UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gašper Mlakar

Pripomoček za pisanje na podlagi besedilne analize

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: prof. dr. Marko Robnik Šikonja

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00013/2012
Datum: 05.03.2012

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

Kandidat: **GAŠPER MLAKAR**

Naslov: **PRIPOMOČEK ZA PISANJE NA PODLAGI BESEDILNE ANALIZE**
A TEXTUAL ANALYTICS BASED WRITTING AID

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

S tehnikami tekstovne analize in tekstovnega rudarjenja lahko iz prostega besedila ugotovimo mnogo značilnosti, od preprostih, kot je frekvenca besed, do zapletenejših, kot je identifikacija entitet ali sloga pisanja. Orodja za tovrstno napredno analizo obsegajo tokenizacijo, lematizacijo, stavčno označevanje, gramatike, klasifikacijo in grupiranje teksta. Vedenje o tekstu, ki je na razpolago s pomočjo teh orodij, želimo usmeriti v pomoč pri pisanju besedil.
Izdelajte prototipni pripomoček, ki dani tekst najprej analizira in v njem identificira nekatere vrste slovničnih napak in slogovnih pomanjkljivosti, nato pa predlaga možne izboljšave. Primeri so prepogosta uporaba besed, mašila, besede, ki se večkrat zapored ponovijo v neposredni bližini, predolgi in prezapleteni stavki, napačna raba ločil, itd. Program naj temelji na analizi danega besedila, njegovi klasifikaciji in primerjavi z besedili iz korpusa.

Mentor:
prof. dr. Marko Robnik Šikonja

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Gašper Mlakar,

z vpisno številko 63010098,

sem avtor/-ica diplomskega dela z naslovom:

Pripomoček za pisanje na podlagi besedilne analize.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marka Robnik Šikonje

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 15. 9. 2012 Podpis avtorja:

ZAHVALA

Zahvaljujem se mentorju, prof. dr. Marku Robnik Šikonji, za strokovno pomoč in vodenje pri opravljanju diplomskega dela ter potrpežljivost.

Posebna zahvala gre tudi moji družini za spodbudo in podporo.

KAZALO

POVZETEK

Preučili smo najpogostejše tehnike za procesiranje naravnega jezika in tekstovno rudarjenje. Izdelali smo aplikacijo s pripomočki za pisanje na osnovi analize besedil. Vgradili smo orodja za klasifikacijo besedil na osnovi vsebine in sloga pisanja. Uporabili smo naivni Bayesov klasifikator in tudi njegovo izboljšavo AODE. Izdelali smo orodje za primerjanje besedil na osnovi kosinusne podobnosti vektorskih predstavitev besedil in ga uporabili za gručenje. Uporabili smo nekaj referenčnih korpusov besedil in izdelali enostavno orodje za njihovo pregledovanje in spreminjanje. Izdelali smo tudi pripomočke na osnovi slovarjev besed in sopomenk ter črkovalnik.

KLJUČNE BESEDE:

procesiranje naravnega jezika, tekstovno rudarjenje, klasifikacija, naivni Bayesov klasifikator, AODE, kosinusna podobnost, gručenje, korpus besedil

ABSTRACT

We examined the most common techniques for natural language processing and text mining. We built an application with writing aids on the basis of the text analysis. We embedded tools for the classification of texts based on the content and writing style. We used Naive Bayes classifier and its improved version AODE. We compared texts with cosine similarity of the vector text representation and used the similarity for clustering. We used a few reference text corpora and built a simple tool for their viewing and editing. We built a few utilities based on dictionaries and synonyms, e.g; a simple spelling checker.

KEY WORDS:

natural language processing, text mining, classification, Naive Bayes classifier, AODE, cosine similarity, clustering, text corpus

1. UVOD

Že tisočletja človek zapisuje svoje misli, ideje in spomine. S svojimi zapisi in besedili ohranja in preneša na naslednje rodove svoja znanja, spoznanja in kulturo družbe, v kateri živi. Skozi zgodovino se je spreminjal način zapisovanja besedil, jezik in medij, na katerega je zapisoval. Od kamenja, usnja in papirja smo prišli do različnih oblik elektronskega zapisa, ki je zaradi napredka v tehnologiji vse cenejši in hitrejši, s tem pa se je tudi močno povečala količina besedil. Iz te velike množice besedil je težko izbrati tiste, ki jih v danem trenutnku želimo oziroma potrebujemo, zato je nujno, da si izbiro olajšamo tako, da jih na nek način sortiramo. Pomagamo si tudi z izvlečki vsebine in glavnih značilnosti (jezik, zvrst, pogoste besede, stavčne zveze, itd). Ročno pregledovanje besedil je časovno potratno, zato v praksi velikokrat ni izvedljivo. Z razvojem računalništva smo dobili boljšo platformo za izdelavo avtomatiziranih orodij za analizo in klasifikacijo besedil. To so raznovrstni algoritmi, ki lahko razmeroma hitro izluščijo osnovne značilnosti velike količine podatkov - v našem primeru zbirke besedil oziroma korpusa.

Čeprav je postopek zapisovanja, prepisovanja in prenašanja pri elektronskih oblikah besedil enostaven, hiter in poceni, je sestavljanje novih besedil še vedno zapleten in dolgotrajen postopek. Za kvalitetno napisano besedilo, ki bi doseglo namen njegovega ustvarjalca in imelo za bralca neko vrednost, morajo biti izpolnjeni pogoji, kot so: upoštevanje slovnice in pravopisa, pravilna izbira sloga jezika za posamezno zvrst besedila in tudi sama oblika (velikost in oblika črk, členjenje besedila, ipd.). Tudi pri nekaterih od teh nalog, si lahko pomagamo z računalniškimi (programskimi) pripomočki - zelo razširjeno je avtomatsko preverjanje črkovanja besed, oblikovanja besedil pa si brez računalnikov skoraj ne moremo več predstavljati.

V diplomskem delu smo se spoznali z nekaterimi pristopi za računalniško obdelavo naravnega jezika in klasifikacijo besedil ter jih uporabili v praktični aplikaciji. Zastavili smo si cilj, da z nekaj programskimi pripomočki uporabniku olajšamo pisanje novih besedil.

V drugem poglavju so predstavljene glavne naloge, s katerimi se srečujemo pri analizi besedil ter algoritmi za njihovo reševanje. Nekoliko podrobneje smo si pogledali lematizacijo in označevanje besed ter iskanje in klasifikacijo besedil. Opisali smo delovanje algoritmov, ki te naloge rešujejo, in sicer naivni Bayesov klasifikator, algoritem AODE in rangirano iskanje besedil s pomočjo kosinusne razdalje ter gručenje. Na kratko smo predstavili tudi nekatere druge praktične primere uporabe teh algoritmov ter našteli nekaj slovenskih korpusnih in drugih besedilnih virov, ki so uporabni za računalniško analizo besedil.

V tretjem poglavju smo opisali aplikacijo za analizo besedil, ki je nastala v okviru te diplomske naloge. Predstavili smo delovanje glavnih orodij, uporabljene algoritme ter zgradbo podatkovnih struktur in programa. Na kratko smo opisali potek dela in predstavili določene probleme, ki so se pri tem pojavili. Naredili smo tudi nekaj izboljšav

osnovnih verzij algoritmov. Spoznali smo se z osnovnim Bayesovim algoritmom za klasifikacijo, ki smo ga uporabili pri dveh ločenih nalogah, ter njegovo izpeljanko AODE. Uporabili smo kosinusno podobnost vektorkih predstavitev besedil za primerjavo z besedili v korpusu in za gručenje.

V sklepu smo analizirali uspešnost uporabljenih algoritmov in predlagali nekaj izboljšav.

2. ANALIZA BESEDIL

Računalniška obdelava besedil in procesiranje naravnega jezika sta hitro razvijajoči se panogi, ki gresta v korak z razvojem novih računalniških in komunikacijskih tehnologij. Z nastajanjem obsežnih zbirk besedil (korpusov) v elektronskih obliki se razvijajo tudi vedno učinkovitejši algoritmi za tekstovno rudarjenje, ki večinoma spadajo na področje umetne inteligence in strojnega učenja. Pogledali smo si nekatere najbolj razširjene pristope.
Obstaja široka paleta izdelkov in pripomočkov za analizo besedil: od komercialnih do brezplačnih ter od programerskih pripomočkov (knjižnic) do polno funkcionalnih aplikacij in spletnih servisov. Našteli smo nekaj takih programskih rešitev.

2.1 BESEDNA OBDELAVA

Besedilo lahko analiziramo na besedni ravni. Pri tem vsako besedo obravnavamo ločeno od ostalih. Običajno gre za relativno preproste in hitre postopke preverjanja besedil, ki so ravno zaradi hitrosti zelo koristni za uporabnika in se veliko uporabljajo. To so npr. osnovne statistike, kot je število besed, število stavkov ipd. Malo naprednejši primer je preverjanje pravilnosti črkovanja z uporabo slovarja pravilnih besed. Naprednejši črkovalniki so zmožni tudi preverjanja pravilnosti zapisa glede na kontekst, zato upoštevajo tudi ožjo okolico besede (stavek). Črkovalniki so običajno vgrajeni v program za urejenje besedila. Naprednejši urejevalniki besedil vsebujejo še pripomočke za preverjanje slovnice, slovar sopomenk (tezaver) in preizkus berljivosti besedila po metodi Flesch-Kincaid. Ta deluje na osnovi povprečne dolžine stavka in povprečnega števila zlogov na besedo. Uporabljata se dve različici te metode, in sicer Flesch Reading Ease ter Flesch-Kincaid Grade Level [6]. Vsaka različica ima svojo enačbo.

Obrazec za metodo Flesch Reading Ease:

$$206{,}835 - 1{,}015 \cdot PDS - 84{,}6 \cdot PZD\,, \tag{2.1}$$

kjer je *PDS* povprečna dolžina stavka (število besed deljeno s številom stavkov) in *PZD* povprečno število zlogov na besedo (število zlogov deljeno s številom besed).
Rezultat je običajno med 1 in 100. Višji kot je rezultat, bolj razumljivo je besedilo.

Flesch-Kincaid Grade Level pa izračunamo:

$$0{,}39 \cdot PDS + 11{,}8 \cdot PZD - 15{,}59\,, \tag{2.2}$$

kjer je *PDS* povprečna dolžina stavka in *PZD* povprečno število zlogov na besedo. Rezultat pomeni razred v ameriškem šolskem sistemu, na nivoju katerega je obravnavano besedilo.

2.1.1 Primeri urejevalnikov besedil

Obstaja veliko različnih urejevalnikov besedil, ki nudijo tudi zelo različne nabore pripomočkov za pisanje:

1. Najbolj razširjen je **Microsoft Word**, ki v trenutni različici za Office 2010 ponuja napreden črkovalnik besed (črkovanje v kontekstu), preverjanje slovnice, slovar sopomenk (tezaver) in preizkus berljivosti besedila po obeh različicah metode Flesch-Kincaid. Vse funkcije podpirajo uporabo različnih jezikov, med drugim tudi slovenščine.

2. Drugi najbolj razširjen urejevalnik besedil je program **Writer** iz brezplačne pisarniške zbirke LibreOffice. V različici 3.5 med drugim ponuja: preverjanje črkovanja, dopolnjevanje besed, ki med pisanjem ponudi izbiro najpogosteje uporabljanih izrazov, in slovar sopomenk.

3. Vse bolj razširjena je tudi uporaba spletnih urejevalnikov za delo z datotekami v oblaku. Svojo spletno različico ima tudi Microsoft Word, brezplačna alternativa pa je Google Drive. Spletne različice so zaenkrat še manj zmogljive in ne ponujajo pripomočkov, razen preverjanja črkovanja.

2.2 PROCESIRANJE NARAVNEGA JEZIKA IN TEKSTOVNO RUDARJENJE

Procesiranje naravnega jezika (v angleščini »Natural Language Processing«, kratica NLP) je področje, ki se ukvarja z obdelavo besedil napisanih v človeškem (oziroma naravnem) jeziku. Področje spada na širše področje tehnologij za obdelavo naravnega jezika (angl. »Human Language Technology« oz. s kratico HLT), ki se nanaša na področje umetne inteligence. HLT sicer zajema še področja, kot so: razpoznavanje govora, strojno prevajanje, sinteza besedil in tekstovno rudarjenje [5].

2.2.1 Naloge pri analizi besedil

Tipične naloge s področja predprocesiranja naravnega jezika zajemajo: razčlenjevanje, krnjenje oz. korenjenje, lematizacijo, označevanje besednih vrst, normalizacijo sinonimov, razreševanje sklicev in dvoumnosti ipd. V principu gre vsako besedilo, ki ga želimo računalniško analizirati, najprej skozi postopek predprocesiranja. Cilj v NLP je obdelati nestrukturirano besedilo v naravnem jeziku tako, da je strojno razumljivo, kar je bistveno za uspešno jezikovno interakcijo med strojem in človekom [5]. Temu lahko sledijo postopki tekstovnega rudarjenja, ki zajemajo: iskanje, statistiko, klasifikacijo, gručenje, izdelavo povzetkov itd.

2.2.1.1 Krnjenje in lematizacija

Za nekatere statistične obdelave besedil so različne skladenjske oblike istih besed moteče, saj jih računalnik razpozna kot različne besede. Krnjenje (imenovano tudi korenjenje) je postopek, ki besedam odstrani končnice (ostane samo koren besede). Podoben postopek je tudi lematizacija, ki besedam poišče njihove osnovne oblike. Pri večini algoritmov strojnega učenja zadostuje, če se izvede lematizacija - v tem primeru krnjenja ne potrebujemo.

Lematizacijo se najlažje izvaja s pomočjo morfološkega slovarja, kjer imamo zapisane pare izpeljanka - osnovna oblika, vendar ta pristop ne deluje v vseh primerih oz. v vseh jezikih. Včasih je za določanje korena beseda potrebno ugotoviti kontekst te besede, kar zahteva predhodno izvedbo PoS označevanja (angl. »Part-of-Speech Tagging«). Razlika med lematizacijo in krnjenjem je v tem, da krnjenje ne zahteva pretvorbe v pravo osnovno obliko, operira zgolj nad samo besedo in ga je zato lažje implementirati. Lematizacija zahteva točno osnovno obliko, kar zahteva poznavanje konteksta, s tem pa potrebo po obdelavi širše okolice besede [5]. Ker sta nalogi lematizacije in označevanja besednih vrst (PoS) povezani, ju velikokrat opravljamo skupaj ali eno za drugo.

2.2.1.2 Označevanje besednih vrst (označevanje PoS)

Označevanje PoS (kratica iz angleškega izraza »Part-of-Speech«) je postopek določanja besednih vrst (npr. samostalnik, pridevnik, glagol, veznik itd.) in skladenjskih oblik (sklon, število, oseba, število ...) besed v stavku. Uporabljajo se algoritmi za strojno učenje, ki je v večini primerov nadzorovano, zato potrebujemo že rešene učne primere (označen učni korpus).

2.2.1.3 Iskanje besedil

Mnogokrat se pojavi problem, kako pri velikem številu besedil najti ustrezno, ki vsebuje željeno informacijo. V dobi interneta in velikih zbirk podatkov je to še posebno pereče. Na medmrežju in v aplikacijah se zato srečujemo z različnimi implementacijami iskalnikov. Iskanje je zelo pogosta naloga tekstovnega rudarjenja, zato mora biti hitra in učinkovita. Običajno se dejanski pomen besed ne upošteva, pač pa se ugotavlja stopnja ustreznosti posameznega besedila glede na iskalni niz, ki ga poda uporabnik. Za hitrejše iskanje se uporabi indeksiranje. To je postopek izdelave kataloga besedil, na katerem poteka iskanje. Iskanje pogosto sloni na uporabi vektorskih prostorov ali nevronskih mrež.

2.2.1.4 Klasifikacija

Ena od glavnih nalog s področja analize besedil in tekstovnega rudarjenja je zaznavanje vzorcev v besedilu, ki jih razberemo iz statistične obdelave, npr. frekvenca posameznih besed ali struktura stavkov. Na podlagi teh statistik se izvaja klasifikacija besedil (angleško »text classification«). Klasifikacija je postopek, ki besedilu dodeli oznako razreda oziroma kategorije. Klasifikacijo lahko posplošimo na vse vrste datotek (angl. »document classification«), kot so besedila, slike, glasba, itd. [4].

Naloga klasifikatorja je za objekt (besedilo), opisan z množico značilk (atributov), določiti, kateremu izmed možnih razredov pripada. Značilke so (neodvisne) zvezne ali diskretne spremenljivke, s katerimi opisujemo objekte, razred pa je (odvisna) diskretna spremenljivka, ki ji določimo vrednost (izberemo razred) glede na vrednosti atributov. Zato, da lahko klasifikator določi razred, mora imeti na nek način predstavljeno diskretno funkcijo, ki preslika prostor atributov v razred. Ta funkcija je lahko podana vnaprej ali pa je naučena iz podatkov (korpusa). Naloga učnega algoritma je torej iz množice vzorcev z znanimi pripadajočimi razredi zgraditi pravilo, ki ga lahko uporabimo pri klasifikaciji [3].

Klasifikatorji uporabljajo različne algoritme s področja strojnega učenja.

2.2.2 Predstavitev nekaterih pristopov k reševanju nalog analize besedil in algoritmov

Večina nalog iz NLP in tekstovnega rudarjenja spada med težke probleme s področja umetne inteligence. Moderni algoritmi NLP so zasnovani na principih strojnega učenja (angleško »machine learning«), ki se delijo na tri glavne skupine:

- *Nadzorovano učenje* (angl. »supervised learning«): algoritem zahteva, da zunanji nadzornik (npr. človek) določi razrede, ki jim pripadajo primeri iz učne množice (pri obdelavi besedil je to korpus besedil). Na osnovi teh primerov se algoritem nauči pravil določanja razredov. Algoritmi iz te skupine uporabljajo različne tehnike učenja: odločitvena drevesa, naivni Bayesov klasifikator (NB), Bayesove mreže, klasifikatorje ODE (iz angl. »One-Dependence Estimator«), k-ti najbližji sosed, umetne nevronske mreže in različne regresijske metode [3].

- *Nenadzorovano učenje* (angl. »unsupervised learning«): naloga algoritma je iz učnih primerov brez podanih razredov določiti razmeroma majhno število skupin vzorcev, ki so si med seboj najbolj podobni. Podobnost vzorcev je odvisna od predhodno podane mere podobnosti, ki je odločilnega pomena za razultat razvrščanja po skupinah (razredih). Obstajata dva pristopa: od spodaj navzgor (na začetku je vsak vzorec v svojem razredu, nato pa algoritem iterativno združuje najbolj podobne razrede) in od zgoraj navzdol (na začetku so vsi vzorci v enem razredu, ki ga algoritem iterativno razbija na manjše enote) [3]. Pri nenadzorovani klasifikaciji (*gručenju*) prednjačita hierarhična metoda in metoda k-sredin (angl. »k-means«).

- *Spodbujevano učenje* (angl. »reinforcement learning«): algoritem se uči z nagrajevanjem in kaznovanjem akcij. Učeni algoritem je lahko konzervativen in se zanaša na že naučeno, ali pa poskuša z novimi akcijami hitreje priti do rezultata. Strategijo učenja izbiramo na podlagi maksimalne vrednosti utežene vsote nagrad oziroma minimalne vrednosti utežene vsote kazni [3].

2.2.2.1 Naivni Bayesov klasifikator

Na podlagi zbirke učnih primerov (korpusa) želimo klasificirati besedilo. Vektor $x = (x_1, ..., x_n)$ predstavlja vrednosti n atributov danega besedila. Atributi so izbrane lastnosti besedil, ponavadi določene besede ali ločila. Algoritem deluje na podlagi enačbe, ki je izpeljana iz Bayesovega teorema.

Naj bo y spremenljivka, ki pripada množici k razredov $\{c_1, ..., c_k\}$. Verjetnost razreda y pri dani vrednosti x je tedaj enaka [2]:

$$P(y \mid x) = P(y) \cdot P(x \mid y), \tag{2.3}$$

kjer je $P(y)$ apriorna verjetnost razreda y in $P(x \mid y)$ verjetnost besedila x pri danem razredu y.

Ob predpostavki pogojne neodvisnosti atributov danega razreda (zaradi te predpostavke je algoritem označen kot naivni) velja [2]:

$$P(x \mid y) = \prod_{i=1}^{n} P(x_i \mid y), \tag{2.4}$$

Algoritem išče maksimalno verjetnost $P(y \mid x)$ po vseh možnih vrednostih y iz množice razredov [2]:

$$\arg\max_{y} \left(P(y) \cdot \prod_{i=1}^{n} P(x_i \mid y) \right), \tag{2.5}$$

kjer so verjetnosti $P(y)$ in $P(x_i \mid y)$ ocenjene na podlagi vzorčnih primerov (korpusa). Pri zajemu vzorcev lahko uporabimo tudi postopek glajenja (angl. »smoothing«), ki odpravi problem ničelne pojavitve določenih atributov pri posameznih razredih, ki bi verjetnost celotnega razreda ocenjenila z 0.

2.2.2.2 Vreča besed

Vreča besed (angl. »Bag of Words«) je priljubljena predstavitev besedil, ki jo pogosto uporabimo z naivnim Bayesovim klasifikatorjem. Kot atributi se določijo posamezne besede. Izbira besed je navadno prepuščen nadzorniku (človeku), lahko so to redke besede (ki imajo veliko razločevalno vrednost), lahko so pogoste ali pa kar vse možne besede, razen tistih, ki so na posebnem seznamu prepovedanih besed (angl. »stop-list«).

Algoritem prešteje, v koliko in katerih besedilih se posamezne besede pojavijo, in na podlagi funkcije verjetja (angl. »likelihood function«) določi vrednosti produkta iz enačbe (2.4). Funkcijo verjetja določa enačba [8, pog. 6]:

$$P(atribut_i \mid razred) = \frac{count(beseda_i, razred)}{count(razred)}, \tag{2.6}$$

kjer je *count(beseda*$_i$*, razred)* število besedil danega *razreda*, ki vsebujejo *besedo*$_i$ in *count(razred)* število vseh besedil tega razreda. Pri štetju navadno uporabimo še postopek glajenja, s katerim se izognemo ničelnim vrednostim števcev pojavitev.

2.2.2.3 Glajenje

Eno najpogostejših in enostavnejših postopkov glajenja je aditivno ali Laplacevo glajenje. Pri njem upoštevamo formulo za izračun verjetnosti dogodka [9]:

$$p = \frac{k + a}{n + a \cdot d}, \tag{2.7}$$

kjer je *n* število vseh dogodkov, *k* število pozitivnih dogodkov, *d* število atributov in *a* parameter glajenja, tipično $a = 1$.

V tej diplomski nalogi smo uporabili drug način glajenja, angleško imenovani »m-estimate smoothing«, ki deluje po enačbi:

$$p = \frac{k + p_a \cdot m}{n + d}, \tag{2.8}$$

kjer je p_a apriorna verjetnost dogodka in *m* je parameter glajenja, tipično $m = 2$.

2.2.2.4 SPODE in AODE

Skupina algoritmov ODE (iz angl. »One-Dependence Estimators«) [2] je izpeljanka naivnega Bayesovega algoritma. Pri slednjem je predpostavljeno, da so vsi atributi razreda med seboj pogojno neodvisni. Razlika pri ODE algoritmih je, da predpostavimo, da je vsak atribut odvisen od natanko enega drugega atributa.

Ob predpostavki, da so atributi x_1, ..., x_n med seboj neodvisni pri danem razredu y in atributu x_i, sledi [10]:

$$P(y, x_1, ...x_n) = P(y, x_i) \cdot \prod_{j=1}^{n} P(x_j \mid y, x_i) . \qquad (2.9)$$

V primeru, da so vsi atributi odvisni od istega (»super parent«) atributa, dobimo različico algoritma imenovano SPODE (iz angleškega izraza »Super Parent One-Dependence Estimator«).

Če povprečimo dobljene verjetnosti po enačbi (2.9), ko x_i zavzame vrednosti vseh atributov, dobimo algoritem AODE (»Average One-Dependence Estimators«) [2]. Algoritem izbere razred z najvišjim povprečjem verjetnosti razredov SPODE klasifikatorjev. Namesto računanja povprečij, lahko postopek poenostavimo in računamo le vsoto, saj izbira najbolj verjetnega razreda v tem primeru ostane enaka.

2.2.2.5 Rangirano iskanje in primerjava besedil na podlagi kosinusne razdalje vektorskih predstavitev besedil

Rangirano iskanje se je najprej uveljavilo v spletnem iskanju, kasneje pa se je preselilo tudi v druge sisteme za pridobivanje dokumentov. Glavna značilnost tega iskanja je, da kot rezultat dobimo množico dokumentov sortirano glede na pomembnost. Vhodnemu besedilu sistem odstrani nepotrebne besede iz seznama prepovedanih besed (»stop-words«), dokumente in povpraševanje pa lematizira in jih zapiše v vektorski predstavitvi: vsaka osnovna beseda je ena dimenzija, frekvenca te besede v besedilu pa razsežnost v tej dimenziji [1].

Podobnost med povpraševanjem in dokumentom lahko izračunamo kot razdaljo vektorskih predstavitev dokumenta in povpraševanja. Ob predpostavki, da so dimenzije vektorjev ortogonalne (predpostavljamo neodvisnost pojavitev posameznih besed glede na druge besede v besedilu, kar je poenostavitev, ki ni vedno povsem resnična), lahko razdaljo med vektorjema izračunamo kot kosinus kota med njima.

Za dokumenta oziroma vektorja A in B velja mera podobnosti [1]:

$$\cos(\Theta) = \frac{A \cdot B}{|A||B|}. \tag{2.10}$$

Vse besede v dokumentu niso enako pomembne, za ločevanje med dokumenti so pomembnejše tiste, ki se pojavljajo redkeje, saj vsebujejo več informacije. Vpeljemo pojem relativna redkost besed, *idf* (»inverse document frequency«). Redkost besede *b* izračunamo kot [1]:

$$idf_b = log(N / n_b), \tag{2.11}$$

kjer je N število dokumentov v zbirki, n_b pa število dokumentov z besedo b. S pomočjo mere *idf* lahko besede, ki predstavljajo dimenzije vektorjev, utežimo. Bolj redke besede dobijo večjo utež. Vektorje frekvenc besed $f_{b,d}$ zamenjamo z vektorji uteži besed, kjer posamezno utež $w_{b,d}$ besede b v besedilu d izračunamo po enačbi [1]:

$$w_{b,d} = f_{b,d} \cdot idf_b \,. \tag{2.12}$$

Na podlagi tega definiramo uteženo podobnost med povpraševanjem q in dokumentom d, ki upošteva uteži vseh besed v vektorju in izračuna kosinus kota med uteženim vektorjem povpraševanja in dokumenta [1]:

$$podobnost(q,d) = \frac{\sum_b w_{b,d} \cdot w_{b,q}}{\sqrt{\sum_b w_{b,d}^2} \cdot \sqrt{\sum_b w_{b,q}^2}}. \tag{2.13}$$

Ta algoritem lahko poleg iskanja s popraševanjem uporabimo tudi za primerjavo podobnosti dveh dokumentov ali za iskanje najbolj podobnih dokumentov v korpusu.

Ker mera podobnosti predstavlja kosinus kota Θ med vektorjema, so njene vrednosti na intervalu med 1 in 0, ko gre kot Θ od 0 proti π/2. Kot 0 pomeni najmanjšo razdaljo (največjo podobnost) med besediloma, takrat velja:

$$podobnost_{max} = cos(0) = 1. \tag{2.14}$$

2.2.2.6 Gručenje po metodi najkrajše razdalje

Gručenje (angl. »clustering«) ali rojenje je metoda nenadzorovanega strojnega učenja (ne potrebujemo učnega vzorca). Obstaja več vrst gručenja, dve najpomembnejši vrsti sta hierarhično (angl. »hierarchical«) in metoda k-sredin (angl. »k-means«). Metoda najkrajše razdalje (angl. »shortest distance«), imenovana tudi metoda najbližjega soseda (angl. »nearest neighbour«) ali enojna povezanost (angl. »single-linkage«) spada med hierarhična gručenja [31]. Ideja je preprosta: na osnovi neke mere podobnosti (oziroma bližine) postopoma združujemo najbližje sosede, dokler ne pridemo do ene same gruče oziroma do neke meje podobnosti. Za mero razdalje pri besedilih lahko uporabimo kosinusno razdaljo vektorjev, ki smo jo opisali v prejšnjem razdelku.

2.2.3 Nekaj programskih pripomočkov in orodij

Na trgu in prosto dostopnih na spletu obstaja že mnogo različnih izdelkov iz tega področja. Ker je nemogoče predstaviti vse, smo se omejili na glavne odprtokodne izdelke in take, ki znajo delati s slovenskim jezikom:

1. **NLTK** - Natural Language Toolkit [11] je vodilna platforma za izgradnjo programov, ki obdelujejo podatke v obliki človeškega jezika, na osnovi programskega jezika Python. Vsebuje več kot 50 korpusov in mnogo knjižnic z algoritmi za procesiranje naravnega jezika, npr. za klasifikacijo, tokenizacijo, krnjenje, označevanje in analizo jezikovne zgradbe besedila. NLTK je odprtokoden projekt, ki deluje na vseh treh glavnih operacijskih sistemih namiznih računalnikov: Windows, Linux in Mac OS X. Za lažji začetek dela je na voljo tudi brezplačna spletna različica knjige [8]. Na žalost zaenkrat NLTK še ni polno funkcionalen za slovenski jezik.

2. **Apache OpenNLP** [12] programska knjižnica je zbirka pripomočkov v javi. Ponuja algoritme za vse osnovne operace pri analizi besedil: tokenizacijo, stavčno segmentacijo, označevanje PoS in druge, ki jih uporabljamo za izgradnjo kompleksnejših aplikacij. Gre za odprtokodni projekt. Na voljo je tudi vsa dokumentacija in navodila za lažjo uporabo (wiki).

3. **Ogrodje GATE (**GATE je angl. kratica za »General Architecture for Text Engienering«) je generični programski paket za procesiranje besedil. Danes je GATE najbolj razširjen programski paket na tem področju. Je prosto dostopen pod licenco LGPL in omogoča neomejeno komercialno rabo. V celoti je implementiran v programskem jeziku java in deluje na vseh platformah, ki omogočajo vsaj Java 5 [5, str. 21]. V osnovnem paketu dobimo nekaj procesnih virov, ki so splošno uporabni v aplikacijah za procesiranje naravnega jezika.

4. **Oblikoslovni označevalnik za slovenski jezik** je na razpolago kot neplačljiv spletni servis [13] in kot skupek programov v okolju .NET, ki jih lahko prosto naložimo s spletne strani. Oblikoslovni označevalnik je orodje za označevanje PoS (»Part-of-Speech«) in lematizacijo neformatiranih besedil. Rezultat označevanja zapiše v izhodno datoteko v obliki XML TEI. Text Encoding Initiative (TEI) [14] je združenje, ki razvija in ohranja standarde za predstavitve besedil v digitalni obliki.

5. **Projekt JOS (**jezikoslovno označevanje slovenskega jezika) je razvil označene korpuse slovenskega jezika in pridružene vire, namenjene spodbujanju razvoja jezikovnih tehnologij za slovenski jezik [15]. Na voljo sta dva spletna servisa:

    - konkordančni in frekvenčni seznami korpusov JOS [16],
    - označevalnik besedila JOS ToLaTe text analyser [17]: spletni servis vrne vnešeno besedilo, razdeljeno na stavke, pri čemer je vsaka beseda označena s svojo oblikoskladenjsko oznako JOS in lemo.

2.3 KORPUSI, SLOVARJI IN BESEDILNI VIRI

Besedilni korpusi so obsežne zbirke besedil v naravnem jeziku, zajete v določenem obdobju iz množičnih medijev (časopisi, revije), knjižne produkcije, stripov, interneta, reklamnih besedil, navodil priloženih izdelkom široke potrošnje, prepisov parlamentarnih razprav in drugih virov. Shranjene so v strukturirani oblik na digitalnih medijih in s pomočjo jezikovnih tehnologij pogosto opremljene z označbami. Za sodobno jezikoslovje so podatkovna infrastruktura, podobna kot so za sodobne družboslovne znanosti različne baze družboslovnih podatkov [22].

2.3.1 Referenčni korpusi

Posebna vrsta korpusov so referenčni korpusi. So najobsežnejši in metodološko zagotavljajo reprezentativen izbor besedil iz določene jezikovni skupnosti ali naroda, zato služijo temeljnim raziskavam na področju slovnice in slovarjev [22]. Referenčni korpusi so eden glavnih virov informacij za algoritme strojnega učenja pri računalniški analizi besedil. Korpusi, ki jih želimo uporabiti v računalniški obdelavi, morajo biti ustrezno strukturirani in oblikovani v skladu z določenimi pravili. V zadnjem času se najpogosteje uporabljajo XML sheme po specifikacijah TEI P5 [14] ter oblikoskladenjske specifikacije MULTEXT-East V4 [23].

2.3.2 Primeri slovenskih korpusov

Nekateri korpusi:

1. Korpus **FidaPLUS** [18] je referenčni korpus slovenskega jezika. Vsebuje besedila najrazličnejših zvrsti iz večine slovenskih dnevnih časopisov, mnogih revij ter knjižnih publikacij različnih založb. Poleg tega korpus zajema tudi besedila z interneta, prepise parlamentarnih govorov, t. i. besedilni drobiž (reklamna besedila, plačilni listki, računi) itd. Skupno število besed presega 620 milijonov.

2. V okviru projekta **JOS** [15] sta nastala dva jezikoslovno označena korpusa *jos100k* (vsebuje sto tisoč besed) in *jos1M* (milijon besed), ki vsebujta vzorčne odstavke iz korpusa FidaPLUS. Obe sta v zapisu XML in temeljita na vzorčnih shemah TEI P5 [14].

3. **Nova beseda**: Korpus je nastal je v okviru Inštituta za slovenski jezik Frana Ramovša in obsega 318 milijonov besed. Korpus je namenjen predvsem potrebam Inštituta, obenem pa tudi vsem drugim, ki se ukvarjajo z izobraževanjem in raziskovanjem slovenskega jezika. Na voljo je spletni servis [24], ki deluje kot konkordančnik in iskalnik besed.

4. V okviru projekta **"Sporazumevanje v slovenskem jeziku"** [25] je v nastajanju referenčni korpus v obsegu več 100 milijonov besed in leksikalna baza s slovničnim analizatorjem.

2.3.3 Drugi jezikovni viri

Kot dodatni viri (primarni vir so korpusi) podatkov za analizo besedil se lahko uporabljajo tudi posamezna neobdelana besedila v elektronski obliki, čeprav niso zbrana v korpusih. Iz takih besedil je mnogo težje izluščiti željeno informacijo in običajno je potrebna predobdelava. Prav tako pri analizi besedil uporabljamo tudi slovarje za avtomatsko preverjanje črkovanja, tezaver za iskanja sinonimov in druge urejene zbirke besed.

2.3.2.1 Primeri virov

Naštejmo nekaj slovenskih jezikovnih virov:

1. Slovar slovenskega knjižnega jezika na internetu: spletni servis na naslovu http://bos.zrc-sazu.si/sskj.html.

2. Odprti Tezaver [7]: odprtokodni tezaver za slovenski jezik, 15 tisoč besed, preko 6 tisoč množic sopomenk.

3. sloWNet: v okviru doktorske disertacije [19] je nastala slovenska različica leksikalnih zbirk tipa WordNet [20] ter aplikacije za njihovo uporabo. Razvita so bila tudi orodja za avtomatizirano gradnjo leksikonov. Aplikacija za uporabo sloWNeta je dostopna na spletni strani [21].

4. Odprti kop [http://www.rtvslo.si/odprtikop/o_odprtem_kopu] je sistem, ki na podlagi podnapisov in video posnetkov oddaj samodejno generira spletne strani teh oddaj, ki jih je zato mogoče najti s spletnimi iskalniki.

5. Wikivir: odprto skladišče izvornih besedil v slovenščini, ki jih lahko vsakdo ureja [26]. Vsebuje knjige (leposlovje, enciklopedije, leksikone, slovarje, …), pisma in druga posamezna besedila.

6. Zbirka slovenskih leposlovnih besedil [27].

7. Digitalna knjižnica Slovenije [28].

8. Iskalniki slovenskih besedil, ki se nahajajo v korpusih FidaPLUS in Nove beseda ter drugih virih (Wikivir, Digitalna knjižnica).

9. Slovensko leposlovje na spletu: http://641.gvs.arnes.si/slovlit/index.php.

10. Slovensko leposlovje na spletu: http://slovenskaliteratura.ff.uni-lj.si/sl.html.

3. PRIPOMOČKI ZA PISANJE Z UPORABO METOD ZA ANALIZO BESEDIL

Program s pripomočki za pisanje smo si zamislili kot razširitev osnovne aplikacije za urejanje besedil, ki smo ji dodali nekaj naprednih funkcij. Stilno oblikovanje črk in besedila ni pomembno za analizo besedil, zato ga v okviru tega diplomskega dela nismo implementirali. Odločili smo se za programiranje v okolju Microsoft Visual Studio 2010 in za programski jezik C#. Program omogoča delo z navadnimi besedilnimi datotekami ter nalaganje in shranjevanje XML datotek po shemi TEI [14], ki vsebujejo oblikoslovno označeno in lematizirano besedilo. Za izvajanje postopkov lematizacije in označevanja smo uporabili že obstoječe prosto dostopne programe, ki smo jih opisali v poglavju 2.2.3 [str. 13, točka 4].

Ko je besedilo označeno (»Part-of-Speech Tagging«) in lematizirano, ga lahko analiziramo - nad njim izvajamo postopke procesiranja naravnega jezika in tekstovnega rudarjenja. Algoritmi strojnega učenja potrebujejo učno množico primerov besedil. V ta namen smo uporabljali predvsem korpusa JOS, ki sta opisana v poglavju 2.3.2. Naredili smo tudi preprosto orodje za delo s korpusi, ki omogoča pregled, izpis, dodajanje in odstranjevanje besedil obstoječega korpusa ter kreiranje novega korpusa. Vsa besedila, ki jih dodajamo v korpus, morajo biti predhodno označena in lematizirana. Novo kreirani korpusi posnemajo obliko korpusov JOS, zato so shranjeni v TEI-XML datotekah. Program lahko uporablja več korpusov naenkrat.

Kot glavno nalogo smo si zadali primerjanje danega besedila z besedili v korpusih in iskanje podobnih besedil glede na slog in vsebino. V ta namen smo uporabili Bayesov algoritem za klasifikacijo v dveh različicah, izboljšani algoritem AODE ter postopek primerjanja besedil z računanjem kosinusne razdalje vektorjev podobnosti [poglavje 2.2.2]. V pomoč pri pisanju pa smo dodali še iskalec in števec besed, ki zna iskati tudi po oznakah in lemah, preprost črkovalnik in orodje, ki napisano besedilo slogovno preveri, ter ponudi boljše rešitve. V okviru tega preverjanja smo implementirali algoritem za iskanje (pre)pogosto uporabljanih besed in dodali slovar sopomenk (tezaver) [poglavje 2.3.2.1], ki ponudi alternativne izraze. Programu smo dodali še navodila za uporabo v obliki HTML.

3.1 KLASIFIKACIJA BESEDIL GLEDE NA SLOG

Pomembna prvina sloga pisanja so vrste in pogostost uporabljenih besed in ločil. Zato smo se odločili uporabiti naivni Bayesov klasifikator [poglavje 2.2.2.1], ki bo deloval na podlagi pogostosti izbranih elementov besedila - atributov. Atribute klasifikatorja določi uporabnik, in sicer so to lahko oblikoslovne oznake besed, besedne vrste, leme besed ali ločila. Razredi besedil se določijo avtomatično, vendar na podlagi kategorije, ki jo izbere uporabnik. Kategorije predstavljajo različna opisna polja besedil v korpusu, kot so na primer: avtor, založnik, leto, zvrst itd. Izgradnja klasifikatorja je relativno dolg postopek, zato ga je možno shraniti v XML dokument in znova uporabiti pri klasifikacijah drugih besedil.

Pri izdelavi naivnega Bayesovega klasifikatorja [enačba 2.4], smo naleteli na problem, kako določiti pogojno verjetnost $P(x_i \mid y)$ atributa x_i pri danem razredu y. Enostavna formula (2.6) pri vreči besed v tem primeru ne deluje, saj atributi niso binarni (beseda se pojavi v besedilu ali se ne pojavi), pač pa je pomembno, kolikokrat se pojavi glede na dolžino besedila oziroma *relativno pogostost pojavljanja* v posameznem besedilu. Kot rešitev smo uporabili funkcijo gostote verjetnosti neke statistične porazdelitve, s katero določimo verjetnost $P(x_i \mid y)$. Izbrali smo normalno (Gaussovo) porazdelitev s funkcijo gostote verjetnosti [29]:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right), \tag{3.1}$$

kjer je μ pričakovana verjetnost oz. povprečje in σ^2 varianca. Ti dve vrednosti se izračunata za vsak atribut v vsakem razredu. Namesto običajne formule za varianco, ki lahko prinese slabe rezultate pri numeričnem računanju z majhnimi števili (pride do relativno velike napake), smo uporabili »on-line« algoritem enega obhoda [30].

3.1.1 PODATKOVNE STRUKTURE IN GLAVNE FUNKCIJE

Kot osnovo dinamičnih podatkovnih struktur smo uporabili generični seznam tipa List<>, ki smo jih združevali v višjih strukturah (razredih), ki smo jim dodali tudi metode za osnovne operacije nad hranjenimi podatki.

- Razred *oznake* poskrbi za nalaganje in hranjenje oblikoslovnih oznak besed ter oznak zvrsti besedil.

- Razred *statistika* vsebuje seznam besedil (podatki o besedilih: naslov, avtor, zvrst, leto itd.) in sezname lem besed, oblikoslovnih oznak besed in ločil ter njihove pripadajoče števce. Ko so besedila prebrana, jih lahko razvrstimo v skupine na podlagi kategorije razvrščanja. Te skupine besedil so osnova za določanje razredov pri gradnji klasifikatorja. Leme, oznake in ločila v seznamih pa tvorijo atribute. Razred vsebuje metode za štetje besed, ki so v seznamih, v danem besedilu oz. korpusu. Dve najpomembnejši metodi razreda *statistika* sta *ZgradiKlasifikator()*, ki iz statistike prebranih besed izračuna vrednosti atributov klasifikatorja in vrne objekt *klasifikator* ter metoda *Klasificiraj()*, ki na podlagi danega objekta *klasifikator* in statistike besed danega besedila izračuna verjetnosti pripadajočih razredov.

- Poglavitna vloga razreda *klasifikator* je hranjenje podatkov o *razredih* (oznaka razreda, število besedil) in *atributih*. Objekti tipa *razred* in *atribut* so hranjeni v dveh seznamih tipa List<>. Podatkovna struktura tipa *atribut* vsebuje: *oznako*, *seznam varianc* in *seznam povprečij*. Seznama sta dolžine, ki je enaka številu vseh objektov tipa *razred* (vsak atribut hrani tabelo podatkov o povprečju in varianci za vsak *razred* posebej). Klasifikator lahko tudi shranimo v datoteko s klicem metode *ZapisiKlasifikator()* in naložimo iz datoteke s klicem *PreberiKlasifikator()*.

3.1.2 POSTOPEK IZGRADNJE KLASIFIKATORJA

Klasifikator zgradimo takole:

1.) Izbira korpusov: uporabnik iz seznama korpusov izbere tiste, ki bodo na voljo učnemu algoritmu, ki bo gradil klasifikator.

2.) Izbira atributov: uporabnik določi leme, vrste besed in ločila, ki se bodo upoštevala kot atributi besedil. V pomoč pri izbiri je statistika korpusa (preštevanje besed in ločil), ki jo opravi objekt *statistika*. Poteka v več korakih, za vsak tip atributov posebej.

3.) Določitev razredov: uporabnik izbere kategorijo in določi, katera besedila iz korpusa se upoštevajo (privzeto se upoštevajo vsa besedila). Metoda objekta *statistika* določi skupine besedil (kasneje so to razredi), na osnovi izbrane kategorije. Če je kategorija avtor, nastane toliko različnih skupin, kolikor je različnih avtorjev, v vsako skupino pa se dodajo reference na besedila tega avtorja. Pri tem smo naleteli na manjšo težavo: veliko število besedil v korpusih JOS nima znanega avtorja (na primer pri časopisnih člankih). Uvedli smo kategorijo avtor-ali-založnik, pri kateri ob neobstoju avtorja upoštevamo ime založnika.

4.) Izračun vrednosti statističnih parametrov atributov: iz števcev pojavitev in dolžine posameznih besedil se izračuna pogostost atributa v vsakem izmed besedil:

$$pogostost_a = števec(a, b) / dolžina(b), \quad (3.2)$$

kjer je *števec(a, b)* funkcija, ki prešteje, kolikokrat se atribut *a* pojavi v besedilu *b*; *dolžina(b)* je število vseh besed besedila b.

Pri atributih tipa lema, je možna tudi drugačna mera pogostosti, kjer za dolžino besedila ne upoštevamo vseh besed, temveč le besede iste besedne vrste (primer: pri atributu, ki predstavlja veznik »in«, se bodo štele le besede, ki so vezniki).

Če se atribut ne pojavi v nekem razredu, bomo dobili varianco enako nič, kar bo prineslo množenja z 0 v enačbi (2.4). Zaradi tega problema smo vpeljali glajenje »m-estimate« [poglavje 2.2.2.3]. Popravljena enačba za izračun pogostosti je:

$$pogostost_a = (števec(a, b) + p_a * m) / (dolžina(b) + N_a), \quad (3.3)$$

kjer je p_a apriorna verjetnost atributa *a*, *m* utež glajenja in N_a število vseh atributov tega tipa (ločil ne upoštevamo pri besedah). Vrednost uteži *m* smo nastavili na 2.

Iz pogostosti atributa v vseh besedilih nekega razreda se izračunajo statistični parametri (povprečje in varianca) atributa za ta razred. Uporabi se »on-line« algoritem [30].

5.) Ko so vse vrednosti atributov izračunane, jih skupaj z oznakami in opisi razredov shranimo v datoteko oblike XML. Primer takega zapisa:

```
<klasifikator kategorija="ZVRST" stevilo_besedil="688" opis="">
 <razred oznaka="Ft.P.P.O.P.C.D" opis="" stevilo_besedil="99" stevilo_besed="37784"
stevilo_locil="6359"/>
 <razred oznaka="Ft.Z.N.N" opis="" stevilo_besedil="180" stevilo_besed="67602"
stevilo_locil="11841"/>
 <razred oznaka="Ft.L.D" opis="" stevilo_besedil="191" stevilo_besed="82218"
stevilo_locil="14812"/>
 <razred oznaka="Ft.P.P.O.P.R.M" opis="" stevilo_besedil="29" stevilo_besed="9606"
stevilo_locil="1645"/>
 <razred oznaka="Ft.Z.N.S.N" opis="" stevilo_besedil="20" stevilo_besed="6522"
stevilo_locil="1128"/>
...
 <atribut oznaka="pri" tip="1" opis="_lema_@D**" nacin="1">
  <vrednost st_razreda="0" pogostost="0,0178127094748778" varianca="0,000307148902021787"
/>
  <vrednost st_razreda="1" pogostost="0,0186420088429356" varianca="0,000449670900457131"
/>
  <vrednost st_razreda="2" pogostost="0,0214122788562937" varianca="0,000556889898704126"
/>
  <vrednost st_razreda="3" pogostost="0,036080950796518" varianca="0,00118228718855665"
/>
  <vrednost st_razreda="4" pogostost="0,0342666001810276" varianca="0,00107304737788602"
/>
 ...
</atribut>
<atribut>
 ...
</atribut>
</klasifikator>
```

Za lažje pregledovanje in urejanje vrednosti atributov pri klasifikatorjih smo v okviru uporabniškega vmesnika za izgradnjo klasifikatorjev in klasificiranje izdelali tudi pripomoček za te namene.

3.1.3 POSTOPEK KLASIFIKACIJE

Klasifikacijo besedil v neko kategorijo izvedemo po korakih:

1.) Uporabnik napiše ali naloži besedilo.

2.) Če besedilo ni označeno in lematizirano, se uporabijo vgrajeni programi za ta namen.

3.) Uporabnik določi datoteke s klasifikatorji, ki se bodo uporabili.

4.) Ustvari se nov objekt *klasifikator* in vanj se naložijo vrednosti iz dane XML datoteke s trenutnim klasifikatorjem.

5.) Objekt *statistika* prešteje pojavitve atributov v klasificiranem besedilu in izračuna pogostosti.

6.) Na osnovi izraza (3.1) za funkcijo gostote verjetnosti se izračunajo pogojne verjetnosti $P(x_i|y)$ atributa i pri razredu y. Tu smo naleteli na manjšo težavo: ker so nekatere vrednosti blizu nič, se pri numeričnem računanju (množenje $\prod P(x_i|y)$, za vsak atribut i) lahko zgodi, da vrednost pade pod stopnjo natančnosti spremenljivke tipa double. Enačbo logaritmiramo in namesto zmnožka dobimo seštevek $\sum logP(x_i|y)$. Pri logaritmiranju se poenostavi tudi enačba funkcije gostote verjetnosti. Predpostavili smo, da je apriorna verjetnost razreda $P(y)$ enaka za vse razrede, zato je nismo upoštevali pri enačbi (2.3) oziroma (2.5), saj se vrstni red razredov glede na verjetnost ne spremeni.

7.) Izračunane verjetnosti razredov uredimo po vrsti, normaliziramo in izpišemo rezultat.

Postopek je podrobneje opisan v [dodatku A], ki je priložen diplomskemu delu.

3.2 KLASIFIKACIJA BESEDIL GLEDE NA VSEBINO

Želimo poiskati skupino besedil v korpusu, ki je kar najbolj podobno danemu besedilu, glede na besede, ki se v njem pojavljajo. Za razliko od prejšnjega razdelka, tukaj ne štejemo pojavitev besed v besedilu in dolžine besedila, ampak je važno le, če se beseda pojavi ali ne. Uporabili smo dva algoritma za klasifikacijo (uporabnik ima izbiro). Prvi je naivni Bayesov algoritem z vrečo besed [poglavji 2.2.2.1 in 2.2.2.2], drugi pa njegova izpeljanka »Average One-Dependence Estimator« [pog. 2.2.2.4]. Atribute klasifikatorja in kategorijo razredov določi uporabnik. Pri zajemu podatkov smo uporabili tehniko glajenja »m-estimate smoothing« [enačba 2.8].

3.2.1 PODATKOVNE STRUKTURE IN GLAVNE FUNKCIJE

Celoten postopek klasifikacije se opravi v razredu *klasifikacija_v2*, ki je izpeljan iz razreda osnovnega okna System.Windows.Forms.Form, zato opravlja tudi funkcijo uporabniškega vmesnika za to orodje.

Program določi vse različne leme vhodnega besedila (oblikoslovno označeno, lematizirano in v obliki TEI-XML) in jih prešteje. Te podatke shrani v seznamu besed *vhodno_besedilo*:

```
class beseda
{
        public string lema;
        public int stevec;
}
List<beseda> vhodno_besedilo;
```

Razred *besedilo* oziroma seznam *besedila* hrani podatke za kasnejše razvrščanje po kategorijah:

```
class besedilo
{
        string oznaka;
        string avtor;
        string tip;     //prenosnik, medij
        string zvrst;   //zvrst
}
List<besedilo> besedila;
```

Prepovedane besede (angl. »stop words«) prebere iz datoteke in shrani v seznam *stop_seznam*.

Podatkovna struktura vreče besed oziroma seznam atributov:

```
class atribut
{
        string lema;
        List<int> stevci;   //števci pojavitev po besedilih
        int st_besedil;     //število besedil, kjer se pojavi ta beseda
        atribut(string lem, int st_vseh_besedil) { /* inic …*/  }
}
List<atribut> atributi;
```

Vse podatke posameznega razreda hrani razred *razred*:

```
class razred
{
        string oznaka;
        List<int> idx_besedil;   //katera besedila pripadajo temu razredu
        List<double> stevci_atributov;
        double verjetnost_razreda;

        razred(string ozn, int st_atr) { /* inic …*/  }
}
```

Ob izvajanju programa so kreirani seznami razredov.

Glavne funkcije:

- *preberi_besedilo*(): prebere XML datoteko vhodnega besedila in napolni seznam besed *vhodno_besedilo*;
- *doloci_besedila*(): prebere glavo korpusa in napolni seznam *besedila*;
- *doloci_razrede*(): poišče vse različne oznake besedil pri dani kategoriji in ustvari seznam teh razredov, ki jim nato določi pripadajoča besedila;
- *preberi_besedila*(): prebere korpus in prešteje pojavitve atributov za posamezna besedila (števce shrani v seznam *atributi*);
- *Bayes_SPODE*(): izračuna verjetnosti razredov ob predpostavki, da so vsi atributi pogojno neodvisni (naivni Bayes - NB); ob dodatnem parametru, ki podaja indeks atributa, ki je trenutni »super parent« (SP), pa to postane algoritem SPODE.

3.2.2 POSTOPEK KLASIFIKACIJE Z ALGORITMOM NB

Z naivnim Bayesovim klasifikatorjem lahko besedila klasificiramo takole:

1.) Uporabnik napiše ali naloži besedilo in v meniju uporabniškega vmesnika izbere klasifikacijo po vsebini. Če besedilo ni označeno in lematizirano, se uporabijo vgrajeni programi za ta namen.

2.) Uporabnik nastavi parametre delovanja:

- določi datoteke s korpusi, ki se bodo uporabili;
- izbere kategorijo razredov;
- izbere algoritem (NB - naivni Bayes);
- utež glajenja: prednastavljena vrednost je 2;
- določi, ali naj se upošteva apriorna verjetnost razredov *P(y)* v enačbi (2.5): če predpostavljamo, da je vzorec besedil v uporabljenih korpusih podoben realnemu, je smiselno upoštevati apriorno verjetnost razredov, ki je izračunana na podlagi korpusa (razred z več besedili ima večjo apriorno verjetnost), drugače pa je bolje predpostaviti, da so vsa besedila enako verjetna, zato ta člen v enačbi zanemarimo;
- seznam atributov: program ponudi vse možne leme, ki se pojavijo v vhodnem besedilu (uporaba funkcije *preberi_besedilo*); upoštevajo se le tiste leme, ki pripadajo dovoljenim besednim vrstam (besedne vrste nastavi uporabnik - privzeto so to samostalniki, pridevniki in glagoli) in ki hkrati niso na seznamu prepovedanih besed (seznam je možno izključiti); leme so sortirane po padajoči frekvenci glede na vhodno besedilo.

3.) Ko je končal z izbiro, uporabnik potrdi izbrane parametre. Program začne postopek klasifikacije.

4.) Priprava podatkov:

- funkcija *določi_besedila*() prebere glave izbranih korpusov in zgradi seznam besedil;
- funkcija *preberi_besedila*() prešteje vse pojavitve izbranih besed (atributov) za vsako besedilo;

- program iz seznama *atributi* odstrani atribute, ki se v izbranih korpusih pojavijo manjkrat kot določa *meja_pojavitve*;

- program preveri, če imamo dovolj atributov in razredov za klasifikacijo, sicer javi napako in zaključi.

5.) Kliče se funkcija *Bayes_SPODE(-1)*, ki najprej določi razrede s klicem *doloci_razrede*(-1). Ob podani vrednosti parametra -1, ki ne kaže na noben atribut kot »super-parent«, deluje kot navaden naivni Bayesov klasifikator. Funkcija vrne vektor verjetnosti dolžine N, kjer je N število razredov, njegove vrednosti pa predstavljajo verjetnosti razredov pri istem indeksu.

6.) Pri normalizaciji rezultatov verjetnosti se vsaka verjetnost deli z vsoto vseh verjetnosti. Izvede se sortiranje rezultatov in njim pripadajočih razredov po padajočem vrstnem redu. Tako dobimo na prvem mestu razred z najvišjo verjetnostjo, kar predstavlja rezultat klasifikacije.

7.) Rezultati (verjetnosti razredov v padajočem vrstnem redu) se izpišejo v prikazno okno. Dobljeni rezultati se množijo s 100, da pomenijo odstotke.

3.2.3 POSTOPEK KLASIFIKACIJE Z ALGORITMOM AODE

Z algoritmom AODE se postopek v točkah 1 - 4 izvede enako kot v primeru klasifikacije z NB (razdelek 3.2.3). Nekoliko drugačno je nadaljevanje:

5a.) Za vsak atribut *i* iz seznama *atributi* se kliče funkcija *Bayes_SPODE(i)*: v tem primeru razredom določimo iste oznake, upoštevamo pa le besedila, ki pri danem razredu vsebuje tudi *i*-ti atribut. Ustvari se dvodimenzionalna tabela verjetnosti: za vse razrede in hkrati za vsak atribut kot *predhodnik* (angl. »super-parent«) pri vsakem razredu. Z drugimi besedami: dobimo *i* klasifikatorjev tipa SPODE.

5b.) Povprečenje vrednosti SPODE klasifikatorjev po vseh različnih atributih. Povprečenje smo poenostavili in izpustili deljenje s številom vseh atributov, ker je enako za vse razrede.

Nadaljevanje poteka programa (točki 6 in 7) je enako kot v razdelku 3.2.3.

3.3 PRIMERJAVA BESEDIL

Besedila primerjamo na osnovi kosinusne podobnosti vektorskih predstavitev besedil [razdelek 2.2.2.5]. Uporabili smo dve metodi: primerjava danega besedila z vsemi v korpusu in polavtomatsko grajenje gruče podobnih besedil (gručenje) [razdelek 2.2.2.6]. V obeh primerih uporabnik izbere korpuse za primerjavo (oziroma posamezna besedila v njih). Ker lahko pri veliki količini besedil nastane problem, če jih primerjamo glede na vse možne besede, smo uporabniku dali možnost, da določi oblikoslovne oznake upoštevanih besed. Ostale besede se ignorirajo. Prav tako se ignorirajo tudi leme besed, ki so na stop-seznamu. Po končanem postopku določitve besed program zgradi vektorske predstavitve vseh izbranih besedil.

Najprej se za izbrana besedila zgradijo vektorji frekvenc besed, nato se izračunajo koeficienti relativne redkosti besed *IDF* (angl. »inverse document frequency«) za vse besede [enačba 2.11]. Na podlagi enačbe (2.12) se iz vektorjev frekvenc in faktorjev IDF izračunajo utežni vektorji besedil.

3.3.1 UPOŠTEVANJE LOKALNE POGOSTOSTI BESED

Želeli smo, da se pri primerjavi besedil upošteva tudi, ali se neka beseda pogosto pojavlja v določeni okolici. Izbrali smo mero pogostosti, ki je nasprotna relativni redkosti (*IDF*). Lokalno pogostost *LP* smo definirali kot:

$$LP = e^{\frac{n_b}{K}}, \tag{3.4}$$

kjer je n_b število besed, ki se pojavijo v okolici, in K velikost okolice. Vsako pojavljanje besede v besedilu ima svojo lokalno pogostost. Vse *LP* iste besede v istem besedilu lahko povprečimo.

Z upoštevanjem povprečnih vrednosti lokalnih pogostosti $LP_{b,d}$ za besedo b pri besedilu d izračunamo uteži besed z enačbo:

$$w_{b,d} = f_{b,d} \cdot IDF_b \cdot \overline{LP}_{b,d}, \tag{3.5}$$

kjer je $f_{b,d}$ frekvenca besede b v besedilu d in IDF_b relativna redkost besede b.

Ko je program izračunal vse utežne vektorje besedil, se dano besedilo primerja z besedili korpusa na enega od načinov, opisanih v spodnjih podpoglavjih.

Primer dveh besedil:

»Besedilo je z besedami izražena misel. Vsako besedno sporočilo še ni besedilo. Med besedila prištevamo le nekatera sporočila. Te besede niso besedilo.«

»To besedilo sestavljajo besede. Tudi sporočilo sestavjajo besede. Te besede so sporočilo.«

Prvo besedilo sestavljajo naslednje leme besed: *{besedilo, biti, z, beseda, izražen, misel, vsak, beseden, sporočilo, še, med, prištevati, le, nekateri, sporočilo, ta}*.

Množica lem drugega besedila je *{besedilo, biti, beseda, sporočilo, ta, sestavljati, tudi}*.

Uporabnik določi, da se upoštevajo le samostalniške besede. Dobimo naslednji množici besed, zapisani z mesti (indeksi) pojavitev besed v posameznem besedilu in seštevkom (frekvenco) vseh pojavitev v besedilu:

```
{
        besedilo:   0,   11,   13,   21;        [4]
        beseda:     1,   10,   20;              [3]
        misel:      5;                          [1]
        sporočilo:  8,   17;                    [2]
}
```

```
{
        besedilo:   1;                          [1]
        beseda:     3,   7,    9;               [3]
        sporočilo:  5,   11;                    [2]
}
```

Na podlagi zgornjih množic besed (lem) program zgradi vektorja frekvenc za obe besedili:

	besedilo	*beseda*	*misel*	*sporočilo*	
fr_A = [	*4,*	*3,*	*1,*	*2*	*]*
fr_B = [	*1,*	*3,*	*0,*	*2*	*]*

Izračunamo faktorje relativne redkosti besed, pri čemer za ta primer predpostavimo, da imamo 100 besedil, upoštevane besede pa se pojavijo v 100, 80, 30 in 50 besedilih.

$IDF_{besedilo} = ln(100/100) = ln(1) = 0$

$IDF_{beseda} = ln(100/80) = ln(1.25) = 0.223$

$IDF_{misel} = ln(100/30) = 1.204$

$IDF_{sporočilo} = ln(100/50) = 0.693$

Opazimo, da besede, ki se pojavijo v več različnih besedilih, dobijo manjšo težo. Če se neka beseda pojavi v vsakem besedilu, dobi utež enako 0 pri vseh vektorjih, zato se zanemari.

Ob okolici 10 besed, se lokalne pogostosti izračunajo kot:

$pojavitve\_v\_okolici_{besedilo,A} = (0,\ 1,\ 2,\ 1)$

$$
\begin{aligned}
LP_{besedilo,A} &= povp\{ e^{(0/20)}, && e^{(1/20)}, && e^{(2/20)}, && e^{(1/20)} \} \\
&= povp\{ e^{0}, && e^{0,05}, && e^{0,1}, && e^{0,05} \} \\
&= povp\{ 1, && 1.05, && 1.11, && 1.05 \} \\
&= 1.05
\end{aligned}
$$

$pojavitve\_v\_okolici_{beseda,A} = (1,\ 1)$

$LP_{beseda,A} = povp\{ e^{(0/20)}, e^{(0/20)}, \} = povp\{ 1, 1\} = 1$

$pojavitve\_v\_okolici_{misel,A} = (0)$

$LP_{misel,A} = povp\{ 1 \} = 1$

$pojavitve\_v\_okolici_{sporočilo,A} = (1,\ 1)$

$LP_{sporočilo,A} = povp\{ 1.05,\ 1.05 \} = 1.05$

$LP_{besedilo,B} = 1$

$LP_{beseda,B} = 1.11$

$LP_{misel,B} = 0$

$LP_{sporočilo,B} = 1.05$

V naslednjem koraku program izračuna vektorje uteži:

$$
\begin{aligned}
W_A &= [\ 4*0*1.05, && 3*0.223*1, && 1*1.204*1, && 2*0.693*1.05\ ] \\
&= [\ 0, && 0.669, && 1.204, && 1.455\ ] \\
\\
W_B &= [\ 1*0*1, && 3*0.223*1.11, && 0*1.204*0, && 2*0.693*1.05\ ] \\
&= [0, && 0.743, && 0, && 1.455\ ]
\end{aligned}
$$

3.3.2 KOSINUSNA PODOBNOST UTEŽNIH VEKTORJEV

Podane imamo utežne vektorje besedil, izračunane na podlagi (2.12) ali (3.5). Program paroma izračuna kosinusno podobnost utežnih vektorjev vseh besedil iz korpusa napram podanemu besedilu po enačbi (2.13). Izračunane vrednosti so omejene na interval med 0 in 1, kjer je 1 maksimalna podobnost (kot med vektorjema je 0). Sledi urejanje vrstnega reda besedil in izpis tistih, kjer je podobnost presegla mejo, ki jo je določil uporabnik.

Nadaljevanje primera:

$podobnost_{A,B} = W_A * W_B \ / \ |W_A| * |W_B| =$

$= (0 + 0.669*0.743 + 0 + 1.455*1.455) / (\sqrt{(0+0.669^2+1.204^2+1.455^2)} * \sqrt{(0+0.743^2+0+1.455^2)}) =$

$= (0.497 + 2.117) / (\sqrt{(0.448 + 1.450 + 2.117)} * \sqrt{(0.552+2.117)})$

$= 2.614 / (2.004 * 1.634) = 2.614 / 3.274 = 0.798$

$podobnost_{A,B} = 0.798$

3.3.3 GRUČENJE

Uporabili smo hierarhično metodo najbližjih sosedov (najkrajše razdalje). Naivni algoritem poteka tako:

1. Naredi toliko gruč, kolikor je besedil. Vsako besedilo daj v svojo gručo.
2. Poišči besedili z najmanjšo razdaljo (treba je izračunati kosinusno razdaljo vektorjev vseh parov gruč).
3. Če je najmanjša razdalja dveh gruč manjša od neke določene meje, nadaljuj, sicer zaključi.
4. Združi najbližji gruči v novo gručo ter izbriši obe stari. Pri združevanju v novo gruču, se utežna vektorja, ki predstavljata gruči, seštejeta (kot da bi sešteli besede obeh besedil).
5. Če je ostala le ena gruča, končaj, sicer se vrni na točko 2.

Uporabljena podatkovne struktura za gruče je:

```
class gruca
{
        List<int> besedila_idx;          //indeksi besedil v seznamu
        List<double> vektor_utezi;
}
List<gruca> gruce = new List<gruca>();
```

Jedro implementacije algoritma gručenja:

```
bool nadaljuj = gruce.Count > 1;
while (nadaljuj)
{
//poišči največjo podobnost dveh gruč (ter si zapomni katerih dveh):
 int idx_gruca1 = -1;
 int idx_gruca2 = -1;
 double najvecja_podobnost = 0;

 for (int i = 0; i < gruce.Count; i++)
 {
  for (int j = i; j < gruce.Count; j++)
  {
   if (i != j)    //ne primerjamo gruče same s sabo
   {
    double podobnost = pripomocki.COS_vektorjev (gruce[i].vektor_utezi,
                gruce[j].vektor_utezi, out podobnost);
    if (podobnost > najvecja_podobnost)
    {
     najvecja_podobnost = podobnost;
     idx_gruca1 = i;
     idx_gruca2 = j;
    }
   }
  }
 }
 if (najvecja_podobnost >= meja_podobnosti && idx_gruca1>=0 && idx_gruca2>=0)
 {  //združevanje gruč:
  gruca nova_gruca = new gruca();
  nova_gruca.besedila_idx.AddRange(gruce[idx_gruca1].besedila_idx);
  nova_gruca.besedila_idx.AddRange(gruce[idx_gruca2].besedila_idx);
  for (int k=0; k<gruce[idx_gruca1].vektor_utezi.Count; k++)
  {
   nova_gruca.vektor_utezi.Add(gruce[idx_gruca1].vektor_utezi[k]
                + gruce[idx_gruca2].vektor_utezi[k]);
 }
 //odstranjevanje starih gruč:
 if (idx_gruca1 < idx_gruca2)
 {
   gruce.RemoveAt(idx_gruca1);
  gruce.RemoveAt(idx_gruca2 - 1);
 }
 else
 {
  gruce.RemoveAt(idx_gruca2);
  gruce.RemoveAt(idx_gruca1 - 1);
 }
 //dodaj novo gručo:
 gruce.Add(nova_gruca);
 }
 else nadaljuj = false;  //dosegli smo mejo podobnosti gruč, zato končaj
 if (gruce.Count == 1) nadaljuj = false;
}
```

4. SKLEPNE UGOTOVITVE

Preučili smo nekaj pristopov k analizi besedil ter jih preizkusili v aplikaciji, ki smo jo razvili v ta namen. Ukvarjali smo se s klasificiranjem in primerjanjem besedil, izdelavo in urejanjem korpusov ter uporabo zbirk besed (slovarjev) in sorodnih izrazov (tezaver) v raznih pripomočkih. Uporabili smo več algoritmov, od naivnega Bayesovega klasifikatorja, AODE, računanje podobnosti na osnovi kosinusnih razdalj vektorskih predstavitev besedil, pa do hierarhičnega gručenja. Dodali smo jim tudi nekaj svojih izboljšav. Različne metode so se izkazale za različno uspešne pri posameznih vrstah problemov. V začetni fazi razvoja aplikacije nismo naredili okvirnega načrta strukture in delovanja programa, pač pa smo sproti in po potrebi dodajali nove funkcionalnosti in izboljšave. Posledica tega je težja preglednost in obvladljivost napak v programu.

V splošnem za algoritme strojnega učenja velja, da so tem bolj uspešni, čim več učnih podatkov imajo na voljo. Pri veliki količini podatkov nastopijo težave zaradi počasnosti in velike porabe pomnilnika. Z našo aplikacijo imamo že pri uporabi malo večjega korpusa z milijonom besed lahko težave na starejših računalnikih. Korpusi morajo biti ustrezno oblikoslovno označeni in lematizirani. Prosto dostopnih označenih korpusov ni prav veliko. Lahko sicer zgradimo novega, če uporabimo pripomoček za gradnjo korpusov, ki smo ga v ta namen dodali naši aplikaciji, toda označevanje besedil ter vpisovanje podatkov o avtorju, založbi, zvrsti, letu itd. je zelo zamudno. Ko besedilo označimo, zasede več prostora na disku oziroma v pomnilniku računalnika. Pri datotekah večjih od 100MB ima veliko XML urejevalnikov težave. Poleg tega se moramo pri vsaki gradnji učnih podatkov vprašati, ali so reprezentativen vzorec realnega sveta.

Težave s počasnostjo in pomanjkanjem spomina se pojavijo tudi pri obdelavi manjših korpusov, če uporabimo več atributov za klasifikacijo. Pri klasifikaciji je uporaba velike množice atributov (besed), tudi takih, ki se ne pojavijo v klasificiranem besedilu, ampak le v drugih besedilih korpusa, nesmiselna. Pogojna verjetnost razreda za atribut, ki ne obstaja v danem besedilu, je enaka nič, zato se pri računanju ne spremeni rezultat klasifikacije, porabi pa se procesorski čas in pomnilnik. Smiselno je, da za klasifikacijo ne izberemo vseh besed, ampak le tiste, ki vsebujejo največ informacije, ki jo želimo izluščiti iz množice besedil. Postavi se vprašanje, katere so te besede. Pri ugotavljanju sloga pisanja besedil je smiselno opazovati predvsem besede, ki same po sebi ne nosijo velike sporočilnosti in so relativno redke, a so velikokrat značilne pri posameznih avtorjih. Tak primer so prislovi ali predlogi.

Veliko število atributov pomeni počasnejše štetje njihovih pojavitev v besedilih korpusov. Za vsako besedo mora program preveriti, ali se nahaja na seznamu atributov in h kateremu besedilu spada. Hitrejše delovanje bi lahko zagotovili z uporabo urejenih seznamov, ki pa smo jih uporabili le občasno.

Pri kosinusni razdalji v vektorski predstavitvi besedil imajo tudi besede, ki se ne pojavijo v danem besedilu, vpliv na podobnost. Vsaka različna beseda predstavlja eno dimenzijo v vektorju besedila, kar pomeni, da se pri dodajanju nove besede (dimenzije) spremeni

dolžina vektorja, zato se spremeni tudi izračun kosinusne podobnosti. Različnih lem je več tisoč, kar ima za posledico ogromne vektorje. Pri velikem številu različnih besedil, bo algoritem potreboval mnogo prostora v pomnilniku, kjer bo vektorje hranil. Algoritmu za primerjanje podobnosti besedil bi lahko zmanjšali prostorsko zahtevnost tako, da bi prebral in hranil v pomnilniku le eno besedilo iz korpusa naenkrat. Ta sprememba ne pride v poštev pri algoritmu hierarhičnega gručenja, kjer je veliko primerjanj gruč, ravno pri tem algoritmu pa so težave s pomnilniškim prostorom najbolj očitne. Ta algoritem je časovno precej zahteven, velikostnega reda $O(n^3)$. Boljši algoritem (imenovan SLINK), ki je velikostnega reda $O(n^2)$, bi lahko bil ena od izboljšav, ki se jih zavedamo. Pri računaju kosinusne razdalje vektorjev bi lahko uporabili tudi več različnih formul za izračun uteži besed, npr. TF-IDF uteži (»Term Frequency - Inverse Document Frequency«).

Razviti pripomočki za pisanje besedil so še v začetni prototipni fazi in bi potrebovali še mnogo razširitev in izboljšav, da bi bili konkurenčni današnjim urejevalnikom besedil. Iskanje slovničnih napak, preverjanje črkovanja v kontekstu, avtomatsko dokončanje besed in ponujanje boljših izbir pri ugotovljenih napakah so le nekatere od možnosti izboljšav. Slovar, ki ga uporablja črkovalnik v našem programu, je avtomatsko zgrajen iz korpusa jos1M, zato ne vsebuje nekaterih oblik besed, vsebuje pa določene besede, ki so le skupek znakov in niso knjižno pravilne. Slovar bi bilo potrebno izboljšati ali pa uporabiti nek drug slovar.

Za računalniško oblikoslovno označevanje in lematizacijo besed smo uporabili tuje programe. Uporaba označevanja je zato okorna (vsi podatki morajo biti v datotekah) in počasna, opazili smo tudi nekaj napak pri delovanju. Označevalnik in lematizator bi zato morali implementirati znotraj naše aplikacije.

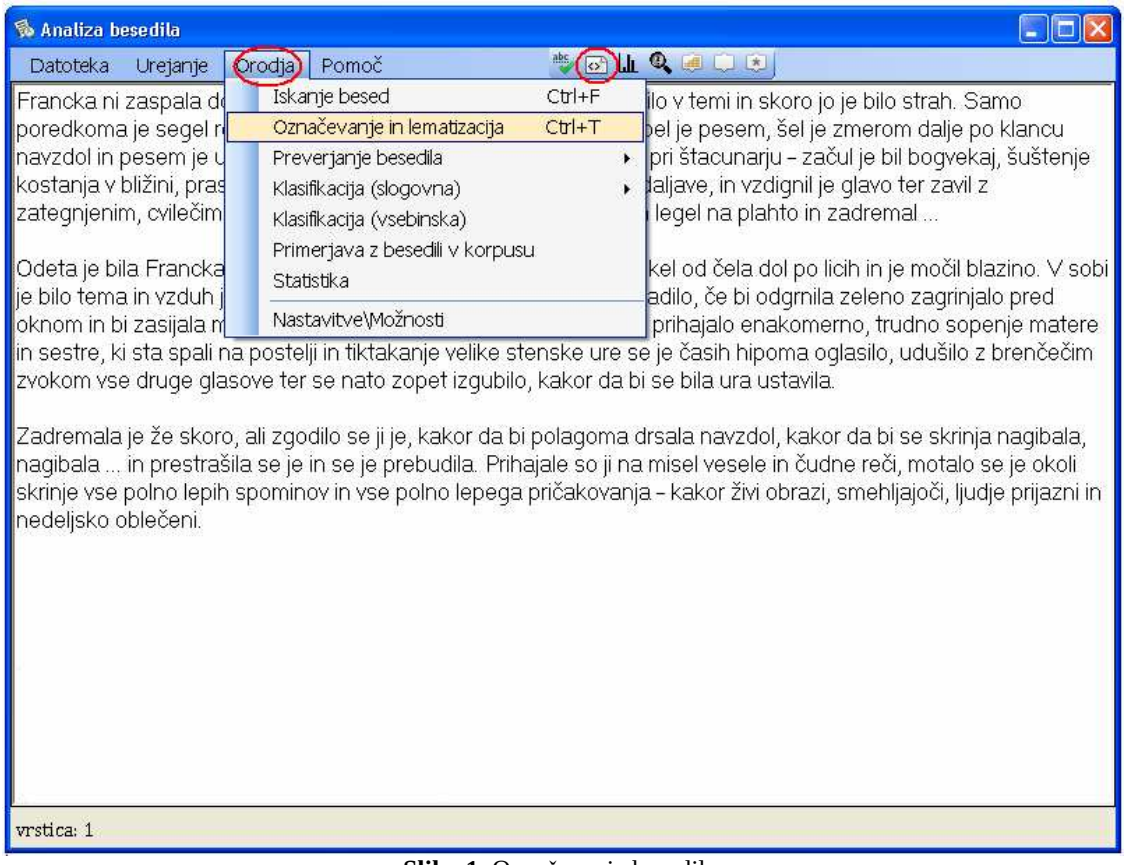
Dodatek A: Uporabniški vmesnik in primeri uporabe aplikacije

V dodatku opisujemo uporabniški vmesnik in navajamo nekaj primerov uporabe aplikacije s pripomočki za pisanje in analizo besedil. Aplikacija teče na ogrodju .NET v okolju Windows.

A.1 Označevanje in lematizacija besedila

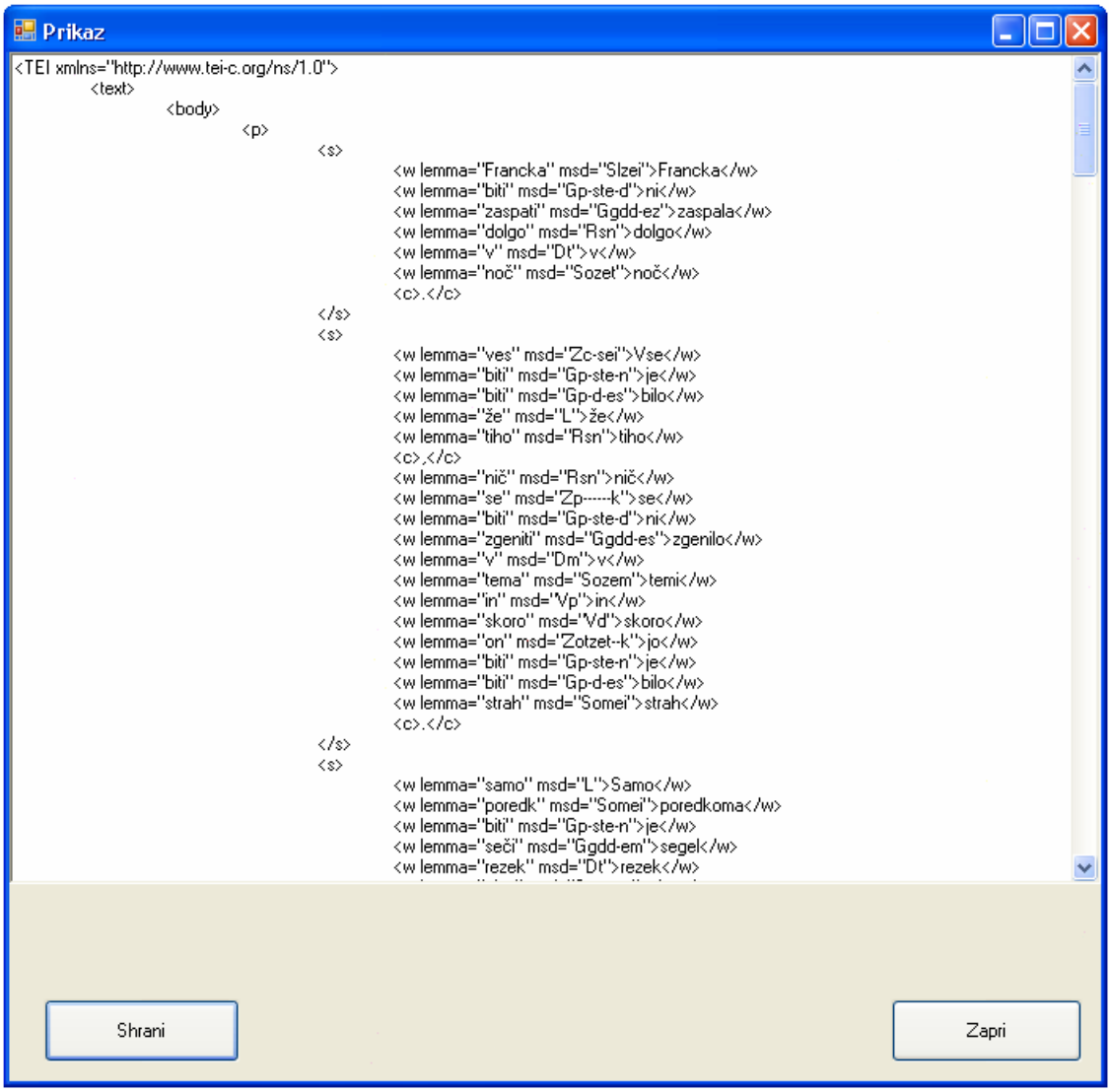
Besedila, ki jih želimo analizirati, morajo biti predhodno lematizirana in označena. To pomeni, da se za vsako besedo določi njena oblikoslovna oblika (besedna vrsta, sklon, število, sklanjatev ...) ter njena lema (osnovna oblika besede).

V glavnem meniju izberemo *Orodja* in nato *Označevanje in lematizacija* ali pa kliknemo bljižnico v orodni vrstici.



Slika 1. Označevanje besedila.

Požene se poseben program [13], ki opravi avtomatsko lematizacijo in oblikoslovno označevanje. Ko je postopek končan, se odpre novo okno s prikazom označenega besedila v obliki TEI-XML [14]. Besedilo lahko shranimo kot XML datoteko.



Slika 2. Prikaz označenega besedila v TEI-XML shemi.

A.2 Nalaganje označenega besedila iz datotek v zapisu TEI-XML

1.) Besedilo, ki je označeno in lematizirano ter shranjeno v XML datoteki, odpremo z izbiro v glavnem meniju:

Datoteka -> Odpri -> Označeno besedilo (xml)

Besedilo se pretvori v običajno (neoznačeno) obliko in prikaže v glavnem oknu. Lahko ga popravljamo in shranimo kot besedilno datoteko brez oblikovanja (txt). Če želimo shraniti spremenjeno besedilo v TEI-XML format, moramo še enkrat pognati orodje za označevanje in lematizacijo.

2.) Besedilo lahko izvlečemo iz zbirke označenih besedil (korpusa) in ga shranimo v ločeno datoteko z orodjem za pregledovanje in urejanje korpusov [poglavje A.3].

A.3 Delo z besedilnimi korpusi

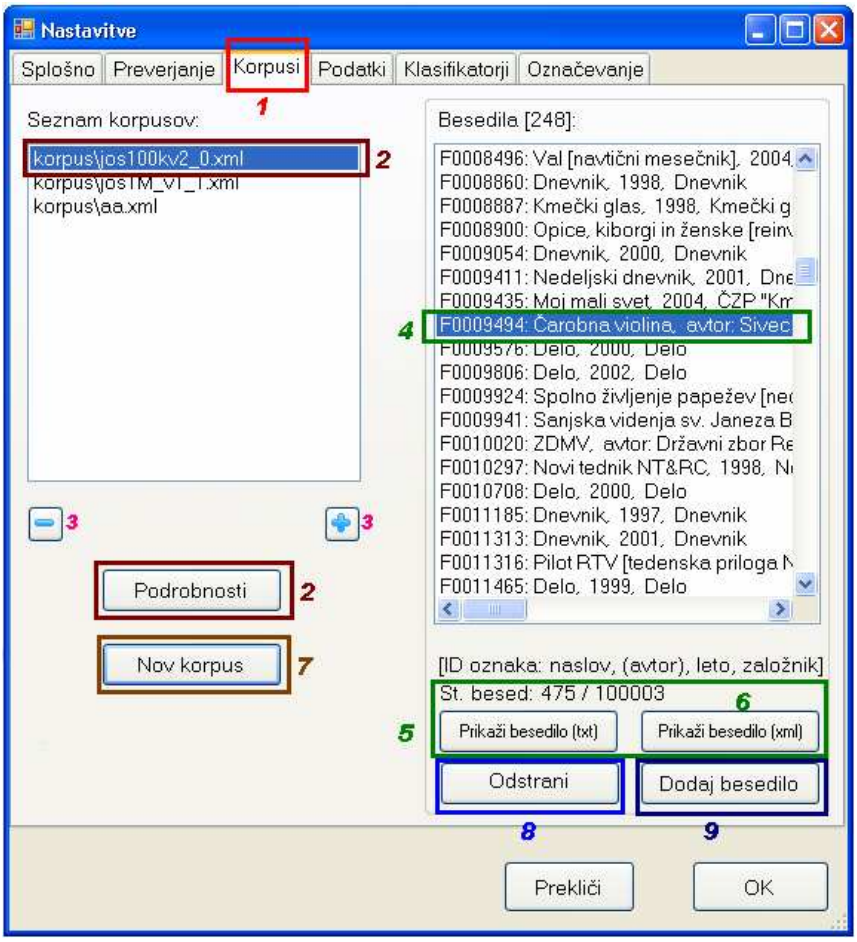
V glavnem meniju izberemo: *Orodja -> Nastavitve\Možnosti*. Prikaže se okno za nastavitve [slika 3], kjer izberemo zavihek *Korpusi* [slika 3, točka 1].

A.3.1 Prikaz besedil v korpusu:

- Izberemo korpus iz seznama korpusov in kliknemo gumb *Podrobnosti* [slika 3, točka 2].

- Korpus dodamo na seznam oziroma odstranimo iz seznama z gumboma + in - [slika 3, točka 3].

- Na seznamu besedil prikazanega korpusa izberemo željeno besedilo [slika 3, točka 4]. Na voljo imamo 2 načina prikaza: kot neoznačeno besedilo [slika 3, točka 5] ali v označeni obliki [slika 3, točka 6]. Izbrano besedilo se odpre v *prikaznem oknu* [slika 2], kjer ga lahko shranimo v datoteko.

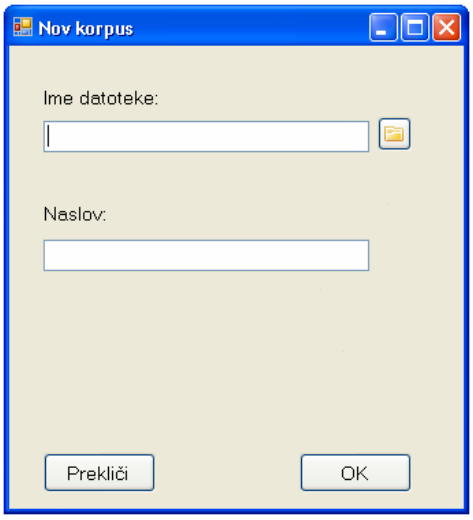


Slika 3. Urejanje korpusov

A.3.2 Kreiranje novega korpusa:

- Kliknemo gumb *Nov korpus* [slika 3, točka 7].

- Odpre se okno za vpis podatkov o novem korpusu [slika 4], kamor vpišemo njegov naslov (ime) in izberemo datoteko, kamor se bo shranil.

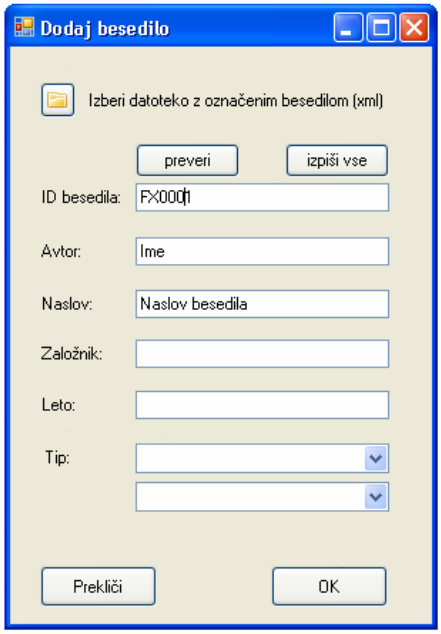


Slika 4. Kreiranje novega korpusa.

A.3.3 Urejanje korpusa:

- Izbrano besedilo [slika 3, točka 4] odstranimo z gumbom *Odstrani* [slika 3, točka 8]. Pojavi se opozorilo, za potrditev kliknemo *OK*.

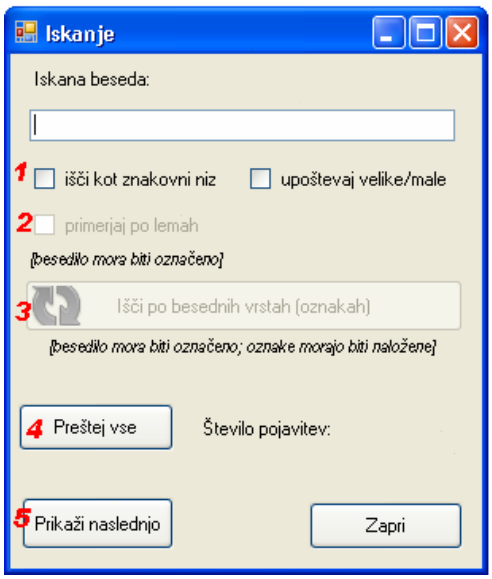
- Besedila, ki so shranjena v označeni obliki TEI-XML in shranjena v datoteki, lahko dodamo v korpus s klikom na gumb *Dodaj besedilo* [slika 3, točka 9]. Odpre se okno za vpis podatkov o besedilu [slika 5]. Določiti moramo izvorno datoteko, izbrati unikatno identifikacijsko oznako znotraj korpusa in vpisati podatke o besedilu.



Slika 5. Dodajanje besedila v korpus.

A.4 Iskanje besed

Okno za iskanje besed odpremo v meniju: *Orodja -> Iskanje besed* ali z bližnjico Ctrl+F.



Slika 6. Navadno iskanje.

Za iskanje po lemah in oznakah besed mora biti besedilo predhodno označeno. Opis elementov okna in delovanja:

1.) Če je možnost *»išči kot znakovni niz«* omogočena, se kot rezultat upoštevajo vsi podnizi besede (npr. »ata« v besedi »vratar«), drugače pa išče le po celih besedah.

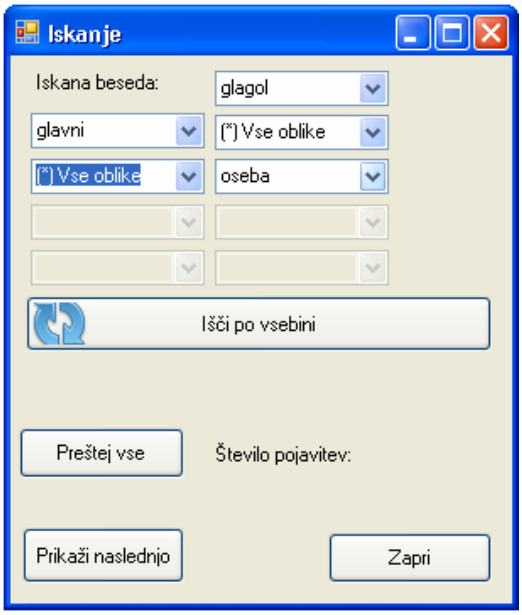
2.) Možnost *»primerjaj po lemah«* pomeni, da program išče v označenem besedilu in primerja, če je lema besede enaka iskani.

3.) Gumb za spreminjanje načina delovanja: navadno iskanje (po vsebini besed) ali po oznakah besed.

4.) Prešteje vse pojavitve iskane besede.

5.) Pomakne kazalnik besedila na naslednjo besedo, ki ustreza kriteriju iskanja.

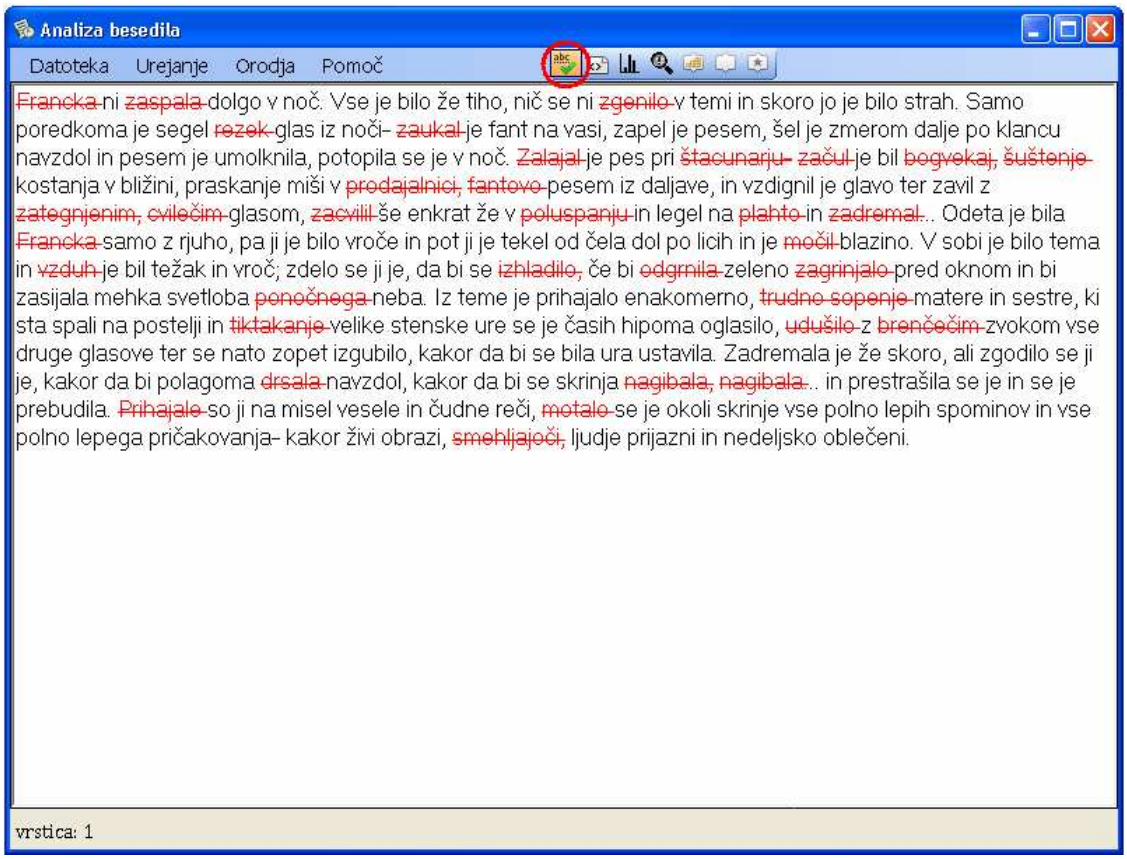
Pri označanem besedilu deluje iskanje tudi v načinu [slika 6, točka 3] za iskanje besed po oznakah. V tem načinu namesto znakovnega niza izberemo oblikoslovne oblike besed, ki jih želimo iskati [slika 7].



Slika 7. Iskanje po oblikoslovnih oznakah.

A.5 Preverjanje črkovanja

Preverjanje črkovanja besed vklopimo ali izklopimo z ikono v orodni vrstici poleg glavnega menija ali s tipkama Alt+C.



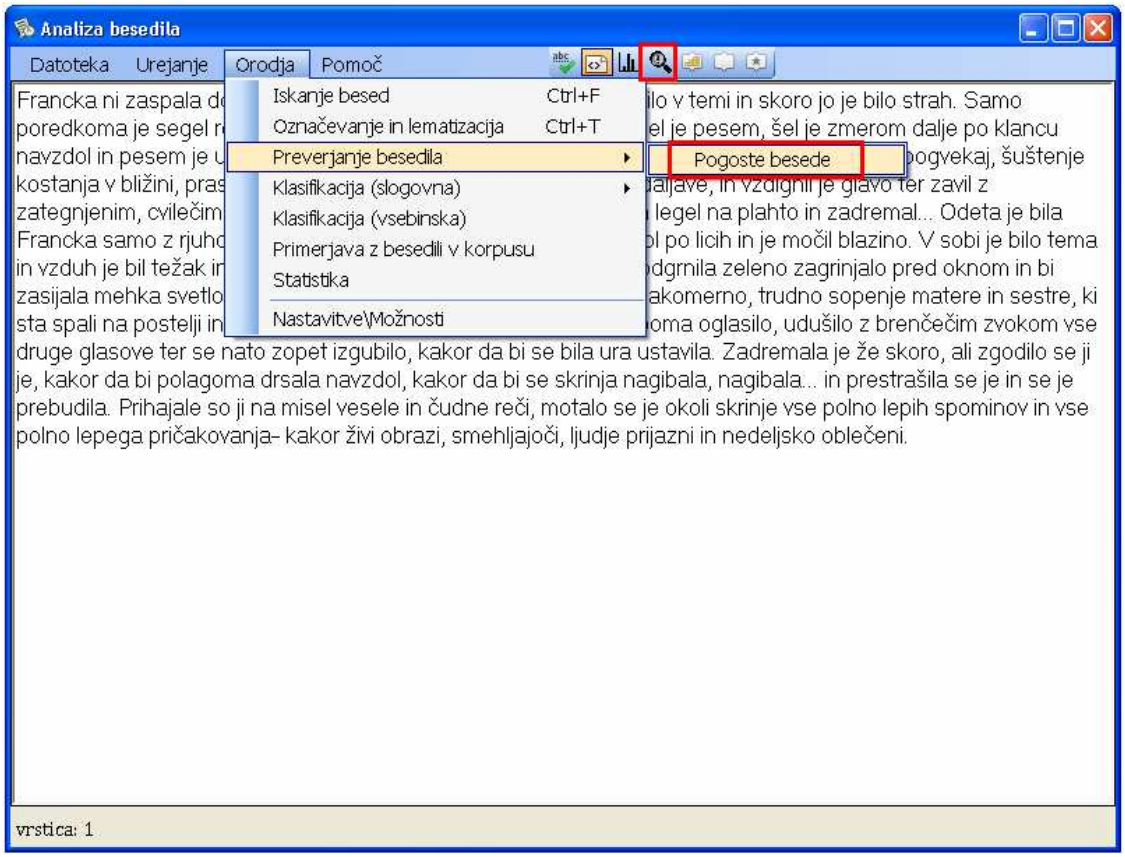
Slika 8. Vklop in delovanje črkovalnika.

Črkovalnik označi vse besede, ki jih ne najde v slovarju. Med delovanjem črkovalnika lahko uporabimo desni klik na katerokoli besedo v našem besedilu. V tem primeru se pojavi okno z možnostjo dodajanja oziroma odstranjevanja besede iz slovarja.

Nastavimo lahko barvo in tip oznake neznanih besed. To storimo v oknu za nastavitve, ki ga odpremo z izbiro *Orodja -> Nastavitve\Možnosti*. Nastavitve črkovalnika se nahajajo v zavihku *Splošno*.

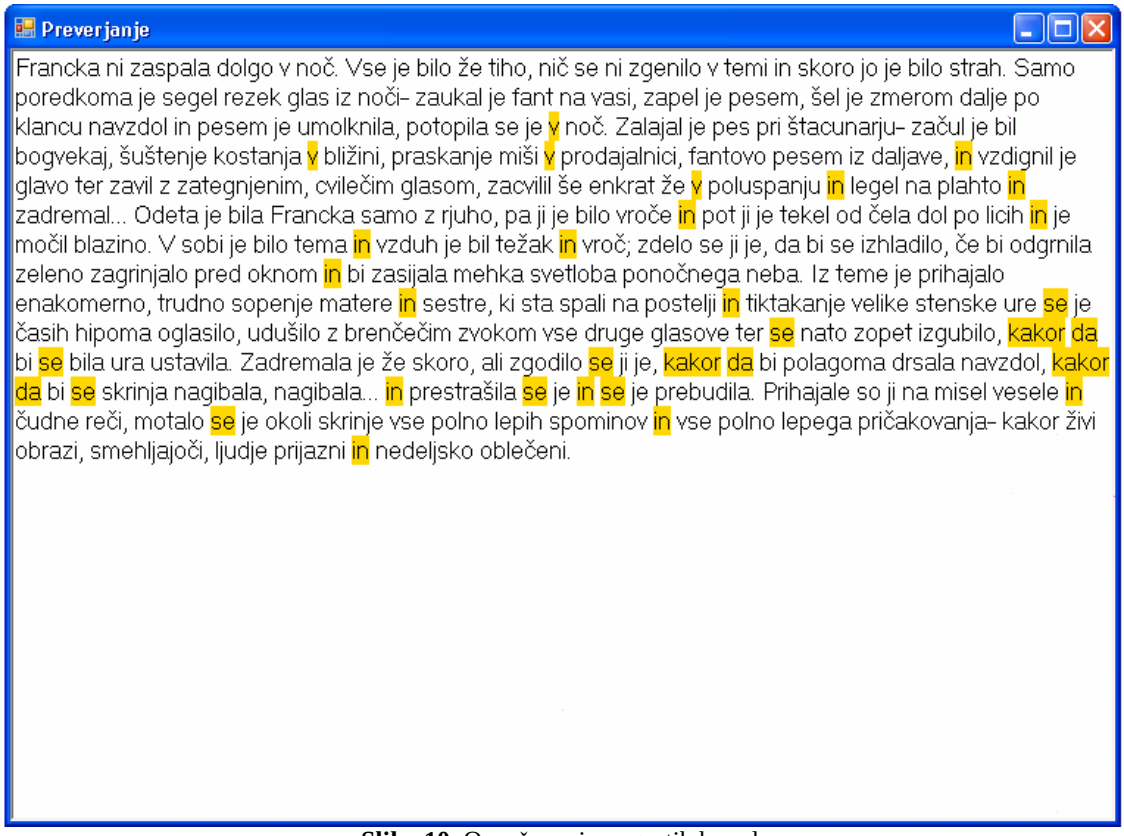
A.6 Preverjanje pogostosti besed in tezaver

Preverjanje pogostih besed v besedilu začnemo s klikom na *Orodja -> Preverjanje besedila -> Pogoste besede* ali s klikom na četrto ikono v orodni vrstici poleg glavnega menija. Besedilo ne sme biti prekratko. Če ni oblikoslovno označeno in lematizirano, se ta postopek avtomatsko zažene.



Slika 9. Začetek preverjanje pogostih besed.

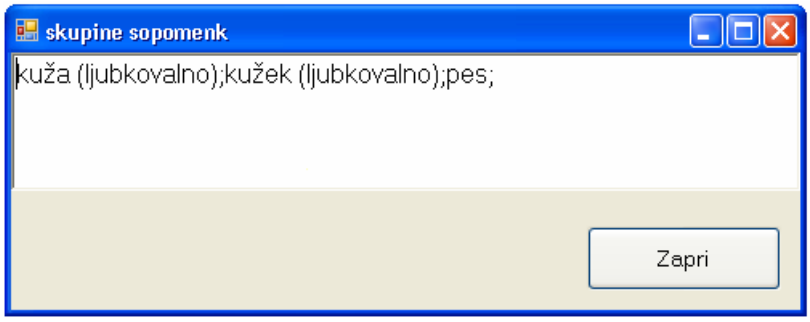
Odpre se dodatno okno za preverjanje besedila (pogostosti besed), kjer se besede, ki presegajo določeno pogostost v okolici, obarvajo [slika 10].



Slika 10. Označevanje pogostih besed.

A.6.1 Tezaver

Z desnim klikom na besedo v oknu za preverjanje besedila dobimo možnost iskanja sopomenk izbrane besede. Odpre se okno, kjer se izpišejo morebitne skupine sopomenk [slika 11]. V primeru smo s klikom izbrali besedo »pes« v tretji vrstici besedila.

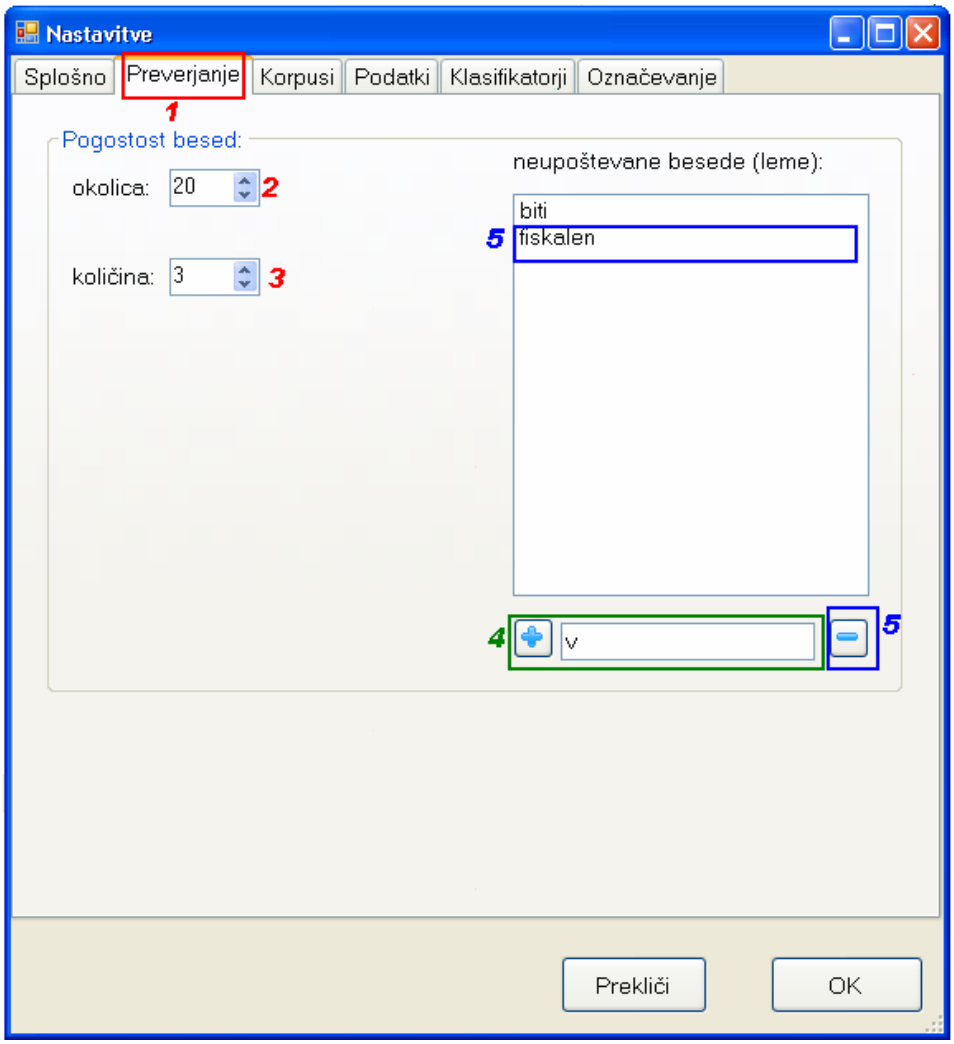


Slika 11. Tezaver

Vsaka vrstica pomeni svojo skupino sopomenk, ki se lahko razlikujejo po namenu uporabe in pomenu.

A.6.2 Nastavitve iskanja pogostih besed

Odpremo okno z nastavitvami: *Orodja -> Nastavitve\Možnosti*. Izberemo zavihek *Preverjanje* [slika 12, točka 1].



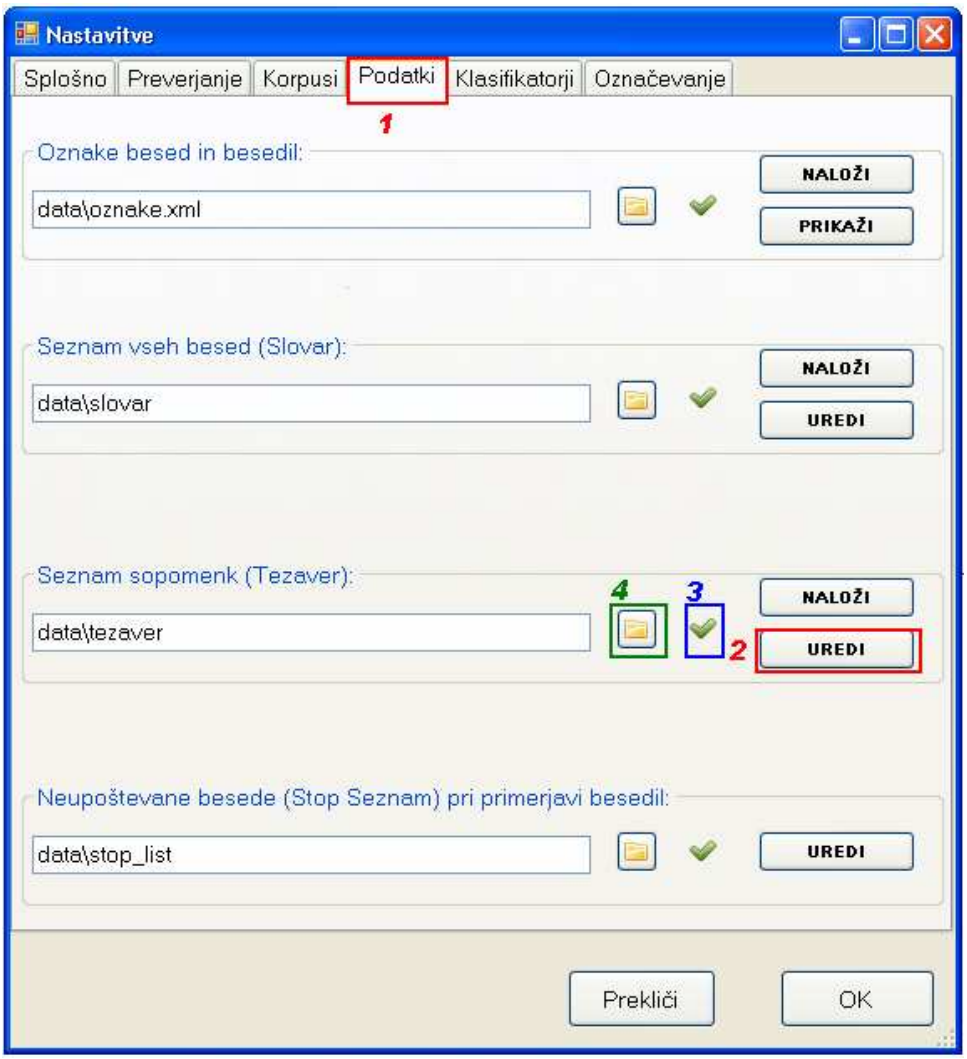
Slika 12. Nastavitve iskanja pogostih besed.

Nastavimo lahko:

- okolico besede [slika 12, točka 2], to je število zaporednih besed, ki jih upoštevamo pri štetju ponovitev besede;
- količino [slika 12, točka 3], ki predstavlja mejo števila ponovitev besed;
- neupoštevane besede, so besede (oz. njihove leme), ki jim ne štejemo ponovitev. V seznamu lahko leme dodajamo [slika 12, točka 4] ali odstranjujemo [slika 12, točka 5].

A.6.3 Urejanje tezavra

Tezaver lahko urejamo v nastavitvah: *Orodja -> Nastavitve\Možnosti* in izberemo zavihek *Podatki* [slika 13, točka 1]. Kliknemo gumb *Uredi* [slika 13, točka 2]. Ime datoteke, kjer so shranjeni vnosi, lahko zamenjamo s klikom na gumb za izbiro datoteke [slika 13, točka 4]. Oznaka [slika 13, točka 3] poleg gumba pomeni, da datoteka obstaja in je tezaver naložen.

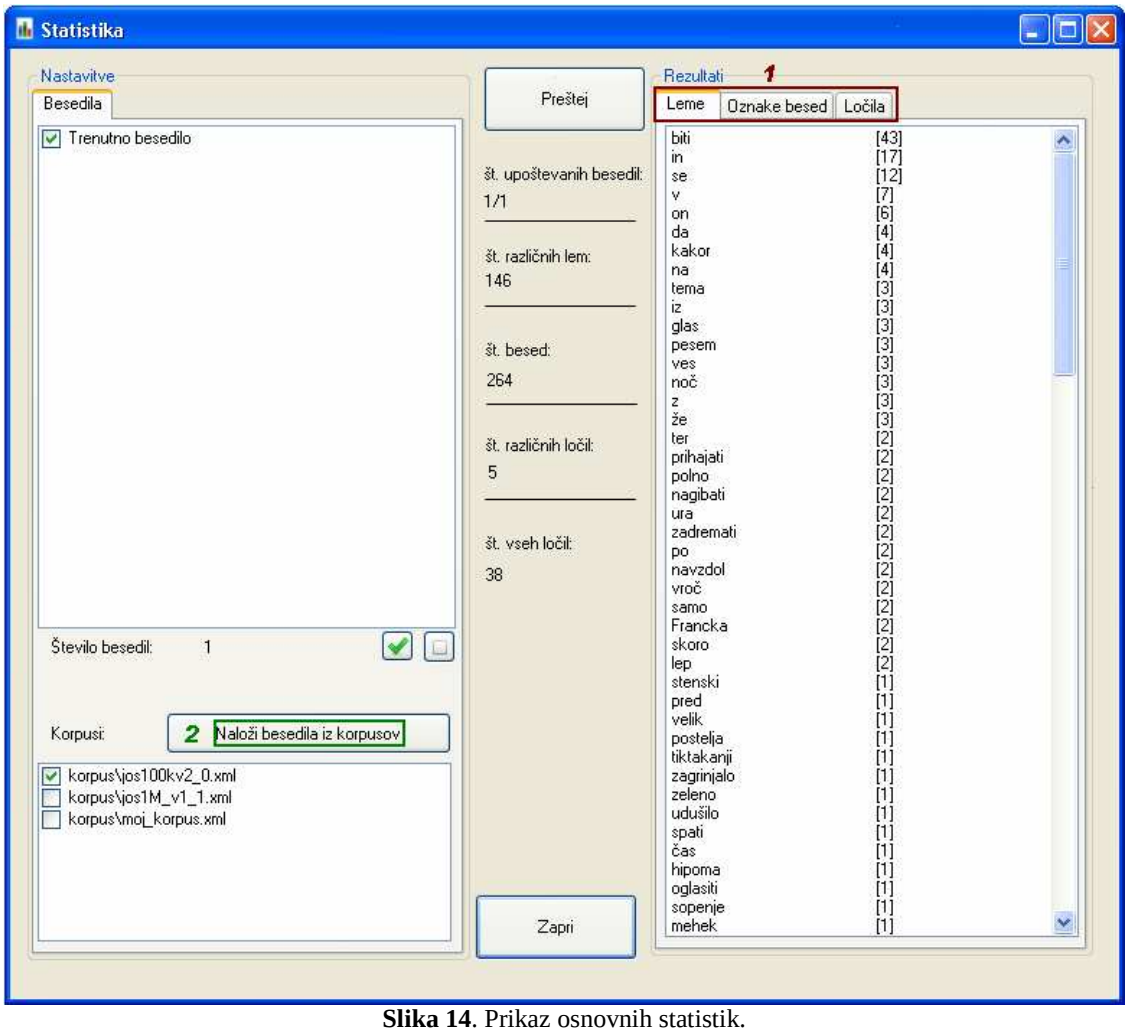


Slika 13. Nastavitev datoteke s tezavrom

Ko kliknemo *Uredi*, se odpre prikazno-urejevalno okno, kjer lahko popravljamo in dodajamo vnose. Vsaka vrstica pomeni novo skupino sopomenk. Sopomenke iste skupine so med seboj ločene z znakom ' ; '.

A.7 Izračun osnovnih statistik besedila

Za besedilo, ki ga trenutno obdelujemo (in je prikazano v glavnem oknu), ali za besedila shranjena v korpusih, lahko preštejemo vse besede glede na leme ali oblikoslovne oblike. To storimo z orodjem za prikaz statistike, ki ga poženemo v glavnem meniju: *Orodja -> Statistika* ali s klikom na ikono v orodni vrstici. Če želimo, da se statistično obdela trenutno besedilo, mora biti to označeno in lematizirano. V nasprotnem primeru se statistika trenutnega besedila ne prikaže in lahko pregledujemo le statistike besedil iz korpusov.



Slika 14. Prikaz osnovnih statistik.

Odpre se okno, s statistiko trenutnega besedila. Pogled rezultatov spreminjamo z zavihki [slika 14, točka 1]. Izbiramo med lemami, oblikoslovnimi oznakami in ločili. Oznake so zapisane v kratki obliki, če želimo videti njihov pomen, to storimo s klikom na izbrano vrstico z oznako. Če želimo prikaz statistik za besedila iz korpusov, določimo korpuse iz seznama in kliknemo *Naloži besedila iz korpusov* [slika 14, točka 2].

A.8 Klasifikacija glede na slog besedila

Besedilo, ki je prikazano v glavnem oknu, lahko klasificiramo. V glavnem meniju izberemo *Orodja -> Klasifikacija (slogovna) -> Klasifikacija besedila* ali kliknemo ikono  v orodni vrstici.

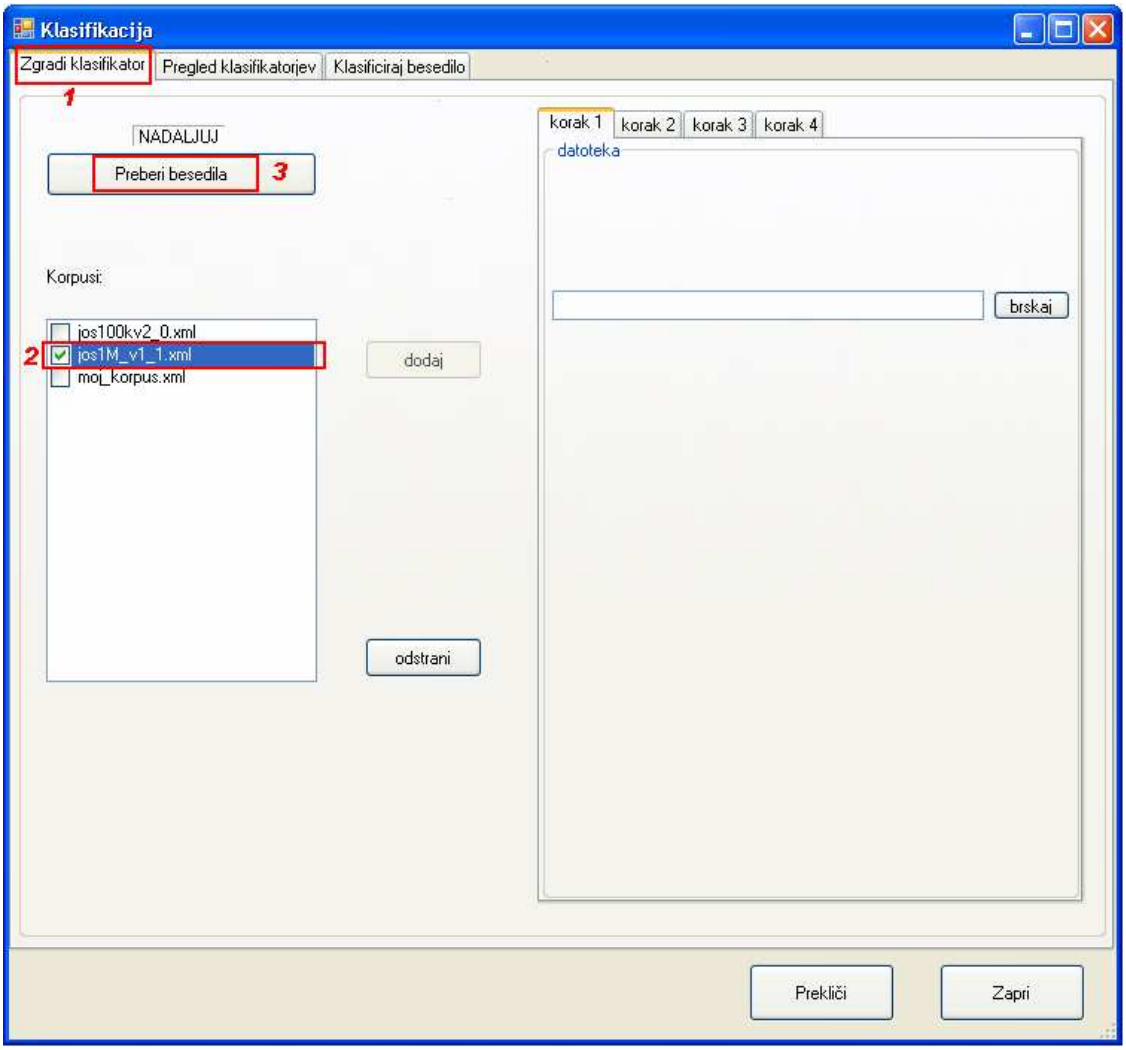
A.8.1 Izgradnja klasifikatorja

Če še nimamo zgrajenega (in shranjenega v datoteki) nobenega klasifikatorja ali če želimo ustvariti novega, to storimo z izbiro zavihka *Zgradi klasifikator* [slika 15, točka 1] ali z izbiro v glavnem meniju: *Orodja -> Klasifikacija (slogovna) -> Nov klasifikator*.

Gradnja klasifikatorja poteka v šestih korakih:

1. izbiranje korpusov [slika 15],
2. izbiranje atributov - besednih vrst [slika 16],
3. izbiranje atributov - lem [slika 17],
4. izbiranje atributov - ločil [slika 18],
5. določanje kategorije razredov [slika 19],
6. shranjevanje v datoteko.

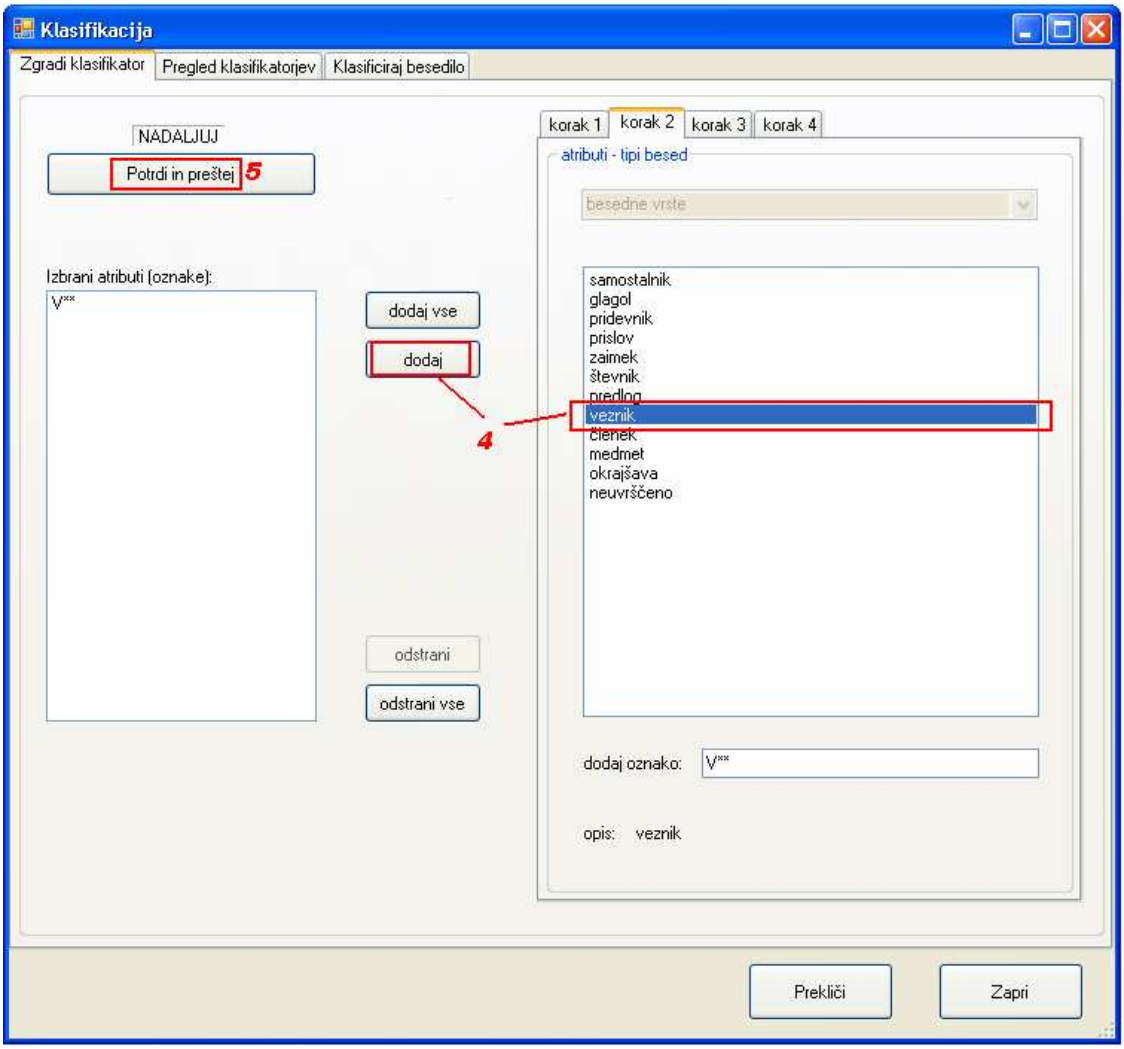
A.8.1.1 Izbiranje korpusov



Slika 15. Gradnja klasifikatorja - izbira korpusov.

Na seznamu korpusov izberemo željene korpuse [slika 15, točka 2], ki se bodo uporabili pri gradnji klasifikatorja. Na seznam korpusov lahko dodamo nove datoteke, ki vsebujejo označene in lematizirane korpuse v obliki TEI-XML, lahko pa jih tudi odstranimo. Ko smo končali izbiro korpusov, kliknemo gumb *Preberi besedila* [slika 15, točka 3].

A.8.1.2 Izbiranje atributov - besednih vrst

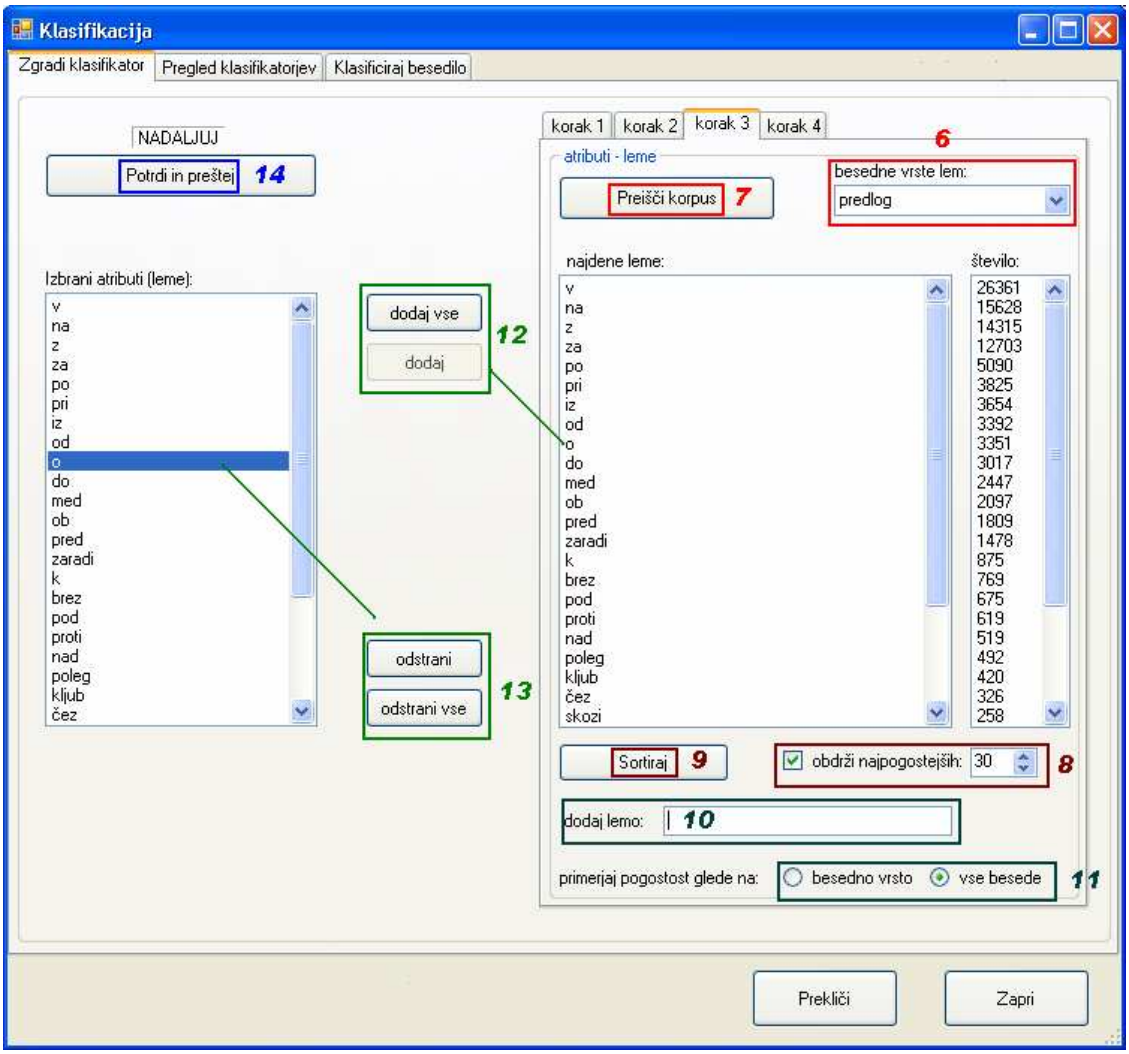


Slika 16. Gradnja klasifikatorja - izbira atributov (besedne vrste).

Dodamo lahko poljubno mnogo različnih atributov z izbiro na seznamu možnih in klikom na *dodaj* [slika 16, točka 4]. Že izbrani atribut lahko odstranimo s klikom na seznamu Izbranih atributov in gumbom *odstrani*.

Ko smo končali, izbiro potrdimo s klikom na *Potrdi in preštej* [slika 16, točka 5].

A.8.1.3 Izbiranje atributov - lem



Slika 17. Gradnja klasifikatorja - izbira atributov (leme besed).

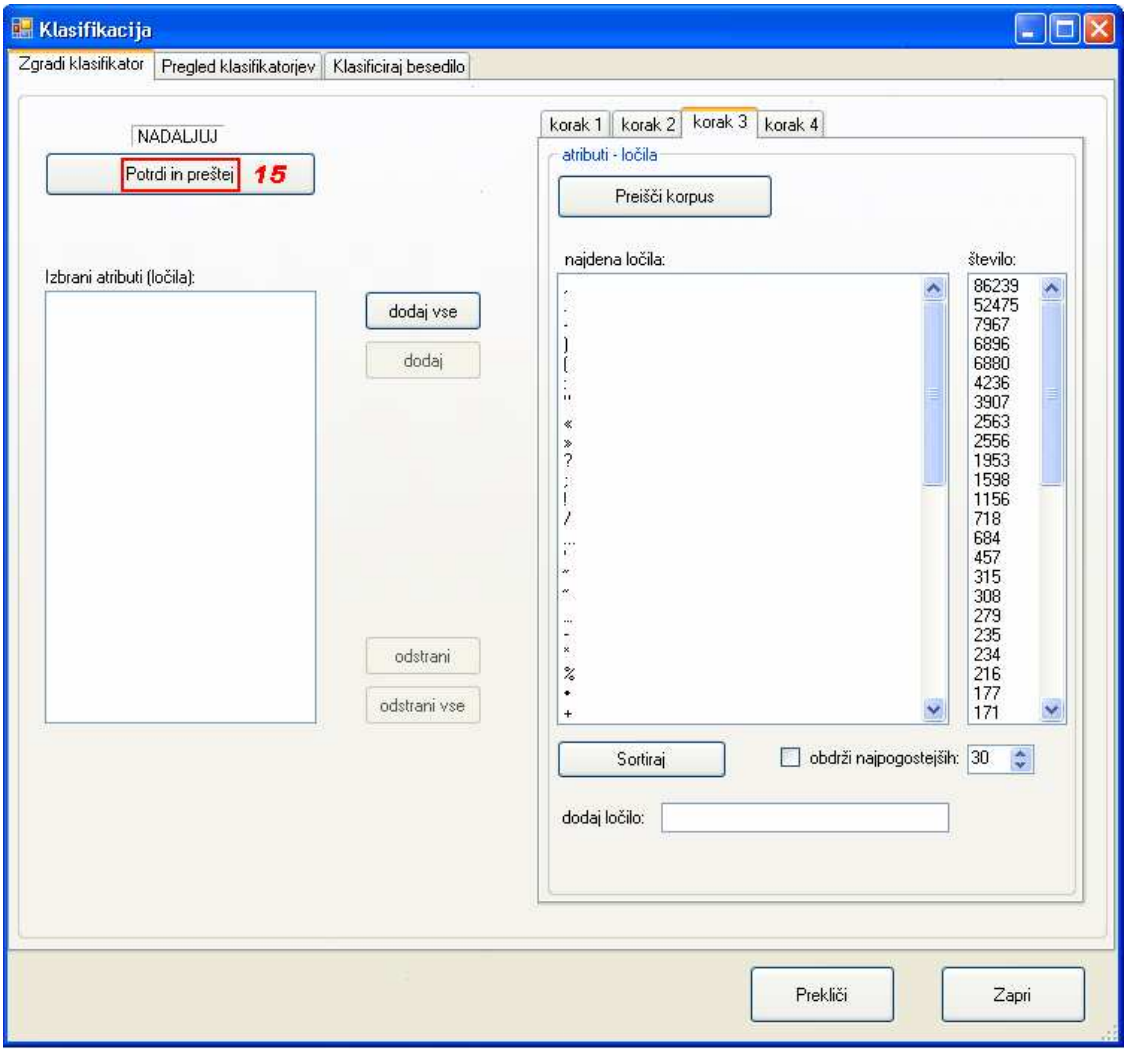
Določimo besedno vrsto lem [slika 17, točka 6]. Če želimo, lahko program preveri, katere leme te vrste se nahajajo v predhodno izbranih korpusih [slika 17, točka 7]. Poiskane leme lahko sortiramo po številu pojavitev [slika 17, točka 9], še prej pa določimo, ali naj program obdrži na seznamu le določeno število najpogostejših lem [slika 17, točka 8].

Lemo, ki jo želimo dodati na seznam atributov, vpišemo v okno za dodajanje [slika 17, točka 10] ali izberemo s klikom na seznamu najdenih (pri tem se avtomatsko vpiše v okno za dodajanje). V oknu za način računanja pogostosti atributov/lem [slika 17, točka 11] izberemo, ali naj se pogostost izračuna glede število vseh besed besedila ali glede na število vse besed iste besedne vrste (v našem primeru število predlogov).

Leme dodajamo na seznam izbranih atributov [slika 17, okno na levi polovici] s klikom na gumba *dodaj* ali *dodaj vse* [slika 17, točka 12], ki doda med atribute vse leme iz seznama najdenih lem. Že izbrane atribute odstranjujemo z gumboma *odstrani* in *odstrani vse* [slika 17, točka 13]. Izbiro atributov-lem končamo s klikom na *Potrdi in preštej* [slika 17, točka 14].

A.8.1.4 Izbiranje atributov - ločil

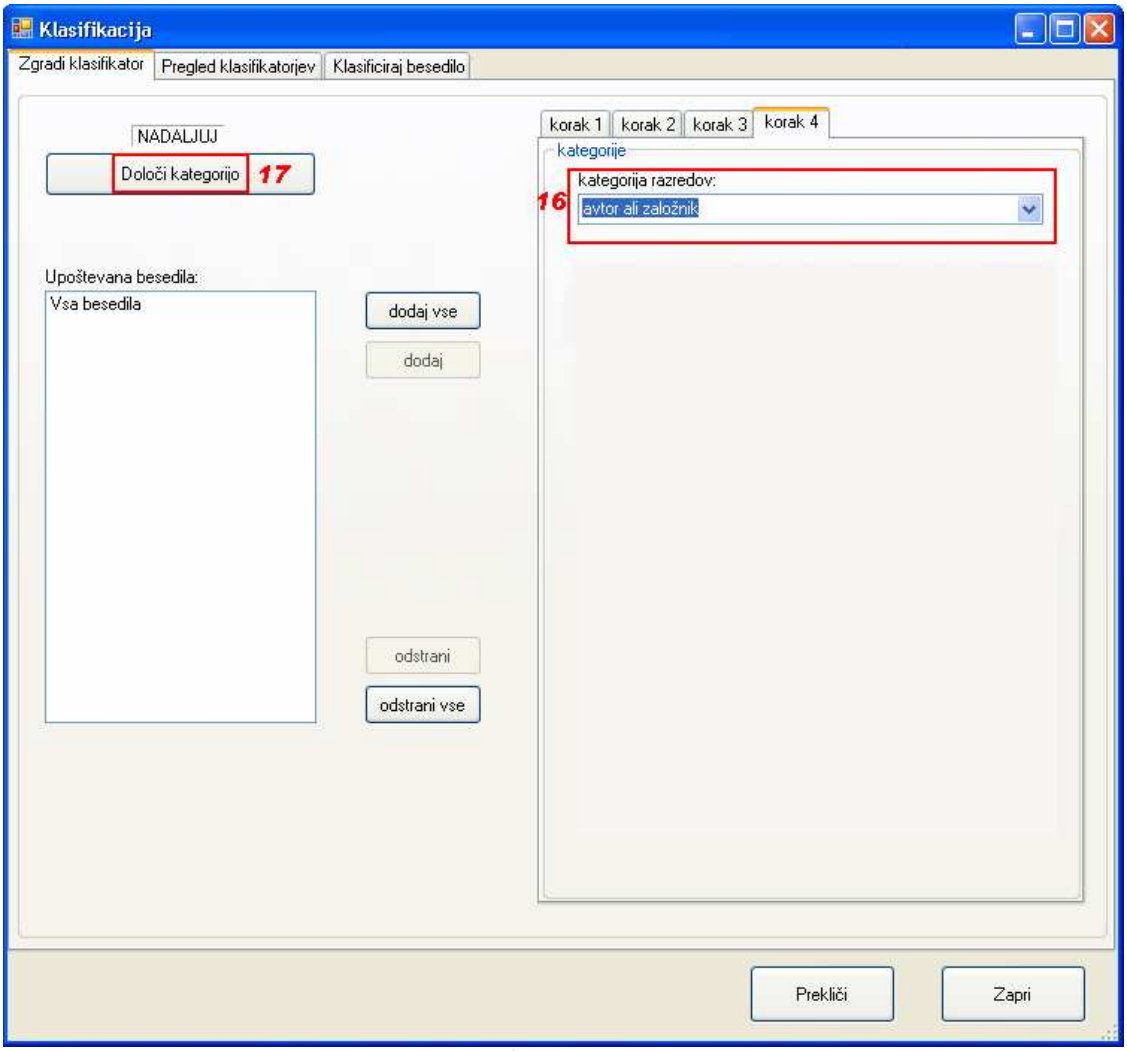
Ločila dodajamo med atribute po enakem postopku kot leme [poglavje A.8.1.3]. V našem primeru nismo izbrali nobenega ločila, kliknemo *Potrdi in preštej*.



Slika 18. Gradnja klasifikatorja - izbira atributov (ločila).

A.8.1.5 Določanje kategorij

Izberemo kategorijo iz izvlečnega seznama [slika 19, točka 16]. V našem primeru smo izbrali kategorijo »*avtor ali založnik*«, zato se besedila klasificirajo po avtorjih, besedila, ki nimajo znanih avtorjev, pa po založniku. Za naslednji korak kliknemo *Določi kategorijo* [slika 19, točka 17].



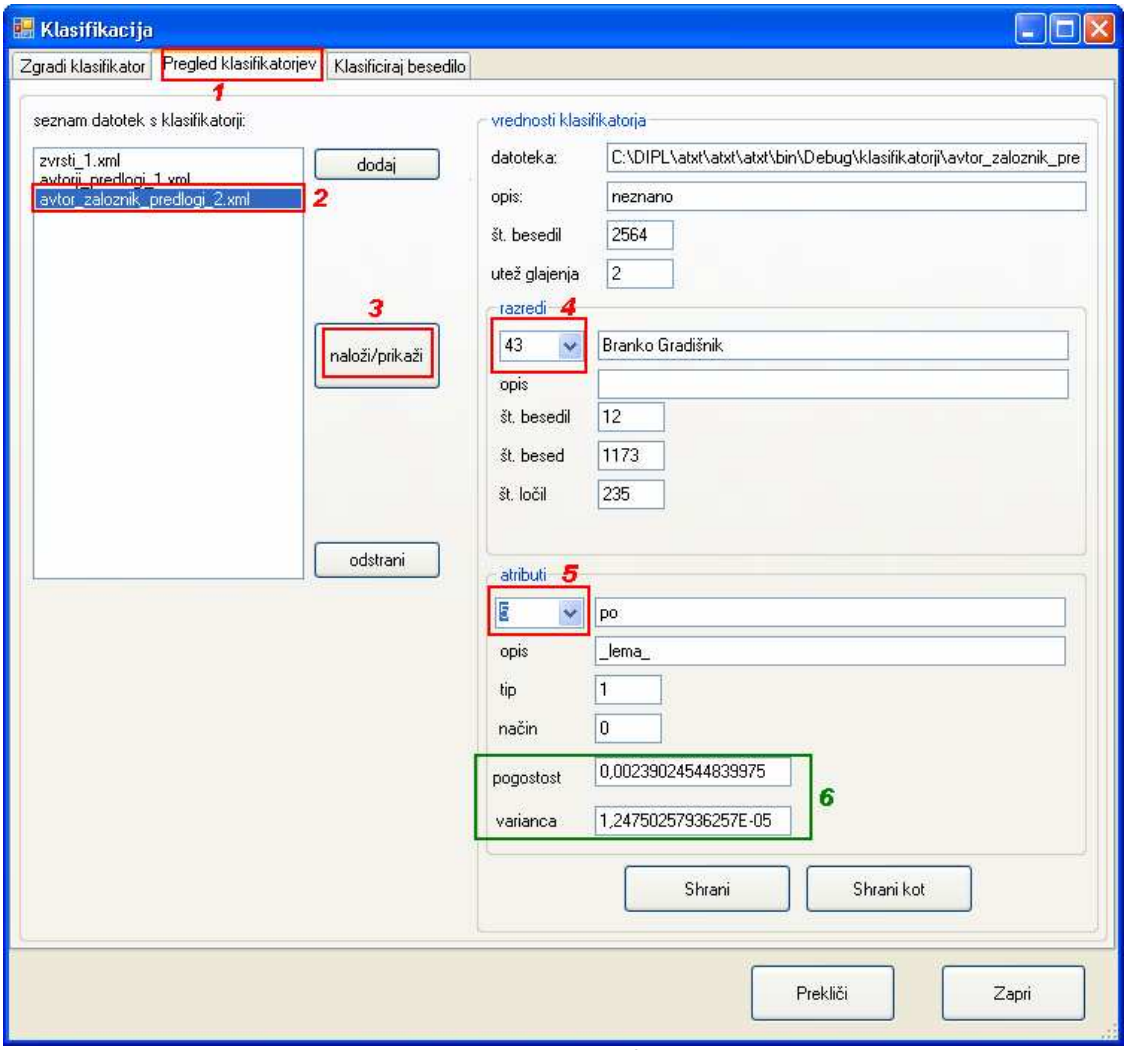
Slika 19. Gradnja klasifikatorja - izbira atributov (ločila).

A.8.1.6 Shranjevanje klasifikatorja v datoteko

Kliknemo *Shrani v datoteko* in določimo pot in ime xml datoteke, kamor se bodo shranili izračunani podatki klasifikatorja.

A.8.2 Pregled klasifikatorjev

V glavnem meniju izberemo *Orodja -> Klasifikacija (slogovna) -> Pregled klasifikatorjev* oziroma v oknu klasifikacije izberemo zavihek *Pregled klasifikatorjev* [slika 20, točka 1].



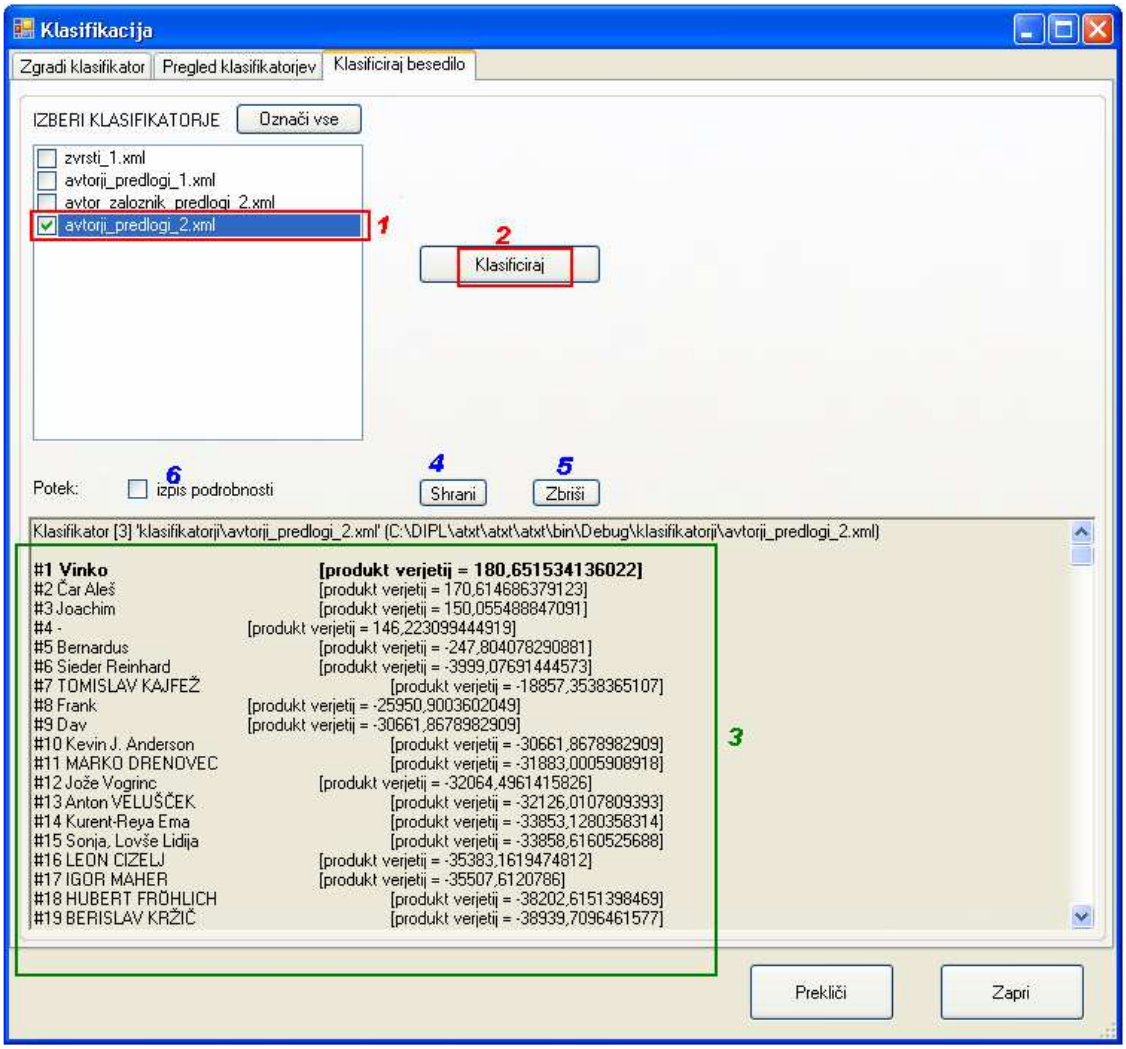
Slika 20. Pregled klasifikatorjev.

Na seznamu izberemo klasifikator in ga naložimo [slika 20, točki 2 in 3]. Pregledujemo lahko razrede in atribute, ki so razporejeni po zaporednih številkah [slika 20, točki 4 in 5].

Pregledujemo in spreminjamo lahko statistične parametre atributov za posamezne razrede [slika 20, točka 6]. Spremenjeni klasifikator lahko shranimo.

A.8.3 Klasifikacija

V glavnem meniju izberemo *Orodja -> Klasifikacija (slogovna) -> Klasifikacija besedila* ali kliknemo ikono v orodni vrstici. Izbran naj bo zavihek *Klasificiraj besedilo*.



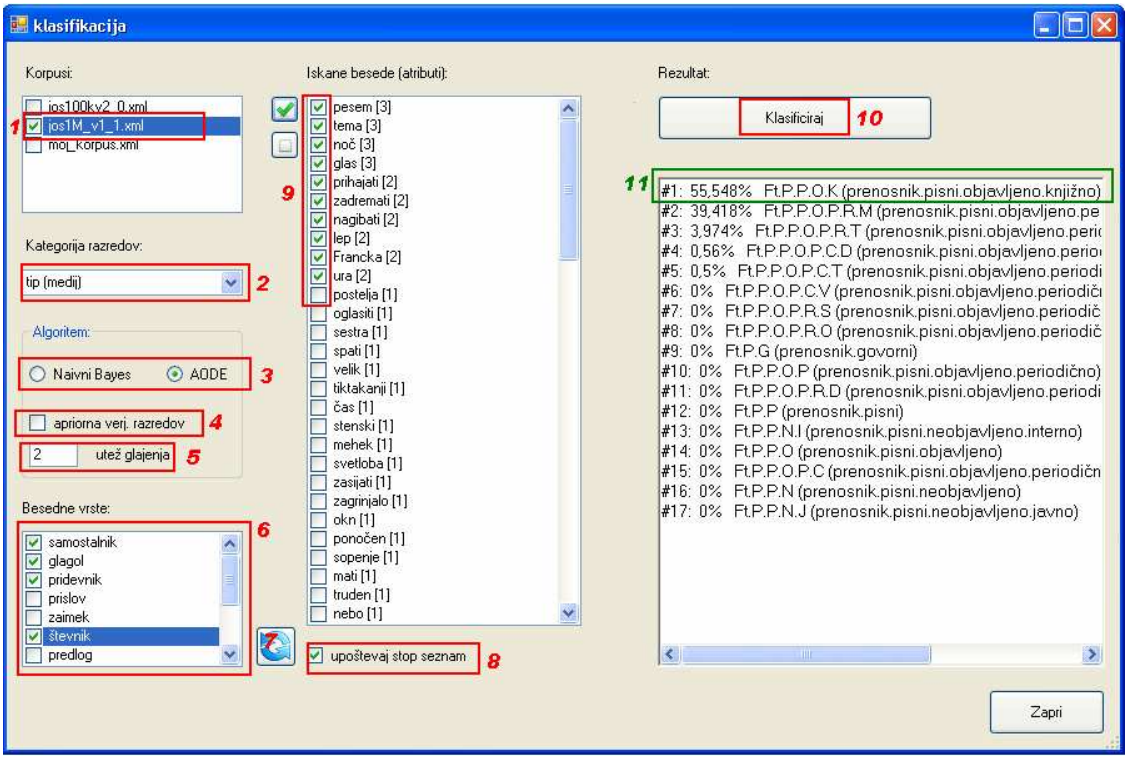
Slika 21. Uporaba klasifikatorjev.

Izberemo datoteke s klasifikatorji iz seznama [slika 21, točka 1]. Za začetek klasifikacije kliknemo na gumb *Klasificiraj* [slika 21, točka 2]. Nastavimo lahko izpis več podrobnosti med postopkom klasifikacije [slika 21, točka 6]. V spodnjem delu okna [slika 21, točka 3] se izpišejo rezultati klasifikacije. Rezultate lahko shranimo v datoteko ali jih pobrišemo [slika 21, točki 4 in 5].

Razred z najvišjim produktom gostote verjetij (angl. »likelihood«) je izbran kot rezultat klasifikacije (kot številka 1 pri izpisu rezultatov, odebeljeno besedilo).

A.9 Klasifikacija glede na vsebino besedila

V glavnem meniju izberemo *Orodja -> Klasifikacija (vsebinska) -> Klasifikacija besedila* ali kliknemo ikono v orodni vrstici.



Slika 22. Klasifikacija glede na besede, ki se pojavijo v besedilu.

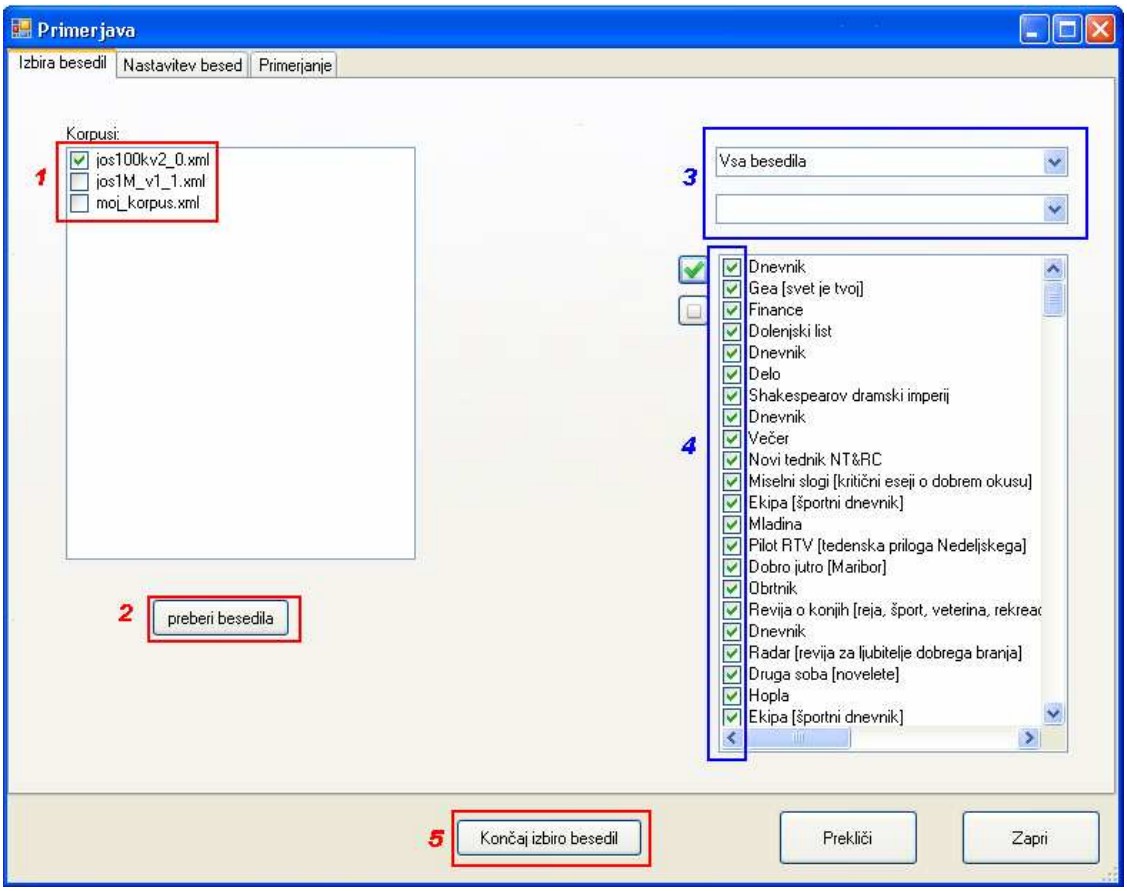
Postopek:

1. Izberemo korpus.
2. Izberemo kategorijo razredov: avtor, zvrst ali tip besedil.
3. Določimo algoritem: naivni Bayesov klasifikator ali AODE.
4. Določimo, ali se upošteva apriorna verjetnost razredov. Če sklepamo, da besedila v korpusu niso reprezentativna glede na pogostost v realnosti, potem ne izberemo te možnosti.
5. Nastavimo lahko utež glajenja.
6. Izberemo besedne vrste lem, ki se upoštevajo.
7. Osvežitev seznama z izbiro lem, ki se pojavijo v danem besedilu.
8. Izberemo, ali se upošteva seznam nepotrebnih besed (»stop words«).
9. Izberemo atribute med lemami, ki se pojavijo v danem besedilu.
10. Poženemo postopek klasifikacije
11. Izpis rezultatov. Produkti verjetij (»likelihood«) razredov so normalizirani in pretvorjeni v odstotke.

A.10 Primerjanje besedil s kosinusno razdaljo in gručenje

V glavnem meniju izberemo *Orodja -> Primerjava z besedili v korpusu* ali kliknemo ikono v orodni vrstici. Postopek primerjanja se izvede v treh korakih.

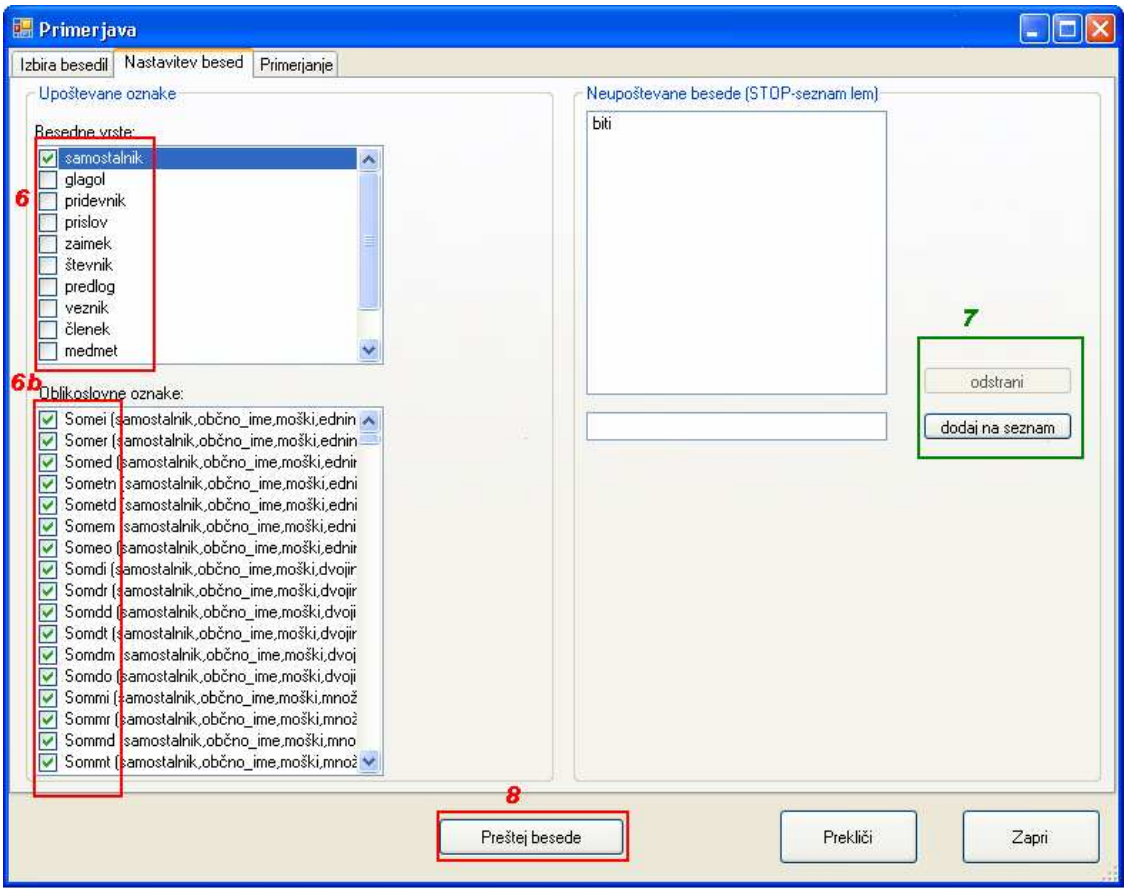
A.10.1 Izbira besedil za primerjavo



Slika 23. Izbira besedil za primerjavo.

1. Izberemo korpuse.
2. Program prebere besedila iz korpusov in izpiše njihove podatke [slika 23, okno na desni strani].
3. Seznam prikazanih besedil lahko filtriramo po kategorijah.
4. Določimo besedila, ki se bodo upoštevala za primerjavo.
5. Končamo izbiro besedil.

A.10.2 Izbira upoštevanih besed



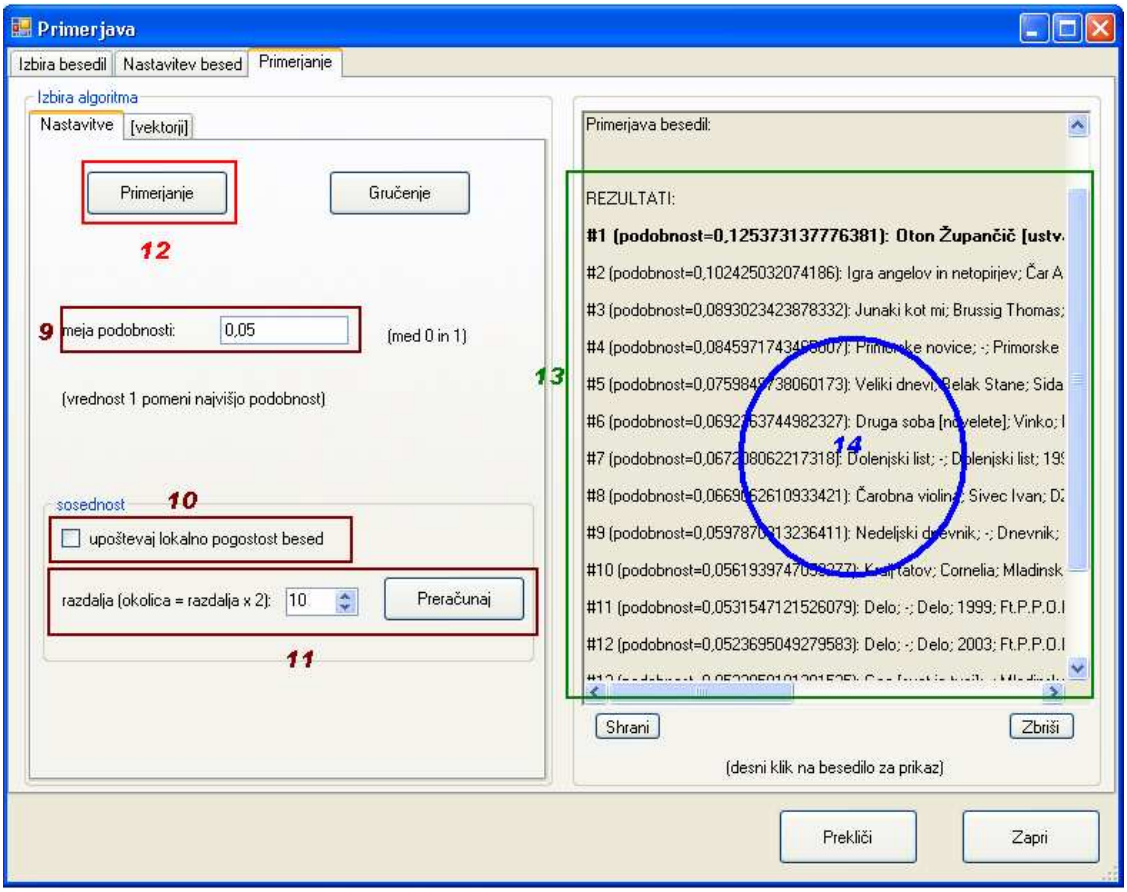
Slika 23. Izbira upoštevanih besed predstavitvenih vektorjev.

6. Izberemo besedne vrste upoštevanih besed. Podrobno lahko nastavimo tudi podoznake [slika 23, točka 6b].

7. Nastavimo seznam nepotrebnih besed (angl. »stop-list«). Na seznam lahko dodajamo leme ali jih iz njega odvzemamo.

8. S klikom na gumb *Preštej besede* se program loti štetja vseh besed, ki zadoščajo nastavljenim kriterijem, za vsa izbrana besedila.

A.10.3 Primerjava besedil



Slika 24. Primerjanje besedil s kosinusno razdaljo predstavitvenih vektorjev

9. Nastavimo mejo podobnosti (med 0 in 1, ker smo za podobnost uporabili kosinusno funkcijo).

10. Izberemo, ali se naj upošteva lokalna pogostost pojavljanja iste besede v določeni razdalji.

11. Nastavimo lahko okolico, kjer se upoštevajo ponovitve iste besede. Če nastavimo drugačno razdaljo, se mora vektor uteži preračunati.

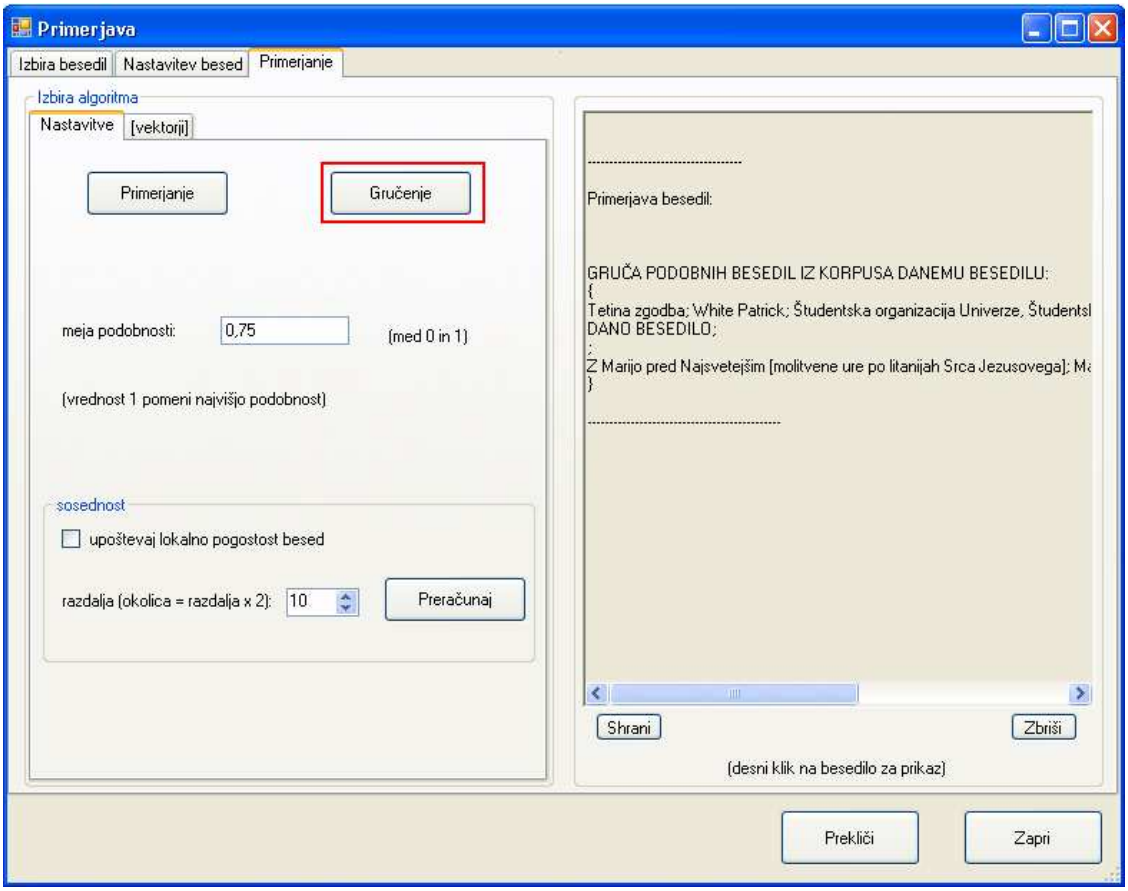
12. Poženemo primerjavo besedil.

13. Program izpiše rezultate.

14. Besedila, ki so našteta v izpisu rezultatov, si lahko ogledamo z desnim klikom na izbrano vrstico z opisom besedila.

A.10.4 Gručenje

Ponovimo enak postopek kot pri primerjanju [poglavja A.10.1 do A.10.3]. Namesto gumba za preverjanje kliknemo na *Gručenje*. Program izpiše gručo podobnih besedil našemu besedilu.



Slika 25. Gručenje.

Opomba:
Implementirani algoritem gručenja je osnovna oblika hierarhičnega gručenja in ni optimiziran. Pri velikih vektorjih (primerja se mnogo besed iz besedila) je zelo počasen in prostorsko požrešen. Pri dolgem besedilu, ki ga uvrščamo, ter velikem korpusu se pojavi veliko različnih besed, kar pomeni razraščanje vektorjev besedil. Zato je dobro izbrati take besedne vrste, ki jim pripada manj besed, ali pa uvrstiti mnogo lem v seznam nepotrebnih besed (»stop-list«). V našem primeru smo izbrali manjši korpus in le vezniške besede.

VIRI IN LITERATURA

[1] I. Kononenko, M. Robnik Šikonja. »Obdelava naravnega jezika« v knjigi *Inteligentni sistemi*, poglavje 11, Ljubljana, Založba FE in FRI, 2010

[2] Geoffrey I. Webb, Janice R. Boughton, Zhihai Wang. »Not So Naive Bayes: Aggregating One-Dependence Estimators«, *Machine Learning*, vol. 58, 5-24, 2005, dostopno na: http://www.springerlink.com/content/u8w306673m1p866k [31.08.2012]

[3] K. Polanec. »Strojno učenje«. Dostopno na: http://www.publikacije.dat.si/Article/Strojno-u--269-enje/66 [31.08.2012]

[4] Wikipedia: Document classification, http://en.wikipedia.org/wiki/Document_classification [31.08.2012]

[5] J.Brezovnik. »Programsko ogrodje za procesiranje besedil v naravnem jeziku«. Magistrsko delo, Univerza v Mariboru, Slovenija, 2009

[6] Wikipedia: Flesch–Kincaid readability test, http://en.wikipedia.org/wiki/Flesch-Kincaid_readability_test [30.08.2012]

[7] OdprtiTezaver - slovenski tezaver, http://www.tezaver.si [30.08.2012]

[8] Steven Bird, Ewan Klein, Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009, dostopno na: https://sites.google.com/site/naturallanguagetoolkit/book [31.08.2012]

[9] Wikipedia: Additive Smoothing, http://en.wikipedia.org/wiki/Additive_smoothing [31.08.2012]

[10] Wikipedia: Average one-dependence estimators, http://en.wikipedia.org/wiki/AODE [31.08.2012]

[11] Natural Language Toolkit, http://nltk.org [01.09.2012]

[12] Apache OpenNLP, http://opennlp.apache.org [01.09.2012]

[13] Oblikoslovni označevalnik za slovenski jezik, http://oznacevalnik.slovenscina.eu/Vsebine/Sl/SpletniServis/SpletniServis.aspx [01.09.2012]

[14] TEI: Text Encoding Initiative, http://www.tei-c.org/index.xml [01.09.2012]

[15] Projekt JOS: jezikovno označevanje slovenskega jezika, http://nl.ijs.si/jos [01.09.2012]

[16] JOS concordance service, http://nl.ijs.si/jos/cqp [01.09.2012]

[17] JOS ToLaTe text analyser, http://nl.ijs.si/jos/analyse [01.09.2012]

[18] FidaPLUS: korpus slovenskega jezika, http://www.fidaplus.net [01.09.2012]

[19] D. Fisher. »Izdelava slovenskega semantičnega leksikona z uporabo eno- in večjezičnih jezikovnih virov«. Doktorska disertacija, Univerza v Ljubljani, Ljubljana, 2009

[20] WordNet, http://wordnet.princeton.edu [01.09.2012]

[21] slowNet, http://lojze.lugos.si/~darja/slownet.html [01.09.2012]

[22] Wikipedija: Besedilni korpus, http://sl.wikipedia.org/wiki/Korpus_(jezikoslovje) [31.08.2012]

[23] MULTEXT-East Morphosyntactic Specifications (Version 4), http://nl.ijs.si/ME/V4/msd/html/msd-sl.html [31.08.2012]

[24] Nova beseda, http://bos.zrc-sazu.si/s_beseda.html [01.09.2012]

[25] Projekt »Sporazumevanje v slovenskem jeziku«, http://www.slovenscina.eu/Vsebine/Sl/Domov/Domov.aspx [01.09.2012]

[26] Wikivir, http://sl.wikisource.org/wiki/Glavna_stran [01.09.2012]

[27] Zbirka slovenskih leposlovnih besedil, http://lit.ijs.si/leposl.html [01.09.2012]

[28] Digitalna knjižnica Slovenije, http://www.dlib.si [01.09.2012]

[29] Wikipedija: Normalna porazdelitev, http://sl.wikipedia.org/wiki/Normalna_porazdelitev [31.08.2012]

[30] Wikipedia: Algorithms for calculating variance, http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance [31.08.2012]

[31] Wikipedia: Single-linkage clustering, http://en.wikipedia.org/wiki/Single-linkage_clustering [31.08.2012]