SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5672

Sustav za upravljanje i pretraživanje baze PDF dokumenata

Luka Čupić

Zagreb, lipanj 2018.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Područje analize i pretraživanja teksta neizbježno je u današnjem svijetu tehnologije. Od internetskih tražilica koje pretražuju enormne količine podataka baziranih na zadanom upitu, osobnih pomoćnika na pametnim mobitelima koji procesiraju izgovorene riječi pa sve do analize i prepoznavanja *spam* elektroničkih poruka. Kratki osvrt na ove te mnoge druge primjene ukazuju na nepobitnu činjenicu da je pretraživanje teksta...

2. Pregled područja

Korištenje računala za povrat informacija (engl. *information retrieval*) datira sve do četrdesetih godina dvadesetog stoljeća, daleko prije komercijalizacije računala, odnosno njihovog korištenja u osobne svrhe. Pogledamo li samo neke od relevantnih problema poput digitalizacije knjižnica i automatizacije knjižničnih poslova, statističku analizu tekstualnih podataka u svrhe X, procesiranje korisničkih upita kako bi se pronašli relevantni dokumenti, ... Očito je da je područje analize i pretraživanja teksta vrlo zastupljeno u današnjem digitalnom svijetu u kojem se količina informacija povećava za X posto svake godine (referenca na paper), očito je da je domena primjene vrlo široka te da su analiza i pretraživanje teksta praktički zastupljeni u svakom većem području koje iziskuje nekakvu vrstu analize i pretraživanja teksta odnosno povrata informacija.

3. Model dokumenata

3.1. Prikaz dokumenata

Prije svega, valja definirati što u kontekstu analize i pretraživanja teksta predstavlja dokument. Dokument neformalno možemo definirati kao kolekciju riječi. Ovakva kolekcija ne mora nužno biti skup, pošto dokument može imati više ponavljanja istih riječi, stoga na dokument možemo gledati upravo kao na poredanu kolekciju riječi u kojoj može biti ponavljanja. Ovakva definicija dokumenta bit će bitna u nastavku gdje se opisuje semantička sličnost dokumenata. Primjerice, neki dokument koji sadrži više pojavljivanja riječi "računalo", "algoritam" te "memorija" očito će biti relevantan u kontekstu dokumenata vezanih za računarsku znanost. Postavlja se pitanje kako tako definirane dokumente prikazati u računalu. Jedan (smisleni) pokušaj bio bi predstaviti riječi kao vektore, gdje bi svaki znak te riječi predstavljao jednu komponentu vektora. Ovakav model naziva se *Word2Vec* te služi za predstavljanje riječi iz nekog vokabulara na način da svaka riječ dobije odgovarajući položaj u više-dimenzijskom prostoru. Tako definiran model ima za posljedicu to da će semantički sličnije riječi biti bliže (odnosno da pripadajući im vektori međusobno zatvarati manji kut) i obrnuto. Ovakav se model često koristi u kontekstu semantičke analize riječi, primjerice u pronalaženju sličnih riječi, sinonima, antonima itd. No ipak, u kontekstu ovog rada fokus će imati semantička sličnost *dokumenata*, stoga će za predstavljanje dokumenata biti korišten tzv. model vreće riječi (engl. *bag of words model*). U modelu vreće riječi, tekst dokumenta predstavljen je multiskupom riječi. Multiskup jest proširenje klasičnog skupa u smislu da dozvoljava više pojavljivanja elemenata, odnosno u ovom kontekstu, riječi. Kod modela vreće riječi dakle nije bitna semantika samih dokumenata, pa čak niti poredak riječi, već je bitno samo koje se riječi pojavljuju u određenom dokumentu, odnosno koja je učestalost njihovog pojavljivanja. Primjer modela vreće riječi prikazan je u nastavku: Neka su zadani dokumenti d_1 = "Marko jako voli domaćice. Domaćice su dobre." te d_2 = "Marko voli domaćice i programiranje." Za ovako zadane dokumente,

prikazane su dobivene dobivene vreće riječi u JSON formatu:

$$BoW_1 = \{\text{"Marko":1, "jako":1, "voli":1, "domaćice":2, "su":1, "dobre":1}\}, \quad (3.1)$$

$$BoW_2 = \{\text{"Marko":1, "voli":1, "domaćice":2, "i":1, "programiranje":1}\}, \quad (3.2)$$

Kako bi se uopće moglo započeti sa semantičkom analizom dokumenata, potrebno je imati kolekciju dokumenata koji će se međusobno uspoređivati Korištenje modela vreće riječi zamišljeno je tako da se na početku iz svih dokumenata iz kolekcije dokumenata (u daljnjem tekstu: zbirka) izvade sve riječi te potom uklone nebitne riječi (o kojima će više riječi biti u poglavlju 6) a od preostalih se riječi izgradi vektor koji će *de-facto* predstavljati dokument. Ovakav prethodno opisani model često je korišten u području procesiranja prirodnog jezika te povrata informacija kako iz dokumenata tako iz drugih tekstualnih izvora.

U prethodnom primjeru prikazana je konstrukcija vreće riječi dvaju dokumenata. U tom primjeru, kao riječi koje se pojavljuju u vrećama uzete su sve riječi iz ulaznih dokumenata, što u praksi neće uvijek biti slučaj. Naime, u dokumentima se često znaju pronaći riječi koje nemaju nikakvo značenje za sam dokument. Takve su riječi primjerice zamjenice, pomoćni glagoli, veznici itd. Ovakve riječi nazivaju se zaustavne riječi (engl. *stop words*) te su to riječi koje su učestale u nekom jeziku te su stoga nebitne za sam postupak analize teksta, pošto na nikakav način ne doprinose sadržaju dokumenta. Zaustavne se riječi stoga izbacuju iz vreća riječi te se zadržavaju samo riječi "bitne" za semantiku dokumenata. Empirijski je pokazano da uklanjanje zaustavnih riječi ima pozitivan utjecaj na daljnju obradu dokumenata. Primjeri nekih zaustavnih riječi u hrvatskom jeziku su: *aha*, *nešto*, *okolo* te *zaboga*. Kako bi se dokumenti mogli predstaviti u obliku vreća riječi, potrebno je odrediti vokabular—skup svih riječi koje se nalaze u svim dokumentima promatrane zbirke. Poznavanje vokabulara ključno je za semantičku sličnost dokumenata, što će i biti pokazano kasnije. Iz dobivenog vokabulara potrebno je na početku obrade ukloniti zaustavne riječi iz već opisanih razloga. Osim zaustavnih riječi, dodatna obrada teksta može se obaviti tzv. *stemanjem* (engl. *stemming*). Ova metoda ima zadaću svesti riječi na njihov kanonski oblik, kako bi riječi istog korijena bile svedene na istu riječ. Svođenje na kanonski oblik ne znači nužno i svođenje na morfološki korijen riječi. Primjerice, riječi "mačka", "mačkama" te "mačke" bile bi svedene na "mačk". Nebitno je dakle je li riječ napisana u jednini ili množini ili pak u kojem je padežu već je bitan samo njezin kanonski oblik. Nakon stvaranja vokabulara te predobrade dokumenata (izbacivanje zaustavnih riječi, stemanje) sljedeći korak jest predstavljanje dokumenata. Radi praktičnosti, najčešća metoda predstavljanja dokumenata je uz pomoć vektora. Najjednostavnija metoda vektorskog

predstavljanja dokumenata jest binarna: za svaku riječ iz vokabulara naprosto se provjeri nalazi li se u danom dokumentu te ukoliko se nalazi, odgovarajuća komponenta vektora (indeks riječi u vokabularu) biti će 1, a u suprotnom 0. Primjerice, za prethodno definirane dokumente d_1 i d_2, vokabular će nakon uklanjanja svih nebitnih znakova i stop riječi biti: V = {"Marko", "jako", "voli", "domaćice", "dobre", "programiranje"}. Valja primijetiti kako su riječi "i" i "su" izbačene iz vokabulara. Koristeći binarnu metodu reprezentacije dokumenata, odgovarajući vektori će iznositi

$$d_1 = [1, 1, 1, 1, 1, 0], \tag{3.3}$$

$$d_2 = [1, 0, 1, 1, 0, 1] \tag{3.4}$$

zbog toga što prvi dokument ne sadrži riječ "programiranje" (zadnja komponenta) dok drugi dokument ne sadrži riječi "jako" (druga komponenta) te "dobre" (predzadnja komponenta). Nadograđujući se na prethodnu metodu, dolazi se do frekvencijskog prikaza vektora. Umjesto obične binarne reprezentacije u kojoj se pamti samo nalazi li se riječ u dokumentu ili ne, u frekvencijskom prikazu pamti se i koliko se puta određena riječ pojavljuje u dokumentu; komponente vektora zapravo su frekvencija (tj. broj) pojavljivanja određene riječi u dokumentu. Gledajući isti vokabular i dokumente kao u prethodnom primjeru, novi vektori će u ovom slučaju iznositi:

$$d_1 = [1, 1, 1, 2, 1, 0] \tag{3.5}$$

$$d_2 = [1, 0, 1, 1, 0, 1] \tag{3.6}$$

Jedina razlika u odnosu na prethodni primjer jest četvrta komponenta prvog vektora koja ukazuje na to da se riječ "domaćica" u prvom dokumentu pojavljuje dvaput. Naposlijetku dolazimo i do najčešće metode vektorskog prikaza dokumenata u kontekstu analize i pretraživanja teksta — TF-IDF (engl. *term frequency–inverse document frequency*). Ova metoda zasniva se na dvije intuitivne pretpostavke:

- riječ je važnija za semantiku dokumenta što se češće u njemu pojavljuje (TF komponenta)
- riječ je manje važna za semantiku dokumenta što se češće pojavljuje u drugim dokumentima (IDF komponenta)

TF i IDF dakle predstavljaju dvije komponente vektora kojima će se predstavljati dokumenti. Prva komponenta je već spomenuta frekvencija pojavljivanja riječi *w* u dokumentu *d*, odnosno $f_{w,\,d}$, dok je druga komponenta obrnuta frekvencija pojavljivanja

riječi u cijeloj zbirci. Za riječ *w* i dokument *d*, TF i IDF komponene računaju se na sljedeći način:

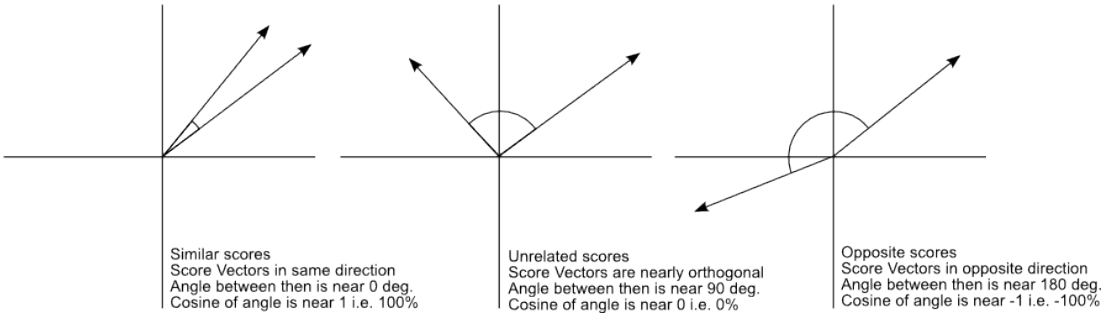
$$\mathrm{tf}(t, d) = f_{t,d} \tag{3.7}$$

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{3.8}$$

Izraz za TF komponentu je intuitivan i trivijalan dok izraz za IDF komponentu zahtjeva kratki osvrt: riječ će biti bitnija za neki dokument što se rijeđe pojavljuje u ostalim dokumentima zbirke, odnosno drugim riječima: riječ će biti manje bitna za neki dokument što se češće pojavljuje u drugim dokumentima. Ovo ima smisla zato što neke riječi mogu biti česte u dokumentima čisto zbog same prirode jezika (kao što je ranije pokazano u slučaju zamjenica, veznika itd.) pa stoga ima smisla takve riječi manje uzimati u obzir prilikom računanja relevantnosti riječi. Dakle: što je riječ češća u ostalim dokumentima, IDF vrijednost se smanjuje te riječ postaje manje bitna za neki dokument. Naposlijetku, cijeli se omjer logaritamski skalira kako bi se u smanjio utjecaj velikog i/ili malog broja dokumenata koji sadrže određenu riječ. U nastavku je prikazan primjer izračuna TF-IDF vektora za prethodno prikazane dokumente d_1 i d_2: (Je li ovo bitno ?!)

3.2. Semantička sličnost dokumenata

Nakon izgrađene vektorske reprezentacije svih dokumenata zbirke, sljedeći korak jest samo uspoređivanje dokumenata. U sklopu ovog završnog rada, uspoređivanja dokumenata ostvaruje se na dva semantički različita načina: uspoređivanje korisničkog upita (engl. *user input*, *query*) sa zbirkom dokumenata odnosno uspoređivanje pojedinog dokumenta sa zbirkom dokumenata. Ova dva, naizgled različita problema, zapravo se svode na jedan: uspoređivanje kolekcije riječi sa zbirkom dokumenata. Ideja je dakle sljedeća: gleda se koliko riječi (bilo iz korisničkog upita, bilo iz dokumenta, u daljnjem tekstu: ulazni vektor) iz ulaznog vektora odgovara riječima iz vokabulara, tj. koliko riječi iz ulaznog vektora odgovaraju riječima iz pojedinih dokumenata u zbirci. Što je veća korespondencija određenog ulaznog vektora s vektorom pojedinog dokumenta (tj. što više riječi dijele zajedno), to kažemo da su ta dva dokumenta sličnija. Primjerice, ukoliko se u zbirci nalazi dokument o Zvjezdanim ratovima, a kao ulazni vektor dovedemo frazu poput "May the Force be with you", taj ulazni vektor i taj dokument imati će relativno visoku mjeru sličnosti. Ovo dovodi do sljedeće definicije:
Mjeru sličnosti dokumenata (engl. *document similarity*) definiramo kao vrijednost na



Slika 3.1: Prikaz sličnosti dvaju vektora u 2D koordinatnom sustavu

skupu pozitivnih [1] realnih brojeva, koja ukazuje na to koliko su dva dokumenta slična — što je brojka veća, dokumenti su sličniji. U nastavku će se razmotriti nekoliko metoda za izračun mjere sličnosti dokumenata.

3.2.1. Metoda kosinusne sličnosti

Kako su dokumenti zapravo predstavljeni vektorima u više-dimenzijskom prostoru, nad njima (odnosno njihovim vektorima), možemo primijenjivati operacije linearne algebre, odnosno vektorske operacije. Počevši od definicije skalarnog umnoška dvaju vektora

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta, \tag{3.9}$$

dolazimo do mjere kosinusne sličnosti dvaju vektora (dokumenata):

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{3.10}$$

Naime, sličnost dvaju dokumenata u ovom kontekstu prikazujemo kao vrijednost kosinusa kuta između njihovih vektora. Što su dokumenti sličniji, kosinus kuta biti će bliži jedinici, odnosno što su dokumenti različitiji, kosinus kuta biti će bliži nuli. Intuicija ovoga je sljedeća: ukoliko radi jednostavnosti zamislimo da vektori imaju samo dvije dimenzije, tada će sličnost dokumenata koje predstavljaju biti to veća što su oni "bliži" u 2D koordinatnom sustavu, tj. što je kosinus kuta među njima manji. Vrijedi i da će dokumenti biti manje slični što je kosinus kuta njihovih vektora veći. Grafička interpretacija prikazana je na slici 3.1.

Ovo saznanje o kosinusnoj mjeri sličnosti dokumenata možemo iskoristiti u izgradnji sljedećeg modela: Neka je v_{d_i} vektorska reprezentacija (binarna, frekvencijska ili

[1] Postoji i definicija koja mjeru sličnosti definira nad cijelim skupom realnih brojeva, no u kontekstu ovog rada definicija nad pozitivnim brojevima biti će sasvim dostatna

TF-IDF) dokumenta d_i. Tada se sličnost dvaju dokumenata mjeri kao:

$$\text{similarity}(d_i, d_j) = \frac{v_{d_i} \cdot v_{d_j}}{||v_{d_i}|| \cdot ||v_{d_j}||} \tag{3.11}$$

Pošto se skalarni produkt dva vektora svodi na sumu umnožaka pripadajućih komponenti (prva s prvom, druga s drugom itd.), ovo intuitivno možemo zamisliti tako da naprosto zbrajamo korespondencije odgovarajućih riječi te na kraju dijelimo sve s umnoškom njihovih normi kako bi normalizirali rezultat na interval [0, 1]. Ako se riječ nalazi u oba dokumenta, tada će taj umnožak biti pozitivan te će se pridodati mjeri sličnosti, odnosno povećati ju. Ako neka riječ ne postoji u dokumentu, tada će taj umnožak biti nula pa će automatski sličnost biti manja. Iz činjenice da je mjera sličnosti dokumenata za metodu kosinusne sličnosti definirana na intervalu [0, 1] slijedi da će dva dokumenta biti posve različita (tj. neće imati nikakvih sličnosti) ako je njihova mjera sličnosti jednaka nuli, odnosno da će dva dokumenta biti jednaka ako im je mjera sličnosti jednaka 1.

3.2.2. Metoda Okapi BM25

Za razliku od metode kosinusne sličnosti, BM25 je funkcija rangiranja koja ima za zadaću direktno rangirati dokumente po relevantnosti određenom korisničkom upitu. Iako na prvi pogled ove dvije metode izgledaju različito, zapravo se svode na istu stvar pošto je dokument zapravo samo poopćeni oblik korisnikovog upita (više o tome u poglavlju 6). Za dokument Q, koji sadrži riječi q_1,...,q_n, BM25 mjera sličnosti nekog dokumenta D iz zbirke računa se kao:

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(((1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}, \tag{3.12}$$

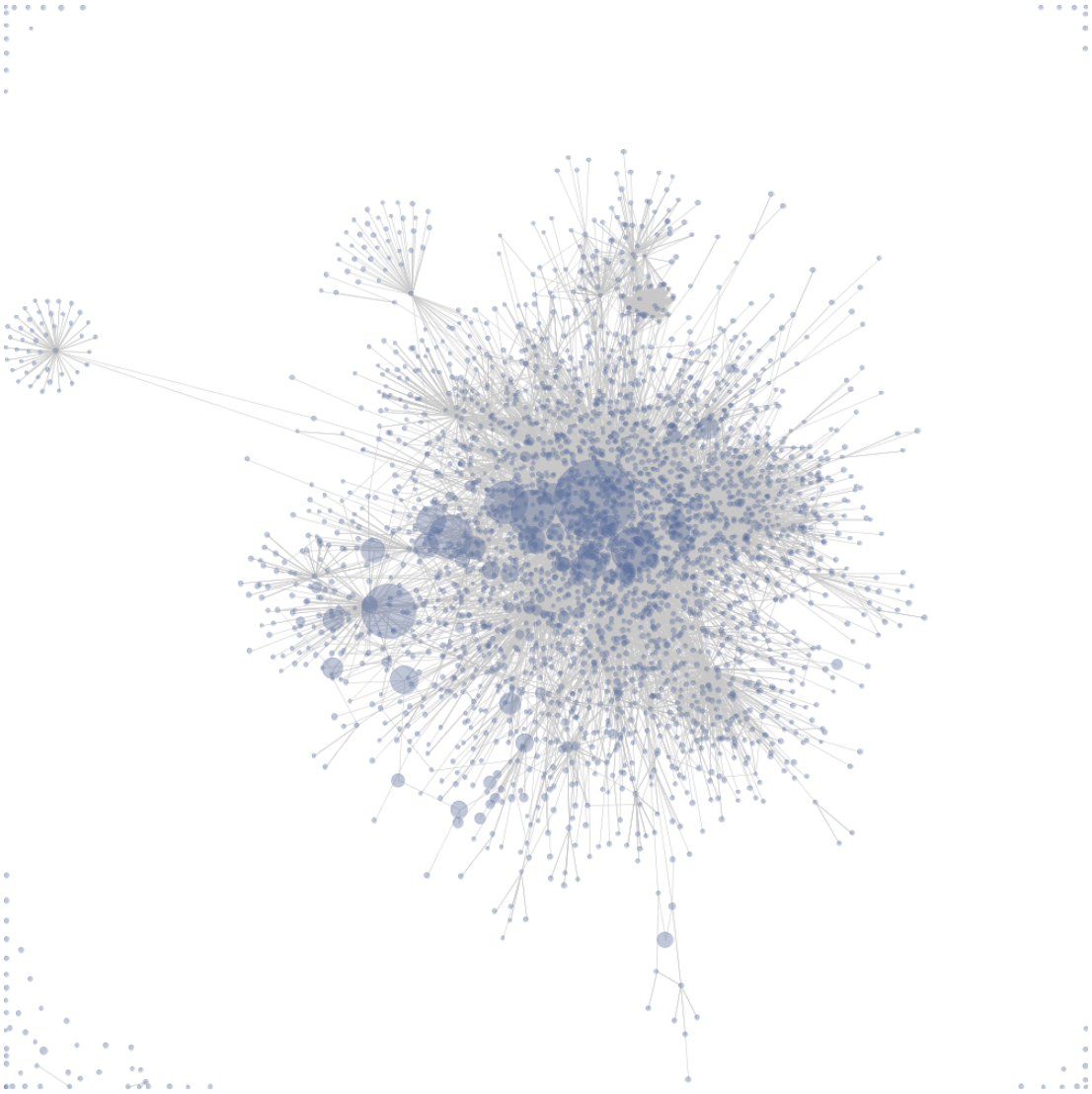
gdje je $f(q_i, D)$ frekvencija od q_i u dokumentu D, $|D|$ je broj riječi u dokumentu D, a avgdl je prosječan broj riječi u dokumentima iz zbirke. k_1 i b slobodni su parametri koji se uglavnom uzimaju kao $k_1 \in [1.2, 2.0]$ te $b = 0.75$. IDF(q_i) je IDF vrijednost komponente q_i. Analizirajući prethodnu formulu, može se zaključiti kako metoda BM25 nema zatvoreni interval sličnosti, u odnosu na metodu kosinusne sličnosti za koju se sličnost definira na intervalu [0, 1]. Naime, koristeći metodu BM25 jedino što se može zaključiti o odnosu ulaznog vektora i pojedinog dokumenta zbirke jest kakva je mjera njihove sličnosti u odnosu da mjeru sličnosti istog ulaznog vektora i nekog drugog dokumenta iz zbirke. Dakle, pošto metoda BM25 nema ograničeni interval za mjeru sličnosti, jedina njezina svrha u ovom kontekstu jest rangiranje dokumenata po sličnosti. Ovaj će nedostatak metode BM25 doći do izražaja u poglavlju 6.

4. Prikaz dokumenata u 2D koordinatnom sustavu

Kako prethodno navedene metode ispituju sličnost različitih dokumenata, sljedeći prirodan korak bio bi vizualizacija sličnosti dobivene među dokumentima. Ovdje međutim, nastaje jedan problem. Naime, dokumenti čija se međusobna sličnost želi prikazati grafički, predstavljeni su vektorima sačinjenim od onoliko komponenata kolika je veličina vokabulara. Uzmemo li kao primjer prosječnu duljinu znanstvenog rada koja je po PAPER tipično između 3.000 i 10.000 riječi, to bi značilo da se i veličina vokabulara takve zbirke dokumenata također mjeri u tisućama riječi. Pošto je magnituda svakog od vektora (tj. broj komponenata) upravo veličina vokabulara, to bi značilo da svaki vektor ima tisuće komponenata koje je nemoguće prikazati u 2D ili 3D koordinatnom sustavu u svrhu vizualizacije sličnosti dokumenata. Tom se problemu u sklopu ovog rada doskače silom usmjerenim crtanjem grafova (engl. *Force-directed graph drawing*).

4.1. Silom usmjereno crtanje grafova

Silom usmjereno crtanje grafova jedna je od metoda crtanja grafova koja se oslanja na simuliranje fizikalne pojave privlačnih i odbojnih sila među česticama. Naime, čvorovi grafa predstavljeni su metalnim prstenovima dok su bridovi predstavljeni oprugama. Opruge koja spajaju prstenove imaju ulogu privlačne elastične sile (Hookeov zakon), dok je odbojna sila zapravo električna sila između prstenova. Algoritam funkcionira tako da se u svakom koraku za svaki čvor odredi rezultantna sila prema svim ostalim čvorovima te se čvor pomiče u tom smjeru za određeni korak. Ovaj se postupak iterativno ponavlja te je cilj algoritma minimizirati ukupnu energiju sustava što će se dogoditi kada se privlačne i odbojne sile svih čvorova izjednače, odnosno kada algoritam odradi maksimalan broj koraka (koji se zadaje kao parametar algoritma).



Slika 4.1: Primjer grafa dobivenog silom usmjerenim crtanjem grafova

4.2. Grupiranje dokumenata

Jedna od često korištenih metoda u kontekstu analize i pretraživanja teksta jest grupiranje dokumenata (engl. *document clustering*). Cilj grupiranja jest izdvojiti dokumente neke zbirke u grupe tako da su dokumenti u jednoj grupi na neki način međusobno slični. Primjer jedne grupe dokumenata bili bi dokumenti o nogometu, košarci i odbojci pošto se sva tri dokumenta tiču sportskih aktivnosti. Kako bi se dokumenti mogli svrstati u grupe, potrebno je iskoristiti neki od algoritama za grupiranje. Jedan takav algoritam jest grupiranje k-sredina (engl. *k-means clustering*).

4.2.1. Grupiranje k-sredina

Grupiranje k-sredina je metoda koja spada u nenadzirano učenje (engl. *unsupervised learning*) a koja se koristi kada podaci nisu označeni—u kontekstu grupiranja dokumenata to bi značilo da za dokumente nisu unaprijed poznate grupe kojima ti dokumenti pripadaju. Cilj algoritma grupiranja k-sredina jest napraviti upravo to: odvojiti postojeće podatke u *k* grupa (odnosno pronaći iste), po čemu je algoritam i dobio naziv. Algoritam iterativno svakoj točki dodjeljuje grupu ovisno o sličnostima u odnosu na sve ostale točke (u kontekstu grupiranja dokumenata, ta sličnost je upravo mjera sličnosti dokumenata). Ulaz u algoritam jesu podaci (engl. *dataset*) te parametar *k*. Na početku se slučajnim mehanizmom odabere *k* grupa nakon čega slijedi ponavljanje sljedeća dva koraka:

1. **Dodjela grupa podacima.** U svakom koraku, za svaki podatak odredi se kvadratna euklidska udaljenost do najbliže grupe (centroida). Formalno, potrebno je dakle pronaći grupu c_i takvu da vrijedi:

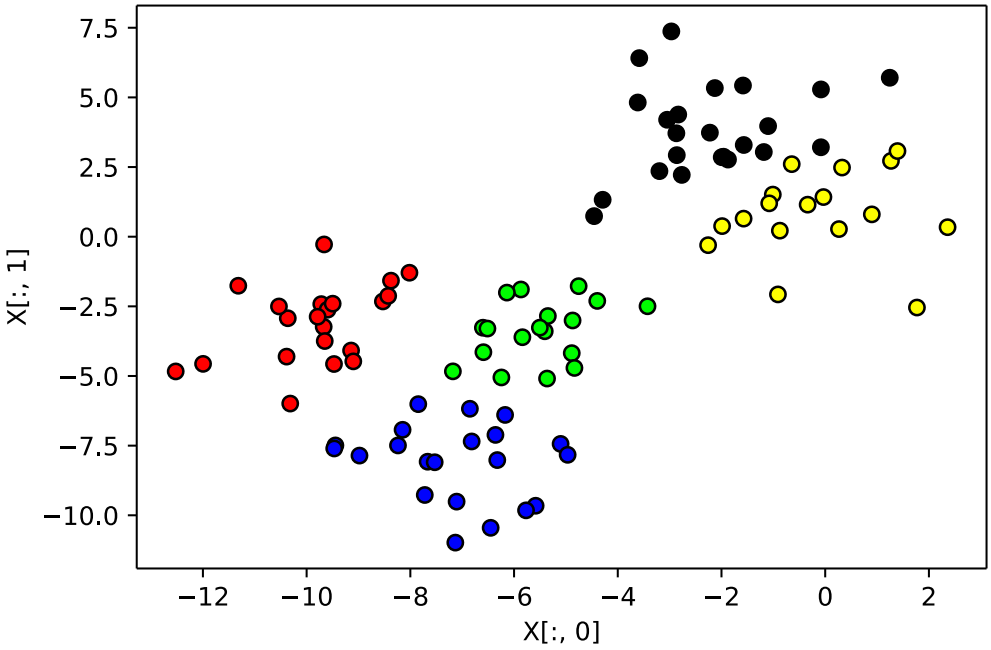
$$\arg\min_{c_i \in C} dist(c_i, x)^2 \tag{4.1}$$

   gdje je $dist(c_i, x)$ već spomenuta euklidska udaljenost točke x do centroida c_i.

2. **Ažuriranje centroida.** Za svaki od centroida, izračuna se aritmetička sredina svih točaka koje su u prethodnom koraku bile dodijeljene tom centroidu.

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i \tag{4.2}$$

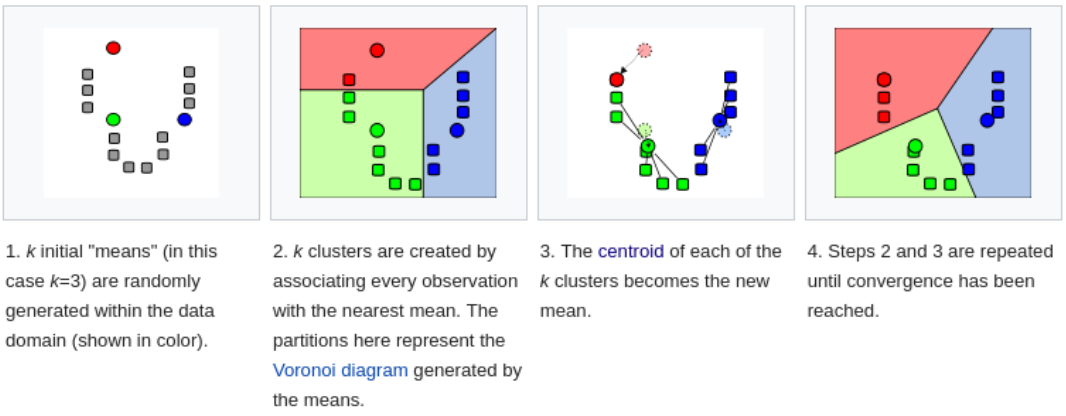
Algoritam dakle iterira kroz opisane korake dok nijedna točka ne promijeni grupu u kojoj se nalazi, odnosno dok ne bude ispunjen neki od uvjeta konvergencije (npr. dosezanje maksimalnog broja iteracija). Demonstracija algoritma prikazana je na slici 4.3.



Slika 4.2: Primjer grupiranja podataka koristeći algoritam k-sredina

4.3. Primjena na prikaz dokumenata

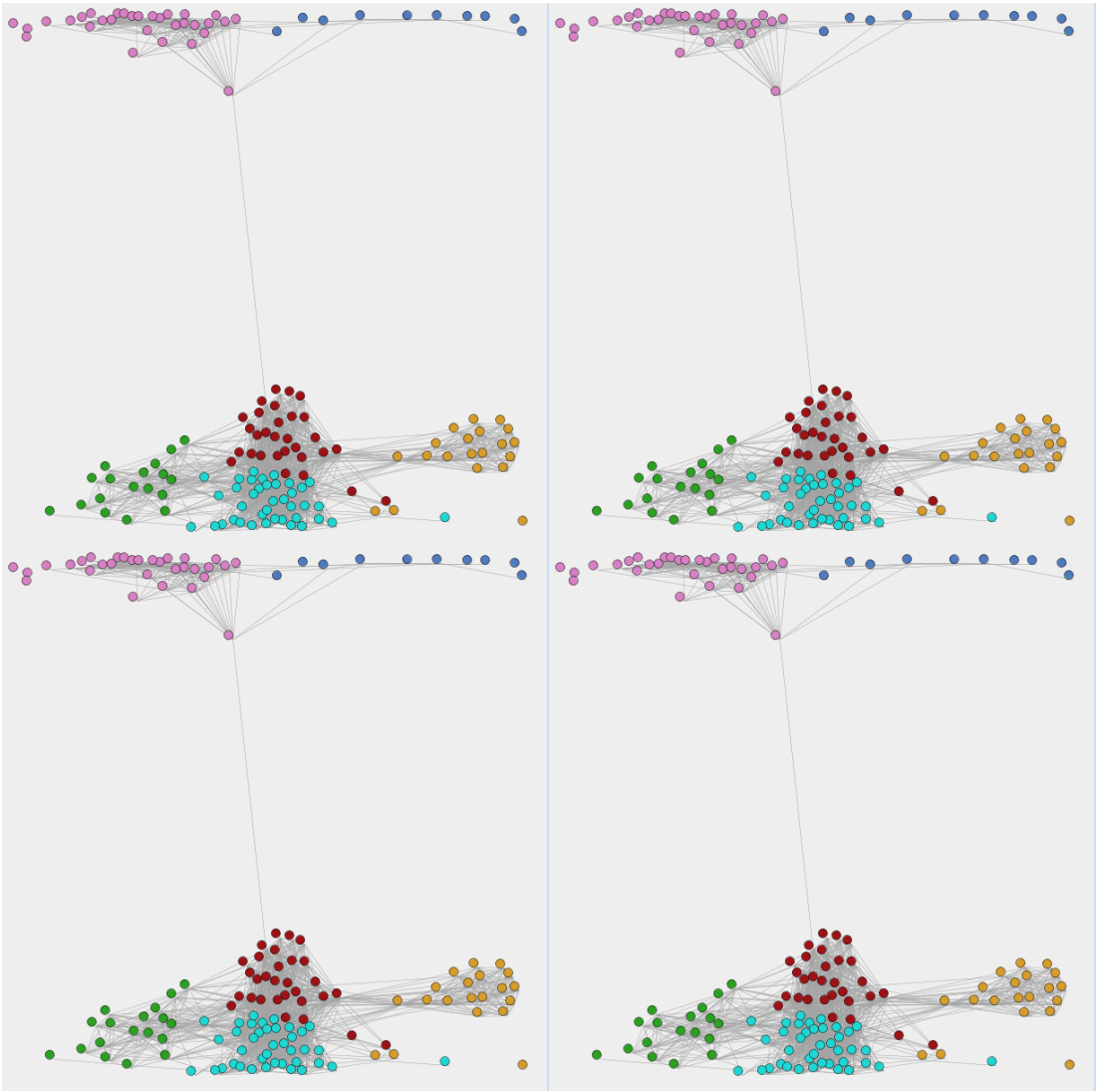
Prethodno definirane metode (silom usmjereno crtanje grafova te grupiranje k-sredina) mogu se iskoristiti upravo za prikaz i grupiranje dokumenata, odnosno njihovih međusobnih sličnosti u 2D koordinatnom sustavu. Prije početka ijednog od algoritama, svi dokumenti i pripadajući im vektori moraju biti učitani, odnosno inicijalizirani. Nadalje, za svaki par dokumenata d_i i d_j, $i \neq j$, izračuna se njihova sličnost (koristeći neku od metoda prikazanih u potpoglavlju 3.2) te se čvorovi (dokumenti) i bridovi (slič-



Slika 4.3: Demonstracija algoritma grupiranja k-sredina

nosti) predaju algoritmu koji odsimulira postupak opisan u potpoglavlju 4.1. Nakon što je algoritam završio sa simulacijom, postupak prestaje i prikazuje se nacrtani graf. Nad tako nacrtanim grafom dalje se može primijeniti algoritam grupiranja k-sredina koji prvo procjenjuje hiperparametar k uz pomoć već opisane heuristike te nakon toga provodi postupak grupiranja dokumenata. Nakon što se oba algoritma izvrše, dobiveni rezultat jest upravo prikaz dokumenata u 2D koordinatnom sustavu s naznačenim grupama koje predstavljaju aproksimaciju (broja) kategorija dokumenata iz zbirke.

Primjena algoritma k-sredina na grupiranje dokumenata jest sljedeća: grupirati dokumente u k grupa na način da se svakom dokumentu—točki u 2D prostoru koja ga predstavlja—dodijeli grupa do čijeg je centra ta točka najbliža. Algoritam započinje tako da slučajnim mehanizmom odabere k grupa te dodijeli dokumente u najbliže im grupe. Nakon inicijalne dodjele u grupe, računa se novih k grupa te se postupak iterativno ponavlja do konvergencije. Nakon završenog algoritma, svaki će se dokument nalaziti u najbližoj mu grupi, zajedno s ostalim dokumentima koji su mu najsličniji. Nažalost, grupiranje dokumenta u ovome kontekstu nije izravno moguće zbog toga što algoritam apriorno (lat. *a priori*) nema informaciju o broju grupa dokumenata iz zbirke. Razlog tome jest sama priroda problema koji se rješava, a to je da su na početku nepoznate grupe dokumenata (kao i njihov broj), odnosno jedini podaci dostupni programu su sami dokumenti. No ipak, broj grupa zbirke, k, ipak se može procijeniti određenim heuristikama kao što će biti pokazano u potpoglavlju 6.8.



Slika 4.4: Rezultat algoritma grupiranja k-sredina za *k=2* (gore lijevo), *k=6* (gore desno), *k=10* (dolje lijevo), *k=20* (dolje desno)

5. Isprobavanje metoda...

Jedan od ciljeva ovog rada jest istražiti već ranije spomenute metode analize i pretraživanja teksta.

Prilikom istraživanja različitih metoda rangiranja za potrebe ovog rada, metode koje su se pokazale najzanimljivijima (a koje su svejedno poprilično bazične, u smislu da ne iziskuju kompleksnije alate poput strojnog učenja) su metoda kosinusne sličnosti te metoda Okapi BM25.

6. Implementacija i rezultati

Programska implementacija ovog završnog rada napisana je u programskom jeziku Java koji je zbog svoje objektne metodologije, nativne podrške apstraktnih kolekcija podataka te postojana podrška raznih vanjskih biblioteka idealan za implementaciju problema iz domene analize i pretraživanja teksta. Korištene biblioteke su Apache PDFBox za parsiranje PDF dokumenata, Apache Commons Math za proračun k-sredina, Apache Digest Utils za izračun MD5 *hash* vrijednosti te Jung biblioteke za proračun te vizualizaciju grafova.

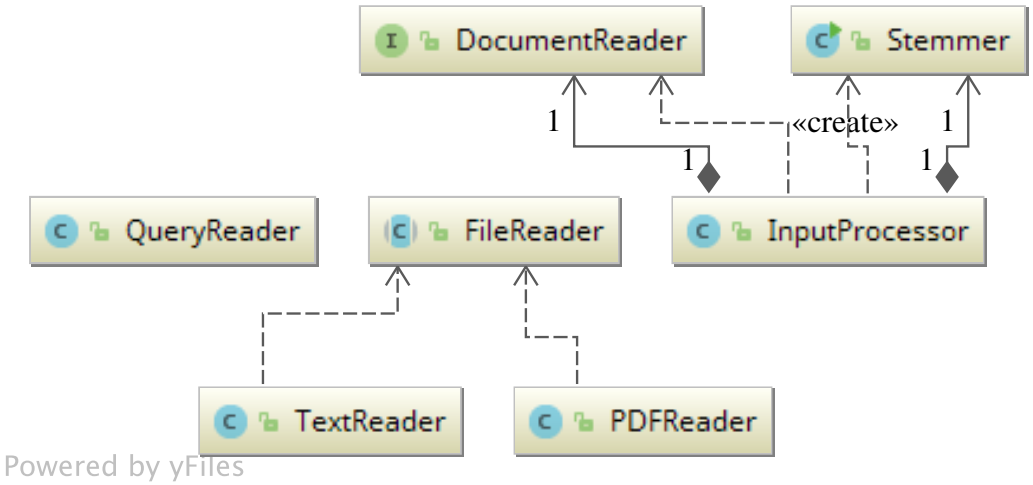
Kao što je već ranije spomenuto, pročitani dokumenti reprezentirani su vektorima obzirom da je to jedan od najjednostavnijih i najefikasnijih način prikaza dokumenata. Naime, u memoriji se na taj način ne trebaju eksplicitno spremati riječi za svaki dokument već se mogu spremiti samo brojke koje govore koliko je dotična riječ relevantna za dokument. Kako se u sklopu ovog rada koristi TF-IDF reprezentacija dokumenata, to znači da se za svaki dokument treba izračunati njegova TF-IDF (vektorska) reprezentacija. Prije samog izračuna komponenata TF-IDF vektora, potrebno je pročitati PDF dokumenti smještene na disku. Pošto su PDF dokumenti zapravo binarne datoteke, po svojoj strukturi nisu trivijalno parsabilni, što znači da ih nije moguće pročitati odnosno dekodirati na jednostavan način kao što je to moguće s primjerice tekstualnim datotekama. Zbog toga se za njihovo čitanje, odnosno parsiranje koristi vanjska biblioteka Apache PDFBox koja iz zadanog PDF dokumenta ekstrahira Unicode znakove koje može pročitati te ih vrati kao rezultat. Tako dobiveni tekst dodatno se obrađuje pri čemu se uklanjaju bilo kakvi interpunkcijski znakovi, dijakritici, brojke, odnosno svi znakovi koji nisu mala ili velika slova engleske abecede. Ovakav postupak nužan je kako bi se što više smanjio utjecaj nebitnih znakova na točnost uspoređivanja dokumenata. Jednom kada je tekst isfiltriran, nad njim se provodi daljnja predobrada koja: 1) uklanja iz teksta stop-riječi te 2) vrši stemanje nad dobivenim riječima dokumenata. Obje metode opisane su u potpoglavlju 3.1. Nakon završene predobrade teksta, stvaraju se vokabular, vektori (za reprezentaciju dokumenata) te nekoliko dodatnih pomoćnih struktura podataka. Nakon ovog koraka vrši se još i izračun sličnosti

dokumenata kako bi jednom izračunati podaci bili spremljeni za ponovno korištenje bez da se moraju svaki puta iznova računati.

6.1. Čitanje i obrada riječi

Programsko rješenje ovog završnog rada nudi korisniku interaktivan način pretraživanja postojeće zbirke dokumenata postavljanjem upita kroz grafičko korisničko sučelje. Naime, nakon odabrane putanje do zbirke dokumenata, korisnik postavlja upit te program pretražuje zbirku dokumenata i korisniku prikazuje dokumente sortirane po relevantnosti korisničkom upitu. Korisnički se upit procesira kao što je već ranije spomenuto u poglavlju 3: zanemaruju se svi znakovi koji nisu slova engleske abecede, uklanjaju se stop riječi, provodi se stemanje te se nakon toga korisnički unos procesira. Naime, za svaku riječ provodi se metoda kosinisne sličnosti odnosno BM25 te se od korisničkog unosa izgradi vektor koji taj unos predstavlja u *n*-dimenzijskom koordinatnom sustavu, gdje *n* predstavlja veličinu (broj riječi) vokabulara. Nakon izračuna sličnosti s dokumentima zbirke, program korisniku prikazuje popis svih relevantnih dokumenata, zajedno s odgovarajućim koeficijentima sličnosti.

Za procesiranje korisničkog upita (odnosno bilo kakvog unosa od strane korisnika, kao što će biti pokazano u iduća dva potpoglavlja) koristi se razred *InputProcessor* iz paketa *hr.fer.zemris.zavrsni.input*. Ovaj razred koristi se kad god treba pročitati tekst zadan od strane korisnika: bilo to kroz korisnički upit, bilo kroz učitavanje dokumenata s diska. Naime, on sadrži popis zaustavnih riječi te reference na primjerke razreda koji implementiraju sučelje *DocumentReader* (kojemu se delegira posao samog čitanja riječi) i Stemmer (kojemu se delegira posao stemanja riječi). Razred *Stemmer* predstavlja implementaciju tzv. *Porter stemming algorithm* koja je u Javi javno dostupna na Internetskoj stranici samog algoritma. Pozivatelj (glavni program) na početku kroz statičku metodu razreda postavlja objekt tipa *DocumentReader* koji "zna" kako odraditi čitanje. Razred *InputProcessor* će na početku pozvati metodu *readDocument* objekta *DocumentReader* koji će pročitati sve riječi iz odgovarajućeg izvora riječi. Nakon toga *InputProcessor* koristi *Stemmer* kako bi svim riječima uklonio sufikse te ih sveo na kanonski oblik. Ovako obrađene riječi *InputProcessor* vraća pozivatelju.



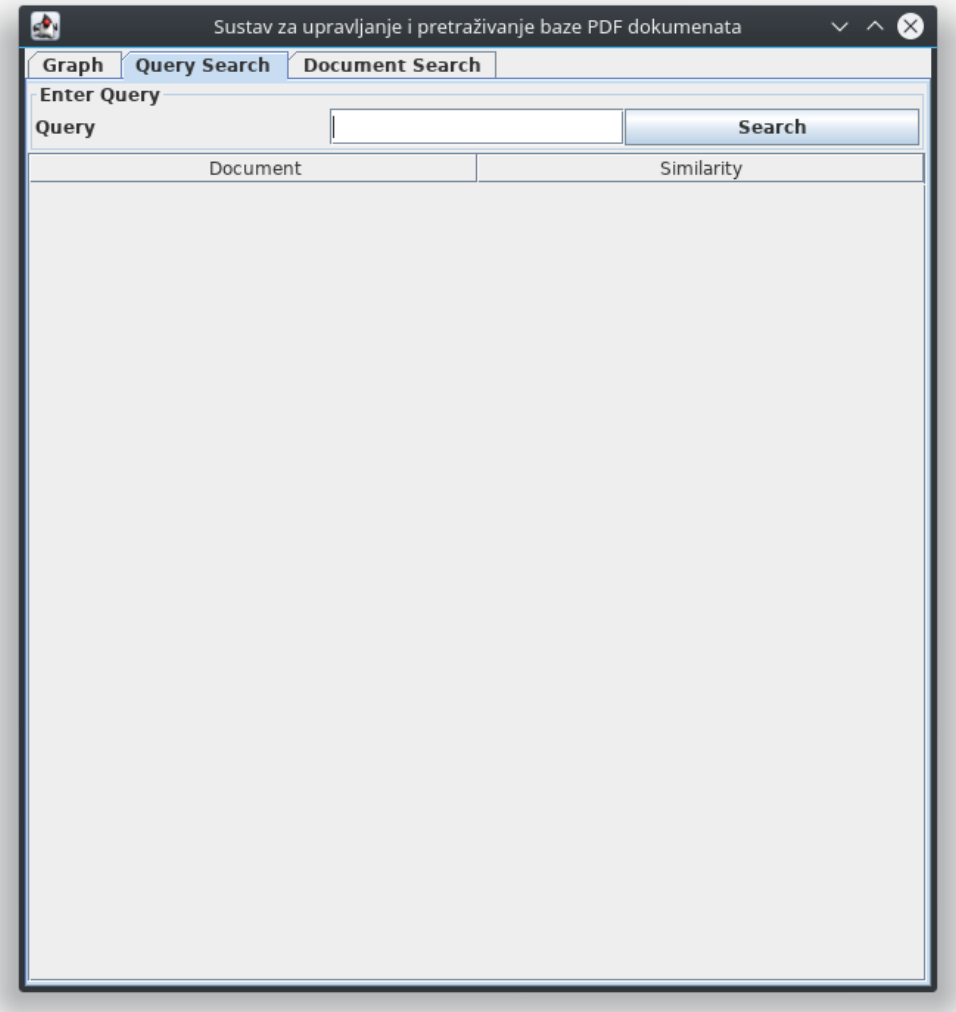
Slika 6.1: Dijagram razreda zaduženih za čitanje riječi (paket *hr.fer.zemris.zavrsni.input*)

6.2. Obrada korisničkog upita

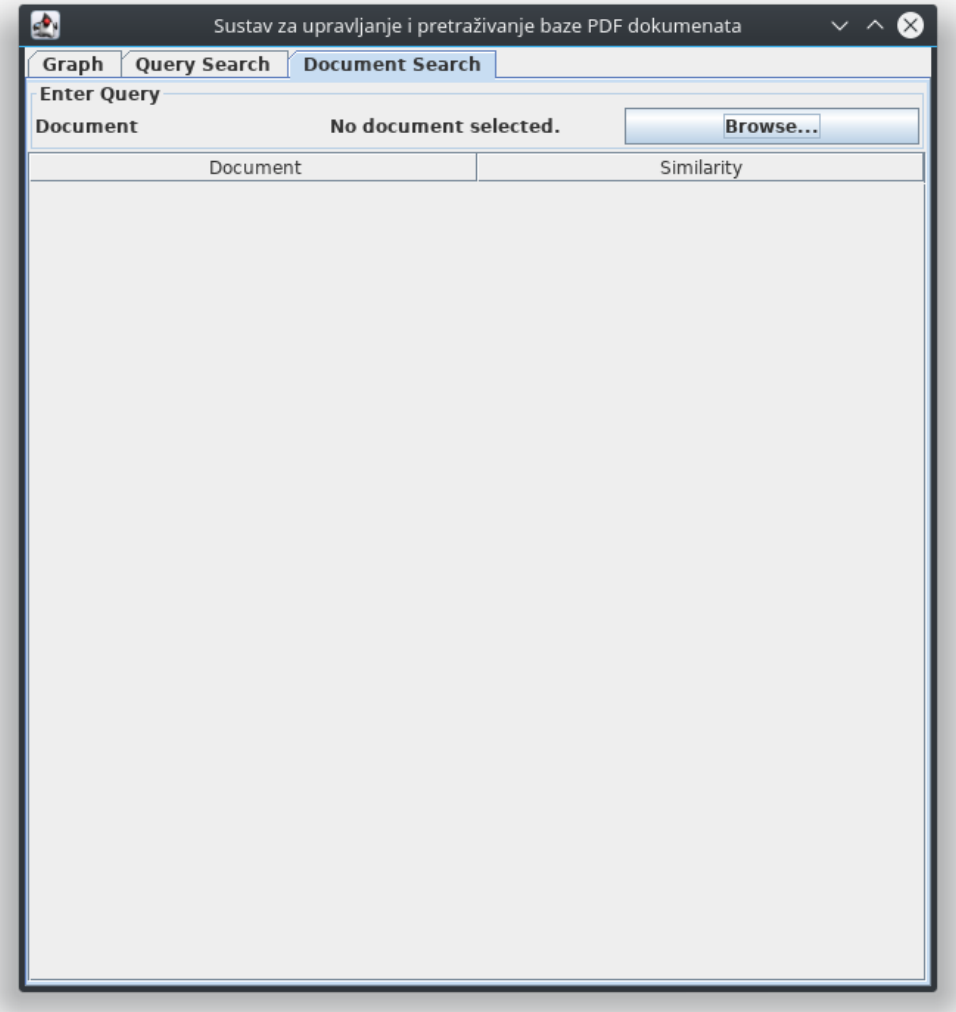
Za obradu korisničkog upita zadužen je razred *QueryReader* koji implementira sučelje *DocumentReader* koje predstavlja apstakciju najviše razine za objekte koji obavljaju akciju čitanja. Kada korisnik unese upit u znakovno polje (engl. *text field*), primjerak razreda *QueryReader* sprema zadani unos u privatnu člansku varijablu. Pozivatelj (glavni program) tada postavlja taj primjerak kao *DocumentReader* razreda *InputProcessor* koji tada koristi objekt kako bi dohvatio upit korisnika. Dohvaćeni tekst se nakon toga procesira (uklanjanje zaustavnih riječi, steming) te je spreman za daljnju obradu (vektorizacija i sama usporedba s dokumentima).

6.3. Obrada dokumenata

Osim pretraživanja zbirke dokumenata prema korisničkom upitu, program nudi mogućnost pronalaska sličnih dokumenata proizvoljno odabranom dokumentu koji se ne nalazi u zbirci. Nakon pokretanja programa i odabira putanje do zbirke dokumenata, korisnik može odabrati proizvoljan dokument s diska kako bi pronašao njemu slične dokumente iz zbirke. Za učitavanje dokumenata s diska (kao i za obradu korisničkog upita), koristi se razred *InputProcessor* koji ne koristi više *QueryReader* kao što to radi prilikom obrade korisničkog upita, već koristi primjerke razreda izvedenih iz apstraktnog razreda *FileReader*—*TextReader* te *PDFReader*. *TextReader* ostvaruje funkcionalnost čitanja riječi iz obične tekstualne datoteke (nastavak *.txt*). Ovaj raz-



Slika 6.2: Prikaz dijela programa za unos korisničkog upita



Slika 6.3: Prikaz dijela programa za odabir proizvoljnog dokumenta

red ne koristi se u finalnoj verziji programske implementacije, no bio je vrlo značajan za potrebe testiranja u implementacijskoj fazi izrade programske potpore. Za čitanje PDF dokumenata koristi se *PDFReader* koji pak posao parsiranja samih PDF dokumenata delegira vanjskoj biblioteci *Apache PDFBox*. Tako dohvaćeni tekst se, kao i kod obrade korisničkog upita, nakon toga procesira te je spreman za daljnju obradu.

6.4. Usporedba korisničkog unosa s dokumentima iz zbirke

Nakaon što pozivatelj dobije kolekciju riječi od razreda *InputProcessor* (bilo dobivenih iz korisničkog upita ili pročitanih iz dokumenata), stvara njihovu TF-IDF vektorsku

reprezentaciju koju modelira razred *Vector* iz paketa *hr.fer.zemris.zavrsni.model*. Tako stvoreni vektor potom se preda javnom konstuktoru razreda *Document* iz istog paketa koji modelira virtualni dokument. *Virtualni* u ovom kontekstu znači da primjerak tog razreda ne predstavlja doslovno dokument koji se nalazi na disku korisnika, već enkapsulira izgrađeni vektor te omogućuje pristup ostalim razredima kroz prikladno definirano sučelje.

```
private void calculateSimilarities() {
   Map<DocumentPair, Double> similiarities = new HashMap<>();
   List<Document> documents = new
      ArrayList<>(documentsMap.values());
   for (int i = 0; i < documents.size(); i++) {
      for (int j = 0; j < documents.size(); j++) {
         if (i >= j) continue;

         Document d1 = documents.get(i);
         Document d2 = documents.get(j);

         double sim = d1.sim(d2) / d1.sim(d1);
         similiarities.put(new DocumentPair(d1, d2), sim);
      }
   }
   datasetInfo.similarities = similiarities;
}
```

Listing 6.1: Isječak programskog koda za generičku usporedbu dokumenata

6.5. Vizualizacija dokumenata

Kao što je već ranije spomenuto u potpoglavlju 4.3, dokumenti zbirke mogu se grafički prikazati u 2D koordinatnom sustavu tako da se uzme u obzir njihova međusobna sličnost.

Ulazni vektor (dobiven bilo iz korisničkog upita ili iz dokumenta učitanog s diska) može se iskoristiti kako bi se korisnički upit, odnosno učitani dokument mogli prikazati grafički. Naime, osim prikazanih dokumenata zbirke i njihovih međusobnih sličnosti, moguće je vidjeti i gdje pripada korisnički upit, odnosno učitani dokument.

6.6. Provjera integriteta zbirke dokumenata

Kako bi program bio u mogućnosti detektirati promjene nad zbirkom dokumenata (primjerice, ako je novi dokument dodan, ako je neki dokument uklonjen itd.), koristi se tehnika provjere kontrolne sume (engl. *checksum*) dokumenata. Naime, nakon što korisnik odabere direktorij na disku koji predstavlja zbirku dokumenata, nad njim se provede rekurzivan postupak izračuna MD5 *hash* (engl. *hash*) vrijednosti svakog od dokumenata u tom direktoriju. Dobivene *hash* vrijednosti se tada usporede s postojećim *hash* vrijednostima koje su spremljene u posebnoj datoteci. Ukoliko se vrijednosti poklapaju, nad zbirkom nije bilo nikakvih promjena od zadnjeg pokretanja te program učitava serijalizirane podatke o zbirci dokumenata i postaje spreman za korištenje. Ako je međutim uočena razlika između postojeće i izračunate *hash* vrijednosti, očito je da je zbirka promijenjena te se nanovo računaju sve relevantne informacije.

```
private static boolean isDatasetCorrect(Path dataset) throws
    IOException {
   String md5 = new MD5Visitor(dataset).getMd5();
   String md5Real;

   if (new File(md5Filename).exists()) {
      md5Real = IOUtils.readFromTextFile(md5Filename);

   } else {
      IOUtils.writeToTextFile(md5Filename, md5);
      return false;
   }

   if (new File(datasetInfoFilename).exists()) {
      if (md5.equals(md5Real)) {
         return true;

      } else {
         IOUtils.writeToTextFile(md5Filename, md5);
         return false;
      }

   } else {
      return false;
   }
}
```

Listing 6.2: Isječak programskog koda za provjeru ispravnosti zbirke

6.7. Optimizacija izvođenja programa

Kako bi upiti korisnika bili što optimiraniji, nakon što se prvi put izvrši čitav gore opisan postupak (preprocesiranje dokumenata, stvaranje vokabulara, računanje sličnosti dokumenata itd.), dobiveni se podaci spremaju u keš (engl. *cache*) memoriju, odnosno bivaju serijalizirani (engl. *serialization*) na disk. Svrha ovog postupka jest jednom dobivene i izračunate relevantne podatke zbirke dokumenata spremiti u perzistentnu memoriju računala kako bi se po svakom sljedećem pokretanju programa, umjesto ponovnog prikupljanja i izračuna svih relevantnih podataka, isti mogli efikasnije isčitati iz zapisane datoteke te deserijalizirati u odgovarajuće strukture podataka čime se uvelike dobija na brzini izvođenja programa, kao što se može vidjeti u Tablici 6.1 u kojoj

Br. dokumenata	Ponovno čitanje (sek)	Deserijalizacija (sek)
7	63.94	0.76
157	670.81	21.05

Tablica 6.1: Usporedba trajanja ponovnog čitanja dokumenata i deserijalizacije

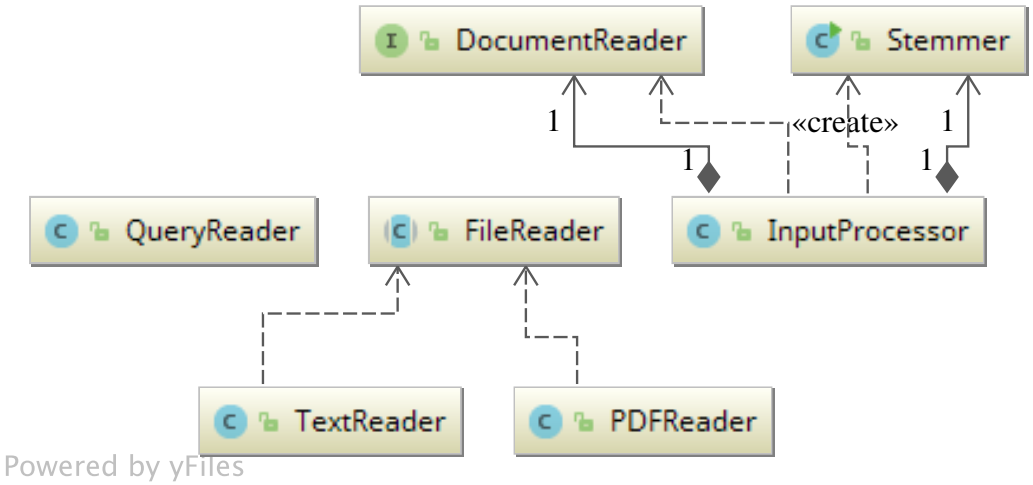
je vidljivo da se korištenjem serijalizacije postiže prosječno ubrzanje od čak 58 puta prilikom svakog (ne-inicijalnog) pokretanja programa.

6.8. Optimizacija hiperparametra *k* algoritma k-sredina

Kao što je već spomenuto u potpoglavlju 4.2.1, određivanje hiperparametra *k* nije izravno moguće iz razloga što algoritmu grupiranja k-means (štoviše, niti cijelome programu) nije dostupna informacija o broju grupa. Ovaj podatak inherentno je nepoznat cijelome programu iz razloga što prije početka izvođenja programa nije poznato koliko se grupa nalazi u zbirci dokumenata. Štoviše, jedan od ciljeva implementacije ovog rada jest i automatsko grupiranje dokumenata koje pak ne mora nužno biti egzaktno, u smislu da točno odredi koji dokumenti spadaju u koju kategoriju dokumenata. Naime, ako bi takva kategorizacija bila relevantna, tada nastaje problem definicije kategorije dokumenata. Budući da, je sličnost dokumenata po definiciji kontinuirana vrijednost, nije moguće odrediti "granicu" koja će odrediti kada neki dokument prestaje pripadati jednoj, a počinje pripadati drugoj kategoriji. Zbog ovog razloga, a i zbog inherentnog nepoznavanja broja kategorija od strane korisnika na početku izvođenja programa, parametar *k* nije moguće odrediti sa sigurnošću. Kako bi se riješio problem određivanja parametra *k*, moguće je iskoristiti određene heuristike koje procijenjuju parametar na temelju empirijskih podataka. Jedna relativno dobra heuristika za optimiranje parametra *k* jest metoda koljena (engl. *elbow method*).

6.8.1. Metoda koljena

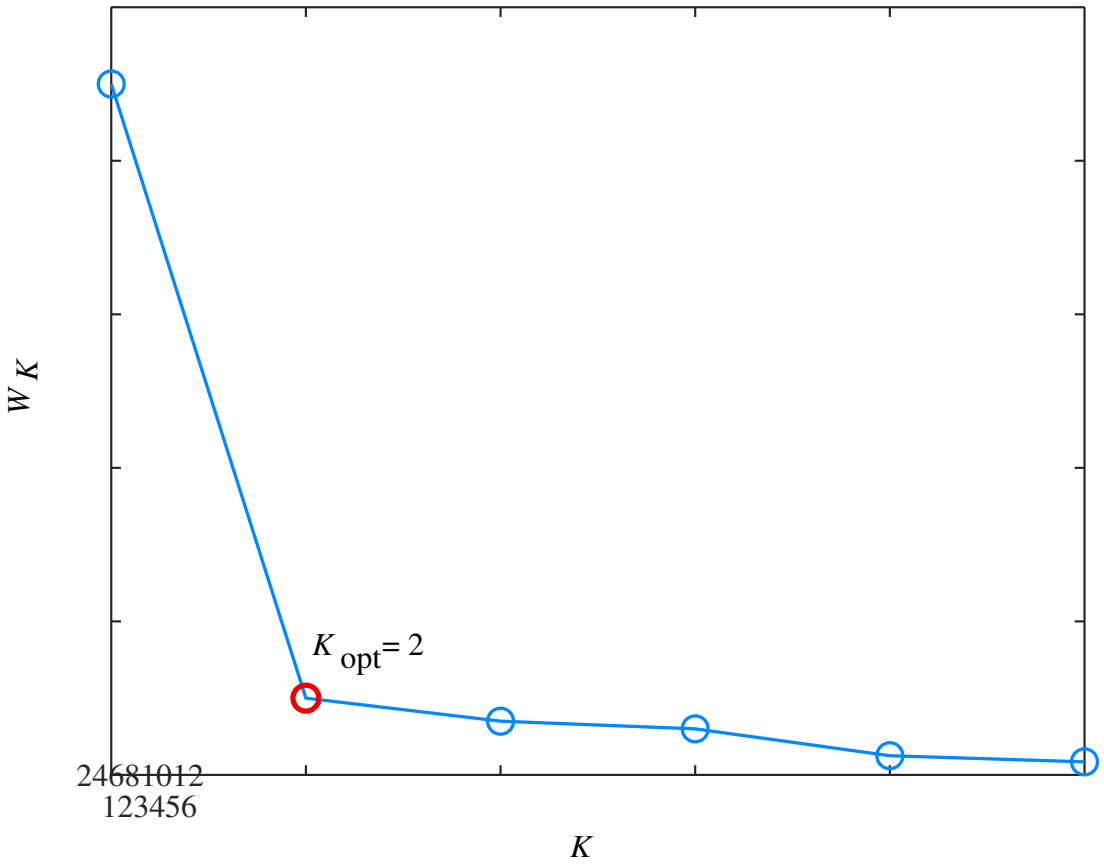
Metoda koljena je jedna od metoda za određivanje broja grupa. Ideja metode je jednostavna: algoritam k-sredina iterativno se izvede za vrijednosti *k* u nekom rasponu (npr. [1, 10]) te se u svakoj iteraciji izračuna suma srednjih kvadratnih pogrešaka (engl. *residual sum of squares, RSS*) udaljenosti svake od točaka do centroida dodijeljene joj grupe. Nakon izvršenog postuka, biti će izračunate sume kvadratnih pogrešaka za svaku od iteracija koje su se izvršile, odnosno za svaku vrijednost parametra *k* iz pret-



Slika 6.4: Dijagram razreda zaduženih za čitanje riječi (paket *hr.fer.zemris.zavrsni.input*)

hodno zadanog raspona. Nakon izvršenog postupka, za svaku od vrijednosti parametra k nacrta se pripadajuća suma kvadratnih pogrešaka te se promotri u kojoj točki (tj. za koji k) se nalazi "koljeno", u smislu najveći pad vrijednosti sume pogreške. Jedan takav primjer prikazan je na slici 6.5 na kojoj se jasno vidi veliko smanjenje pogreške između $k=1$ i $k=2$ što znači da će se kao vrijednost parametra k uzeti vrijednost 2 zbog toga što je upravo za tu vrijednost pronađena najveći pad pogreške, što ukazuje na to da je algoritam pronašao optimalan broj grupa. Čitatelj bi se mogao zapitati zbog čega se kao broj grupa onda ne uzme neka veća vrijednost, primjerice onu za koju je pogreška najmanja (nula). No odgovor na ovo je jednostavan: kada bi suma pogrešaka bila nula, to znači da bi broj grupa bio upravo jednak broju dokumenata. No ovo nema smisla, pošto u kontekstu ovog problema želimo grupirati više dokumenata u grupe. Zaključak je dakle da treba odabrati takvu vrijednost parametra k za kojeg postoji najveći pad pogreške. Ovakav odabir skoro uvijek biti će optimalan.

Metoda koljena bila bi idealna za optimizaciju parametra k kada bi postojala jednostavna metoda određivanja već spomenutog "koljena". No ovdje nastaje novi problem programskog određivanja točke najvećeg pada pogreške. Tome problemu moglo bi se doskočiti novim heuristikama (koje bi s obzirom na početni problem zapravo bile metaheuristike), no u tom slučaju je rješenje problema postalo novi problem kojeg treba riješiti te razina apstrakcije time značajno raste. Ovakav pokušaj nema previše smisla za automatizirano računanja, stoga ova metoda nažalost nije dovoljno dobra u kontekstu ovog rada. U sljedećem potpoglavlju biti će pokazana još jedna heuristika koja se može iskoristiti za problem određivanja parametra k.



Slika 6.5: Ovisnost vrijednosti pogreške o parametru k

6.8.2. Metoda polovičnog korijena

Jednostavna heuristika koja se koristi u ovom radu je sljedeća:

$$k = \sqrt{\frac{|D|}{2}}, \tag{6.1}$$

gdje je D zbirka dokumenata. Ovakva heuristika ne dovodi nužno do optimalnog rješenja (tj. do egzaktnog broja grupa), no služi kao relativno dobra aproksimacija, što je u ovome kontekstu često puta sasvim dovoljno. Primjerice, za zbirku dokumenata u kojoj na početku imamo dvije jasno odvojene kategorije, npr. engleski nogomet i srednjevjekovno mačevanje, zanimljivo je promotriti parametar k ovisi o broju dokumenata u pojedinoj kategoriji.

Iz tablice 6.2, zanimljivo je uočiti kako porast broja dokumenata u kategorijama dovodi do (vrlo) pogrešne aproksimacije broja kategorija. Naime, očito je (prilikom konstrukcije gornjeg izraza) empirijski ustanovljeno kako svaka od kategorija u prosjeku nema prevelik broj dokumenata, zbog čega aproksimacija za velik broj dokumenata; u ovom konkretnom primjeru, za približno 10-ak dokumenata po kategoriji,

Engleski nogomet (br. dok.)	Srednjevjekovno mačevanje (br. dok.)	k
3	4	1.87
5	5	2.24
11	17	3.74
42	53	6.87

Tablica 6.2: Ovisnost parametra k o broju dokumenata pojedine kategorije

pogreška aproksimacije parametra k doseže čak 100%! Ovaj podatak govori o problematici heuristika: često puta daju dovoljno dobru aproksimaciju, no ponekad mogu i poprilično pogriješiti.

7. Zaključak

Zaključak.

LITERATURA

Sustav za upravljanje i pretraživanje baze PDF dokumenata

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.