UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO
FAKULTETA ZA MATEMATIKO IN FIZIKO

Ruben Sipoš

MODELIRANJE SOPOJAVITEV BESED Z METODAMI STROJNEGA UČENJA

Diplomska naloga
na interdisciplinarnem
univerzitetnem študiju

Mentor: doc. dr. Janez Demšar
Somentorica: doc. dr. Dunja Mladenić

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00010/2009
Datum: 05.04.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **RUBEN SIPOŠ**

Naslov: **MODELIRANJE SOPOJAVITEV BESED Z METODAMI STROJNEGA UČENJA**

MODELLING WORDS CO-OCCURRENCE WITH MACHINE LEARNING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Metode strojnega učenja ponujajo vrsto možnosti za modeliranje podatkov. Pred kratkim je Google objavil seznam pogostih sopojavitev besed, zgrajen na osnovi indeksa njihovega iskalnika. Z uporabo strojnega učenja lahko poskusimo zgraditi model, ki bi na enostaven način opisal del informacij, ki so zajete v sopojavitvah besed.

V nalogi preučite, kako bi odkrili in opisali povezave med besedami, ki jih vsebuje ta zbirka podatkov. Preučite tudi možnost izgradnje modelov, temelječih na sopojavitvah besed, z uporabo metod strojnega učenja in razpoložljivega predznanja. Njihovo kvaliteto poskusite oceniti s primerjavo z morebitnimi alternativnimi pristopi. Pri modeliranju posvetite posebno pozornost ravnovesju med splošnostjo in specifičnostjo modela.

Mentor:

doc. dr. Janez Demšar

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Franc Solina

Somentor:

doc. dr. Dunja Mladenić

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden izda izdelek v vezavo!

Zahvala

Zahvaljujem se mentorju doc. dr. Janezu Demšarju in somentorici doc. dr. Dunji Mladenić za pomoč in usmerjanje pri izdelavi diplomske naloge.

Prav tako bi se zahvalil sodelavcem Odseka za tehnologije znanja Inštituta »Jožef Stefan« (predvsem Marku Grobelniku, Janezu Branku in Simonu Kreku) za pomoč pri delu.

Zahvala gre tudi moji družini, ki me je med študijem podpirala in mi stala ob strani.

Kazalo

Povzetek

Z razvojem metod strojnega učenja in vse večjo zmogljivostjo računalnikov, se nam odpirajo nove možnosti obdelave podatkov in avtomatiziranega pridobivanja informacij. Pri avtomatski analizi besedila se zadnje čase vse pogosteje pojavlja vprašanje, kako zajeti vsaj del semantične informacije. Eden izmed možnih pristopov je, da poskusimo modelirati semantiko, ki jo nosijo sopojavitve besed v besedilu.

V okviru tega diplomskega dela smo razvili pristop, s katerim lahko na podlagi n-terk besed zgradimo modele predstavljene kot trojke osebek-povedek-predmet. Najprej smo predstavili podatke in njihovo predpripravo. Uporabili smo Googleove n-terke zgrajene na podlagi njihovega indeksa spletnih strani. Posebno pozornost smo namenili tudi temu, kako uspešno obdelamo takšno količino podatkov, saj gre za eno izmed večjih podatkovnih zbirk tega tipa. Nato argumentiramo, zakaj je sopojavitve besed dobro predstaviti kot trojke in kako le-te dovolj hitro izračunamo, saj so trenutno razširjeni pristopi veliko prepočasni.

Predlagali in razložili smo postopek za gradnjo modelov sopojavitev besed. Z njegovo pomočjo lahko izračunamo modele, ki s splošnejšimi koncepti opisujejo večje število konkretnih trojk. Podali smo tudi evaluacijo rezultatov, ki nakazuje na potencialno uporabnost naših rezultatov. Na koncu nakažemo še nekaj zanimivih možnosti za nadaljnje raziskave.

Ključne besede:

strojno učenje, n-terke besed, trojke, izračun trojk, modeliranje sopojavitev besed, posploševanje konceptov (z uporabo predznanja)

Abstract

Advances in machine learning and increasing computing power are providing new possibilities for data processing and knowledge acquisition. One of the key questions in automatic text analysis is how to acquire semantic information. A possible approach is to model semantics using word co-occurrence.

In the context of this work we have developed an approach which enables us to build models, represented as triples consisting of subject, predicate and object, based on word n-grams. We used Google n-grams constructed on the basis of their index of web pages. Special attention was also given to how to efficiently process this quantity of data, because it is one of the largest datasets of this type. Also, we provide justification for choosing representation using triples and describe how to efficiently compute triples because current approaches are time consuming

We propose a new procedure for construction of models of word co-occurrences. Each model represents a set of triples using more general concepts. We also give the results of evaluation, which indicate the potential usefulness of our results. We conclude with some interesting ideas for further research.

Keywords:

machine learning, word n-grams, triples, triple extraction, modeling word co-occurrences, concept abstraction (using background knowledge)

1 Uvod

Živimo v času, ko smo priča bliskovitemu razvoju na področju računalništva, tako na strojnem področju, kjer še vedno dokaj lepo sledimo Moorovemu zakonu, kot tudi na programskem področju. Obdelava visokoločljivostnega videa, uporaba spletnih storitev, kot je recimo strojno prevajanje in hramba velike večine podatkov v podatkovnih bazah namesto na papirju, je samo nekaj primerov uporabe računalnika, ki je postala samoumevna.

Temelj vseh ugodnosti, ki nam jih računalniki ponujajo, je njihova računska moč. Procesorji v današnjih računalnikih lahko opravijo po več milijard operacij na sekundo. Človeška sposobnost izvajanja enostavnih matematičnih in logičnih operacij je povsem neprimerljivo majhna. Čeprav so računalniki še vedno zelo daleč od tega, kakor jih vidimo v naših sanjah, če za primer pomislimo na vizije o umetni inteligenci izpred nekaj desetletij, kljub temu že sedaj obiramo precej plodov. Če nas zanima odgovor na vprašanja kot recimo »Kolikokrat se pojavi črka *a* v vseh slovenskih besedilih?« in »Ali je na kateri izmed milijard spletnih strani napisano kaj o gobarstvu?«, nam lahko računalnik priskrbi odgovor v trenutku. Enostavna opravila, kot na primer preverjanje ali poštne številke ustrezajo vpisanim imenom pošt v podatkovni bazi, opravimo veliko hitreje in tudi ceneje s pomočjo računalnika.

Ob vsej računski moči, s katero razpolagamo, pričakujemo, da bi lahko s pomočjo računalnikov naredili še kaj več kot zgolj reševali trivialne probleme. Zelo uspešen pristop je strojno učenje. Na podlagi primerno velike količine podatkov (opažanj ob opazovanju nekega pojava) lahko avtomatsko zgradimo model. Uporabimo ga lahko kot dodatno pomoč za boljše razumevanje pojava. Na podlagi modela je tudi možno napovedati izide prihodnjih eksperimentov.

Kadar želimo uporabiti računalnik za reševanje kompleksnejših problemov, se marsikdaj pokaže, da je potrebno vključiti tudi dele splošnega predznanja. Pri avtomatski analizi besedila nam kot predznanje lahko pomagajo sopomenke, nadpomenke, poznavanje entitet itd. Kadar ljudje poskušamo priti do ugotovitev na podlagi danih podatkov, velikokrat uporabimo tudi znanje, ki smo ga pridobili že prej tekom našega življenja in zato spada med predznaje. Računalniki tega dodatnega znanja nimajo oz. ni dostopno v primerni obliki. Možna rešitev je, da znanje pridobimo od eksperta in ga primerno oblikovanega uporabimo, kadar je to potrebno. Vse večja dostopnost velikih količin podatkov (predvsem z razvojem interneta in digitalizacijo podatkov) nam odpira nove možnosti avtomatskega zbiranja specifičnega predznanja.

Pred kratkim je Google objavil seznam pogostejših n-terk besed iz njihovega spletnega indeksa. Ker spletni iskalniki dandanes s svojimi indeksi pokrivajo večinski del spleta, lahko predpostavimo, da vsebina objavljenih n-terk približno ustreza vsebini »interneta«. Ob predpostavki, da ljudje na spletnih straneh objavljajo tudi informativne članke iz raznovrstnih področij, nam n-terke predstavljajo zanimiv vir informacij.

Cilj te naloge je bil preučiti povezave med besedami znotraj n-terk in nato razviti pristop za izgradnjo modela, ki bi opisal del informacij zajetih v sopojavitvah besed. Objavljene n-terke so v resnici izseki realnega objavljenega besedila. Da bi dosegli nekoliko boljšo abstrakcijo, obravnavamo raje sopojavitve besed. Na podlagi teh lahko

zgradimo model jezika, ki preko sopojavitev besed opisuje del vsebine besedil. Model smo želeli zgraditi s pomočjo metod strojnega učenja ob uporabi razpoložljivega predznanja. Pozornost smo želeli posvetiti tudi iskanju primernega nivoja abstrakcije, da bi s tem povečali nadaljnjo uporabnost našega modela (na primer kot dodatne uteži pri odločanju ob analizi besedil).

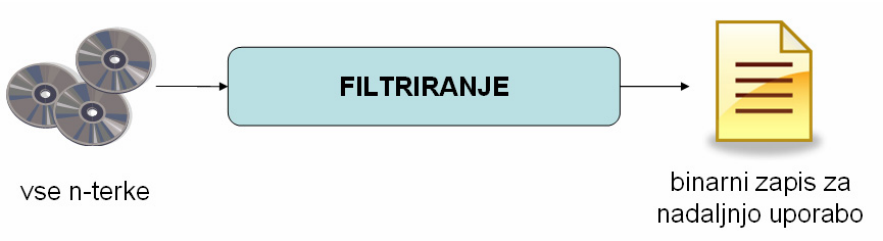
Celotna rešitev, ki jo predstavljamo v tej nalogi, je zgrajena dokaj modularno. Grob oris večjih sestavnih komponent lahko vidimo na sliki 1. Vsaka komponenta bo v sledečih poglavjih podrobno opisana.

Slika 1: Trije glavni deli predstavljenega pristopa.

Diplomska naloga je sestavljena iz petih poglavij. V uvodu smo si ogledali, kakšno vlogo imajo računalniki v našem življenju in kakšne prednosti nam prinaša njihova uporaba. V naslednjem poglavju bomo predstavili javno dostopen Google-ov seznam pogostih sopojavitev besed, ki predstavlja naše vhodne podatke. Nato si bomo ogledali dva pristopa za izračun trojk na podlagi n-terk in ju primerjali. Za tem sledi opis naše metode za modeliranje sopojavitev in njena evaluacija. Na koncu sledi opis rezultatov naloge in predlogi za morebitno nadaljnje delo.

2 Podatki

V tem poglavju predstavljamo prvo komponento naše rešitve iz slike 1. Grafični prikaz transformacij izvedenih nad vhodnimi podatki je podrobneje prikazan na sliki 2.



Slika 2: Priprava podatkov.

2.1 Opis podatkov

Oglejmo si najprej nekaj osnovnih dejstev o objavljenih n-terkah. Zbirka podatkov vsebuje n-terke angleških besed in število njihovih ponovitev. Objavljene n-terke so dolge od ene do pet besed. Objavljene so bile predvsem za namene statističnega modeliranja jezika kot tudi za morebitne druge aplikacije. N-terke in število njihovih pojavitev je bilo izračunano na podlagi enega bilijona besed besedila na spletnih straneh. Avtorji so poskusili zajeti zgolj angleško besedilo, vendar lahko zaradi nepreciznosti uporabljenih (oz. razpoložljivih) metod najdemo tudi kakšno n-terko zgrajeno na podlagi besedila v kakšnem drugem jeziku. Podatki so bili zajeti v januarju 2006 in zato besedilo, ki je nastalo ali bilo spremenjeno kasneje, ni zajeto.

Že pred izračunom n-terk je bilo opravljenega nekoliko filtriranja besed. Zato so bile odstranjene vse besede z očitno neželenimi lastnostmi. Tako so bile na primer odstranjene vse predolge besede, besede z znaki, ki niso v latinici, besede s preveč črkami z naglasnimi znamenji, nesmiselne kombinacije ločil ipd. Vse besede, ki jih je označil filter in vse besede pod določeno mejo pogostosti pojavitev, so predstavljene z besedo <UNK>.

Meje med povedmi so bile ugotovljene avtomatsko. Začetek povedi označuje posebna beseda <S>, konec pa </S>. Obe posebni besedi sta bili šteti tako kot vse ostale in jih zato najdemo tudi v n-terkah. Tako je na primer število pojavitev besede <S> enako številu povedi, v katere je bilo razdeljeno besedilo.

Zaradi ogromne količine izvornega besedila bi bilo nepraktično ohraniti povsem vse n-terke, ki se pojavijo v besedilu. Zato so v zbirki podatkov le tiste besede, ki se pojavijo vsaj 200 krat (kar pomeni vsaj enkrat vsakih 5 milijard besed). Vse ostale besede predstavlja posebna beseda <UNK>. N-terka se mora ponoviti vsaj 40 krat (t.j. vsaj enkrat na 25 milijard), da je vključena med objavljene.

Podatki so bili objavljeni[1] na 6 DVD-jih (približno 26Gb podatkov). N-terke so v urejenem zaporedju shranjene v več tekstovnih datotekah stisnjenih z gzip. Vsaka vrstica v datoteki predstavlja eno n-terko; besede so ločene s presledki, na koncu vrstice pa je napisano še število vseh pojavitev.

Dolžina n-terk	Število različnih n-terk
1	13,588,391
2	314,843,401
3	977,069,902
4	1,313,818,354
5	1,176,470,663
Število povedi	95,119,665,584
Število besed	1,024,908,267,229

Tabela 1: Število različnih n-terk glede na njihovo dolžino.

Iz tabele 1 je razvidno, da gre za res ogromno količino podatkov. Na podlagi tega upamo, da ta zbirka podatkov vsebuje dovolj redundance in koristnih informacij, da bomo lahko na njeni podlagi zgradili dovolj kvaliteten model. Za primerjavo si poglejmo Reutersovo zbirko novic [1], ki se jo precej uporablja, kadar potrebujemo večjo količino besedila za testiranje razvitih metod za analizo besedila. Prav tako je velikokrat uporabljena kot primer vira podatkov za metode, ki nam omogočajo lažji dostop do informacij zajetih v besedilu [2]. Zbirka je velika nekaj več kot 2Gb in obsega 806.791 člankov; vsak je v povprečju dolg 206,8 besed. Torej je skupna velikost približno 167 milijonov besed, kar je skoraj 10-krat manj, kot je različnih peterk v Googleovi objavljeni zbirki. Ob tej primerjavi moramo imeti v mislih, da peterke verjetno nosijo precej več informacij kot besede.

Reutersova zbirka novic pokriva pretežno le finančno področje, medtem ko Googleove n-terke pokrivajo skoraj celoten splet. Zato lahko najdemo zanimive primere iz mnogih popolnoma različnih področij v tabeli 2.

n-terka	področje
cow eats grass	živali, splošno znanje
humans currently reside on earth	ljudje, splošno znanje
iraq declared war	zgodovina, dogajanje po svetu

Tabela 2: Primeri n-terk in tematskih področij, ki se jih dotikajo.

2.2 Priprava podatkov

Spekter ciljnih uporabnikov objavljenih n-terk je zelo širok, zato je bila na njih pred objavo izvedena zgolj najnujnejša obdelava. Za naše potrebe so nekatere lastnosti preveč splošno zastavljene, zato je najprej potreba priprava podatkov. S tem dodatnim korakom zagotovimo nekaj dodatnih lepih lastnosti in poenostavimo nadaljnjo obdelavo.

[1] http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13

Največji del podatkov predstavljajo peterke, ki so shranjene v 118 tekstovnih datotekah stisnjenih z gzip. Skupaj zavzemajo 8,86Gb, v nestisnjeni obliki pa bi potrebovali vsaj 3-krat več prostora. Ob podrobnejšem pregledu opazimo, da je med njimi veliko neuporabnih (npr. »2000) , 12 SCLR«) in precej pogojno uporabnih. Če izločimo skoraj popolnoma neuporabne, katerih besede so pretežno sestavljene iz ločil in ostalih znakov in zato niso angleške besede, ostanejo po grobem filtriranju le datoteke z zaporednimi številkami od 21 do 97 (skupaj 77).

Med preostalimi peterkami je še vedno precej zgolj pogojno uporabnih. Primer takšne peterke je »as I 'm alive </S>«, ki vsebuje mejo med dvema povedma. Ker vsebuje mejo med dvema različnima povedma, to v primeru uporabe našega modela (ki ga bomo opisali v poglavju 3.1) pomeni, da moramo takšno peterko obravnavati kot eno oz. dve krajši n-terki. Ker smo zaenkrat izvedli poskuse zgolj na peterkah (saj so le-te najprimernejše za naše potrebe), vse primere, kjer po obdelavi dobimo eno ali več krajših n-terk, zavržemo (oz. bi jih dodali na seznam krajših).

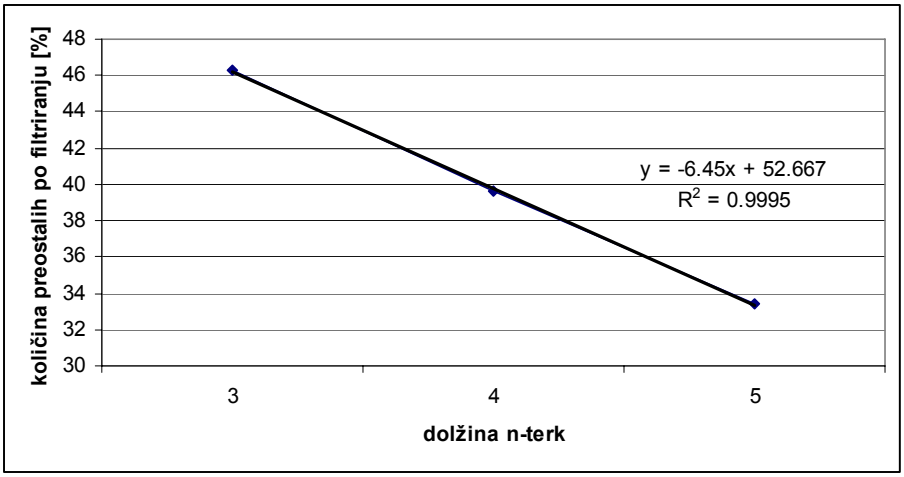
Najenostavnejši in najhitrejši pristop k filtriranju je, da zavržemo (oz. dodamo h kandidatom za krajše n-terke) vse, ki ne vsebujejo zgolj alfa-numeričnih znakov. Res je, da s tem pristopom mogoče odstranimo kakšen primer, ki bi se sicer izkazal za dobrega kasneje, vendar upamo, da je med podatki dovolj redundance, ki nekoliko ublaži posledice agresivnega filtriranja. Zaradi samega obsega podatkov tudi ob zelo strogem filtriranju še vedno razpolagamo z ogromno količino podatkov. Hkrati pa si poenostavimo nadaljnje delo, saj imamo sedaj zelo natančno določeno, kaj so veljavni podatki. V primeru, da želimo kljub temu obdržati določene primere, ki bi jih sicer s filtriranjem odstranili, lahko ročno napišemo pravilo, ki poskrbi, da jih obdržimo za nadaljnjo obravnavo.

Odločili smo se, da podatke shranimo v binarni obliki (namesto kot več stisnjenih tekstovnih datotek), saj je na ta način lažje dostopati do njih. Vsaka peterka je predstavljena s petimi celimi števili, kjer vsako število predstavlja določeno besedo glede na izbrano preslikavo. Posledično je možno tudi izračunati odmik v datoteki za vsako peterko, saj vse zasedejo enako prostora. Ker so originalne datoteke že abecedno urejene, je tudi naša binarna datoteka urejena in nam omogoča binarno iskanje. Dodatna prednost binarnega zapisa je manjša poraba prostora. Tako po filtriranju potrebujemo le 8,79Gb prostora za vse peterke. Tabela s preslikavo med besedami in celimi števili je dolga manj kot 50Mb. Glede na to, da imajo današnji zmogljivejši osebni računalniki že po 8Gb glavnega pomnilnika, je to obvladljiva velikost, saj lahko vse podatke naložimo v pomnilnik.

Število preostalih n-terk po opravljenem filtriranju je predstavljeno v tabeli 3. Največ n-terk ostane pri filtriranju n-terk dolgih štiri besede, medtem ko je najmanj n-terk dolgih pet besed. Zanimiva lastnost, ki smo jo mimogrede opazili, je dokaj dobra linearna odvisnost med dolžino n-terk in deležem preostalih n-terk po opravljenem filtriranju, kar je prikazano na sliki 3. Verjetnost, da gre za naključje je dokaj velika, saj imamo le tri različne dolžine n-terk, vendar bi vseeno bilo zanimivo v prihodnosti podrobneje pogledati to in morebitne ostale lastnosti v namene boljšega razumevanja osnovnih zakonitosti n-terk.

Dolžina n-terk	Število n-terk	Število različnih besed	Velikost [Gb]	Preostale po filtriranju [%]
3	452,562,469	1,881,345	6,74	46,3
4	519,945,560	1,469,700	9,68	39,6
5	393,507,635	1,231,036	8,79	33,4

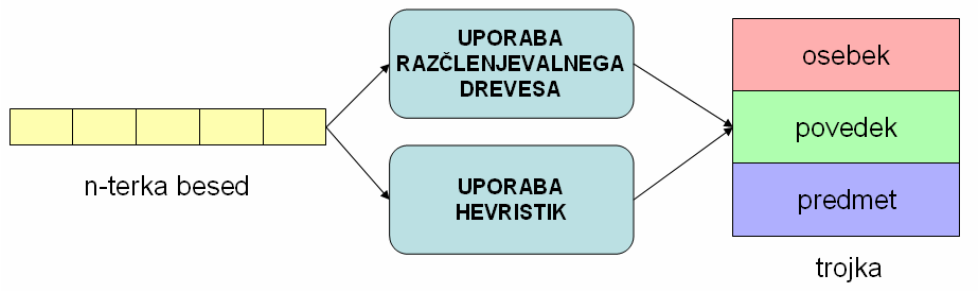
Tabela 3: Število preostalih n-terk po opravljenem filtriranju.



Slika 3: Linearna odvisnost med dolžino n-terk in številom preostalih n-terk po filtriranju.

3 Iskanje povezav med besedami

To poglavje obravnava tematiko predstavitve sopojavitev besed z uporabo trojk. Predstavljeni sta tudi dve metodi za izračun trojk na podlagi n-terk besed in njuna primerjava. Slika 4 prikazuje oris te komponente.



Slika 4: Gradnja trojk na podlagi n-terk.

3.1 Predstavitev s trojkami

Pri analizi besedil in kadar uporabljamo strojno učenje na podatkih, ki so predstavljeni kot besedilo, velikokrat za predstavitev besedila uporabimo predstavitev z vrečo besed. Namesto da bi uporabili celotno besedilo (predstavljeno kot niz znakov oz. bajtov), uporabimo zgolj izbrane lastnosti, ki jih izračunamo na podlagi danega besedila. Podatke predstavimo tako, da posamezne primere predstavljajo dokumenti, njihovi atributi pa so lastnosti besedila, ki smo jih izbrali. Pri uporabi predstavitve z vrečo besed so lastnosti dokumentov števila ponovitev posameznih besed v njih. Izračun števila ponovitev besed v dokumentih je zelo enostaven in hiter; dobljena predstavitev pa je neposredno uporabna kot učna množica v večini metod strojnega učenja.

Kljub temu, da predstavitev z vrečo besed ne ohrani strukture besedila (končna predstavitev z vrečo besed je povsem enaka, tudi če naključno premešamo vse pozicije besed v besedilu), jo precej pogosto uporabljamo, saj z njo dosežemo dobre rezultate pri nekaterih nalogah (na primer klasifikacija besedila v vnaprej določene kategorije [3]). V nekaterih primerih lahko rezultate še nekoliko izboljšamo tako, da z atributi zajamemo tudi nekaj strukture besedila. Najenostavnejši pristop k temu je, da v vrečo besed vključimo tudi število pojavitev n-terk v besedilu (t.j. strnjenih besednih fraz).

Omejitev uporabnosti n-terk v tem pristopu je njihova specifika. Ker je osnovna enota n-terka in nas ne zanima, katere besede jo sestavljajo oz. kakšen je njihov pomen, to pomeni, da vsaka najmanjša razlika v zapisu pomeni, da gre za dve popolnoma različni n-terki. Čeprav je pomen lahko identičen, bomo n-terki, ki se razlikujeta na primer zgolj v sklonu ene besede ali vsebujeta različni sopomenki, obravnavali kot popolnoma različni in neodvisni.

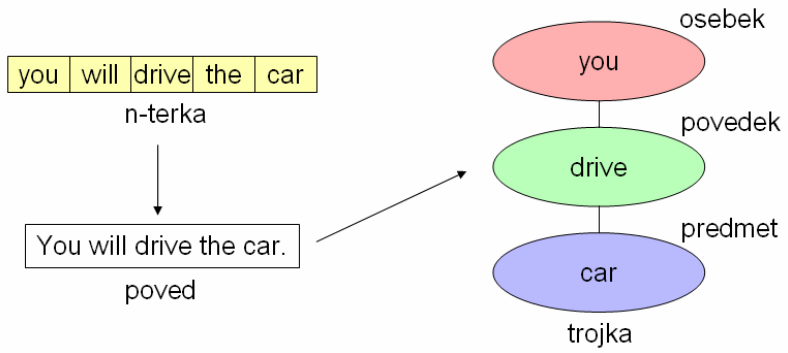
Precej uporabljan način upoštevanja zaporedja besed v besedilu (in s tem vsebine in pomena) je uporaba informacij o sopojavitvah besed. Najosnovnejši pristop je

upoštevanje tega, kolikokrat se določeni besedi v besedilu pojavita zaporedoma tik ena ob drugi. Intuicija za tem pristopom je, da se določene besede pogosteje pojavljajo v kombinaciji z nekaterimi (npr. »morska deklica«) in da nam lahko različne kombinacije besede pomagajo natančneje opredeliti vsebino besedila (npr. »morska deklica« v primerjavi z »morska solata«). Ta pristop je identičen uporabi n-terk dolžine dve. Možna izboljšava oz. razširitev pristopa je, da ob iskanju sopojavitev besed dovolimo, da le-te niso nujno samo sosedne, ampak je med njimi lahko tudi nekaj drugih besed, ki jih ignoriramo. V določenih primerih nam to lahko izboljša rezultate, saj nas velikokrat zanimajo sopojavitve besed, ki nosijo večji del pomena (npr. samostalniki), vendar zaradi drugih besed (npr. vezniki) ne stojijo vedno tesno skupaj. Obe varianti uporabe sopojavitev se pogosto uporabljata, predvsem v namene razreševanja dvoumnosti besed (npr. [4]) in klasifikacije dokumentov v kategorije glede na vsebino (npr. [5]).

Zbirka n-terk, ki jo imamo na voljo, vsebuje tudi dolge n-terke. Pri manjših obsegih besedila (v primerjavi s količino dostopno na spletu) redkokdaj uporabljamo n-terke daljše od treh besed. Eden izmed razlogov je gotovo zamudnost in zahtevnost izračuna, saj zgolj z naivnimi pristopi kaj hitro ne obvladamo več kompleksnosti (oz. presežemo mejo truda, ki smo ga še pripravljeni vložiti glede na pričakovane izboljšave rezultatov). Druga slabost uporabe daljših n-terk na manjših korpusih besedila je razmeroma majhno število ponovitev enakih n-terk. Zaradi pestrosti jezika se redko zgodi, da isto idejo izrazimo s popolnoma enakim zaporedjem besed.

Oglejmo si najprej peterke, ki jih imamo na razpolago. Vsako sestavlja pet zaporednih besed, ki so del neke povedi (n-terke vsebujoče mejo med dvema povedma smo že predhodno odstranili). V primeru, da so bili stavki znotraj povedi ločeni z ločili, je peterka nastala na podlagi takšnega kosa besedila poravnana znotraj enega stavka. Pet besed pa je dovolj, da z njimi zapišemo celotno oz. vsaj velik del krajše in enostavnejše povedi oz. stavka. Zato lahko obravnavamo peterke kot krajše povedi, ki so mogoče nekoliko obrezane. Enostavno poved si lahko predstavljamo kot zaporedje osebka, povedka in predmeta. S tem modelom ne pokrijemo vseh možnosti, saj gre tukaj za precejšnjo poenostavitev, vendar se izkaže, da podatki sprejemljivo dobro ustrezajo temu modelu. V najkrajšem možnem primeru (glede na naš model) so potrebne vsaj tri besede (npr. »Deklica nese košaro.«), zato so za naše potrebe uporabne vse n-terke dolžine vsaj treh besed. Ob uporabi peterk imamo tako na razpolago še dve dodatni besedi, zato da lahko zajamemo tudi povedi, ki vsebujejo samostalniške besedne zveze.

Odločili smo se, da bomo n-terke obravnavali kot povedi, ki vsebujejo kot glavne tri dele osebek, povedek in predmet. Nadalje lahko iz povedi izločimo posebej osebek, povedek in predmet, ter označimo trojico kot relacijo med temi tremi sestavnimi deli povedi [6]. Alternativen pogled je, da nam povedek predstavlja tip relacije med osebkom in predmetom. V obeh primerih smo prišli iz n-terke preko razlage s pomočjo povedi do neke relacije med besedami. Za razliko od najosnovnejšega pristopa k sopojavitvam, imamo sedaj veliko bolj konkretno definirano strukturo in njene lastnosti.



Slika 5: Primer izračunane trojke na podlagi peterke.

Preostale besede v n-terki, ki niso glavni del osebka, povedka ali predmeta, označimo kot dodatne atribute le-teh. S tem jih ohranimo za primer, da jih bomo kasneje še potrebovali. Če pogledamo zgolj trojke brez dodatnih atributov in jih primerjamo z n-terkami, lahko opazimo, da je prostor možnih zapisov sedaj precej manjši. Zaradi pestrosti jezika so lahko n-terke med seboj različne kljub enakemu pomenu, medtem ko so trojke zgrajene iz njih enake, kot to vidimo v tabeli 4. Razlike med trojkami s skoraj enakim pomenom ostajajo le v primeru uporabe sopomenk ali zaradi različnih sklonov.

vhodne n-terke	**izračunana trojka**
you drive his car you will drive the car you drive this car you drive licensed cars you could drive any car	you drive car
passive smoking increased the risk smoking significantly increased the risk smoking might increase the risk smoking may increase the risk smoking can increase your risk	smoking increase risk
apple releases a new ipod apple released the original ipod apple released the new ipod apple will release an ipod apple could release an ipod	apple release ipod

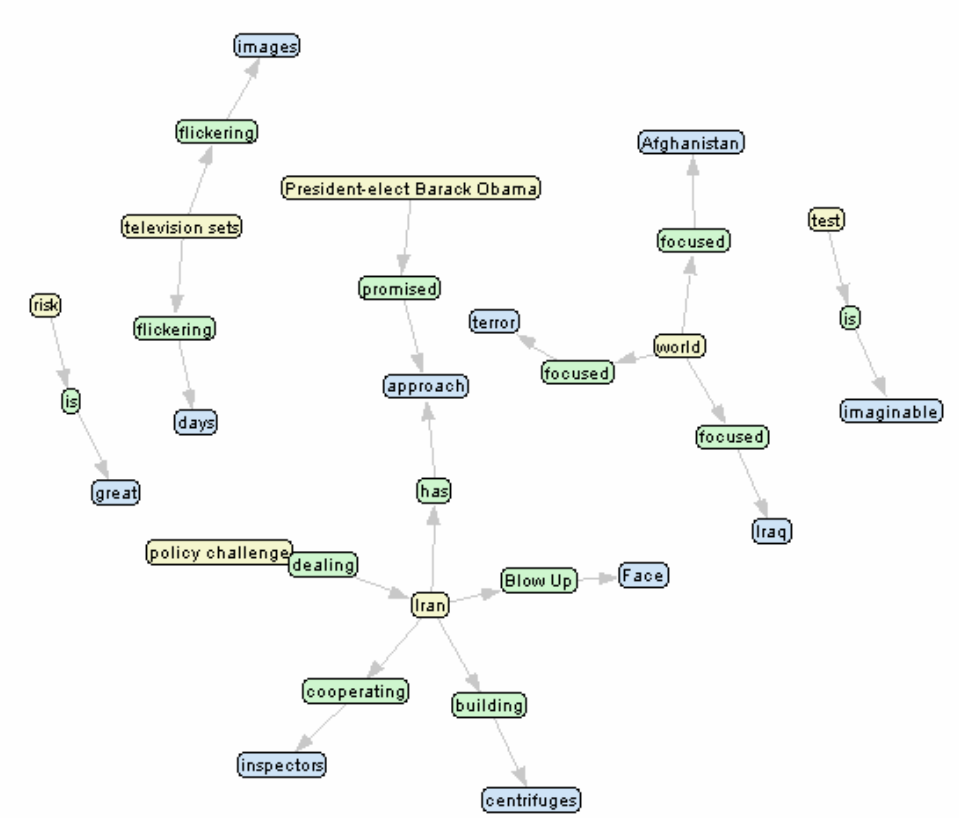
Tabela 4: Primeri n-terk iz katerih ustvarimo enake trojke.

Glavna prednost predstavitve s trojkami je, da ne operiramo več z zaporedjem besed kot pri n-terkah, ampak z relacijami med koncepti, ki so resda še vedno predstavljeni z besedami, vendar zato sedaj vemo, katere besede so centralne in kako so povezane med seboj. Nova predstavitev ni več eksplicitno sekvenčna, ampak je po svojih lastnostih bližje usmerjenemu grafu, čeprav majhnemu z zgolj tremi vozlišči.

Eden izmed pomembnejših argumentov, ki je vplival na našo odločitev za predstavitev s trojkami je tudi dejstvo, da jih uporablja precej sodobnih pristopov opisanih v novih člankih na temo iskanja znanja iz besedila. Zelo nazoren primer, kjer se je uporaba trojk izkazala za zelo uspešen pristop, je povzemanje besedila [7]. Če vsako

poved v besedilu predstavimo kot trojko in iz njih zgradimo graf ter uporabimo le pomembnejši del grafa, dobimo lepo predstavljen povzetek dokumenta kot, recimo, na sliki 6. Še en dober primer prednosti predstavitve s trojkami so sistemi za odgovarjanje na vprašanja, ki znajo avtomatsko poiskati možne odgovore na postavljeno vprašanje [2].

Kaj takšnega ni mogoče narediti, če bi uporabljali predstavitev besedila z vrečo besed. Metode, ki bi zanesljivo analizirale celotno besedilo, še ne obstajajo, zato so trojke vmesen korak k večjemu upoštevanju semantike besedila in ne zgolj statistike besed.



Slika 6: Primer povzetka dokumenta predstavljenega s trojkami.

Razvoj poteka tudi na področju hranjenja in obdelave velikih količin trojk. Obstajajo namenska skladišča za shranjevanje trojk in hitro iskanje po njih [8]. Za vzpodbudo raziskav v tej smeri je bilo organiziranih nekaj tekmovanj. Eno izmed njih je bilo »Billion Triple Challenge[2]«. Tekmovalci so imeli na voljo 1 milijardo trojk (kar je več kot jih imamo mi), ki so jih morali čim bolj koristno uporabiti. Trojke so bile zbrane iz prosto dostopnih baz na spletu (npr. DBpedia[3]). V primerjavi z našimi trojkami iz Googleovih n-terk so veliko bolj čiste (manj šumnih primerov), vendar zato tudi pokrivajo precej bolj ozko področje in vsebujejo ogromno specifičnih primerov (konkretne instance konceptov). Verjamemo, da je za naše cilje bolj primerna izbrana osnova, saj predvidevamo, da zajema več splošnih in za nas zanimivih informacij kot trenutno obstoječe ročno ustvarjene baze trojk.

[2] http://challenge.semanticweb.org/
[3] http://dbpedia.org/About

Ob tem naj omenimo še format RDF[4] (Resource Description Framework). Ta format je primeren za shranjevanje trojk v obliki osebek-povedek-predmet. Uporabljajo ga skoraj vse aplikacije za delo s trojkami. Za izvajanje operacij nad podatki v RDF formatu obstaja SPARQL[5], ki je namenski jezik za poizvedovanje in omogoča vse pogoste operacije nad takšnimi podatkovnimi zbirkami. Naš model predstavitve podatkov je kompatibilen z idejo za RDF, zato bi lahko v primeru potrebe naše podatke enostavno izvozili v obliki, ki bi jo lahko nato uporabili v že obstoječih aplikacijah.

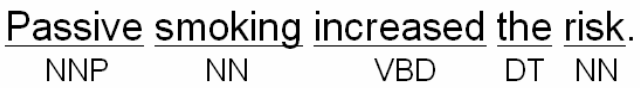
3.2 Pristop z globinskim skladenjskim razčlenjevanjem

Odločili smo se, da bomo na podlagi n-terk zgradili trojke, katerih osnovna struktura je osebek-povedek-predmet. Da bi lahko naredili to, moramo najprej opraviti stavčno analizo in označiti kam spadajo posamezne besede. Na spletu so dostopna brezplačna orodja za namene procesiranja naravnega jezika (NLP – natural language processing). Ena izmed funkcionalnosti, ki jih ponujajo, je tudi stavčna analiza.

Javno dostopna skladenjska razčlenjevalnika sta Stanford Parser in OpenNLP. Oba lahko kot izhod generirata drevesnico. Stanford Parser je razčlenjevalnik za naravne jezike, ki so ga razvili v The Stanford Natural Language Processing Group[6]. Del programskega paketa je javanska implementacija verjetnostnega skladenjskega razčlenjevalnika. OpenNLP[7] je skupina projektov za procesiranje naravnega jezika. V tem delu smo uporabili SharpNLP[8], ki je implementiran v C#. Oba razčlenjevalnika sta naučena na korpusu Treebank[9], ki vsebuje ročno in v celoti označeno besedilo.

Glede na rezultate objavljene v [9] je SharpNLP hitrejši, zato smo se odločili, da uporabimo le-tega, saj je hitrost pomembna pri tolikšni količini podatkov. Obdelava podatkov se vrši poved za povedjo (oz. n-terko). Knjižnica po korakih najprej doda oblikoskladenjske oznake k besedam, nato razbije poved v več besednih zvez, nazadnje pa zgradi drevesnico.

Oblikoskladenjske oznake, dodane k besedam, označujejo besedno vrsto. Ta korak ni trivialen zato, ker imajo lahko besede različno vlogo glede na to, kje v povedi stojijo. Na sliki 7 je viden primer oblikoskladenjskih oznak (razlaga nekaterih izmed njih je vidna v tabeli 5; razlago ostalih lahko najdemo na spletu[10] in v [10]), s katerimi je bila označena poved »Passive smoking increased the risk«.



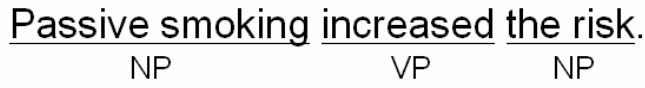
Slika 7: Poved z dodanimi oblikoskladenjskimi oznakami.

[4] http://www.w3.org/RDF/
[5] http://www.w3.org/TR/rdf-sparql-query/
[6] http://nlp.stanford.edu/
[7] http://opennlp.sourceforge.net/
[8] http://www.codeplex.com/sharpnlp
[9] http://www.cis.upenn.edu/~treebank/
[10] http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

oblikoskladenjska kategorija	razlaga
NN	samostalnik, občni, ednina ali snovno ime oz. množinski samostalnik
NNP	samostalnik, lastno ime, ednina
VB	glagol, osnovna oblika
VBD	glagol, preteklik
DT	določilnik

Tabela 5: Primeri oblikoskladenjskih oznak.

Naslednji korak pri analizi povedi je razbitje le-te v več kosov. Vsak kos je sestavljen iz besed, ki skupaj predstavljajo en stavčni člen. Imena vrst besednih zvez, ki so pomembni pri naši nadaljnji obdelavi n-terk in njihove oznake, so povzeti v tabeli 6 (razlaga ostalih oznak je dostopna na spletu[11] in v [10]). Če izvedemo ta korak analize na primeru povedi iz slike 7, dobimo rezultat na sliki 8.



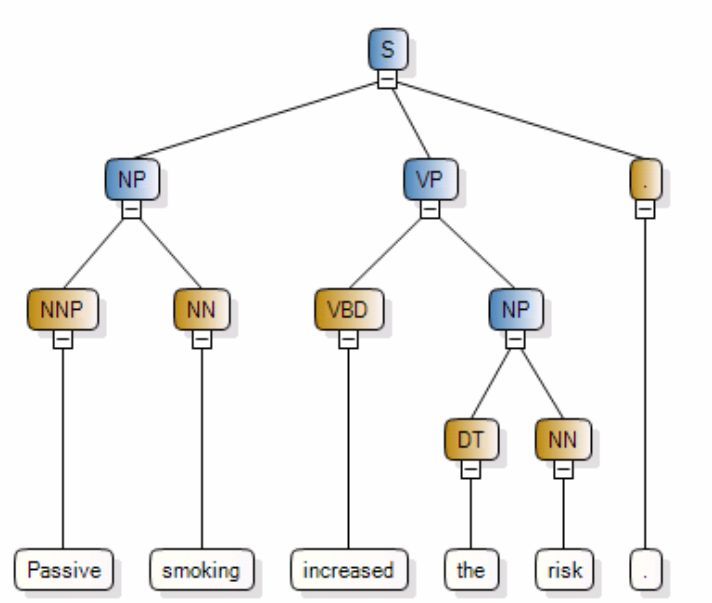
Slika 8: Poved razbita na besedne zveze.

oznaka	kategorija besedne zveze
NP	samostalniška
VP	glagolska
PP	predložna
ADVP	prislovna
ADJP	pridevniška

Tabela 6: Primeri oznak besednih zvez.

Končni rezultat analize po končanem tretjem koraku je celotna drevesnica, katerega primer je na sliki 9. S tem je končana stavčna analiza povedi. Naša naslednja naloga je, da z uporabo rezultatov stavčne analize poiščemo osebek, povedek in predmet v povedi in tako ustvarimo trojko. Algoritem, ki smo ga uporabili v naši implementaciji, je bil objavljen v [9], kjer je podana tudi psevdokoda. Poleg trojke osebek-povedek-predmet, kjer je vsak element predstavljen z eno besedo, algoritem na podlagi drevesnice poišče tudi prilastke. Ker je vsak izmed izbranih stavčnih členov v trojki predstavljen zgolj z jedrom besedne zveze, v primeru besedne zveze ostale besede označimo za prilastke in jih shranimo kot dodatno informacijo v trojki. Primer trojke nastale na podlagi drevesnice na sliki 9 je prikazan na sliki 10. Dopisani so tudi prilastki, ki jih najde algoritem in doda kot dodatno informacijo.

[11] http://bulba.sdsu.edu/jeanette/thesis/PennTags.html



Slika 9: Primer drevesnice.

Slika 10: Trojka s pripadajočimi prilastki zgrajena na osnovi drevesnice.

V tem poglavju smo predstavili gradnjo trojk na podlagi n-terk z uporabo t.i. globinskega skladenjskega razčlenjevanja in tako dobljenih drevesnic. Naslednje poglavje zajema opis alternativnega pristopa, ki smo ga razvili, da bi odpravili nekatere slabosti pristopa z uporabo razčlenjevalnika. Primerjava obeh metod je podana v poglavju 3.4.

3.3 Pristop s hevristikami

Metoda, opisana v prejšnjem poglavju, je uporabna za gradnjo trojk na podlagi n-terk, vendar ima nekaj slabosti, ki smo jih želeli odpraviti. Največji problem je njena počasnost. Že pri uporabi za analizo znatno manjših količin podatkov kot je naša, je dolgotrajnost postopka pereč problem. Zato je bil naš cilj doseči dovolj veliko pospešitev, da bo možno obdelati vse naše n-terke v razumnem časovnem okviru. Pri tem smo se sprijaznili s tem, da se bo kvaliteta rezultatov morebiti poslabšala, vendar je to sprejemljivo, saj bi bilo v siceršnjem primeru nemogoče obdelati vse podatke.

Drugi večji problem prej opisane metode je neveljavnost predpostavke, da analiziramo cele povedi. Učni podatki za Stanford Parser in OpenNLP so označene

povedi. V našem primeru operiramo z n-terkami, ki so v večini primerov fragmenti povedi. Posledica tega je, da so dobljeni rezultati slabši od pričakovanih.

Naša ideja, ki je osnova za alternativni pristop, je enostavna. Ker je analiza do te mere, da dobimo drevesnico, časovno zahtevna, poskusimo zgraditi trojko zgolj z uporabo informacij o besednozveznih enotah iz prejšnjih korakov analize. Tako se izognemo gradnji drevesnice, ki je najbolj dolgotrajen del stavčne analize. Razviti moramo nov algoritem, ki bo lahko, kljub nepopolni stavčni analizi, našel osebek, povedek in predmet ter njihove prilastke.

Ogledali smo si primerno količino n-terk, katerih besede so bile označene s oblikoskladenjskimi oznakami in razdeljene v besedne zveze z uporabo OpenNLP knjižnice, da bi odkrili morebitne vzorce, s pomočjo katerih bi lahko zgradili trojke. Izkaže se, da lahko že v zelo majhnem vzorcu (nekaj deset primerov) odkrijemo nekaj pogostih vzorcev, ki pokrijejo večino možnosti. Zato smo se odločili, da bomo poskusili rešiti problem iskanja trojk z uporabo hevristik.

primeri n-terk z dodanimi oblikoskladenjskimi oznakami
apple released the new ipod NN VBN DT JJ NN NP VP NP
i have image FW VBP NN NP VP NP
crime have victim NN VBP NN NP VP NP

Tabela 7: Tabela podaja nekaj primerov n-terk z dodanimi oblikoskladenjskimi oznakami.

V tabeli 7 je predstavljenih nekaj primerov n-terk z dodanimi oblikoskladenjskimi oznakami in označenimi besednimi zvezami. Najenostavnejši primer (prva vrstica v tabeli 7) je, ko našo n-terko oz. poved sestavljajo zgolj tri besednozvezne enote: NP, VP, NP. Določitev trojke je sedaj trivialna: prvi NP del predstavlja osebek, VP del predstavlja povedek in zadnji NP del predstavlja predmet. Da bi dobili trojko, je potrebno poiskati, kaj je jedro vsakega izmed treh delov in ostale besede označiti za prilastke. Bolj zapletene hevristike najdemo v tabeli 8. Dobili smo jih tako, da smo pregledali nekaj primerov označenih n-terk in poiskali pogoste vzorce. Nato smo odstranili vse primere, ki so jih pokrile že napisane hevristike in ponovno poskusili najti vzorce v preostalih primerih. Že po nekaj iteracijah uspemo najti dovolj vzorcev, da z njimi pokrijemo veliko večino zanimivih primerov.

Vzorec predstavljen z zaporedjem kategorij besednih zvez
NP, VP, NP
NP, VP, PP, NP
NP, ADVP, VP, NP
NP, ADVP, VP, PP, NP
NP, ADJP, VP, NP
NP, ADJP, VP, PP, NP
NP, PP, VP, NP
NP, PP, VP, PP, NP
NP, VP, ADVP, NP
NP, VP, PRT, NP

Tabela 8: Vzorci uporabljeni pri gradnji trojk našteti po prioriteti.

Naša metoda uporablja deset hevristik, ki podajajo zaporedje vrst besednih zvez za gradnjo trojk iz n-terk. Pri obdelavi n-terke preverimo ali zaporedje vrst besednih zvez, ki je rezultat analize, vsebuje katero izmed zaporedij podanih v tabeli 8 kot strnjeno podzaporedje. V primeru neuspeha nismo uspeli najti primerne razlage vsebine n-terke ali pa je le-ta nesmiselna in ni dovolj podobna povedi, zato trojke ni mogoče zgraditi in nadaljujemo z naslednjo n-terko. V nasprotnem primeru smo našli možno razlago in zato zgradimo trojko, za katero uporabimo le tiste besedne zveze, ki jih zajema hevristika in so bili najdeni kot podzaporedje. Trojko ustvarimo, tako da najprej razbijemo n-terko na osebek, povedek in predmet (ter morebitne preostale besede na začetku ali koncu n-terke, ki niso vključene v noben del) s hevristiko, ki temelji na izbiri VP dela povedi (kot vidimo v tabeli 8 je to res v vseh uporabljenih vzorcih). Besede v VP delu predstavljajo povedek, vse besede pred tem so del osebka, vse za VP delom pa spadajo v predmet.

del trojke	oblikoskladenjske kategorije
osebek, predmet	NN, NNS, NNP, NNPS
povedek	VB, VBD, VBG, VBN, VBP, VBZ

Tabela 9: Izbrane oblikoskladenjske oznake, na podlagi katerih izberemo jedro.

Poiskati moramo še jedro v vsakem izmed treh delov. Ker nimamo drevesnice, uporabimo za pomoč oblikoskladenjske oznake besed. Odločili smo se, da kot jedrno besedo označimo prvo besedo v osebku, povedku ali predmetu, ki ima primerno oblikoskladenjsko oznako. Tabela 9 podaja našo izbiro, za katero smo se odločili ob upoštevanju pomena oblikoskladenjskih oznak in pregledu lastnosti konkretnih primerov. Psevdokoda celotnega algoritma je podana v algoritmu 1.

```
//v zanki obdelamo vhodne podatke
foreach ngram in INPUT
    //opravimo stavčno analizo
    posTags = OpenNLP.posTag(ngram)
    parts = OpenNLP.markChunks(ngram, posTags)
    //poiščemo ustrezen vzorec
    matchingPattern = NULL
    foreach template in allTemplates
        if ContainsAsSubstring(parts, template)
            matchingPattern = template
            break
    if matchingPattern == NULL
        continue
    //ustvarimo trojko
    triple[3] = new()
    offset = FindStartOffset(parts, matchingPattern)
    currentPosition = 0
    while parts[offset+currentPosition] != "VP"
        triple[0] += ngram[offset+currentPosition++]
    while parts[offset+currentPosition] == "VP"
        triple[1] += ngram[offset+currentPosition++]
    while currentPosition < template.length
        triple[2] += ngram[offset+currentPosition++]
    //poiščemo glavne besede
    for i=0..2
        foreach word in triple[i]
            if posTags[word] in mainWordsTagsSet
                triple[i].mainWord = word
                break
    //dodamo med že obdelane
    OUTPUT += triple
```

Algoritem 1: Izračun trojk na podlagi n-terk.

Opisana metoda je izboljšava v primerjavi z globinskim razčlenjevanjem, saj je veliko hitrejša in se ne zanaša v tolikšni meri na predpostavko, da gre za cele povedi. Pristop je obetaven, saj se je izkazalo, da tudi nekateri drugi raziskovalci razvijajo precej sorodne pristope našemu za gradnjo trojk iz povedi brez uporabe drevesnic. V bližnji prihodnosti verjetno lahko pričakujemo objave člankov s to tematiko.

Paralelno procesiranje

Kljub veliki pohitritvi v primerjavi z uporabo drevesnic, je naš algoritem, ki uporablja hevristike za gradnjo trojk iz n-terk, še vedno časovno zahteven. Da bi pohitrili obdelavo podatkov in izkoristili vse zmogljivosti novih večprocesorskih računalnikov, smo želeli naš program za gradnjo trojk iz n-terk napisati s podporo za večnitnost. Našo obdelavo podatkov je enostavno paralelizirati – obdelujemo lahko poljubno število n-terk hkrati, saj so med seboj neodvisne. Ker je naš program za gradnjo trojk napisan v C#, kar je pogojeno z uporabo knjižnice SharpNLP, je delo z nitmi preprosto.

Predelava programa v večnitnega zahteva nekaj dodatne pazljivosti s sinhronizacijo niti, velika večina kode pa lahko ob tem ostane ista. Vendar se je izkazalo, da ta rešitev ne deluje tako, kot bi si želeli. Nekaj časa za tem, ko poženemo program, se

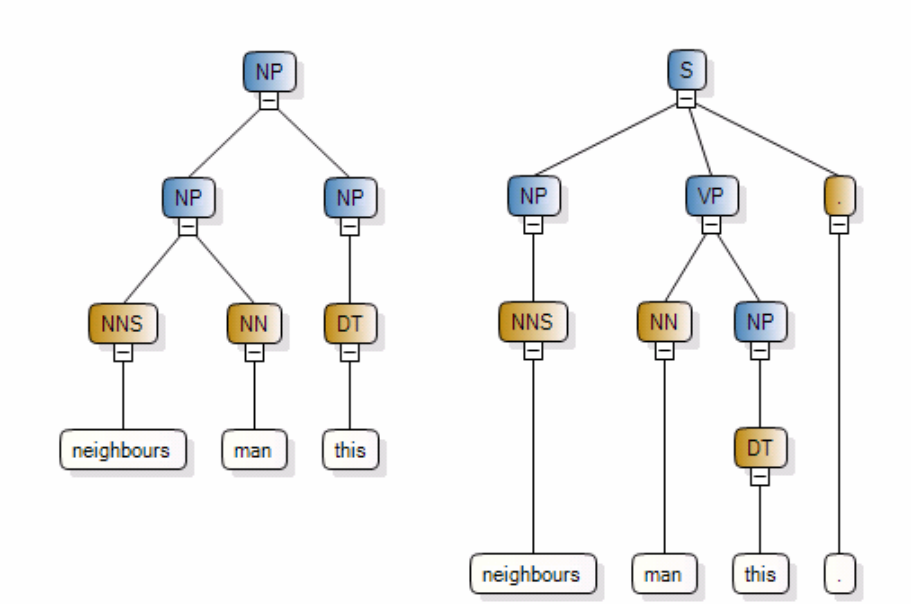
obdelava n-terk upočasni. Ob pregledu raznih statistik izvajanja kode smo opazili, da vedno, razen čisto na začetku, program več kot 90% procesorskega časa porabi za samodejno sproščanje pomnilnika. Uporaba strežniškega načina dela s pomnilnikom (vsak procesor ima svojo kopico; ko se sproži sproščanje, se le-to izvaja vzporedno na vseh procesorjih) situacijo nekoliko izboljša, vendar je hitrost procesiranja na vsaki niti še vedno očitno nižja kot v primeru enonitnega programa. Sklepamo, da je del krivde na ShrapNLP knjižnici, ki pri obdelavi ustvarja ogromno novih objektov in tudi sicer ni napisana za uporabo v večnitnih aplikacijah. Na koncu smo bili prisiljeni doseči paralelnost s sočasnim izvajanjem več instanc programa za obdelavo.

3.4 Primerjava metod

Predstavili smo dva pristopa za gradnjo trojk iz n-terk. Metoda, ki uporablja drevesnice, je znana in se uporablja v raznih projektih, medtem ko je metoda temelječa na hevristikah nova, in prinaša nekaj izboljšav. Oglejmo si primerjavo njunih prednosti in slabosti.

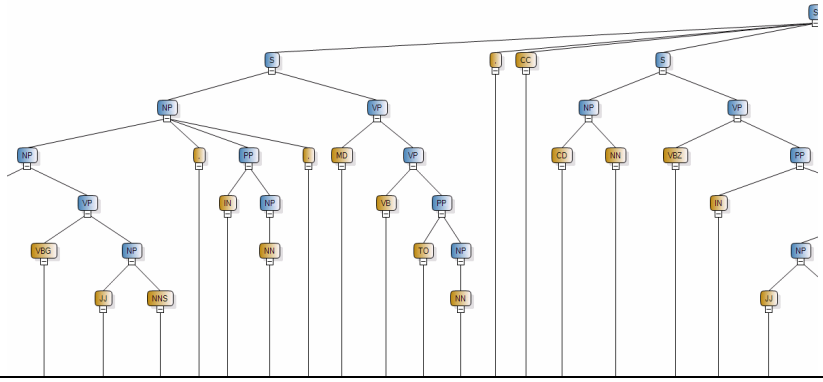
Poglavitna prednost novega pristopa je hitrost. Za obdelavo 393,507,635 peterk smo potrebovali okrog 1,2 milijona sekund, kar je skoraj 14 dni procesorskega časa na 2,2GHz procesorju Opteron 875. Rezultat je bil 63,814,809 ustvarjenih trojk, kar pomeni 16,2% najdenih vzorcev s hevristikami. Potreben čas za obdelavo ene n-terke je 3ms. Če to primerjamo z globinskim razčlenjevanjem, kjer je potreben čas za obdelavo ene povedi velikostnega reda 1s (glede na rezultate v [9] je potrebnih 0,3s). Naš pristop s hevristikami torej približno stokrat hitrejši. Da bi obdelali vse naše podatke z uporabo globinskega razčlenjevanja, bi potrebovali eno leto na štirijedrnem procesorju. Nekoliko (vendar najverjetneje ne dovolj) bi lahko pohitrili gradnjo drevesnice, če bi program napisali na primer v C-ju namesto v C#, vendar bi morali potem sami napisati tudi razčlenjevalnik.

Druga prednost našega pristopa je manjša občutljivost na to, da n-terke niso cele povedi, ampak v večini primerov le fragmenti. Knjižnice za gradnjo drevesnic so bile naučene na označenih primerih, ki so bili pravilne povedi. Zato lahko velikokrat opazimo napake v drevesnici, če postopek izvedemo na okrnjeni povedi. Primer napačne drevesnice dobljene iz še vedno smiselnega fragmenta povedi je podan na sliki 11. Ta problem bi lahko poskusili rešiti tako, da bi razčlenjevalnik naučili na umetno ustvarjenih fragmentih povedi, ki bi morali dovolj dobro modelirati fragmente v naši zbirki podatkov, iz označenega korpusa Treebank. Ta možnost še ostaja neraziskana in je ena izmed tem za nadaljnje raziskave.



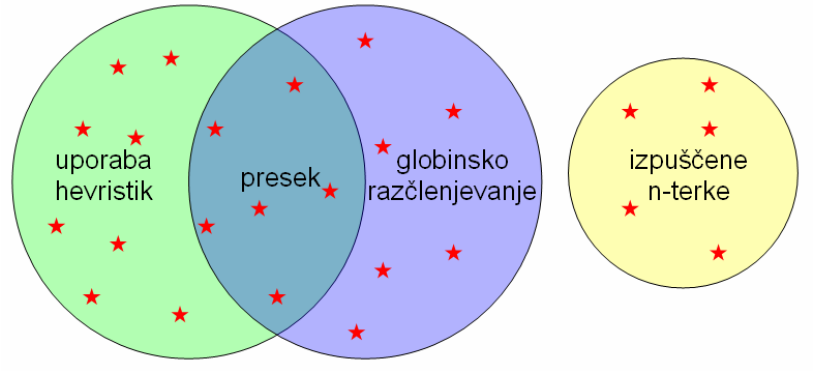
Slika 11: Primer neuporabne drevesnice dobljene iz smiselnega fragmenta besedila. Opazimo lahko, da že majhne spremembe (npr. prisotnost pike) precej vplivajo na obliko drevesnice.

Slabost uporabe hevristik so slabši rezultati v primeru kompleksnejših stavčnih struktur. Medtem ko globinsko razčlenjevanje upošteva celotno poved naenkrat, naši vzorci gledajo zgolj lokalne strukture. Kompleksnejši primeri (kot na primer poved na sliki 12) lahko vsebujejo dele, ki so lokalno podobni kateremu izmed vzorcev, vendar je potrebno upoštevati celotno sliko, da dobimo pravilno trojko. Ker so hevristike sestavljene na podlagi konkretnih primerov, se lahko zgodi, da smo kateri vzorec spregledali zaradi redkosti pojavljanja. Rezultate lahko nekoliko izboljšamo s tem, da dodamo nove vzorce, kadar opazimo, da je to potrebno.

A sentence describing successive numbers, for example, could stretch to infinity, and one concatenating clauses with grammatical conjunctions such as and could go on as long as material may be supplied.
[NP A/DT sentence/NN] [VP describing/VBG] [NP successive/JJ numbers/NNS] ,/, [PP for/IN] [NP example/NN] ,/, [VP could/MD stretch/VB] [PP to/TO] [NP infinity/NN] ,/, and/CC [NP one/CD concatenating/NN clauses/NNS] [PP with/IN] [NP grammatical/JJ conjunctions/NNS] [PP such/JJ as/IN] and/CC [VP could/MD go/VB] [PP on/IN] [NP as/RB long/JJ as/IN material/NN] [VP may/MD be/VB supplied/VBN] ./.


Slika 12: Primer kompleksnejše povedi in izračunanih oblikoskladenjskih oznak ter dela drevesnice.

Zanimala nas je tudi primerjava kvalitete rezultatov obeh metod, zato smo želeli priti do kvantitativne ocene razlik med pristopoma. Najenostavneje bi lahko preverili naše rezultate, če bi imeli na razpolago pravilne oznake kot na primer podatke o tem, katere besede n-terke spadajo v kateri del trojke, ter katere besede so zgolj prilastki, za vsak del vhodnih podatkov. Vendar pravilne oznake za naše podatke (še) ne obstajajo, saj so naši podatki relativno novi in zaenkrat malo uporabljani; prav tako pa priprava takšnih referenčnih podatkov zahteva veliko količino ročnega dela. Če bi želeli uporabiti za evaluacijo druge podatke, za katere obstajajo oznake, bi le-ti verjetno bili predstavljeni kot cele povedi. Zato bi bilo najprej potrebno iz njih ustvariti fragmente in tako simulirati n-terke, ob tem pa upoštevati še morebitne dodatne lastnosti n-terk, ki izvirajo iz svetovnega spleta. Iz teh razlogov smo se odločili, da bomo rezultate ocenili ročno. Izbrali smo naključen vzorec (podmnožica vhodnih n-terk), iz njega z uporabo različnih metod izračunali trojke in ročno ocenili njihovo kvaliteto.



Slika 13: Množici vrnjenih trojk pri primerjavi metod.

Ob uporabi dveh različnih metod za izračun trojk, bi v idealnem primeru pričakovali enake rezultate. Slika 13 ponazarja, kaj lahko pričakujemo v realnem primeru, saj nobena metoda ni popolnoma točna. Če si ogledamo trojke, dobljene ob uporabi obeh metod, opazimo štiri množice:

- v preseku so trojke, ki sta jih vrnili obe metodi
- nekatere trojke vrne zgolj globinsko razčlenjevanje
- nekatere trojke vrne samo metoda s hevristikami
- nekaterih trojk ne vrne nobena metoda, čeprav bi morale biti med rezultati

Najprej smo želeli ugotoviti, kolikokrat se zgodi, da noben izmed predstavljenih pristopov ne uspe ustvariti trojke na podlagi dane n-terke. V idealnem primeru se naj bi to zgodilo samo takrat, kadar je vsebina n-terke neuporabna in ni niti približno podobna kratki povedi. Ker takšna n-terka krši našo glavno predpostavko o lastnostih n-terk in ne vsebuje relacije oblike osebek-povedek-predmet, iz nje ne moremo ustvariti trojke. Nobena metoda ne daje zagotovljeno pravilnih rezultatov v vseh primerih, zato pričakujemo, da morebiti obstajajo n-terke, iz katerih ne uspemo ustvariti trojke s pomočjo računalnika, kljub temu, da bi to ročno lahko storili. Analizirali smo naključen vzorec 100 n-terk in ročno pregledali primere, v katerih noben pristop ne vrne trojke. Rezultati so predstavljeni v tabeli 10.

	število n-terk
velikost vzorca	100
ustvarjene trojke z uporabo globinskega razčlenjevanja	6
ustvarjene trojke z uporabo hevristik	17
obe metodi vrneta trojko	4
popolnoma neuspešni primeri	81
neuporabne n-terke med neuspešnimi primeri	79
uporabne n-terke med neuspešnimi primeri	2

Tabela 10: Prikazani so rezultati ročne evaluacije, s katero smo želeli pregledati primere, v katerih obe metodi odpovesta.

Opazimo, da v približno 80% primerov ne uspemo ustvariti trojke ne z enim in ne z drugim pristopom. To razmerje je razvidno tudi pri večjem vzorcu prikazanem v tabeli 11. Pregledali smo vseh 81 problematičnih primerov in odkrili, da v veliki večini (79 od 81 primerov) n-terke niso dovolj podobne povedim, da bi lahko pričakovali uspešen izračun trojke. Tako se je na primer le za dve n-terki izkazalo, da bi morali dobiti trojki, vendar sta obe metodi zatajili. Najbolj pogosta razloga za neuporabnost n-terk sta bila, da so bile sestavljene zgolj iz samostalnikov (v njih ne moremo najti povedka) ali pa so se pričele z glagolom (in zato v njih ni mogoče najti osebka). Nekaj preostalih razlogov je: vsebina n-terke je zgolj šum, n-terka je fragment vprašalne povedi, ponekod manjka glavna beseda predmeta, ipd.

V drugem delu primerjave metod smo se osredotočili na uspešno ustvarjene trojke in ocenili njihovo kvaliteto. Ročno smo pregledali trojke ustvarjene na podlagi naključnega vzorca 1000 peterk. Rezultati so predstavljeni v tabeli 11.

metoda, ki je uspešno ustvarila trojko	kvaliteta ustvarjenih trojk	število trojk oz. n-terk	delež
uporaba drevesnic	pravilne	33	52 %
	uporabne	4	6 %
	prisiljene	18	29 %
	napačne	8	13 %
	skupaj	**63**	
uporaba hevristik	pravilne	89	62 %
	uporabne	33	23 %
	napačne	22	15 %
	skupaj	**144**	
obe metodi hkrati	popolnoma enaki	14	40 %
	enaki	20	57 %
	različni	1	3 %
	skupaj	**35**	
	neuspešnih	828	

Tabela 11: Rezultati ročne evaluacije kvalitete trojk. Naključni vzorec je vseboval 1000 peterk.

Najprej smo ocenili rezultate vsake metode posebej. Ročno smo pogledali n-terke in trojke, ki so bile zgrajene iz njih. Kvaliteto trojke smo definirali kot:

- **pravilna**: Trojka popolnoma ustreza vsebini n-terke. Če bi morali ročno poiskati osebek, povedek in predmet iz dane n-terke in določiti morebitne prilastke, bi dobili točno takšen rezultat kot ga vrne metoda. Primer: »discussion document | is | a sample«
- **uporabna**: Vsebina dobljene trojke ne ustreza vsebini n-terke, vendar še vedno predstavlja smiselno relacijo in ostaja razmeroma podobna izvorni n-terki. Primer: »little advice | to help | you«
- **prisiljena**: Opazili smo, da pristop z uporabo globinskega razčlenjevanja velikokrat v n-terki, ki je sestavljena iz samih samostalnikov, nasilno najde glagol. Očitno je, da so bile besede v izvornem besedilu samostalniki, vendar v primeru obstoja možnosti uporabe kot glagol, le-to metoda izkoristi in ustvari trojko, čeprav je posledično popolnoma nesmiselna. Primer: »Pictures Random | Thoughts | Rants Recreation«
- **napačna**: Ustvarjena trojka je nesmiselna in neuporabna. V veliki večini teh primerov je tudi izvorna n-terka nesmiselna in neuporabna, vendar je še vedno delno podobna povedi. Zato tudi v teh primerih dobimo trojko, saj metodi poskusita izvesti transformacijo ne glede na vsebino (oz. odsotnost le-te). Primer: »order | to understand | these files«

Nato smo pregledali še primere, v katerih obe metodi vrneta trojko in jih primerjali med seboj. Pare trojk smo uvrstili v tri kategorije:

- **popolnoma enaki**: Vrnjeni trojki sta identični.
- **enaki**: Vrnjeni trojki se razlikujeta le v prilastkih, vse tri jedrne besede pa so enake v obeh primerih.
- **različni**: Trojki se razlikujeta v eni ali več jedrnih besedah.

Rezultati kažejo veliko večjo uspešnost pristopa s hevristikami. Z njim uspemo ustvariti 122 trojk (pravilnih in uporabnih), medtem ko pristop z globinskim razčlenjevanjem vrne le 37 trojk (pravilnih in uporabnih). Prav tako ima predlagani pristop veliko manjši delež napačnih trojk (15%) v primerjavi s trenutno najbolj uporabljanim pristopom (41% napačnih trojk). Opazimo tudi, da so skoraj vse trojke dobljene ob uporabi drevesnic v preseku in jih vrne tudi pristop s hevristikami. V našem primeru je metoda z uporabo hevristik spregledala le 2 (5%) izmed pravilnih in uporabnih trojk dobljenih s pomočjo drevesnic.

Za naše potrebe se je pristop s hevristikami izkazal za boljšega. Poglavitna prednost je hitrost, saj ob uporabi pristopa z globinskim razčlenjevanjem ne bi mogli obdelati vseh razpoložljivih n-terk. Kvaliteta in količina dobljenih trojk ob uporabi našega pristopa je boljša in večja kot pri uporabi globinskega račlenjevanja, kar je velika prednost.

4 Gradnja modela

Končni cilj te diplomske naloge je opisati del informacij zajetih v sopojavitvah besed. V prejšnjem sklopu smo opisali, za kakšno predstavitev konkretnih sopojavitev besed smo se odločili. Tako na mikro nivoju opišemo vsako primerno n-terko iz izvorne podatkovne zbirke s pomočjo trojke oblike osebek-povedek-predmet. Vsaka takšna trojka posredno zajema informacijo o odvisnostih med temi tremi deli. Da bi lahko to izrazili na bolj ekspliciten in robusten način, poskusimo najti splošnejše modele, s katerimi pokrijemo več konkretnih primerov.

Odločili smo se, da bo tudi na makro nivoju naša predstavitev podatkov osnovana na trojkah, zato smo poskusili najti postopek za izgradnjo splošnejših trojk, ki modelirajo naše konkretne podatke. V naslednjih nekaj poglavjih bomo predstavili naš pristop.

4.1 Razpoložljivo predznanje

Primerno predznanje nam je pri gradnji modelov z uporabo strojnega učenja velikokrat v precejšnjo pomoč. V njem je zajeto znanje o domeni, iz katere prihajajo podatki. Še posebej se pokaže pomembnost predznanja pri problemih, ki so povezani z razumevanjem naravnega jezika. Ljudje se že v otroštvu naučimo mnogo relacij med besedami in lastnosti konceptov, ki jih predstavljajo. Tako je recimo dejstvo, da predmet predstavljen z besedo »žoga« ne spada v skupino »živih bitij«, ljudem očitno, četudi tega ne moremo razbrati iz nekega konkretnega kosa besedila. Pri analizi besedila s pomočjo računalnika tega ne vemo, saj so za nas besede le nizi bajtov.

V prejšnjih poglavjih smo na podlagi n-terk izračunali trojke. Sedaj vemo, katere besede sestavljajo osebek, povedek in predmet. Na podlagi tega lahko sklepamo, kakšna relacija je s tem predstavljena. Na primer: besede v povedku najverjetneje predstavljajo akcijo, katere vršilec je opisan v osebku in katere cilj je opisan z besedami v predmetu. Ob tem pa še vedno ne vemo ničesar o konceptih, ki jih te besede predstavljajo, ampak zgolj njihov dogovorjeni zapis s črkami. Ker želimo na podlagi trojk zgraditi splošnejše modele, želimo poznati vsaj hierarhijo konceptov predstavljenih v njih, saj lahko s pomočjo le-te poiščemo splošnejše koncepte.

Primer predznanja, ki ga lahko uporabimo ob analizi besedila je WordNet. WordNet je velika leksikalna podatkovna baza za angleški jezik, ki jo sestavljajo na Princeton University od leta 1985 naprej. Vsebuje skupine angleških sopomenk, kratke definicije in razne semantične relacije med njimi. Eden izmed poglavitnih vzrokov za njen obstoj je uporaba pri avtomatski analizi besedila. WordNet je prosto dostopen[12] pod licenco BSD[13]; do njega pa lahko dostopamo tudi preko spleta.

Leta 2006 je bilo v WordNet vključenih 150.000 besed uvrščenih v 115.000 sinsetov, kar nam da 207.000 parov beseda-pomen. Sinset predstavlja množico vseh

[12] http://wordnet.princeton.edu/
[13] http://en.wikipedia.org/wiki/BSD_license

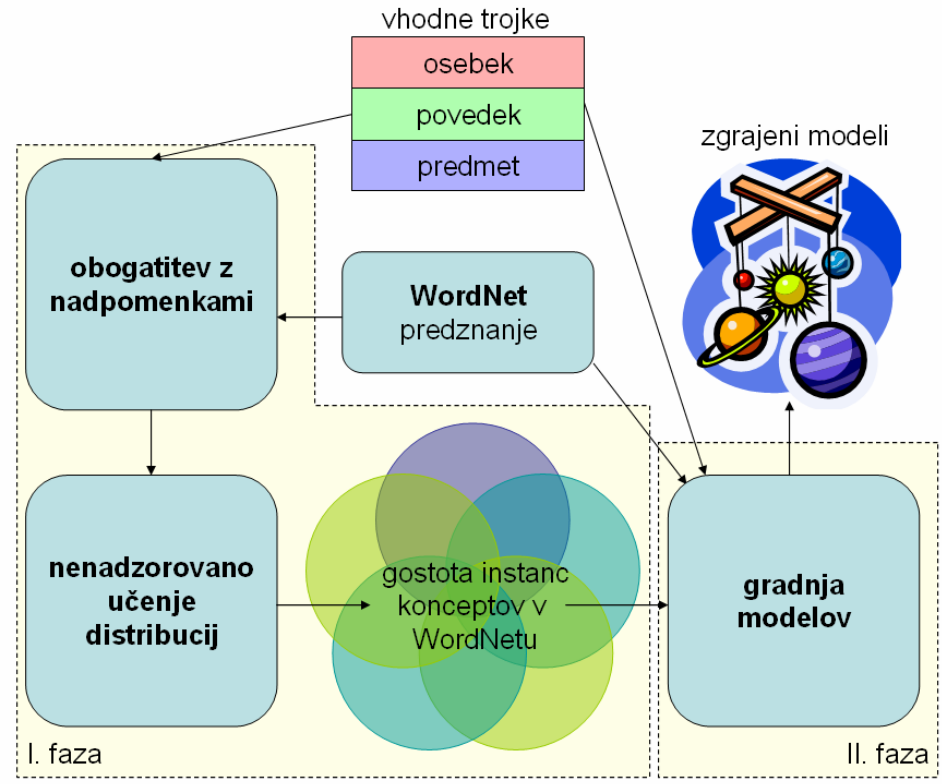
besed, ki so semantično ekvivalentne in predstavljajo isti koncept. Besede so ločeno obravnavane glede na to ali spadajo med samostalnike, glagole, pridevnike ali prislove. Med večino sinsetov obstajajo semantične relacije. Možne relacije so na primer: nadpomenka (»animal« in »organism«), podpomenka, protipomenka itd.

Samostalniki in glagoli so urejeni v hierarhije definirane na podlagi nadpomenk oz. »is-a« relacije. Primer dela hierarhije za besedo »dog« je prikazan na sliki 15. Če pogledamo hierarhijo za besedo »amoeba« prikazano na sliki 15, lahko opazimo, da je veliko bolj podrobna. Nekatere besede imajo zelo dolgo pot do korena hierarhije, medtem ko je ta pri nekaterih precej kratka. Ta lastnost WordNeta je posledica tega, da so ob snovanju imeli na voljo znanje ekspertov iz določenih področij in so lahko nekatere dele hierarhije veliko bolj dodelali kot druge. To posebnost WordNeta bomo še posebej omenili in obravnavali v poglavju 4.2.1, ker pomembno vpliva na nekatere vidike gradnje modelov.

WordNet smo v tej diplomski nalogi uporabili kot predznanje o hierarhiji konceptov pri gradnji modelov. Natančen opis, kako smo ga uporabili, bo podan v poglavju 4.2.2.

4.2 Metoda za posploševanje

Naša metoda za gradnjo modelov na podlagi trojk je sestavljena iz dveh faz, kot je to prikazano na diagramu na sliki 14. Modele lahko zgradimo tudi, če preskočimo prvo fazo, saj je njen namen predvsem poskus izboljšanja rezultatov z upoštevanjem distribucij in določitev variabilnih mej posploševanja besed.

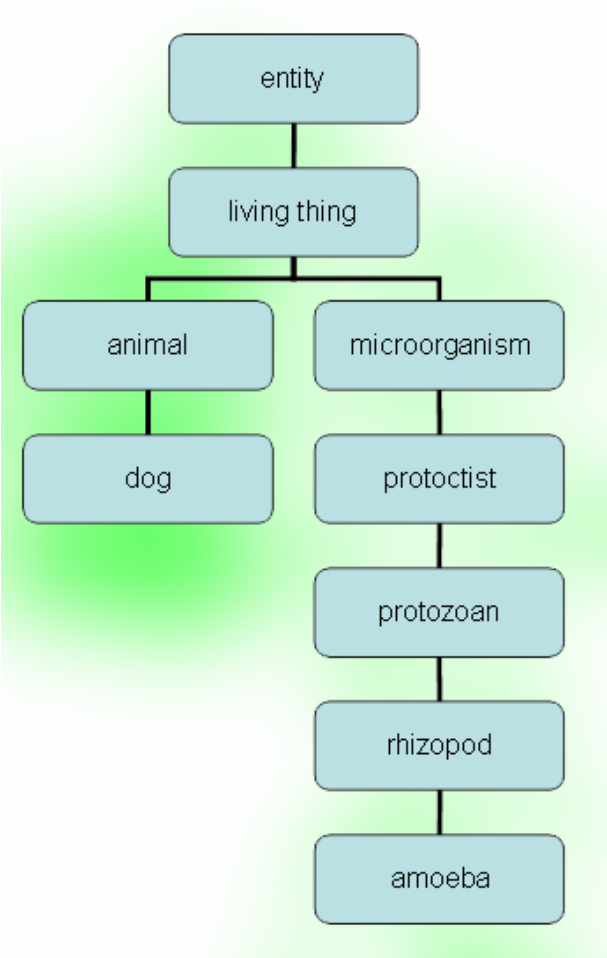


Slika 14: Shematični prikaz metode za gradnjo modelov.

4.2.1 Ocena distribucij

Oglejmo si najprej primer, kako lahko posplošimo besedo, tako da jo predstavimo z enim izmed višjih konceptov v hierarhiji nadpomenk. Če pogledamo del hierarhije za besedo »dog« (»pes«) v dodatku A, lahko s konceptom »animal« (»žival«) še vedno zajamemo koncept »dog«. Prav tako bi bila veljavna posplošitev besede »dog« lahko »entity« (»entiteta«), ki je precej abstrakten koncept. Kako daleč po hierarhiji se sprehodimo oz. do kakšnega nivoja abstrakcije posplošimo besedo, je odvisno od naših ciljev.

V našem primeru želimo najti posplošitve, ki bodo dovolj splošne, da bodo zajele primerno število konkretnih primerov znotraj enega modela, a hkrati ne bodo popolnoma abstraktne. Tako je v primeru vhodne besede »amoeba« posplošitev »rhizopod« najverjetneje še vedno preveč specifična, medtem ko je »living thing« preveč abstraktna. Mogoče bi bila primerna izbira »microorganism«, vendar je to odvisno od kriterijev, ki si jih zastavimo.



Slika 15: Poenostavljen primer hierarhije v WordNetu.

Potrebno je bilo najti pristop, s katerim lahko do neke mere kompenziramo različne stopnje podrobnosti v različnih vejah hierarhije. Diagram na sliki 15 prikazuje poenostavljen primer dela hierarhije. Če želimo za besedi »dog« in »amoeba« najti skupen koncept, je potrebno za eno besedo veliko več korakov posploševanja kot za drugo, saj je tisti del hierarhije veliko bolj podrobno razdelan. Predpostavimo, da so lastnosti distribucije besed v vhodnih n-terkah lepe in intuitivne. Če vse besede priredimo ustreznim vozliščem v WordNet hierarhiji, pričakujemo, da bo gostota primerov v zelo podrobnih vejah nizka, medtem ko bo gostota v bližini pogosto uporabljanih konceptov visoka. Izmišljen primer prikazuje diagram na sliki 15, kjer zasičenost barve predstavlja gostoto bližnjih instanc konceptov v vozliščih.

Do potrebnih distribucij lahko pridemo z uporabo nenadzorovanega učenja. Naši vhodni podatki so trojke, ki smo jih izračunali v poglavju 3.3. Ker je nenadzorovano učenje eden izmed pogostih prijemov v analizi besedil z metodami strojnega učenja [11], lahko uporabimo nekatere dodatne prijeme, za katere se je že izkazalo, da lahko izboljšajo rezultate. Na trojke lahko gledamo kot na zelo majhne kose besedila, zato jih lahko predstavimo z vrečo besed, ki smo jo omenili že na začetku poglavja 3.1. Vse tri dele trojke sedaj obravnavamo skupaj z upanjem, da bomo mogoče zaradi tega ohranili nekoliko informacije o odvisnosti. Vreča besed zgrajena na podlagi trojk se razlikuje od

vreče besed zgrajene na podlagi n-terk v tem, da vsebuje zgolj jedrne besede in ne vsebuje dodatnega šuma, ki ga predstavljajo ostale besede.

Dobljene vreče besed dodatno razširimo z uporabo sorodnih besed, kar je idejno sorodno razširjanju poizvedb opisanemu v [12]. V našem primeru naknadno dodamo besede, ki so posplošitve besed v trojki. Pri tem se omejimo na 1-2 koraka posploševanja z uporabo WordNeta. Na ta način povečamo prekrivanje med primeri in nekoliko omilimo veliko redkih podatkov. Prav tako s tem umetno dodamo nekoliko prekrivanja med sorodnimi koncepti in zgladimo distribucijo.

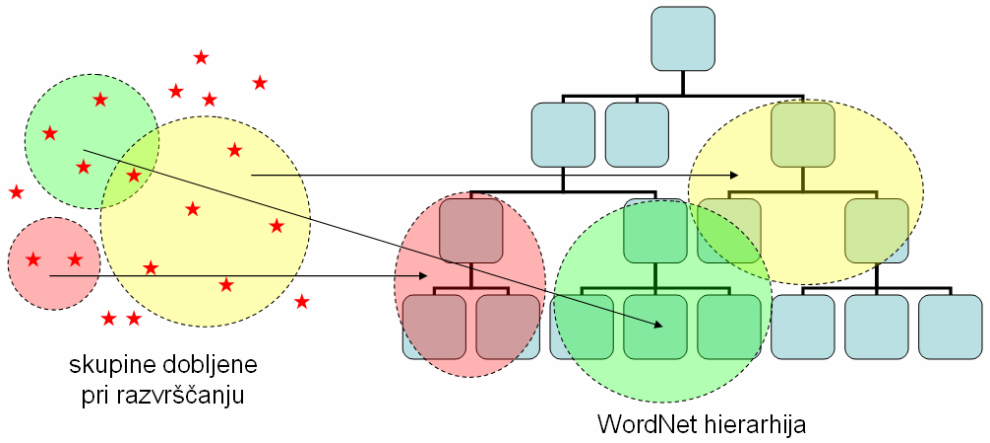
Metoda nenadzorovanega učenja, ki smo jo uporabili, spada med postopke za razvrščanje. Zaradi enostavnosti in hitrosti smo se odločili za razvrščanje z metodo voditeljev [14] (angl. k-means). Razdalje med posameznimi primeri so poračunane z uporabo kosinusne podobnosti, ki je definirana kot

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}.$$

Enačba 1: Definicija kosinusne podobnosti vektorjev A in B, ki predstavljata število posameznih besed v primeru.

V predstavitvi besedila z vrečo besed je vsak dokument oz. primer predstavljen s redkim vektorjem. Vsaka komponenta predstavlja določeno besedo in vsebuje število pojavitev le-te v besedilu. Če želimo izračunati podobnost med dvema primeroma oz. dokumentoma, izračunamo skalarni produkt vektorjev in ga normaliziramo. Na ta način dobimo kot med vektorjema, ki ga lahko uporabimo za merilo podobnosti: pravokotna vektorja si med seboj nista podobna, vzporedna vektorja pa sta enaka.

Po končanem postopku razvrščanja uporabimo dobljene centroide, ki jih uporabimo kot vodilo pri določitvi pokritosti WordNet hierarhije. Vsak centroid vsebuje podatke o povprečju primerov v ustrezni skupini. Predpostavimo, da je največji del mase skupine zbran v okolici centroida in zanemarimo vse preostale primere, saj so najverjetneje izolirani primeri. Nato preslikamo področja večje gostote iz prostora vreče besed v prostor vozlišč v WordNet hierarhiji. Konceptualno predstavitev tega koraka vidimo na sliki 16.



Slika 16: Preslikava skupin na WordNet hierarhijo.

[14] http://en.wikipedia.org/wiki/K-means_algorithm

Ko imamo na voljo ocene gostote primerov v vozliščih hierarhije, lahko to uporabimo za vodilo pri posploševanju besed. Naša ideja je, da je potrebno v območjih manjše gostote primerov narediti več korakov posplošitve, da bi dosegli primeren koncept in po možnosti zajeli čim več sorodnih primerov, saj so koncepti iz tega dela hierarhije manj zanimivi in manj uporabljani (glede na lastnosti vhodnih podatkov). Nasprotno moramo biti pri posploševanju bolj gosto pokritih delov hierarhije bolj konzervativni, saj so koncepti tam zanimivi in z njimi zlahka pokrijemo veliko konkretnih instanc.

4.2.2 Gradnja modelov

Ugotovitve iz poglavja 4.2.1 lahko uporabimo za posploševanje besed, ob katerem se trudimo izbrati zanimive splošnejše koncepte. Naša naslednja naloga je, da ta pristop priredimo za uporabo na trojkah, ki so sestavljene iz treh glavnih besed. Naiven pristop, ki bi poskusil posplošiti vse tri dele neodvisno, kaj hitro zabrede v probleme na račun kombinatorične kompleksnosti, saj se opazi vsako povečanje v asimptotični zahtevnosti izračunov pri tolikšni količini podatkov.

Kombinatorična kompleksnost se znatno poveča tudi na račun tega, da veliko besed sodi v več različnih sinsetov. Da bi uspešno določili, kateri koncept izmed več možnih predstavlja določena beseda, uporabimo njen kontekst v besedilu. Za razreševanje dvoumnosti pomena včasih zadostuje uporaba ostalih besed v povedi; občasno pa je potrebno poznavanje vsebine večjega kosa besedila (npr. celotnega odstavka). V našem primeru je konteksta zelo malo. Če upoštevamo še dejstvo, da je celo ob uporabi velike količine konteksta razreševanje dvoumnosti težek problem in ga obstoječi pristopi ne rešijo vedno pravilno, lahko sklepamo, da je v našem primeru nerešljiv. Zato se sprijaznimo, da ne moremo ugotoviti, kateri sinset oz. koncept predstavlja beseda.

Posploševanje trojk

Za gradnjo modelov uporabimo pristop, ki bi ga lahko uvrstili med nenadzorovano učenje. Ob tem se v veliki meri zanašamo tudi na redundanco in dobro statistiko zaradi velike količine razpoložljivih podatkov, saj je v podatkih prisotnega veliko šuma.

Ker je popolnoma neodvisno obravnavanje delov trojke preveč zahtevno (tako časovno kot prostorsko), moramo najti učinkovitejšo rešitev. Prav tako moramo pregledati veliko večino možnih modelov z metodo grobe sile, saj zaenkrat nimamo primerne hevristike, ki bi nas vodila pri izbiri posplošitev konceptov in iskanju primernega modela. Definiramo globino besede v hierarhiji kot število potrebnih korakov od korena do koncepta, ki ga predstavlja. Predpostavili smo, da so globine jedrnih besed v trojki približno podobne, saj sklepamo, da so primeri kot »Ana riše abstraktno entiteto« v realnih besedilih precej redki. Zato iščemo kandidate za modele v več prehodih; vsak izmed njih pri fiksirani skupni globini. Skupno globino trojke smo definirali kot vsoto

globin vseh treh jedrnih besed. S tem smo določili zaporedje preiskovanja prostora možnih modelov. Da dobimo trojko z želeno skupno globino, moramo koncepte v njej primerno posplošiti z uporabo WordNeta.

Razreševanje dvoumnosti pomenov besed v našem primeru ni možna, zato moramo najti primeren pristop, ki bo deloval kljub tej omejitvi. Uporabili smo trivialno rešitev in enostavno razpihnili primere, tako da vsak nov primer prestavlja en možen pomen. Ker ima precej besed veliko različnih pomenov (sinsetov), kot to lahko vidimo v tabeli 12 za nekaj primerov, na ta način nekajkrat povečamo količino vhodnih podatkov.

beseda	možni pomeni v WordNetu	besedna vrsta
dog	* a member of the genus Canis	sam.
	* a dull unattractive unpleasant girl or woman	sam.
	* informal term for a man	sam.
	* someone who is morally reprehensible	sam.
	* a smooth-textured sausage of minced beef or pork	sam.
	* a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward	sam.
	* metal supports for logs in a fireplace	sam.
man	* an adult male person	sam.
	* someone who serves in the armed forces	sam.
	* any human being	sam.
	* all of the inhabitants of the earth	sam.
	* a male subordinate	sam.
	* take charge of a certain job	glag.
	* provide with men	glag.
radio	* medium for communication	sam.
	* an electronic receiver	sam.
	* a communication system	sam.
	* transmit messages via radio waves	glag.
	* indicating radiation or radioactivity	prid.

Tabela 12: Nekatere besede uporabljamo, da z njimi predstavimo več različnih konceptov. Tukaj je podanih nekaj takšnih primerov.

Možna rešitev bi bila, da bi uporabili le tiste pomene, ki se pogosteje pojavljajo. Vendar smo želeli najprej razviti celotno metodo, nato pa iskati morebitne pohitritve s selektivno izbiro pomenov, saj bi lahko potem primerjali vplive sprememb na rezultate. Ob upoštevanju vseh možnih pomenov je možno poiskati kandidate za modele na podmnožici vhodnih podatkov; potrebnega časa za obdelavo vseh podatkov pa še nismo uspeli oceniti, ker bi bilo potrebno najprej našo metodo dobro optimizirati, saj so poudarki trenutne implementacije predvsem na pravilnosti in jasnosti.

Pomembna vodila, ki jih lahko uporabimo ob iskanju najprimernejših modelov, so število pojavitev izvorne n-terke na svetovnem spletu, število različnih pomenov besed in pogostost pojavljanja posameznega pomena v realnih besedilih ter izračunana gostota

primerov v bližini obravnavanih konceptov. Kako jih uporabimo, je odvisno od tega, kakšne modele trojk si želimo. Število pojavitev n-terke je posreden pokazatelj njene pomembnosti in zanesljivosti informacije, ki jo vsebuje. Na podlagi podatkov o možnih pomenih besed lahko sklepamo, kateri pomeni in kako verjetno so pravilna razlaga besed. Gostota primerov v okolici pa nam pomaga pri odločanju, do kakšne mere naj posplošimo določen koncept.

V WordNetu so vsebovane samo osnovne oblike besed. Ker v besedilu najdemo besede v raznih sklonih, časih ipd. je potrebna lematizacija. Z njo poiščemo osnovno obliko besed. Najenostavnejši pristop k temu je uporaba ročno sestavljenih pravil. Primer takšnega pravila za angleški jezik je, da v večini primerov dobimo osnovno obliko glagola tako, da odstranimo končnico »ed«. Takšen pristop s pravili uporablja Porter Stemmer[15], ki ga lahko uporabimo za lematizacijo angleških besed. Na žalost naravni jeziki vsebujejo ogromno izjem in zato pravila ne veljajo v vseh primerih. V naši implementaciji smo zato uporabili lematizator, ki uporablja pravila določena z uporabo strojnega učenja na pravilnih primerih. Ugotovili smo, da so rezultati lematizacije precej boljši in povečajo verjetnost nahajanja besede v WordNetu.

Kot v večini pristopov za strojno analizo besedila, smo tudi mi izločili nekatere neinformativne besede (angl. stop-words), ki se zelo pogosto pojavljajo. Primeri angleških neinformativnih besed so »a«, »the«, »is« itd. Seznam blokiranih besed smo nekoliko priredili našim potrebam, saj so besede kot na primer »he« in »is« za nas pomembne, ker so gradniki relacij, za katere bi pričakovali, da so pomembne. V primeru, da ne odstranimo nobene besede, se zgodi, da dobimo veliko relacij, ki nosijo zelo malo informacije (na primer »it is this«), vendar so zaradi pogostosti uvrščene na sam vrh kandidatov za modele.

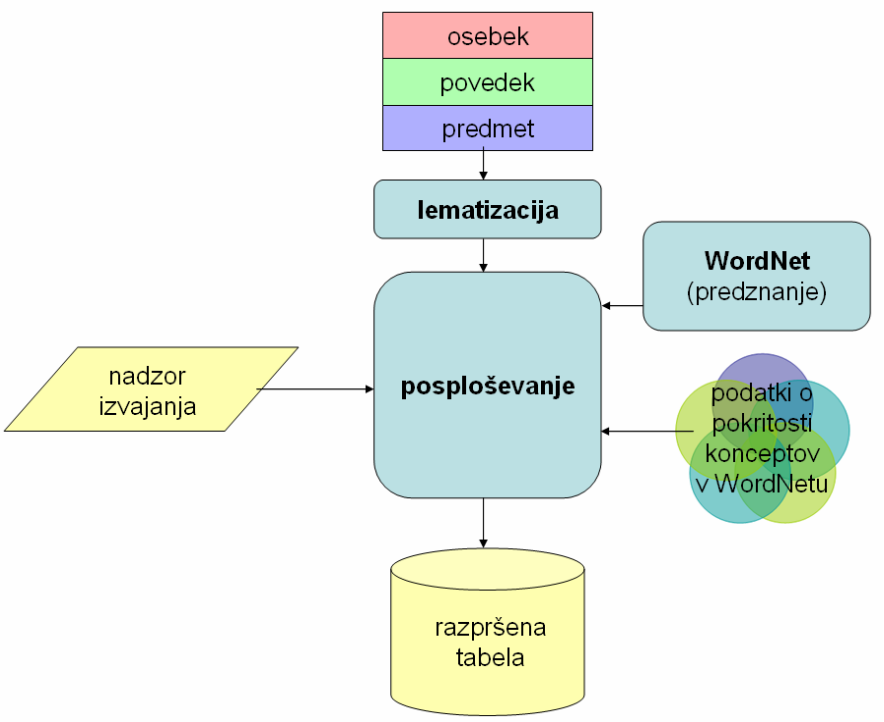
Celoten algoritem je predstavljen s psevdokodo v algoritmu 2. Diagram na sliki 17 prikazuje dogajanje s podatki med izvajanjem algoritma. Nekaj konkretnih primerov je predstavljenih v tabeli 13, večji primer rezultatov pa je viden v dodatku B.

[15] http://tartarus.org/~martin/PorterStemmer/

```
function AbstractTriple(wordSenses[3], desiredDepth[3])
//posplošimo vse tri dele podane trojke
newTriple[3] = new()
for i=0..2
  sense = wordSenses[i]
  depth = desiredDepth[i]
  //pogledamo informacijo o pokritosti
  //konceptov v WordNetu, ki smo jo dobili
  //z uporabo nenadzorovanega učenja
  density = GetDensity(sense)
  //določimo število dovoljenih korakov posploševanja
  if density < DENSITY_TRESH
    maxSteps = LOW_DENSITY_MAXSTEPS;
  else
    maxSteps = HIGH_DENSITY_MAXSTEPS;
  //preverimo izpolnjenost kriterijev
  currentDepth = WordNet.getDepth(sense)
  if |depth - currentDepth| <= maxSteps
    newTriple[i] = WordNet.getHyper(sense, depth)
return newTriple


function BuildModels()
//podatke obdelamo v več prehodih
//glede na skupno globino trojk
for depthSum=MIN_DEPTH..MAX_DEPTH
  foreach (xd, yd, zd) where xd+yd+zd == depthSum
    //obdelamo vse trojke
    foreach triple in allTriples
      //obdelamo vse možne pomene besed
      foreach si in WordNet.getSenses(triple[0])
        foreach sj in WordNet.getSenses(triple[1])
          foreach sk in WordNet.getSenses(triple[2])
            //posplošimo trojko
            model = AbstractTriple({si, sj, sk}, {xd, yd, zd})
            //in jo dodamo med kandidate
            mHash[model] += triple;
  //ohranimo samo modele z dovolj veliko podporo
  ApplyTreshold(mHash, MIN_SUPPORT)
```

Algoritem 2: Gradnja modelov.



Slika 17: Shematski prikaz algoritma za gradnjo modelov.

activity | *act move* | *person individual someone somebody mortal soul* - creation use asian - crime commit connection - crime commit indian - crime commit individual - crime commit man
causal_agent cause causal_agency | *make create* | *abstraction abstract_entity* - crab give birth - creator create universe - creature compose body - creditor attempt collection - creditor give notice
authority | *grant give* | *person individual someone somebody mortal soul* - authority grant administrator - authority grant agent - authority grant borrower - authority grant individual - authority grant person

Tabela 13: Primeri modelov in trojk, ki so konkretne instance, na podlagi katerih je bil izbran model.

Največji del k časovni zahtevnosti prinese vključitev vseh možnih pomenov besed, saj nismo mogli izvesti razreševanja dvoumnosti. Prostorska zahtevnost algoritma je pogojena z razpršeno tabelo, ki hrani vse trenutne kandidate za modele. Možnosti

paralelizacije so omejene predvsem zaradi potrebe po hranjenju vseh kandidatov za modele. Tako smo prisiljeni ali poskrbeti za sinhronizacijo oz. distribucijo med več razpršenimi tabelami ali pa se sprijazniti s podvajanjem podatkov v pomnilniku.

Končni rezultat algoritma je množica kandidatov za modele. Za vsakega izmed njih poznamo skupno globino trojke, s katero je predstavljen, in število konkretnih primerov (kjer vsak možen pomen besede štejemo za ločen primer), ki jih pokrije. Sedaj moramo izmed njih izbrati tiste, za katere mislimo, da so najbolj uporabni, saj jih je sicer veliko preveč.

Da smo zmanjšali število kandidatov, smo postavili nekaj omejitev. Postavili smo spodnjo mejo, koliko primerov mora model zajeti, da ga sprejmemo med primerne. Ta izbira je predvsem odvisna od vsebine in količine podatkov, s katerimi razpolagamo. Za nadaljnjo zožitev izbire lahko omejimo tudi skupne globine in izberemo le modele, ki so podprti z največ konkretnimi primeri. Konkretna izbira je odvisna od tega, v kakšne namene bomo kasneje uporabili dobljene modele.

V tem poglavju smo predstavili algoritem, s pomočjo katerega sestavimo množico trojk, ki modelira ostale trojke. Modeli so trojke, opisane z bolj splošnimi koncepti in pokrivajo določeno število konkretnih primerov. Na ta način smo prišli do relacij tipa osebek-povedek-predmet, ki jih lahko uporabimo kot predznanje v raznih aplikacijah.

4.2.3 Ocena kvalitete

Pristop, predstavljen v tej diplomski nalogi, smo ovrednotili. V tem poglavju bomo predstavili rezultate ročne evaluacije modelov sopojavitev besed zgrajenih na podlagi Google-ove zbirke n-terk.

Najprej si poglejmo uspešnost lematizacije in ostale dejavnike, ki tudi vplivajo na to, katere trojke uspemo uporabiti v našem pristopu. V primeru, da je beseda napačno lematizirana ali na primer že v izvornem besedilu napačno črkovana, je navadno ne moremo najti v WordNetu in zato trojke, ki jo vsebuje, ne moremo obravnavati. Pogost primer problematičnih besed so tudi lastna imena, saj jih WordNet iz praktičnih razlogov vsebuje relativno malo. Nekaj informacij o uspešnosti poravnave med besedami v trojkah in koncepti v WordNetu je predstavljenih v tabeli 14.

	osebek	povedek	predmet
indeksirana jedra besednih zvez	348.588	116.279	357.356
besede najdene v WordNetu	34.327	7.186	35.080
število zajetih trojk	23.435.234	25.177.604	25.474.913
vseh trojk	33.781.792		

Tabela 14: Podatki o številu uspešno najdenih besed v WordNetu.

Za nalogo, ki jo poskušamo rešiti v tej diplomski nalogi, (še) ne obstaja testna množica, za katero bi bili že znani želeni oz. pravilni rezultati in bi jo lahko uporabili za evaluacijo. Prav tako je skoraj nemogoče ustvariti sintetični nabor podatkov, ki bi dovolj dobro simuliral realne podatke, saj zaenkrat še ne poznamo vseh zakonitosti, ki jih pogojujejo. Zato smo se odločili, da bomo rezultate ročno ocenili, tako da bomo

pregledali modele, ki jih naš pristop izračuna na podlagi naključne podmnožice vhodnih podatkov.

Da lahko ročno ocenimo rezultate, je potrebno postaviti dovolj specifične kriterije, kako ocenimo določen primer. Ob sestavljanju kriterijev smo imeli v mislih predvsem možno nadaljnjo uporabo naših rezultatov v namene avtomatske analize besedila, gradnje ontologij, uporabe za dopolnitev baze znanja pri sklepanju in iskanju relacij, v katerih pogosto nastopajo določeni koncepti. Odločili smo se za sledeče kriterije, ki služijo kot vodilo ob ocenjevanju rezultatov:

- primeren nivo abstrakcije
    - vsi trije deli trojke naj bodo na približno istem nivoju abstrakcije
    - abstraktni koncepti niso uporabni (kot na primer »entity«)
    - lastna imena so preveč specifična
    - redko uporabljani koncepti niso zaželeni (na primer botanična imena rastlin)
- uporabnost informacije
    - trivialne relacije z zaimki ipd. so neuporabne (na primer »it is this«)
    - relacija mora biti smiselna
- enostavnost interpretacije
    - zaželeni so nedvoumni koncepti
    - metafore ipd. so po večini neprimerne za nadaljnjo uporabo

Upoštevati moramo, da ni potrebno vseh zaželenih lastnosti enako močno obtežiti. Prav tako ob ocenjevanju ponavadi upoštevamo vse kriterije hkrati in ne ocenjujemo izpolnjenosti vsakega posebej. Poskušamo dobiti predvsem kvalitativno oceno, ki jo ljudje lažje podamo, če smo nekoliko manj omejeni. Ob tem se moramo zavedati, da je sama ocena precej subjektivna in v veliki meri odvisna od ocenjevalca, zato jo lahko uporabimo zgolj kot približen pokazatelj kvalitete rezultatov in ne kot absolutno merilo za primerjavo. Na podlagi postavljenih kriterijev smo ocenili najbolj obetavne modele z oceno:

+1	ustreza večini kriterijev
0	število izpolnjenih kriterijev je približno enako številu neizpolnjenih
-1	večina kriterijev ni izpolnjenih

Zaradi odločitve, da v našem pristopu poskusimo rešiti probleme, nastale zato ker nismo izvedli razreševanja dvoumnosti, tako da upoštevamo vse možne pomene besed, smo najprej želeli preveriti v kolikšni meri izračunani modeli zajamejo pravilen pomen besede. Ročno smo pregledali najbolj obetavne modele in ocenili, v kolikšni meri vključujejo pravilno razrešene dvoumnosti konceptov besed. Kot pravilne smo označili tiste rezultate, ki se ujemajo s tem, kateri koncept bi besedi pripisali ljudje ob poznavanju razpoložljivega konteksta (t.j. ostalih besed n-terke). Na enak način smo ocenili tudi skupine najobetavnejših modelov. Za člane skupine smo vzeli tiste modele, ki so bili podprti z istimi trojkami in predstavljajo njihovo skupno posplošitev. Upoštevali smo le tiste skupine, ki so vsebovale vsaj tri modele. Pregledali smo tudi trojke, ki so spadale v te skupine modelov, in ocenili, v kolikšni meri so si pomensko podobne in zato primerne, da se jih opiše s skupnim modelom. V oceno smo vključili le skupine z vsaj pet trojkami. Dobljene rezultate ob pregledu vzorca A podaja tabela 15.

število posameznih modelov		
pravilni	pretežno pravilni	napačni
31	28	11

število skupin modelov (opisujejo vsaj pet trojk)		
pravilne	pretežno pravilne	napačne
6	4	0

Tabela 15: Pravilnost izbire konceptov v modelih zgrajenih na osnovi vzorca A.

Ob oceni skupin trojk, ki jih opisujejo posamezni obetavni modeli zgrajeni na vzorcu A, smo v vseh primerih ugotovili, da so kvalitetne. To pomeni, da so bile pomensko precej homogene in zato pričakujemo obstoj modela, ki jih opisuje.

V tabeli 16 podajamo rezultate ročne ocene najobetavnejših modelov zgrajenih na vzorcu A. Kvaliteto smo ocenili na podlagi prej opisanih kriterijev. Ocenili smo tudi skupine modelov, ki opisujejo iste množice trojk in se zato razlikujejo le v izbranih pomenih besed.

število posameznih modelov		
dobri (+1)	mejni(0)	slabi(-1)
44	19	3+9

število skupin modelov		
dobri (+1)	mejni(0)	slabi(-1)
13	2	6

Tabela 16: Kvaliteta najobetavnejših modelov zgrajenih na osnovi vzorca A.

Izmed negativno ocenjenih modelov smo se za takšno oceno pri 9 izmed njih odločili zato, ker so vsebovali koncept »abstract entity«, česar si seveda ne želimo. Ta problem lahko enostavno rešimo z dodatnim filtriranjem zgrajenih modelov ali z upoštevanjem zgolj modelov z večjo skupno globino besed. Zanimiva lastnost vzorca A je tudi to, da so bili najobetavnejši modeli zgrajeni za ciljno skupno globino od 14 do 29 enaki. V veliki meri je to posledica majhnosti vzorca in posledično zelo omejenega prostora možnih modelov.

Za primerjavo smo naš algoritem spremenili tako, da smo pri gradnji upoštevali zgolj prvi pomen vsake besede. Rezultati dobljeni na vzorcu A po omenjeni spremembi algoritma so prikazani v tabeli 17.

število posameznih modelov		
dobri (+1)	mejni(0)	slabi(-1)
29	26	13+16

Tabela 17: Kvaliteta najobetavnejših modelov zgrajenih na osnovi vzorca A pri upoštevanju zgolj prvega pomena besed.

Razmerje med števili modelov s posameznimi ocenami se po tej spremembi poslabša, kar kaže na to, da je potrebno upoštevati več oz. vse možne pomene besed. Med

rezultati je bilo 16 modelov, ki so vsebovali »abstract entity« in so bili posledične negativno ocenjeni.

Da bi preverili vpliv izbire vzorca na končne rezultate, smo ocenili dobljene modele še za tri druge vzorce. Tabela 18 podaja rezultate ocenjevanja za vzorec B, pri katerem je bila negativna ocena zaradi prevelike abstraktnosti podeljena trikrat. Za vzorca A' in A" pa se je izkazalo, da so rezultati precej slabši, kot pri vzorcih A in B. V obeh primerih je bila manj kot tretjina modelov ocenjena s **+1** in prav tako manj kot tretjina z **0**. Skoraj dve tretjini modelov smo ocenili z **-1**, v veliki večini primerov zato, ker so vsebovali »abstract entity«.

število posameznih modelov		
dobri (+1)	mejni(0)	slabi(-1)
11	5	2+3

Tabela 18: Kvaliteta najobetavnejših modelov zgrajenih na osnovi vzorca B.

Vzorci A, A' in A" so vključevali 1000 trojk. Za nekoliko več kot 50% trojk v teh vzorcih smo uspeli najti opise konceptov za vse besede v njih. Preostalih trojk nismo uporabili pri gradnji modelov, saj jih ni bilo možno posplošiti z uporabo WordNeta. Vsi trije vzorci so bili dobljeni tako, da smo naključno izbrali več skupin zaporednih trojk v vhodni datoteki. V primeru, da bi izbrali zgolj naključne trojke, bi pri tako majhnem vzorcu (večjega namreč ne bi mogli ročno oceniti) verjetno bile med seboj pomensko zelo različne in ne bi uspeli na podlagi njih zgraditi uporabnih modelov. Ta pričakovanja v grobem ustrezajo našim ugotovitvam ob pregledu rezultatov (tukaj jih ne navajamo, ker so bili pregledani zalo na grobo in zato nimamo dovolj dobrih ocen) dobljenih na popolnoma naključnih vzorcih. Ker so trojke v vhodni datoteki bile delno urejene, ob vključitvi zaporednih trojk v vzorec deloma dosežemo prekrivanje njihove vsebine, kar tudi pričakujemo pri obdelavi vseh trojk hkrati.

Izbira trojk v vzorcu B je bila opravljena nekoliko drugače. Vanj je vključenih 5000 trojk. Te smo izbrali tako, da smo za nekaj naključno izbranih osebkov, katerih jedra so bila vključena v WordNet, naključno izbrali nekaj trojk, ki vsebujejo ta osebek. Tako smo dosegli delno prekrivanje vsebine trojk, ki je potrebno iz istih razlogov kot pri izbiri ostalih vzorcev.

Vse vzorce smo ocenili le pri skupnih globinah besed 5, 10, 15 in 20. Razlog za to je, da bi sicer ročno ocenjevanje vzelo preveč časa. Vpliv tega na natančnost ocene je precej verjetno zanemarljiv, saj se je izkazalo, da je pri tako majhnih vzorcih veliko zgrajenih modelov enakih pri različnih skupnih globinah besed. Prav tako je bila varianca kvalitete med različnimi skupnimi globinami dokaj majhna. Pri vzorcih A in B so bili modeli na vseh ocenjevanih skupnih globinah približno enako dobri. V vzorcih A' in A" pa so bili vsi modeli približno enako slabi ne glede na skupno globino.

Pomemben podatek je tudi, kako smo izbrali najobetavnejše kandidate vključene v ocenjevanje. Že pri majhnem številu vhodnih trojk je med tekom algoritma v razpršeni tabeli ogromno kandidatov za modele. Pri vzorcu B se to število giblje med približno 20.000 in 4.000.000. Za ostale nekoliko manjše vzorce je število kandidatov za modele med približno 10.000 in 1.500.000. Izjema je uporaba nekoliko spremenjenega algoritma (upoštevanje zgolj prvega možnega pomena besede), pri katerem je to število med 10 in 90.000. Različno število kandidatov med posameznimi prehodi algoritma (različna

skupna globina) je posledica različno velikega prostora možnih modelov. Pri višjem nivoju abstrakcije je razpoložljivih konceptov veliko manj. Ker so med kandidate vključeni popolnoma vsi možni modeli, je na koncu potrebno med njimi izbrati najobetavnejše. Odločili smo se, da jih izberemo za vsako skupno globino posebej, saj jih lahko nato primerjamo in natančneje preučimo vpliv omejitve izbranih modelov na določeno skupno globino. Za vsak model tekom algoritma izračunamo tudi, koliko konkretnih trojk zajema oz. ga podpirajo. Ta podatek smo uporabili kot vodilo za izbiro obetavnih modelov in tako v ocenjevanje vključili 5-10 modelov z največjo podporo iz izbranih (5, 10, 15, 20) skupnih globin.

Iz predstavljenih rezultatov ocenjevanja lahko sklepamo, da najbolj obetavni modeli zadovoljivo ustrezajo zadanim kriterijem. Sklepamo tudi, da lahko pričakujemo še nekoliko boljše rezultate ob uporabi tega pristopa na vseh trojkah hkrati (in ne samo majhnem vzorcu). Majhni vzorci so namreč pri tako veliko količini podatkov pristranski, saj je odvisno od naključja, katere trojke vključimo: smiselne ali ne, vsebinsko podobne ali različne, vsebujoče zelo splošne ali specifične besede itd. Ob pregledu dobljenih rezultatov smo tudi ugotovili, da bi bilo potrebno nekatere koncepte predhodno obtežiti, saj sicer pogosto kvarijo rezultate. Tako se na primer beseda »he« v najobetavnejših modelih pogosto pojavlja kot koncept: kemijski element helij, čeprav bi pričakovali, glede na naše človeško predznanje o vsebini besedil in pomenu besed glede na njihov kontekst, da bi moral v modelih nastopati koncept recimo osebe.

Drugi pristop k oceni rezultatov bi bil, če bi jih ocenjevali posredno, tako da ocenimo uspešnost aplikacije, ki jih uporablja. Na primer, če bi naše modele uporabili pri analizi besedila, katere uspešnost lahko merimo, ker imamo na voljo pravilne rezultate, potem lahko posredno ocenimo tudi kvaliteto naših rezultatov. Ker metode, ki bi uporabljale naše rezultate, zaenkrat še ne obstajajo, tega ne moremo storiti. Pričakujemo pa, da bo to možno v bližnji prihodnosti, saj se že kažejo potencialne možnosti uporabe.

5 Zaključek

V tej diplomski nalogi smo predstavili pristop, s katerim lahko na podlagi n-terk zgradimo modele, ki zajemajo del informacij o sopojavitvah besed. Ogledali smo si nekaj možnih predstavitev, ki ohranjajo informacijo o sopojavitvah besed. Prikazali smo prednosti uporabe trojk in razložili njihov izračun. S primerjavo z eno izmed najsodobnejših metod smo pokazali, da v našem primeru metoda, ki smo jo razvili in temelji na hevristikah, daje boljše rezultate. V nadaljevanju smo opisali, kako s pomočjo posploševanja besed oz. konceptov lahko modeliramo trojke. Kvaliteto rezultatov smo nato ocenili z ročno evaluacijo.

Področje raziskav, katerega se dotika ta diploma, je relativno novo in se aktivno razvija. Iz tega razloga je težko primerjati rezultate z morebitnimi že obstoječimi, saj marsikateri pristop še ni bil objavljen, čeprav ga ljudje raziskujejo. Dodaten problem je pomanjkanje aplikacij, ki bi uporabljale trojke. Čeprav že obstajajo mnogi programi za obdelavo in shranjevanje trojk, pa se delo na njihovi uporabi šele pričenja.

Velik izziv je bil in še vedno ostaja velika količina podatkov. Google-ova zbirka n-terk je trenutno eden izmed redkih dostopnih naborov podatkov tega obsega. Pri implementaciji algoritmov za obdelavo takšnih količin podatkov je potrebno vložiti precej truda v optimizacijo in premišljene pristope, da lahko naše obdelave izvedemo v razumnem času. Posebnost n-terk, ki smo jih mi uporabljali, je tudi to, da izvirajo iz svetovnega spleta. Zaradi tega se v nekaterih vidikih drastično razlikujejo od ročno ali polavtomatsko ustvarjenih zbirk trojk, ki so trenutno dostopne. Predvsem je očitna veliko večja stopnja šuma.

Ugotovili smo, da je trojke možno učinkovito izračunati. Skoraj 100-kratna pospešitev, ki smo jo dosegli, odpira nove možnosti tudi na drugih področjih. Dosedanji postopki so velika ovira pri poskusih predstavitve večjih količin besedila s trojkami. Sklepamo, da bo možno s še nekaj dodatnega razvoja uporabiti naš pristop s hevristikami v splošnih primerih. Zanimivo bi bilo na primer uporabiti vse članke objavljene na Wikipediji in jih uporabiti namesto Googleovih n-terk ter primerjati dobljene rezultate.

Predstavljeno metodo za posploševanje in modeliranje trojk je potrebno še natančneje preizkusiti. Upamo, da se bodo v kratkem pojavile aplikacije, ki bodo znale uporabiti naše rezultate, in bomo lahko z njihovo pomočjo posredno, a bolj smiselno, ocenili kvaliteto rezultatov. Z uporabo kvantitativnih ocen bo možno izbrati parametre, ki so trenutno bili nastavljeni predvsem po občutku na podlagi razmisleka.

Iz rezultatov naše evaluacije lahko sklepamo, da so naši modeli sopojavitev predstavljenih s trojkami najverjetneje uporabni. Uspešnost ali neuspešnost pristopa, ki smo ga predlagali v tej diplomski nalogi, se bo izkazala šele čez nekaj časa, ko bodo dokončane aplikacije, ki bodo znale uporabiti naše rezultate. V vsakem primeru pa smo natančno obdelali en možen pristop, ki sedaj lahko služi kot referenca ob nadaljnjih raziskavah raznih možnosti zajema informacij iz besedila z uporabo računalnika.

A Celotna hierarhija za besedo »dog« v WordNetu

```
-> dog domestic_dog Canis_familiaris (001FCCE7)
  -> canine canid (001FCA12)
    -> carnivore (001FAAA0)
      -> placental placental_mammal eutherian eutherian_mammal (001CCA24)
        -> mammal mammalian (001C6892)
          -> vertebrate craniate (001674C2)
            -> chordate (00165F91)
              -> animal animate_being beast brute creature fauna (00003C1C)
                -> organism being (0000117B)
                  -> living_thing animate_thing (000010A2)
                    -> whole unit (00000DE1)
                      -> object physical_object (00000A7C)
                        -> physical_entity (0000078A)
                          -> entity (000006CC)
  -> domestic_animal domesticated_animal (00141AA5)
    -> animal animate_being beast brute creature fauna (00003C1C)
      -> organism being (0000117B)
        -> living_thing animate_thing (000010A2)
          -> whole unit (00000DE1)
            -> object physical_object (00000A7C)
              -> physical_entity (0000078A)
                -> entity (000006CC)
-> frump dog (009A54A1)
  -> unpleasant_woman disagreeable_woman (00A3DFB4)
    -> unpleasant_person disagreeable_person (0092F6E7)
      -> unwelcome_person persona_non_grata (0092F599)
        -> person individual someone somebody mortal soul (00001EA6)
          -> organism being (0000117B)
            -> living_thing animate_thing (000010A2)
              -> whole unit (00000DE1)
                -> object physical_object (00000A7C)
                  -> physical_entity (0000078A)
                    -> entity (000006CC)
          -> causal_agent cause causal_agency (00001CB3)
            -> physical_entity (0000078A)
              -> entity (000006CC)
-> dog (0098F07F)
  -> chap fellow feller fella lad gent blighter cuss bloke (00972F39)
    -> male male_person (0092DA68)
      -> person individual someone somebody mortal soul (00001EA6)
        -> organism being (0000117B)
          -> living_thing animate_thing (000010A2)
            -> whole unit (00000DE1)
              -> object physical_object (00000A7C)
                -> physical_entity (0000078A)
                  -> entity (000006CC)
        -> causal_agent cause causal_agency (00001CB3)
          -> physical_entity (0000078A)
            -> entity (000006CC)
-> cad bounder blackguard dog hound heel (0096DA0C)
  -> villain scoundrel (00A4160A)
    -> unwelcome_person persona_non_grata (0092F599)
      -> person individual someone somebody mortal soul (00001EA6)
        -> organism being (0000117B)
          -> living_thing animate_thing (000010A2)
            -> whole unit (00000DE1)
              -> object physical_object (00000A7C)
                -> physical_entity (0000078A)
                  -> entity (000006CC)
        -> causal_agent cause causal_agency (00001CB3)
          -> physical_entity (0000078A)
            -> entity (000006CC)
-> frank frankfurter hotdog hot_dog dog wiener wienerwurst weenie (007522BA)
```

```
  -> sausage (00751EEB)
    -> meat (0074BA3E)
      -> food solid_food (00734B17)
        -> solid (00E598F4)
          -> matter (0000515B)
            -> physical_entity (0000078A)
              -> entity (000006CC)
-> pawl detent click dog (003B886C)
  -> catch stop (002D8386)
    -> restraint constraint (003E48B4)
      -> device (003091E8)
        -> instrumentality instrumentation (00368DC8)
          -> artifact artefact (000055B3)
            -> whole unit (00000DE1)
              -> object physical_object (00000A7C)
                -> physical_entity (0000078A)
                  -> entity (000006CC)
-> andiron firedog dog dog-iron (00295A1C)
  -> support (004285A5)
    -> device (003091E8)
      -> instrumentality instrumentation (00368DC8)
        -> artifact artefact (000055B3)
          -> whole unit (00000DE1)
            -> object physical_object (00000A7C)
              -> physical_entity (0000078A)
                -> entity (000006CC)
```

B Primeri zgrajenih modelov sopojavitev besed

d = 2, vzorec B
abstraction abstract_entity * change * abstraction abstract_entity
change * transfer * increase
abstraction abstract_entity * change alter modify * abstraction abstract_entity
abstraction abstract_entity * be * abstraction abstract_entity
abstraction abstract_entity * make create * abstraction abstract_entity

d = 15, vzorec B
person individual someone somebody mortal soul * date date_stamp * woman adult_female
woman adult_female * appearance visual_aspect * person individual someone somebody mortal soul
woman adult_female * supply provide render furnish * woman adult_female
woman adult_female * date date_stamp * person individual someone somebody mortal soul
person individual someone somebody mortal soul * inform * person individual someone somebody mortal soul

d = 16, B
woman adult_female * date * woman adult_female
athlete jock * request bespeak call_for quest * contestant
contestant * request bespeak call_for quest * athlete jock
person individual someone somebody mortal soul * appearance visual_aspect * person individual someone somebody mortal soul
person individual someone somebody mortal soul * sensing perception * person individual someone somebody mortal soul

d = 10, vzorec A
authority (9824609) * grant give (2317094) * person individual someone somebody mortal soul
authority (9824609) * give (2316868) * person individual someone somebody mortal soul (7846)
authority (9824609) * give gift present (2200686) * person individual someone somebody mortal soul (7846)
authority (9824609) * agree hold concur concord (805376) * person individual someone somebody mortal soul (7846)
expert (9617867) * grant give (2317094) * person individual someone somebody mortal soul (7846)
expert (9617867) * give (2316868) * person individual someone somebody mortal soul (7846)
statement (6722453) * change alter modify (126264) * time_period period_of_time period (15113229)
message content subject_matter (6598915) * change alter modify (126264) * time_period period_of_time period (15113229)
informing making_known (7212190) * end stop finish terminate cease (2609764) * time_period period_of_time period (15113229)
informing making_known (7212190) * be (2604760) * time_period period_of_time period (15113229)
informing making_known (7212190) * change alter modify (126264) * time_period period_of_time period (15113229)
speech_act (7160883) * end terminate (2735418) * time_period period_of_time period (15113229)
speech_act (7160883) * end stop finish terminate cease (2609764) * calendar_month month (15209413)
speech_act (7160883) * end stop finish terminate cease (2609764) * time_period period_of_time period (15113229)
approval commendation (6686736) * grant give (2317094) * person individual someone somebody mortal soul (7846)
approval commendation (6686736) * give (2316868) * person individual someone somebody mortal soul (7846)
approval commendation (6686736) * give gift present (2200686) * person individual someone somebody mortal soul (7846)
approval commendation (6686736) * agree hold concur concord (805376) * person individual someone somebody mortal soul (7846)
power powerfulness (5190804) * grant give (2317094) * person individual someone somebody mortal soul (7846)
power powerfulness (5190804) * give (2316868) * person individual someone somebody mortal soul (7846)
unit social_unit (8189659) * accept take have (2236124) * message content subject_matter substance (6598915)

Slovarček izrazov iz področja analize besedil

- deep parsing: globinsko (skladenjsko) razčlenjevanje
- parser: (skladenjski) razčlenjevalnik
- probabilistic parser: verjetnostni (skladenjski) razčlenjevalnik
- POS tags, POS tagging: oblikoskladenjske oznake, oblikoskladenjsko označevanje
- chunks, chunking: besedne zveze ali besednozvezne enote, prepoznavanje besednih zvez ali besednozveznih enot
- parse tree: drevesnica[16]
- synset (v WordNetu): sinset[17]
- lemmatisation, lemmatiser: lematizacija, lematizator
- stop-words: neinformativne besede
- stop-word list: seznam blokiranih besed
- stemmer: krnilnik (stem: krn, stemming: krnenje)

[16] http://nl.ijs.si/sdt/
[17] Darja Fišer, http://lojze.lugos.si/~darja/slownet.html

Seznam slik

Seznam tabel

Literatura

[1] Reuters Corpus, Volume 1, English Language, 1996-08-20 to 1997-08-19. Dostopno na http://about.reuters.com/researchandstandards/corpus/. Objavljeno novembra 2000.

[2] L. Dali, D. Rusu, B. Fortuna, D. Mladenić, M. Grobelnik: *Question Answering Based on Semantic Graphs.* WWW 2009, april 2009, Madrid, Španija.

[3] F. Sebastiani: *Machine learning in automated text categorization*. ACM Computing Surveys 34(1), 1–47 (2002).

[4] Peter Turney: *Word Sense Disambiguation by Web Mining for Word Co-occurrence Probabilities* (2004).

[5] D. Mladenić, M. Grobelnik: *Feature selection on hierarchy of web documents.* Decision support systems, 2003, zvezek 35, str. 45-87.

[6] J. Leskovec, M. Grobelnik in N. Milic-Frayling: *Learning Sub-structures of Document Semantic Graphs for Document Summarization*. Workshop on Link Analysis and Group Detection (LinkKDD), KDD 2004 (Seattle, ZDA, 22. – 24. avgust 2004).

[7] Delia Rusu, Blaž Fortuna, Dunja Mladenić, Marko Grobelnik, Ruben Sipoš: *Document Visualization Based on Semantic Graphs.* IV09.

[8] L. Zhang, Q. Liu, J. Zhang, H. Wang, Y. Pan, Y. Yu: Semplore: *An ir approach to scalable hybrid query of semantic web data.* ISWC/ASWC '07, 2007, str. 652-665.

[9] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik in D. Mladenić: *Triplet Extraction from Sentences*. V zborniku 10. International Multiconference "Information Society - IS 2007", Ljubljana, Slovenija, 8. – 12. oktober 2007, str. 218 – 222.

[10] Ann Bies, Mark Ferguson, Karen Katz in Robert Mac-Intyre: *Bracketing guidelines for Treebank II style Penn Treebank project.* Technical report, University of Pennsylvania, 1995.

[11] M. Grobelnik, D. Mladenić: *Text Mining Recipes.* Springer, Heidelberg, 2009, http://www.textmining.net

[12] F. Cuna Ekmekcioglu, Alexander M. Robertson, Peter Willett: *Effectiveness of query expansion in ranked-output document retrieval systems.* Journal of Information Science, zvezek 18, št. 2, str. 139-147, 1992.

Izjava

Izjavljam, da sem diplomsko nalogo izdelal samostojno pod vodstvom mentorja doc. dr. Janeza Demšarja in somentorice doc. dr. Dunje Mladenić. Izkazano pomoč drugih svetovalcev sem v celoti navedel v zahvali.

Ljubljana, 5. junij 2009 Ruben Sipoš