikačného modelu.



Obr. 6.2 Všeobecná schéma princípu distribúcie tvorby klasifikačného modelu na podúlohy

Schému distribúcie ktorú sme experimentálne realizovali a otestovali na dvoch textových kolekciách znázorňuje Obr. 6.2. Vstupom celého procesu je kolekcia dokumentov. Na tejto najskôr prebehne fáza predspracovania na hlavnom uzle. Táto zahŕňa najprv lexikálnu analýzu – tokenizáciu, ktorá sa realizuje pomocou štandardného tokenizéru (dokáže rozlíšiť viacero prvkov v texte ako čísla, slová, dátumy, akronymy, adresy elektronickej pošty, interpunkciu). Následne sa konvertujú všetky písmená na malé a odstránenia sa nevýznamové slová. Výstupom prvej fázy sú štatistiky, frekvencie výskytu, zoznam kategórií a slovník termov, ktoré sa uložia do samostatných súborov. Vytvorí sa vektorová reprezentácia (váhovanie na základe *tfidf* schémy) a výsledné vektory sa normalizujú. Normalizované vektory sa uložia špeciálnym spôsobom (ukladajú sa len nenulové prvky tak, aby bolo jasné, ktorému termu odpovedajú) do výsledného súboru, ktorý tak obsahuje *tfidf* profily dokumentov a je základným vstupom pre distribúciu - spájanie.

V ďalšom kroku sa na hlavnom uzle sa zo štatistík dátovej množiny (z počtu kategórií) vypočíta, koľko binárnych klasifikátorov je potrebné skonštruovať na vybudovanie výsledného klasifikačného modelu. Zároveň sa zistí počet dostupných pracovných uzlov v sieti. Tento počet si volí sám používateľ ako parameter v konfiguračnom súbore, kde zadá počet dostupných uzlov spolu s ich URL adresami. Na základe týchto informácií potom začne prideľovať

jednotlivé podúlohy (úlohy vybudovania parciálnych binárnych klasifikátorov) na jednotlivé pracovné uzly.

Ak je n počet kategórií, teda počet binárnych klasifikátorov ktoré je potrebné vybudovať a u je počet dostupných uzlov Gridu, potom:

- Ak $n \leq u$, do fronty úloh prvých n uzlov sa priradí práve jedna čiastková úloha.
- Ak $n > u$, v prvej iterácii sa prvých u čiastkových úloh priradí prvým u uzlom, v nasledovných iteráciách sa priradzujú zvyšné čiastkové úlohy zaradom na jednotlivé uzly dovtedy, kým už nie je čo priradzovať.

Z toho je teda evidentné, že ak je počet pracovných uzlov menší ako počet čiastkových úloh, pracovné uzly potom dostanú viac čiastkových úloh tak, že počet čiastkových úloh na jeden uzol je rovnaký (maximálny rozdiel jedna čiastková úloha), ale bez ohľadu na výpočtovú zložitosť konkrétnej čiastkovej úlohy. Na základe tohto faktu bolo neskôr po sérii experimentov doplnené jednoduché optimalizovanie pracovného rozloženia.

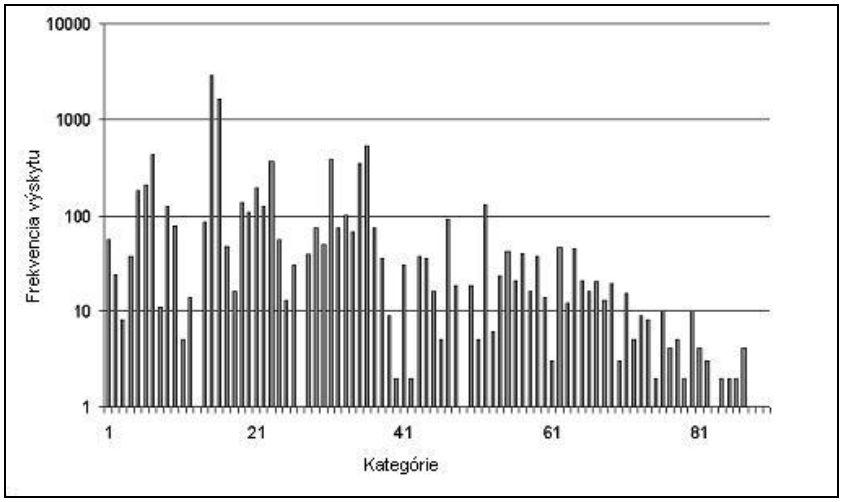
6.1.1.1 Popis experimentov s distribuovanou kategorizáciou textov na gride

Uvedené experimenty boli vykonávané na lokálnej sieti na Institute of Scientific Computing vo Viedni. Ako testovacie prostredie poslúžila sieť pracovných staníc, Sun Blade 1500 s konfiguráciou 1062MHz Sparc CPU, 1,5 GB RAM, spojených 100MBit sieťou. Ako trénovacia množina bola použitá kolekcia Reuters–21578 [Lewis 1999], konkrétne ModApte rozdelenie na trénovaciu a testovaciu množinu [Apté a kol. 1994]. Kolekcia dát Reuters je štandardný voľne dostupný korpus textových dokumentov. Obsahuje 12,902 dokumentov. Každý dokument je novým článkom s určitou témou, väčšinou ekonomického charakteru. Pre účel nášho testovania boli vybrané dokumenty z roku 1987, ich počet bol 7769, počet kategórií 90.

Hlavným cieľom experimentov bolo potvrdiť, že navrhnutý systém distribúcie dokáže s použitím aj menej výkonných pracovných staníc efektívne znižovať nároky spojené so spracovaním obsiahlych kolekcií textových dokumentov. V experimentoch boli porovnávané výsledky distribuovanej verzie aj so sekvenčnou verziou algoritmu spustenou v rovnakom prostredí a čas potrebný na vybudovanie finálneho modelu dosiahol hodnoty v priemere 32,5 minút (jednotlivé služby boli spúšťané 3–krát, výsledné časy sú spriemerované).

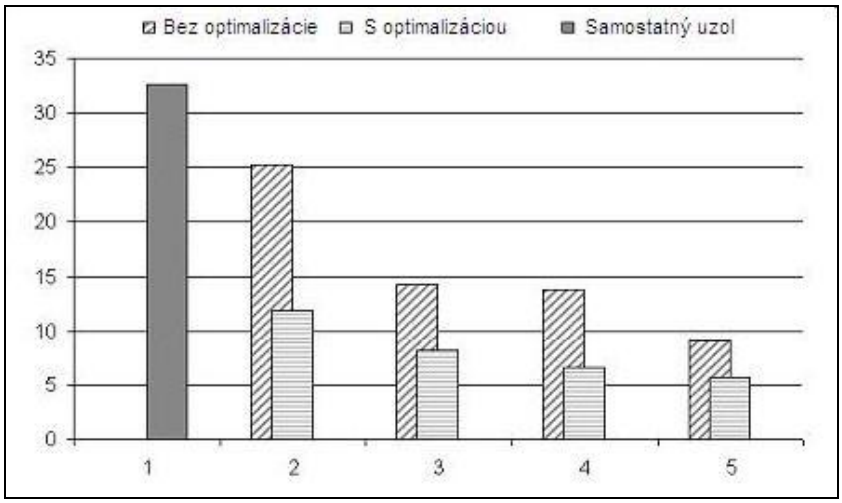
Prvá séria testov distribuovanej verzie prebehla bez použitia optimalizácie rozdelenia pracovného zaťaženia na jednotlivé uzly. V tomto prípade dostali pracovné uzly rovnaký počet „kategórií“ na spracovanie, teda vytvárali rovnaký počet čiastkových binárnych stromov. Výsledky zobrazuje Obr. 6.4. Detailné preštudovanie výsledkov tohto experimentu poukázalo na fakt, že celkový čas vybudovania výsledného modelu bol silne závislý na výkone prvého z pracovných uzlov. To bol dôsledok distribúcie frekvencií kategórií v dátovej kolekcii, inými slovami, uvedený pracovný uzol dostal v každom prípade spracovať kategórie s najvyšším počtom priradených dokumentov, tzn. najzložitejšie z čiastkových modelov (konkrétne kategória 14, obsahujúca 2780 dokumentov). Je to spôsobené nelineárnou distribúciou kategórií

v kolekcii (viď. Obr. 6.3), ktorý znázorňuje distribúciu frekvencií kategórií v dátovej množine Reuters.



Obr. 6.3 Logaritmická distribúcia frekvencií kategórií v dátovej množine Reuters

Po prvej sérii experimentov bola implementovaná jednoduchá optimalizácia prideľovania kategórií na pracovné uzly. Stratégia prideľovania je založená na zistení frekvencií výskytov dokumentov v jednotlivých kategóriách v trénovacej množine. Kategórie sa potom takýmto spôsobom usporiadajú a pri rozdeľovaní na jednotlivé pracovné uzly sa zároveň kontroluje aj počet dokumentov, ktoré patria do tých kategórií, pre ktoré sa na danom uzle počítajú čiastkové binárne klasifikátory. To znamená, že po jej aplikovaní pracovné uzly dostávali na spracovanie rovnaké množstvo kategórií, ale čo je podstatné aj zároveň s približne rovnakým rozložením dokumentov. Výsledky rovnakých experimentov s takto upravenou službou znázorňuje Obr. 6.4.

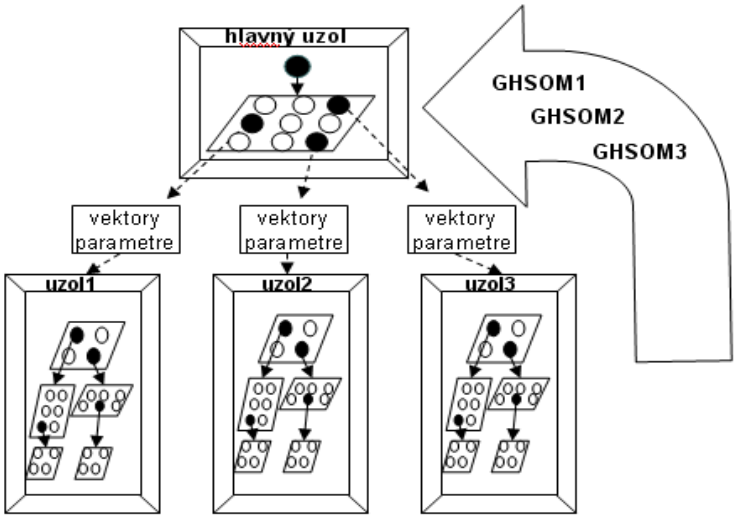


Obr. 6.4 Graf dosiahnutých časov výpočtu (v minútach, os y) na dátach Reuters pre rôzny počet použitých uzlov (os x) gridu použitím neoptimalizovanej a optimalizovanej distribúcie v porovnaní so sekvenčnou metódou

6.1.2 Distribuované zhlukovanie textových dokumentov

Proces zhlukovania dokumentov pomocou algoritmu GHSOM môže pri analýze veľkého množstva dokumentov predstavovať vážny problém z časového hľadiska. Z tohto hľadiska je algoritmus GHSOM ideálny kandidát pre paralelizáciu a distribúciu. Jeho princípy boli detailne popísané v časti 4.2.3.1. Na tomto mieste bude popísaný iba mechanizmus jeho distribúcie a paralelizácie. Pri zhlukovaní použitím algoritmu GHSOM vzniká po vytvorení mapy GSOM prvej úrovne niekoľko samostatných zhlukovacích procesov – budovanie hierarchických podstromov GHSOM pozostávajúcich z hierarchicky usporiadaných máp GSOM. Základnou myšlienkou návrhu je paralelné vykonávanie týchto zhlukovacích procesov [Sarnovský a kol. 2008].

- Na hlavnom uzle sa vyráta celková odchýlka vstupných dát, vytvorí sa mapa 1. úrovne a označia sa tie neuróny, ktoré spĺňajú podmienku expanzie do podmapy. Následne sa expanduje ten neurón, ktorého stredná kvadratická chyba je väčšia ako τ_u násobok strednej kvadratickej chyby mapy prvej úrovne a zároveň počet vektorov prislúchajúci danému neurónu je väčší ako definovaný minimálny počet vektorov nutných pre vytvorenie ďalšej podmapy. Z množiny vstupných vektorov sa následne vyberú vektory prislúchajúce neurónom, ktoré je potrebné expandovať. Tieto vektory sa rozpošlú na jednotlivé uzly gridu.
- Na uzloch gridu sa vektory stanú vstupom pre algoritmus GHSOM, ktorý vytvorí celý hierarchický podstrom. Keď sa na uzloch Gridu splní podmienka ukončenia algoritmu GHSOM, teda dosiahne sa definovaná maximálna hĺbka hierarchie alebo sa už nenašiel neurón, ktorý spĺňa podmienku expanzie, odošle sa celý čiastkový model GHSOM naspäť na hlavný uzol.
- Na hlavnom uzle sa prijaté čiastkové modely GHSOM spoja do jedného výsledného hierarchického modelu GHSOM.



Obr. 6.5 Schematické znázornenie algoritmu distribuovaného zhlukovania textových dokumentov

Použitú schému distribúcie znázorňuje Obr. 6.5. Vstupom prvej fázy je kolekcia dokumentov, na ktorých prebehnú presne tie isté fázy algoritmy predspracovania a vytvorenia vektorovej reprezentácie dokumentov za pomoci knižnice JBowl presne tak, ako pri experimentoch s distribuovanou kategorizáciou (viď. 6.1.1.1).

Normalizované vektory uložené v samostatnom súbore, ktorý tak obsahuje *tfidf* profily dokumentov, je potom základným vstupom pre blok distribúcia – spájanie. Zo vstupných vektorov dokumentov sa použitím tried knižnice JBowl vytvorí mapa rastúceho SOM (GSOM) prvej úrovne.

V ďalšom kroku sa vytvorí zoznam neurónov, ktoré spĺňajú podmienku expanzie do mapy nižšej úrovne. Následne sa zistí počet momentálne dostupných uzlov Gridu a vytvorí sa zoznam ich ukazovateľov. Počet uzlov Gridu je kľúčovým pri vytváraní fronty úloh na spracovanie uzlami gridu.

Každá úloha zhlukovania obsahuje identifikátor neurónu v rámci 1. mapy, zoznam (identifikátorov) vektorov namapovaných na daný neurón a parametre algoritmu GHSOM: *tau1* (τ_u), t*au2* (τ_m), minimálny počet vektorov potrebný pre vytvorenie ďalšej podmapy, maximálnu hĺbku hierarchie. Priradzovanie zhlukovacích úloh jednotlivým uzlom gridu prebieha nasledovne.

Nech n je počet expandovateľných neurónov a u nech je počet dostupných uzlov Gridu, potom :

- Ak $n \leq u$, do fronty úloh prvých *n* uzlov sa priradí práve jedna zhlukovacia úloha.
- Ak $n > u$, v prvej iterácii sa prvých *u* zhlukovacích úloh priradí prvým u uzlom, v nasledovných iteráciách sa priradzujú zvyšné úlohy zhlukovania zaradom na jednotlivé uzly dovtedy, kým už nie je čo priradzovať.

Po naplnení fronty úloh sa postupne z tejto fronty rozposielajú úlohy zhlukovania na jednotlivé uzly gridu. Na uzloch gridu sa zo vstupných vektorov vytvoria čiastkové hierarchické modely GHSOM použitím definovaných parametrov. Po vytvorení modelu sa tento model odošle späť na hlavný uzol, kde sa uloží a následne sa na ten istý uzol Gridu odošle nasledujúca úloha zhlukovania z fronty úloh (ak je front prázdna, výpočet na danom uzle končí).

Keď sa vyprázdnia všetky fronty úloh a sú prijaté všetky čiastkové hierarchické modely GHSOM, realizuje sa spájanie týchto modelov do výsledného hierarchického modelu. Spájanie modelov prebieha nasledovne: referencie na rodiča mapy uzla 1. úrovne čiastkového modelu sa nastavia tak, aby odkazovali na mapu 1. úrovne vytvorenej na začiatku procesu. Následne sa zmenia referencie na „potomkov“ mapy uzla 1. úrovne a to tak, aby ukazovali na mapy uzlov 1. úrovne čiastkových modelov. Po tejto zmene sa uloží výsledný model GHSOM ako perzistentný serializovaný Java objekt do súboru.

6.1.2.1 Popis experimentov s distribuovaným zhlukovaním textov na gride

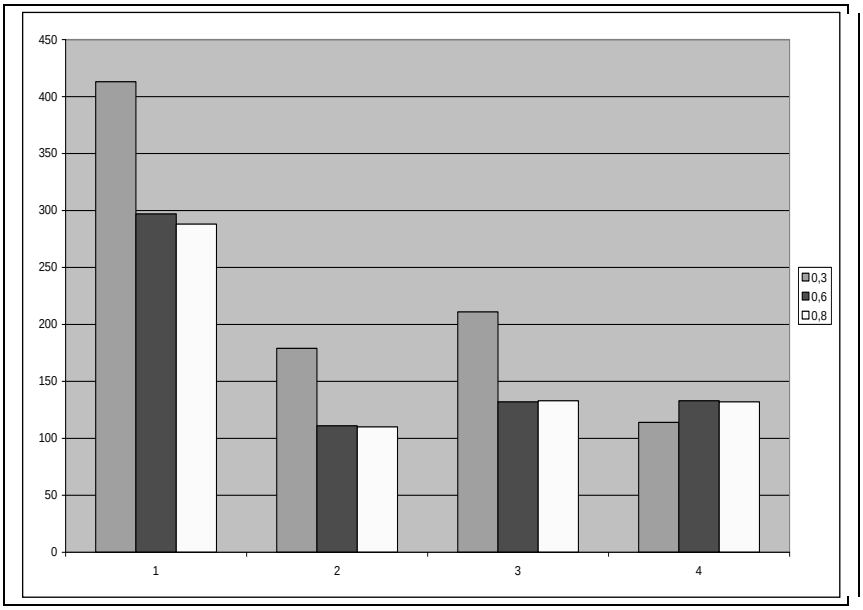
Cieľom experimentov bolo porovnať časovú náročnosť klasického, t.j. sekvenčného algoritmu GHSOM s jeho distribuovanou verziou. Experimenty sme podobne ako pri experimentoch s kategorizáciou realizovali v prostredí, ktoré nebolo izolované (na použité servery a pracovné stanice mal prístup prakticky ktokoľvek a kedykoľvek) a bežali na ňom aj iné služby. Kvôli získaniu čo najpresnejšieho času výpočtu sme každý experiment pre tie isté nastavenia parametrov algoritmu GHSOM opakovali trikrát. Výsledný čas bol určený ako priemer časov získaných pri týchto opakovaných experimentoch.

V niektorých prípadoch nastala situácia, že výsledný čas pre jedno opakovanie bol výrazne vyšší ako zvyšné dva časy, čo mohlo byť spôsobené náhlym vyťažením procesora inou službou servera resp. iným používateľom. Takýto čas sme potom nahradili priemerom zvyšných dvoch časov. V experimentoch sa menil počet uzlov a parameter *tau1*, ktorý riadi výslednú kvalitu zhlukov, čím ovplyvňuje veľkosť vytváraných máp GSOM a celkovú hĺbku hierarchie modelu GHSOM. Čím menší je parameter *tau1*, tým väčšie mapy sa vytvárajú a hierarchia je plytšia. A naopak, čím je tento parameter väčší, tým menšie sú mapy a výsledná hierarchia je hlbšia.

Distribuovaná verzia algoritmu GHSOM bola testovaná v testovacom gridovom prostredí pozostávajúceho zo servera (4 x UltraSPARC-III 750MHz, 8GB RAM), ktorý predstavoval hlavný uzol Gridu a zo šiestich pracovných staníc SUN rôznych konfigurácií. Uzly Gridu boli prepojené sieťou 100Mbit/s. Experimenty sme realizovali na dvoch kolekciách textových dokumentov: Times60 (420 dokumentov) a Reuters–21578 (12 902 dokumentov), ktorá bola popísaná vyššie v časti 6.1.1.1.

Times60 je kolekcia dokumentov – článkov magazínu Times zo šesťdesiatych rokov minulého storočia. Obsahuje 420 dokumentov. Články sa týkajú medzinárodných vzťahov, ekonomickej situácie a histórie krajín ako Rusko, Veľká Británia, Francúzsko, Malajzia, Egypt a Sýria. Hranica minimálnej dokumentovej frekvencie termov bola nastavená na 10 a maximálna na 100. Tomuto kritériu vyhovovalo 1925 termov. Pri textovej kolekcii Reuters sme nastavili hranicu minimálnej dokumentovej frekvencie termov na 20 a maximálnej na 100. Veľkosť vstupných vektorov dokumentov potom bola 1362.

Experimenty na dátach Times60 na výpočtovom gride sme vykonali s nastaveniami parametra *tau1* = 0.3, resp. 0.6 a 0.8. Najskôr sme testovali sekvenčnú verziu algoritmu na jednom uzle Gridu. S nastavením parametra *tau1* = 0.3 sme potom testovali distribuovanú verziu postupne pre 2, 3, 4, 5 a 6 uzlov gridu (z mapy 1. úrovne sa expandovalo 12 zhlukov). Pre parametre *tau1* 0.6 a 0.8 sme testovali na 2, 3 a 4 uzloch Gridu, keďže z mapy 1. úrovne sa v tomto prípade expandovali iba 4 neuróny.

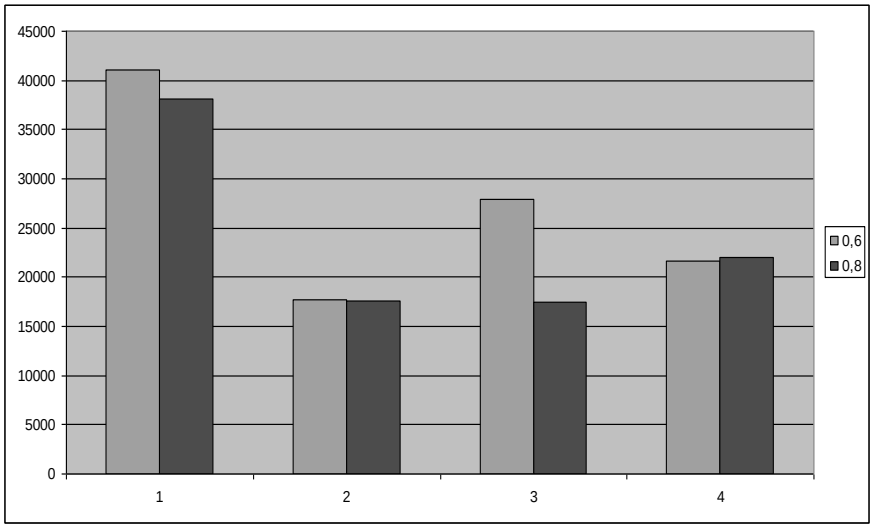


Obr. 6.6 Graf dosiahnutých časov výpočtu (v sekundách, os y) na dátach Times60 pre rôzny počet použitých uzlov (os x) Gridu pri rôznych hodnotách parametra *tau1* (viď. legenda)

Časové zefektívnenie algoritmu s parametrom *tau1* 0.3 predstavovalo pri výpočte na dvoch uzloch 59%, na troch uzloch 50%, na štyroch uzloch 73%, na piatich uzloch 70% a na šiestich uzloch 66%. Časové zefektívnenie algoritmu s parametrom *tau1* 0.3 a použitím optimalizácie rozdelenia pracovného zaťaženia (použitý rovnaký postup, ako v prípade rozhodovacích stromov) predstavovalo pri výpočte na dvoch uzloch 66%, na troch uzloch 73%, na štyroch uzloch 76%, na piatich uzloch 71% a na šiestich uzloch 76%. Časové zefektívnenie algoritmu s parametrom *tau1* 0.6 predstavovalo pri výpočte na dvoch uzloch 63%, na troch uzloch 55% a na štyroch uzloch 55%. Časové zefektívnenie algoritmu s parametrom *tau1* 0.6 a použitím optimalizácie predstavovalo pri výpočte na dvoch uzloch 63%, na troch uzloch 67% a na štyroch uzloch 56%. Grafické znázornenie výsledkov pre optimalizovanú verziu algoritmu pre 1 až 4 uzly gridu ukazuje Obr. 6.6.

Pri experimentoch na kolekcii Reuters sme použili hodnoty parametra *tau1* 0.6 a 0.8. Obdobne ako pri testoch na dátach Times60 sme najskôr testovali sekvenčnú verziu algoritmu na jednom uzle a následne distribuovanú verziu postupne na dvoch, troch a štyroch uzloch Gridu.

Časové zefektívnenie algoritmu s parametrom *tau1* 0.6 predstavovalo pri výpočte na dvoch uzloch 56%, na troch uzloch 58% a na štyroch uzloch 49%. Časové zefektívnenie algoritmu s parametrom *tau1* 0.6 a použitím optimalizácie predstavovalo pri výpočte na dvoch uzloch 57%, na troch uzloch 55% a na štyroch uzloch 47%. Časové zefektívnenie algoritmu s parametrom *tau1* 0.8 predstavovalo pri výpočte na dvoch uzloch 53%, na troch uzloch 55% a na štyroch uzloch 43%. Časové zefektívnenie algoritmu s parametrom *tau1* 0.6 a použitím optimalizácie predstavovalo pri výpočte na dvoch uzloch 57%, na troch uzloch 59% a na štyroch uzloch 42%. Grafické znázornenie výsledkov na dátach Reuters ukazuje Obr. 6.7.



Obr. 6.7 Graf dosiahnutých časov výpočtu (v sekundách, os y) na dátach Reuters pre rôzny počet použitých uzlov (os x) Gridu pri rôznych hodnotách parametra *tau1* (viď. legenda)

Výsledky ukazujú celkové urýchlenie algoritmu GHSOM pri jeho distribuovanej verzii, ako aj to, že urýchlenie nebolo vždy plynulé, t.j. nie vždy vyšší počet uzlov znamenal urýchlenie oproti nižšiemu počtu uzlov. Jednou z príčin je nerovnomerné rozdelenie dát na jednotlivé uzly a tiež veľkosť chyby, ktorú daný uzol znižoval. Ďalšou príčinou bol rôzny výpočtový výkon jednotlivých uzlov testovacieho Gridu (frekvencia procesorov na prvých dvoch uzloch bola o 500MHz vyššia ako u zvyšných uzlov).

K nerovnomernej časovej závislosti mohla prispieť aj náhodná inicializácia váh neurónov. Jednoduchá optimalizácia, ktorú sme navrhli za účelom rovnomerného zaťaženia uzlov, neprinášala vždy zlepšenie oproti neoptimalizovanej distribúcii. Ďalšou možnosťou ako rovnomernejšie a efektívnejšie využívať uzly Gridu, by mohla byť taká distribúcia algoritmu GHSOM, pri ktorej by sa na uzloch Gridu vytvárali iba mapy GSOM a nie celé podstromy GHSOM. Takýto spôsob by síce zvýšil komunikačné nároky a tým aj nároky na priepustnosť počítačovej siete, ale zároveň by umožnil využiť výpočtový výkon uzlov, ktoré po ukončení výpočtu mrhajú svojím výpočtovým časom pri čakaní na pomalšie uzly.

Kritickým miestom pri distribúcii algoritmu GHSOM je vytváranie mapy 1. úrovne. Tento proces môže pri určitých dátach a nastaveniach algoritmu predstavovať viac ako 50% celkového času výpočtu. Kombinácia paralelizácie vytvárania mapy GSOM 1. úrovne (napr. na počítačovom klastri) a následná distribúcia zhlukov tejto mapy na uzly Gridu by mohla priniesť veľmi zaujímavé výsledky z hľadiska celkového urýchlenia algoritmu.

7 Použitá literatúra

ACE. 2004: *Annotation Guideline for Entity Detection and Tracking.* Version 4.2.6 200400401, 2004. Dostupné na internete: http://projects.ldc.upenn .edu/ ace/docs/EnglishEDTV4-2-6.pdf

Androutsopoulos I., Koutsias J., Chandrinos J. K., Spyropoulos C. D. 2000. *An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages*. Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, 160-167, 2000.

Apté C., Damerau F., Weiss S. M. 1994. *Towards language independent automated learning of text categorisation models*. In Research and Development in Information Retrieval, pages 23–30, 1994.

Appelt D. E., Israel D. J. 1999. *Introduction to Information Extraction Technology.* A Tutorial Prepared for IJCAI-99, 1999. Dostupné na internete: https://user.phil-fak.uni-duesseldorf.de/~rumpf/SS2005/ Informationsextraktion/ Pub/AppIsr99 .pdf

Baker L. D., McCallum A. K. 1998. *Distributional clustering of words for text classification*. In: Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval, 1998, s. 96-103

Bednár P. 2004. *Automatická klasifikácia textov na základe ich obsahu*. Písomná práca k dizertačnej skúške. Katedra kybernetiky a umelej inteligencie FEI, Technická univerzita v Košiciach, Košice, 2004.

Berkhin P. 2002. *Survey of Clustering Data Mining Techniques*. Accrue Software, Inc, 2002.

Besancon R., Rajman M., Chappelier J. C. 1999. *Textual Simmilarities based on Distributional Approach*. In: Proceedings of the Tenth International Workshop on Database and Expert Systems Applications (DEXA99), Firenze (Italy), 1999, s. 180-184.

Carreras X., Màrquez L. 2001. *Boosting trees for anti-spam email filtering*. Proceedings of RANLP-01, 4th International Conference on Recent Advances in Natural Language Processing. 2001

Clark P., Niblett T. 1989. *The CN2 induction algorithm*. Machine Learning Journal, 3(4): 261-283, 1989.

Cooper W. S. 1969. *Is interindexer consistency a hobgoblin?* American documentation, 20: 268-278, 1969.

Cucerzan S., Yarowsky D.d. 1999. *Language Independent Named Entity Recognition Combining Morphological and Contextual Evidence*. In: Proceedings of the Joint SIGDAT Conference on EMNLP and VLC, 1999, s. 90-99.

Cunningham H. 2004. *Information Extraction, Automatic*. Dept. of Computer Science, University of Sheffield, 2004.

Deerwester S., Dumais S. T., Landauer T. K., Furnas G. W., Harshman R. A. 1990. *Indexing by latent semantic analysis*. In: Journal of the American Society for Information Science, 41(6), 1990, s. 391-407.

Ding X., Liu B., Yu P. S. 2008. *A holistic lexicon-based approach to opinion mining*. In WSDM '08: Proceedings of the international conference on Web search and web data mining. New York, NY, USA: ACM, 2008, pp. 231–240, 2008.

Dittenbach M., Rauber A., Merkl D. 2000. *Using Growing Hierarchical Self-Organizing Map for document classification*. In Proceedings of European Symposium on Artificial Neural Networks (ESANN 2000), str. 7-12, Bruges, Belgicko, 2000.

Dumais S. 1991. *Improving the retrieval of information from external sources*. Behavior Research Methods, Instruments, and Computers, 23(2):229–236, 1991.

Farrow J. F. 1991. *A cognitive process model of document indexing*. Journal of Documentation, 47(2): 149-166.

Feldman R., Sanger J. 2007. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2007.

Forróová M., Garabík R., Gianitsová L., Horák A., Šimková M. 2003. *Tokenizácia, lematizácia a morfologická anotácia SNK.* Seminár „Návrh morfologického tagsetu Slovenského národného korpusu". JÚĽŠ SAV, 2003. Dostupné na internete: http://korpus.juls.savba.sk/archive/seminare/2003-11-10/tagset.pdf

Foster I., Kesselman C. 1999. *Computational Grids*, The Grid – Blueprint for a new Computing Infrastructure, Morgan Kaufmann, 1999.

Foster I., Kesselman, C. 2004. *The Grid 2*, Blueprint for a New Computing Intrastructure, Morgan Kaufmann, 2004.

Fritzke B. 1995. *Growing Grid – a self-organizing network with constant neighbourhood and adaptive strength*. Neural Processing Letters, 2(5), 1995.

Fuhr N. 1992. *Probabilistic models in information retrieval*. The Computer Journal, 35(3):243–255, 1992.

Furdík K. 1998. *Automatické určovanie obsahu odborného textu*. In: Varia VII. Zborník materiálov zo siedmeho kolokvia mladých jazykovedcov (Modra-Piesok, 3. - 5. december 1997). Ed. M. Nábělková. Bratislava, SJS pri SAV, 1998, s. 52-63.

Furdík K. 2001. *Identifikácia paradigmatických a syntagmatických vztahov v texte*. In: Slovenčina a čeština v počítačovom spracovaní, Ed. A.Jarošová. Bratislava: VEDA, 2001, s. 34-49.

Furdík K. 2003. *Získavanie informácií v prirodzenom jazyku s použitím hypertextových štruktúr*. Doktorandská dizertačná práca. Technická univerzita v Košiciach, FEI, KKUI, Košice, 2003. Dostupné na internete: http://web.tuke .sk/fei-cit/furdik/publik/Dizertacia_furdik_2003.pdf

Furdík K. 2008. *Algoritmy predspracovania textu pre úlohy klasifikácie a zhlukovania v systéme elektronickej výučby*: XVII. Kolokvium mladých jazykovedcov. [on-line] [cit.: 21.1.2008] Dostupné na internete: http://web.tuke.sk/fei-cit/furdik/publik/Kolokvium07_furdik_2008_PoZnaT .pdf

Furdík F., Paralič J., Babič F., Butka P., Bednár P. 2010. Design and Evaluation of a Web System Supporting Various Text Mining Tasks for the Purposes of Education and Research. Acta Electrotechnica et Informatica, Vol. 10, No. 1 (2010), pp. 51-58. ISSN 1335-8243

Galamboš L. 2001. *Lemmatizer for Document Information Retrieval Systems in JAVA*. In: SOFSEM 2001: Theory and Practice of Informatics. Proceedings. LNCS 2234, Springer Berlin / Heidelberg, 2001.

Garabík R., Gianitsová L., Horák A., Šimková M. 2004. *Tokenizácia, lematizácia a morfologická anotácia Slovenského národného korpusu*. Slovenský národný korpus, interný materiál. Bratislava, 2004. Dostupné na internete: http://korpus.juls.savba.sk/publications/block2/2004-garabik-gianitsova-horak-simkova-tokenizacia/2004-garabik-gianitsova-horak-simkova-tokenizacia.pdf

Garabík R. 2007. *Slovak morphology analyzer based on Levenshtein edit operations*. In: 1st Workshop on Intelligent and Knowledge oriented Technologies. Proceedings of the WIKT'06 conference. Ed.: Laclavík M., Budinská I., Hluchý L. UI SAV, Bratislava, 2007, s. 2–5.

Goldszmidt M., Sahami M. 1998. *A probabilistic approach to full-text document clustering*. Technical Report ITAD-433/MS/98/044, SRI International, 1998.

Habala O., Paralič M., Bartaloš P., Rozinajová V. 2009. Semantically-aided Data-aware Service Workflow Composition, 35th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2009, Springer, LNCS 5404, pp 317-328

Hajič J. 2001. *Statistické modelování a automatická analýza přirozeného jazyka (morfologie, syntax, překlad)*. In: Slovenčina a čeština v počítačovom spracovaní, Ed. A. Jarošová. Bratislava: VEDA, 2001, s.11-33.

Harris Z. S. 1954. *Distributional Structure*. In: Word 10 (2/3), 1954, s. 146–62.

Hatcher E., Gospodnetić O. 2004. *Lucene in Action*. Manning Publications Co., 2004. 456 s., ISBN: 1932394281

Hatzivassiloglou V., McKeown K. R. 1997. *Predicting the semantic orientation of adjectives*. In proceedings of the eighth conference on European

chapter of the Association for Computational Linguistics. Morristown, NJ, USA: Association for Computational Linguistics, 1997, pp. 174–181.

Hatzivassiloglou V., Wiebe J. 2000. *Effects of adjective orientation and gradability on sentence subjectivity*. 2000. [on-line]. Dostupné na internete: http://citeseer.ist. psu.edu/hatzivassiloglou00effects.html

Hayes P. J., Weinstein S. P. 1990. *Construe/Tis: a system for content-based indexing of a database of news stories*. Proceedings of IAAI-90, 2nd conference on innovative applications of artificial intelligence, 49-66, 1990.

Hearst M. A. 1999. *Untangling text data mining*, Proc. of ACL'99: the 37th annual meeting of the association for computational linguistics, University of Maryland, p. 3-10, June 1999.

IANA–ChS. 2007. *Character Sets. IANA Assigment*. [on-line] [cit.: 4.9.2008], 2007. Dostupné na internete: http://www.iana.org/assignments/ character-sets

IANA–MIME. 2007. *The Internet Corporation for Assigned Names and Numbers: IANA MIME Media Types*. [on-line] [cit.: 4.9.2008], 2007. Dostupné na internete: http://www.iana.org/assignments/media-types/

Jain A. K., Murty M. N., Flynn P.J. 1999. *Data Clustering: A Review*. In ACM Computing Surveys, Vol. 31, No. 3, September, 1999.

Jančiak I., Sarnovský M., Brezany P. 2006a. *Text Mining Within the Grid Miner Framework*, 2nd Dialogue Workshop, Edinburgh, 2006.

Jančiak I., Sarnovský M., Tjoa A Min, Brezany P. 2006b. *Distributed classification of textual documents on the Grid*. In: High Performance Computing and Communications : Second international conference, HPCC 2006, Munich, Germany, September 13-15, 2006 : Proceedings. Berlin : Springer, 2006. p. 710-718. ISBN 3-540-39368-4

Java–Intl. 2008: *Sun Developer Network: Java Internationalization*. [on-line] [cit.: 4.9.2008], 2008. Dostupné na internete: http://java.sun.com/javase/ technologies/core/basic/intl/.

Kim S.-M., Hovy E. 2004. *Determining the sentiment of opinions*. In COLING '04: Proceedings of the 20th international conference on Computational Linguistics. Morristown, NJ, USA: Association for Computational Linguistics, 2004, p. 1367.

Kohonen T. 1995. *Self-organizing maps*. Springer-Verlag, Berlín, 1995.

Koštiaľ I. 2002. *Spracovanie a kategorizácia textových dokumentov pre účely semiautomatického linkovania na znalostný model*. Diplomová práca. FEI KKUI, Technická univerzita v Košiciach. Košice, 2002.

Kroeze J. H., Matthee M. C., Bothma T. JD. 2003. *Differentiating Data - and Text-Mining Terminology*, ACM Digital Library, Proc. of SAICSIT 2003, p. 93–101.

Krajči S., Laclavík M., Novotný R., Turlíková L. 2009. *The tool Morphonary/ Tvaroslovnik: Using of word lemmatization in processing of documents in Slovak*. In: P. Návrat, D. Chudá (eds.), Proceedings of the 8th annual conference Znalosti (Knowledge) 2009, Brno, 4.-6. február 2009. Vydavateľstvo STU, Bratislava, 2009, s. 119-130.

Laclavík M., Šeleng M, Ciglan M. 2007. Vyhľadávanie informácií. Učebný text, máj 2007. [cit.: 30.4.2010] Dostupné na internete: http://ikt.ui.sav.sk/vi/vi_laclavik.pdf

Laclavík M., Ciglan M., Krajčí S., Hluchý L., Furdík K. 2007. *Dostupné zdroje a výzvy pre počítačové spracovanie informačných zdrojov v slovenskom jazyku*. In: WIKT 2006 Proceedings of the 1st Workshop on Intelligent and Knowledge oriented Technologies. Eds. M. Laclavík, I. Budinská, L. Hluchý. Ústav informatiky SAV, Bratislava, 2007, s. 92-98.

Luhn H. 1953. *A new method of recording and searching information*. In: American Documentation, 4/1953, s. 14-16.

Larkey L. S., Croft W. B. 1996. *Combining classifiers in text categorization*. Proceedings of SIGIR-96, 19th ACM International conference on research and development in informationa retriaval, Zürich, 289-297, 1996.

Larkey L. S. 1999. *A patent search and classification system*. Proceedings of DL-99, 4th ACM conference on digital libraries, Berkley, 179-187, 1999.

Lewis D. D. 1999. *Reuters-21578 text categorization test collection distribution 1.0*. 1999. Dostupné na internete: http://www.research.att .com/lewis

Liu B. 2008. *Opinion mining*. Invited contribution to Encyclopedia of Database Systems, to complete in July, 2008.

Machová K. 2002. *Strojové učenie: princípy a algoritmy.* Technická univerzita v Košiciach, Košice, 2002, 117 s., ISBN 80-89066-51-8

Machová K. 2009. Strojové učenie v systémoch spracovania informácií. Technická univerzita v Košiciach, 2009, 85 s., ISBN 978-80-8086-130-8

Manning Ch. D., Raghavan P., Schütze H. 2008. *Introduction to Information Retrieval*. Cambridge University Press, 2008, 482 s.

Maron M. 1961. *Automatic indexing: an experimental inquiry*. Journal of the ACM, 8(3): 404-417, 1961.

Masson M. E. 1982. *Cognitive process in skimming stories*. Journal of Experimental Psychology: Learning, Memory and Cognition, 8:400-417.

Meadow C. 1992. *Text information retrieval systems*. Academic Press Inc., San Diego, California, 1992.

Mitchell T. M. 1996. *Machine Learning*. McGraw Hill, New Your, 1996.

Moens M. F. 2006. *Information Extraction: Algorithms and Prospects in a Retrieval Context* (The Information Retrieval Series); Springer: 2006.

Ng H. T., Goh W. B., Low, K. L. 1997. *Feature selection, perceptron learning, and a usability case study for text categorization*. Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval, 67-73, 1997.

O'Brien G. W. 1994. *Information management tools for updating an SVD-encoded indexing scheme*. Technical Report UT-CS-94-258, 1994.

Páleš E. 1994. *SAPFO – Parafrázovač slovenčiny*. 1. vyd. Bratislava: VEDA, 1994, 305 s.

Paralič J. 2003. Objavovanie znalostí v databázach. Elfa, Košice, 2003, 80 s. ISBN 80-89066-60-7

Paralič J. 2008. *Manažment znalostí (3-4)*. Prezentácie k prednáškam. [on-line] [cit.: 4.9.2008] Dostupné na internete http://people.tuke.sk/jan. paralic/prezentacie/MZ/MZ3-4.pdf

Paralič J., Bednár P. 2003. Text Mining for Documents Annotation and Ontology Support. A book chapter in: "Intelligent Systems at the Service of Mankind" (W. Elmenreich, T. Machado, I.J. Rudas eds.), Ubooks, Germany, November 2003, pp. 237-248, ISBN 3-935798-25-3

Paralič J., Kostiaľ I. 2003. A Document Retrieval Method Based on Ontology Associations. In Journal of Information and Organizational Sciences, Varaždin, Croatia, Vol. 27 (2003), Nr. 2, pp. 93-100, ISSN 0351-804

Porter M. F. 1980. *An algorithm for suffix stripping*. In: V. Program 14(3), 1980, s.130-137.

Quinlan J. R. 1996. *Learning first-order definitions of functions*. Journal of Artificial Intelligence Research, 5: 139-161, 1996.

Rauber A. 1999. *On the labeling of self-organizing maps*. In Proceedings of IJCNN, Washington, 1999.

van Rijsbergen C. J. 1979. *Information retrieval*. Butterworths, London, 1979.

Rosenblatt F. 1988. *The perceptron: A probabilistic model for information storage and organization in the brain*. Neurocomputing, MIT Press, Cambridge, 1988.

Ruiz M. E., Srinivasan, P. 2002. *Hierarchical neural networks for text categorization*. *Informational retrieval*, 5: 87-118, 2002.

Rychlý P. 2000. *Korpusové manažery a jejich efektivní implementace*. Dizertačná práca. Fakulta informatiky, Masarykova universita v Brne, 2000.

Řezanková H., Húsek D., Snášel V. 2007. *Shluková analýza dat*. Professional Publishing, Praha, 196 s, 2007.

Salton G., Wong A., Yang C. S. 1975. *A vector space model for automatic indexing*. In: Communications of the ACM, Vol. 18, Issue 11, 1975, s. 613-620.

Salton G., Buckley C. 1988. *Term weighting approaches in automatic text retrieval.* In: Information Processing and Management, 24(5), 1988, s. 513-523.

Sarawagi S. 2007. *Information extraction: Foundations and Trends in Databases*, vol. 1, no. 3, pp. 261-377, 2007.

Sarnovský M. 2009. Objavovanie znalostí v textoch za pomoci gridového počítania. Dizertačná práca, Katedra kybernetiky a umelej inteligencie, Technická univerzita v Košiciach, marec 2009, 110 s.

Sarnovský M., Butka P., Paralič J. 2009. Grid-based Support for Different Text Mining Tasks. Acta Polytechnica Hungarica, Journal of Applied Sciences, Vol. 6, Nr. 4, 2009, ISSN 1785-8860, s. 5-27

Sarnovský M., Butka P., Safko V. 2008. *Distribuované zhlukovanie textových dokumentov v prostredí Gridu*. In: Znalosti 2008, Bratislava, Slovakia, pp.192-203, ISBN 978-80-227-2827-0.

Schwarz J. 2003. *Současný stav a trendy automatické indexace dokumentů*: Propojení analytických záznamů s plnými texty. 2003. [on-line] [cit.: 21.1.2008] Dostupné na internete: http://full.nkp.cz/nkdb/docs/studie/MAIobsah.html

Sebastiani F. 2002. *Machine Learning in Automated Text Categorization.* Consiglio Nazionale delle Ricerche. Italy. 2002. Dostupné na internete: http://www.isti.cnr.it/People/F.Sebastiani/Publications/ACMCS02.pdf

Shapire R., Singer Y., Singhal A. 1998. *Boosting and Rocchio applied to text filtering*. Proceedings of SIGIR-98, 21st ACM international conference on research and development in informational retrieval, 215-223, 1998.

Shapire R., Singer Y. 1999. *Improved boosting algorithms using confidence-rated predictions*. Machine Learning, 37(3): 297-336, 1999.

Slawski B. 2008. *Google Stopwords Patent*. SEO by the Sea, Internet Marketing and Search Engine Optimization Services, Consulting, and Research, 2008. Dostupné na internete: http://www.seobythesea.com/?p=1109

Sparck-Jones K., Willett P. 1997. *Readings in Information Retrieval*. Academic Press/Morgan Kaufmann, 1997.

Sullivan D. 2001. Document Warehousing and Text Mining: Techniques for Improving Business Operations, Marketing and Sales. John Willey & Sons, Inc. 2001.

Tarr D., Borko H. 1974. *Factors influencing inter-indexing consistency*. Proceedings of the American Society for Information Science 37th Annual Meeting, 50-55, 1974.

Turney P. D. 2001. *Thumbs up or thumbs down*?: semantic orientation applied to unsupervised classification of reviews," in ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. Morristown, NJ, USA: Association for Computational Linguistics, pp. 417–424, 2001

Unicode 5.1. 2008. *Inc.: Unicode Standard, Version 5.1.0.* [on-line] [cit.: 2008-09-04], 2008. Dostupné na internete: http://www.unicode.org/ versions/Unicode5.1.0/

Wiebe J. M. 1994. *Tracking point of view in narrative*. Computational Linguistics, vol. 20, pp. 233–287, 1994.

Wilson D. R., Martinez T. R. 2000. *Reduction techniques for Instance-Based Learning Algorithms*. Machine Learning, 38(3): 257-286, 2000.

Yang Y., Pedersen J. O. 1997. *A comparative study on feature selection in text categorization*. In: Proc. of the 14th International Conference on Machine Learning ICML97, 1997, s. 412-420.

Yang Y., Ault T., Pierce T., Lattimer C. W. 2000. *Improving text categorization methods for event tracking*. Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'00), 65-72, 2000.

Yang Y., Slattery S., Ghani R. 2002. *A study of approaches to hypertext categorization.* Journal of Intelligent Information Systems, 18(2-3): 219-241, 2002.

Zeman D. 2009. *Korpusy a první úkoly*. Podklady k přednáškám „Počítačové zpracování přirozeného jazyka" (MFF UK) a „Počítače a přirozený jazyk" (FJFI ČVUT). 2009. Dostupné na internete: http://ufal.mff. cuni.cz/~zeman/vyuka/ podklady/pzpj03-prvni_ukoly.ppt

Časť B (praktická časť)

B.1 Základný popis architektúry a funkčnosti systému JBowl

Systém JBowl (*Java Bag-of-words Library*) je softvérová knižnica implementovaná v jazyku Java, ktorá poskytuje objektový model a rozhrania (API) pre vytváranie aplikácií získavania znalostí a dolovania v textoch. Knižnica JBowl obsahuje prostriedky na správu, indexáciu a manipuláciu s textovými dokumentmi, predovšetkým na komplexnú štatistickú analýzu textu vrátane podpory spracovania prirodzeného jazyka. Súčasťou knižnice sú implementácie viacerých algoritmov kontrolovaného a nekontrolovaného strojového učenia s voliteľnými vstupnými parametrami a metódami na vyhodnocovanie kvality modelov dolovania v textoch.

B.1.1 Základné údaje

Umiestnenie: http://sourceforge.net/projects/jbowl/

Aktuálna verzia (november 2009): 0.9_08 (jbowl-0.9_08.zip, 8.9 MB)

Platforma: Java JDK/JRE 1.6

Odporúčané vývojové prostredie: NetBeans ver. 6.7.1

Závislosti, použité komponenty:

- JSR 173: jsr173_1.0_api.jar. API pre segmentáciu (parsing) XML formátu,
- Jakarta Lucene: lucene-core-2.0.0.jar. Indexovací a vyhľadávací stroj.
- sg-cdb-1.0.4.jar. API pre konštantné databázy [StrangeGismo].
- sjsxp.jar. Parser prúdu dát v XML formáte od Sun Microsystems.

Licencia:

- FSF Lesser GNU General Public License, ver. 2.1 (február 1999). Text licencie je súčasťou distribúcie.
- SVM solver (trieda org.jbowl.model.supervised.svm.Solver) je implementovaný na základe libsvm knižnice [LIBSVM], poskytovanej pod licenciou: Copyright (c) 2000-2005 Chih-Chung Chang and Chih-Jen Lin. All rights reserved.

B.1.2 Vznik a história systému JBowl

Systém JBowl začal vznikať od roku 2003 na pôde Katedry kybernetiky a umelej inteligencie FEI TU v Košiciach. V tom čase existovalo v rámci katedry viacero návrhov a čiastkových implementácií rôznych algoritmov na predspracovanie, analýzu a získavanie znalostí z textových dokumentov. Snaha o funkčnú integráciu týchto návrhov a parciálnych riešení do jedného efektívneho a dostatočne flexibilného systému, založeného na progresívnej a na katedre najpoužívanejšej implementačno-vývojovej platforme Java, vyústila do návrhu a postupnej realizácie samostatného softvérového

systému na podporu inteligentného získavania znalostí z textov v prirodzenom jazyku. V priebehu dvoch rokov sa postupne podarilo vytvoriť systém JBowl, ktorý sa úspešne uplatnil v mnohých projektoch, praktických aplikáciách, i vo výučbe a v záverečných prácach študentov.

S cieľom zvýšiť implementačný a aplikačný potenciál bol v druhej polovici roku 2005 systém JBowl zaregistrovaný vo webovom úložisku Open source projektov SourceForge.net[25]. Ťažisko vývoja systému sa po roku 2005 presunulo do Centra pre informačné technológie (CIT), spoločného pracoviska Ústavu informatiky SAV a Technickej univerzity v Košiciach, kde kontinuálne pokračuje dodnes.

Vývoj systému JBowl bol motivovaný snahou navrhnúť a implementovať ucelený, dostatočne flexibilný a rozšíriteľný systém, ktorý by na jednom mieste združoval algoritmy a technológie potrebné pre oblasti skúmané v rámci CIT. Sú to predovšetkým manažment a reprezentácia znalostí, dolovanie a objavovanie znalostí v textoch, vyhľadávanie a extrakcia informácií, sémantický web a sémantické technológie vo všeobecnosti. Vo všetkých týchto oblastiach je primárnym zdrojom údajov písaný text, organizovaný do potenciálne rozsiahlej štruktúry súborov elektronických textových dokumentov. Z toho vyplynuli všeobecné kritériá pre budovanie systému JBowl, ktorými sú jednoduchá rozšíriteľnosť a modulárna konštrukcia vnútorných modulov na predspracovanie, jazykovú analýzu, indexáciu a ďalšiu analýzu veľkých textových súborov. Následne boli definované konkrétne požiadavky, ktoré má systém pre tieto účely spĺňať [Bednár, Butka 2005]:

- Schopnosť efektívne predspracovávať rozsiahle kolekcie textových dokumentov pomocou flexibilnej množiny dostupných techník predspracovania, adaptabilných na rôzne typy a formáty textu (napr. čistý text, HTML alebo XML).
- Schopnosť spracovávať súbory textov v rôznych jazykoch, predovšetkým v angličtine a v slovenčine. Zohľadnenie skutočnosti, že rôzne jazyky vyžadujú rozdielne prístupy vo fáze predspracovania a jazykovej analýzy.
- Podpora pre indexáciu a vyhľadávanie v rozsiahlych súboroch textových dokumentov. Možnosť využitia na experimenty s rôznymi vyhľadávacími technikami.
- Návrh a implementácia škálovateľného súboru najpoužívanejších klasifikačných algoritmov s voliteľnými nastaveniami parametrov, podpora rôznych metrík vyhodnocovania kvality klasifikácie.
- Návrh a implementácia súboru algoritmov nekontrolovaného strojového učenia, zhlukovania textových dokumentov a doplnkových algoritmov popisu zhlukov pomocou charakteristických termov.
- Dobre navrhnuté rozhranie pre znalostné štruktúry, akými sú napríklad ontológie, kontrolované slovníky, alebo databáza WordNet.

[25] Registrácia projektu JBowl na SourceForge.net má dátum 7. 9. 2005.

Samotnej implementácii systému JBowl predchádzala detailná analýza existujúcich Open source nástrojov s funkcionalitou blízkou formulovaným požiadavkám. Boli identifikované štyri skupiny nástrojov, a to:

- Nástroje pre indexáciu a vyhľadávacie mechanizmy:
  - Jakarta Lucene, http://lucene.apache.org.
- Nástroje pre spracovanie textu:
  - GATE - General Architecture for Text Engineering, http://gate.ac.uk,
  - JavaNLP, http://nlp.stanford.edu/javanlp/.
- Nástroje pre podporu objavovania znalostí v databázach:
  - WEKA, http://www.cs.waikato.ac.nz/ml/weka/,
  - KDD package, http://people.tuke.sk/jan.paralic/KDD/ [Bednár, Paralič 2003],
  - JDM API - Java Data Mining Package, http://www.jdmp.org.
- Nástroje pre prácu s ontológiami:
  - KAON - Karlsruhe ontology, http://kaon.semanticweb.org.

Ako je ukázané v [Bednár, Butka 2005], každá z týchto skupín pokrýva iba jednu, nanajvýš dve z deklarovnaých požiadaviek. Pre požadovanú úroveň podpory dolovania v textoch a sémantického vyhľadávania bolo teda potrebné vyvinúť knižnicu JBowl ako modulárny, flexibilný a rozšíriteľný rámec s ľahko pochopiteľnou vnútornou štruktúrou, poskytujúci mechanizmy pre predspracovanie, jazykovú analýzu a indexáciu rozsiahlych kolekcií textových dokumentov, a tiež pre tvorbu, testovanie a vyhodnocovanie modelov dolovania v textoch algoritmami kontrolovaného aj nekontrolovaného učenia.

Vývoj a implementácia systému JBowl podľa definovaných požiadaviek prebieha na pôde Centra pre informačné technológie TU v Košiciach kontinuálne od roku 2005 dodnes. Priebežne, na základe testovania a nasadenia systému v rôznych pilotných aplikáciách, boli špecifikované dodatočné požiadavky, napríklad na podporu štandardizovaného úložiska textových dokumentov, na zvýšenie efektívnosti klasifikačných a zhlukovacích algoritmov, na možnosť paralelných výpočtov pri indexácii a dolovaní v textoch, a podobne. Prehľad histórie nasadení jednotlivých verzií systému od jeho vzniku do súčasnosti je uvedený v Tab. B.1.1.

Tab. B.1.1 História nasadení systému JBowl

Verzia	Dátum nasadenia	Popis funkčnosti
0.1	7.9.2005	Počiatočná verzia pre Source Forge. Distribúcia bola rozdelená na API a implementáciu algoritmov, ktoré boli distribuované samostatne. Viacero API rozhraní bolo v štádiu návrhu.
0.2	20.9.2005	Implementácia balíku org.jbowl.analysis.tagging pre prístup k morfologickému slovníku. Podpora klasifikácie do tried usporiadaných v taxonómii.
0.3	3.11.2005	Implementácia balíku org.jbowl.analysis.tagging pre prístup k morfologickému slovníku. Podpora klasifikácie do tried usporiadaných v taxonómii.
0.4	13.2.2006	Spojenie API a algoritmov do jednej distribúcie. Nové balíky s príkladmi indexovania a budovania modelov a príklad implementácie nového algoritmu (perceptron). Implementácia algoritmov k-Means a Boostingu.
0.5	17.3.2006	Oprava chýb. Minoritné rozšírenia tried a rozhraní.
0.6	15.4.2007	Oprava chýb. Minoritné rozšírenia tried a rozhraní.
0.7	3.9.2007	Počiatočný návrh rozhraní pre štruktúrované úložisko objektov, pre uloženie predspracovaných dát, artefactov (štatistík atď.) a modelov.
0.8	30.4.2009	Stabilizácia rozhraní pre predspracovanie dát (balík org.jbowl.process a org.jbowl.task). Implementácia stroja pre paralelné spúšťanie úloh. Implementácia rozhraní pre vytváranie pracovných tokov pre úlohy predspracovania dát a budovania a validácie modelov. Nový dátový typ inštancií, okrem vektorovej reprezentácie je možné dokumenty reprezentovať ako sekvenciu zaindexovaných termov.

B.1.3 Aplikácie

Systém JBowl bol od začiatku navrhovaný a budovaný ako podporný nástroj pre nasadenie v praktických aplikáciách a projektoch. Práve orientácia na praktickú použiteľnosť bola hnacou silou dopĺňania jednotlivých algortimov, a zároveň aj testovacou platformou pre úspešnosť a efektívnosť konkrétnej implementácie. V nasledujúcom prehľade sú vymenované najdôležitejšie výstupy implementácie systému JBowl – projekty z rôznych aplikačných oblastí, diplomové a dizertačné práce realizované na Katedre kybernetiky a umelej inteligencie, resp. v Centre pre informačné technológie, Fakulty elektrotechniky a informatiky, Technickej univerzity v Košiciach v rokoch 2003-2009:

Projekt Webocracy, http://www.webocrat.sk

Projekt 5. rámcového programu EÚ, kontrakt č. FP5 IST-1999-20364.

Aplikačná oblasť: elektronická verejná správa, eGovernment

Realizácia v rokoch 2000-2004.

Systém JBowl bol použitý na podporu inteligentného vyhľadávania informácií, konkrétne na predspracovanie a jazykovú analýzu textových dokumentov a ich následnú indexáciu, tvorbu vektorovej reprezentácie dokumentov a na dolovanie v textoch. Boli testované tri rôzne prístupy dolovania znalostí z textov [Paralič, Bednár 2003]:

- Pri tvorbe a budovaní ontológií pre oblasť verejnej správy sa využilo zhlukovanie textov pomocou asociačných pravidiel.
- Následne sa pre identifikáciu anotačných relácií medzi textami a ontológiou uplatnili algoritmy kategorizácie textov. Podpora semi-automatickej anotácie textov pojmami znalostného modelu je azda najdôležitejšou ukážkou komplexného využitia funkcionality ponúkanej systémom JBowl.
- Napokon sa index a vektorová reprezentácia dokumentov využili aj pri fulltextovom vyhľadávaní, integrovanom do konceptuálneho vyhľadávania podľa asociovaných pojmov ontológie.

Projekt GridMiner, http://www.gridminer.org

Projekt GridMiner bol jednou z prvých aktivít v oblasti dolovania dát v distribuovanom gridovom prostredí. Hlavným cieľom bolo poskytnúť systém, ktorý pokrýva všetky fázy procesu objavovania znalostí vrátane predspracovania, samotného dolovania a vizualizácie výsledkov a integruje ich do servisne orientovanej gridovej aplikácie [Brezany et al. 2004].

Aplikačná oblasť: analýza medicínskych údajov, objavovanie znalostí v databázach a neštruktúrovaných textoch

Realizácia v rokoch 2004-2006.

V projekte bol systém JBowl využitý na objavovanie znalostí v heterogénnych a distribuovaných priestoroch údajov. Aplikačnou oblasťou boli texty z oblasti medicíny, písané v angličtine. Pomocou funkcií systému JBowl sa uskutočovalo predspracovanie textov (delenie na lexikálne jednotky, značkovanie, filtrovanie stop-slov a morfologická analýza), indexácia textov, transformácia do matice term-dokument, nastavenie váh pomocou algoritmu TF-IDF. Z algoritmov dolovania v textoch boli použité klasifikačné aj zhlukovacie algoritmy, výstupné modely boli reprezentované v jazyku PMML. Metódy na vyhodnotenie kvality a úspešnosti klasifikácie a zhlukovania poskytla tiež knižnica JBowl, vizualizáciu výsledkov a ich následné využitie zabezpečoval samotný systém GridMiner.

Projekt KP-Lab, http://www.kp-lab.org

Integrovaný IST projekt 6. rámcového programu EÚ, kontrakt č. FP6 IST-2004-27490.

Aplikačná oblasť: elektronická výučba, distribuovaný kolaboratívny systém pre podporu procesov tvorby nových znalostí

Realizácia v rokoch 2006-2011.

Systém JBowl sa využil na podporu anotácií a sémantického popisu znalostných artefaktov – elementárnych jednotiek výučbového procesu [Smrž a kol. 2007], [Furdík a kol. 2008a]. Konkrétne sa uplatili:

- algoritmy kategorizácie na klasifikáciu znalostných artefaktov do predde-finovaných kategórií podľa ich textového popisu,
- algoritmy zhlukovania na identifikáciu skupín vzájomne obsahovo podobných artefaktov,
- algoritmy extrakcie kľúčových slov a sumarizácie na výber najdôležitejších pojmov z textového popisu artefaktov, následne na tvorbu inicializačného slovníka pre budovanie ontológie.

Príslušný modul systému KP-Lab, implementovaný na báze knižnice JBowl, je vrátane podrobného popisu a zdrojových kódov k dispozícii na adrese http://cit.fei.tuke.sk:8080/TMSClassify/.

Projekt PoZnaŤ, http://web.tuke.sk/fei-cit/poznat/

APVV projekt č. RPEU-0011-06, nadväzujúci na FP6 EÚ projekt KP-Lab.

Aplikačná oblasť: elektronická výučba, eLearning, jazyková analýza textov v slovenčine

Realizácia v rokoch 2007-2009.

V rámci projektu bol systém JBowl, v rozsahu funkcionality implementovanej pre projekt KP-Lab, aplikovaný na spracovanie textov v slovenčine. Boli doplnené funkcie na analýzu slovenských textov, najmä jazykovo-špecifické metódy lematizácie a čiastočnej morfologickej analýzy. Bol navrhnutý a implementovaný paralelný spôsob vykonávania zhlukovacích a klasifikačných algoritmov, čím sa podstatne zvýšila efektívnosť činnosti systému JBowl pri úlohách získavania znalostí z textov [Butka a kol. 2009]. Ďalšie rozšírenie sa týkalo samostatných aplikácií kolaboratívneho systému KP-Lab a systému JBowl vo vyučovacom procese. V rámci týchto aplikácií boli experimentálne testované jednotlivé funkčnosti s cieľom ďalšieho vývoja a zlepšení. Na sledovanie aktivít a činností používateľov (študentov) v kolaboratívnom systéme a následnej analýzy získaných údajov bolo implementované rozhranie poskytujúce webovské služby pre registráciu, ukladanie, vyhľadávanie a analýzu záznamov o používateľských aktivitách pri práci so systémom [Paralič, Babič 2008]. Napokon, bolo vytvorené webovské používateľské rozhranie, ktoré sprístupňuje niektoré z funkcií systému JBowl, najmä klasifikačné úlohy, študentom v používateľsky prístupnej forme štandardnej webovskej aplikácie [Furdík 2007]. Popis tohto webového rozhrania, tzv. obslužnej konzoly systému JBowl, je uvedený v časti B.1.6.

Projekt Gemin, http://cit.fei.tuke.sk/Gemin/

MVTS projekt „Extrakcia informácií a získavanie znalostí pre podporu výskumu dedičných ochorení (Eco-Net)“ s označením Fr/ČR/SR/TUKE07. Spoločný projekt CIT FEI TU v Košiciach a Ústavu biologických a ekologických vied Lekárskej fakulty UPJŠ.

Aplikačná oblasť: získavanie znalostí z medicínskych textov

Realizácia v rokoch 2007-2010.

Pomocou systému JBowl sa vyhľadávali informácie a získavali sa znalosti z voľne dostupných biomedicínskych databáz, najmä NCBI a OMIM [Jasem a kol. 2008]. Údaje v týchto databázach sú dostupné vo forme neštruktúrovaných textov v PDF formáte, resp. vo forme čiastočne štruktúrovaných a sémanticky ohodnotených textov vo formáte XML. Metódami systému JBowl sa tieto texty získavali z externých úložísk, vo fáze predspracovania sa segmentovali a indexovali. Následne sa využili algoritmy zhlukovania a kategorizácie na identifikáciu vzájomných podobností a obsahových súvislostí medzi textovými položkami. Funkcie jazykovej analýzy a kategorizácie sa použili na tvorbu sémantických filtrov, ktoré výrazne zefektívnili vyhľadávanie v rozsiahlych a dynamicky sa meniacich biomedicínskych databázach.

Projekt Dolovanie v textoch pre extrakciu metadát a sémantické vyhľadávanie, http://web.tuke.sk/fei-cit/daad.html

DAAD projekt č. 8/2004

Realizácia v rokoch 2005-2006.

V rámci tohto projektu prebiehal intenzívny vývoj systému JBowl, najmä metód pre klasifikáciu a zhlukovanie textov, vrátane výslednej vizualizácie hierarchických máp typu GHSOM. Okrem týchto metód aplikovaných na extrakciu metadát z kolekcií textových dokumentov, bolo nemeckým partnerom (Univerzita Regensburg) implementované rozšírenie vektorového modelu pre spracovanie fuzzy dopytov na báze RDF. Takýmto spôsobom bola výrane vylepšená schopnosť sémantického vyhľadávania.

ATN siete, syntakticko-sémantická analýza slovenčiny

M. Tymeš v roku 2006, v rámci svojej diplomovej práce [Tymeš 2006], implementoval do systému JBowl moduly jazykovej analýzy pre spracovanie textov v slovenčine. Upravené boli mechanizmy tokenizácie, lematizácie a následnej morfologickej anotácie podľa spôsobov použitých pri morfologickej anotácii Slovenského národného korpusu [Garabík a kol. 2004]. Ťažiskom práce bola realizácia modulu syntakticko-sémantickej analýzy, založeného na ATN sieťach a na viazaní gramatických kategórií medzi lexikálnymi jednotkami. Na základe údajov z reálnych textov bol vytvorený slovník ATN sietí pre slovenčinu a výsledná aplikácia bola úspešne testovaná v systéme na extrakciu informácií.

Meta-učenie klasifikátorov

G. Tutoky v roku 2008 vo svojej diplomovej práci [Tutoky 2008] rozpracoval prístup meta-učenia pre klasifikáciu textov, využívajúci princípy MUDOF algoritmu [Wai, Kwok-Yin 2001]. Systém JBowl ponúka pre klasifikačné úlohy celú škálu algoritmov, napríklad Perceptron, SVM, kNN, rozhodovacie stromy, atď. (porov. v časti B.1.5.1.4). Prístup meta-učenia spočíva vo výbere najvhodnejšieho z týchto klasifikačných algoritmov pre konkrétnu úlohu, pričom miera vhodnosti sa vypočítava na základe charakteristík danej trénovacej množiny textových dokumentov. Softvérové riešenie meta-učenia klasifikátorov bolo navrhnuté a implementované ako rozšírenie systému JBowl. Vhodnosť riešenia bola experimentálne overená vyhodnotením kvality klasifikácie (vyjadrenej presnosťou, úplnosťou a mierou F1) dosiahnutej meta-učením v porovnaní s výsledkami klasických algoritmov používaných pri klasifikácií textov [Furdík a kol. 2008b].

Využitie formálnej konceptovej analýzy pre tvorbu hierarchie konceptov z textových dokumentov

M. Zeher v roku 2006 vo svojej diplomovej práci [Zeher 2006] rozpracoval prístup pre tvorbu hierarchie konceptov z textových dokumentov kombináciou zhlukovania algoritmom GHSOM a následným použitím metódy formálnej konceptovej analýzy (FCA – Formal Concept Analysis [Ganter, Wille 1997]) na dekomponované podmnožiny dát. Výsledkom boli lokálne modely na báze FCA, ktoré sa kombinovali pomocou aglomeratívneho zhlukovania do jednotného modelu hierarchie konceptov. Implementovaný prístup podrobnejšie rozpracoval pôvodný návrh dekompozície na základe práce [Butka 2006]. Navrhnutý algoritmus bol implementovaný a otestovaný aj v distribuovanom gridovom prostredí. E. Tkáč vo svojej diplomovej práci z roku 2007 tento prístup implementoval ako gridovú službu a experimentálne otestoval v reálnom distribuovanom prostredí [Tkáč 2007]. Softvérové riešenie bolo v oboch prípadoch založené na knižnici JBowl. Prezentované výsledky poukazujú na schopnosť algoritmu extrahovať koncepty z textových dokumentov, pričom dekompozícia a jej distribúcia poskytujú priestor pre súčasné zlepšenie efektívnosti výpočtu výsledného modelu pomocou paralelného a distribuovaného spracovania dát.

Rozšírenia realizované v uvedených prácach a projektoch väčšinou neboli zahrnuté do oficiálnej distribúcie systému JBowl a ostali iba v prototypových riešeniach. Predstavujú však podnetné aplikačné smery možného využitia systému a je pravdepodobné, že niektoré z týchto rozšírení nájdu uplatnenie v budúcich verziách.

B.1.4 Architektúra

V tejto časti popisujeme systém JBowl vo verzii 0.9_08, ktorá je prístupná na portáli SourceForge.org a ktorej základné charakteristiky boli uvedené na začiatku kapitoly B.1. Rozšírenia realizované v rámci jednotlivých projektov a aplikácií, popísané v časti B.1.3, poväčšine nie sú súčasťou oficiálnych vydaní základného systému JBowl a preto nie sú zahrnuté do popisu architektúry.

Systém JBowl bol od začiatku budovaný na princípoch stavby architektúry, akú má štandardné rozhranie Java Data Mining API (JSR 73, špecifikácia 6) [JSR 73]. Táto architektúra pozostáva z troch základných komponentov, ktoré môžu byť implementované buď samostatne, alebo v distribuovanom prostredí:

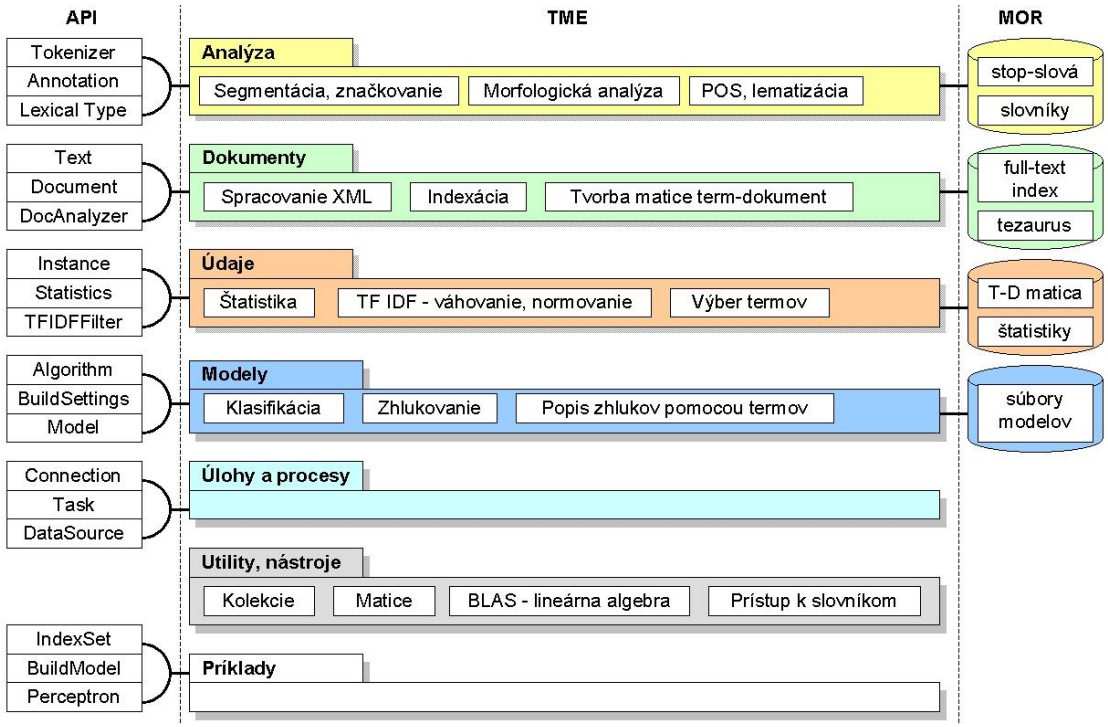
1. **Aplikačné programovacie rozhranie** (API) obsahuje množinu používateľsky viditeľných tried a rozhraní, ktoré dovoľujú prístup k poskytovaným službám pomocou prostriedkov dolovania v textoch (TME). Pre aplikácie využívajúce knižnicu JBowl stačí poznať iba toto API rozhranie, nie je potrebná detailná znalosť ďalších podporných komponentov.
2. **Prostriedky dolovania v textoch** (Text Mining Engine, TME) poskytujú infraštruktúru, ktorá ponúka množinu služieb dolovania v textoch pre API klientov. TME časť môže byť v aplkáciách implementovaná ako lokálna knižnica, alebo ako server v architektúre klient – server.
3. **Sklady dolovaných objektov** (Mining Objects Repository, MOR) – TME využíva sklady dolovaných objektov, ktoré slúžia na umiestnenie modelov dolovania v textoch pre algoritmy kontrolovaného a nekontrolovaného učenia. MOR poskytuje aj prostriedky správy a prístupu k úložiskám údajov pre metódy jazykovej analýzy a indexácie textov.

TME riadi realizáciu všeobecných úloh dolovania v textoch, ktorými sú napríklad jazyková analýza textu dokumentu, budovanie a testovanie modelu, aplikovanie modelu na nové údaje, výpočet štatistík, import a export existujúcich objektov dolovania z a do MOR.

Z funkčného hľadiska pozostáva knižnica JBowl z modulov, ktoré možno rozdeliť do siedmich úrovní (Obr. B.1.1):

1. **Analýza**, ktorá zahŕňa triedy z balíka org.jbowl.analysis.

   Do tejto úrovne patria všetky operácie predspracovania a jazykovej analýzy textu: načítanie XML formátu vstupných textov, segmentácia na slová a vety (parsing, sentence chunking), značkovanie, identifikácia slovných druhov (POS tagging), morfologická analýza (stemming), lematizácia, správa anotačných vrstiev, prístup k slovníkom vrátane databázy WordNet.



Obr. B.1.1 Schéma komponentov a úrovní systému JBowl

Rozdelenie podľa komponentov architektúry:

- API: balík org.jbowl.analysis, triedy LexicalType, Token, TokenFilter a Tokenizer balíka org.jbowl.analysis.tokens.
- TME: vnútorná implementácia rôznych druhov segmentácie textu na slová (balík org.jbowl.analysis.tokens) a na vety (balík org.jbowl.analysis.sentences), prístup k slovníkom slovných druhov a morfologických kategórií (balík org.jbowl.analysis.tagging), prístup k databáze WordNet (balík org.jbowl.analysis.wordnet).
- MOR: zoznam stop-slov (súbor stopwords_en.properties v balíku org.jbowl.analysis.tokens), slovník pre členenie textu na vety (súbor sentencechunks_en.properties v balíku org.jbowl.analysis.sentences), slovníky slovných druhov a morfologických kategórií, databáza.

2. **Dokumenty**. Táto úroveň zahŕňa triedy z balíka org.jbowl.document.

   Na tejto úrovni sú implementované mechanizmy pre reprezentáciu textových dokumentov v XML formáte (delenie na kapitoly, odseky, meta-údaje), pre indexáciu a vytváranie matice term-dokument, fulltextový index (klasické vyhľadávanie v plnom texte) a tezaurus pre popis obsahu dokumentu pomocou kľúčových slov – termov.

   Rozdelenie podľa komponentov architektúry:

   - API: balíky org.jbowl.document a org.jbowl.document.processing.

- TME: implementácia fulltextovej indexácie a podpora vyhľadávania v plnom texte (balík org.jbowl.document.lucene), prístup k dokumentom v XML formáte vrátane ich indexácie (balík org.jbowl.document.xml).
- MOR: fulltextový index, tezaurus.

3. **Údaje**. Táto úroveň zahŕňa triedy z balíka org.jbowl.data.

   V tejto úrovni sa nachádzajú funkcie na manipuláciu s vektorovým modelom, čiže údajovou reprezentáciou textov po predspracovaní a jazykovej analýze. Patria sem mechanizmy výberu vhodných termov (kľúčových slov, fráz, konceptuálnych popisov), rôzne spôsoby váhovania a normovania termov (binárne, na základe frekvencie výskytu, alebo TF-IDF váhovanie), a napokon štatistické operácie na vyhodnocovanie matice term-dokument.

   Rozdelenie podľa komponentov architektúry:

   - API: balíky org.jbowl.data a org.jbowl.data.processing.
   - TME: vnútorná implementácia rozhraní pre prístup k vektorovej reprezentácii textov (balík org.jbowl.data.impl), triedy pre rôzne spôsoby štatistického vyhodnocovania matice term-dokument (balík org.jbowl.data.termselection).
   - MOR: v súbore uložená matica term-dokument, súbory štatistík indexácie, termov a kategórií.

4. **Modely**. Táto úroveň zahŕňa triedy z balíka org.jbowl.model.

   Úroveň obsahuje implementáciu viacerých algoritmov pre dolovanie v textoch využívajúcich kontrolované učenie (algoritmy klasifikácie, resp. kategorizácie textov) aj nekontrolované učenie (algoritmy zhlukovania a doplnkové algoritmy na popis zhlukov pomocou termov).

   Rozdelenie podľa komponentov architektúry:

   - API: balík org.jbowl.model, trieda ClassificationModel z balíka org.jbowl.model.supervised a trieda ClusteringModel z balíka org.jbowl.model.unsupervised.
   - TME: vnútorná implementácia algoritmov kontrolovaného učenia (balík org.jbowl.model.supervised a od neho odvodené balíky s implementáciou jednotlivých klasifikačných algoritmov), algoritmov nekontrolovaného učenia (balík org.jbowl.model.unsupervised a od neho odvodený balík org.jbowl.model.unsupervised.kmeans s implementáciou metó-dy K-Means) a algoritmov popis zhlukov pomocou termov (balík org.jbowl.model.descriptive).
   - MOR: v súboroch uložené klasifikačné, zhlukovacie a popisné modely.

5. **Úlohy a procesy**. Táto úroveň zahŕňa triedy z balíkov org.jbowl.task a org.jbowl.process.

   Na tejto úrovni sú implementované mechanizmy pre manipuláciu s údajmi modelov pre dolovanie v textoch. Úlohy (balík org.jbowl.task) predstavujú rôzne typy operácií nad údajmi; procesy (balík org.jbowl.process) obsahujú implementáciu strojov (engines), ktoré spúšťajú dané operácie nad údajmi.

   Všetky triedy tejto úrovne patria do komponentu API – sprostredkúvajú prístup k úlohám indexácie, tvorby a vyhodnotenia všetkých podporovaných typov modelov dolovania v textoch.

6. **Utility, nástroje**. Táto úroveň zahŕňa triedy z balíkov org.jbowl.util a tools.

   Na tejto úrovni sa nachádzajú podporné a obslužné programy pre efektívnu prácu s maticovou a vektorovou reprezentáciou dokumentov (balík org.jbowl.util) a nástroje pre prístup k údajom zo špecializovaných slovníkov (balík tools).

   Všetky triedy tejto úrovne patria do komponentu TME.

7. **Príklady**. Táto úroveň zahŕňa triedy z balíka examples.

   Úroveň obsahuje ukážkové príklady použitia systému JBowl pre úlohy predspracovania, tvorby a vyhodnotenia klasifikačných modelov (balík examples.task), a tiež typickú ukážku implementácie jednoduchého klasifikačného modelu Perceptron (balík examples.supervised).

   Všetky triedy tejto úrovne možno zaradiť do komponentu API (hoci nepredstavujú skutočné rozhranie, ale skôr konkrétnu implementáciu rozhraní poskytovaných inými úrovňami).

B.1.5 Príklady použitia a implementácie

Systém JBowl je určený na použitie v úlohách kontrolovaného a nekontrolovaného strojového učenia pre vytváranie aplikácií získavania znalostí a dolovania v textoch. V tejto časti je popísaný spôsob použitia systému JBowl pre oba typy úloh, t.j. pre klasifikáciu a pre zhlukovanie textových dokumentov.

B.1.5.1 Kontrolované učenie – klasifikácia

Úloha kontrolovaného strojového učenia, t.j. klasifikácia (resp. kategorizácia) textov na základe modelu vytvoreného z vopred danej trénovacej množiny vzorových príkladov, sa v systéme JBowl realizuje v etapách:

1. príprava vstupu, t.j. formátovanie trénovacej a testovacej množiny,
2. predspracovanie a jazyková analýza,
3. indexácia,
4. tvorba klasifikačného modelu,

5. vyhodnotenie kvality vytvoreného klasifikačného modelu,
6. samotná klasifikácia

V ďalšom sú podrobne popísané spôsoby aplikovania systému JBowl pre každú z týchto etáp, vrátane implementačných detailov a ukážok Java kódu.

B.1.5.1.1 Príprava vstupu

Vstupom pre algoritmy kontolovaného učenia je trénovacia a testovacia množina textových dokumentov ohodnotených kategóriami. Pretože na takto vopred kategorizované texty neexistuje štandardizovaný formát, používa systém JBowl špecifickú formu XML formátu. Na vstupe sa vyžaduje trénovacia a testovacia množina pozostávajúca z kategorizovaných dokumentov vyjadrených v tvare[26]:

```
<document id="19" name="Reuters-21578-19" dataset="ModApte-
train">
<category>topics.grain</category>
<category>topics.wheat</category>
<title>BONUS WHEAT FLOUR FOR NORTH YEMEN  - USDA</title>
<text>The Commodity Credit Corporation, CCC, has accepted
an export bonus offer to cover the sale of 37,000 long tons
of wheat flour to North Yemen, the U.S. Agriculture
Department said.
... </text>
</document>
```

Hlavičku dokumentu predstavuje element `document`, ktorý môže byť ohodnotený viacerými (nepovinnými) parametrami – napr. identifikačné číslo dokumentu `id`, jeho interný názov `name`, názov dátovej množiny `dataset` (ktorý môže vyjadrovať príslušnosť k trénovacej či testovacej množine), a podobne. V hlavičke, ak je to potrebné pre tú-ktorú aplikáciu, môžu byť uvedené aj ďalšie parametre napríklad informácie o autoroch, dátum vzniku alebo poslednej modifikácie dokumentu, atď.

Po hlavičke nasleduje zoznam názvov kategórií, ktoré sú priradené danému dokumentu. Každý z názvov kategórií je uzavretý do samostnatného XML elementu `category`.

Ďalej je uvedený názov, resp. titulok dokumentu, uzavretý do elementu `title`. Po ňom, v elemente `text`, nasleduje samotný text dokumentu. Členenie XML dokumentu je zachované v objektovom API dokumentov, t.j. napr. z elementu `text` sa vytvorí sekcia, ktorej sa priradí text elementu, ktorý môže byť ďalej rozdelený do podsekcií podľa členenia vnorených XML značiek.

Knižnica JBowl neobsahuje nástroje na konverziu rôznych proprietárnych formátov, napr. PDF, DOC/DOCX, HTML, a podobne, na žiadanú formu XML formátu. Dôvodom je vyššie spomínaná neexistencia štandardu pre texty ohodnotené kategóriami. Na prípravu vstupu do systému JBowl je preto

[26] Príklady textov v kapitolách B.1.5.1 a B.1.5.2 sú vytvorené z testovacej kolekcie Reuters-21578, ktorá je súčasťou distribúcie systému JBowl.

potrebné extrahovať z toho-ktorého proprietárneho formátu čistý text, a tento, spolu so zoznamom jemu príslušných kategórií a parametrov, upraviť do požadovaného XML formátu.

B.1.5.1.2 Predspracovanie a jazyková analýza

Pre klasifikačné algoritmy dolovania v textoch je potrebné transformovať vstupný XML formát trénovacej a testovacej množiny na vektorovú reprezentáciu, v ktorej je dokument reprezentovaný ako vektor v n-rozmernom príznakovom priestore (n je počet príznakov, termov). Príznakmi sú najčastejšie plnovýznamové slová alebo korene slov, môžu nimi však byť aj frázy, n-gramy, významy slov (napr. WordNet synsety), a podobne. Voľba príznakov závisí od konkrétnej aplikácie, a tiež od jazyka použitého v textoch trénovacej a testovacej množiny. Vo všeobecnosti však platí, že zo zvyšovaním zložitosti jazykovej analýzy rastie výpočtová náročnosť procesu predspracovania.

Proces transformácie vstupného textu v XML formáte do vektora termov zahŕňa členenie na slová (segmentáciu, parsing), tokenizáciu (značkovanie pozícií výskytov slov v texte), lematizáciu (úpravu na základný tvar), odstránenie neplnovýznamových slov, váhovanie a normovanie termov.

Predspracovanie a jazyková analýza sa v systéme JBowl vykonávajú pomocou funkcie, ktorá môže byť definovaná napríklad takto:

```
static DataTarget<AnnotatedText> createTextAnalyzer() {
    return new    SimpleTokenizer(new StopFilter(new
PorterStemmer()));
}
```

V tomto prípade sa analýza textu vykonáva pomocou Java tried implementovaných v balíku org.jbowl.analysis.tokens. Trieda `SimpleTokenizer` člení text na slová podľa výskytu alfanumerických znakov. Parametrom jej konštruktora je trieda `StopFilter`, ktorá z textu odstráni neplnovýznamové slová vymenované v súbore stopwords_en. properties. Parametrom v konštruktore tejto triedy je trieda `PorterStemmer`, ktorá pomocou pravidiel Porterovho stemmera transformuje anglické slová v texte na ich základný tvar (t.j. nahradí anglické slová ich kmeňovým základom).

Systém JBowl ponúka na konštrukciu funkcie `createTextAnalyzer` aj ďalšie triedy, ktorých kombináciou je možné navrhnúť sofistikovanejší proces predspracovania a jazykovej analýzy. Celkový prehľad týchto tried a ich použitia je prezentovaný v Tab. B.1.2.

Výsledok analýzy je reprezentovaný objektovým modelom anotovaného textu (balík org.jbowl.analysis). Základným rozhraním je trieda org.jbowl.analysis. Annotation, ktorá reprezentuje ohraničenú časť textu ku ktorej je možné pridať rôzne metadáta. Pomocou anotácií sú napr. v texte vyhradené jednotlivé tokeny, ktorým je priradený ich lexikálny typ (t.j. či ide o reťazec písmen, prázdne znaky, číslo, zmiešaný alfanumerický reťazec, atď.). Ďalším príkladom metadát, ktoré môžu byť priradené ohraničenej časti textu je napr. lemma (základný tvar slova), alebo morfologické značky (rod, pád, číslo, slovesný vid, atď.).

Tab. B.1.2 Triedy na predspracovanie a jazykovú analýzu textu

Funkcia / trieda	Popis, použitie
Členenie textu na slová	
SimpleTokenizer	Tokenizér, ktorý rozdelí text podľa prázdnych znakov a prevedie všetky písmena na malé.
RegExTokenizer	Tokenizér, ktorý rozdeľuje text podľa regulárnych výrazov. Každý token je potom označený podľa jeho lexikálneho typu (prednastavené regulárne výrazy označujú prázdne znaky, alfanumerické reťazce, reťazce z písmen a čísla).
WhitespaceTokenizer	Tokenizér, ktorý rozdelí text podľa prázdnych znakov.
Tokenizer	Základná trieda pre všetky tokenizéry. Tokenizér je komponent, ktorý rozdelí vstupný text (sekvenciu znakov) na lexikálne jednotky – tokeny.
Odstránenie neplnovýznamových slov	
StopFilter	Filter, ktorý z anotácií odstráni všetky tokeny, ktoré sa vyskytujú v zozname stop slov (prednastavený zoznam je pre anglický jazyk zo súboru stopwords_en.properties)
Lematizácia, úprava na základný tvar	
PorterStemmer	Filter, ktorý prevedie tokeny, ktoré označujú slová v anglickom jazyku na ich koreň podľa Porterovho algoritmu (napr. students prevedie na student, alebo walking na walk, atď.).
Pomocné triedy, definícia objektov	
LexicalType	Lexikálny typ tokenov. Okrem označenia definuje aj regulárny výraz podľa ktorého je možné daný typ rozoznať v texte (viď. RegExTokenizer)
Token	Anotácia ktorá označuje v texte token – t.j. základnú lexikálnu jednotku ktorú je možné zaindexovať, resp. ktorej je možné priradiť ďalšie metadáta (t.j. väčšina ostatných metód pre analýzu textu už vyžaduje aby bol text rozdelený na tokeny)
TokenFilter	Základná trieda pre filter tokenov, ktorý slúži na zmenu / filtrovanie tokenov identifikovaných v texte niektorým tokenizérom.

Anotácie sú hierarchicky usporiadané do tzv. anotačných rovín, ktoré obsahujú anotácie rovnakého typu (anotácie na rôznych rovinách sa však môžu prekrývať, t.j. napr. jeden podreťazec môže byť označený ako token zložený z písmen, a zároveň môže byť označený ako slovo s daným základným tvarom a značkovaním).

Okrem analýzy na úrovni token ďalej balík org.jbowl.analysis obsahuje nasledujúce odvodené balíky.

Tab. B.1.3 Odvodené balíky v balíku org.jbowl.analysis

Funkcia / balík	Popis, použitie
org.jbowl.analysis.sentences	Filtre pre delenie textov na vety. Vstupný text už musí byť rozdelený na tokeny. Základnou triedou v balíku je `HeuristicSentenceChunker`, ktorá identifikuje koniec podľa toho či token interpunkcie, predchádzajúci token a nasledujúci token patrí do zoznamu možných ukočení viet, zoznamu tokenov, ktoré nemôžu predchádzať ukončeniu vety (napr. časti skratiek, čísla, atď.) a zoznamu tokenov, ktoré nemôžu nasledovať za ukončením vety. Prednastavené delenie pre anglický jazyk je uložené v súbore `sentencechunks_en.properties`
org.jbowl.analysis.tagging	Programové rozhranie pre prácu s morfologickým slovníkom. Základnou triedou je `Dictionary`, ktorá umožňuje vyhľadávanie slov (trieda `TaggedWord`). Každému slovu je potom priradený odkaz na základný tvar a morfologická značka, ktorá obsahuje hodnoty morfologických kategórií (t.j. napr. rod, pád, číslo, vid, atď.). V balíku tools sú pomocné triedy, ktoré vytvoria slovník vygenerovaný zo Slovenského národného korpusu.
org.jbowl.analysis.wordnet	Programové rozhranie pre prácu s lexikálnou databázou WordNet. Základnou triedou je `Dictionary`, ktorá umožňuje vyhľadávanie slov a synsetov (synset je množina synoním). Ku každému slovu je priradený odkaz na množinu významov, ktoré slovo v texte môže nadobúdať. Významy sú reprezentované ako množina synsetov. Slová a synsety sú ďalej prepojené lexikálnymi reláciami ako napr. hyperonymum/hyponymum, meronymum, atď.

Výsledkom jazykovej analýzy sú identifikované a lematizované termy, ktoré reprezentujú daný dokument. Poslednou etapou predspracovania je následné váhovnie a normovanie termov. Jednotlivé termy sa číselne ohodnotia podľa ich relatívnej dôležitosti, príspevku k celkovému obsahu dokumentu, a to vzhľadom na rozloženie všetkých termov v celom korpuse

dokumentov.

Váhovanie a normovanie termov je v systéme JBowl implementované v triede `TFIDFFilter`, ktorá je súčasťou balíka org.jbowl.data.processing. Konkrétny typ váhovania a normovania sa dá v kóde nastaviť pomocou premennej `TFIDF_SCHEME`:

```
static final String TFIDF_SCHEME = "ltc";
```

Hodnotou tejto premennej sú vždy tri znaky, ktoré, podľa svojej pozície, vyjadrujú:

- typ *TF* schémy: jeden zo znakov {a, n, b, m, s, l}
- typ *IDF* schémy: jeden zo znakov {n, t, p, f, s}
- typ normovania: jeden zo znakov {n, s, c, m}

Nastavenie premennej na reťazec `"ltc"` teda znamená, že hodnoty zložiek vektora sú určené vynásobením `l` schémy pre *TF* a `t` schémy pre *IDF*, a celý vektor je znormovaný vydelením normou `c`. Význam jednotlivých znakov popisuje Tab. B.1.4, kde pre dokument i a term j, je tf frekvencia termu j v dokumente i, $maxtf$ je maximálna frekvencia termu j, ktorý sa vyskytol v dokumente i najčastejšie, df je počet dokumentov v trénovacej množine v ktorých sa vyskytol term j a N je počet dokumentov v trénovacej množine.

Tab. B.1.4 Parametre váhovania a normovanie termov

Typ / znak	Hodnota
Typ TF schémy	
a	$0.5 + 0.5 * (tf / maxtf)$
n	tf (jednoduché tf váhovanie)
b	1 (binárne váhovanie)
m	$tf / maxtf$
s	$tf * tf$
l	$\log(tf) + 1$
Typ IDF schémy	
n	1 Bez inverznej dokumentovej frekvencie.
t	$\log(N / df)$ Inverzná dokumentová frekvencia.
p	$\log(N - df)$
f	$1 / df$
s	$\log(N / df)^2$
Normovanie	
n	Bez normalizácie.
s	Normalizácia sumou zložiek vektora.

c	Normalizácia na 1 dĺžku vektora
m	Normalizácia maximálnou hodnotou vo vektore dokumentu.

B.1.5.1.3 Indexácia

Cieľom indexácie je transformácia vektorov váhovaných a normovaných termov, identifikovaných vo fáze predspracovania pre všetky dokumenty trénovacej a testovacej množiny, do objektovej reprezentácie matice term-dokument, vhodnej pre aplikovanie klasifikačných algoritmov a vytvorenie klasifikačných modelov.

Na začiatku indexácie pomocou systému JBowl sa vytvoria objekty pre indexovacie množiny termov a kategórií, do ktorých sú jednotlivé dokumenty zaradené:

```
IndexedSet<String> terms = new HashIndexedSet<String>();
IndexedSet<String> categories = new
        HashIndexedSet<String>();
```

Deklaruje sa objekt pre výstupné štatistiky. Argument funkcie `InstanceOutputStream()` je dočasný súbor, ktorý slúži na priebežné ukladanie údajov počas indexácie:

```
File tmp = File.createTempFile("tmp", null);
tmp.deleteOnExit();
Statistics trainStats = new Statistics(
        new Utils.ProgressLogger(
        new InstanceOutputStream(tmp)));
```

Vytvorí sa indexovací objekt. Vstupným parametrom je nastavenie predspracovania a jazykovej analýzy, implementované vo funkcii `createTextAnalyzer()`. Ďalšími vstupnými parametrmi sú indexovacie množiny termov `terms` a kategórií `categories`, a tiež objekt `trainStats` pre výstupné štatistiky:

```
XMLDocumentIndexer indexer = new XMLDocumentIndexer(
        new DocumentAnalyzer(createTextAnalyzer(),
        new DocumentIndexer(terms, categories,
                trainStats)));
```

Následne sa vykoná proces samotnej indexácie volaním funkcie `indexFile()` indexovacieho objektu. Vstupným parametrom funkcie je cesta k súboru obsahujúcemu XML reprezentácie dokumentov trénovacej (resp. testovacej) množiny – v našom prípade je to data/Reuters-21578_ModApte/train.xml:

```
indexer.indexFile("data/Reuters-21578_ModApte/train.xml");
indexer.close();
```

V tejto etape sú pre všetky dokumenty zo vstupnej množiny identifikované príslušné vektory lematizovaných termov. Následne je potrebné vykonať

váhovanie a normalizáciu. Načítajú sa vstupy z dočasného súboru `tmp`, na ne sa aplikujú algoritmy váhovania a normovania podľa nastavení premennej `TFIDF_SCHEME`. Štatistiky vypočítané počas indexácie sa zapíšu do objektu `trainStats`. Výsledná matica term-dokument sa v kódovanom tvare uloží do súboru určeného vstupným parametrom funkcie `InstanceOutputStream()` – v našom prípade je to súbor temp/train-instances:

```
new InstanceInputStream(tmp).processAll(
        new TFIDFFilter(TFIDF_SCHEME, trainStats,
        new Utils.ProgressLogger(
        new InstanceOutputStream("temp/train-
                instances"))));
```

Index je týmto vytvorený a existuje v objektovej reprezentácii. Vypočítané štatistiky sa nachádzajú v objekte `trainStats`. Indexovacie množiny termov a kategórií sú naplnené indexačnými údajmi, ktoré sú zapísané v bjektoch `terms` a `categories`. Index sa z objektovej reprezentácie uloží do súborov pomocou funkcie `writeObject` objektu `Utils`, pričom cesty k príslušným súborom sú vstupnými parametrami tejto funkcie:

```
Utils.writeObject(trainStats, "temp/train-stats");
Utils.writeObject(terms, "temp/term-dictionary");
Utils.writeObject(categories, "temp/category-dictionary");
```

Index je v uvedených súboroch uložený v kódovanom binárnom tvare, optimalizovanom pre ďalšie použitie v algoritmoch klasifikácie a zhlukovania. Pre čitateľný výpis výsledných štatistík do textových súborov možno použiť funkciu `printTermStats` objektu `Utils`:

```
Utils.printTermStats( trainStats, terms, "temp/term-train-
        stats.txt");
Utils.printCategoryStats(trainStats, categories,
        "temp/category-train-stats.txt");
```

Textové výstupy majú tvar trojstĺpcových tabuliek, kde v prvom stĺpci je poradové číslo termu alebo kategórie, v druhom stĺpci je frekvencia termu alebo kategórie v celom korpuse dokumentov a v treťom stĺpci je názov príslušného termu alebo kategórie.

B.1.5.1.4 Vytvorenie klasifikačného modelu

Z indexovanej trénovacej množiny sa pomocou zvoleného klasifikačného algoritmu vytvorí klasifikačný model, ktorý sa následne dá použiť na klasifikáciu ďalších ľubovoľných textových dokumentov. Procedúra tvorby klasifikačného modelu má v systéme JBowl tieto vstupné parametre:

- klasifikačný algoritmus a jeho nastavenia,
- cesta k súboru obsahujúcemu indexovanú trénovaciu množinu,
- cesta k súboru, do ktorého sa zapíše vytvorený klasifikačný model.

Nasledujúci príklad ukazuje, akým spôsobom sa v systéme JBowl vytvorí klasifikačný model využívajúci klasifikačný algoritmus *Perceptron*. Najprv sa

inicializuje objekt `task` reprezentujúci úlohu vytvorenia klasifikačného modelu:

```
BuildModelTask task = new BuildModelTask();
```

Nastavia sa príslušné vstupné parametre. Funkcia `setAlgorithm()` nastaví algoritmus klasifikácie, pričom jej argumentom je názov úplnej cesty k implementácii objektu reprezentujúceho daný algoritmus – v tomto prípade je ním reťazec "examples.supervised.PerceptronAlgorithm":

```
task.setAlgorithm("examples.supervised.
                  PerceptronAlgorithm");
```

Funkcia `setBuildData()` nastaví fyzickú cestu k súboru, ktorý obsahuje indexovanú trénovaciu množinu:

```
task.setBuildData(new InstanceInputStream("temp/train-
        instances"));
```

Funkcia `setBuildSettings()` nastaví parametre použitého klasifikačného algoritmu, ktorým je v tomto prípade algoritmus *Perceptron*. Volanie konštruktora `PerceptronSettings()` nastaví predvolené hodnoty parametrov (napr. `learningRate = 0.1;`) a sprístupní metódy na prípadnú modifikáciu týchto parametrov v kóde:

```
task.setBuildSettings(new PerceptronSettings());
```

Funkcia `setModelName()` nastaví fyzickú cestu k súboru, do ktorého sa zapíše vytvorený klasifikačný model:

```
task.setModelName("temp/model");
```

Získa sa pripojenie na vykonávací objekt:

```
Connection conn =
        ConnectionFactory.createFactory().getConnection();
```

Iniciuje sa vykonanie úlohy na vytvorenie klasifikačného modelu:

```
ExecutionHandler handler = conn.execute(task);
```

Čaká sa na dokončenie tvorby klasifikačného modelu:

```
handler.waitForCompletion();
```

Na tomto mieste sa môže vyhodnotiť kvalita vytvoreného klasifikačného modelu (triedu `EvaluateModel` a funkcie na vyhodnocovanie kvality podrobnejšie popisujeme v nasledujúcej časti, kap. B.1.5.1.5):

```
EvaluateModel.main(argv);
```

Vykonávací objekt sa uvoľní:

```
TaskExecutor.shutdown();
```

Výsledkom je vytvorená objektová reprezentácia klasifikačného modelu, ktorá je uložená v súbore danom vstupným parametrom funkcie `setModelName()` (v tomto prípade je klasifikačný model uložený v súbore temp/model).

Systém JBowl umožňuje na tvorbu klasifikačných modelov použiť škálu algoritmov, ktorých objektové implementácie sa nachádzajú v balíkoch org.jbowl.model.supervised resp. examples.supervised. Každý z algoritmov má špecifické parametre, pomocou ktorých je možné ovplyvňovať a riadiť proces tvorby klasifikačného modelu. V nasledujúcom zozname prezentujeme klasifikačné algoritmy, ich parametre a spôsob implementácie v kóde na vytvorenie príslušného klasifikačného modelu:

Perceptron algoritmus

Umiestnenie: examples.supervised

Parametre (trieda `PerceptronSettings`):

- learningRate: Rýchlosť učenia perceptróna. Typ: double > 0. Predvolená hodnota: 0,1.

Príklad implementácie:

```
task.setAlgorithm("examples.supervised.PerceptronAlgorithm"
        );
BuildSettings settings = new PerceptronSettings();
settings.setLearningRate(0.1);
task.setBuildSettings(settings);
```

Boosting algoritmus

Umiestnenie: org.jbowl.model.supervised.boosting

Parametre (trieda `BoostingSettings`):

- maxNumOfIterations: Maximálny počet základných klasifikátorov, ktoré sa skombinujú do výsledného zloženého klasifikátora Boostingu. Typ: int. Predvolená hodnota: 1000.

Príklad implementácie:

```
task.setAlgorithm("org.jbowl.model.supervised.boosting.Boos
        tingAlgorithm");
BuildSettings settings = new BoostingSettings();
settings.setMaxNumberOfIterations(1000);
task.setBuildSettings(settings);
```

BoosTexter algoritmus

Umiestnenie: org.jbowl.model.supervised.boosting

Parametre (trieda `BoosTexterSettings`):

- abstaining: Určuje, či sa pri klasifikácii bude zohľadňovať aj to či sa term v dokumente nevyskytol (hodnota *false*), alebo sa bude dokument klasifikovať len na základe výskytu termu (hodnota *true*). Typ: boolean. Predvolená hodnota: false.

Príklad implementácie:

```
task.setAlgorithm("org.jbowl.model.supervised.boosting.Boos
        TexterAlgorithm");
BuildSettings settings = new BoosTexterSettings();
settings.setAbstaining(false);
task.setBuildSettings(settings);
```

kNN algoritmus

Umiestnenie: org.jbowl.model.supervised.knn

Parametre (trieda `kNNSettings`):

- k: počet susedov. Typ: int. Predvolená hodnota: 45.
- similarity: Funkcia podobnosti použitá pri výpočte najbližších susedov. Typ: objekt org.jbowl.util.math.Similarity. Predvolená hodnota: cosineSimilarity.

Príklad implementácie:

```
task.setAlgorithm("org.jbowl.model.supervised.knn.kNNAlgori
        thm");
BuildSettings settings = new kNNSettings();
settings.setK(45);
settings.setSimilarity(Similarity.cosineSimilarity());
task.setBuildSettings(settings);
```

Rule algoritmus (t.j. rozhodovacie pravidlá)

Umiestnenie: org.jbowl.model.supervised.rule

Parametre (trieda RuleSettings):

- maxNumOfRules: Maximálny počet rozhodnovacích pravidiel, ktoré sa vygenerujú. Typ: int. Predvolená hodnota: -1. (t.j. neohraničený počet pravidiel)
- maxDepth: Maximálny počet termov, ktoré sa zahrnú do podmienky pravidla. Typ: int. Predvolená hodnota: -1. (t.j. neohraničený počet)
- impurity: Miera „variability“ pokrytých príkladov, ktorá sa použije pre výber podmienky zaradenej do pravidla. Typ: objekt org.jbowl.model.supervised.rule.ImpurityMeasure. Predvolená hodnota: foil.

- binaryValues: Definuje, či sa bude brať do úvahy frekvencia termu (hodnota false), alebo iba binárna hodnota či sa term vyskytol/nevyskytol (hodnota true). Typ: boolean. Predvolená hodnota: false.

Príklad implementácie:

```
task.setAlgorithm("org.jbowl.model.supervised.rule.RuleAlgo
rithm");
BuildSettings settings = new RuleSettings();
settings.setMaxNumberOfRules(-1);
settings.setMaxDepth(-1);
settings.setImpurityMeasure(ImpurityMeasure.entropy);
settings.setBinaryValues(false);
task.setBuildSettings(settings);
```

Tree algoritmus (t.j. rozhodovacie stromy)

Umiestnenie: org.jbowl.model.supervised.tree

Parametre (trieda TreeSettings):

- maxDepth: Maximálna hĺbka generovaného stromu. Typ: int. Predvolená hodnota: -1. (t.j. neobmedzená hĺbka)
- impurity: Miera „variability“ pokrytých príkladov, ktorá sa použije pre výber delenia uzla. Typ: objekt org.jbowl.model.supervised.rule.ImpurityMeasure. Predvolená hodnota: entropy.
- binaryValues: Definuje, či sa bude brať do úvahy frekvencia termu (hodnota *false*), alebo iba binárna hodnota či sa term vyskytol/nevyskytol (hodnota *true*). Typ: boolean. Predvolená hodnota: false.

Príklad implementácie:

```
task.setAlgorithm("org.jbowl.model.supervised.tree.TreeAlgo
rithm");
BuildSettings settings = new TreeSettings();
settings.setMaxDepth(-1);
settings.setImpurityMeasure(ImpurityMeasure.entropy);
settings.setBinaryValues(false);
task.setBuildSettings(settings);
```

SVM algoritmus (t.j. Support Vector Machines)

Umiestnenie: org.jbowl.model.supervised.svm

Parametre (trieda SVMSettings):

- Cp, Cn: Parameter. ktorý určuje do akej miery sa bude uprednostňovať kritérium maximálnej vzdialenosti príkladov od deliacej roviny pred chybou klasifikácie spôsobenou nelineárne separovatelnými dátami, Cp určuje tento pomer pre pozitívne príklady a Cn pre negatívne. Nastavením rôznych hodnôt je možné

uprednostniť chyby na pozitívnych resp. negatívnych príkladoch. Typ: double. Predvolená hodnota: 1. (zvyčajne sa nastavuje postupne na 10^{-4}, 10^{-3}, ..., 1, ..., 10^{4}, pričom sa zvolí hodnota testovaná na validačnej množine).

- epsilon: : Požiadovaná presnosť pri riešení optimalizačného problému SVM. Typ: double. Predvolená hodnota: 0.001.
- cacheSize: Percentuálna veľkosť vyrovnávacej pamäte použitej pre uchovávanie vypočítaných hodnôt kernelovej funkcie. Typ: double. Predvolená hodnota: 0.25.
- kernel: Kernelová funkcia. Typ: objekt org.jbowl.util.math.Kernel. Predvolená hodnota: LinearKernel.

Príklad implementácie:

```
task.setAlgorithm("org.jbowl.model.supervised.svm.SVMAlgori
thm");
BuildSettings settings = new SVMSettings();
settings.setCp(1);
settings.setCn(1);
settings.setEpsilon(0.001);
settings.setCacheSize(0.25);
settings.setKernel(Kernel.linearKernel);
task.setBuildSettings(settings);
```

B.1.5.1.5 Vyhodnotenie kvality klasifikačného modelu

Po vytvorení klasifikačného modelu je vhodné overiť jeho kvalitu porovnaním s vopred kategorizovanými dokumentami z testovacej množiny. Predpokladom je, že testovacia množina bola indexovaná tým istým spôsobom, ako množina trénovacia. Potom po dokončení tvorby klasifikačného modelu, čiže po volaní funkcie `handler.waitForCompletion()`, môže nasledovať vyhodnotenie kvality vytvoreného modelu, realizované napríklad v implementácii triedy `EvaluateModel`. Funkcia `main(argv)` tejto triedy potom bude obsahovať nasledujúcu postupnosť príkazov:

Najprv sa inicializuje objekt `task` reprezentujúci úlohu vyhodnotenia klasifikačného modelu voči indexovanej testovacej množine:

```
EvaluateModelTask task = new EvaluateModelTask();
```

Nastavia sa príslušné vstupné parametre. Funkcia `setEvaluationData()` nastaví fyzickú cestu k súboru, ktorý obsahuje indexovanú testovaciu množinu:

```
task.setEvaluationData(new InstanceInputStream("temp/test-
        instances"));
```

Funkcia `setModelName()` nastaví fyzickú cestu k súboru obsahujúcemu klasifikačný model, ktorého kvalita sa má vyhodnocovať:

```
task.setModelName("temp/model");
```

Funkcia `setEvaluationMetricsName()` nastaví fyzickú cestu k súboru, do ktorého sa zapíšu údaje vyhodnotenia kvality modelu:

```
task.setEvaluationMetricsName("temp/model-eval-metrics");
```

Získa sa pripojenie na vykonávací objekt:

```
Connection conn =
        ConnectionFactory.createFactory().getConnection();
```

Iniciuje sa vykonanie úlohy na vyhodnotenie klasifikačného modelu:

```
ExecutionHandler handler = conn.execute(task);
```

Čaká sa na dokončenie úlohy vyhodnotenia:

```
handler.waitForCompletion();
```

Do objektu `metrics` sa načítajú vypočítané údaje o vyhodnotení:

```
PrecisionRecallMetrics metrics =
        (PrecisionRecallMetrics)Utils.readObject(task.getEv
        aluationMetricsName());
```

Z objektu `metrics` sa získajú objektové reprezentácie metrík mikro a makro-spriemerňovania:

```
PrecisionRecall micro = metrics.microAverage();
PrecisionRecall macro = metrics.microAverage();
```

Pomocou objektov `micro` a `macro` sú prístupné nasledujúce údaje o globálnych mierach kvality klasifikačného modelu:

- Mikro presnosť: `(double) micro.precision()`
- Mikro návratnosť: `(double) micro.recall()`
- Mikro miera F1: `(double) micro.F1()`
- Makro presnosť: `(double) macro.precision()`
- Makro návratnosť: `(double) macro.recall()`
- Makro miera F1: `(double) macro.F1()`

Objekt `metrics` obsahuje zoznam objektov typu `PrecisionRecall`, zodpovedajúcich jednotlivým kategóriám, do ktorých boli na začiatku zaradené dokumenty z testovacej množiny. Výpis hodnôt dosiahnutých mier kvality pre jednotlivé kategórie možno získať napríklad pomocou funkcie `printAccuracyStats(metrics, "temp/model-eval-metrics.txt")`, ktorej prvým parametrom je naplnený objekt `metrics` a druhým parametrom je fyzická cesta k textovému súboru, do ktorého sa zapíšu hodnoty mier kvality. Implementácia tejto funkcie má tvar:

```
static void printAccuracyStats(List<PrecisionRecall> stats,
        String file) throws IOException {
    PrintStream out = new PrintStream(new
            FileOutputStream(file));
    out.println("TP\tFP\tTN\tFN\tprecision\trecall\tF1");
    for (PrecisionRecall pr : stats) {
        if (pr != null) {
            out.print(pr.truePositive() + "\t");
            out.print(pr.falsePositive() + "\t");
            out.print(pr.trueNegative() + "\t");
            out.print(pr.falseNegative() + "\t");
            out.print(pr.precision() + "\t");
            out.print(pr.recall() + "\t");
            out.print(pr.F1() + "\t");
        } else {
            out.print("-\t-\t-\t-\t-\t-");
        }
        out.println();
    }
    out.close();
}
```

Výsledkom tohto výpisu je tabuľka, v ktorej riadky zodpovedajú klasifikačným kategóriám (resp. klasifikačným triedam). Sedem stĺpcov má pre príslušnú kategóriu nasledujúci význam:

- `pr.truePositive()`: správne predikované pozitívne príklady,
- `pr.falsePositive()`: nesprávne predikované pozitívne príklady,
- `pr.trueNegative()`: správne predikované negatívne príklady,
- `pr.falseNegative()`: nesprávne predikované negatívne príklady,
- `pr.precision()`: presnosť,
- `pr.recall()`: úplnosť,
- `pr.F1()`: miera F1.

Na záver, po spracovaní (získaní, vypísaní, a pod.) hodnôt vyjadrujúcich kvalitu klasifikačného modelu, je potrebné uvoľniť vykonávací objekt:

```
TaskExecutor.shutdown();
```

Ak získané hodnoty kvality klasifikačného modelu nie sú v akceptovateľnej tolerancii, je potrebné zmeniť parametre klasifikačného algoritmu alebo použiť iný dostupný algoritmus. Proces sa opakuje od tvorby klasifikačného modelu s novými nastaveniami, pričom sa znova klasifikujú všetky dokumenty z trénovacej aj testovacej množiny.

B.1.5.1.6 Klasifikácia

Ak je vytvorený klasifikačný model vykazujúci požadovanú úroveň hodnôt kvality, možno ho použiť na klasifikáciu ďalších textových dokumentov. Tieto dokumenty by mali byť obsahovo zodpovedajúce dokumentom z trénovacej množiny, na základe ktorých bol vytvorený klasifikačný model. Na výstupe procesu klasifikácie sa získajú kategórie odporúčané pre dané vstupné

dokumenty, pričom celkový zoznam kategórií je znova určený trénovacou množinou (t.j. model klasifikuje iba do tých kategórií, do ktorých boli zaradené dokumenty v trénovacej množine).

Systém JBowl vyžaduje, aby aj dokumenty určené na klasifikáciu boli reprezentované v špecifickom vstupnom formáte XML (porov. vyššie v časti B.1.5.1.1 Príprava vstupu). Keďže v tomto prípade, na rozdiel od trénovacej či testovacej množiny, zaradenie dokumentov do kategórií nie je známe (resp. je práve tým, čo má byť zistené v procese klasifikácie), vstupný XML formát nebude obsahovať XML elementy `category`. Ďalšie elementy formátu ostávajú rovnaké ako pri spracovaní dokumentov z trénovacej a testovacej množiny.

Nasledujúci príklad ukazuje spôsob implementácie procesu klasifikácie neznámych (t.j. vopred nekategorizovaných) dokumentov pomocou existujúceho klasifikačného modelu v systéme JBowl:

Dokumenty v XML formáte treba najprv predspracovať, vykonať jazykovú analýzu a indexovať. Pritom spôsob predspracovania a nastavení jazykovej analýzy musí byť rovnaký ako spôsob použitý pri predspracovaní a analýze dokumentov z trénovacej množiny. Iba pri dodržaní tejto podmienky budú výsledky klasifikácie korektné.

Na začiatku indexácie pomocou systému JBowl sa vytvorí dočasný súbor `tmp`, ktorý slúži ako úložisko údajov počas procesu indexácie:

```
File tmp = File.createTempFile("tmp", null);
tmp.deleteOnExit();
```

Deklarujú sa objekty pre indexovacie množiny termov, kategórií a štatistík, ktoré sa inicializujú na základe existujúcich súborov vytvorených pri indexácii trénovacej množiny. Vstupnými parametrami pre funkcie `readObject()` triedy `Utils` sú fyzické cesty k príslušným súborom:

```
IndexedSet<String> terms = (HashIndexedSet<String>)
        Utils.readObject("temp/term-dictionary");

IndexedSet<String> categories = (HashIndexedSet<String>)
        Utils .readObject("temp/category-dictionary");

Statistics trainStats = (Statistics)
        Utils.readObject("temp/train-stats");
```

Deklaruje sa objekt `testStats` pre výstupné štatistiky spracovávaných dokumentov (t.j. vstupných dokumentov, ktoré sa majú klasifikovať). Vstupom vnorenej funkcie `TFIDFFilter` musí byť reťazec `TFIDF_SCHEME` s rovnakým nastavením, aké bolo použité pri indexácii trénovacej množiny. Druhým vstupným parametrom funkcie `TFIDFFilter` je objekt `trainStats` obsahujúci štatistiky trénovacej množiny:

```
Statistics testStats = new Statistics(
        new TFIDFFilter(TFIDF_SCHEME, trainStats,
        new Utils.ProgressLogger(
        new InstanceOutputStream(tmp))));
```

Vytvorí sa indexovací objekt `indexer`, ktorého vstupným parametrom je nastavenie predspracovania a jazykovej analýzy, implementované vo funkcii `createTextAnalyzer()`. Aj táto funkcia musí byť rovanká ako funkcia použitá pri jazykovej analýze trénovacej množiny. Ďalšími vstupnými parametrmi sú indexovacie množiny termov `terms` a kategórií `categories`, a tiež objekt `testStats` pre výstupné štatistiky:

```
XMLDocumentIndexer indexer = new XMLDocumentIndexer(
        new DocumentAnalyzer(createTextAnalyzer(),
        new DocumentIndexer(terms,categories,testStats)));
```

Následne sa vykoná proces samotnej indexácie volaním funkcie `indexFile()` indexovacieho objektu. Vstupným parametrom funkcie je cesta k súboru, ktorý obsahuje XML reprezentácie dokumentov určených na klasifikáciu – v našom prípade je to data/in.xml:

```
indexer.indexFile("data/in.xml");
indexer.close();
```

Indexácia je v tomto kroku skončená a nasleduje proces klasifikácie. Deklaruje sa objekt `classifier`, ktorý sa inicializuje na existujúci klasifikačný model vytvorený z trénovacej množiny. Vstupným parametrom funkcie `readObject` objektu `Utils`, volanej v konštruktore triedy `ClassificationModel`, je fyzická cesta k súboru obsahujúcemu daný klasifikačný model:

```
ClassificationModel classifier = (ClassificationModel)
        Utils.readObject("temp/model");
```

Deklaruje sa zoznam indexovacích inštancií `instances` a inicializuje sa na dočasné úložisko indexačných údajov, ktorým je súbor `tmp`. Tento súbor už obsahuje objektovú reprezentáciu indexu vstupných dokumentov, ktoré sa majú klasifikovať (napr. počet vstupných dokumentov sa dá získať volaním funkcie `instances.size()`):

```
Instances instances = new Instances(new
        InstanceInputStream(tmp));
```

Zoznam indexovacích inštancií `instances` obsahuje položky zodpovedajúce vektorom termov jednotlivých dokumentov zo spracovávaného vstupného súboru data/in.xml. Klasifikácia týchto dokumentov sa realizuje v cykle pre všetky položky zoznamu `instances`. Klasifikáciu jedného dokumentu (t.j. jemu príslušného vektora termov, uloženého v premennej `instance`) vykoná funkcia `classifier.classify()`, ktorá vo výsledku vráti zoznam `classification` identifikátorov tých kategórií, do ktorých bol dokument klasifikovaný na základe použitého klasifikačného modelu. Zoznam všetkých kategórií modelu obsahuje indexovacia množina `categories`. Názvy tých kategórií, do ktorých bol zaradený jeden zo vstupných dokumentov, možno získať volaním funkcií `categories.get(classification.get(i))` pre všetky prvky

zoznamu `classification`:

```
for (VectorInstance instance : instances) {
    CategorySet classification =
              classifier.classify(instance);
    for (int i = 0; i < classification.size(); i++) {
         System.out.println("Klasifikacna trieda: " +
                   categories.get(classification.get(i)));
     }
}
```

B.1.5.2 Nekontrolované učenie – zhlukovanie dokumentov

Cieľom úlohy zhlukovania je nájsť vo vstupnej množine (korpuse) textových dokumentov určité „skryté“, explicitne nevyjadrené kategórie, zhluky či skupiny istým spôsobom vzájomne podobných dokumentov, pričom ich podobnosť je daná sémantickou blízkosťou obsahov ich textov. Na rozdiel od klasifikácie, zhlukovanie je príkladom nekontrolovaného učenia. Kým pri klasifikačných úlohách je tvorba príslušného modelu kontrolovaná, riadená vopred známou trénovacou množinou úkážkových príkladov, pri zhlukovaní sa nevyužíva žiadna apriórna informácia.

Úloha zhlukovania textových dokumentov sa v systéme JBowl vykonáva v etapách:

1. príprava vstupu, ktorá zahŕňa:
   - formátovanie vstupnej množiny dokumentov,
   - predspracovanie a jazykovú analýzu,
   - indexáciu vstupnej množiny,
2. tvorba modelu zhlukovania,
3. tvorba popisného modelu zhlukov, vizualizácia a analýza získanej štruktúry zhlukov,
4. použitie vytvorených modelov zhlukovania.

Pre každú z týchto etáp sú ďalej popísané spôsoby aplikovania systému JBowl, vrátane implementačných detailov a ukážok Java kódu.

B.1.5.2.1 Príprava vstupu

Etapa prípravy vstupu pre zhlukovanie je veľmi podobná príprave vstupu pre klasifikáciu, popísanej v časti B.1.5.1.1. Sú tu však dva podstatné rozdiely, a to:

1. vstup tvorí iba jedna množina textových dokumentov (kým pri klasifikačnej úlohe bol vstup tvorený dvoma množinami – trénovaou a testovacou),
2. dokumenty vo vstupnej množine nie sú vopred klasifikované – naopak, cieľom zhlukovania je identifikovať skupiny vzájomne podobných dokumentov, ktoré sa až následne môžu považovať za klasifikačné kategórie.

Systém JBowl vyžaduje pre úlohu zhlukovania reprezentáciu vstupnej množiny v špecifickom XML formáte, ktorý je v zásade rovnaký ako pre klasifikáciu. Vstupná množina je XML súbor pozostávajúci z dokumentov označených párovým XML elementom `document`, ktorý je ďalej členený elementmi `title` a `text`. Na rozdiel od klasifikácie však dokumenty vo vstupnej množine nie sú vopred kategorizované, preto neobsahujú elementy `category`. XML reprezentácia jedného dokumentu vo vstupnej množine pre úlohu zhlukovania sa teda predpokladá v tvare:

```
<document id="19" name="Reuters-21578-19" dataset="ModApte-
train">
<title>BONUS WHEAT FLOUR FOR NORTH YEMEN  - USDA</title>
<text>
The Commodity Credit Corporation, CCC, has accepted an
export bonus offer to cover the sale of 37,000 long tons of
wheat flour to North Yemen, the U.S. Agriculture Department
said.
...
</text>
</document>
```

Ďalšie spracovanie vstupnej množiny dokumentov je identické s postupom používaným pri klasifikačných úlohách tak, ako to bolo uvedené v časti B.1.5.1.1. Počas jazykovej analýzy sa vykonáva členenie textu na slová, tokenizácia, lematizácia a odstránenie neplnovýznamových slov. Termy reprezentujúce jednotlivé dokumenty sú následne váhované a normované. Napokon sa vektory termov indexujú volaním funkcie `indexFile()` indexovacieho objektu, ktorý bol inicializovaný vstupnými parametrami príslušných nastavení predspracovania a jazykovej analýzy. Parametrom funkcie `indexFile()` je fyzická cesta k súboru obsahujúcemu XML reprezentácie dokumentov vstupnej množiny, napríklad:

```
indexer.indexFile("data/Reuters-21578_ModApte/input.xml");
```

Vytvorí sa objektová reprezentácia matice term-dokument, ktorá sa v kódovanom tvare uloží do súboru určeného vstupným parametrom funkcie `InstanceOutputStream()` – v našom prípade do súboru temp/input-instances:

```
new InstanceInputStream(tmp).processAll(
        new TFIDFFilter(TFIDF_SCHEME, inputStats,
        new Utils.ProgressLogger(
        new InstanceOutputStream("temp/input-
instances"))));
```

Index termov `terms` a vypočítané štatistiky `inputStats` sa uložia do súborov pomocou funkcie `writeObject` objektu `Utils`:

```
Utils.writeObject(inputStats, "temp/input-stats");
Utils.writeObject(terms, "temp/term-dictionary");
```

Index kategórií `categories` je pri úlohe zhlukovania prázdny, pretože

dokumenty vo vstupnej množine neboli kategorizované – zaradenie dokumentov do kategórií je práve cieľom úlohy zhlukovania a vykoná sa v nasledujúcej fáze, pomocou príslušného modelu vytvoreného zvoleným zhlukovacím algoritmom.

B.1.5.2.2 Vytvorenie modelu zhlukovania

Z indexovanej vstupnej množiny sa pomocou zvoleného zhlukovacieho algoritmu vytvorí model, ktorý rozdelí dokumenty do tried (zhlukov) podľa podobnosti vektorov termov vytvorených z textov dokumentov. Procedúra tvorby zhlukovacieho modelu má v systéme JBowl tieto vstupné parametre:

- zhlukovací algoritmus a jeho nastavenia,
- cesta k súboru obsahujúcemu indexovanú vstupnú množinu,
- cesta k súboru, do ktorého sa zapíše vytvorený zhlukovací model.

Nasledujúci príklad demonštruje spôsob, akým sa v systéme JBowl vytvorí zhlukovací model pomocou algoritmu *K-means*. Najprv sa inicializuje objekt `task` reprezentujúci úlohu vytvorenia zhlukovacieho modelu:

```
BuildModelTask task = new BuildModelTask();
```

Vytvorí sa objekt `settings`, v ktorom sa nastavia parametre zvoleného zhlukovacieho algoritmu. Typ objektu `settings` je daný použitým algoritmom. Pre algoritmus *K-means* sa nastaví žiadaný počet zhlukov *K* na 10, maximálny počet cyklov učenia sa algoritmu na 10 a metrika výpočtu podobnosti vektorov termov na kosínusovú podobnosť:

```
KMeansSettings settings = new KMeansSettings();
settings.setK(10);
settings.setMaxNumberOfCycles(10);
settings.setSimilarity(Similarity.cosineSimilarity);
```

Pomocou funkcie `setAlgorithm()` objektu `task` sa nastaví algoritmus zhlukovania. Argumentom tejto funkcie je názov úplnej cesty k implementácii objektu reprezentujúceho daný algoritmus – v tomto prípade je ním reťazec "org.jbowl.model.unsupervised.kmeans.KMeansAlgorithm":

```
task.setAlgorithm("org.jbowl.model.unsupervised.kmeans.KMea
nsAlgorithm");
```

Funkcia `setBuildData()` nastaví fyzickú cestu k súboru, ktorý obsahuje indexovanú vstupnú množinu dokumentov:

```
task.setBuildData(new InstanceInputStream("temp/input-
instances"));
```

Funkcia `setBuildSettings()` priradí nastavené parametre zhlukovacieho algoritmu k objektu `task` reprezentujúcemu úlohu zhlukovania:

```
task.setBuildSettings(settings);
```

Fyzická cesta k súboru, do ktorého sa zapíše vytvorený zhlukovací model, sa nastaví pomocou funkcie `setModelName()`:

```
task.setModelName("temp/modelKMeans");
```

Získa sa pripojenie na vykonávací objekt `conn`, iniciuje sa vykonanie úlohy `task` a čaká sa na dokončenie vytvorenia modelu zhlukovania:

```
Connection conn =
ConnectionFactory.createFactory().getConnection();
ExecutionHandler handler = conn.execute(task);
handler.waitForCompletion();
```

Na zhlukovanie textových dokumentov poskytuje systém JBowl viacero zhlukovacích algoritmov, ktorých implementácie sa nachádzajú v balíku org.jbowl.model.unsupervised. Okrem K-stredového algoritmu *K-means* je to najmä prístup založený na samoorganizujúcich sa mapách (*self-organizing maps*, SOM) [Kohonen 1995]. Toto riešenie je vhodné pre mnohorozmerné údaje, aké sa produkujú pri úlohách dolovania v textoch. Metóda SOM je nelineárne zobraznenie, ktoré transformuje mnohorozmernú kolekciu vstupných údajov na dvojrozmernú výstupnú mapu. Istou nevýhodou metódy SOM je jej statická architektúra, preto bola do knižnice JBowl doplnená aj implementácia metódy GHSOM (*Growing Hierarchical SOM*) s modifikovaným algoritmom, zamedzujúcim problémom s rastom mapy a inicializáciou nových vrstiev [Buka 2003]. Centroidné modely (algoritmy) sa implementujú cez triedu CentroidClustering, hierarchické metódy využívajúce hierarchickú dekopozíciu dát (ako GHSOM) si naviac vyžadujú implementáciu na základe triedy HierarchicalAlgorithm. Príklad algoritmu v oficiálnej verzii je K-means,

K-means algoritmus

Umiestnenie: org.jbowl.model.unsupervised.kmeans

Parametre (trieda KMeansSettings):

- K: žiadaný počet zhlukov vo výslednom modeli. Typ: int. Predvolená hodnota: 2.
- maxNumberOfCycles: maximálny počet cyklov učenia sa algoritmu. Typ: int. Povolené hodnoty: -1 (učenie končí v momente, keď nedošlo k zmene priradení v rámci aktuálneho cyklu) alebo kladné celé číslo (presný počet cyklov). Predvolená hodnota: -1.
- similarity: použitá metrika pre porovnanie dokumentov. Typ: objekt org.jbowl.util.math.Similarity. Predvolená hodnota: cosineSimilarity.

Hierarchical algoritmus

Umiestnenie: org.jbowl.model.unsupervised

Dekomponujúce hierarchické algoritmy (ako GHSOM) implementujú dedením triedu HierarchicalAlgorithm a upravujú ju podľa svojej potreby. Algoritmus

GHSOM bol implementovaný ako HierarchicalAlgorithm dekomponujúci vstupujúce údaje a vytvárajúci lokálne mapy (SOM) v hierarchickej štruktúre.

Parametre (trieda HierarchicalBuildSettings):

- maxDepth: maximálna hĺbka – maximálny počet vrstiev v hierarchizácii výsledného modelu. Typ: int. Predvolená hodnota: nastavuje sa pre konkrétnu implementáciu hierarchického algoritmu.
- minNumberOfInstances: minimálny počet inštancií – počet príkladov potrebných pre vytvorenie novej inštancie algoritmu pri hierarchickej dekompozícii na novú úroveň. Typ: int. Predvolená hodnota: nastavuje sa pre konkrétnu implementáciu hierarchického algoritmu.

B.1.5.2.3 Vytvorenie popisného modelu zhlukov

Model zhlukovania, vytvorený v predchádzajúcej etape, predpisuje rozdelenie dokumentov zo vstupnej množiny do zhlukov – skupín, tried vzájomne podobných dokumentov, pričom podobnosť je daná matematickou blízkosťou vektorov termov reprezentujúcich dané dokumenty. Zhluky vo vytvorenom modeli sú však určené iba číselným identifikátorom a príslušnosťou vektorov termov k tomu-ktorému zhluku. Takýto spôsob určenia však sťažuje interpretáciu výslednej štruktúry zhlukov. Je žiadúce, aby zhluky v modeli boli popísané spôsobom, ktorý dovoľuje ľahšie určiť a interpretovať ich význam (napr. v zmysle klasifikačných kategórií). Na popis zhlukov vytvorených v modeli sa ponúka

Zhluky vo vytvorenom modeli možno popísať napríklad pomocou výberu najvýznamnejších[27] termov z vektorov tých dokumentov, ktoré do daného zhluku patria. Na určenie týchto termov a následný popis zhlukov sa v systéme JBowl využíva tzv. popisný model, ktorý je implementovaný v triedach balíka org.jbowl.model.descriptive.

Popisný model zahŕňa metódy na extrakciu charakteristických termov k danému zhluku, resp. ku klasifikačnej kategórii[28]. Je to doplnková metóda pre popis zhlukov alebo kategórií, nie skutočná metóda učenia. Termy sa v rámci zhluku (kategórie) vyhodnocujú na základe rôznych implementácií triedy TermEvaluator z balíka org.jbowl.data.termselection (napr. InfoGainEvaluator, MutualInfoEvaluator, ChiStatisticEvaluator, a pod.)

Nasledujúci príklad ukazuje, akým spôsobom sa v systéme JBowl vytvorí popisný model zhlukov pre už existujúci model zhlukovania. Inicializuje sa objekt `taskLabeling` reprezentujúci úlohu vytvorenia popisného modelu:

```
BuildModelTask taskLabeling = new BuildModelTask();
```

[27] T.j. najfrekventovanejších, ohodnotených najväčšou váhou, najbližších k stredu zhluku, a pod.

[28] Popisný model sa dá, okrem zhlukovania, použiť aj pri úlohách klasifikácie. Slovne označené a vopred známe klasifikačné kategórie sa môžu dodatočne, po vytvorení klasifikačného modelu, ohodnotiť charakteristickými termami z textov tých dokumentov, ktoré boli do danej kategórie zaradené. Takýto popis kategórií bližšie určuje skutočný obsah klasifikačných kategórií, interpretuje kategórie vzhľadom k množine klasifikovaných dokumentov.

Deklaruje sa objekt `modelForLabeling`, ktorý je typu `ClassificationModel`. Tento objekt sa inicializuje na existujúci zhlukovací model, ktorý je v našom prípade vytvorený pomocou algoritmu K-Means a je uložený v súbore temp/modelKMeans. Vstupným parametrom funkcie `readObject` objektu `Utils`, volanej v konštruktore triedy `ClassificationModel`, je fyzická cesta k súboru obsahujúcemu daný zhlukovací model:

```
ClassificationModel modelForLabeling =
        (ClassificationModel)
        Utils.readObject("temp/modelKMeans");
```

Vytvorí sa objekt `settingsLabeling`, ktorý reprezentuje parametre tvorby popisného modelu. Tieto parametre sa nastavia na príslušné hodnoty – vstupným modelom bude objektová reprezentácia existujúceho zhlukovacieho modelu `modelForLabeling`, maximálny počet popisovaných termov pre jeden zhluk sa nastaví na hodnotu 10. Spôsob vyhodnotenia sa nastaví na úplnú cestu k objektovej reprezentácii použitého popisného algoritmu. V tomto prípade je ním algoritmus evaluácie informačného zisku, implementovaný v objekte `InfoGainEvaluator` z balíka org.jbowl.data.termselection:

```
TermEvaluatorLabelingSettings settingsLabeling = new
        TermEvaluatorLabelingSettings();
settingsLabeling.setInputModel(modelForLabeling);
settingsLabeling.setMaxNumberOfTerms(10);
settingsLabeling.setEvaluator("org.jbowl.data.termselection
        .InfoGainEvaluator");
```

Funkcia `setAlgorithm()` objektu úlohy `taskLabeling` nastaví algoritmus tvorby popisného modelu. Argumentom funkcie je názov úplnej cesty k implementácii objektu reprezentujúceho daný algoritmus – v tomto prípade je ním reťazec "org.jbowl.model.descriptive.labeling.TermEvaluatorLabeling ":

Nasledujúci príklad ukazuje, akým spôsobom sa v systéme JBowl vytvorí popisný model zhlukov pre už existujúci model zhlukovania. Inicializuje sa objekt `taskLabeling` reprezentujúci úlohu vytvorenia popisného modelu:

```
taskLabeling.setAlgorithm("org.jbowl.model.descriptive.
        labeling.TermEvaluatorLabeling");
```

Funkcia `setBuildData()` nastaví fyzickú cestu k súboru, ktorý obsahuje maticu term-dokument, t.j. indexovanú vstupnú množinu tých dokumentov, z ktorých sa vytváral zhlukovací model:

```
taskLabeling.setBuildData(new InstanceInputStream
        ("temp/input-instances"));
```

Nastavia sa parametre použitého popisného algoritmu a funkciou `setModelName()` sa nastaví fyzická cesta k súboru, do ktorého sa zapíše

vytvorený popisný model:

```
taskLabeling.setBuildSettings(settingsLabeling);
taskLabeling.setModelName("temp/labelingOfKMeansModel");
```

Získa sa pripojenie na vykonávací objekt:

```
Connection connLabeling = ConnectionFactory.
        createFactory().getConnection();
```

Iniciuje sa vykonanie úlohy na vytvorenie popisného modelu a čaká sa na dokončenie úlohy:

```
ExecutionHandler handlerLabeling =
        connLabeling.execute(taskLabeling);
handlerLabeling.waitForCompletion();
```

Na tomto mieste je popisný model zhlukovania už vytvorený a jeho objektová reprezentácia je uložená v súbore temp/labelingOfKMeansModel.

Nasledujúca sekvencia príkazov vypíše zoznam zhlukov popísaných príslušnými termami. Najprv sa zo súboru načíta popisný model zhlukovania a inicializuje sa objektová reprezentácia modelu `labelingOfKMeansModel`:

```
LabelingModel labelingOfKMeansModel = (LabelingModel)
        Utils.readObject("temp/labelingOfKMeansModel");
```

Do premennej `terms` sa načíta slovník termov, vytvorený počas indexácie vstupnej množiny dokumentov. Tento slovník obsahuje zoznam skutočných názvov termov, ktoré sa zobrazia vo výpise (kým popisný model obsahuje iba číselné identifikátory termov):

```
IndexedSet<String> terms = (IndexedSet<String>)
        Utils.readObject("temp/term-dictionary");
```

Do premennej `stats` sa načítajú výstupné štatistiky popisného modelu zhlukovania, ktoré obsahujú zoznam zhlukov a k nim príslušné zoznamy popisujúcich termov:

```
Statistics stats = labelingOfKMeansModel.getStatistics();
```

Vo vnorenom cykle `for()` sa vypíšu jednotlivé zhluky, ku každému sa zobrazí zoznam popisujúcich termov a ich váh. Na konci sa uvoľní vykonávací objekt volaním funkcie `TaskExecutor.shutdown()`:

```
System.out.println("KMeans vystup - popisane zhluky:");
for(int i = 0; i < labelingOfKMeansModel.numOfCategories();
        i++){
    System.out.println("Zhluk " + i + " (" +
             stats.categoryFrequency(i) +" vektorov)");
    System.out.print("\t Termy[vahy]: ");
    int selectedTerms[] =
             labelingOfKMeansModel.getSelectedTerms(i);
    for(int j = 0; j < selectedTerms.length; j++){
        System.out.print(terms.get(selectedTerms[j]) + "["
                 + labelingOfKMeansModel.getTermEvaluator()
                 .getScore(selectedTerms[j], i) +"] ");
    }
    System.out.println();
}
TaskExecutor.shutdown();
```

Počet zhlukov aj počet termov popisujúcich tieto zhluky bol v kóde pri inicializácii príslušných objektov nastavený na hodnotu 10. Preto sa vo výpise zobrazí desať zhlukov. Výpis jedného zhluku, napríklad zhluku s poradovým číslom 0, má tvar:

```
Zhluk 0 (1065 vektorov)
Termy[vahy]: oil[0.05515479687457972]
net[0.0453068041348027] shr[0.04532077348215835]
ct[0.038027649203457164] qtr[0.03608315406135895]
produc[0.04271174565233335] countri[0.04113800743094426]
quota[0.03646345344861229] barrel[0.04386716275913535]
minist[0.0464106860239078]
```

Tento výpis znamená, že do zhluku č. 0 patrí 1065 dokumentov reprezentovaných príslušnými vektormi termov. Naviac, tento zhluk je popísaný termami {*oil, net, shr, ct, qtr, produc, countri, quota, barrel, minist*}. Tieto termy charakterizujú daný zhluk a môžu sa považovať za reprezentáciu súhrnného obsahu všetkých dokumentov, ktoré do zhluku patria. Inak povedané, možno prehlásiť, že obsah daných 1065 dokumentov patriacich do zhluku č. 0 sa týka slov, z ktorých vznikli termy *oil, net, quota, barrel, minist*, atď. Popisujúce termy týmto spôsobom interpretujú spoločné obsahové črty dokumentov patriacich do daného zhluku.

Termy v popise zhluku sú ohodnotené váhou, ktorá vyjadruje mieru dôležitosti toho-ktorého termu pre celkový popis zhluku. Z uvedeného príkladu je pre popis zhluku č. 0 najvýznamnejším term *oil*, ktorému zodpovedá percentuálna váha 5,52%. Ďalej nasledujú termy *minist* (4,64%), *shr* (4,53%), *net* (4,53%), *barrel* (4,39%), atď. Z takto zoradených termov sa už väčšinou dá formulovať interpretácia, resp. slovný popis zhluku. Pre zhluk č. 0 z uvedeného príkladu by tento slovný popis mohol mať napríklad tvar: „vládne intervencie do vývoja ťažby a cien ropy na svetových trhoch“.

B.1.5.2.4 Použitie modelu zhlukovania

Model zhlukovania sa dá v systéme JBowl priamo použiť aj ako klasifikátor, ktorý zaradí nové dokumenty do vytvorených zhlukov. Postup je veľmi podobný, ako pri klasifikácii pomocou klasifikačného modelu (kap. B.1.5.1.6). Nové dokumenty sa predpokladajú v špecifickom vstupnom formáte XML.

Predspracujú sa metódami jazykovej analýzy, váženia a normovania termov. Pre korektnosť klasifikácie je potrebné pri týchto metódach dodržať rovnaké nastavenia, aké sa použili pri spracovaní vstupnej množiny dokumentov pre vytváranie modelu zhlukovania. Následne sa predspracované nové dokumenty indexujú volaním funkcie `indexFile()`. Parametrom funkcie je fyzická cesta k súboru, ktorý obsahuje XML reprezentácie nových dokumentov určených na klasifikáciu – v tomto prípade je to data/in.xml:

```
indexer.indexFile("data/in.xml");
indexer.close();
```

Objektová reprezentácia indexu, t.j. matice term-dokument, sa uloží do dočasného súboru `tmp`, vopred vytvoreného a nastaveného v objekte `indexer`. Indexácia nových dokumentov je v tomto kroku skončená a nasleduje proces klasifikácie. Deklaruje sa objekt `classifier`, ktorý sa inicializuje na existujúci zhlukovací model, uložený v súbore temp/modelKMeans:

```
ClassificationModel classifier = (ClassificationModel)
        Utils.readObject("temp/modelKMeans");
```

Kvôli prehľadnejšiemu výpisu výsledkov sa vytvorí aj objekt popisného modelu `labelingOfKMeansModel`. Tento sa inicializuje na existujúci popisný model, ktorý bol vytvorený k modelu zhlukovania a je uložený v súbore temp/labelingOfKMeansModel:

```
LabelingModel labelingOfKMeansModel = (LabelingModel)
        Utils.readObject("temp/labelingOfKMeansModel");
```

Deklaruje sa zoznam indexovacích inštancií `instances` a inicializuje sa na dočasné úložisko indexačných údajov, ktorým je súbor `tmp`:

```
Instances instances = new Instances(new
        InstanceInputStream(tmp));
```

Klasifikácia nových dokumentov, ktorých vektory termov sú uložené v súbore `tmp`, sa vykoná v cykle pre všetky položky zoznamu `instances`. Klasifikáciu jedného vektora termov vykoná funkcia `classifier.classify()`, ktorá vo výsledku vráti zoznam `classification` identifikátorov tých kategórií, do ktorých bol dokument klasifikovaný na základe použitého zhlukovacieho modelu. Zoznam všetkých kategórií (zhlukov) modelu, spolu s príslušným zoznamom termov popisujúcich daný zhluk, obsahuje objekt `labelingOfKMeansModel`:

```
for (VectorInstance instance : instances) {
   CategorySet classification =
             classifier.classify(instance);
   for (int i = 0; i < classification.size(); i++) {
        System.out.println("Dokument: " +
                  instance.getSignature());
        int cat = classification.get(i);
        System.out.println("Klasifikacna trieda: " + cat);
        int selectedTerms[] =
                  labelingOfKMeansModel.getSelectedTerms(cat
                  );
        for(int j = 0; j < selectedTerms.length; j++){
            System.out.print(terms.get(selectedTerms[j]) +
                      "[" + labelingOfKMeansModel.getTermEv
                      aluator().getScore(selectedTerms[j],
                      cat) +"] ");
        }
    }
    System.out.println();
}
```

Podobným spôsobom možno klasifikovať dokumeny zo vstupnej množiny, na základe ktorej bol vytvorený zhlukovací model. V tomto prípade sú už dokumenty indexované a ich index je uložený v súbore temp/input-instances. Zoznam `instances` sa inicializuje z tohto súboru:

```
Instances instances = new Instances(new
InstanceInputStream("temp/input-instances"));
```

Následná klasifikácia a výpis výsledkov sa vykoná v cykle pre všetky položky zoznamu `instances`. Kód je rovnaký ako pri klasifikácii nových dokumentov.

B.1.6 Obslužná konzola systému JBowl

Návrh a vývoj obslužnej konzoly systému JBowl bol motivovaný snahou vytvoriť webovské rozhranie, ktoré by sprostredkovalo jednoduché interaktívne a vizuálne použitie základných funkcií systému, zameraných predovšetkým na riešenie úloh klasifikácie a kategorizácie textov. Obslužná konzola bola navrhnutá a realizovaná ako webovská databázová aplikácia typu klient – server, s použitím technológie JSP a databázy MySql [Furdík 2007; Furdik a kol. 2008a]. Implementácia prebiehala v roku 2007 v Centre pre informačné technológie FEI TU v Košiciach, v rámci riešenia projektu PoZnaŤ (viď kap. B.1.3). Začiatkom roka 2008 bola aplikácia obslužnej konzoly spustená do prevádzky a je prístupná na adrese http://cit.fei.tuke.sk:8080/textminingweb/.

Počas zimného sementra školského roka 2008/2009 bola obslužná konzola prvýkrát použitá vo výučbe, konkrétne v predmete Manažment znalostí. Študenti vytvárali tématicky rôzne zamerané korpusy slovenských a anglických textových dokumentov, následne upravovali texty do formy trénovacích a testovacích množín a robili pokusy s vytváraním klasifikačných modelov. Pomocou obslužnej konzoly sa skúmala vhodnosť použitia rôznych klasifikačných algoritmov a ich nastavení pre dané vstupné množiny,

vyhodnocovala sa kvalita klasifikácie vzhľadom k jazyku dokumentov, použitým algoritmom a spôsobom predspracovania textov. Dosiahnuté výsledky, ktoré študenti prezentovali v rámci semestrálnych zadaní, sú zverejnené aj na wiki stránke predmetu http://kplab.fei.tuke.sk/mz/.

B.1.6.1 Štruktúra a funkcionalita obslužnej konzoly

Obslužná konzola poskytuje webovské rozhranie pre úlohy klasifikácie a kategorizácie textových dokumentov, realizované prostriedkami systému JBowl. Táto aplikácia je určená predovšetkým na výskumné účely, najmä na skúmanie vhodnosti rôznych klasifikačných algoritmov a ich nastavení pre klasifikáciu určitého typu textových dokumentov.

Výskumné zameranie aplikácie ovplyvnilo návrh štruktúry rozhrania obslužnej konzoly, pri ktorom sa uplatnil projektovo-orientovaný prístup. Každá samostatná klasifikačná úloha je definovaná ako tzv. *klasifikačný projekt*, ktorý pozostáva z nasledujúcich súčastí:

- množina preddefinovaných klasifikačných kategórií,
- trénovacia množina dokumentov, vopred zaradených do klasifikačných kategórií,
- množina klasifikačných algoritmov a ich nastavení,
- množina dokumentov určených na klasifikáciu, ktoré nie sú zaradené do kategórií.

Na základe takto definovaného klasifikačného projektu poskytuje obslužná konzola systému JBowl nasledujúce funkcie:

1. *Definovanie klasifikačného projektu*, čím sa vymedzí priestor jednu pre klasifikačnú úlohu. Parametrami klasifikačného projektu sú jeho názov a popis. K projektu sa následne definujú jeho zložky – trénovacia množina dokumentov, klasifikačné algoritmy a dokumenty určené na klasifikáciu.
2. V aplikácii môže používateľ defnovať ľubovoľný počet vlastných klasifikačných projektov, avšak iba jeden z projektov smie byť tzv. aktívny – t.j. taký, v ktorom je možné meniť jeho súčasti (trénovaciu množinu, klasifikačné algoritmy, atď.). Používateľ môže ľubovoľne nastavovať a vypínať aktívne projekty.
3. *Špecifikácia klasifikačných kategórií* vo forme jednoúrovňového zoznamu. V prípade použitia zložitejšej, napríklad stromovej štruktúry, sa odporúča definovať klasifikačné kategórie pomocou listových uzlov stromu.
4. *Tvorba trénovacej množiny dokumentov*. Konverzia formátov PDF, MS Word a HTML na čistý text. Import trénovacej množiny z XML formátu alebo z adresárovej štruktúry na disku. Možnosť dopĺňať a upravovať dokumenty v trénovacej množine, vrátane ich kategorizácie.
5. *Vytváranie klasifikačného modelu*. Výber jedného z ôsmich implementovaných klasifikačných algoritmov je možné kombinovať s nastaveniami váhovania a normatlizácie termov. Pre jeden

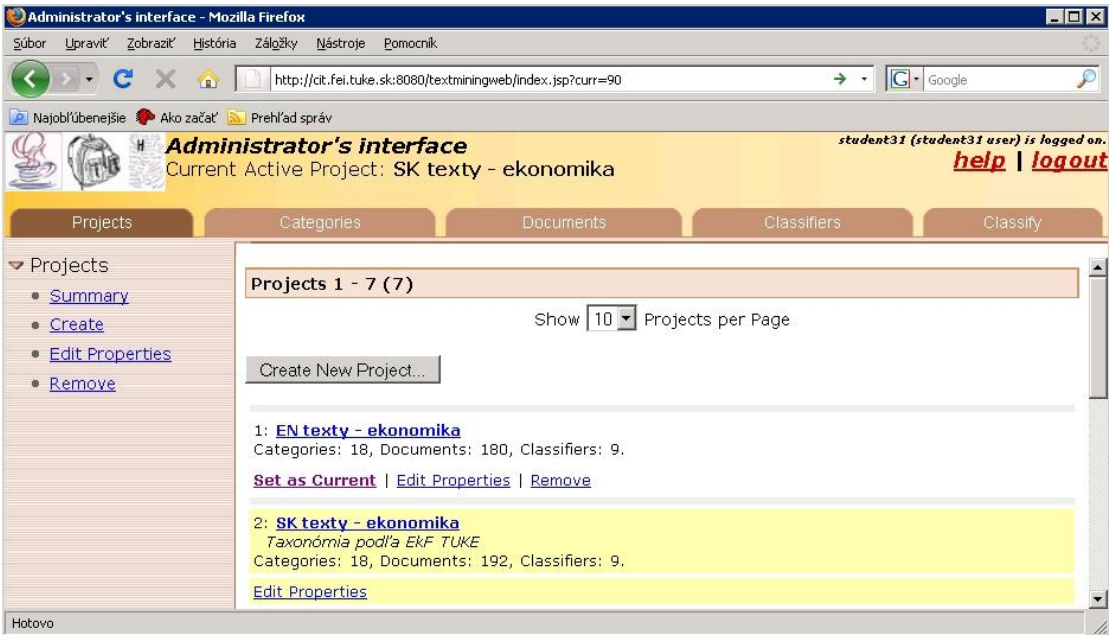
klasifikačný projekt sa môže definovať ľubovoľný počet rôznych klasifikátorov, ktorých nastavenia sa dajú priebežne meniť. Indexácia trénovacej množiny, vytvorenie matice term-dokument a klasifikačného modelu.

6. *Klasifikácia dokumentov*, ktoré nie sú zaradené do kategórií. Podobne ako pri trénovacej množine, podporovaná je konverzia formátov PDF, MS Word a HTML na čistý text. Možnosť dopĺňať a upravovať zoznam dokumentov určených na klasifikáciu. Klasifikácia všetkých dokumentov v zozname pomocou niektorého z vytvorených klasifikačných modelov. Zobrazenie výsledkov klasifikácie v HTML formáte.

Súčasná verzia obslužnej konzoly nepodporuje automatické vyhodnotenie kvality klasifikácie. Vyhodnotenie dosiahnutej miery presnosti, úplnosti, či iných kvalitatívnych mier je potrebné robiť dodatočne, analýzou získaných výsledkov klasifikácie.

B.1.6.2 Ovládanie aplikácie

Po prihlásení sa do aplikácie obslužnej konzoly sa používateľovi zobrazí rozhranie, prezentované na Obr. B.1.2. V hornej časti je päť riadiacich záložiek, ktorými je možné prepínať zobrazenie zoznamu projektov (záložka *Projects*), trénovacej množiny (*Categories* a *Documents*), zoznamu klasifikátorov (*Classifiers*) a nekategorizovaných dokumentov určených na klasifikáciu (*Classify*).



Obr. B.1.2 Zoznam klasifikačných projektov, zobrazený v obslužnej konzole systému JBowl

Na Obr. B.1.2 je vybratá záložka *Projects* a zobrazený je zoznam vytvorených klasifikačných projektov, pričom projekt, ktorý je práve aktívny, je označený žltým pozadím. Zmeniť aktívny projekt sa dá v záložke *Projects*,

kliknutím na linku „Set as Current“ pod príslušným klasifikačným projektom. Po kliknutí na linku „Edit Properties“ je možné meniť názov a popis projektu, a tiež inicializovať trénovaciu množinu z XML súboru alebo z adresárovej štruktúry uloženej na disku servra. Vymazať existujúci klasifikačný projekt sa dá kliknutím na linku „Remove“. Pridať nový projekt dovoľuje tlačidlo „Create New Project“, prípadne linka „Create“ v ľavej časti obrazovky.

Aktívnym projektom na obrázku Obr. B.1.2 je klasifikačný projekt s názvom „SK texty – ekonomika“. To znamená, že zoznamy pod ďalšími štyroma záložkami zobrazia trénovaciu množinu, klasifikátory a dokumenty patriace pod tento aktívny projekt. V záložke *Categories* je možné upravovať jednoúrovňový zoznam klasifikačných kategórií a priradiť ku danej kategórii niektoré z dokumentov trénovacej množiny.

Zoznam dokumentov trénovacej množiny sa zobrazí v záložke *Documents* (Obr. B.1.3). Kliknutím na tlačidlo „Insert New Documents“ možno vkladať dokumenty formátov PDF, MS Word a HTML – pre tieto formáty obsahuje aplikácia prostriedky na extrakciu čistého textu. Pre trénovaciu množinu sa odporúča používať väčší počet (cca 100-200) rozsahom kratších dokumentov (cca do 5 strán textu, s minimom grafiky a obrázkov). Dodržanie týchto odporúčaní pomôže zefektívniť a zrýchliť procesy extrakcie čistého textu a indexácie.

Obr. B.1.3 Zoznam dokumentov trénovacej množiny, zobrazený v obslužnej konzole systému JBowl

Dokumenty v trénovacej množine možno jednotlivo priradiť ku klasifikačným kategóriám kliknutím na linku „Categories (N)“ pod príslušným dokumentom. Po vložení žiadaného počtu dokumentov a po ich priradení ku kategóriám je potrebné celú trénovaciu množinu konvertovať do XML formátu požadovaného systémom JBowl. Táto konverzia sa vykoná kliknutím na tlačidlo „Convert All to XML“. Po konverzii je možné prezrieť si XML formát niektorého z dokumentov kliknutím na jemu príslušnú linku „XML File“.

Ak je trénovacia množina konvertovaná na XML formát, možno v záložke *Classifiers* vytvárať klasifikátory. Po kliknutí na tlačidlo „Create New Classifier“ sa zobrazí formulár, kde sa, okrem názvu a popisu, nastaví klasifikačný algoritmus a parametre predspracovania. Súčasná

implementácia obslužnej konzoly systému JBowl dovoľuje definovať klasifikátor pomocou parametrov a algoritmov uvedených v Tab. B.1.5 (porov. aj s Tab. B.1.2 a Tab. B.1.4 v B.1.5.1):

Tab. B.1.5 Parametre klasifikátorov v obslužnej konzole systému JBowl

Typ TF schémy – frekvencia termov	
b	Binárne váhovanie.
n	Frekvencia termu.
m	Normalizovaná frekvencia termu.
a	Normalizovaná frekvencia termu (normalizácia maximálnou hodnotou, interval 0,5 - 1,0)
Typ IDF schémy – frekvencia dokumentov	
n	Bez inverznej dokumentovej frekvencie.
t	Inverzná dokumentová frekvencia.
p	Pravdepodobnostná inverzná dokumentová frekvencia.
Normovanie	
n	Bez normalizácie.
c	Normalizácia na 1 dĺžku vektora.
Typ značkovania (tokenizácie) – jazyková analýza	
1	jednoduché značkovanie, použitý zoznam
2	jednoduché značkovanie, heuristické členenie viet
3	značkovanie regulárnymi výrazmi
Tvorba modelu – použitý klasifikačný algoritmus	
Perceptron	Rozhodovacie stromy
SVM – Support Vector Machines	Rozhodovacie pravidlá
Lineárny SVM	Boosting
kNN – k najbližších susedov	BoosTexter

Vytvorené klasifikátory sa zobrazia v zozname pod záložkou *Classifiers* (Obr. B.1.4). Pod každým z klasifikátorov, ku ktorému bol úspešne vytvorený klasifikačný model, sú linky na štatistiky frekvencie termov a kategórií v matici term-dokument.

Obr. B.1.4 Zoznam klasifikátorov, zobrazený v obslužnej konzole systému JBowl

Vytvorené klasifikátory sa dajú použiť na klasifikáciu ďalších dokumentov, ktoré sa naplnia do zoznamu pod záložkou *Classify*. Rovnako ako pri trénovacej množine, aj pri dokumentoch určených na klasifikáciu akceptuje obslužná konzola dokumenty vo formátoch PDF, MS Word a HTML. Po vložení do zoznamu sa automaticky vykoná extrakcia čistého textu a konverzia na XML formát.

Ak sú v zozname naplnené všetky požadované dokumenty, možno pristúpiť k ich klasifikácii. Z ponuky v hornej časti sa vyberie niektorý z vytvorených klasifikátorov a klasifikácia sa spustí kliknutím na tlačidlo „Classify All“. Po skončení procesu klasifikácie sa zobrazí linka na HTML dokument obsahujúci výsledky klasifikácie – zoznam názvov dokumentov a kategórií, do ktorých boli dokumenty zaradené po aplikovaní zvoleného klasifikačného modelu. Výsledky klasifikácie sú pre možnosť budúcich porovnaní uložené na disku servra a sú prístupné po kliknutí na linku „History“ v ľavej časti obrazovky pod záložkou *Classify*.

B.2 Realizácia aplikácie založenej na kategorizácii textov

V tejto časti sú použité príklady úloh, ktoré riešia študenti v rámci projektovej práce z predmetu Manažment znalostí na FEI, TU v Košiciach. Stanovené úlohy boli navrhnuté tak, aby pokryli čo najširšie doménu dolovania znalostí z textov a poskytli študentom rôzne pohľady využitia tejto metódy. Ako testovacia platforma bola použitá web aplikácia[29], ktorá poskytuje zdieľaný prístup k rôznym metódam a algoritmom dolovania znalostí z textov, ktoré sú implementované v rámci knižnice JBowl popísanou vyššie v časti B.1. Dosiahnuté výsledky a použité postupy študenti majú študenti za úlohu podrobne dokumentovať na wiki stránkach predmetu Manažment znalostí[30].

B.2.1 Paralelná klasifikácia slovenských a anglických textov

Cieľom úlohy je z textov na Wikipédii (http://www.wikipedia.org) vytvoriť paralelný korpus (súbor) dokumentov v slovenčine a v angličtine (z rôznych oblastí). Ďalej je potrebné zhodnotiť vyváženosť a reprezentatívnosť korpusu. Následne majú študenti vytvoriť klasifikačnú štruktúru pojmov (ontológiu) pokrývajúcu oblasti a témy, do ktorých patria dokumenty vo vytvorenom korpuse.

Korpus je potrebné rozdeliť na testovaciu a trénovaciu množinu pre klasifikáciu (buď manuálnou anotáciou, alebo s použitím heuristických algoritmov), a to osobitne pre slovenské a pre anglické texty. Pomocou web aplikácie sa ďalej vytvoria príslušné klasifikátory na trénovacej množine. Pritom je možné použiť rôzne klasifikačné algoritmy s rôznymi nastaveniami ich parametrov, ako aj parametrov predspracovania, napr. parametrov pre indexáciu a váhovanie.

Následne sa vytvorené klasifikátory testujú na testovacej množine, a to osobitne pre slovenské a pre anglické texty. Výsledky je potrebné vyhodnotiť, okomentovať a zdôvodniť. Pri vyhodnotení je potrebné diskutovať vhodnosť nastavení parametrov predspracovania pre texty v slovenčine a v angličtine a analyzovať možné spôsoby zlepšenia kvality klasifikácie vzhľadom na jazyk klasifikovaných textov.

Postup riešenia:

1. Prvým krokom je oboznámiť sa s doménou dolovania znalostí z textov, ako aj s aplikáciou, prostredníctvom ktorej sa budú vykonávať experimenty.
2. Druhým krokom je vytvorenie klasifikačnej schémy, ktorá dostatočne popisuje skúmanú doménu. V tomto prípade ide o rôzne zamerané články z webového portálu Wikipédia. Vytvorená klasifikačná schéma umožní klasifikáciu získaných textov do vopred stanovených kategórií z tejto schémy.

[29] http://cit.fei.tuke.sk:8080/textminingweb/login.jsp
[30] http://kplab.fei.tuke.sk/mz/Main_Page

V tomto prípade študenti vytvorili dvojúrovňovú schému, kde koreňové uzly predstavovali hlavné kategórie a listové uzly podkategórie. Hlavné kategórie predstavovali: Humanitné vedy, Počítače, Prírodné vedy a Šport.

Najviac podkategórií obsahovala trieda „Počítače“ – 16, najmenej „Humanitné vedy“ – tri, viď. Obr. B.2.1.

Obr. B.2.1 Navrhnutá klasifikačná schéma

3. Pri vytváraní zdrojového korpusu je možné využiť dva prístupy. Motivačným aspektom pri výbere z týchto dvoch metód sú charakteristiky kvality korpusu, t.j. vyváženosť a reprezentatívnosť korpusu. Reprezentatívnosť hovorí o počte textových dokumentov, textových slov, spôsobe pokrytia domény, vyváženosti o variabilite dokumentov. Spravidla je dobré vhodne kombinovať oba nižšie uvedené prístupy. Dôležitou otázkou je aj samotný rozsah korpusu, čiže koľko textov by mal obsahovať. V analyzovanom prípade boli použité 4 hlavné kategórie a okolo 30 podkategórií v rámci klasifikačnej schémy a každá podkategória obsahovala 10-20 textov o veľkosti 1-5 strán.
   - Automatický predpokladá využitie rôznych typov počítačových programov nazvaných web crawler. Pohyb takéhoto programu v rámci stránky je smerovaný pomocou odkazov definovaných v štruktúre stránky. Web crawleru sa zadefinuje počiatok cesty a on sa bude ďalej sám navigovať v štruktúre a snažiť sa získať z každej stránky tzv. čistý text. Tento získaný čistý text by mal anotovať, čiže popísať pomocou metadát uložených v hlavičke stránky. Takto získané dáta sa ukladajú do pripraveného úložiska v navrhnutej štruktúre spoločne s anotáciami, url adresou stránky, textom, príp. surovou formou html pre ďalšiu analýzu.
   - Druhý prístup je podstatne prácnejší, keďže zahŕňa manuálne sťahovanie článkov a ich transformáciu do požadovanej podoby.

V tomto konkrétnom prípade študenti po niekoľkých experimentoch zvolili druhý postup, najmä vzhľadom na kvalitu výsledku práce s vybraným web crawlerom.

4. Pre správne využitie metód dolovania znalostí z textov, čiže vybraných algoritmov, je nutné rozdeliť pôvodný korpus na trénovaciu a testovaciu množinu. Trénovacia množina textov slúži na vytvorenie klasifikátora, t.j. naučenie daného algoritmu, aby vedel texty z tejto množiny klasifikovať do správnych tried. Tento učiaci cyklus je založený na fakte, že texty v rámci trénovacej množiny majú pridelenú kategóriu, t.j. každý text má prostredníctvom jedného parametra definovanú príslušnosť k danej kategórii. Dokumenty v testovacej množine tento parameter nemajú, ten im je prideľovaný pomocou vytvoreného klasifikátora. Z dôvodu čo najlepšiemu "naučeniu sa" klasifikačného modelu je optimálnym spôsobom čo najväčší počet dokumentov zaradiť do trénovacej množiny. Je však potrebné, aby trénovacia množina obsahovala aspoň 60% dokumentov z celkovej množiny dokumentov. V našom prípade sme sa rozhodli pre deliaci pomer 80% a 20%.

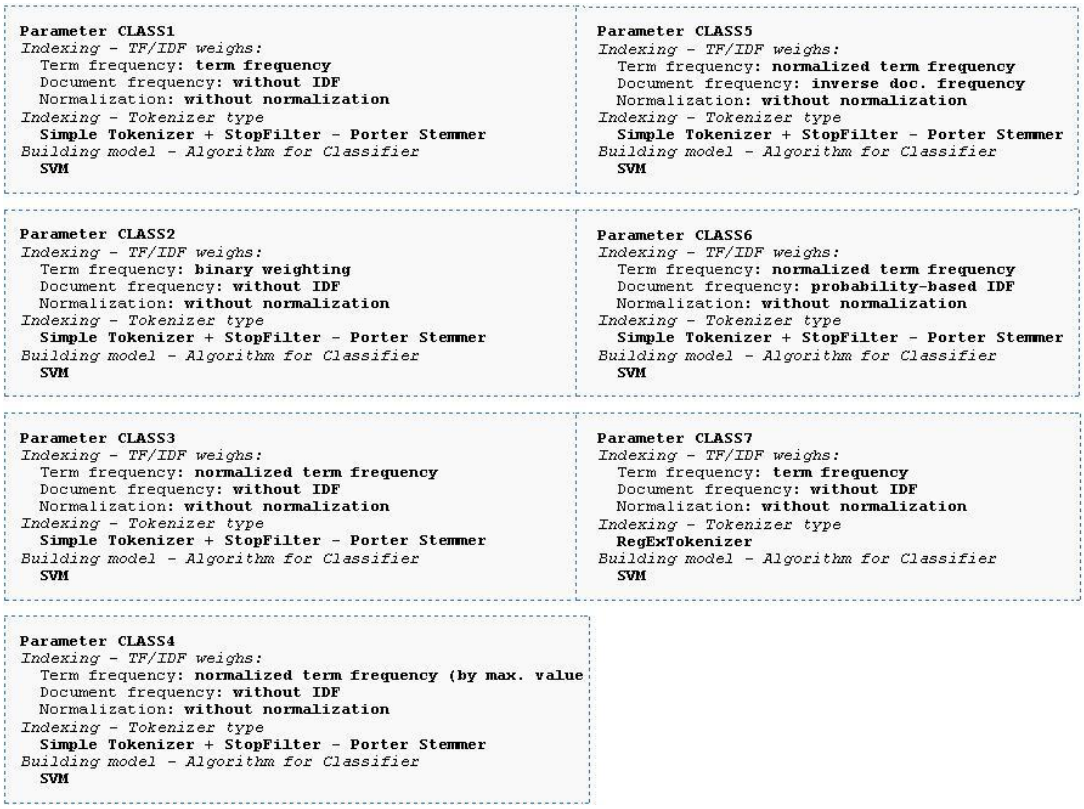
5. Predpokladom vytvorenia klasifikačného modelu je správne kategorizovanie textov trénovacej množine vzhľadom na vytvorenú klasifikačnú schému. Tento proces je možné vykonať automaticky, kde si vytvoríme naraz celú trénovaciu množinu alebo manuálne prostredníctvom webovej aplikácie.

   Vytvorenie trénovacej množiny, čiže rozdelenie vybraných textov do príslušných kategórií, bolo v tomto prípade založené na informáciách, ktoré poskytovali zdrojové texty, čiže bola použitá pôvodná kategorizácia ktorú ponúka samotná wikipédia. Technicky je takýto proces možné realizovať dvoma spôsobmi, t.j. manuálnou kategorizáciou jednotlivých textov prostredníctvom existujúcej aplikácie, čo však predstavuje dosť zložitý proces najmä z časového hľadiska. Druhý prístup poskytuje určitý stupeň komfortu, keďže umožňuje vložiť do aplikácie celý balík textov s priradenými kategóriami naraz. Texty musia byť formátované v niektorom z bežných textových formátov ako doc, pdf alebo plain text. Importovaný korpus je zložený z dvoch častí, kde trénovacia množina predstavuje stromovú štruktúru adresárov, ktoré reprezentujú vytvorenú klasifikačnú schému a jednotlivé adresáre obsahujú príslušné dokumenty.

   V prezentovanom príklade si študenti vybrali možnosť automatického importu, ale pre účely experimentálneho overenia si otestovali aj funkciu manuálnej kategorizácie, ktorá je dostupná priamo v aplikácii.

6. Dôležitým krokom je určiť si metriky, na základe ktorých budeme hodnotiť kvalitu vytvorených klasifikátorov. Používa sa najmä presnosť a návratnosť, prípadne ich kombinácia – metrika F, resp. ich mikro- alebo makro- spriemernené hodnoty (definované boli vyššie v časti 3.3.1).

7. Webová aplikácia pre podporu dolovania znalostí z textov ponúka možnosť výberu reprezentácie textov ako vstupov do klasifikátora. V našom prípade bol použitý vektorový model s *tf-idf* váhovaním. Pri tomto prístupe je nutné nastaviť aj niekoľko ďalších parametrov ako frekvencia termov, frekvencia dokumentu a normalizácia (podrobnejšie vysvetlenie viď. časť 2.7.3). Podobne je potrebné vybrať požadovaný typ klasifikačného algoritmu a nastaviť prípadne jeho parametre. V prípade tejto úlohy sa študenti rozhodli vytvoriť sedem rôznych klasifikátorov, ktoré boli postupne aplikované na oba korpusy, viď. Obr. B.2.2.



Obr. B.2.2 Nastavenia parametrov siedmych klasifikátorov pre anglický korpus

Klasifikátory s rovnakými nastaveniami parametrov boli aplikované aj na slovenský korpus.

8. Prezentácia dosiahnutých výsledkov a ich zhodnotenie. Dosiahnuté výsledky v jednom konkrétnom prípade dokumentuje Tab. B.2.1.

V tomto prípade dosiahol najlepšie výsledky pre oba korpusy klasifikátor č.3, ktorý reprezentuje algoritmus SVM (viď. jeho popis v časti 3.3.2) a pri jeho vytváraní boli použité tie isté parametre ako v prípade vyššie (presné nastavenia sú uvedené na Obr. B.2.2).

Dosiahnuté výsledky boli predmetom záverečnej diskusie, v rámci ktorej študenti zhodnotili celý proces práce na zadaní. Pozitívne hodnotili napríklad možnosť prakticky si odskúšať proces objavovania znalostí z textov

Tab. B.2.1 Príklad dosiahnutých výsledkov pre klasifikáciu slovenských a anglických textov z Wikipédie

	Presnosť (%)		**Návratnosť (%)**	
Typ použitého klasifikátora	Anglický korpus	Slovenský korpus	Anglický korpus	Slovenský korpus
Klasifikátor 1	47	95	49	54
Klasifikátor 2	48	97	47	42
Klasifikátor 3	**96**	**100**	**66**	**58**
Klasifikátor 4	93	**100**	38	43
Klasifikátor 5	93	**100**	56	57
Klasifikátor 6	95	**100**	54	57
Klasifikátor 7	95	82	57	61

prostredníctvom existujúcej aplikácie, založenej na knižnici JBOWL. V tomto prípade však konštatovali niekoľko nedostatkov, ktorých odstránenie by mohlo priniesť jednoduchšiu a úspešnejšiu prácu v budúcnosti, napr. možnosť paralelného spúšťania viacerých experimentov, možnosť automatického importu korpusu zo strany používateľa, možnosť aplikovať viacero algoritmov predspracovania resp. samotnej kategorizácie, alebo detailnejšie informácie o význame jednotlivých parametroch príslušných klasifikátorov.

B.3 Realizácia aplikácie na extrakciu informácií

Cieľom prezentovanej úlohy je analyzovať medicínske články tak, že sa zameriame na skúmanie článkov zaoberajúcich sa rovnakou oblasťou z medicíny (napríklad skúmanie chorôb, popis stavby tela či popis biomolekulárnych tekutín). Pokúsime sa v čo najkratšom čase získať relevantné články z dostupnej množiny a porovnáme rýchlosť manuálneho vyhľadania článkov (čítaním každého článku) a vyhľadávania za pomoci aplikácie na extrakciu entít v texte.

Postup riešenia:

1. Vyberte si oblasť vášho skúmania a definujte si okruh problematiky
2. Manuálna analýza:
    - Čítaním a manuálnou analýzou jednotlivých článkov (použite testovaciu[31] alebo si vytvorte vlastnú množinu článkov), vyberte podmnožinu článkov relevantných ku vašej problematike
3. Analýza pomocou aplikácie:
    - Po spustení aplikácie si definujte vlastné slovníky alebo zvoľte jeden alebo viacero z preddefinovaných slovníkov (postupujte poľa návodu ku aplikácií).
    - Zvoľte požadované články/adresáre pre spustenie extrakcie entít
    - Podľa návodu zvoľte potrebné nastavenia pre beh extrakčného algoritmu
    - Spustite extrakciu, urobte analýzu výstupných súborov a vyberte podmnožinu článkov relevantných ku vašej problematike

B.3.1 Návod ku aplikácii na extrakciu entít

Aplikácia tvorí nadstavbu nad knižnicu JBowl (viac v kap. B.1), ktorá je vyvíjaná na Katedre kybernetiky a umelej inteligencie Technickej univerzity v Košiciach. Je možné ju spúšťať v konzolovom aj grafickom móde.

Obsahom aplikácie sú knižnice v adresári `lib`, ktoré sú potrebné pre jej spustenie, ukážková množina dát pre testovanie uložená v adresári `data/annotataions/articles` a slovníky extrahované z medicínskych ontológií Mesh a UMLS. Slovníky sú rozdelené do troch kategórií, tak by jednotlivé termíny reprezentovali:

- *biomol* – názvy génu, bielkoviny alebo inej biologickej molekuly,
- *body* – názvy hociktorej časti tela, anatomickej alebo histologickej štruktúry,
- *disease* – názvy chorôb či neprirodzených stavov

[31] Množina testovacích článkov je obsiahnutá v balíku s aplikáciou na extrakciu entít.

Jednotlivé slovníky sú uložené v adresári `data/annotataions/dics`.

Ďalej aplikácia obsahuje v adresári `data/annotataions/settings` súbor `default.set` s predvolenými nastaveniami, ktoré je možné meniť editáciou daného súboru, resp. vytvorením nového alebo priamo v okne aplikácie pri grafickom móde aplikácie.

B.3.1.1 Konzolový mód

Spustenie aplikácie v konzolovom móde uskutočnite príkazom, ktorý definuje aj názov súboru z ktorého sa majú načítať nastavenia pre beh aplikácie:

```
java -classpath .;jbowl-ner.jar
examples.task.SimpleTextAnnotation
data/annotations/settings/default.set
```

V súbore s nastaveniami je možné nastaviť nasledujúce položky:

- `dictionary_file` – definuje cestu a názov k jednému slovníku (súbor s názvom *.dic)
- `dictionary_dir` – definuje adresár so slovníkmi (je potrebné, aby v adresári boli len súbory slovníkov)
- `negative_dictionary_file` – definuje cestu a názov „negatívneho“ slovníka, t.j. slovník obsahuje slová, ktoré za žiadnych okolností nemajú byť anotované, napríklad stop slová a pod. (používa sa vtedy, ak sa nastaví vysoká tolerancia pri hľadaní „podobných“ termínov, viac max_words_distance)
- `input_file` – cesta a názov súboru, ktorý má byť anotovaný
- `input_dir` – cesta a názov adresára z ktorého všetky súbory podporovaných súborových typov budú anotované, podporované typy súborov: TXT, PDF, HTM, HTML, XML, RTF, DOC, XSL, PPT a Open Office súbory (ODC, ODM, ODS, ODP, ODT)
- `output_in_txt` – definuje či výstupom má byť *.txt súbor (hodnoty: true ak áno, false ak nie)
- `output_in_html` - definuje či výstupom má byť *.html súbor (hodnoty: true ak áno, false ak nie)
- `ignore_case` – ignorovanie veľkých a malých písmen v slovách (hodnoty: true ak áno, false ak nie)
- `max_words_distance` – udáva maximálny percentuálny rozdiel spracovávaných termínov a slovníkových záznamov (hodnoty: <0 – presná zhoda, 1 – úplna nezhoda[32]>), rozdiel termínov je rátaný na základe Levenshteinovej editačnej vzdialenosti
- `switchable_lexical_type` – definuje lexikálne typy termínov, ktoré budú pri porovnávaní automatický preskakované, hodnoty:

[32] Každý spracovávaný termín bude v zhode s každým slovníkovým záznamom.

- `java.whitespace` – biele znaky (medzery, tabelátory a iné)
- `java.lowercase` – termíny písané iba malými písmenami
- `java.uppercase` – termíny s veľkým počiatočným písmenom
- `java.capitalized` – termíny so všetkými veľkými písmenami
- `digit` – číslice
- `alphanum` – termíny obsahujúce písmena a čísla
- `punctuation` – interpunkčné znamienka (bodka, čiarka, pomlčka, otáznik, atď.)

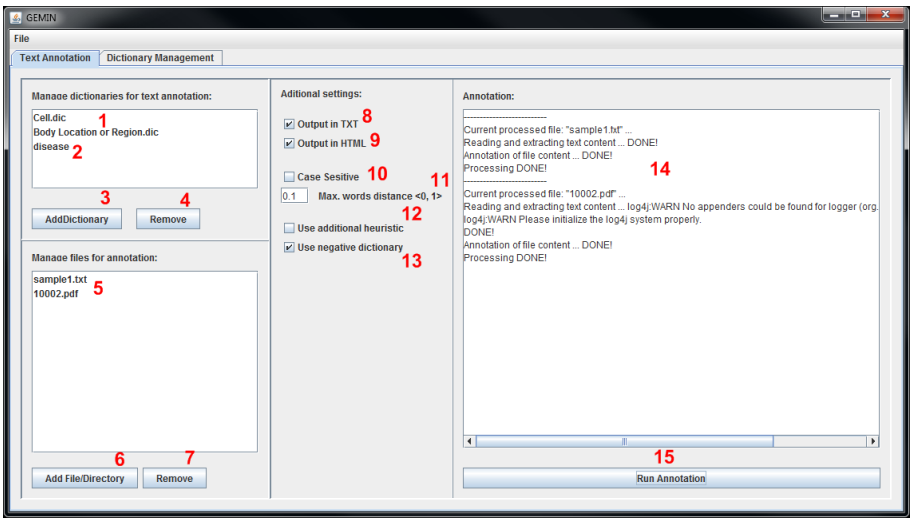
- `non_comparable_lexical_type` – definuje lexikálne typy termínov, pri ktorých nie je nevyhnutné aby boli porovnávané, napr. keď ako `non_comparable_lexical_type` je nastavená hodnota `punctuation`, pre označenia antibiotika „*A-23187*“ je jedno, či je písané ako „*A-23187*“, „*A_23187*“, či „*A/23187*“. Hodnoty sú rovnaké ako pri switchable_lexical_type.

B.3.1.2 Grafický mód

Spustenie aplikácie v grafickom móde uskutočnite príkazom nižšie. V tomto prípade už nie je nutné definovať súbor s nastaveniami, ako to je v prípade konzolového módu, ten je už automatický načítaný (súbor `data/annotations/settings/default.set`). Príkaz pre spustenie:

```
java -jar jbowl-ner.jar
```

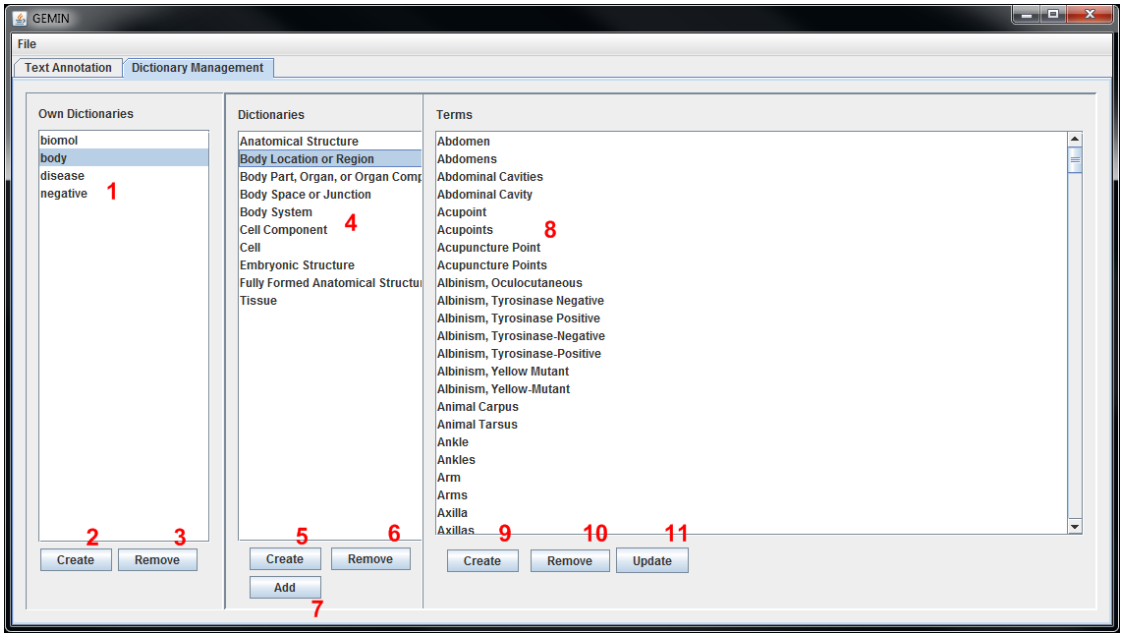
Po spustení sa zobrazí okno aplikácie obsahujúce dve záložky. Prvá z nich `Text Annotation` slúži na nastavenie aplikácie pre spustenie anotácie, druhá záložka `Dictionary Management` slúži výlučne len na prácu so slovníkmi a na ich spravovanie.



Obr. B.3.3 Označenie jednotlivých prvkov v záložke `Text Annotation` v grafickom móde aplikácie.

Záložka Text Annotations (viď. Obr. B.3.3) obsahuje nasledovné informácie a ovládacie prvky:

1. Konkrétne špecifikované slovníky
2. Adresár so slovníkmi
3. Pridanie jedného slovníka/adresára
4. Odobratie vybranej položky
5. Zoznam súbore ktoré budú anotované (možnosť vybrať rôzne typy súborov)
6. Pridanie jedného súboru/adresára
7. Odobratie vybranej položky
8. Nastavenie výstupu vo formáte *.txt
9. Nastavenie výstupu vo formáte *.html
10. Zapnutie/vypnutie rozlišovania veľkých a malých písmen
11. Nastavenie „podobnosti“ slov
12. Použije sa čo najviac vhodné nastavenie pre extrakciu informácií (najvhodnejšie nastavenie `switchable_lexical_type` a `non_comparable_lexical_type`)
13. Použije sa negatívny slovník
14. Zobrazenie výstupu
15. Spustenie extrakcie informácií a anotácie textu



Obr. B.3.4 Označenie jednotlivých prvkov v záložke `Dictionary Management` v grafickom móde aplikácie.

Záložka Dictionary Management (viď.Obr. B.3.4) obsahuje nasledovné informácie a ovládacie prvky:

1. Vlastné a predvolené množiny (adresáre) slovníkov – položky odrážajú reálnu adresárovú štruktúru
2. Vytvorenie množiny (adresára) slovníkov
3. Odstránenie vybranej položky
4. Jednotlivé slovníky v rámci vybranej množiny slovníkov
5. Vytvorenie vlastného slovníka (súboru *.dic)
6. Odstránenie vybranej položky
7. Pridanie z predvolenej množiny slovníkov (z adresára `all` na disku)
8. Termíny vybraného slovníka
9. Vloženie termínu do slovníka
10. Odstránenie vybranej položky
11. Editácia/úprava vybraného termínu

B.3.1.3 Riešenie úlohy

Uvedieme si teraz postup riešenia konkrétneho príkladu. Ako cieľ úlohy sme si zvolili analýzu medicínskych článkov obsiahnutých v balíku aplikácie z pohľadu ich relevantnosti ku popisu ľubovoľnej časti tela, anatomickej alebo histologickej štruktúry. Na základe toho sme použili už preddefinovaný slovník – *body*, ktorý je súčasťou aplikácie.

Analýzu sme vykonali na všetkých testovacích dokumentoch s výstupom do HTML súboru. Po ich otvorení[33] v ľubovoľnom prehliadači webových stránok sme pristúpili ku ich vizuálnej analýze. Sledovali sme počet (hustotu) vyznačených termínov a ich farebné označenie, ktoré je odlíšené podľa sémantických kategórií, do ktorých jednotlivé termíny patria[34].

Výsledkom analýzy bolo zistenie, že jednotlivé články môžeme rozdeliť do štyroch skupín:

- *Články s veľkým výskytom termínov z analyzovanej oblasti*, výbornými príkladmi takýchto článkov sú články s indexom 10002, 10007 a 10042 (pozri Obr. B.3.5).
- *Úzko špecializované články*, t.j. články s veľkým výskytom jedného resp. niekoľko málo rôznych termínov z analyzovanej oblasti, príkladmi sú 10009, 10016, 10024 či 10049 (pozri Obr. B.3.6).
- *Články s malým výskytom termínov z analyzovanej oblasti*, príkladmi sú 10001, 10014 a 10015 (pozri Obr. B.3.7).

[33] Pre správne otvorenie výstupných HTML súborov je potrebné, aby sa v adresári, kde sú dané HTML súbory uložené, nachádzal aj adresár „html_data“, ktorý je štandardne uložený v adresári „data/annotations/articles/“.

[34] Termíny sú kategorizované podľa ontológií Mesh, ktorá obsahuje 133 sémantických kategórií

- *Články písané v inom jazyku*, teda v jazyku, ktorý nie je pokrytý použitým slovníkom (prezentovaná aplikácia používa slovníky len s angl. výrazmi).

10007.pdf

Corneal keratocytes retain neural crest progenitor cell propertiesPeter Y. Lwigale, Paola A. Cressy, Marianne Bronner-Fraser *California Institute of Technology, Pasadena, CA 91125, USAReceived for publication 19 August 2005, revised 27 September 2005, accepted 30 September 2005Available online 2 November 2005AbstractCorneal keratocytes have a remarkable ability to heal the cornea throughout life. Given their developmental origin from the cranial neural crest,we asked whether this regenerative ability was related to the stem cell-like properties of their neural crest precursors. To this end, we challengedcorneal stromal keratocytes by injecting them into a new environment along cranial neural crest migratory pathways. The results show thatinjected stromal keratocytes change their phenotype, proliferate and migrate ventrally adjacent to host neural crest cells. They then contribute tothe corneal endothelial and stromal layers, the musculature of the eye, mandibular process, blood vessels and cardiac cushion tissue of the host.However, they fail to form neurons in cranial ganglia or branchial arch cartilage, illustrating that they are at least partially restricted progenitorsrather than stem cells. The data show that, even at late embryonic stages, corneal keratocytes are not terminally differentiated, but maintainplasticity and multipotentiality, contributing to non-neuronal cranial neural crest derivatives.D 2005 Elsevier Inc. All rights reserved.Keywords: Cornea; Neural crest; Keratocyte; DifferentiationIntroductionThe cornea is a transparent tissue located at the anterior-mostsurface of the eye that transmits and refracts light to the retina.Corneal keratocytes are able to heal wounds and, throughout life,can regenerate the cornea after injury or surgery. However, littleis known about the relationship between the early developmentof the cornea and its subsequent plasticity and ability toregenerate after wounding.In all vertebrate embryos, the cornea is initially comprised ofa layer of ectoderm overlying the lens. In the chick embryo,development of the cornea begins at embryonic day 3 (E3) whenthe optic cup and lens induce the overlying ectoderm tosynthesize an acellular primary stroma consisting of collagenfibrils (Hay and Revel, 1969; Hendrix et al., 1982; Fitch et al.,1988). This is followed at E4 by a wave of invasion of neuralcrest mesenchyme that form an endothelial layer on the innersurface of the corneal primary stroma, adjacent to the lens.Shortly thereafter, a second wave of neural crest cells invadesand contributes to the primary stroma (Hay, 1980; Hay andRevel, 1969).Within the primary stroma, mesenchymal neural crestdifferentiate into keratocytes by E6 and begin to synthesizeand secrete an extracellular matrix composed of collagens I, Vand VI and proteoglycans (Hart, 1976; Hay et al., 1979;Linsenmayer et al., 1983, 1986; Funderburgh et al., 1986). Asmaturation proceeds, the stroma dehydrates becoming thin andtransparent, containing flattened and interconnected keratocyes(Jester et al., 1994). In normal corneas, keratocytes appearquiescent but can resume migration, mitosis, wound healingand repair after injury.A major problem in corneal repair is that improper healingcan result in formation of scar tissue (Rawe et al., 1992; Melles etal., 1995), suggesting that wound repair does not necessarilyreiterate the normal process of development. Therefore, under-standing the developmental potential of differentiated cornealstromal cells is important for understanding the mechanisms ofrepair. However, little is known about the relationship betweencorneal stromal cells and the neural crest precursors from whichthey are derived. One interesting possibility is that theregenerative ability of the cornea is related to the stem cell-like properties of neural crest cells.To examine this relationship, we have utilized quail/chickchimeric grafts to follow the invasion of the cornea byneural crest precursors and

Obr. B.3.5 Článok s veľkým výskytom termínov z analyzovanej oblasti

reduced in Bmpr2 H11546 / H11546 embryos To evaluate how the lack of BMPR2 affects BMP signaling in the embryo, we examined the distribution of phosphorylated Smad1/5 (p-Smad1) in Bmpr2 H11002 / H11002 embryos by immunohisto-f_luorescence staining. In wild-type embryos, p-Smad1 was found in the nuclei of epiblast and primitive endoderm cells at E4.5 and of epiblast and VE cells at E5.2 (Fig. 1 N and Fig. 2, A and B). [ID] The distribution of p-Smad1 changed quickly between E5.2 and E5.5 and had shifted to the proximal epiblast and VE, ex-cluding DVE, at E5.5 (Fig. 2 C and Fig. S3, A ? J, available at http://www.jcb.org/cgi/content/full/jcb.200808044/DC1). In Bmpr2 H11002 / H11002 embryos, the distribution of p-Smad1 was similar to that in wild-type embryos at E4.5 but showed two distinct pat-terns at later stages (Fig. 1 N and Fig. 2 , A ? ? C ?). In severely af-fected mutant embryos (8/14 embryos at E5.2 and 8/15 embryos at E5.5), p-Smad1 was apparent only in the proximal VE at E5.2 (8/8 embryos; Fig. 2 B ?) and was barely detected at E5.5 (8/8 embryos; Fig. 2 C ?). In mildly affected embryos (6/14 embryos at E5.5 and 7/15 embryos at E5.5), p-Smad1 was found in the same regions as in wild-type embryos at E5.2, but its abundance was lower than that in the wild type (6/6 embryos; Fig. 1 N and Fig. S2). It was not detected in the epiblast and there were fewer positive cells in the VE of the mildly affected embryos at E5.5 Results DVE formation is impaired in Bmpr2 H11546 / H11546 embryos Formation of the primitive streak is impaired in Bmpr2 H11002 / H11002 em-bryos (Beppu et al., 2000). To determine whether formation of the A-P axis occurs normally in these mutant embryos, we f_iexamined the expression of AVE or DVE marker genes at E6.5 and E5.5, respectively. In wild-type embryos at E5.5, f_i DVE marker genes, Lefty1 , Cerl , Dkk1 , Lim1 , and Hex , are expressed in VE at the distal tip (Fig. 1, A ? E). [I Expression of Hex , Hesx1 , and Cerl is absent at E5.2 but is apparent at E5.5 (Fig. S1, A ? C and E ? G, available at http://www.jcb.org/cgi/content/full/jcb.200808044/DC1), whereas Lefty1 expression is maintained be-tween E4.0 and E5.5 (Takaoka et al., 2006 ; Fig. S1, D and H), indicating that cells positive for a full range of DVE markers are formed between E5.2 and E5.5. In Bmpr2 H11002 / H11002 embryos, how-ever, expression of AVE marker genes at E6.5 was absent or re-duced compared with that in wild-type embryos (Fig. S2, A ? D, A ? ? D ? , and A ? ? ? D ? ?). Dkk1 expression was lost (Fig. S2 C ?) or remained relatively normal (Fig. S2 C ? ?). At E5.5, expression of Lefty1 , Cerl , Dkk1 , and Lim1 was absent (4/7, 3/7, 3/7, and 3/6 embryos, respectively) or markedly reduced (3/7, 4/7, 4/7, and 3/6 embryos, respectively), and that of Hex was also lost (3/3 embryos; Fig. 1 , A ? ? E ? and N; and Fig. S2, I and I ?). To determine the region of the embryo in which BMP signaling exerts the observed effects, we examined expression of DVE marker genes in green embryonic stem (ES) FM260 cell ? ? Bmpr2 H11002 / H11002 tetraploid chimeric embryos, which were generated by aggregation of ES cells expressing EGFP with Bmpr2 H11002 / H11002 tetraploid embryos. In such chimeras, expression of Hex (n = 3) and Lefty1 (n = 3) was absent at E6.5 (Fig. 1 , F ? H and F ? ? H ?). This phenotype was indistinguishable from that of Bmpr2 H11002 / H11002 embryos, suggesting that BMPR2 in the extra-embryonic region is required for DVE formation. We next examined whether VE is formed normally in Bmpr2 H11002 / H11002 embryos. VE, which is composed of embryonic VE and extraembryonic VE at E5.5, is derived from the primitive endoderm of the E4.0 ? 4.5 embryo. Expression of Hex , which is a marker of the primitive endoderm, was maintained in Bmpr2 H11002 / H11002 Figure 2. p-Smad1 in wild-type and Bmpr2 H11546 / H11546 embryos. Wild-type (A ? C) or Bmpr2 H11002 / H11002 (A ? ? C ?) embryos at the indicated stages of development were subjected to immunohistofluorescence staining with antibodies to p-Smad1 (pS1; green); merged images with staining of nuclei by YOYO-1 (Nuc; red) are also shown. Staining for p-Smad1 was decreased in Bmpr2 H11002 / H11002 embryos. Bars, 50 μ m. on January 27, 2009 jcb.rupress.orgDownloaded from JCBp-Smad1 staining was observed in green ES FM260 cell ? ? Bmpr2 H11002 / H11002 ,Actr2b+/ H11002 tetraploid chimeric embryos (Fig. S3, K and L). These results suggested that both BMPR2 and ActR2b act as receptors for BMP in VE. We next examined DVE markers in Bmpr2 H11002 / H11002 ,Actr2b+/ H11002 embryos at E5.5 to determine whether BMP signaling is required for DVE formation. Expression of DVE markers was detected in some of the Bmpr2 H11002 / H11002

Obr. B.3.6 Článok s veľkým výskytom jedného resp. niekoľko málo rôznych termínov z analyzovanej oblasti. Na uvedenom článku takýmito slovami boli „*embryo*“ a „*endoderm*“

10014.pdf

The American College of Surgeons94th Annual Clinical CongressOctober 12?16, 2008San Francisco, CAABSTRACTSforThe 63rd Annual Sessions of theOwen H Wangensteen Forum on Fundamental Surgical ProblemsPROCEEDINGS FOR THE 63rdANNUAL SESSIONS OF THEOWEN H WANGENSTEEN FORUMON FUNDAMENTAL SURGICALPROBLEMS 2008CLINICAL CONGRESSOCTOBER 12?16, 2008SAN FRANCISCO, CACOMMITTEE FOR THE FORUM ONFUNDAMENTAL SURGICALPROBLEMSVOLUME LXIStanley W Ashley, MD, FACS, ChairBoston, MAOri D Rotstein, MD, FACS, Vice-ChairToronto, ONJohn D Birkmeyer, MD, FACSAnn Arbor, MIYolanda L Colson, MD, FACSBoston, MAE Sander Connolly Jr, MD, FACSNew York, NYEdward Alan Deitch, MD, FACSNewark, NJJames S Economou, MD, PhD, FACSLos Angeles, CACarlos O Esquivel, MD, FACSPalo Alto, CAHenri Ronald Ford, MD, FACSLos Angeles, CADouglas L Fraker, MD, FACSPhiladelphia, PAJames C Hebert, MD, FACSBurlington, VTRichard A Hodin, MD, FACSBoston, MAKaren D Horvath, MD, FACSSeattle, WADixon B Kaufman, MD, FACSChicago, ILNatalie Christine Kerr, MD, FACSMemphis, TNTien C Ko, MD, FACSHouston, TXMichael T Longaker, MD, MBA, FACSStanford, CAD Scott McMeekin, MD, FACSOklahoma City, OKLeigh Anne Neumayer, MD, FACSSalt Lake City, UTC Keith Ozaki, MD, FACSGainesville, FLR Lor Randall, MD, FACSSalt Lake City, UTMarshall Z Schwartz, MD, FACSPhiladelphia, PAFrank William Sellke, MD, FACSBoston, MARobert W Thompson, MD, FACSSt. Louis, MOHunter B Wessells, MD, FACSSeattle, WADavid M Young, MD, FACSSan Francisco, CAThe 261 presentations, dealing withoriginal research conducted by youngsurgical investigators, are groupedinto 24 general topics and arrangedchronologically in this section.Alphabetical Index to SessionsAlimentary Tract IAlimentary Tract IIApoptosis/Cellular InjuryCardiothoracic Surgery ICardiothoracic Surgery IICritical Care ICritical Care IIGenetic DeterminantsGeriatric SurgeryOrthopaedic SurgeryPediatric SurgeryPlastic Surgery IPlastic Surgery IIProgenitor CellsQuality and Outcomes IQuality and Outcomes IIQuality, Outcomes, and Costs IIISurgical Education ISurgical Education IISurgical OncologyTargeted TherapiesTransplantationUrology/Reproductive SurgeryVascular SurgeryThe American College of SurgeonsSURGICAL FORUM 2008Alimentary Tract I: Basic ScienceS11 Heat shock pretreatment protects gastric mucosal cellsagainst oxygen free radical damage. Anna M Leung MD,Maria J Redlak PhD, Thomas A Miller MDS11 Omeprazole induces cell death in gastric glands througha thiol oxidation pathway. Jonathan Emerson Kohler MD,Amy L Blass BS, Jingjing Liu MD, PhD, Jeff Mathew MD,David I Soybel MD, FACSS11 Up-regulation of the insulin signaling pathwaycontributes to amelioration of diabetes during catabolichormonal milieu of gastric bypass.Michael M Meguid MD, PhD, Sandra Bonhomme MS,Ana Guijarro PhD, Susumu Suzuki MD, PhD,Carolina G Goncalves MD, Chung Chen PhDS12 Glucose deprivation promotes chemoresistance togemcitabine through an Akt-dependent mechanism.Michael Abramson MD, Edward W Swanson BAS,Edward E Whang MD, FACSS12 Inhibition of p38 MAP kinase as a target forprophylaxis of postoperative ileus. Sven Wehner PhD,Stefan Straesser, Mariola Lysson, Andreas Hirner MD,Joerg C Kalff MD, FACSS12 Rapid upregulation of the NaH11545/glucose cotransporterSGLT1 in response to an unexpected glucose load.Adam T Stearns MD, Anita Balakrishnan MD,David B Rhoads PhD, Stanley W Ashley MD, FACS,Ali Tavakkolizadeh MDS13 Controlling mechanisms of cell migration in intestinalmucosa cells subjected to repetitive deformation.Christopher P Gayer MD, LS Chaturvedi PhD,Marc Basson MD, PhD, MBAS13 Molecular hydrogen prevents intestinal I/R injury.Takeshi Tsukamoto MD, Bettina M Buchholz

Obr. B.3.7 Článok s malým výskytom termínov z analyzovanej oblasti

Na základe prezentovanej analýzy je už jednoduché vybrať relevantné články ku zvolenej oblasti skúmania a vyhnúť sa tak zbytočnému a prácnemu čisto manuálnemu triedeniu. Toto je možné ďalej ešte „zjemniť“ podrobnejšou analýzou a to napr. redukovaním vybraných sémantických kategórií termínov v preddefinovanom slovníku. Aplikácia taktiež umožňuje „vyskladať“ si vlastný slovník v ktorom používateľ vyberie požadované kategórie termínov, napr. pokiaľ má používateľ záujem o vyhľadanie článkov na tému „*Vírusové ochorenia buniek*“ tak si zvolí kategórie „*Cell*“, „*Cell or Molekular Dysfunction*“, „*Virus*“, resp. ďalšie.

B.4 Použitá literatúra

Bednár P., Butka P., Paralič J. 2005. *Java Library for Support of Text Mining and Retrieval*. In: Proceedings of the 4th annual conference Znalosti 2005. Eds. L. Popelínský, M. Krátký. VŠB TU Ostrava 2005, s. 162-169.

Bednár P., Paralič J. 2003. *KDD package*. In: Proceedings of the 2nd annual conference Znalosti 2003. VŠB TU Ostrava 2003, s. 113-122, ISBN 80-248-0229-5.

Brezany P., Jančiak I., Woehrer A., Tjoa A Min, 2004. GridMiner: A Framework for Knowledge Discovery on the Grid - from a Vision to Design and Implementation Cracow Grid Workshop, Cracow, December 12-15, 2004.

Butka P. 2003. *Clustering of textual documents: modifications of GHSOM algorithm*. In Proceedings of the 4th Slovak-Austrian Student Workshop on Data Analysis, WDA 2003, Malinô Brdo, Slovakia, June 2003, pp.53-65, ISBN 80-7097-523-7.).

Butka P. 2006. *Kombinácia redukcie problému a fuzzy FCA prístupu pri tvorbe konceptuálnych modelov z textových dokumentov*. Znalosti 2006: 5. ročník konference, VŠB-TU Ostrava, Február 2006, pp.71-82, ISBN 80-248-1001-8.

Butka P., Bednár P., Babič F. 2009. *Use of task-based text-mining execution engine in support of knowledge creation processes*. In: Znalosti 2009 : 8. ročník konferencie, Brno, Ceska republika. s. 289-292. ISBN 978-80-227-3015-0.

Furdík K. 2007. *Prototype of a tool for classification of textual documents in eLearning environment*. In: WIKT 2007 Proceedings – 2nd Workshop on Intelligent and Knowledge oriented Technologies (Košice, 15. – 16. november 2007). F. Babič, J. Paralič (eds.), Centrum pre informačné technológie, FEI TU Košice, 2008, s. 41-45. ISBN 978-80-89284-10-8.

Furdík K., Paralič J., Smrž P. 2008. *Classification and automatic concept map creation in eLearning environment*. In: Snášel V. (ed.), Proceedings of the Czech-Slovak scientific conference Znalosti (Knowledge) 2008, Bratislava, február 2008, s. 78 - 89, ISBN 978-80-227-2827-0.

Furdík K., Paralič J., Tutoky G. 2008. *Meta-learning Method for Automatic Selection of Algorithms for Text Classification*. In: Proc. of the Central European Conference on Information and Intelligent Systems (CECIIS 2008), 24. - 26. September 2008, Varaždin, Chorvátsko, s. 477 - 484, ISBN 978-953-6071-04-3.

Furdík K., Bednár P. 2009. *Využitie knižnice JBowl pri spracovaní prirodzeného jazyka*. In: Múcsková G. (ed.), Varia XVI. Zborník materiálov zo XVI. kolokvia mladých jazykovedcov (Častá-Papiernička, 8.-10. 11. 2006). Slovenská jazykovedná spoločnosť pri SAV, Bratislava, 2009, s. 122-131. ISBN 80-89037-04-6.

Ganter B., Wille R. 1997. *Fromal Concept Analysis. Mathematical Foundations*. Springer-Verlag, Berlín, 1997.

Garabík R., Gianitsová L., Horák A., Šimková M. 2004. *Tokenizácia, lematizácia a morfologická anotácia Slovenského národného korpusu*. Slovenský národný korpus, interný materiál. Bratislava, 2004. Dostupné na internete: http://korpus.juls.savba.sk/publications/block2/2004-garabik-gianitsova-horak-simkova-token izacia/2004-garabik-gianitsova-horak-simkova-tokenizacia.pdf

Jasem P., Dolinská S., Paralič J., Dudáš M. 2008. *Automatic data mining and structuring for research on birth defects*. In Proceedings of the 6th IEEE International Symposium on Applied Machine Intelligence (Herľany, Slovakia, Jan 21-22, 2008). SAMI 2008. 137-139. ISBN 978-1-4244-2106-0.

JSR 73. *Data Mining API*. Java Specification Requests. Community Development of Java Technology Specification. Dostupné na internete: http://www.jcp.org/en/jsr/detail?id=73

Kohonen T. 1995. *Self-organizing maps*. Springer-Verlag, Berlín, 1995.

LIBSVM - *A Library for Support Vector Machines*. Chih-Chung Chang and Chih-Jen Lin. Dostupné na internete: http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

Paralič J., Bednár P. 2003. *Text Mining for Documents Annotation and Ontology Support*. In: Intelligent Systems and the Service of Mankind. Eds. Elmenreich W., Machado T., Rudas I. J. Ubooks, Germany, 2003, s. 237-248.

Paralič J., Babič F. 2008. *Support of innovative processes in a computer-based collaborative system*. Basys 2008, Porto, Portugalsko. s.145-152, New York : Springer, 2008, ISBN 978-0-387-09491-5.

Smrž P., Paralič J., Smatana P., Furdík K. 2007. *Text Mining Services for Trialogical Learning*. In: P. Mikulecký, J. Dvorský, M. Krátký (eds.), Proceedings of the Czech-Slovak scientific conference Znalosti (Knowledge) 2007, Ostrava, Česká republika, február 2007, s. 97 - 108, ISBN 978-80-248-1279-3.

StrangeGismo.com: *Web stránka softvérového balíka sg-cdb*. Java implementácia a API ku konštantným databázam D. J. Bernsteina. StrangeGizmo.com. Dostupné na internete: http://www.strangegizmo.com/products/sg-cdb/.

Tkáč E. 2007. *Distribuovaný algoritmus pre extrakciu hierarchie konceptov v prostredí Gridu.* Diplomová práca. Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, 2007.

Tutoky G. 2008. *Meta-učenie pre automatický výber algoritmov na klasifikáciu textov*. Diplomová práca. Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, 2008.

Tymeš M. 2006. *Extrahovanie informácií pomocou metód spracovania prirodzeného jazyka*. Diplomová práca. Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, 2006.

Wai L., Kwok-Yin L. 2001. *A meta-learning approach for text categorization.* Proc. of the 24th ACM SIGIR conference, New Orleans, USA, 2001, pp. 303-309.

Zeher M. 2006. *Využitie hierarchickej dekompozície pre budovanie ontológie z textov*. Diplomová práca. Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach, 2006.

Autori: Ján Paralič
Karol Furdík
Gabriel Tutoky
Peter Bednár
Martin Sarnovský
Peter Butka
František Babič
Názov: Dolovanie znalostí z textov
Vydanie: prvé
Náklad: 80
Rozsah: 182
Vydavateľ: Equilibria, s.r.o., Košice
Formát: B5
Tlač: Equilibria, s.r.o., Košice
Vytlačené: máj 2010

ISBN 978-80-89284-62-7

ISBN 978-80-89284-62-7
EAN 9788089284627