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

Slavko Žitnik

INTELIGENTNO PRIDOBIVANJE PODATKOV IZ POLJUBNIH VIROV

DIPLOMSKO DELO
NA INTERDISCIPLINARNEM UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 00017/2010
Datum: 01.09.2010

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

Kandidat: **SLAVKO ŽITNIK**

Naslov: **INTELIGENTNO PRIDOBIVANJE PODATKOV IZ POLJUBNIH VIROV**

INTELLIGENT DATA RETRIEVAL FROM ARBITRARY DATA SOURCES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Na svetovnem spletu so danes na voljo enormne količine dokumentov. To je obenem tudi težava, saj se v poplavi podatkov težko znajdemo, t.j. izluščimo tiste vsebine, ki so za nas relevantne. Dodaten izziv predstavljajo raznovrstne oblike in nestrukturirani viri, v katerih se podatki »skrivajo«. Raziskave v gospodarstvu kažejo, da podjetja pridobijo več kot 50% podatkov iz delno ali nestrukturiranih virov.

V okviru diplomske naloge izdelajte zasnovo sistema, ki bo omogočal iskanje po različnih podatkovnih virih. Sistem naj omogoča različne, inovativne načine zajema poizvedbe, rezultat, ki ga vrne, pa naj ne bo le seznam povezav, temveč analiza vsebin, na katere povezave kažejo. Opišite delovanje takšnega sistema po posameznih komponentah.

Mentor:

prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Franc Solina

Dekan Fakultete za matematiko in fiziko:

prof. dr. Andrej Likar

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 6

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Slavko Žitnik,

z vpisno številko 63060254,

sem avtor/-ica diplomskega dela z naslovom:

INTELIGENTNO PRIDOBIVANJE PODATKOV IZ POLJUBNIH VIROV

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom izr. prof. dr. Marka Bajca
- 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 5.9.2010 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se prof. dr. Marku Bajcu za predloge in vodenje pri izdelavi diplomskega dela.

Prav tako se želim zahvaliti Lovru Šublju, Štefanu Furlanu in Gabru Terseglavu, ki so bili pripravljeni na pogovor o izbrani tematiki.

Za odpravo slovničnih napak se zahvaljujem sestri Marinki.

Sometimes a research is a lot of hard work in looking for the easy way

Kazalo

Seznam uporabljenih kratic in simbolov

Oznaka	Pomen
iCORE	intelligent context-aware (multisource) information retrieval
XML	Extensible Markup Language
TF-IDF	term frequency - inverse document frequency
SVM	metoda podpornih vektorjev (ang. support vector machine)
V	množica različnih besed v sistemu
w_{ij}	teža besede i v dokumentu j
d_j	j-ti dokument v vektorskem zapisu
$Pr(\cdot)$	verjetnost
q	poizvedba v vektorskem zapisu
$\vec{q}_{opt}$	optimalna poizvedba
$\vec{q}_m$	premaknjena poizvedba
α, β, γ	uteži $\in [0, 1]$
C_{nr}, C_r	ne/-relevantni dokumenti v celotnem sistemu
D_{nr}, D_r	ne/-relevantni dokumenti med rezultati
$C(\cdot)$	centralnost
$PR(\cdot)$	PageRank ocena
$V(i)$	množica dokumentov, ki kažejo na dokument i
$I(i)$	množica dokumentov, na katere kaže dokument i
P	prehodna matrika
y	ocena rangiranja
$\vec{x}$	vektor ocen atributov
$R_{D,I}$	ocena rangiranja dokumenta D iskalnika I
$MI(D)$	množica iskalnikov, ki so vrnili D
$ICORERank(q, D)$	ocena algoritma za rangiranje sistema iCORE
$sim_x(\cdot, \cdot)$	ocena podobnosti

Povzetek

Iskalnik je sistem, ki ga uporablja vsak izmed nas. Če ne uporabljamo ravno spletnega iskalnika, smo gotovo že uporabili katerega drugega, saj vsaka večja količina predmetov potrebuje indeks, ki nam omogoča iskanje, pa naj si bo to le kazalo v knjigi.

Namen te diplomske naloge je predstaviti nov iskalnik oziroma sistem za priklic podatkov. Poimenovali smo ga iCORE - intelligent context-aware (multisource) information retrieval. Predlagani sistem skuša narediti iskanje še bolj presonalizirano. Zaradi strukture omogoča uporabo na katerikoli napravi in beleženje uporabnikovih aktivnosti. Sistem iCORE želi uporabnika spremljati do takšne mere, da mu bo prikazoval strukturirane rezultate poizvedbe brez da bi mu jih eksplicitno podal. Uporablja lahko več specializiranih iskalnikov - metaiskalnik ali virov, sprocesira zadetke in iz njih izlušči informacije. Uporabnik kot rezultat sistema iCORE prejme sestavljene informacije, ki so ustrezno prikazane glede na njihovo strukturo. Uporabnik mora imeti možnost ročnega nastavljanja preferenc, usmerjenega iskanja in navigacije med vrnjenimi entitetami, ki so med seboj relacijsko povezane.

Pri predlogih izvedb komponent sistema uporabljamo vrhunske algoritme s področja strojnega učenja. Komponente lahko zato testiramo po znanih postopkih, celotni sistem iCORE lahko najbolje ocenimo z njegovo uporabo v realnem svetu.

Tekom razvoja iskalnikov je bilo razvitih več metaiskalnikov in vsi so povprečno izboljšali rezultate. Torej bo sistem iCORE, ki ni zgolj metaiskalnik, temveč še mnogo drugega, gotovo izboljšal celostno uporabniško izkušnjo in rezultate.

Ključne besede:

priklic podatkov, iskalnik, kontekst, personalizacija, iCORE

Abstract

Search engine is a system used by everyone. When not using just web search engine, we definitely have used some other kind of it. Every larger corpus of items has its own style of search engine, just like table of contents in a book.

The aim of this diploma thesis is to present a new search engine and information retrieval system respectively. We have named it iCORE - intelligent context-aware (multisource) information retrieval. This system personalizes search even more than others. It is structured in the way that can be used on any possible device to also track the user. iCORE wants to focus on the user and retrieve relevant information even before user actually tries to give an explicit input. It uses other specialized search engines or data collections. Results are presented as structured information, which are suitably shown. The user has to have options such as setting his preferences, faceted search and navigating through results.

System is based on the state-of-the-art algorithms, also used in machine learning. Therefore each component can be tested using known datasets. The best way to evaluate iCORE system as a whole is just by starting using it for real purposes.

During search engine developments there have been built many meta-search engines, all of which have on average returned better results. iCORE system is a lot more than just a pure meta-search engine, so it will definitely improve results and user experience.

Key words:

information retrieval, search engine, context, personalization, iCORE

Uvod

Motivacija

Odkar je človek začel zbirati podatke in jih hraniti v informacijah, je vedno skušal poskrbeti, da so bili čim enostavneje dosegljivi. Značilni primer organizacije podatkov je kazalo v knjigi. Z začetkom tiskanja knjig in ustanavljanjem knjižnic se je uveljavilo urejevanje publikacij po tematskih področjih, naslovu in avtorju.

Podobno se je s podatki dogajalo v računalništvu. Sprva je bilo le malo dokumentov, ki smo jih lahko obvladali. Zaradi potrebe po izmenjevanju virov se je rodil svetovni splet. Tipično se vsak dokument, podobno kot ljudje, nahaja na določenem naslovu. Ta naslov moramo eksplicitno poznati, da lahko do njega dostopamo. Pravzaprav so se spletni iskalniki pojavili za reševanje teh težav. Omogočajo uporabo širšim množicam ljudi, izjemno poenostavljajo dostop do dokumentov in pohitrijo uporabo interneta. Vprašajte se: "*Kam pogledate, ko želite pridobiti neke informacije?*" Prepričani smo, da je vodilni odgovor le eden, ki sliši na ime najbolj popularnega iskalnika v današnjem času.

Čeprav so vam spletni iskalniki gotovo najbližje, so avtomatski ali polavtomatski sistemi za pridobivanje podatkov že dolgo med nami. Najprej so se začeli uporabljati v knjižnicah, kjer je bila potreba največja. Iskalni sistem je vsak, ki poizveduje nad podatkovno bazo. Gotovo ste se že znašli v primeru, ko vas je uradnik vprašal: "*Ali mi poveste vaše ime in priimek, da najdem vaše podatke?*". V tem primeru mu je sistem omogočal poizvedovanje do podatkov le po imenu in priimku.

Prvotna beseda za iskanje je bila priklic podatkov:

Definicija: Priklic podatkov (ang. IR - information retrieval) je pridobivanje dokumentov iz velike zbirke podatkov, ki zadovolji uporabnikovo potrebo po informacijah.

Termin iskanje se je pogosteje uporabljal in je zato ponarodel.

Torej, če vse to imamo, **kaj sploh še lahko izboljšamo?** Trdimo naslednje:

Data is the New Oil.

If data is the New Oil, then we need a bigger drill.

Omenjeni večji vrtalnik bomo zasnovali mi. Do sedaj smo omenjali le pridobivanje dokumentov. Res je, da smo tekom razvoja izboljševali načine iskanja, personalizacijo rezultatov, dodajali podporo za različne vrste dokumentov, prikazovali relevantne oglase, ... Noben od sistemov pa se ni ukvarjal z informacijami v dokumentih, njihovo obdelavo in prikazovanjem. V raziskovalnih krogih so se porodile zamisli o semantičnem svetovnem spletu (ang. semantic web), a je iluzorno pričakovati, da bo do množične uporabo sploh kdaj prišlo.

V nadaljevanju bomo predstavili revolucionarni sistem iCORE, ki vsebuje vse značilnosti sodobnih iskalnih sistemov, poskuša razumeti uporabnika in mu predstavi informacije v obliki, ki jih želi.

V prvem poglavju bomo predstavili razvoj sistemov za priklic podatkov in osnovne ideje njihovega delovanja. Nadaljevali bomo s kratko predstavitvijo sistema iCORE, kjer bomo opisali koncept delovanja in njegovo strukturo. V tretjem poglavju bomo obdelali vsako komponento sistema posebej, opisali njeno delovanje in predlagali metode za njeno implementacijo.

Poglavje 1

Priklic podatkov

1.1 Razvoj sistemov

1.1.1 Zgodovina

Vizionar Vannevar Bush si je v sredini prejšnjega stoletja zamislil koncept hranjenja dokumentov [26]. Napovedal je bodočo Wikipedijo, brezplačen dostop do nje vsem ljudem, sistem za priklic podatkov... Predlagani sistem "*memex*" je za današnje čase res primitiven, vendar je bilo tedaj takšno razmišljanje revolucionarno. Osredotočil se je predvsem na iskanje po knjigah. Prvi sistemi za priklic podatkov so se pojavili v knjižnicah, kjer so čutili največjo potrebo po njih.

V 1960-ih letih je profesor Gerard Salton z raziskovalci na Cornell University začel razvjati sistem SMART (ang. System for the Mechanical Analysis and Retrieval of Text). SMART velja za enega prvih iskalnih sistemov. Uporabljal je vektorski model priklica podatkov (glej razdelek 1.2), Rocchio algoritem (glej razdelek 3.1.1) in TF-IDF uteži (glej enačbo 1.5) za besede. Salton je naredil velik premik na področju priklica podatkov in bil prvi, ki mu je bila za to področje podeljena nagrada, ki se sedaj imenuje *Gerard Salton Award* in se podeljuje na tri leta.

Naslednji velik korak se je zgodil v 1990-ih letih z razmahom uporabe svetovnega spleta. Sisteme, ki so bili narejeni za obvladovanje bolj ali manj statičnih zbirk, so izboljšali in prilagodili v spletne iskalnike.

1.1.2 Uporaba

Učinkovit priklic podatkov je odvisen od uporabnika in sistema. Uporabnikovo nalogo pri iskanju delimo na **oblikovanje poizvedbe** in **brskanje**. Uporabnik mora znati pravilno oblikovati poizvedbo, ki jo lahko poda na način, kot to omogoča sistem. Ko mu sistem vrne rezultate, nastopi naloga brskanja. Med vrnjenimi dokumenti mora ugotoviti, kateri je zanj primeren, in pridobiti želeno informacijo. Sistem mora imeti primerno **logično shranjene dokumente**. Način shranjevanja je odvisen od modela za priklic podatkov in obdelave dokumentov (za razlago glej naslednja razdelka). Sistem iCORE skrbi, da uporabnikovo nalogo minimizira, kolikor se le da.

Glede na namen uporabe spletnega iskalnega sistema razločujemo tri vrste iskalnih sistemov [27]:

Navigacijski: Uporabnik želi obiskati določeno spletno stran, ker jo je že kdaj obiskal ali sklepa, da obstaja. Navadno je pravilen rezultat poizvedbe le eden. Primer: Kot rezultat iskalnega niza "učilnica fri" želim dostopati do strani "ucilnica.fri.uni-lj.si".

Informacijski: Uporabnik želi pridobiti informacije, ki se nahajajo v določenem dokumentu v nestrukturirani obliki.

Transakcijski: Uporabnik želi najti strani, kjer bi opravil transakcijo (nakup, ogled videa, prenos dokumentov, ...).

1.1.3 Sistemi

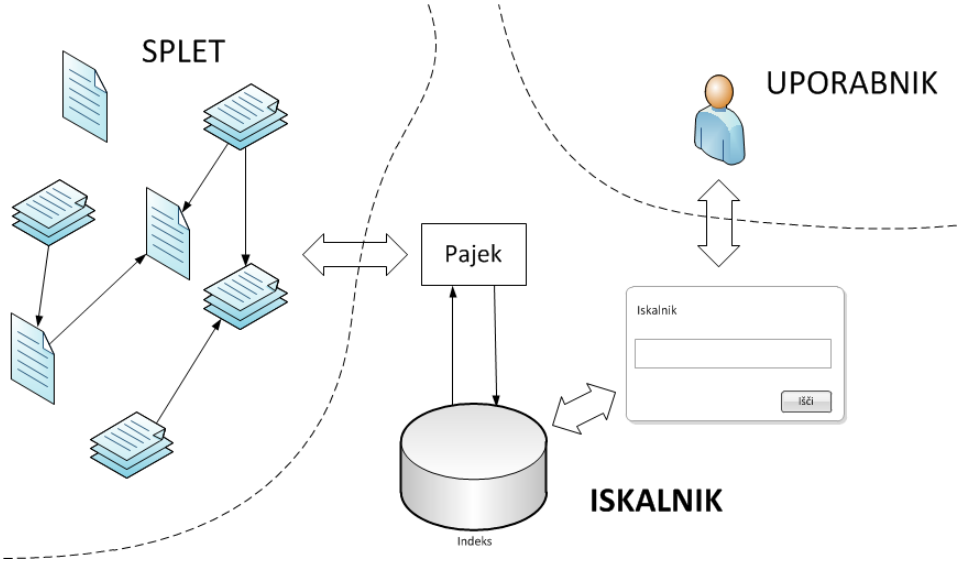
Zgradbo enostavnega iskalnika si lahko ogledate na sliki 1.1.

Sistemu lahko poljubno dodajamo druge uporabne komponente. Ena takšnih je podsistem za oglaševanje, s katerim se ukvarja področje za iskalniški marketing (ang. SEM - search engine marketing). Deluje kot samostojen iskalnik in poskuša uporabniku prikazati oglase, ki bi ga utegnili zanimati. Lastniki iskalnikov tržijo oglasni prostor kot zakupljenega za določen čas ali ga obračunavajo glede na število klikov uporabnika.

Poseben primer iskalnika je metaiskalnik (Slika 1.2), ki uporablja druge iskalnike, rangira njihove rezultate in jih prikaže kot svoje.

Iskalniki so se razvijali sledeče [29, 30]:

Traditional Search 1.0 (1996): Sestavni deli iskalnikov so bili t.i. pajki (glej razdelek 1.4), ki so obiskovali strani in jih kopirali v zasebni indeks. Ko je uporabnik podal poizvedbo, je iskalnik pregledal dokumente in jih



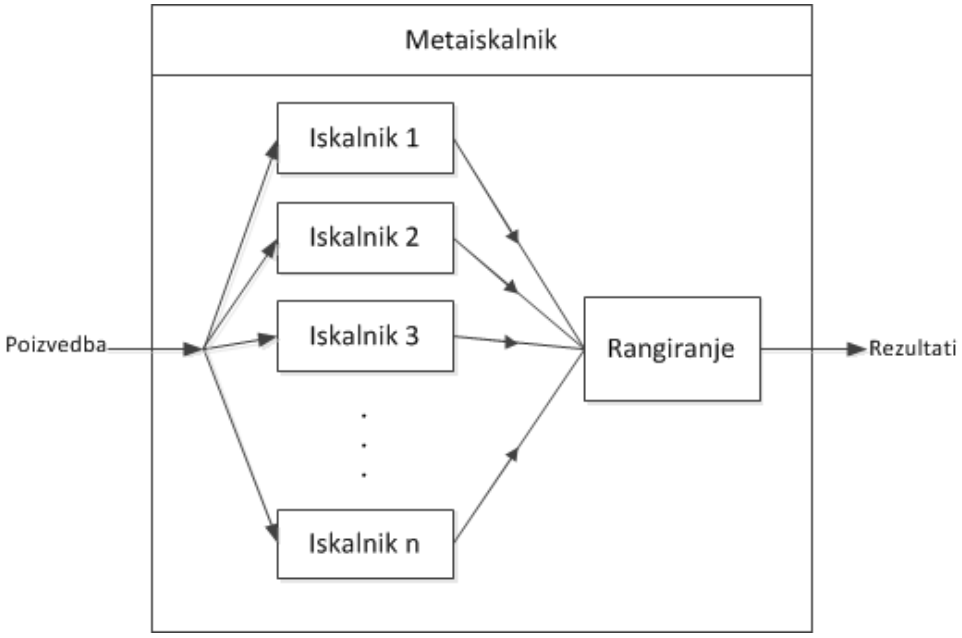
Slika 1.1: Shema enostavnega iskalnika.

rangiral glede na število ujemajočih besed in njihovo lokacijo. Če se je beseda nahajala v naslovu, je bilo pomembneje kot v besedilu. Glede na oceno je uporabniku vrnil zgornjih k dokumentov.

Search 2.0 (1999): Nekateri so hitro ugotovili, kako iskalniki delujejo, zato so na straneh skrili ključne besede v obliki nevidnega teksta in si s tem povečali oceno v rezultatih. Področju, ki se še vedno ukvarja z zviševanjem ocen določene strani v iskalnikih, pravimo optimizacija iskalnikov (ang. SEO - Search Engine Optimization). Naenkrat se je pojavilo mnogo strani z vsiljivo vsebino (ang. spam), zato so se raziskovalci odločili upoštevati še povezave med stranmi. Izhajali so iz stališča, da resne strani ne kažejo na vsiljive. Znan algoritem za računanje takšnih ocen je PageRank (glej razdelek 3.2.1).

Search 3.0 (2007): Do tedaj so se uporabljali splošni iskalniki, ki so iskali po spletnih straneh ali specializirani iskalniki za slike, zvok, video, knjige,... Iskanje je bilo torej sestavljeno iz **horizontalnega** in **vertikalnega** dela. Horizontalni del določa, da si uporabnik izbere temo zanimanja in nato vertikalno išče v globino želene teme. Iskalniki so omogočili združevanje rezultatov in primeren prikaz za vsako kategorijo rezultata.

Search 4.0 (2009): Pojavili so se iskalniki z izboljšano personalizacijo oz. vključenim socialnim iskanjem. Skoraj vsi uporabniki informacijske teh-



Slika 1.2: Metaiskalnik.

nologije in socialnih omrežij so ugotovili, kako se jih da uporabiti v dobre in slabe namene. Razvoj iskalnikov je bil posvečen učenju klasifikatorjev glede na uporabnikove prijatelje. V razširjenih socialnih omrežjih lahko določene dokumente oceni veliko uporabnikov, kar so uporabili tudi iskalniki.

Search 5.0 (2011): V sistemu iCORE nadaljujemo v smeri prejšnjih iskalnikov in predlagamo nove funkcionalnosti. Iskalnik naj ne bo le iskalnik, temveč naj bo tudi v stiku z uporabnikom kot njegov najboljši prijatelj. Naj mu predstavi ne le podatkov, ampak tudi informacije v strukturirani obliki. Semantični splet (glej razdelek 3.5) ne obstaja, zato je naloga iCORE-a, da ga virtualizira (več o našem sistemu v naslednjih poglavjih).

Sistem iCORE uporablja postopke spletnega rudarjenja (ang. web mining). Algoritmi izhajajo iz podatkovnega rudarjenja (ang. data mining) in so prilagojeni za uporabo nad spletnimi dokumenti. Podatkovno rudarjenje se uporablja za odkrivanje vzorcev ali znanja iz različnih virov podatkov. Naučene vzorce nato uporabljamo za predstavitev ali klasificiranje novih primerov. Proces spletnega rudarjenja je podoben podatkovnemu in ga delimo v tri skupine. S **strukturnim rudarjenjem** želimo odkrivati zakonitosti iz

povezav med dokumenti, kot so uporabniki z enakimi interesi. **Vsebinsko rudarjenje** poskuša odkrivati podobnosti v vsebini dokumentov in iz njih izluščiti pomembne informacije, ki so lahko celotne entitete. **Uporabniško rudarjenje** omogoča klasificiranje uporabnikov glede na njihovo obnašanje - izbiranje določenih povezav, vpisovanje poizvedb, itd.

1.2 Modeli za priklic podatkov

Model za priklic podatkov določa, kako v sistemu predstavimo dokumente in poizvedbe. V nadaljevanju bomo predstavili najbolj uporabljane modele.

Vsi modeli obravnavajo dokumente kot vrečo besed (ang. bag of words). Naj $V = \{t_1, t_2, \ldots, t_{|V|}\}$ predstavlja slovar vseh besed, ki jih poznamo. Naj $w_{ij} \geq 0$ predstavlja težo besede t_i v dokumentu $d_j \in D$. Vsak dokument torej predstavimo z vektorjem:

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{|V|j}). \tag{1.1}$$

Pred iskanjem poizvedbo na enak način kot dokument pretvorimo v vektor. Iskanje izvedemo tako, da vektor poizvedbe primerjamo z vsemi vektorji dokumentov glede na model.

1.2.1 Logični model

Logični model je najstarejši in najenostavnejši.

Uteži besed v dokumentih so določene binarno:

$$w_{ij} = \begin{cases} 1 & t_i \in d_j \\ 0 & sicer \end{cases} \tag{1.2}$$

Poizvedba je podana z besedami, ki jih vežejo logični operatorji **AND**, **OR** in **NOT**. Sistem vrne množico vseh dokumentov, ki izpolnjujejo poizvedbo.

1.2.2 Vektorski model

Dokument je predstavljen kot vektor uteži, kjer so uteži realno število. Za računanje uteži se najbolj uporablja TF-IDF shema.

TF-IDF (ang. term frequency inverse document frequency) predviveva, da je beseda pomembna, če se večkrat pojavi v malo dokumentih, kot če jo vsebuje vsak dokument. Naj bo n število dokumentov v sistemu. Število df_i predstavlja število dokumentov, kjer se beseda t_i pojavi vsaj enkrat. Naj bo

f_{ij} število pojavitev besede t_i v d_j in je tf_{ij} njena normalizirana vrednost. Težo w_{ij} izračunamo:

$$tf_{ij} = \frac{f_{ij}}{max\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\}} \tag{1.3}$$

$$idf_i = \log \frac{n}{df_i} \tag{1.4}$$

$$w_{ij} = tfidf_{ij} = tf_{ij} \times idf_i \tag{1.5}$$

Podobnost med poizvedbo in dokumentom izračunamo s kosinusno podobnostjo (glej enačbo 3.1), ki predstavlja normaliziran skalarni produkt dveh vektorjev.

Pri logičnem modelu smo dobili kot rezultat le vrednost 0 ali 1, pri vektorskem modelu pa lahko dobimo katerokoli realno število, glede na katero lahko rangiramo rezultate in prikažemo le najboljših k.

1.2.3 Jezikovni model

Jezikovni modeli izhajajo iz teorije verjetnosti.

Naj bo poizvedba q sestavljena iz zaporedja besed $q = q_1 q_2 \ldots q_m$. Sedaj iščemo dokument, ki naj bi bil najbolj verjeten, če poznamo q. Z uporabo Bayesovega pravila dobimo:

$$Pr(d_j|q) = \frac{Pr(q|d_j)Pr(d_j)}{Pr(q)} \tag{1.6}$$

Verjetnost poizvedbe $Pr(q)$ lahko izpustimo, saj je vedno enaka, ne glede na dokument. $Pr(d_j)$ lahko razumemmo kot konstanto, ki ne bo vplivala na rangiranje in jo zato lahko izpustimo. Torej moramo izračunati le $Pr(q|d_j)$.

Predpostavimo, da so besede v dokumentih med seboj neodvisne. Vrednost f_{iq} naj predstavlja število pojavitev besede t_i v poizvedbi q. Za vsak dokument $d_j \in D$ lahko sedaj ocenimo:

$$Pr(q = q_1 q_2 \ldots q_m|d_j) = \prod_{i=1}^{m} Pr(q_i|d_j) = \prod_{i=1}^{|V|} Pr(t_i|d_j)^{f_{iq}} \tag{1.7}$$

Vemo, da velja $\sum_{i=1}^{|V|} Pr(t_i|d_j) = 1$. Vsako verjetnost $Pr(t_i|d_j)$ lahko izračunamo:

$$Pr(t_i|d_j) = \frac{f_{ij}}{|d_j|} \tag{1.8}$$

Pri takšni oceni prihaja do problemov. Predpostavimo, da imamo v sistemu natanko en dokument, ki vsebuje skoraj vse besede iz poizvedbe, ostali pa nobene. V tem primeru sistem ne bo vrnil nobenega dokumenta, ker bodo vsi dobili oceno 0. To popravimo z uporabo Laplaceove ocene ($\lambda = 1$) in se na ta način izognemo mejnima vrednostima 0 in 1 za $Pr(t_i|d_j)$:

$$Pr(t_i|d_j) = \frac{\lambda + f_{ij}}{\lambda|V| + |d_j|} \tag{1.9}$$

1.2.4 Latentno semantično indeksiranje

Prejšnji modeli upoštevajo le besede, ki so podane in jih sistemi lokalno vsebujejo. Latentno semantično indeksiranje (ang. latent semantic indexing) upošteva še sopomenke oz. besede, ki se uporabljajo v podobnih besedilih.

Pri računanju dokumente sistema predstavimo v matriki kot vektorje besed. Predpostavimo, da imamo n dokumentov in m možnih besed. Matrika A bo dimenzije $m \times n$. Običajno je takšna matrika velika, zato jo zaradi lažjega računanja razcepimo po postopku singularnega razcepa. Naj bo r rang matrike A. Potem matrika U dimenzije $m \times r$ predstavlja vektorje besed, Σ diagonalno matriko $r \times r$ singularnih vrednosti in V^T matriko $r \times n$ vektorjev dokumentov.

$$A = U\Sigma V^T \tag{1.10}$$

Pomembna lastnost singularnega razcepa je, da lahko razcepljenim matrikam odstranimo manj pomembne dimenzije in bomo vseeno dovolj dobro apriksimirali prvotno matriko A. Zato izberemo k največjih singularnih vrednosti iz matrike Σ, kjer $k \leq r$. Matriko A aproksimiramo z matriko A_k.

$$A_k = U_k\Sigma_k V_k^T \tag{1.11}$$

Zaradi manjših razsežnosti matrik U, Σ, V^T je A_k precej lažje izračunati. V praksi se za k uporabljajo vrednosti od 50 do 1000. Poizvedbo q sedaj še pretvorimo v k dimenzij kot q_k. Spremenjeno poizvedbo q_k si predstavljamo kot dokument v V. Iz takšne enačbe izračunamo q_k.

$$q = U_k\Sigma_k q_k^T \tag{1.12}$$
$$q_k = q^T U_k \Sigma_k^{-1} \tag{1.13}$$

Za pridobitev podobnosti dokumentov primerjamo q_k z vrsticami V_k. Vsaka vrstica v V_k predstavlja dokument. Za primerjavo lahko uporabimo kosinusno

oceno. Ko so ocene znane, jih razvrstimo in vrnemo najbolj relevantne dokumente.

Glavna slabost latentnega semantičnega indeksiranja je njegova časovna zahtevnost $O(m^2n)$. Ob začetku uporabe je bila izbira optimalnega števila dimenzij k v singularnem razcepu težavna. Kot splošno pravilo velja, da manj dimenzij pomeni bolj splošno primerjavo poizvedbe in dokumentov, več pa bolj specifično. Raziskave so pokazale, da običajno 300 do 400 dimenzij za večji nabor dokumentov vrača najboljše rezultate [36]. Primer delovanja in izpeljave razcepa ter metode si lahko ogledate v [31, 32].

1.3 Obdelava in hranjenje dokumentov

Sistemi za priklic podatkov lokalno ne hranijo celotnih dokumentov. Dokumentov je ogromno in bi zasedali preveč prostora. Sistemi jih zato obdelajo in shranijo v primerni strukturi za hitro pridobivanje. Osnovni postopek je sledeč:

1. Pridobitev dokumenta
2. Procesiranje dokumenta
3. Dodajanje dokumenta v lokalno strukturo sistema (indeks)

Z uporabo spodnjega primera bomo podrobneje opisali celoten postopek obdelave:

Kak težka, bridka ura je slovesa!
Stojé po licah jima kaplje vroče,
objeta sta, ko bi bila telesa
en'ga, spustiti žnabel žnabla noče;
si z lev'ga óča, desnega očesa
jok briše, ki ga skriti ni mogoče,
ko vidi v táko žalost nju vtopljene,
in de tolažbe zanje ni nobene.

Najprej moramo definirati, kaj je dokument. To je osnovna enota, ki jo obdeluje sistem. Dokument je lahko datoteka na računalniku ali pa le njen del, npr. posamezno e-poštno sporočilo. V našem primeru bo vsaka vrstica besedila predstavljala svoj dokument.

Najprej moramo iz dokumenta pridobiti osnovne elemente - besede (ang. tokenization):

Kak | težka | bridka | ura | je | slovesa

Nato moramo odstraniti besede, ki se zelo pogosto pojavljajo v vseh besedilih in ne vplivajo na iskanje (ang. stopwords). Te besede so predlogi, zaimki, vezniki,... kot so {*in, ali, ampak, čeprav, ki, ko, na, v, kako, kak, že, sem,...*}:

težka | bridka | ura | slovesa

Sedaj je potrebo izvesti normalizacijo besed. Za besede, za katere vemo, da predstavljajo natanko en pomen, izberemo le en zapis (Opozorilo: S tem ne mislimo na sopomenke, vendar na iste besede, ki se drugače zapišejo.). Primer je restavracija *McDonalds*, ki jo lahko zapišemo kot *Mc Donald's* ali napačno zapisano *Mac Donald's*.

Da zmanjšamo število različnih besed in izboljšamo iskanje z upoštevanjem različnih skladenjskih oblik, iz besed izluščimo podstave ali besede postavimo v čimbolj osnovno obliko. Tega se lahko lotimo s hevurističnim postopkom, ki je večinoma pravilen in odstranjuje le predpone in pripone (ang. stemming). Znan hevurističen način za angleški jezik je Porterjev algoritem [28]. Drugi način upošteva pravila jezikoslovcev in morfološke analize (ang. lemmatization). Naš prvi dokument bi torej pretvorili:

težak | bridek | ura | sloves

Tako pridobljene besede moramo še ustrezno shraniti. To lahko storimo z uporabo inverznega indeksa. Za vsako besedo si hranimo seznam, v katerih dokumentih se pojavi, kolikokrat in na katerem mestu. Del indeksa za naš primer si lahko ogledate v tabeli 1.1.

biti:	(1,1,5),(3,1,4)
težak:	(1,1,[2])
bridek:	(1,1,[3])
ura:	(1,1,[4])
sloves:	(1,1,[6])
žnabel:	(4,2,[3,4])
oko:	(5,2,[4,6])
...	

Tabela 1.1: Inverzni indeks. Primer za zgornje besedilo (oznaka dokumenta, št. ponovitev, seznam položajev).

Indeks organiziramo tako, da lahko do želenega zapisa čim hitreje dostopamo.

Enak postopek, kot smo ga opisali zgoraj, izvedemo tudi na poizvedbi in nato poiščemo ustrezne dokumente glede na model priklica podatkov.

1.4 Pajek

Sistem za priklic podatkov ni uporaben, če ne posodablja sprememb, ki se dogajajo z dokumenti. Pajek (ang. crawler) je programski agent, ki avtomatsko išče nove in spremenjene dokumente ter jih dodaja v sistem.

Pajku, ki pregleduje spletne dokumente, podamo seznam spletnih mest, ki jih poznamo. Ta potem začne s pregledovanjem. Do novih dokumentov dostopa preko hiperpovezav. Spletne dokumente, do katerih obstajajo povezave iz drugih dokumentov, ki jih pajek lahko najde, uvrščamo v **površinski splet** (ang. surface web). **Globoki splet** (ang. deep web) predstavljajo dokumenti, ki jih običajni pajki ne najdejo. To so avtomatsko generirane strani, zasebne strani,... Težava se lahko delno reši tako, da pajek avtomatsko poizveduje z obrazci, ki so na voljo v dokumentih.

Pajek mora prioritetno izbirati dokumente, saj so nekateri za uporabnike bolj pomembni in bi jih želeli prej najti. Za vsak obiskani dokument mora določiti čas naslednjega obiska, da preveri morebitne spremembe vsebine. Upoštevati mora lastnosti strežnikov, saj če do nekega strežnika zelo pogosto dostopa, mu lahko strežnik prepove nadaljni dostop ali postane neodziven.

Martijn Koster je v sredini 1990-ih let predlagal protokol, ki omogoča spletnim administratorjem določati dokumente, ki jih pajki ne smejo pregledovati (ang. robot exclusion protocol). Večina komercialnih pajkov ta standard upošteva. Kasneje so v standard dodali še nekaj dodatnih ukazov za minimalni čas med dvema dostopoma do strežnika, frekvenco in čas obiskov. Nekateri pajki poznajo tudi *Sitemap* ukaz, s katerim podamo povezavo do dokumenta, ki opisuje celotno strukturo spletnega mesta. Ta ukaz je posledica poskusa reševanja problema globokega spleta. Protokol predvideva, da se datoteka *robots.txt* nahaja v korenskem imeniku spletnega mesta. Oglejmo si primer:

```
#robots.txt
User-agent: AnnoyingBot #ime pajka, * za vse

Allow:      /zasebno/index.html
Disallow:   /zasebno/

Disallow:   /temp/diploma.pdf
```

```
Request-rate: 1/10    # max. lahko prenese 1 stran na 10s
Visit-time: 0000-0600 # dovoli obiske le med 00:00 in 06:00 UTC

Sitemap: http://zitnik.si/sitemap.xml
```

Poglavje 2

Sistem iCORE

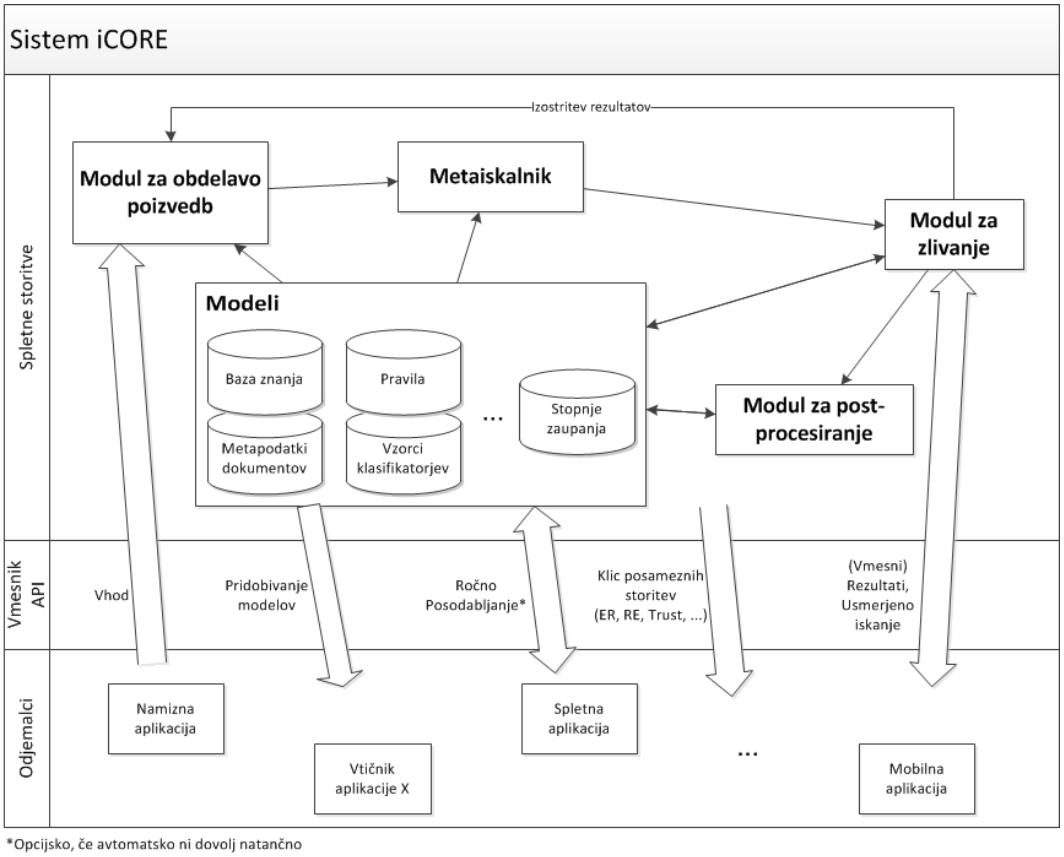
V prejšnjem poglavju smo spoznali osnove priklica podatkov in sisteme, ki so se gradili in uporabljali do sedaj. V tem poglavju bomo predstavili inovativen sistem **iCORE** - **i**ntelligent **CO**ntext-aware (multisource) information **RE**trieval.

Sistem iCORE je zasnovan na ideji, da razume uporabnika in njegove namere. Uporabimo ga lahko za pridobivanje informacij iz poljubnih zbirk podatkov, kot so svetovni splet ali zasebne zbirke podatkov. Sistem deluje kot skupina storitev, ki jih lahko uporabljajo odjemalci kjerkoli in kadarkoli. Uporabnik ni omejen, da uporablja sistem le na enem tipu naprave. Sistem želi delovati v čim večjem kontekstu za posameznega uporabnika, kot mu odjemalec omogoča. Sistem pri izbiri relevantnih dokumentov lahko izkoristi tudi vpliv družbene okolice na uporabnika. Osnovna skica sistema je prikazana na sliki 2.1.

2.1 Storitve

Osnovna storitev, ki jo sistem ponuja uporabniku, je odgovor na njegovo poizvedbo. Uporabniku so na voljo alternativni načini vnosa poizvedbe. Odgovor ni le seznam najbolj relevantnih dokumentov, ampak že strukturirano predstavljene informacije.

Sistem poizvedbo najprej analizira. Glede na uporabnikov kontekst poskuša z interno bazo znanja ugotoviti, kaj uporabnik želi. V kontekst se štejejo vsa uporabnikova dejanja, ki jih sistem zajema. Iz konteksta se iščejo vzorci z upoštevanjem dimenzij časa, uporabnikovega razpoloženja, obnašanja... Nato sistem ustrezno preoblikuje poizvedbo in izračuna, katere iskalnike se izplača uporabiti s pomočjo **modula za obdelavo poizvedb**. Prvotno ni mišljeno,



Slika 2.1: Skica sistema iCORE.

da bi sistem vseboval svoj iskalnik, ampak bi njegov iskalni modul deloval kot **metaiskalnik**. Sistem lahko uporablja tudi zasebne iskalnike, do katerih ima dostop uporabnik ali jih samodejno odkriva. Ko pridobi ustrezne rezultate večih iskalnikov, ki so navadno predstavljeni v obliki povezav do dokumentov, jih rangira glede na ustreznost. Pri tem si pomaga s statičnimi ocenami in dinamičnimi modeli, ki jih posodablja ves čas delovanja. **Modul zlivanja** rangirane dokumente sprocesira, da iz njih izlušči pomembne informacije, s katerimi lahko obogati znanje uporabnika. Sedaj so delni rezultati že pripravljeni za uporabnika. Ključne podkomponente modula za zlivanje so združevanje duplikatov, preverjanje stopnje zaupanja virov v določenem kontekstu in spletno rudarjenje. Sistem ugotovi, da lahko delne rezultate še izboljša z novimi poizvedbami, zato avtomatsko kreira novo poizvedbo in ponovi celoten postopek. Uporabniku sistem načeloma vrne več rezultatov, od katerih uporabnik izbere pravilnega oz. tistega, po katerem naj bi poizvedoval.

Uporabnik lahko izbere tudi bolj natančno nadaljno iskanje, če je bila njegova prvotna poizvedba presplošna. V zadnjem delu mora sistem še posodobiti modele za nadaljno uporabo. To je naloga **modula za post-procesiranje**, ki mu vhodne podatke posredujeta modul za zlivanje in morda tudi odjemalec. **Modeli** vsebujejo ocene virov, metapodatke o virih, stopnje zaupanja, podatke o uporabniku, bazo znanja,... Metapodatki o virih so navodila modulu zlivanja, ki določajo tipe podatkov v določenih poljih dokumenta. Pravilnost in stalna posodobljenost modelov je ključni del za uspešno delovanje sistema.

Ostale storitve, ki so na voljo uporabniku, so še dostop do baze znanja in modelov. Za spreminjanje modelov ali baze znanja potrebujejo uporabniki dodatno avtorizacijo. Uporabnik lahko sistemu doda svoj zasebni iskalnik, če ga le ta ne doda samodejno. Za enega ali več dokumentov po potrebi uporabi modul za zlivanje, ki mu izlušči entitete in jih obogati.

2.2 Vmesnik API

Sistem deluje kot storitev, zato ima definiran splošen vmesnik za komunikacijo z odjemalci. Odjemalci uporabljajo varno komunikacijo in prenašajo sporočila v standardiziranih formatih, kot sta XML ali JSON.

2.3 Odjemalci

Glavna naloga odjemalskih programov je, da izkoristijo svoje značilnosti za izgradnjo ustreznega uporabniškega vmesnika. Posamezni uporabnik značilno uporablja več naprav, zato sklepamo, da bo uporabljal tudi več različnih odjemalcev. Odjemalec na mobilnem telefonu lahko prebira SMS sporočila, obdeluje telefonske pogovore,... Odjemalec v spletnem brskalniku nadzoruje brskanje po spletnih straneh, uporabo anonimnega načina in s tem ustvarjanje dodatnega uporabnikovega konteksta,... Uporabniku moramo torej ponuditi, da ga bo sistem prepoznal ne glede na to, kje ga uporablja.

Za odjemalca je pomembno, kakšne možnosti ponuja za vnos uporabnikove poizvedbe. Ali je to le niz znakov, prilaganje celotnega dokumenta, vpis znanja o neki entiteti z določenimi atributi, slika, zvok? Vnosu nato sledi prikaz in navigacija med rezultati, dodatno filtriranje ali uporaba za nadaljno iskanje.

Trenutno sistemi za priklic podatkov še vedno uporabljajo vnosno polje za podajanje poizvedbe. V prihodnosti se bo to spremenilo, saj se s to težavo ukvarja novo področje o inteligentnih uporabniških vmesnikih. Ali bi nam sistem za priklic podatkov lahko vrnil rezultate, še preden mu podamo poizvedbo?

Sistem iCORE bo poskušal uporabiti vse zgornje načine za zajem uporabnikovega obnašanja. Naučil se bo ključnih vzorcev, da bo znal sestaviti poizvedbo in mu predstaviti rezultate, še preden bi poizvedoval sam uporabnik. Četudi uporabnik morda včasih ne želi poizvedovati določenih stvari, je uporabno, da se mu prikazujejo relevantne informacije glede na njegovo početje, saj so lahko njegove informacije napačne.

Pomemben je seveda tudi prikaz rezultatov. Današnji iskalniki za podano poizvedbo vračajo le seznam zadetkov, iz katerih mora uporabnik sam pridobiti informacije. Sistem iCORE bo uporabniku prikazal že zgrajene informacije v primerni obliki. Pri iskanju oseb bo osebo pokazal z vsemi atributi, ki jih bo našel (ime, priimek, datum rojstva, naslov, zaposlitev,...) in povezavami z ostalimi osebami ali predmeti, s katerimi je v relaciji. Uporabnik bo imel možnost sprehajanja po grafu, ki se bo avtomatsko dopolnjeval, saj sistem zna avtomatsko kreirati poizvedbe. Na enak način bodo prikazane tudi entitete drugih tipov.

Odjemalec je najbližje uporabniku in z beleženjem njegovih dejanj lahko pomaga sistemu povečevati uporabnikov kontekst. V kontekst spadajo vsi podatki, ki jih sistem iCORE lahko uporabi za izboljšanje priklica podatkov. Poleg avtomatskega zajemanja konteksta bo imel uporabnik na voljo tudi ročen vnos in hiter dostop do usmerjenega iskanja. Usmerjeno iskanje (ang. faceted search) omogoča izostritev pogojev poizvedbe, če je za uporabnika na voljo preveč različnih rezultatov.

2.4 Primeri uporabe

Poslovna uporaba: Nenapisano pravilo pravi, da je v poslovnih okoljih 80% informacij pridobljenih iz nestrukturiranih virov. Poslovni analitik Russom je v svoji raziskavi [35] pokazal, da je delež pol- in nestrukturiranih virov v organizaciji 53% in se bo v primerjavi s strukturiranimi še povečeval. Sistem iCORE bo omogočal enostavno pridobivanje in prikazovanje rezultatov iz intraneta, različnih dokumentnih sistemov, e-poštnih sporočil in dokumentov na pomnilnih medijih. Dodana vrednost bodo natančne informacije. Zaposleni bodo lahko hitreje izdelovali poročila, predstavitve, saj jim bo dostop do ustreznih informacij zelo olajšan. Podoben sistem, namenjen namiznemu založništvu za hitro sestavljanje novih člankov, je WorldColor [1]. Uporablja storitve strežnika

[1] http://www.worldcolor.com

MarkLogic [2], ki vsebuje označeno lokalno bazo dokumentov.

Organi pregona, odkrivanje goljufij: Storilci kaznivih dejanj uporabljajo svetovni splet in priljubljena socialna omrežja. Ne zavedajo se še, da jih lahko s pomočjo tega hitro najdemo. Interpol je letos uspešno izvedel akcijo iskanja "kriminalcev" [34]. Prosil je prebivalstvo, da so bili na spletu pozorni na sumljive osebe, statuse na socialnih omrežij, s takšnim načinom so uspešno našli nekaj iskanih oseb. V prihodnosti so zato napovedali razvoj sistema, ki bo lahko takšna iskanja opravljal sam - bi to počel sistem iCORE? Podoben primer je odkrivanje goljufij, pri katerem lahko izboljšane informacije pomagajo odločiti o potencialni goljufiji. V primeru avtomobilskih nesreč bi sistem iCORE lahko ugotovil, če je osumljenec objavljal kakšne slike, javno komentiral dogodek ali pa se nesreča dejansko sploh ni zgodila.

Osebna uporaba: Sistem iCORE bo predstavil nov način iskanja. V primeru uspešnosti ga bodo uporabniki zamenjali za tradicionalne iskalnike, ki bodo lahko posledično zaradi tega uporabili podoben pristop.

Inovativna uporaba: Zamislimo si pametno informatizirano kuhinjo. Kuhinjski informacijski sistem se "zaveda" izdelkov, ki jih hranimo v hladilniku, shrambi, kuhinjskih omarah. Imamo tudi senzorje, ki beležijo, kam postavimo oz. moramo postaviti krožnik, pribor,... Nad delovno površino se nahaja zaslon na dotik. Na tem zaslonu lahko izbiramo recepte, kaj želimo jesti in podobno. Dodan je še socialni pridih, povezujemo se s prijatelji, sosedi, sistem nam mogoče razkrije, kaj imajo oni za kosilo. Sistem iCORE bi uporabili tako, da bi uporabnika obveščali o potencialnih novostih pri receptih. Če bi uporabnik hotel narediti novo jed, ki je postala modna, bi jo le vpisal in iCORE bi mu našel potrebne vsebine – sestavine, recept in ostalo. Iz recepta bi razbral postopek in nas vodil ter opozarjal med pripravo hrane. Sestavil bi nakupovalno košarico in nato predlagal, kdaj in v katero trgovino se splača iti po sestavine.

[2] http://www.marklogic.com

Poglavje 3

Komponente sistema iCORE

V prejšnjem poglavju smo na kratko predstavili osnovni koncept delovanja sistema iCORE. V tem poglavju bomo podrobneje pregledali vse komponente sistema, predstavili njihovo delovanje in predlagali primerne izvedbe.

3.1 Modul za obdelavo poizvedb

Poizvedba v splošnem predstavlja uporabnikovo izražanje zahteve po informaciji. Preko poizvedbe uporabnik predstavi del svojega trenutnega znanja, ki ga bogati pri konvergiranju do želenega cilja. Formalni poizvedovalni jeziki zahtevajo, da se uporabniki držijo sintaktičnih pravil. Dodatno mora biti struktura dokumentov znana uporabniku, zaradi česar sistem postane neprilagodljiv. Sistem iCORE s svojim vmesnikom ponuja odjemalcem več načinov vnosa poizvedb.

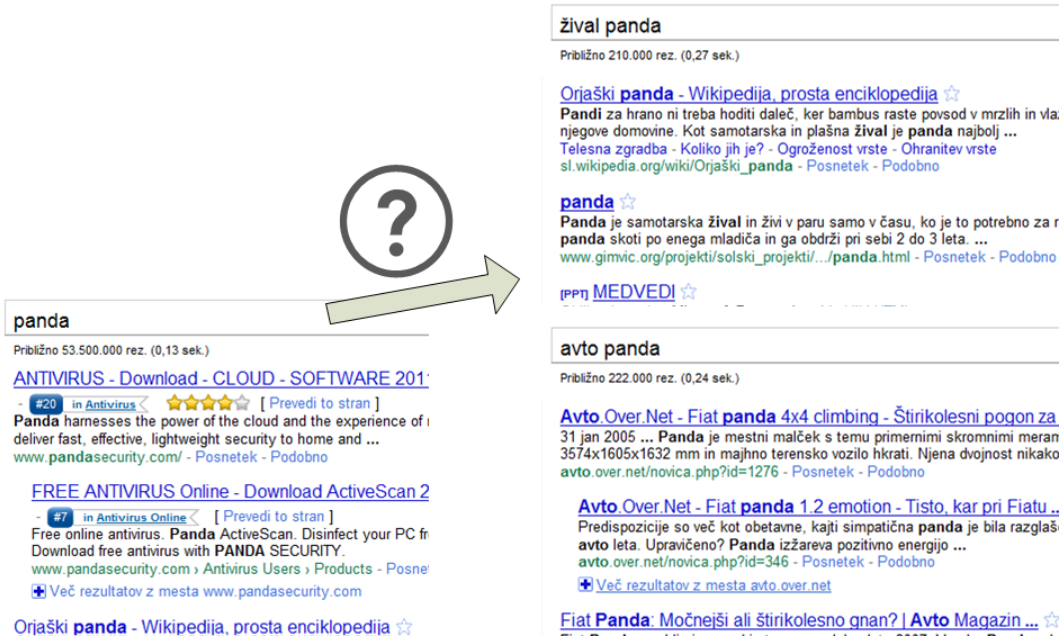
Poizvedbe glede na način podajanja delimo na:

Poizvedovanje s ključnimi besedami: Je najlažje in med uporabniki najbolj uporabljan način. Poizvedba je lahko sestavljena iz **ene besede** ali jo kontekstno razširimo, t.j. dodamo besede, ki naj bi se nahajale v bližini. **Večbesedne** poizvedbe lahko sestavljamo tudi z logičnimi vezniki AND, OR in NOT. To so **logične** poizvedbe, ki niso namenjene rangiranju dokumentov po ustreznosti, saj lahko dokument le ustreza poizvedbi ali ne. Da bi uporabnika popolnoma razbremenili, lahko vnese besede tudi v **naravnem jeziku**. Sistem mora v tem primeru poizvedbo obdelati in določiti pomembne dele, da lahko ustrezno rangira pridobljene rezultate (več o tem v 3.2).

Ujemanje z vzorci: Uporabnik poda poizvedbo kot regularni izraz ali določi omejitve. To je posplošen zgornji način podajanja. Rezultate lahko omejimo s podnizi, obsegom, določimo razdaljo maksimalne napake ali definiramo natančen vzorec.

Strukturirane poizvedbe: Nekatere zbirke podatkov vsebujejo dokumente, ki so določene oblike. Uporabnik lahko za vsak del definira svojo poizvedbo. Knjižničarski sistemi uporabljajo ta način za iskanje knjig, saj so za vsako publikacijo določeni ustrezni metapodatki.

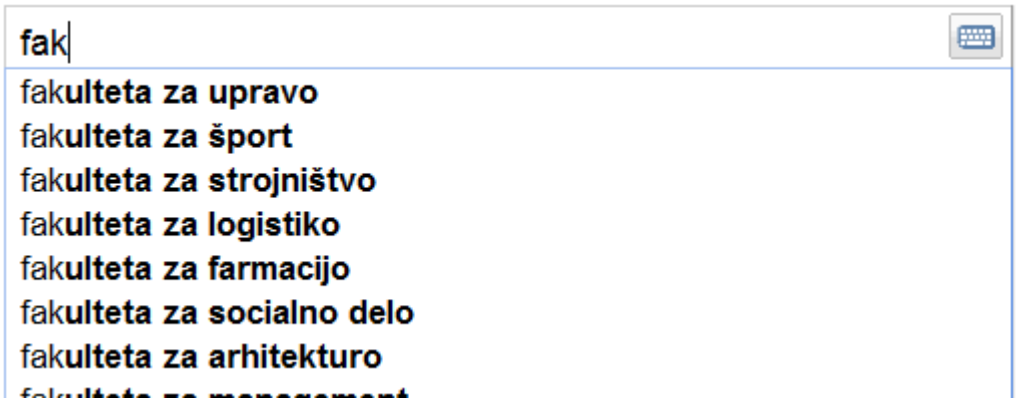
Pozvedovalni protokoli: Ta tip poizvedb imenujemo protokoli, ker jih ne pišejo ljudje, ampak jih uporablja programska ali strojna oprema. Za protokol mora biti poleg sintakse definiran še način vzpostavljanja povezave, vzdrževanja seje in izmenjave podatkov. Sem spadajo protokoli za dostop do pomnilniških virov.



Slika 3.1: Prikaz dvoumnosti poizvedbe. Uporabnik je podal niz *panda*, ki ima lahko več pomenov. Sistem mora avtomatsko ugotoviti ali je mislil na žival, vozilo, programsko opremo ali kaj drugega ter mu to prednostno ponuditi.

Sistem iCORE je kontekstualno orientiran. Uporabnika spremlja in poskuša prilagoditi njegovo poizvedbo, da bo ustrezala njegovim potrebam. Med rezultati lahko nastopa tudi več različnih skupin dokumentov, ki so vsi relevantni,

Slika 3.2: Napaka pri vnosu poizvedbe. Sistem mora ugotoviti, da vrnjeni dokumenti niso dovolj primerni za poizvedbo. Poskusiti mora predlagati novo poizvedbo, ki je glede na določeno razdaljo blizu podane. Na sliki opazimo, da je sistem obravnaval vsako besedo posebej in ne skupaj besedne zveze.



Slika 3.3: Samodokončevanje poizvedb. Sistem mora že med podajanjem poizvedbe poskušati ugotoviti uporabnikov namen in predlagati poizvedbo.

a med seboj nepovezani. Sistem mora ugotoviti, katero skupino je uporabnik mislil in mu jo prednostno predstaviti (primer na sliki 3.1). Pravilno formuliranje ali uporaba ključnih besed je za uporabnika zahtevno opravilo. Če sistem ne zna ugotoviti uporabnikovega namena, mu lahko poleg rezultatov ponudi več različnih poizvedb, katere so morda prave (primer na sliki 3.2). Vsak sistem za priklic podatkov mora upoštevati sopomenke v poizvedbi. Enostaven primer bi bila poizvedba *sožitje*, za katero so relevantni tudi dokumenti, ki uporabljajo besedo *simbioza*. Uporabniku v interaktivno pomoč moramo ponuditi tudi avtomatsko samodokončevanje poizvedbe (primer na sliki 3.3).

Metode, ki se uporabljajo za prilagajanje poizvedb, delimo na lokalne in globalne. Vsak tip metode lahko zahteva interakcijo z uporabnikom ali deluje avtomatsko. Osnovne tehnike, ki jih uporabljajo za spreminjanje poizvedb, so

razširjanje poizvedb z novimi besedami ali ustrezno uteževanje delov poizvedbe.

Cilj modula za obdelavo poizvedb v sistemu iCORE je ustrezna prilagoditev poizvedb. Ugotoviti moramo različne pomenske skupine, kamor poizvedba lahko spada in vsako obravnavati posebej. Poizvedbe s pomočjo uporabnikovega konteksta prilagodimo za vsako skupino posebej in jih posredujemo modulu metaiskalnik.

3.1.1 Lokalne metode

Ideja lokalnih metod je, da uporabnik nad majhnim delom rezultata označi, kateri dokumenti so relevantni, kar se uporabi za nadaljno iskanje. Postopek lahko poteka v več iteracijah, dokler uporabnik ni zadovoljen z rezultatom. Takšnemu načinu lahko rečemo tudi **usmerjeno iskanje**, saj uporabnik vodi sistem, metodam pa **metode za odziv ustreznosti** (ang. relevance feedback). Osnovni postopek je sledeč:

1. Uporabnik vnese kratko, enostavno poizvedbo.
2. Sistem vrne omejen nabor rezultatov.
3. Uporabnik označi rezultate kot relevantne in nerelevantne.
4. Sistem glede na izbiro pridobi boljše rezultate in jih prikaže.

Sistem osnovni poizvedbi dodaja besede, za katere ugotovi, da so ključne v relevantnih dokumentih. V ta namen lahko uporabi frekvence besed ali katero od mer, kot je TF-IDF (predstavljena v 1.2).

Klasični algoritem za implementacijo odziva ustreznosti, je Rocchio algoritem. Razvit je bil za vektorski model priklica podatkov, ki smo ga spoznali v 1.2. Prvi ga je leta 1971 predstavil Gerard Salton v sistemu za priklic podatkov SMART. Predstavljajmo si, da imamo dokumente postavljene v vektorskem prostoru. Iščemo vektor $\vec{q}$, ki bo maksimiziral podobnost z relevantnimi dokumenti in minimiziral podobnost z nerelevantnimi. Če C_r predstavlja množico relevantnih dokumentov in C_{nr} množico nerelevantnih v celotni zbirki dokumentov sistema, lahko zapišemo enačbo za optimalen vektor $\vec{q}_{opt}$. Za funkcijo sim uporabimo kosinusno razdaljo.

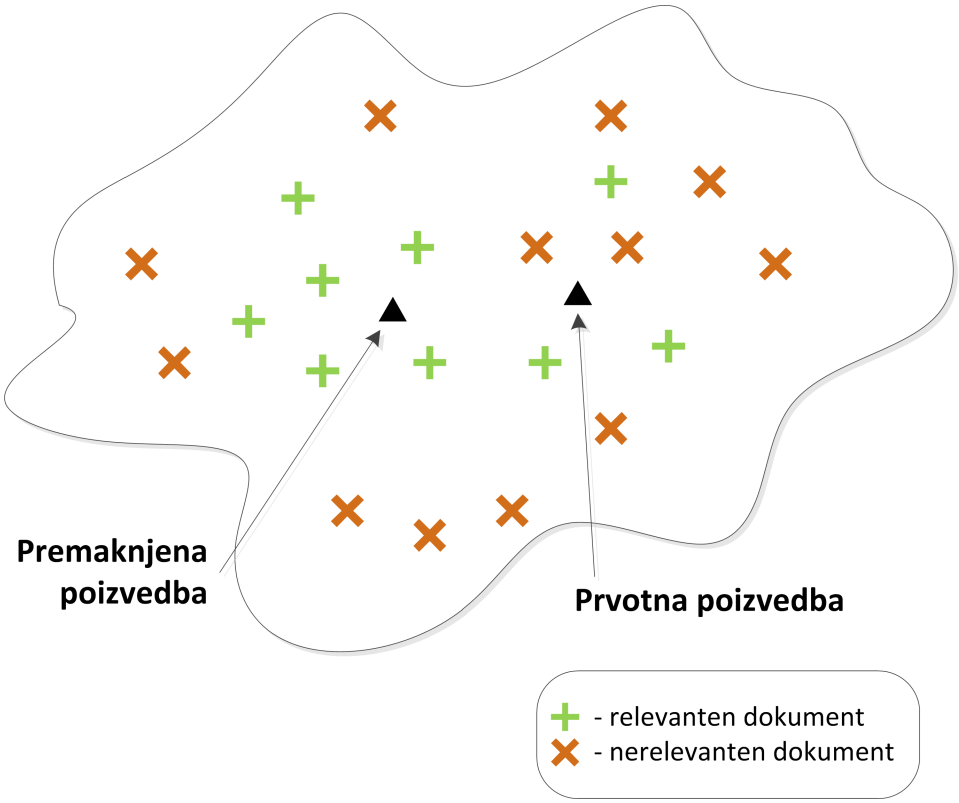
$$\mathrm{sim}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} \tag{3.1}$$

$$\vec{q}_{opt} = \arg\max_{\vec{q}} [\mathrm{sim}(\vec{q}, C_r) - \mathrm{sim}(\vec{q}, C_{nr})] \tag{3.2}$$

V tem primeru smo gotovo našli najboljšo rešitev, a bi uporabnik moral pregledati celotno zbirko dokumentov, kar zaradi velikosti zbirke ni izvedljivo. Zato algoritem za osnovo vzame vektor prvotne poizvedbe $\vec{q}_0$ in ga nato premakne v smeri centroida relevantnih D_r oziroma nerelevantnih D_{nr} dokumentov. Pri tem so D_r in D_{nr} le iz majhnega dela rezultatov, ki jih je uporabnik označil. Z utežmi α, β in γ določamo kateri del se bolj upošteva.

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j \tag{3.3}$$

Pozitivni odziv je več vreden kot negativni, zato naj bo $\beta > \gamma$. V praksi so se za pametno izbiro izkazale vrednosti $\alpha = 1$, $\beta = 0,75$ in $\gamma = 0,15$. Na sliki 3.4 lahko vidimo delovanje algoritma na enostavnem primeru.



Slika 3.4: Rocchio algoritem. Algoritem predlaga bolj primeren vektor glede označene dokumente.

Sistem lahko izvede tudi avtomatsko lokalno analizo. Najlažja rešitev je psevdo-odziv ustreznosti, kjer proglasimo najtežjih k dokumentov kot rele-

vantnih. Pri implicitnem odzivu ustreznosti poskusimo v ocenjevanje relevantnosti vključiti tudi kontekst uporabnika. S pomočjo njegovih zadnjih iskanj zgradimo klasifikator in po relevantnosti klasificiramo dokumente. Podobno lahko storimo z uporabo nenadzorovanega učenja, kjer klasificiramo ciljno skupino.

Z nenadzorovanim učenjem lahko rezultate razdelimo v skupine (ang. clustering). Nato glede na podatke, ki jih imamo o uporabniku, določimo, katera skupina je ustrezna. Podoben primer ročne analize je storitev SearchPoint [2]. Le ta uporabniku prikaže vse rezultate za podano poizvedbo. SearchPoint iz rezultatov pridobi skupine in jih prikaže uporabniku, ki s točko izbira, katera skupina ga bolj zanima. Ob premiku točke se znova izračunajo razdalje do skupin in rezultati poizvedbe se ponovno razvrstijo po relevantnosti.

3.1.2 Globalne metode

Za globalne metode je značilno, da uporabljajo celoten nabor dokumentov sistema za prilagoditev poizvedb. Za delovanje uporabljajo slovar besed. Vsebovati mora sopomenke, asociacije, fraze, v katerih se pojavljajo besede... Zgrajen je lahko ročno ali avtomatsko. Ročno vzdrževanje takšnega slovarja je zaradi velike količine besed nesmiselno, poleg tega ga je treba stalno posodabljati. V modernih sistemih se zato uporablja avtomatska gradnja slovarja.

Pri avtomatski gradnji slovarja sistem pregleda celotno zbirko vsebovanih dokumentov in zbere vse uporabljene besede. Med njimi mora izračunati vrednost podobnosti. To vrednost izračuna glede na pojavitev besed v skupnih dokumentih. Ko uporabnik v sistem vnese poizvedbo, jo sistem poskuša razširjati z besedami, ki so v bližini.

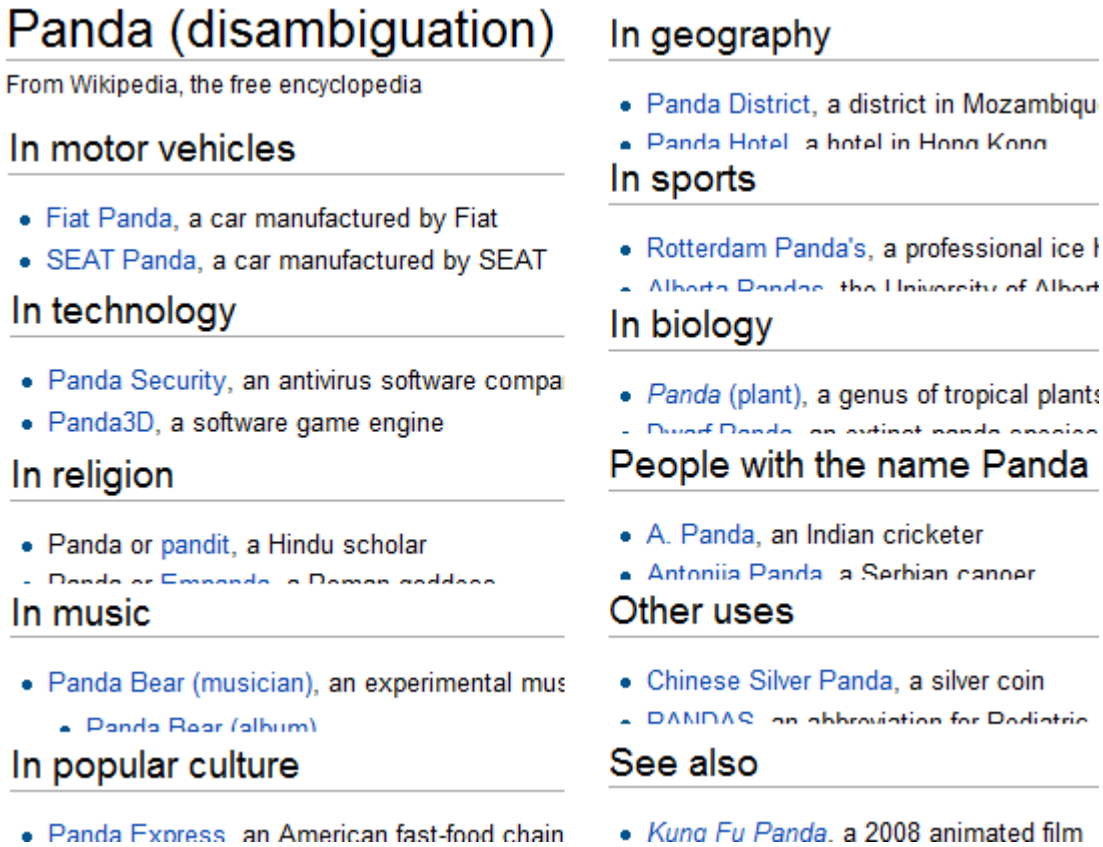
3.1.3 Metode v iCORE

V sistemu iCORE je izbran drugačen način, sicer podoben globalnim metodam. Sistem poskuša spremljati uporabnika tako, da si hrani njegove pretekle poizvedbe, izbiro rezultatov ali druge podatke, ki jih lahko odjemalec zajema. S pomočjo teh ob novi poizvedbi poskuša čimbolj natančno klasificirati, kaj uporabnik želi ter mu s tem omogočiti kontekstualno iskanje. To stori s pomočjo baze znanja, ki jo modul za post-procesiranje tudi posodablja. To nam ne koristi le za prilagajanje poizvedbe, ampak tudi za razumevanje uporabnika, ker se lahko na podlagi tega modul metaiskalnik odloča, kateri iskalnik sploh uporabiti. Odvisno od uporabljanih iskalnikov, se dogaja, da bo za vsak interni iskalnik uporabljena drugačna poizvedba. Če sistem ugotovi,

da uporabnik išče film iz leta 1901, v katerem nastopa igralec A in B, lahko to uporabi za izpolnjevanje polj v strukturiranem iskalniku filmov.

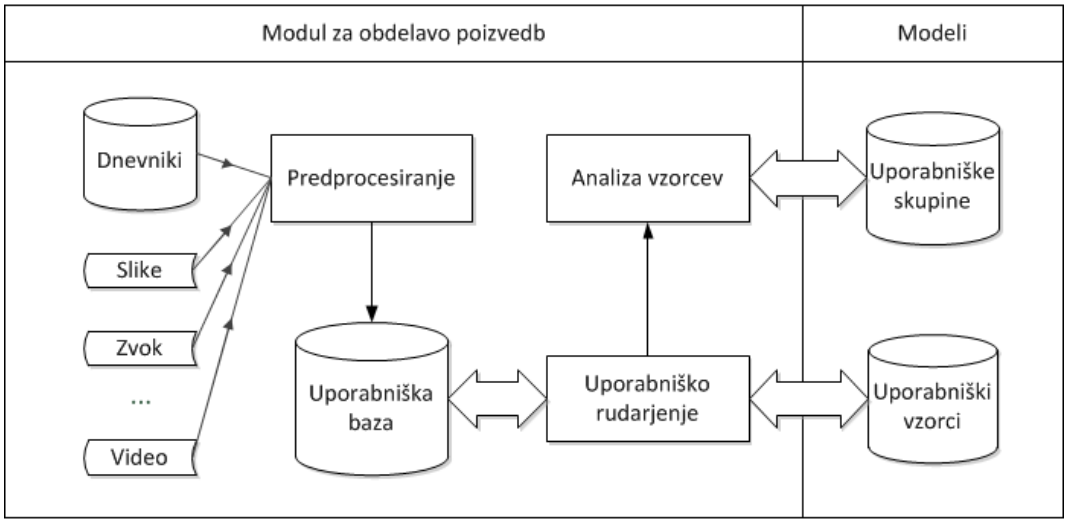
Sistem iCORE uporablja dve bazi znanja. Prva je namenjena ugotavljanju tipa rezultata uporabnikove poizvedbe in poleg sheme vsebuje podatke. Druga baza znanja hrani le shemo.

Koncept uporabe prve baze znanja je predstavil Jian Hu s sodelavci [3]. Ideja temelji na uporabi spletne enciklopedije Wikipedija. Wikipedija je urejena po kategorijah in vsebuje članke za skoraj vsa področja. Za dvoumne besede vsebuje posebne strani z možnimi pomeni, kot je predstavljeno na sliki 3.5. Za izbiro sorodnih pojmov sistem izbere nekaj ključnih besed in poskuša zgraditi graf, kjer so te besede predstavljene v vozliščih in povezane s sorodnimi glede na povezave med članki na Wikipediji. Pri testiranju takšnega načina ugotavljanja uporabnikovega namena za tri različne domene so ugotovili, da metoda izboljša rezultate.



Slika 3.5: Pomeni besede *panda* na Wikipediji.

Druga baza znanja je ontologija, ki ne vsebuje podatkov, ampak le shemo, t.j. entitne tipe, njihove atribute in relacije med njimi. Za priklic podatkov je to uporabno, ker želimo uporabniku rezultate predstaviti strukturirano in ne le kot zbirko relevantnih dokumentov. Namesto obeh baz znanja bi lahko uporabili kar DBpedijo. To je ontologija, ki vsebuje označene podatke iz Wikipedije. Praktično uporabo DBpedije je za namene televizijske hiše predstavil Georgi Kobilarov [4].



Slika 3.6: Izgradnja konteksta uporabnika. Sistem iCORE bo spremljal uporabnika in omogočal čimbolj personaliziran pristop.

Za grajenje konteksta o uporabniku je za sistem uporabno spremljanje njegove interakcije z računalnikom ali drugo napravo. Osnovni koncept izgradnje je prikazan na sliki 3.6. Izgradnja uporablja principe uporabniškega spletnega rudarjenja. Običajno, če pišemo neko besedilo ali beremo kak dokument, nas določena malenkost bolj zanima in jo hočemo raziskati. Podobno lahko sklepamo tudi z zvokom ali telefonskimi pogovori. Uporabimo lahko tudi uporabnikovo obrazno mimiko, podobno kot obstajajo modeli, ki glede na kupca pred izložbo trgovine prikažejo ustrezen oglas. Prav tako so nam na voljo dnevniški zapisi strežnikov. Iz njih lahko izluščimo uporabnikove seje, čase ogledovanja določenih dokumentov in povezave, ki jih je najprej izbral (ang. clickthrough data). Vse te podatke, ki jih pridobimo, s predprocesiranjem pretvorimo v ustrezen (standardno tabelaričen) zapis in shranimo v uporabniško bazo. Nad takšnimi podatki nato odkrivamo zakonitosti in jih pomnimo v uporabnikovih vzorcih. Določene uporabnike po obnašanju lahko

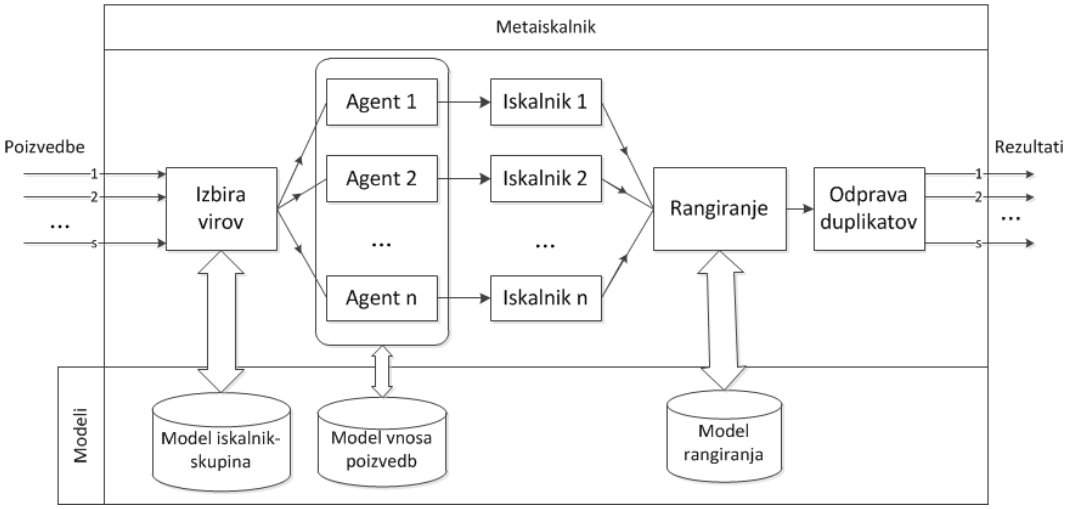
porazporedimo v skupine. Zaradi tega vzorce enega uporabnika primerjamo z ostalimi in ga poskušamo z nenadzorovanim učenjem uvrstiti v eno ali več skupin. Pri novih klasifikacijah lahko nato uporabimo tudi rezultate skupin, v katere spada, če uporabniški vzorci niso dovolj prepričljivi.

3.2 Metaiskalnik

Metaiskalnik je iskalnik, ki uporablja dva ali več drugih iskalnikov, ter predstavi njihove rezultate. Uporabljajo se, ker eden iskalnik ne vsebuje vseh primernih dokumentov. S tem poskušamo uporabniku olajšati ročno iskanje z več iskalniki. Ryen s sodelavci je v raziskavi [5] pokazal, da 36% uporabnikov v kratkem času uporabi več iskalnikov. 12% vpisanih poizvedb je istih na različnih iskalnikih. Metaiskalniki se razlikujejo tudi po različnih funkcijah za rangiranje končnih rezultatov. Cilj vseh je izboljšanje rezultatov in izboljšanje pristranosti (ang. bias). V preteklosti so se že zelo zgodaj pojavili sistemi, ki so združevali rezultate iz različnih zbirk podatkov, eden prvih spletnih metaiskalnikov pa je MetaCrawler iz sredine 90-ih let [6].

Tudi sistem iCORE vsebuje komponento metaiskalnik. Shemo njenega delovanje si lahko ogledamo na sliki 3.7. Komponenta od modula za obdelavo poizvedb dobi s seznamov poizvedb za vsako skupino posebej in tudi vrne s seznamov zadetkov. Potem mora glede na poizvedbo izbrati ustrezne iskalnike, ki naj bi vsebovali relevantne dokumente - izbira virov (ang. source selection). Za to skrbi klasifikator glede na trenutno stanje modelov. Podobno je bil zasnovan sistem ProFusion [7], ki je imel klasifikator za vsak vsebovan iskalnik posebej. Za eno uporabnikovo poizvedbo lahko modul za obdelavo poizvedb oblikuje več specifičnih poizvedb, ki jih lahko prejmejo različni iskalniki. To lahko vrne mnogo dokumentov, ki bi jih moral nato modul za zlivanje obdelati. Zaradi tega pred posredovanjem rezultatov tudi ustrezno rangira pridobljene dokumente in ohrani le njihov omejen nabor. Značilne algoritme za rangiranje bomo predstavili v nadaljevanju. Med dokumenti se lahko pojavljajo tudi duplikati, ki jih moramo odstraniti. Povsem globalna razvrstitev dokumentov ni zadostna za iCORE. Kot smo že omenili, iCORE beleži akcije uporabnika in mu poskuša ponuditi čimbolj personalizirane rezultate. To pomeni, da moramo za dvoumne poizvedbe, kjer so dokumenti v več različnih pomenskih skupinah, za vsako skupino imeti svoj nabor rezultatov. Modul za obdelavo poizvedb bo že vedel ciljno skupino, zato se bo zanjo pridobilo več dokumentov in za ostale manj. Ker klasifikacija uporabnikovega namena ni vedno točna, mu bomo lahko predstavili tudi alternativne odločitve. Nabori dokumentov

za vsako skupino se bodo posredovali modulu za zlivanje na sliki 3.3. Med delovanjem bo modul metaiskalnik imel možnost avtomatskega prepoznavanja in dodajanja novih iskalnikov.



Slika 3.7: iCORE metaiskalnik. Prikaz delovanja komponente v sistemu iCORE.

Modul metaiskalnik v sistemu bo zelo prilagodljiv. Ker bo sistem iCORE lahko deloval tudi samostojno znotraj neke organizacije, bodo imeli razvijalci možnost razviti ali uporabiti kakšen iskalnik za lastne namene. Intranetni sistemi velikih podjetij ponavadi vsebujejo slabe iskalne mehanizme. Enako velja za dokumentne ali druge informacijske sisteme, ki jih uporabljajo.

V nadaljevanju si oglejmo postopke za rangiranje dokumentov.

3.2.1 Postopki za rangiranje dokumentov

Prvotni iskalni sistemi so uporabljali le relevantnost dokumentov za vračanje rezultatov. Kmalu se je na spletu nabralo ogromno vsebin, veliko dokumentov je bilo zavajujočih (ang. spam), zato je bilo treba poiskati načine za obvladovanje teh problemov. Prvotne metode so bile statične. Sistem za priklic podatkov je vsakemu dokumentu priredil globalno oceno. Kasneje se je izkazalo, da je odziv uporabnikov pomemben pri rangiranju in razvili so se dinamični modeli.

Statično ocenjevanje

Statično ocenjevanje oceni vse dokumente, ki jih vsebuje sistem za priklic podatkov z določeno oceno. Navadno se to ocenjevanje izvaja kot paketna obdelava podatkov (ang. batch processing). Ustrezni algoritmi spletnih iskalnikov za ta namen so se razvili iz analize hiperpovezav (ang. link analysis) med dokumenti. Raziskave izvirajo iz analize socialnih omrežij, ki jih lahko predstavimo z grafi. Dokumente predstavimo kot vozlišča, hiperpovezave pa kot usmerjene povezave med dokumenti. Za vsak dokument lahko izračunamo število dokumentov, s katerimi je povezan. Če število povezav iz dokumenta i normaliziramo s številom vseh dokumentov, dobimo centralnost C(i), ki lahko predstavlja zelo naivno oceno:

$$C(i) = \frac{\text{število izhodnih povezav}}{\text{število vseh dokumentov}} \tag{3.4}$$

Pomembno je tudi, koliko in kateri dokumenti kažejo na določen dokument. Posplošeno si lahko to predstavljamo kot glasovanje za Nobelovega nagrajenca. Glas znanstvenika je bolj pomemben kot glas navadnega človeka ali pa je enakovreden skupnemu glasu večje množice ljudi. Ta princip uporablja večina znanih algoritmov za statično ocenjevanje. Malo verjetno je, da bi pomemben znanstvenik glasoval za nekompetenčnega kandidata. To v boju proti zavajujočim dokumentom uspešno izkorišča algoritem PageRank [8], ki ga bomo predstavili.

Naj bo i dokument v sistemu. Množico dokumentov, ki kažejo na i naj predstavlja $V(i)$. Množico dokumentov, na katere kaže i pa $I(i)$. Potem je ocena PageRank PR za dokument i:

$$PR(i) = \sum_{j \in V(i)} \frac{PR(j)}{|I(j)|} \tag{3.5}$$

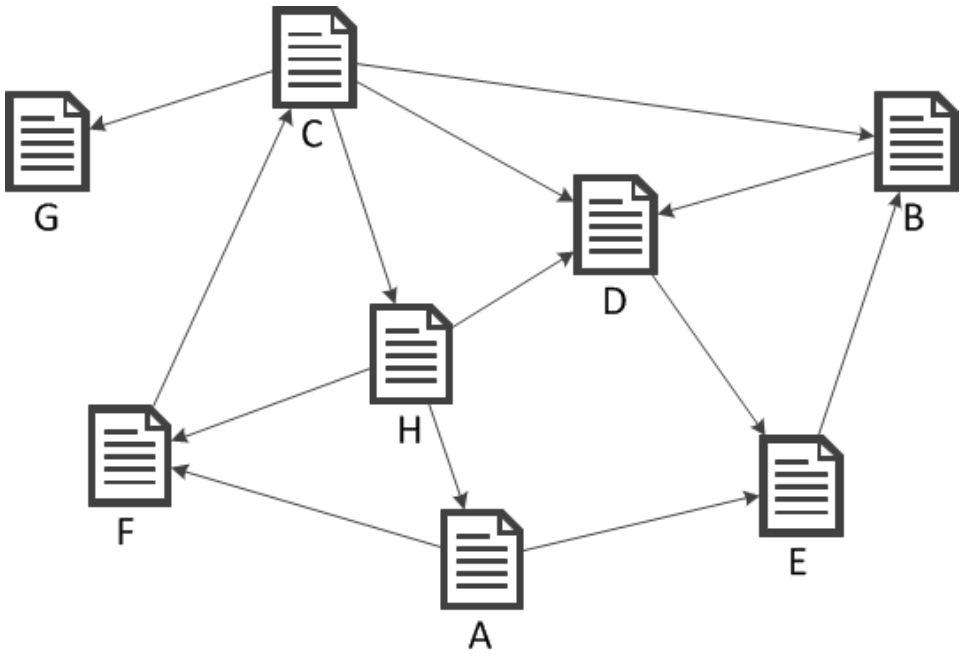
Računanje PageRank vrednosti poteka iterativno, pri čemer si lahko za začetno porazporeditev izberemo naključne ocene. Pri takšnem izračunu se lahko pojavi problem, če med dokumenti obstaja cikel, kjer so povezani le med sabo. Potem se pri iteraciji njihove ocene povečujejo do neskončnosti. Ker v izpeljavi prevedemo problem na iskanje stacionarne porazdelitve v markovski verigi, mora biti naš graf krepko povezan. To pomeni, da mora med vsakima dvema vozliščema v grafu obstajati pot. Stacionarno porazdelitev iščemo z naključnim sprehodom po grafu. Predstavljajmo si uporabnika, ki brska po

spletu in izbira naključne povezave. Če na spletni strani ni nobene povezave ali želi pregledati drugo vsebino, lahko sam enostavno vpiše naslov druge spletne strani. Obema omejitvama bomo zadostili tako, da dodamo povezave med vsemi dokumenti, med katerimi ne obstajajo. Predpostavimo, da imamo N dokumentov. Prehodna matrika markovske verige, je oblike:

$$P_{ij} = \begin{cases} \frac{1}{N} & |I(i)| = \emptyset \\ \frac{1}{|I(i)|} & \text{obstaja povezava od } i \text{ do } j \\ 0 & \text{sicer} \end{cases} \tag{3.6}$$

Sedaj uporabimo enačbo 3.5 s prehodno matriko P^T in faktorjem d. S tem faktorjem nadziramo odločitev, ali uporabnik nadaljuje pot po povezavah ali odide na naključno spletno stran.

$$PR(i) = (1 - d) + d \sum_{j \in V(i)} P_{ji} PR(j) \tag{3.7}$$



Slika 3.8: PageRank primer.

Iteracija	A	B	C	D	E	F	G	H
1	1	1	1	1	1	1	1	1
2	0,539	1,318	1,106	1,602	1,531	0,964	0,468	0,468
3	0,389	1,792	1,076	1,745	1,84	0,618	0,491	0,491
4	0,395	2,055	0,781	2,148	1,904	0,560	0,48	0,48
5	0,393	2,041	0,732	2,306	2,250	0,561	0,422	0,422
...	...	...	...	...	...	...	...	...
20	0,371	2,864	0,706	2,9	2,920	0,529	0,406	0,406
21	0,371	2,888	0,706	2,956	2,931	0,529	0,406	0,406
...	...	...	...	...	...	...	...	...
28	0,371	2,924	0,706	3,004	2,961	0,529	0,406	0,406
...	...	...	...	...	...	...	...	...
50	0,371	2,940	0,706	3,021	2,981	0,529	0,406	0,406

Tabela 3.1: Računanje PageRank ocen. Primer iz slike 3.8 s faktorjem d=0,85. Pomembnost dokumentov v padajočem vrstnem redu: D,E,B,C,F,G,H,A.

Primer računanja PageRank ocen za primer s slike 3.8 si lahko ogledamo v tabeli 3.1. Iteriranje se konča, ko se vrednosti ocen ne spreminjajo več veliko ali skonvergirajo. Predlagatelji [8] algoritma poročajo, da je za 322 milijonov povezav velik graf potrebnih približno 52 iteracij. Podjetje Google za faktor d uporablja vrednost 0,85. Vrednosti ocen po končanem algoritmu so torej lahko med 0,15 in neskončno. Njihova storitev PageRank te ocene razvrsti v cela števila med 0 in 10.

Podobno znan je algoritem HITS [9], ki lahko rangira dokumente glede na temo poizvedbe in se bolj zanaša na izhodne povezave. Njegova glavna slabost je neodpornost proti zavajujočim dokumentom. PageRank je uspešen tudi zaradi uspešnega poslovnega modela iskalnika, ki je bil razvit z namenom njegovega testiranja. Kasneje je bil razvit algoritem SALSA [11], ki poskuša združiti dobre strani PageRank-a in HITS-a.

Dinamično ocenjevanje

Z razvojem iskalnikov so se za rangiranje uporabljali verjetnostni modeli. S stališča uporabnika je rangiranje razporeditev dokumentov v zanj pomembnem vrstnem redu. Glede na način, kako algoritmi ocenjujejo dokumente, ločimo načine:

Točkasto (ang. pointwise): V množici dokumentov izberemo tistega, ki je najbolj relevanten. S ponavljanjem dobimo absolutno urejenost.

Parno (ang. pairwise): Za vsak par dokumentov določimo, kateri dokument je boljši. Dokumente lahko nato relativno razvrstimo. Metoda nam zagotavlja, da dobimo pravilno urejenost, čeprav se lahko zgodi, da kakšen par nenatančno ocenimo.

Seznamsko (ang. listwise): Pri postopku učenja se za instanco problema uporabljajo celotni seznami razvrščenih dokumentov in iz tega se zgradi model.

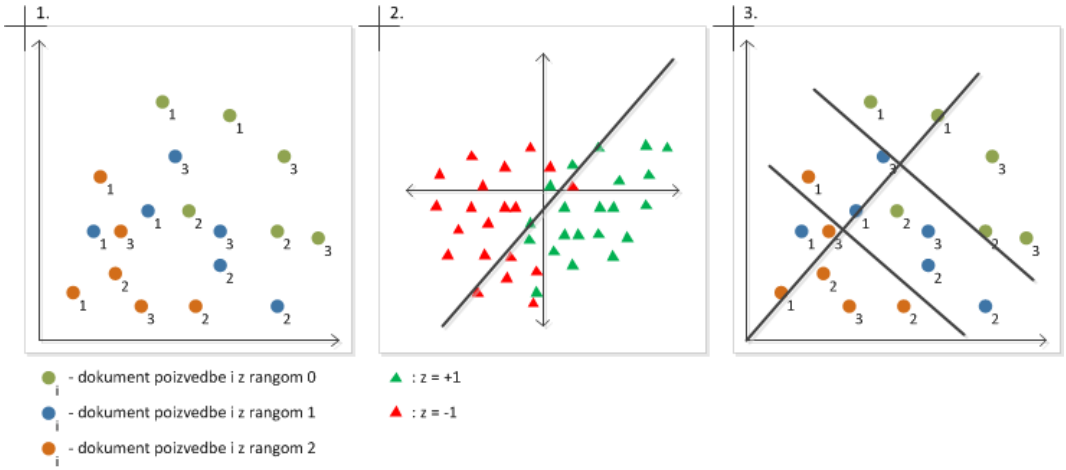
Okoli leta 2000 so se iskalniki razvili in so poleg ujemanja besed med poizvedbo uporabljali še ostale atribute. To je postalo zanimivo za področje strojnega učenja. Glede na klasične klasifikacijske probleme, kjer napovedujemo binaren rezultat, morajo algoritmi za rangiranje napovedovati vrednost na intervalu celih števil in to so regresijski problemi.

Eden zelo uspešnih algoritmov za rangiranje je Ranking SVM [12], ki temelji na metodi podpornih vektorjev (ang. SVM - support vector machine) [14]. Pri učenju uporablja parno ocenjevanje. Zato je njegov cilj minimiziranje števila napačnih parov. Če uporabnik izbere med rezultati npr. 4. dokument pred prvimi štirimi, pomeni, da je algoritem pare (4,1),(4,2) in (4,3) klasificiral napačno. Joachims je predlagal tudi uporabo [13], kako zajeti uporabnikovo odločitev iz iskalniških dnevnikov in jo uporabiti za učenje.

Učni podatki za Ranking SVM so podani v obliki trojic (y, q, $\vec{x}$), kjer y predstavlja oceno rangiranja iz $1, 2, 3, \ldots, r$, q enolično identifikacijsko številko poizvedbe in $\vec{x}$ vektor ocen atributov. V velikih sistemih uporabljamo tudi več kot 500 različnih atributov. Primeri atributov so npr. ali je iskani niz drugačne barve, ali se pojavi v url naslovu, število povezav na strani, PageRank ocena, dolžina URL naslova, ... Te trojice moramo pretvoriti za binarno klasifikacijo za uporabo SVM algoritma [14]. Znotraj vsake poizvedbe i izvedemo pretvorbo med vsemi pari dokumentov v obliko ($\vec{x}_i^{(1)} - \vec{x}_i^{(2)}$, z):

$$z = \begin{cases} +1 & \vec{y}_i^{(1)} > \vec{y}_i^{(2)} \\ -1 & \text{sicer} \end{cases} \tag{3.8}$$

Ko izračunamo ločitveno hiperravnino, jo vstavimo v prostor osnovnih podatkov. Vzdolž te hiperravnine moramo sedaj določiti še $r - 1$ mej. Meje moramo postaviti tako, da je v vsakem intervalu med dvema mejama zbrana čimbolj čista množica dokumentov. Postopek delovanja algoritma je prikazan na sliki 3.9. Novi dokumenti bodo ocenjeni z oceno intervala, v katerem se bodo nahajali.



Slika 3.9: Ranking SVM primer. 1. podatki za tri poizvedbe predstavljeni v ravnini, 2. pretvorba podatkov in izračun ločitvene premice, 3. ločitvena premica med prvotnimi podatki z mejami

3.2.2 Združevanje ocen

Sistem iCORE uporablja metaiskalnik, ki dokumente rangira znotraj skupin in jih posreduje modulu za zlivanje. Nepodvojene dokumente znotraj skupin moramo rangirati v čimbolj ustreznem vrstnem redu. V ta namen predlagamo naslednje rangiranje:

$$rank(q, D) = (1 - \alpha) \sum_{I \in MI(D)} \frac{R_{D,I}}{|MI(D)|} + \alpha ICORERank(q, D) \qquad (3.9)$$

D predstavlja ocenjevan dokument, q podano poizvedbo, ki je bila lahko avtomatsko spremenjena, $MI(D)$ vse iskalnike, ki so D vrnili med rezultati in $R_{D,I}$ rang dokumenta D, ki mu ga je dodelil iskalnik I. $ICORERank(q, D)$ je ocena algoritma za rangiranje sistema iCORE, ki je izračunana z eno izmed metod dinamičnega ocenjevanja, ki smo jih omenili v razdelku 3.2.1.

V teoriji [15] je predlaganih tudi nekaj naivnih ocen, kot je izbira maksimalnega ranga, minimalnega, povprečja in podobno. Bolj uporabne enostavne metode izvirajo iz volitev. Vsak iskalnik se obnaša kot eden izmed volivcev, dokumenti pa predstavljajo volilne kandidate. Na primeru si bomo pogledali tri postopke. Za vsakega bomo izračunali rangiranje glede na primer iz tabele 3.2.

Iskalnik	Rangiranje
I_1	c,a,d,b
I_2	a,d
I_3	d,c,b,a
I_4	c,d,a
I_5	b,c,d

Tabela 3.2: Primer rangiranja dokumentov 5 različnih iskalnikov.

Borda rangiranje: Vsak volivec za kandidata označi število točk, ki mu jih želi dodeliti. Prvemu dodeli n točk, drugemu $n-1$, ... Tistim, ki točk ne dodeli, se jim enakomerno razporedijo. Zmagovalec je tisti kandidat, ki zbere največ točk. Primer:
ocena(a) = 3+4+1+2+1 = 11
ocena(b) = 1+1,5+2+1+4 = 9,5
ocena(c) = 4+1,5+3+4+3 = 15,5
ocena(d) = 2+3+4+3+2 = 14

Končni vrstni red: c,d,a,b

Condorcet rangiranje: Zmagovalni kandidat je tisti, ki premaga vse druge v primerjanju po parih. Če volivec ne izbere kandidata, kandidat izgubi proti vsem drugim. Vsi neizbrani kandidati nekega volilca so izenačeni. Primer:

	a	b	c	d
a	-	3:2:0	1:4:0	2:3:0
b	2:3:0	-	1:3:1	1:4:0
c	4:1:0	3:1:1	-	3:2:0
d	3:2:0	4:1:0	2:3:0	-

Tabela 3.3: Primerjava ocen po parih. Vrednosti: (št. zmag):(št. porazov):(št. izenačenj)

Končni vrstni red: c,d,a,b

Recipročno rangiranje: Podobno kot Borda rangiranje, le da imajo zmagovalni kandidati večjo težo od ostalih. Volivci najboljšemu kandidatu dodelijo 1 točko, drugemu $\frac{1}{2}$, tretjemu $\frac{1}{3}$... Primer:

	Zmaga	Poraz	Izenačenost
a	1	2	0
b	0	3	0
c	3	0	0
d	2	1	0

Tabela 3.4: Primerjava rezultatov za zmago, poraz in izenačenost.

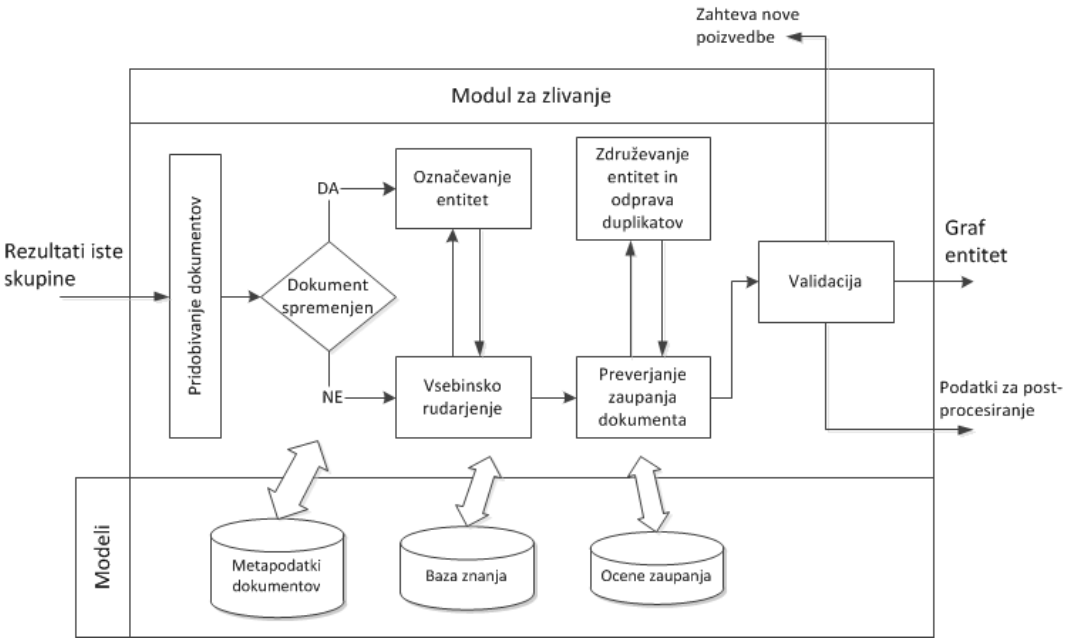
ocena(a) = 1/2+1+1/4+1/3 = 2,08
ocena(b) = 1/4+1/3+1 = 1,58
ocena(c) = 1+1/2+1+1/2 = 3
ocena(d) = 1/3+1/2+1+1/2+1/3 = 2,66

Končni vrstni red: d,a,c,b

3.3 Modul za zlivanje

Modul za zlivanje v sistemu iCORE predstavlja zadnjo enoto pred posredovanjem podatkov uporabniku. Njegova naloga je, da sprejme metaiskalnikove skupine dokumentov in jih sprocesira, da pridobi čimveč informacij. Pri tem mora poleg uporabnikovega konteksta upoštevati še kontekst informacij v dokumentih. Če ugotovi, da bi lahko podatke še bolj obogatil, lahko tudi sam oblikuje poizvedbo za modul za obdelavo poizvedb. Kot rezultat naprej posreduje ustvarjene entitete z rezultati metaiskalnika.

Na sliki 3.10 je prikazano delovanje modula za zlivanje. Pred začetkom obdelave modul najprej prenese dokumente v lokalni spomin sistema iCORE. To se na poti izvajanja prvič in edino le zgodi na tem mestu. Dokument po končani obdelavi še nekaj časa ostane v predpomnilniku, kar se izkaže uporabno zaradi časovne lokalnosti poizvedb. Ko posedujemo dokumente, s pomočjo **podkomponente za vsebinsko rudarjenje** ustrezno zgradimo entitete in jih povežemo glede na medsebojne relacije. Če modeli še ne vsebujejo navodil za pridobitev podatkov iz določenega dokumenta ali se je njegova struktura spremenila, dokument pregledamo z **označevalnikom podatkov/entitet** (ang. data/entity extractor). Ta glede na znanje, ki ga imamo, označi vse dele dokumenta, ki jih razpozna. Uporabni deli so imena oseb, krajev, datumi... Posebej se moramo osredotočiti na entitete, ki bi jih želel uporabnik. Uporabnikov kontekst nam je znan že od analize poizvedbe in ga poskušamo izkoriščati ves čas. Označeni deli dokumenta služijo za ustrezno vsebinsko podatkovno rudarjenje dokumenta. Ko imamo dele dokumentov ustrezno označene, iz rezultatov **iz-**



Slika 3.10: Modul za zlivanje. Shematski prikaz modula za zlivanje v sistemu iCORE.

gradimo graf, kjer povezave predstavljajo relacije med označenimi deli. V takšnem grafu je veliko podvajanj ali različnih delov istih entitet. Ta graf sprejme **podkomponenta za združevanje entitet** (ang. entity resolution) **in odpravljanje duplikatov** (ang. redundancy elimination). Poleg znanega uporabnikovega konteksta moramo na tem mestu razpoznavati še kontekst informacij v dokumentih. Ta kontekst vsebuje avtorja informacij, časovni okvir, starost podatkov, stopnjo zaupanja,... Ko je graf očiščen, **validacijska komponenta** preveri, če je potrebno izvesti še kakšno poizvedbo za bolj natančen graf, sicer podatke predela v splošno obliko (kot je XML zapis) in jih pošlje uporabnikovemu odjemalcu in modulu za post-procesiranje.

3.3.1 Označevalnik entitet

Naloga označevalnika entitet je, da osnovni dokument predela in v njem označi razpoznane dele. Za podano poved:

France Prešeren se je rodil 3. decembra 1800 v Vrbi na Gorenjskem.

bi bil rezultat označena poved:

<**entity type="oseba"**> *France Prešeren* </**entity**> *se je rodil*
<**entity type="datum"**> *3. decembra 1800* </**entity**> *v*
<**entity type="kraj"**> *Vrbi na Gorenjskem* </**entity**>.

Označevalniki entitet delujejo kot samostojni sistemi nad poznano domeno, ki jo hranimo v bazi znanja. Sistem Enrycher [18] deluje kot spletna storitev in omogoča označevanje podanega besedila. Pri uporabi spletnih dokumentov lahko označevanje izboljšamo s pregledovanjem imen oblikovnih stilov CSS[1]. Ponavadi jih spletni oblikovalci ustrezno pomensko poimenujejo.

Poleg baze znanja potrebujemo še vzorce, v katerih se entitete pojavljajo, da jih lahko hitro označimo [20]. Vzorce hranimo v obliki trojic. Prvi element predstavlja besedilo levo od entitete, drugi element entiteto in tretji besedilo na desni strani entitete. Vsak element lahko zapišemo v obliki regularnega izraza in pomnimo seznam potencialnih besed oz. besednih zvez, ki se lahko pojavijo. Sistemu lahko vzorce podamo ročno ali pa se jih poskuša naučiti s pomočjo metod strojnega učenja. Za učenje ne potrebujemo ročno označenih primerov, ker lahko samostojne entitete označimo le s pomočjo baze znanja in se nato naučimo relacij. Primer vzorca za zgornji primer bi bil

(*OSEBA*, **rojena? v**, *KRAJ*),

s katerim bi ujeli relacijo

RojenV: (*France Prešeren*, *Vrbi na Gorenjskem*).

Za sistem iCORE vzorci niso pomembni samo za hitrejše označevanje, ampak jih potrebujemo tudi za lažje vsebinsko rudarjenje, predvsem pa za ustrezno izgradnjo grafa, ki ga podamo podkomponenti za združevanje entitet (glej razdelek 3.3.3).

V razdelku 1.2 smo spoznali, da navadni iskalniki za podano poizvedbo le poiščejo najbolj ustrezne dokumente in jih vrnejo kot rezultat. Sistem iCORE želi ponuditi nekaj več, zato lahko vzorce izkoristi za odgovarjanje na vprašanja uporabnika. Če uporabnik poda poizvedbo *Kje je rojen France Prešeren?*, lahko s pomočjo vzorcev na enak način kot prej izluščimo nepopolno relacijo

RojenV: (*France Prešeren*, **X**).

Med dokumenti, ki nam jih vrne metaiskalnik, poiščemo vse relacije in poskusimo najti ujemanje. Ko dobimo relacijo kot v zgornjem primeru, vrnemo odgovor *Vrbi na Gorenjskem*.

[1] Cascading Style Sheets - predloge, ki določajo izgled spletnih strani.

3.3.2 Vsebinsko rudarjenje

Vsebinsko rudarjenje je veja spletnega rudarjenja. Cilj je doseči povratno inženirstvo (ang. reverse engineering), da iz dokumentov pridobimo nazaj informacije v strukturirani obliki. Pri tem ločimo tri pristope [15]:

Ročni (ang. manual): Programer glede na določeno strukturo dokumenta napiše razčlenjevalnik (ang. parser), ki izlušči podatke.

Z ovojnico (ang. wrapper): Uporabi se nadzorovano učenje. Iz označenih podatkov, kako se v izbranih dokumentih izloči podatke, zgradimo model, ki nam omogoča izločati podatke iz podobnih dokumentov.

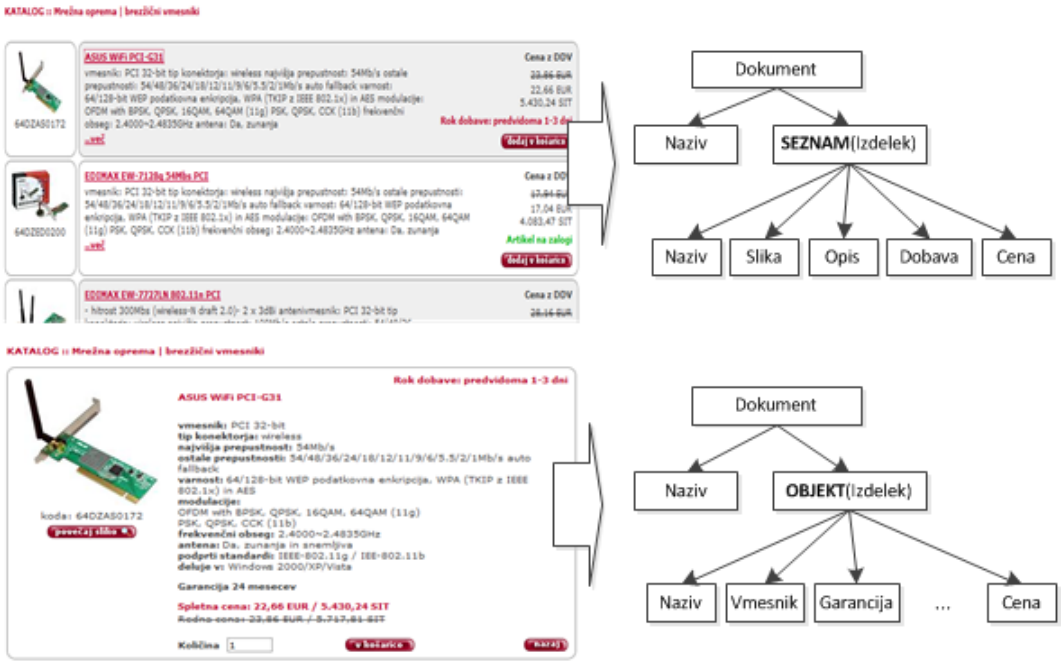
Avtomatični (ang. automatic): Z nenadzorovanim učenjem poiščemo vzorce v dokumentih in poskusimo izluščiti podatke.

Ročni način za sistem iCORE ni uporaben, saj bomo imeli mnogo različnih dokumentov. V poštev pride kombinacija nadzorovanega in nenadzorovanega učenja. Ker bomo v dokumentu že imeli označene entitete, kar je naloga predhodne podkomponente, in bomo iz baze znanja poznali zgradbo entitetnega tipa, smo si nalogo olajšali.

Sistem STALKER [16] dokumente deli na naštevalne in detajlne. Primer delovanja lahko vidimo na sliki 3.11. Sistem potrebuje označene elemente dokumenta in na podlagi tega sklepa o strukturi oz. podatkovnem modelu. Glede na to, kaj se nahaja v okolici označenih elementov, izgradi pravila. Takšnemu načinu gradnje modela pravimo induktivno logično programiranje (ILP) [17]. STALKER ILP problem rešuje s prekrivnim algoritmom. Za probleme ILP potrebujemo pozitivne, negativne primere in znanje. Znanje nam lahko v našem primeru predstavljajo kar označeni deli dokumenta. Pozitivne in negativne primere pa lahko pridobimo iz ostalih modelov za dokumente.

Sistem mora omogočati uporabnikom nadzorovanje preko aktivnega učenja (ang. active learning). Če bo sistem med primeri našel protislovje, se bo moral o rezultatu odločiti uporabnik. Tudi, ko bo uporabnik že dobil končne rezultate, bo moral imeti možnost, da označi spuščene atribute ali odstrani napačne.

Zaradi zasnove sistema iCORE smo pokazali, kako lahko algoritem s pristopom ovojnice uporabljamo kot avtomatskega. Ko odkrijemo osnovno strukturo dokumenta, nam ostane še besedilo, iz katerega moramo izluščiti informacije. Tudi za to potrebujemo označene entitete in relacije med njimi. Uporabljamo metode za odkrivanje zakonitosti v besedilih (ang. text mining), med katerimi sta za nas pomembni učenje relacij med entitetami in nenadzorovano razvrščanje besedil v skupine.



Slika 3.11: Sistem STALKER. Delovanje sistema STALKER na naštevalnih in detajlnih dokumentih.

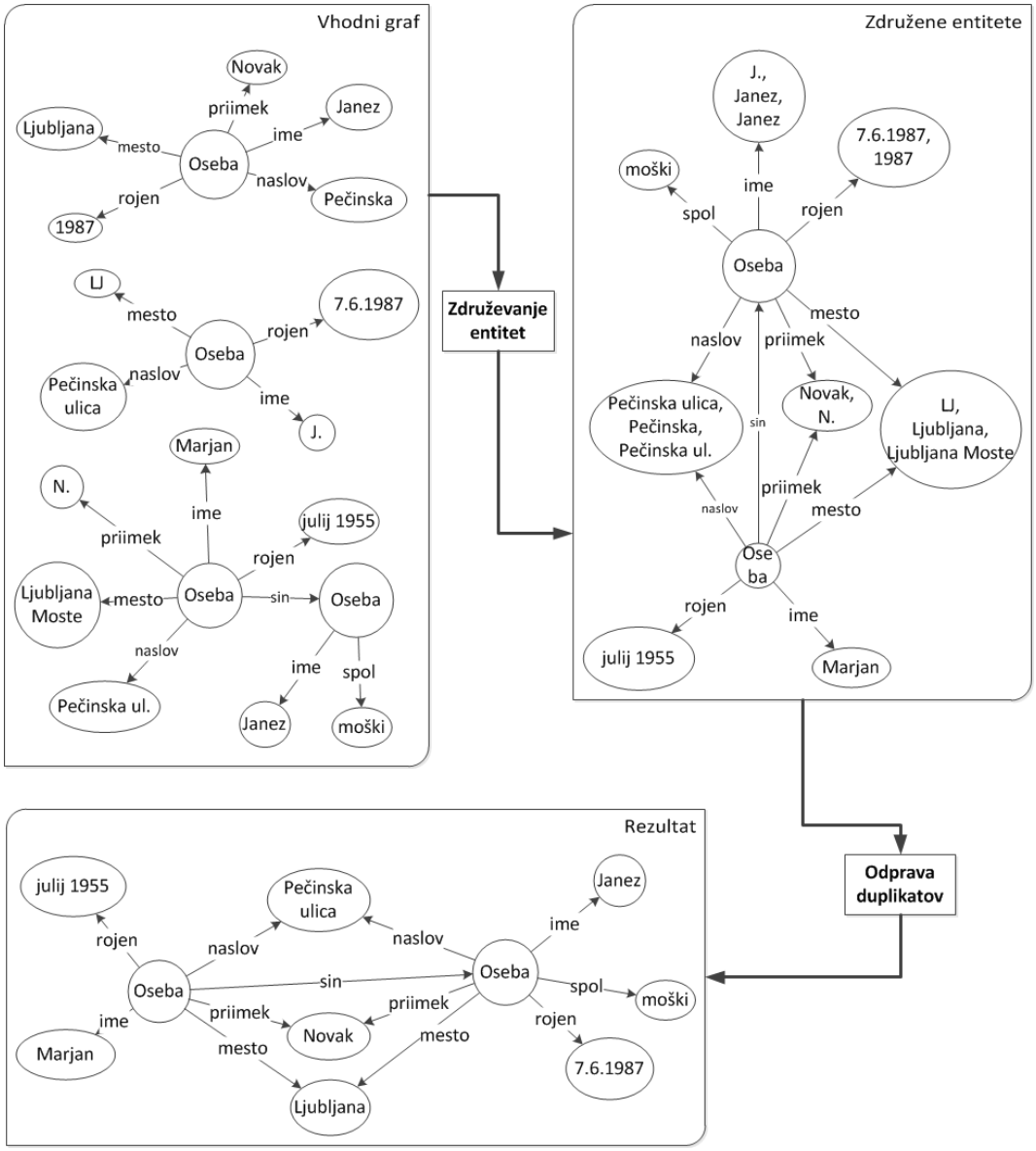
3.3.3 Združevanje entitet in odprava duplikatov

Podkomponenta za vsebinsko rudarjenje nam vrne hierarhično predstavljene entitete z atributi. Te entitete bomo predstavili v grafu. V grafu bomo nato poiskali instance istih entitet in jih združili. Grafi so najbolj primerna predstavitev, ker lahko z njimi enostavno prikažemo relacije med entitetami. Pri združenih entitetah moramo potem še odpraviti duplikate oz. več vrednosti zliti skupaj. Primer delovanja si lahko ogledamo na sliki 3.12.

Algoritme, ki se ukvarjajo z združevanjem entitet, delimo glede na način delovanja na:

Atributski (ang. attribute-based): Za vsaki dve referenci entitetnih tipov izračunamo oceno podobnosti $sim_A(r_i, r_j)$. Združimo le tiste pare, ki imajo oceno nad predhodno izbranim pragom (ang. threshold).

Relacijski (ang. relational): Relacijski pristop predvideva, da če obstajata dve neodvisni referenci iste entitete, sta najbrž povezani s podobnimi sosedi. V ta namen izračunamo še podobnost med vsemi pari sosedov obeh referenc $sim_N(r_i, r_j)$ in določimo vrednost $0 \leq \alpha \leq 1$ za stopnjo



Slika 3.12: Združevanje in odprava duplikatov entitet.

upoštevanja.

$$sim_R(r_i, r_j) = (1 - \alpha) sim_A(r_i, r_j) + \alpha sim_N(r_i, r_j)$$

Celostno relacijski (ang. collective relational): Celostni pristop za razliko od relacijskega upošteva še spremembe med fazo združevanja. Torej, če

združimo dve referenci, se podobnost sosedov obeh referenc še poveča. Združene podobne reference r hranimo v skupinah $S(r)$. Definirati moramo še mero podobnosti skupin $sim_S(r_i, r_j) = sim_R(r_{i'}, r_{j'})$, kjer $(r_{i'}, r_{j'}) \in S(r_{i'}) \times S(r_{j'})$. Z upoštevanjem naslenje mere postopek iterativno ponavljamo:

$$sim_{CR}(r_i, r_j) = (1 - \alpha) sim_A(r_i, r_j) + \alpha sim_S(r_i, r_j)$$

Za računanje podobnosti atributov lahko uporabimo kar nekaj različnih razdalj kot so kosinusna (glej enačbo 3.1), Jaro, Jaro Winkler, Soundex, Levenshteinova, ... Implementacije in opise posameznih si lahko ogledamo v knjižnici SimMetrics [21]. Omenjene razdalje so prirejene za znakovne atribute. Uporaba ustrezne metrike je odvisna od tipa atribura. Atributi v sistemu iCORE so lahko še zvok, slike, video ali drugi tipi podatkov.

Pri celostnem relacijskem pristopu potrebujemo še mero podobnosti med skupinami. Podobno kot za nize jih je bilo razvitih precej. Zelo enostavna in z dobrimi rezultati v praksi [22] je Jaccard-ov koeficient. Funkcija $Nbr(s)$ vrne vse skupine, ki so sosednje referencam, ki nastopajo v skupini s.

$$JaccKoef(s_i, s_j) = \frac{|Nbr(s_i) \bigcap Nbr(s_j)|}{|Nbr(s_i) \bigcup Nbr(s_j)|} \tag{3.10}$$

Uspešen celostno relacijski algoritem za združevanje entitet je predstavila raziskovalka Getoor [22]. Sestoji iz treh faz:

1. **Grupiranje:** Z grupiranjem reference razporedimo po blokih. V vsak blok uvrstimo reference, ki so si podobne glede na hitro atributsko metriko. Ena referenca lahko nastopa v enem ali več blokih. Grupiranje je uporabno, ker imamo ponavadi podanih mnogo entitet za združevanje in tako namesto, da bi pregledali vse možne pare, si zmanjšamo zahtevnost in gledamo le pare v blokih, kjer se nahajajo potencialni duplikati. Pred pričetkom algoritma lahko besede tudi normaliziramo.

2. **Vzpostavitev začetnega stanja:** Celostno relacijski pristop združuje reference po skupinah glede na soseščino in ena združitev vpliva na nadaljne. Ker prvotno podobne entitete niso povezane, sploh če prihajajo iz različnih virov, v tem koraku pripravimo potencialne pare referenc glede na relacijski pristop podobnosti, ki jih združujemo v naslednjem koraku. Če pa že tu ugotovimo, da se referenci popolnoma ujemata, jih lahko takoj združimo v skupino.

3. **Iterativno združevanje:** V tem koraku združujemo potencialne pare skupin glede na celostni relacijski pristop, dokler maksimalna podobnost med različnima skupinama ne pade pod določeno mejo. Na začetku vsaka referenca predstavlja svojo skupino. Glavna lastnost celostnega pristopa se pokaže v tem, da ko združimo dve skupini, posodobimo podobnosti ostalih skupin, na katere je ta združitev vplivala.

Psevdokodo algoritma si lahko ogledamo v dodatku A.

Ko imamo iste entitete v skupinah, moramo še ustrezno odpraviti duplikate (ang. redundancy elimination), tako da izberemo ali sestavimo eno vrednost. Če imamo v skupini za osebo *Janez Novak* več entitet, ki vsebujejo atribut ime kot *J. Novak*, *Janez N.* ali *Janez Novak*, se moramo odločiti za enega. Pri združevanju entitet se lahko zgodi, da imamo v skupini vrednost atributa, ki je pomensko različna od drugih. Do tega lahko pride, če je ta atribut manj utežen in se referenci zelo dobro ujemata v drugih. Pri izbiri vrednosti bomo upoštevali še oceno zaupanja dokumenta D_i, iz katerega izhaja vrednost v_i. To oceno bomo pridobili preko TrustRank algoritma [33]. Ideja za algoritem izhaja iz predhodno predstavljenega PageRank ocenjevanja (glej razdelek 3.2.1). Eksperti ročno označijo začeten nabor zaupanja vrednih dokumentov. Ocene se nato za ostale dokumente računajo glede na povezave med njimi. V sistemu iCORE za dokument D_i izberemo naslednjo oceno:

$$trust(q, D_i) = (1 - \alpha) rank(q, D_i) + \alpha TrustRank(D_i) \tag{3.11}$$

Metodo *rank* smo predstavili v enačbi 3.9. Sedaj izberemo vrednost v_j, ko velja:

$$j = \arg\max_i trust(q, D_i) \tag{3.12}$$

3.4 Modul za post-procesiranje

Naloga modula za post-procesiranje je, da po vsaki izvedeni poizvedbi posodobi morebitne spremembe v modelih. Poleg tega si hrani učne in testne podatke za prilagajanje klasifikatorjev. Omogočati mora tudi ročno vnašanje sprememb v modele za lažjo administracijo sistema. Na voljo ima vse podatke kot so osnovne in spremenjene poizvedbe, uteži iskalnikov, itd. vključno z dokumenti in odziv uporabnika, če je implementiran v odjemalcu. Rezultati njegovega izvajanja niso časovno kritični, zato lahko za svoje posle uporablja sklad in jih odloženo procesira.

Za vsakega uporabnika posebej si shranjuje zgodovino poizvedb v obliki (*rank*, *uid poizvedbe*, *ocene atributov*) (glej razdelek 3.2.1). Te rezultate potem uporabljamo za prilagajanje klasifikatorja za rangiranje dokumentov specifičnega uporabnika (ang. preference learning). Rezultati več uporabnikov služijo učenju globalnega klasifikatorja. Glede na to, kateri iskalniki vračajo za določene poizvedbe bolj relevantne dokumente, določimo tudi uteži za njihovo izbiro v modulu metaiskalnik.

Gradnja klasifikatorja po vsaki poizvedbi na novo bi bila zelo potratna. Zato lahko klasifikatorje posodabljamo, ko pridobimo določen delež novih podatkov. Določimo tudi maksimalno velikost n korpusa, iz katerega se učimo. Potem lahko npr. ko pridobimo 30% novih podatkov, stratificirano izberemo 70% starih in skupaj z novimi zgradimo nov klasifikator. Naslednja možnost je, da klasifikator posodobimo, ko pridobimo primer, ki ga trenutni napačno klasificira. Takšen način posodabljanja je za SVM [14] algoritem predstavila raziskovalka Richa Singh v sistemu za prepoznavanje obrazov [23].

Pri procesiranju dokumentov smo že v modulu za zlivanje pridobili strukturo zapisa. Na tem koraku moramo za dokumente določenega vira posodobiti stopnjo zaupanja. Pri pregledovanju spletnih dokumentov moramo pregledovati spletne obrazce, če vsebujejo nove iskalnike.

Naslednja pomembna komponenta modelov je baza znanja, ki jo moramo vzdrževati. Pri opisovanju modula za obdelavo poizvedb (razdelek 3.1) smo se odločili, da bomo ločeno uporabljali bazo znanja za predstavitev strukture entitet in za podatke. Poleg tega bomo hranili še vzorce, ki jih potrebujemo za uspešno označevanje entitet. Bazo znanja strukture entitet ne posodabljamo avtomatsko, vendar le predlagamo spremembe, ki jih lahko vnese domenski ekspert ali zanje glasujejo uporabniki sistema. Vzorci se posodabljajo avtomatsko. Z njimi se odkrivajo novi načini podajanja relacij med entitetami. Npr. za relacijo igralec igra v ekipi poznamo vzorce $IgraV = \{$[*igralec*] *igra v* [*ekipa*], *najboljši v* [*ekipa*] *je* [*igralec*]$\}$. Pri obdelavi dokumentov komentiranja neke tekme, moramo prepoznati nov vzorec {[*igralec*] [*ekipa*] *je zamenjan z* [*igralec*]}. Z avtomatskim načinon odkrivanja podobnih relacij z uporabo novih iskanj in pregledovanjem dokumentov se je ukvarjal Gijs Geleijnse [24]. Sedaj posodobimo še bazo znanja z entitetami. Prvotni sistemi so uporabljali le vzorce in en dokument. V Yahoo laboratorijih so predstavili platformo, ki zna pametno uporabiti več dokumentov [19]. V našem primeru so dokumenti kar vsi relevantni rezultati. Platforma vsebuje več specializiranih označevalnikov entitet in preverjevalnikov novih entitet. Pred posodobitvijo baze znanja še oceni potencialne entitete in jih ob uspehu doda v bazo znanja.

3.5 Modeli

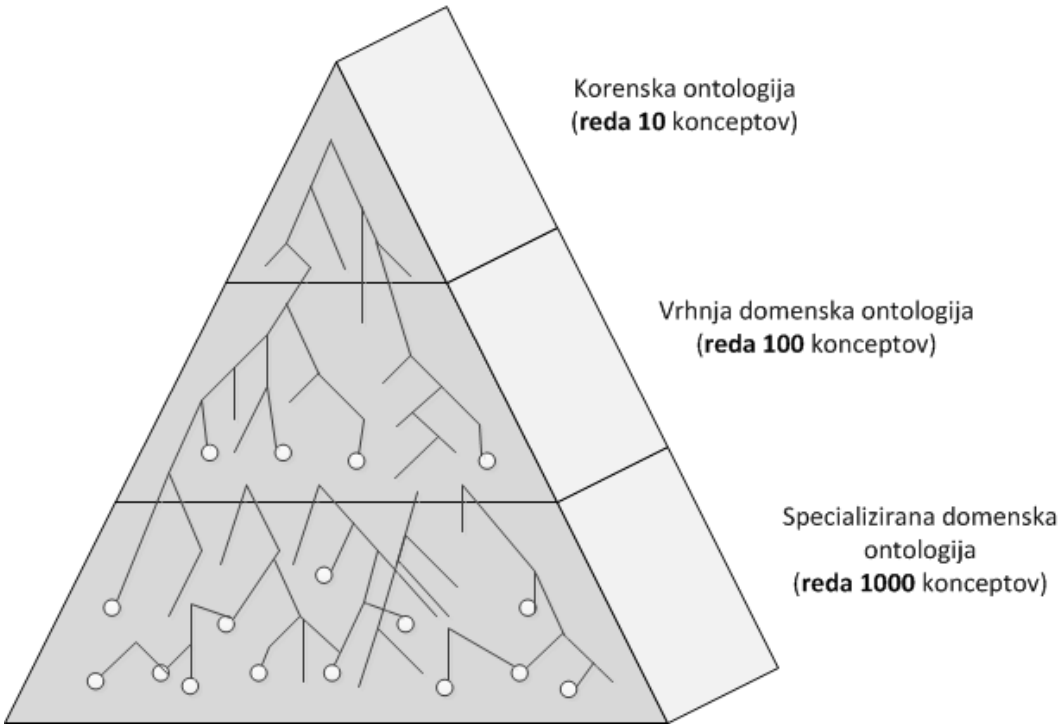
Modeli so statična struktura v sistemu iCORE, ki jih uporabljajo in spreminjajo preostale komponente. Vsebujejo vse znanje, ki ga sistem ima. Hranijo obe bazi znanja (prva namenjena strukturi entitet, druga vsebuje še podatke), kontekste uporabnikov, metapodatke o dokumentih, vzorce za označevanje entitet, seznam in uteži iskalnikov za določeno skupino poizvedb, ocene virov in njihovo zaupanje, naučene modele klasifikatorjev sistema ... Vse komponente sistema iCORE uporabljajo modele, spreminjata jih le modul za zlivanje (glej 3.3) in modul za post-procesiranje (glej 3.4).

Izglede posameznih modelov smo predstavili v zgornjih razdelkih ob vsaki metodi posebej. Tu bomo predstavili še bazo znanja v obliki ontologije [25].

Ontologija v računalništvu predstavlja model za opisovanje sveta. Uporablja se na mnogo področjih, predvsem pa v umetni inteligenci in semantičnem spletu (ang. semantic web). Ontologije lahko vsebujejo objekte oz. entitete, hierarhično predstavljene razrede, v katere spadajo entitete, atribute, relacije med njimi, omejitve in če-potem (ang. if-then) pravila. Glede na število konceptov delimo ontologije v tri razrede, kot vidimo na sliki 3.13. Eden koncept na višjem nivoju ontologije ima lahko več pomenov v domenskih ontologijah. Združevanje več domenskih ontologij v eno je zahteven proces, ker jih ponavadi izdelujejo različni ljudje in za koncepte uporabljajo različna imena. Problem je podoben združevanju entitet ali shem podatkovnih baz. Obstaja nekaj ontologij, kot sta Cyc ali WordNet, ki jih ročno gradijo že več 10 let. V tem času se je razvilo tudi nekaj jezikov za poizvedovanje po ontologijah. Cilj semantičnega spleta je, da bi bila vsa vsebina na spletnih straneh označena z neko ontologijo in bi nam s tem olajšala priklic podatkov. O tem je utopično razmišljati, saj bi moral potem vsak poznati ontologijo z vsemi koncepti in relacijami med njimi. Zaradi tega v sistemu iCORE avtomatsko obdelujemo dokumente in pol-avtomatsko posodabljamo bazo znanja entitet. V viziji semantičnega spleta je tudi razvit opisovalni jezik RDF (ang. Resource Description Framework), ki predvideva, da ima vsak podatek na internetu določen pomen glede na ontologijo. Za poizvedovanje lahko uporabimo jezik SPARQL (ang. Protocol and RDF Query Language). Primer poizvedbe, ki vrne osebe, ki živijo v Zgornjem Dupleku:

```
PREFIX xonto: <http://zitnik.si/druzbenaOntologija/0.1/>
SELECT ?ime ?mail
WHERE {
  ?oseba a xonto:Oseba.
  ?oseba xonto:ime      ?ime.
```

```
  ?oseba xonto:enaslov  ?mail.
  ?oseba xonto:naslov   "Zgornji Duplek".
}
```



Slika 3.13: Hierarhija ontologij.

3.6 Evaluacija

Sistem iCORE ni običajen sistem za priklic podatkov. Običajni sistemi kot rezultat vračajo le relevantne dokumente in krajše izvlečke, medtem ko mi iz njih izluščimo tudi informacije. Podobne sisteme ocenjujemo glede na časovni odziv na poizvedbo in najbolj pomembno z uporabnikovim zadovoljstvom. Uporabnik je zadovoljen, če so njemu ljube informacije prednostno prikazane. Sistem iCORE je sestavljen iz več dodatnih komponent, zato lahko tudi vsako izmed njih posebej testiramo kot v strojnem učenju.

Najbolj osnovna mera v strojnem učenju klasifikacijska točnost (ang. CA - classification accuracy). Za binaren razred pove, kakšen delež primerov smo

pravilno uvrstili. Rezultate takšnih klasifikatorjev je enostavno predstaviti s tabelo (glej tabelo 3.6), iz katere lahko izračunamo več ocen.

		Dejanske vrednosti		
		+	−	
Napovedane vrednosti	+	TP	FP	Napovedani pozitivni
	−	FN	TN	Napovedani negativni
		Pozitivni	Negativni	Vsi

TP (ang. true positive): število pravilno klasificiranih pozitivnih primerov

FP (ang. false positive): število napačno klasificiranih pozitivnih primerov

TN (ang. true negative): število pravilno klasificiranih negativnih primerov

FN (ang. false negative): število napačno klasificiranih negativnih primerov

Tabela 3.5: Tabela rezultatov.

$$CA = \frac{število\ pravilno\ klasificiranih}{število\ vseh} = \frac{TP + TN}{TP + FP + TN + FN} \quad (3.13)$$

Sistemi za priklic podatkov vsebujejo ogromno dokumentov. Uporabniki želijo s posamezno poizvedbo pridobiti le relevantne rezultate, ki jih je navadno zelo malo in je npr. več kot 99% dokumentov v sistemu nerelevantnih. Zaradi tega merjenje uspešnosti sistemov za priklic podatkov s klasifikacijsko natančnostjo ni primerno.

3.6.1 Ocenjevanje nerangiranih rezultatov

Od sistema za priklic podatkov pričakujemo, da nam bo vrnil čimveč relevantnih rezultatov.

Z merama natančnost (ang. precision) in priklic (ang. recall) ocenjujemo množice vrnjenih rezultatov. Natančnost kaže delež relevantnih dokumentov glede na dokumente, ki jih je vrnil sistem. Priklic pa meri delež vrnjenih relevantnih dokumentov glede na vse relevantne dokumente, ki jih vsebuje sistem.

$$Natančnost = \frac{število\ vrnjenih\ relevantnih}{število\ vrnjenih} = \frac{TP}{TP + FP} \quad (3.14)$$

$$Priklic = \frac{število\ vrnjenih\ relevantnih}{število\ relevantnih\ v\ sistemu} = \frac{TP}{TP + FN} \tag{3.15}$$

Vedno lahko dosežemo priklic enak 1 tako, da vrnemo vse dokumente iz sistema, vendar bo natančnost nizka. Funkcija priklica z naraščajočim številom vrnjenih dokumentov je monotono naraščajoča. Obratno velja za natančnost, saj več dokumentov ko vrnemo, večja je verjetnost pomote. Odvisno od uporabe sistema želimo doseči različno stopnjo obeh mer. Uporabnik, ki išče po spletu, hoče dosegati visoko natančnost in ga priklic ne zanima. Obratno želi uporabnik, ki išče dokumente na trdem disku, doseči visok priklic.

Van Rijsbergen je predlagal E oceno, ki s parametrom α uteženo upošteva bolj natančnost oz. priklic:

$$E_\alpha = 1 - \frac{1}{\frac{\alpha}{Natančnost} + \frac{1-\alpha}{Priklic}} \tag{3.16}$$

Na podlagi E ocene se je oblikovala F ocena:

$$F_\beta = (1 + \beta^2)\frac{Natančnost \cdot Priklic}{\beta^2 Natančnost + Priklic} \tag{3.17}$$

Splošno se uporablja F_1 ocena, ki predstavlja harmonično sredino natančnosti in priklica. Visoke vrednosti vrača, če sta visoka tako priklic kot natančnost.

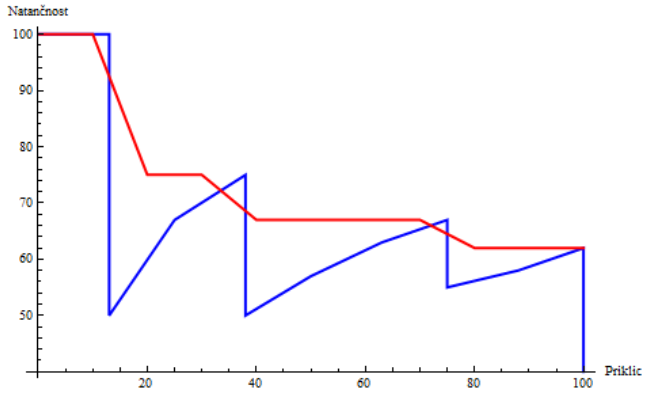
3.6.2 Ocenjevanje rangiranih rezultatov

Sistem za priklic podatkov vrne k dokumentov, za katere oceni, da so relevantni in jih razvrsti padajoče glede na relevantnost. Zgoraj smo ocenjevali množice vrnjenih dokumentov, sedaj bomo upoštevali še vrstni red.

Za natančnost in priklic lahko izrišemo krivuljo natančnost-priklic (ang. precision-recall curve). Delovanje si bomo ogledali na primeru. Predpostavimo, da sistem vsebuje 20 dokumentov in 8 relevantnih. V Dodatku B v tabeli 4 imamo ocenjene vse dokumente glede na podano poizvedbo in izračunane natančnosti $p(i)$ in priklice $r(i)$. Te vrednosti za i smo izračunali tako, da smo upoštevali, da sistem vrne dokumente z oceno od 1 do i. Krivuljo dobimo tako, da te vrednosti vnesemo v ravnino, kjer na abscisni osi vnašamo priklic in na ordinatni natančnost. Za lepšo krivuljo, lahko vrednosti interpoliramo za 11 priklicev (0%, 10%, ..., 100%). Za vsak priklic natančnost izračunamo:

$$p(r_i) = \max_{r_i \le r \le r_{11}} p(r) \tag{3.18}$$

Interpolirane vrednosti za primer so izračunane v Dodatku B v tabeli 4. Obe krivulji sta izrisani na sliki 3.14.



Slika 3.14: Krivulji natančnost-priklic. Modra krivulja predstavlja natančne vrednosti in rdeča interpolirane.

Različne sisteme primerjamo tako, da za rezultate poizvedbe na sistemih izrišemo krivulje in jih primerjamo med seboj. Najboljši sistem izberemo glede na svoje preference. Lahko npr. želimo, da le za nekaj prvih dokumentov dosegamo visoko natančnost, ali da je povprečno najboljša. Mera MAP (ang. mean average precision) upošteva velikost površine pod krivuljo. Recimo, da testiramo z več poizvedbami $q_j \in Q$ in je množica relevantnih dokumentov za q_j označena z $\{d_1, d_2, \ldots, d_{m_j}\}$. Z R_{jk} je označena množica vseh dokumentov glede na vrnjeno rangiranje q_j od vrha do vključno dokumenta d_k.

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Natančnost(R_{jk}) \tag{3.19}$$

Z uporabo algoritmov strojnega učenja za namene rangiranja (od okoli leta 2000 dalje) je postala pomembna ocena DCG oz. normalizirana inačica $NDCG$ (ang. normalized discounted cumulative gain). Ocena primerja vrnjeno rangiranje dokumentov in idealno.

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i} \tag{3.20}$$

$$NDCG_p = \frac{DCG_p}{IdealniDCG_p} \tag{3.21}$$

Vrednost rel_i označuje relevantnost dokumenta in p število rangiranih dokumentov. Recimo, da nam je sistem za priklic podatkov vrnil 7 dokumentov. Ekspert jih označi glede na relevantnost v vrstnem redu 3,4,1,3,2,0,1. Idealen vrstni red bi bil 4,3,3,2,1,1,0. Ocena $NDCG$ za ta primer:

$$NDCG_7 = \frac{3 + 4 + 0,63 + 1,5 + 0,86 + 0 + 0,36}{4 + 3 + 1,89 + 1 + 0,43 + 0,39 + 0} = \frac{10,35}{10,71} = 0,97 \quad (3.22)$$

3.6.3 Testni nabori podatkov

Za kakršnokoli testiranje metod ali sistemov potrebujemo nabor označenih učnih in testnih podatkov. Obširni označeni nabori so nastali za namene raziskovalnih skupin ali konferenc, kjer raziskovalci poskušajo odkrivati boljše algoritme. Zajem takšnih podatkov zahteva veliko časa in ljudi. Pri evaluaciji enostavnejših metod si lahko izgradimo sintetični generator podatkov. V nekaterih primerih lahko uporabimo tudi podatkovne baze, ki že vsebujejo označene podatke in jih samo pretvorimo v želeni format.

Raziskovalec Joachims [13] je svoj sistem za priklic podatkov testiral znotraj svojega raziskovalnega oddelka. Iz dnevnikov je računal odziv uporabnikov glede na to, kateri dokument so izbrali pri določeni poizvedbi. To je tudi uporabil za prilagajanje parametrov klasifikatorja v času delovanja sistema.

Nekaj znanih naborov podatkov za sisteme za priklic podatkov:

Cranfield: Prvi večji nabor podatkov iz 1950-ih let. Vsebuje 1398 člankov o aerodinamiki in 225 poizvedb.

TREC: Več naborov podatkov, ki se kreirajo od leta 1992 dalje za namene TREC (ang. Text REtrieval Conference) konference.

CLEF (ang. Cross Language Evaluation Forum): Dokumenti v več evropskih jezikih za večjezično iskanje.

Reuters: Več naborov podatkov, ki vsebujejo do okoli milijon člankov.

Pri združevanju entitet uporabljamo:

CiteSeer: Vsebuje 2892 referenc za 1165 avtorjev za 1504 dokumentov.

ArXiv: Prvič uporabljen na tekmovanju KDD Cup 2003. Vsebuje 29,555 člankov z 58,515 referencami za 9200 avtorjev.

BioBase: Vsebuje 156,156 člankov v različnih jezikih z 831,991 referencami avtorjev, kjer imamo označenih le 100 avtorjev.

Za označevanje entitet:

MUC (ang. Message Understanding Conference): Sedem naborov podatkov, razvitih za namene konference v 1990-ih letih.

DBpedia: Wikipedija v obliki ontologije. Enako lahko uporabimo tudi druge ontologije, kot je WordNet, Cyc.

Za algoritme rangiranja:

LETOR: Več naborov, ki jih vzdržuje Microsoft Research. Poleg svojih vsebuje še druge kot je OHSUMED, TREC, GOV.

Yahoo!: Dva neoznačena nabora podatkov, uporabljena na tekmovanju Learning To Rank 2010.

Za vse podkomponente lahko dobimo testne podatke. Za testiranje celotnega sistema iCORE smo se odločili za nekajmesečno testno obdobje uporabe znotraj raziskovalne skupine. Če bodo rezultati uspešni, bomo uporabo iCORE-a razširili tudi na nivo organizacije.

Poglavje 4

Ugotovitve in nadaljne delo

V diplomskem delu smo na kratko predstavili delovanje iskalnikov. Kot njihovo izboljšavo smo predlagali sistem za priklic podatkov - iCORE. Sistem smo razdelali po komponentah, jim določili funkcije ter predlagali način izvedbe. Pri vsaki komponenti ali njenem delu smo poskušali najti primerne rešitve, ki lahko služijo kot izhodišče za implementacijo sistema.

Uporabniki postajajo z razvojem tehnologije čedalje bolj zahtevni in od nje pričakujejo vedno več. V današnjem času bi se zdelo nesmiselno graditi nov iskalnik s svojim indeksom, saj je konkurenca premočna. Trdimo, da trenutnim sistemom manjkajo alternativni načini podajanja poizvedb, ki se lahko kreirajo avtomatsko. Poleg tega smo hoteli iz zadetkov, ki nam jih vrnejo iskalniki, pridobiti čimveč informacij in jih uporabniku predstaviti na način, ki mu je blizu in lahko po njih enostavno navigira. Tako kot smo se privadili, da vsak poseduje svoj "osebni" mobilni telefon, ki ima nastavljene personalizirane nastavitve, se bomo lahko v prihodnosti privadili na "osebni" sistem iCORE, ki nas bo spremljal na vsakem koraku.

Večji komercialni iskalni sistemi že ponujajo pripravljene vmesnike, s katerimi lahko komuniciramo s svojo programsko opremo. Nekateri imajo že izoblikovano politiko plačevanja v zameno za njihovo uporabo ali kakšno drugo omejitev, saj je iskalniški marketing eden bolj donosnih. Zaenkrat med iskalniki, ki uporabljajo takšne vmesnike še nismo zasledili podobnega iCORE-u. Večina njih je razvitih kot specializirani iskalnik nad določeno domeno.

V prihodnosti želimo v raziskovalni skupini v celoti razviti predlagani sistem. Pri tem bomo tudi testirali, kako se vrhunski algoritmi s posameznih področjih obnašajo pri naših nalogah, in poskušali predlagati njihove izboljšave.

Dodatki

Dodatek A

Algoritem za združevanje entitet po predlogu Lise Getoor [22]:

Algorithm 1 Združevanje entitet

```
Find similar references using Blocking
Initialize clusters using bootstrapping
for clusters c_i, c_j such that similar(c_i,c_j) do
  Insert < sim(c_i, c_j), c_i, c_j > into priority queue
end for
while priority queue not empty do
  Extract < sim(c_i, c_j), c_i, c_j > from queue
  if sim(c_i, c_j) less than threshold then
    return
  else
    Merge c_i and c_j to new cluster c_ij
    Remove entries for c_i and c_j from queue
    for all cluster c_k such that similar(c_ij, c_k) do
      Insert < sim(c_ij, c_k), c_ij, c_k > into queue
    end for
    for all cluster c_n neighbour of c_ij do
      for c_k such that similar(c_k, c_n) do
        Update sim(c_k, c_n) in queue
      end for
    end for
  end if
end while
```

Dodatek B

Tabelarični podatki za krivuljo natančnost-priklic.

Rank(d)	Relevantnost	$p(i)$	$r(i)$
1	+	$1/1 = 100\%$	$1/8 = 13\%$
2	-	$1/2 = 50\%$	$1/8 = 13\%$
3	+	$2/3 = 67\%$	$2/8 = 25\%$
4	+	$3/4 = 75\%$	$3/8 = 38\%$
5	-	$3/5 = 60\%$	$3/8 = 38\%$
6	-	$3/6 = 50\%$	$3/8 = 38\%$
7	+	$4/7 = 57\%$	$4/8 = 50\%$
8	+	$5/8 = 63\%$	$5/8 = 63\%$
9	+	$6/9 = 67\%$	$6/8 = 75\%$
10	-	$6/10 = 60\%$	$6/8 = 75\%$
11	-	$6/11 = 55\%$	$6/8 = 75\%$
12	+	$7/12 = 58\%$	$7/8 = 88\%$
13	+	$8/13 = 62\%$	$8/8 = 100\%$
14	-	$8/14 = 57\%$	$8/8 = 100\%$
15	-	$8/15 = 53\%$	$8/8 = 100\%$
16	-	$8/16 = 50\%$	$8/8 = 100\%$
17	-	$8/17 = 47\%$	$8/8 = 100\%$
18	-	$8/18 = 44\%$	$8/8 = 100\%$
19	-	$8/19 = 42\%$	$8/8 = 100\%$
20	-	$8/20 = 40\%$	$8/8 = 100\%$

Tabela 1: Vrednosti za natančnost in priklic rangiranih dokumentov celotnega sistema (20) za podano poizvedbo.

i	$p(r_i)$	r_i
1	100%	0%
2	100%	10%
3	75%	20%
4	75%	30%
5	67%	40%
6	67%	50%
7	67%	60%
8	67%	70%
9	62%	80%
10	62%	90%
12	62%	100%

Tabela 2: Interpolirana natančnost za priklic od 0% do 100% s korakom 10%.

Slike

Tabele

Literatura

[1] R. Baeza-Yates, B. Ribeiro-Neto, *Modern Information Retrieval*, New York: ACM Press, 1999, pogl. 4.

[2] B. Pajntar, M. Grobelnik, "SearchPoint - A new Paradigm of Web Search", Ljubljana, 2008, Dostopno na: http://searchpoint.ijs.si/WWW2008_Developers_Track.pdf

[3] Hu J., Wang G., Lochovsky F., Sun J., Chen, Z., "Understanding user's query intent with wikipedia," v zborniku *Proceedings of the 18th international conference on World wide web*, China, Beijing, 2009, str. 471-480.

[4] G. Kobilarov, T. Scott, Y. Raimond, S. Oliver, C. Sizemore, M. Smethurst, C. Bizer, R. Lee, "Media Meets Semantic Web – How the BBC Uses DBpedia and Linked Data to Make Connections," v zborniku *The Semantic Web: Research and Applications*, Berlin, Germany, 2009, str. 723-737.

[5] White R.W., Richardson M., Bilenko M., Heath A.P., "Enhancing web search by promoting multiple search engine use," v zborniku *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ZDA, Redmond, 2008, str. 43-50.

[6] Selberg E., Etzioni O., "Multi-service search and comparison using the MetaCrawler," v zborniku *Proceedings of the 4th international World Wide Web conference*, ZDA, Seattle, 1995, str. 195-208.

[7] Keyhanipour A. H., Piroozmand M., Moshiri B., Lucas, C., "A multi-layer/multi-agent architecture for meta-search engines," v zborniku *Proceedings of ICGST International Conference on Artificial Intelligence and Machine Learning (AIML-05)*, Egypt, Cairo, 2005.

[8] Page L., Brin S., Motwani R., Winograd, T., "The pagerank citation ranking: Bringing order to the web," 1998.

[9] Kleinberg J.M., "Authoritative sources in a hyperlinked environment," v zborniku *Journal of the ACM*, št. 5, 1999, str. 604-632.

[10] Manning C.D., Raghavan P., Schuetze H., "An introduction to information retrieval," New York:Cambridge University Press, 2008.

[11] Lempel R., Moran, S., "The stochastic approach for link-structure analysis (SALSA) and the TKC effect1," v zborniku *Computer Networks*, št. 1-6, zv. 33, str. 387-401, 2000.

[12] Herbrich R., Graepel T., Obermayer K., "Large margin rank boundaries for ordinal regression," v zborniku *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, str. 115-132, 1999.

[13] Joachims T., "Optimizing search engines using clickthrough data," v zborniku *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[14] Cortes C., Vapnik V., "Support-vector networks," v zborniku *Machine learning*, zv. 20, št. 3, str. 273-297, 1995.

[15] Liu B., *Web data mining*, New York: Springer, 2007, pogl. 6.9, 9.

[16] Muslea I., Minton S., Knoblock C., "A hierarchical approach to wrapper induction," v zborniku *Proceedings of the third annual conference on Autonomous Agents*, str. 190-197, 1999.

[17] Muggleton S.H., "Inductive logic programming," v zborniku *New Generation Computing*, zv. 8, št. 4, str. 295-318, 1991.

[18] Štajner T., Rusu D., Dali L., Fortuna B., Mladenič D., Grobelnik M., "ENRYCHER–SERVICE ORIENTED TEXT ENRICHMENT," v zborniku *Proceedings of SiKDD*, 2009.

[19] Pennacchiotti M., Pantel P., "Entity extraction via ensemble semantics," v zborniku *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, str. 238-247, 2009.

[20] Pasca M., Lin D., Bigham J., Lifchits A., Jain A., "Organizing and searching the world wide web of facts-step one: the one-million fact extraction challenge," v zborniku *Proceedings of the National Conference on Artificial Intelligence*, zv. 21, št. 2, str. 1400-1405, 2006.

[21] Knjižnica SimMetrics. Dostopno na:
http://www.dcs.shef.ac.uk/∼sam/simmetrics.html

[22] Bhattacharya I., Getoor L., "Collective entity resolution in relational data," v zborniku *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2007.

[23] Singh R., Vatsa M., Ross A., Noore A., "Biometric classifier update using online learning: A case study in near infrared face verification," v zborniku *Image and Vision Computing*, 2010.

[24] Geleijnse G., Korst J., "Automatic ontology population by googling," v zborniku *Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2005)*, str. 120-126, 2005.

[25] Gruber T.R., "A translation approach to portable ontology specifications," v zborniku *Knowledge acquisition*, str. 199-226, 1993.

[26] Bush V., "AsWe May Think," v zborniku *The Atlantic Monthly*, zv. 176, št. 1, str. 101-108, 1945.

[27] Broder A., "A taxonomy of web search," v zborniku ACM Sigir Forum, zv. 36, št. 2, str. 3-10, 2002.

[28] Porter M. F., "An Algorithm For Suffix Stripping," v zborniku *Program*, zv. 14, št. 3, str. 130-137, 1980.

[29] (2007) Search 3.0: The Blended & Vertical Search Revolution. Dostopno na:
http://searchengineland.com/search-30-the-blended-vertical-search-revolution-12775

[30] (2008) Search 4.0: Social Search Engines & Putting Humans Back In Search. Dostopno na:
http://searchengineland.com/search-40-putting-humans-back-in-search-14086

[31] Deerwester S., Dumais S.T., Furnas G.W., Landauer T.K., Harshman R., "Indexing by latent semantic analysis," v zborniku *Journal of the American society for information science*, zv. 41, št. 6, str. 391-407, 1990.

[32] Golub G.H., Van Loan C.F., "Matrix computations," Baltimore: Johns Hopkins University Press, 1983.

[33] Gyongyi Z., Garcia-Molina H., Pedersen J., "Combating web spam with trustrank," v zborniku *Proceedings of the Thirtieth international conference on Very large data bases*, str. 576-587, 2004.

[34] (2010) INTERPOL Asks Public To Help Find International Fugitives. Dostopno na: http://www.eurasiareview.com/201007074527/interpol-asks-public-to-help-find-international-fugitives.html

[35] Russom P., "BI Search and Text Analytics - New Additions to the BI Technology Stack," v zborniku *TDWI Best Practices Report*, 2007.

[36] Bradford R., "An Empirical Study of Required Dimensionality for Large-scale Latent Semantic Indexing Applications," v zborniku *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, ZDA, California, str. 153–162, 2008.