UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

JANEZ BRANK

OBRAVNAVA UČNIH PROBLEMOV NA VELIKIH HIERARHIJAH RAZREDOV KOT VEČ DVORAZREDNIH PROBLEMOV

DOKTORSKA DISERTACIJA

Mentor:
prof. dr. Ivan Bratko

Somentorica:
izr. prof. dr. Dunja Mladenić

Ljubljana, 2012

Povzetek

Ontologija je konceptualizacija nekega problemskega področja, namenjena praviloma temu, da si jo deli več uporabnikov. Običajno jo sestavljajo koncepti, primeri, hierarhična relacija med koncepti (*is-a*) in potencialno tudi druge relacije in atributi. Naloga obravnava več problemov na ontologijah z vidika strojnega učenja (pri čemer si lahko preprosto ontologijo predstavljamo kot hierarhijo razredov): populacijo (polnjenje) ontologije oz. klasifikacijo vanjo, evaluacijo ontologij, razvoj ontologij skozi čas ter ekstrakcijo ontoloških podatkov iz zbirke besedil.

Največ se ukvarjamo s problemom klasifikacije v hierarhijo razredov, ki si ga lahko predstavljamo kot eno od možnosti za dodajanje podatkov v ontologijo. Večrazrednega klasifikacijskega problema se lahko lotimo tako, da ga predelamo v več dvorazrednih (binarnih) problemov, zanje naučimo binarne klasifikatorje in njihove napovedi skombiniramo z neke vrste glasovanjem. Povezavo med razredi prvotnega problema in novimi binarnimi problemi lahko jedrnato predstavimo s *kodno matriko*. Še posebej zanimivo vprašanje je, ali lahko dobre rezultate pri klasifikaciji dosežemo že z majhnim številom binarnih klasifikatorjev.

Število vseh možnih kodnih matrik (in tako tudi ansamblov binarnih klasifikatorjev, ki jih te matrike določajo) je eksponentno veliko v odvisnosti od števila razredov prvotnega problema. Čeprav je torej ta prostor kodnih matrik v splošnem neobvladljivo velik, ga lahko vendarle preiščemo dobršen del, če je število razredov v prvotnem problemu majhno. Predstavili smo obsežne poskuse na enem takem večrazrednem problemu in raziskali, kakšna je porazdelitev klasifikacijskega uspeha tako dobljenih ansamblov v odvisnosti od števila binarnih klasifikatorjev v ansamblu. Pokazali smo, da je mogoče dobre rezultate pri klasifikaciji doseči že z majhnim ansamblom, vendar je takšne ansamble težko najti; po drugi strani pa, če dovolimo večji ansambel, je lažje doseči dober klasifikacijski uspeh, vendar pa uspeh najboljšega takega ansambla ni nič boljši kot pri manjših ansamblih. Raziskali smo tudi pogosto omenjano trditev, da je za kodno matriko koristno, če so si vrstice oz. stolpci čim bolj različni (npr. po Hammingovi razdalji). Izkaže se, da matrike z bolj različnimi vrsticami oz. stolpci v povprečju res dajejo uspešnejše ansamble, vendar pa ravno najuspešnejši ansambli sploh ne maksimizirajo omenjenih razlik med vrsticami oz. stolpci.

Razvili smo požrešni algoritem, ki gradi kodno matriko postopoma, stolpec za stolpcem, in temelji na zamisli, naj bi se binarni klasifikator, ki ga

določa novi stolpec matrike, osredotočil na razločevanje med tistimi razredi, na katerih je dosedanji ansambel naredil največ napak. Poskusi kažejo, da doseže ta algoritem že z manjšim ansamblom podobno točnost napovedi, za kakršno preprostejši pristop (naključne matrike) potrebuje večji ansambel. Predstavili smo tudi analizo, ki kaže, kako je vpliv posameznega novega klasifikatorja na napovedi ansambla kot celote omejen, celo v primerih, ko dovolimo klasifikatorjem različne uteži pri glasovanju.

Obravnavamo tudi problem evolucije ontologij, torej njihovega spreminjanja skozi čas, še posebej z vidika napovedovanja strukturnih sprememb v ontologiji. Raziskali smo spreminjanje ontologije spletnega imenika Open Directory Project (ODP) skozi več let, identificirali več pogostejših tipov strukturnih sprememb in razvili hevristične pristope, s katerimi jih lahko prepoznavamo in štejemo. Izkaže se, da je najpogostejša vrsta sprememb dodajanje novih kategorij; razvili smo pristop, ki temelji na strojnem učenju in skuša napovedovati, kje v ontologiji se bo pojavila nova podkategorija (ki bo prevzela nekaj dokumentov iz neke že obstoječe kategorije).

Evaluacija (vrednotenje) ontologij obsega številne pristope in tehnike za ocenjevanje in primerjanje ontologij. Predstavili smo pregled teh tehnik in jih razvrstili glede na to, kateri nivo oz. vidik ontologije ocenjujejo in glede na njihov splošni pristop (primerjava z zlatim standardom, uporaba v aplikaciji, primerjava s podatki, ročno ocenjevanje). Definirali smo mero za evaluacijo ontologij v scenarijih, ko je ontologija v bistvu hierarhija razredov in jo želimo primerjati z „zlatim standardom“, ki je v tem primeru še ena hierarhija razredov, zgrajena nad isto množico primerov. Pokazali smo, kako se ta mera odziva na različne vrste strukturnih sprememb v ontologiji.

Na koncu obravnavamo še en pristop k polnjenju ontologije, namenjeno ontologijam, ki so zbirke splošnonamenskega znanja, ne pa hierarhije dokumentov. Cilj tega pristopa je iz zbirke besedil v naravnem jeziku ekstrahirati trojice oblike $\langle koncept_1,\ relacija,\ koncept_2 \rangle$. Poleg trojic, ki se neposredno pojavljajo v vhodnih besedilih, nas zanimajo tudi bolj abstraktne trojice, ki jih dobimo tako, da eno ali več komponent zamenjamo z nadpomenkami. Razvili smo učinkovit algoritem, ki lahko obdela velik korpus besedil in odkrije vse trojice, ki dosegajo nek predpisani minimalni prag podpore ne glede na nivo abstrakcije. Vendar pa visoka podpora sama po sebi še ni zadosten pogoj za to, da je trojica zanimiva z vidika vključitve v ontologijo, saj so mnoge trojice z visoko podporo irelevantne ali pa preveč abstraktne, da bi bile zanimive. Pokazali smo več hevristik, ki skušajo odkriti zanimive trojice tako, da primerjajo podporo trojice s podporo njihovih prednic in sosed v prostoru trojic. Te hevristike smo preizkusili tudi eksperimentalno, pri čemer smo primerjali njihove napovedi z ročnimi ocenami relevantnosti.

Ključne besede: strojno učenje, klasifikacija, kodne matrike, ontologije, evolucija ontologij, evaluacija ontologij, ekstrahiranje informacij, odkrivanje zakonitosti v besedilih

Abstract

Machine learning on large class hierarchies by transformation into multiple binary problems

An ontology is a shared conceptualization of some problem domain, usually consisting of concepts, instances, an *is-a* hierarchy between concepts, and possibly other relations and attributes. This thesis deals with several problems on ontologies from a machine-learning perspective, in which a simple ontology can be seen as a hierarchy of classes: ontology population (classification), ontology evaluation, ontology evolution (predicting structural change) and extraction of ontological data from a corpus of documents.

We particularly focus on the problem of classification into a hierarchy of classes, which can be seen as one way to populate an ontology. One approach to deal with multi-class problems such as this one is to convert them into several binary (two-class) problems and use a voting scheme to combine the predictions of the resulting ensemble of binary classifiers. The relationship between the classes of the original multi-class problem and the new binary problems can be concisely described by a *coding matrix*. A particularly interesting question is whether good classification performance can be achieved with a small number of binary classifiers.

The number of different coding matrices (and thus of the ensembles defined by them) is exponentially large in the number of classes of the original problem. Although this space of coding matrices is intractably large, a substantial amount of it can be explored if the number of classes in the original problem is small. We present extensive experiments on one such small dataset and investigate the distribution of classification performance scores as a function of the number of binary classifiers in the ensemble. We demonstrate that good classification performance can be achieved with a small ensemble, but such an ensemble might be hard to find; on the other hand, allowing a larger ensemble makes it easier to achieve good performance but does not lead to further increases in the maximal performance (over all ensembles of that size). We also investigate the well-known claim that high row and column separation (average Hamming distance between rows/columns of the matrix) are important properties of the coding matrix, and show that while matrices with high separation do indeed tend to perform well, maximizing row/separation does not lead to the best-performing matrices.

We present a greedy algorithm that constructs the coding matrix one column at a time, based on the idea that the binary classifier defined by the new column should focus on separating those pairs of classes which are most frequently confused by the existing ensemble of classifiers. An empirical evaluation shows that this algorithm allows us to achieve comparable performance with a smaller number of classifiers, compared to a baseline

random-matrix approach. We also present an analysis which demonstrates that the impact of adding a single new classifier to the ensemble is necessarily quite limited, even if weighted voting is taken into account.

We also deal with the topic of ontology evolution, in particular of predicting structural changes in an ontology. We studied the evolution of the Open Directory Project (ODP) ontology over several years, identified several common types of structural changes, and developed a heuristic approach to recognize them and quantify their frequency. The most common structural change turned out to be the addition of new categories, and we present a machine-learning approach to predicting where a new subcategory might be added by taking a few documents from an existing category.

Ontology evaluation consists of various approaches and techniques for evaluating and comparing ontologies. We present a survey of such techniques and classify them depending on which level or aspect of ontologies they focus on, as well as depending on their general approach (gold-standard based, application based, data-driven, and manual evaluation). We introduce an ontology evaluation measure for scenarios where the ontology is a hierarchy of classes and is to be compared to a "gold standard" ontology built over the same set of instances. We investigate how this measure responds to various kinds of structural changes in the ontology.

We also discuss another approach to ontology population, aimed at "general knowledge" ontologies rather than document hierarchies. In this approach the goal is to extract useful triples of the form $\langle concept_1, relation, concept_2 \rangle$ from a corpus of natural-language documents. In addition to triples that directly occur in the corpus, we also consider more abstract triples that can be obtained by replacing one or more components by a hypernym. We developed an efficient algorithm that can process a large corpus of documents and extract triples that satisfy a minimum-support threshold at any level of abstraction. However, a high support by itself is not a sufficient condition for a triple to be interesting for inclusion in the ontology, since many triples with a high support are irrelevant or too abstract to be interesting. We show several heuristics that can be used to identify interesting triples by comparing their support to that of their ancestors or neighbors in the triple space. We evaluated these heuristics experimentally by comparing their results to manually assigned relevance labels.

Keywords: machine learning, classification, coding matrices, ontologies, ontology evolution, ontology evaluation, information extraction, text mining

Zahvala

Na tem mestu bi se rad zahvalil: prof. Ivanu Bratku za mentorstvo pri doktorski disertaciji; Dunji Mladenić za somentorstvo in še posebej za številne koristne nasvete in spodbude pri pisanju besedila naloge; Marku Grobelniku, ki je veliko pripomogel k temu, da sem se začel zanimati za probleme, ki so obravnavani v tej disertaciji; prof. Igorju Kononenku in prof. Johnu Shawe-Taylorju za mnoge dobrodošle pripombe in nasvete, ki sta jih podala ob recenziji moje disertacije; Rayidu Ghaniju, čigar delo je spodbudilo moje zanimanje za uporabo kodnih matrik pri klasifikacijskih problemih; in Juretu Leskovcu, Blažu Novaku, Blažu Fortuni ter verjetno še komu za koristne debate in nasvete.

Kazalo

Poglavje 1

Uvod

1.1 Opis ožjega znanstvenega področja

Raziskave, ki jih obravnava ta doktorska disertacija, se uvrščajo na področje strojnega učenja [48, 37], natančneje na področje napovednega modeliranja ali nadzorovanega učenja. Z metodami tega področja obravnavamo več problemov, povezanih s tekstovnimi ontologijami, s poudarkom na klasifikaciji dokumentov v ontologijo. Vsebina disertacije se od naslova nekoliko razlikuje oz. je v primerjavi z njim razširjena; do razlike je prišlo, ker se je tematika raziskave tekom let od odobritve teme naprej nekoliko spreminjala in pojavili so se novi relevantni problemi na ontologijah, ki so ponujali možnosti za obravnavo z metodami strojnega učenja (in ki so sprva obetali več povezave s problematiko klasifikacije, kot pa se je na koncu izkazalo). Zato smo v raziskavo vključili še nekaj problemov na ontologijah: evaluacija ontologije (oz. primerjava ontologij), napovedovanje strukturnih sprememb v ontologiji in dodajanje novih dejstev v ontoogijo.

Nadzorovano učenje se ukvarja s problemi, pri katerih iščemo modele za napovedovanje vrednosti ene ali več ciljnih (odvisnih) spremenljivk v odvisnosti od vrednosti neodvisnih spremenljivk (ki jih pogosto imenujemo tudi atributi). Če je ciljna spremenljivka diskretna in je njena zaloga vrednosti omejena, govorimo o klasifikacijskem problemu, modelu pa v takem primeru pravimo tudi klasifikator. Algoritmi strojnega učenja ponavadi predpostavijo, da so na voljo učni podatki, torej primeri, za katere so znane tako vrednosti atributov kot vrednost ciljne spremenljivke, naloga učnega algoritma pa je sestaviti oz. poiskati primeren model, ki bo pravilno napovedoval ciljno spremenljivko tudi na še nevidenih primerih.

Poseben primer klasifikacijskih problemov so dvorazredni ali binarni problemi, torej taki, v katerih ima ciljna spremenljivka le dve možni vrednosti. Enega od obeh razredov v tem primeru običajno imenujemo pozitivni razred, drugega pa negativni razred. Na področju strojnega učenja je bilo predlaganih že precej metod za učenje klasifikacijskih modelov in nekatere med njimi so zasnovane le za dvorazredne probleme; zelo uspešna taka metoda je na primer metoda podpornih vektorjev [7, 15]. Ker pa imajo mnogi

zanimivi klasifikacijski problemi več kot dva razreda, se pojavi vprašanje, kako si lahko z metodami za učenje dvorazrednih klasifikatorjev pomagamo pri reševanju večrazrednih klasifikacijskih problemov.

V ta namen lahko na podlagi prvotnega večrazrednega problema definiramo več novih dvorazrednih problemov in za vsakega od njih naučimo po en dvorazredni klasifikator, njihove napovedi pa skombiniramo v končno napoved za prvotni večrazredni problem. Posamezni dvorazredni problemi so praviloma določeni tako, da predstavnike nekaterih razredov prvotnega večrazrednega problema obravnavamo kot pozitivne primere, predstavnike nekaterih drugih razredov obravnavamo kot negativne primere, predstavnikov ostalih razredov prvotnega problema pa pri tistem konkretnem dvorazrednem problemu sploh ne uporabimo. Iz tako določene skupine dvorazrednih problemov dobimo po učenju skupino modelov, od katerih zna vsak razločevati nekatere razrede prvotnega problema od nekaterih drugih. Napoved takšnega modela si lahko razlagamo kot glas za razrede, ki smo jih pri gradnji tistega modela uporabili kot pozitivne, in glas proti tistim razredom, ki smo jih pri učenju tega modela uporabili kot negativne. Klasifikator za prvotni večrazredni problem kot svojo napoved uporabi tisti razred, ki je pri tem dobil največ glasov od posameznih dvorazrednih modelov.

Strukturo takšne skupine dvorazrednih modelov za reševanje večrazrednega problema lahko opišemo s kodno matriko — dvodimenzionalno tabelo, v kateri vsak element pove, kako je bil posamezni razred prvotnega problema uporabljen pri gradnji posameznega izmed dvorazrednih modelov. Predlaganih je bilo več pristopov za izbor kodne matrike: eden proti enemu (za vsak par razredov zgradimo en model, ki razločuje ta dva razreda), eden proti ostalim (za vsak razred zgradimo en model, ki razločuje ta razred od ostalih), naključno zapolnjene kodne matrike in kodne matrike, temelječe na kodih za odpravljanje napak iz teorije informacij [17]. Če ima prvotni problem veliko število razredov, ti pristopi zahtevajo učenje velikega števila dvorazrednih modelov, kar pripelje do velike časovne in prostorske zahtevnosti (poraba pomnilnika za hrambo vseh dvorazrednih modelov). V okviru disertacije se ukvarjamo z iskanjem čim boljših kodnih matrik za probleme z velikim številom razredov, s poudarkom na tem, da naj število potrebnih dvorazrednih modelov ostane obvladljivo. Posebej nas zanimajo primeri, ko so razredi prvotnega večrazrednega problema urejeni hierarhično, kar je pri problemih z velikim številom razredov razmeroma pogost pojav.

1.2 Izhodišča in obstoječe metode

Zanima nas uporaba kodnih matrik pri klasifikacijskih problemih z velikim številom razredov, atributov in učnih primerov. V nadaljevanju na kratko predstavljamo pomembnejše metode in raziskave, na katerih bo temeljilo delo v okviru pričujoče doktorske disertacije.

1.2.1 Metoda podpornih vektorjev

Metoda podpornih vektorjev [7, 15] je ena od najbolj znanih metod strojnega učenja na dvorazrednih problemih. Osnovna različica te metode predpostavi, da so primeri opisani z nekaj atributi, zaloga vrednosti vseh atributov pa je množica realnih števil, tako da lahko na primere gledamo kot na točke v nekem večrazsežnem realnem vektorskem prostoru. Naloga učnega algoritma je razdeliti ta prostor na dva dela, namreč tistega, v katerem pripadajo točke pozitivnemu razredu, in tistega, v katerem pripadajo negativnemu razredu. Metoda podpornih vektorjev razmeji oba dela s hiperravnino, ki jo poišče kot rešitev optimizacijskega problema, pri katerem zasleduje dva cilja: učni primeri naj bi ležali na pravi strani ravnine (pozitivni na eni strani, negativni na drugi) in čim dlje stran od nje. Model je tako opisan z enačbo uporabljene hiperravnine, pri klasifikaciji pa je treba le pogledati, na kateri strani ravnine leži posamezen primer. Z majhnimi spremembami lahko metoda podpornih vektorjev namesto hiperravnin odkriva tudi modele, ki uporabljajo nelinearno razmejitveno ploskev (učinek je tak, kot da bi primere pred učenjem nelinearno preslikali v nek nov vektorski prostor in v njem iskali ravnino [70]); podobno lahko metodo razširimo tudi na podatke, ki niso eksplicitno predstavljeni z vektorji. Obstaja tudi več različic te metode, ki uporabljajo drugačne kriterije v optimizacijskem problemu za izbor ravnine [78]. Prilagodili so jo že tudi za nekatere druge probleme strojnega učenja, na primer regresijske probleme [72], za ocenjevanje gostote verjetnostnih porazdelitev [67] in za probleme rangiranja [35]. Obstajajo tudi različice metode podpornih vektorjev za večrazredne klasifikacijske probleme [84], ki pa so pri večjem številu razredov časovno in pomnilniško zelo zahtevne.

Metoda podpornih vektorjev ima več dobrih lastnosti: dobljeni modeli pogosto dosežejo visoko klasifikacijsko točnost; učinkovito deluje tudi v primerih, ko je število atributov veliko; razmeroma uspešno se izogiba pretiranemu prilagajanju; je tudi dobro teoretično podprta [80]. Na nekaterih področjih strojnega učenja, na primer pri klasifikaciji besedil, je metoda podpornih vektorjev ena od standardnih in najpogosteje uporabljanih metod [36, 68]. Med slabosti te metode lahko štejemo razmeroma slabo razumljivost njenih modelov (v primerjavi z nekaterimi drugimi metodami) in pa precejšnjo računsko zahtevnost učenja, sploh v situacijah, ko je število učnih primerov veliko (čeprav je bilo tudi na tem področju že precej napredka [6, 5, 69].

1.2.2 Kodne matrike za večrazredne probleme

Kodne matrike označujejo družino pristopov za prevedbo večrazrednega klasifikacijskega problema v družino dvorazrednih problemov. Mislimo si klasifikacijski problem s k razredi, ki bi ga radi prevedli na m dvorazrednih problemov. Kodna matrika M je tabela velikosti $k \times m$, v kateri element

M_{cj} pove, kako se uporablja primere iz razreda c pri učenju klasifikatorja za j-tega izmed m dvorazrednih problemov: ali obravnavamo te primere kot pozitivne ($M_{cj} = 1$) ali kot negativne ($M_{cj} = -1$) ali pa jih pri učenju tega klasifikatorja sploh ne uporabljamo ($M_{cj} = 0$).

Če odmislimo ničelne elemente matrike, nam c-ta vrstica kodne matrike torej pove, kakšen niz napovedi bi načeloma pričakovali, če bi vsem m dvorazrednim klasifikatorjem predstavili nek primer iz razreda c; od tod izraz kodna matrika: vsak razred od 1 do k smo zakodirali v nek niz m napovedi dvorazrednih klasifikatorjev.

Motivacija za uporabo kodnih matrik je, da lahko z njimi uporabimo tudi na večrazrednih problemih učne algoritme, ki so se izkazali za uspešne pri dvorazrednih problemih. Podobno kot pri metodah, ki temeljijo na ansamblih klasifikatorjev (bagging [10], boosting [22]), se tudi tu lahko kombinacija večjega števila preprostejših modelov lahko izkaže za natančnejšo in robustnejšo, kot če se poskušamo neposredno naučiti nekega večrazrednega modela.

Možnost, da so v matriki tudi ničelni elementi, je načeloma koristna iz več razlogov: posamezni dvorazredni problemi postanejo s tem lažji in imajo manj učnih primerov, zato je učenje dvorazrednih modelov zanje hitrejše, dobljeni modeli pa točnejši. Če je matrika dovolj redka (torej če ima dovolj ničelnih elementov), lahko hranimo v pomnilniku le neničelne elemente in tako prihranimo prostor, pa tudi čas pri klasifikaciji, ko je treba kodno matriko pomnožiti z vektorjem napovedi posameznih dvorazrednih modelov.

Pri klasifikaciji novega primera najprej izračunamo napovedi vseh dvorazrednih modelov in jih združimo v vektor, potem pa ta vektor pomnožimo s kodno matriko. Učinek je ta, da dvorazredni modeli glasujejo o razredu: pozitivno napoved posameznega modela si razlagamo kot glas za tiste razrede, ki smo jih pri učenju tega modela uporabili kot pozitivne, in kot glas proti tistim razredom, ki smo jih pri učenju tega modela uporabili kot negativne; pri negativni napovedi modela pa je ravno obratno. Tak pristop je najpogostejši, obstajajo pa tudi različice, ki napovedi dvorazrednih modelov kombinirajo drugače [55, 87].

Znanih je več družin kodnih matrik. S kodnimi matrikami lahko predstavimo pristope, ki ločujejo en razred od ostalih, in tiste, ki ločujejo posamezne pare razredov med sabo (za vsak par razredov zgradijo po en dvorazredni model, ki razločuje ta dva razreda) [2]. Nekateri avtorji so uporabljali naključno zapolnjene matrike [3]. Sestavimo lahko tudi matriko, v kateri so vsa možna razbitja množice razredov na dva dela, vendar zahteva takšna matrika nepraktično veliko dvorazrednih modelov. Načeloma si od dobre kodne matrike želimo, da bi imela čim bolj različne vrstice in tudi čim bolj različne stolpce. (Različnost med vrsticami skrbi za to, da bomo pri primerih iz različnih razredov dobili čim bolj različne vektorje napovedi dvorazrednih modelov; glasovanje pri klasifikaciji pravzaprav primerja vektor napovedi z vrsticami matrike in bolj ko so si vrstice matrike med sabo različne, manj

je verjetno, da se bomo zaradi napačne napovedi kakšnega od dvorazrednih odločili za napačno vrstico in s tem za napačni razred. Različnost med stolpci pa naj bi zagotovila, da si bodo posamezni dvorazredni problemi čim bolj različni med sabo in se ne bo prepogosto zgodilo, da se več dvorazrednih klasifikatorjev zmoti hkrati na istem primeru; to bi namreč otežilo prepoznavanje pravega razreda tistega primera.) Zato za gradnjo kodnih matrik včasih uporabljajo tudi kode za odpravljanje napak, ki so znani iz teorije informacij in imajo želene lastnosti [17]. Dekel in Singer [16] pa sta kodne matrike posplošila tako, da sta vanje vključila tudi necele elemente. Kodna matrika nam v njuni formulaciji definira preslikavo iz množice razredov v prostor $\mathbb{R}^m$, namesto nabora m dvorazrednih klasifikatorjev pa imamo še eno preslikavo iz prostora primerov (instanc) v $\mathbb{R}^m$; algoritem Dekela in Singerja se uči obeh preslikav skupaj. Ker znamo zdaj tako primere kot razrede preslikati v isti prostor, $\mathbb{R}^m$, lahko napovedujemo preprosto tako, da za dani primer x pogledamo, kateri razred ima v $\mathbb{R}^m$ sliko, ki je najbližja sliki primera x.

1.2.3 Tekstovne ontologije

Ontologija je eksplicitna konceptualizacija nekega problemskega področja, namenjena temu, da jo skupaj uporablja več ljudi [29]. Ponavadi jo sestavljajo koncepti, instance, relacije, atributi in podobne reči, za opis ontologije pa se pogosto uporablja formalne jezike, kakršna sta RDF in OWL. V naši raziskavi se bomo večinoma omejili na tematske hierarhije kot poseben primer ontologij. V takšni ontologiji kot instance (primeri) nastopajo dokumenti (besedila v naravnem jeziku), kot koncepti ali razredi pa nastopajo tematske kategorije; slednje pa so organizirane v hierarhično (drevesasto) relacijo, ki si jo pogosto predstavljamo kot *is-a*, čeprav bi bil njen natančnejši opis nekaj takega kot „če se dokument nanaša na temo A, se nanaša tudi na temo B“. Značilni primeri takšnih ontologij so hierarhična kazala spletnih strani, kot so ODP (Open Directory Project, `http://dmoz.org`, ki bo tudi podlaga za večino naših poskusov), Yahoo!, matkurja.com, www.hr in podobni.

V zvezi s takšno tekstovno ontologijo je mogoče definirati več problemov oz. operacij, ki jih lahko poskusimo reševati na bolj ali manj avtomatizirane načine in pri tem uporabljati tudi metode strojnega učenja. Poleg klasifikacije v ontologijo (kar smo že videli zgoraj) si bomo ogledali še dve: spremljanje in napovedovanje strukturnih sprememb ter primerjanje oz. evaluacijo ontologij.

1.2.4 Strukturne spremembe v ontologijah

Izhodišče, da naj bo ontologija konceptualizacija nekega področja, v praksi pogosto pomeni, da se mora ontologija včasih tudi spreminjati, bodisi zato, ker se spreminja opazovano področje ali pa potrebe uporabnikov glede

njegove konceptualizacije. Področje, ki se ukvarja s spremembami v ontologijah, se imenuje evolucija ontologij; obstoječa dela na tem področju se ukvarjajo med drugim s tem, kakšne so formalne posledice določenih sprememb z vidika formalnega jezika, v katerem je ontologija zapisana; pa z vprašanji kompatibilnosti različnih verzij ontologije in tega, kako vključiti spreminjajoče se ontologije v informacijske sisteme za delo z ontologijami [85]; v novejšem času pa tudi s tem, kako evolucijo ontologij do neke mere avtomatizirati [86].

Pri tematskih hierarhijah so najpogostejše spremembe dodajanja in brisanja dokumentov, poleg njih pa prihaja tudi do strukturnih sprememb, kot so dodajanje in brisanje kategorij, premiki kategorij, razcepitve kategorij in podobno. V naši raziskavi smo spremljali razvoj ontologije ODP (za njeno evolucijo ročno skrbijo človeški uredniki) skozi več let in s primerjavo stanja ontologije iz meseca v mesec identificirali posamezne vrste strukturnih sprememb. Eno od najpogostejših vrst strukturnih sprememb, to je razcepitev kategorije na več podkategorij, smo si ogledali kot problem strojnega učenja in poskušali takšne spremembe v ontologiji tudi napovedovati.

1.2.5 Primerjanje in evaluacija ontologij

Evaluacija ontologij se ukvarja z vprašanjem kako dobra, primerna oz. ustrezna je ontologija z nekega določenega vidika. Znani so številni pristopi k evaluaciji ontologij [8, 9], ki se razlikujejo po tem, na kateri vidik ontologije se osredotočajo (koncepti, terminologija, hierarhija, relacije, sintaksa, struktura oz. design ontologije, njena vklopljenost v širši kontekst) in na kakšen način jo vrednotijo oz. ocenjujejo (primerjava z zlatim standardom, primerjava z zunanjimi podatki o problemskem področju, ocenjevanje prek uporabe v aplikaciji, človeško ocenjevanje prek več kriterijev).

Ena od možnosti za ocenjevanje ontologij je primerjava z neko že obstoječo ontologijo, ki jo vzamemo za zgled oz. „zlati standard“. Potrebujemo torej mero podobnosti med ontologijami, z njo pa lahko potem ocenjujemo ontologije tako, da pogledamo, kako podobne so zlatemu standardu. Osredotočili smo se na scenarij, pri katerem nas zanimajo tematske ontologije, ki jih sestavljajo dokumenti, zloženi v hierarhijo kategorij. Definirali smo mero podobnosti, ki primerja dve takšni ontologiji, zgrajeni nad isto množico dokumentov (ki so povezani vsakič v drugačno hierarhijo), in preučili, kako se ta mera odziva na različne vrste strukturnih sprememb v ontologiji.

1.2.6 Dodajanje trojic v ontologijo

Malo drugačno, a tudi pogosto uporabljano vrsto ontologij sestavljajo splošnonamenske zbirke vsakdanjega znanja (*common sense ontologies*), kot je na primer Cyc [38]. Takšna ontologija mora vsebovati veliko število konceptov, tako konkretnih (ki predstavljajo osebe, kraje, dogodke, dejanja ipd.) kot

bolj abstraktnih, povezani pa so z relacijami, ki skupaj s koncepti tvorijo množico „dejstev“ oz. trojic oblike $\langle koncept_1, relacija, koncept_2 \rangle$. Pomemben sestavni del takšne ontologije so še pravila sklepanja, ki omogočajo avtomatsko sklepanje nad dejstvi. Primer takšne trojice je na primer $\langle person, live, country \rangle$, ki vsebuje (za človeka vsakdanje) znanje o tem, da je za ljudi med drugim značilno to, da lahko živijo v državah, in da je država med drugim nekaj, v čemer lahko ljudje prebivajo.

Pomembno vprašanje pri ontologijah je, kako do neke mere avtomatizirati njihovo sestavljanje oz. gradnjo; temu procesu pogosto pravimo polnjenje oz. poseljevanje ontologije (*ontology population*). Mnogo potencialno zanimivih trojic, ki bi prišle v poštev za splošnonamensko ontologijo, se skriva v nestrukturiranih besedilih v naravnem jeziku, ki jih lahko v velikem obsegu najdemo na spletu ali v raznih besedilnih korpusih. Eden od pristopov za polnjenje splošnonamenske ontologije je, da skušamo iz besedil v naravnem jeziku s sintaktično analizo pridobiti trojice oblike $\langle osebek, povedek, predmet \rangle$, pri čemer sta osebek in predmet dva koncepta, povedek pa odraža relacijo med njima. Za potrebe polnjenja ontologije moramo takšne trojice primerno abstrahirati in oceniti, katere so dovolj zanimive oz. relevantne, da bi jih bilo smiselno dodati v ontologijo. Opisali smo postopek, ki lahko učinkovito obdela velike količine besedil in identificira pogoste trojice na vseh nivojih abstrakcije, predstavili pa smo tudi nekaj hevristik za ocenjevanje relevantnosti trojic.

1.3 Izvirni prispevki disertacije

Prostor kodnih matrik. S sistematičnimi poskusi na majhni problemski domeni smo raziskali prostor kodnih matrik in s tem prostor vseh možnih napovednih modelov, ki jih določajo te matrike. Pri tem smo preučevali povezave med lastnostmi matrike (na primer povprečna Hammingova razdalja med vrsticami ali med stolpci) in klasifikacijsko uspešnostjo iz nje izhajajočega napovednega modela. Pokazali smo, da je mogoče uspešne modele dobiti tudi iz matrik z majhnim številom stolpcev, vendar so veliko redkejši kot v prostoru matrik z večjim številom stolpcev.

Postopna gradnja kodnih matrik. Razvili smo pristope, ki skozi uporabo kodnih matrik za prevedbo večrazrednega klasifikacijskega problema na več dvorazrednih poblemov obravnavajo probleme klasifikacije v hierarhijo razredov (kot poseben primer ontologije, ki se omejuje le na relacijo *is-a*). Naš pristop se od dosedanjega dela na področju kodnih matrik razlikuje po tem, da dopušča hierarhično organiziranost razredov in da matriko gradi postopoma, stolpec za stolpcem, vsak naslednji stolpec pa je izbran tako, da se bo pripadajoči dvorazredni model osredotočal na razrede, na katerih naredi dotedanji nabor dvorazrednih modelov največ napak. Predlagani pristop smo ovrednotili eksperimentalno in analitično in opozorili na

razloge, ki omejujejo vpliv posameznega novega modela na obnašanje celotnega ansambla.

Strukturne spremembe v ontologijah. Z analizo sprememb tematske ontologije ODP (*Open Directory Project*, dmoz.org) skozi čas smo postavili tipologijo strukturnih sprememb, ki jih v ontologijo skozi čas vnašajo njeni uredniki kot odgovor na spremembe na področju, ki ga ontologija obravnava. Osredotočili smo se na eno vrsto strukturnih sprememb, namreč dodajanje novih podkategorij, in predlagali pristop, ki temelji na strojnem učenju in skuša napovedovati, kje in kdaj bo do takšnih strukturnih sprememb prišlo.

Primerjanje in vrednotenje ontologij. Sestavili smo tipologijo številnih pristopov, ki so se v literaturi uporabljali za primerjanje in vrednotenje ontologij. V nadaljevanju smo razvili svoj pristop, namenjen vrednotenju ontologij na podlagi primerjave z zlatim standardom, pri čemer smo se omejili na primere, ko je ontologija sestavljena iz množice dokumentov, razvrščenih v hierarhijo. Opisali smo mero podobnosti za primerjanje dveh različnih hierarhičnih razbitij iste množice dokumentov in s poskusi na realnih podatkih iz ontologije ODP pokazali, kako se opisana mera odziva na različne vrste strukturnih sprememb v ontologiji.

Odkrivanje zanimivih trojic v besedilu. Razvili smo pristop, ki skuša iz korpusa besedil ekstrahirati zanimive relacijske trojice oblike $\langle koncept_1,\ relacija,\ koncept_2 \rangle$, z namenom, da bi tako pomagali pri polnjenju splošnonamenske ontologije ali baze znanja, kot je na primer Cyc [38]. Naša metoda temelji na tem, da iz besedila ekstrahira konkretne relacijske trojice, nato jih z uporabo podatkov o nadpomenkah predela v bolj abstraktne in iz njih na koncu naredi izbor zanimivih trojic. Predstavili smo učinkovit postopek za sistematično odkrivanje vseh posplošenih (abstraktnih) trojic, ki presegajo nek predpisan prag minimalne podpore (pogostosti v korpusu). Predlagamo tudi več hevristik za identifikacijo potencialno zanimivih trojic in jih primerjamo na obsežnem korpusu besedil iz zbirke Reutersovih novic [61].

Poglavje 2

Prostor kodnih matrik

2.1 Uvod

Število možnih kodnih matrik je odvisno od števila razredov v prvotnem večrazrednem problemu in od števila dvorazrednih klasifikatorjev, na katere želimo ta problem prevesti; odvisnost pa je v obeh primerih eksponentna. Pri problemih realistične velikosti je torej prostor vseh možnih kodnih matrik neobvladljivo velik in kakršen koli preiskovalni algoritem ga lahko pregleda le majhen del. Če pa se omejimo na dovolj majhne probleme, postane prostor vseh možnih matrik bolj obvladljive velikosti in ga lahko pregledujemo bolj temeljito. Namen tega poglavja je, da s sistematičnimi poskusi na majhnem problemu razišče lastnosti prostora kodnih matrik in povezave med lastnostmi matrike ter lastnostmi ansambla dvorazrednih klasifikatorjev, ki ga lahko pridobimo iz nje.

2.2 Struktura prostora kodnih matrik

Recimo, da želimo prevesti k-razredni problem na m dvorazrednih problemov. Kodna matrika bo torej velikosti $m \times k$, za vsak element v njej pa so tri možne vrednosti ($+1$, -1 in 0), tako da je načeloma možnih kar 3^{mk} različnih matrik. Mnoge med njimi za naše namene sicer niso primerne, npr. ker vsebujejo po več enakih stolpcev (ali pa je nek stolpec le negacija drugega); ker v kakšnem stolpcu ni prisoten vsaj en element z vrednostjo $+1$ in vsaj eden z vrednostjo -1; ali pa ker kršijo omejitve, ki izvirajo iz hierarhičnih odnosov med razredi. Hierarhični odnosi v tem primeru pomenijo, da so nekateri razredi v odnosu nadrazred/podrazred in da vsak primer, ki pripada podrazredu, s tem implicitno pripada tudi nadrazredu. V razdelku 2.3 bomo na primer videli hierarhijo, v kateri so primeri (instance) dokumenti, razredi pa so tematske kategorije; med njimi so na primer razredi „matematika“, „geometrija“ in „algebra“, pri čemer je prvi nadrazred ostalih dveh. Hierarhični odnos med razredi v tem primeru pomeni, da vsak dokument,

ki govori o geometriji ali algebri, s tem implicitno govori tudi o matematiki. Nekateri dokumenti lahko pripadajo razredu „matematika“ implicitno prek svoje pripadnosti kakšnemu od podrazredov, nekateri pa lahko pripadajo neposredno nadrazredu, ne da bi bili člani kakšnega od podrazredov.

Oceno za število vseh možnih kodnih matrik pri problemu s hierarhijo razredov lahko dobimo z naslednjim razmislekom. Recimo, da naša hierarhija tvori polno drevo s h nivoji, pri čemer so vsi listi na najglobljem, h-tem nivoju, vsako notranje vozlišče pa ima b otrok. Stolpec matrike si lahko predstavljamo tudi kot označitev vozlišč v hierarhiji, pri čemer vsako vozlišče (razred) dobi oznako $+1$, -1 ali 0, odvisno od vrednosti pripadajočega elementa v stolpcu. Naj bo $\hat{f}(h)$ število možnih stanj posameznega stolpca matrike (parametra b ne bomo pisali eksplicitno, ker je pri celotnem razmisleku fiksen), če upoštevamo omejitev, da ko dobi neko vozlišče (razred) v stolpcu neničelno oznako, morajo enako oznako dobiti tudi vsi potomci tega vozlišča. Funkcijo $\hat{f}$ lahko računamo z naslednjim rekurzivnim razmislekom: če ima koren drevesa oznako $+1$, morajo imeti tudi vsa ostala vozlišča oznako $+1$; enako je, če ima koren drevesa oznako -1; če pa ima koren drevesa oznako 0, nam iz tega ne nastanejo nobene dodatne omejitve glede oznak v poddrevesih, tako da lahko vsako od b poddreves označimo na $\hat{f}(h-1)$ različnih načinov. Rekurzivna zveza je torej $\hat{f}(h) = 2 + (\hat{f}(h-1))^b$. Rekurzija se ustavi pri $h = 1$, kjer je celo „drevo“ le še eno samo vozlišče, ki ga lahko označimo poljubno (kot $+1$, -1 ali 0), torej imamo $\hat{f}(1) = 3$.

Ena omejitev, ki je funkcija $\hat{f}$ ni upoštevala, pa je ta, da mora stolpec vsebovati vsaj eno oznako $+1$ in vsaj eno oznako -1, sicer ni smiseln. Naj bo $\tilde{f}(h)$ število stolpcev, ki ne vsebujejo oznak -1. S podobnim razmislekom kot v prejšnjem odstavku pridemo do rekurzivne zveze $\tilde{f}(h) = 1 + (\tilde{f}(h-1))^b$ in $\tilde{f}(1) = 2$. Enako je tudi število stolpcev, ki ne vsebujejo oznak $+1$. Če torej oboje odštejemo od $\hat{f}$ in upoštevamo še, da ni smiselno ločiti med stolpcem in njegovo negacijo, dobimo takšno število vseh možnih stolpcev: $f(h) = (\hat{f}(h) - 2\tilde{f}(h) + 1)/2$. (Prištevanje 1 v tej formuli je posledica tega, da smo stolpec, ki ga sestavljajo same ničle, odšteli dvakrat, saj je zajet v obeh $\tilde{f}(h)$.)

Smiselno je tudi reči, da si ne želimo matrik, v katerih bi bilo po več enakih stolpcev (ali pa en stolpec negacija drugega). Vseh možnih matrik z m stolpci je torej $\binom{f(h)}{m}$.

Pri poskusih v tem poglavju smo vzeli $h = 3$, $b = 2$, kar nam dá hierarhijo s tremi nivoji in 7 kategorijami. Zgoraj opisani razmislek nam pokaže, da je pri $h = 3$, $b = 2$ možnih le 36 različnih stolpcev (ob upoštevanju omejitev, ki izhajajo iz hierarhije med razredi, in če ne ločimo stolpcev od njihovih negacij). Število vseh možnih matrik določene širine je potem takšno:

m	1	2	3	4	5	6	7	8
$\binom{36}{m}$	36	630	7 140	58 905	376 992	1 947 792	8 347 680	30 260 340

Če nas torej na primer zanimajo matrike z največ osmimi stolpci, jih je

vseh skupaj (po vseh širinah od 1 do 8 stolpcev) slabih 41 milijonov. Z nekaj pazljivosti lahko dokaj učinkovito ocenimo klasifikacijski uspeh ansamblov, ki izvirajo iz vseh teh različnih matrik. Upoštevati moramo predvsem to, da imajo vse te številne različne matrike le majhno število različnih stolpcev, torej potrebujemo tudi le majhno število dvorazrednih klasifikatorjev. Vse te dvorazredne klasifikatorje lahko pripravimo vnaprej in izračunamo tudi njihove napovedi pri posameznih testnih primerih. Pri vsaki matriki moramo le še skombinirati napovedi posameznih dvorazrednih klasifikatorjev, iz katerih je sestavljen njen ansambel:

1 Zgeneriramo vse možne stolpce (v našem primeru jih je $f = 36$),
za vsak stolpec $\mathbf{s}$ zgradimo dvorazredni klasifikator in
si zapomnimo napovedi tega klasifikatorja, $\hat{y}_{\mathbf{s}}(x)$, na vseh testnih primerih x. Naj bo $\mathcal{S}$ množica vseh tako dobljenih stolpcev.
2 Naštejemo vseh $\binom{f}{m}$ kombinacij m različnih stolpcev,
torej vse $S \subseteq \mathcal{S}$, za katere je $|S| = m$.
Za vsako od njih:
3 Za vsak testni primer x izračunamo napoved ansambla S:
$$z_S(x) = \arg\max_c \sum_{\mathbf{s}\in S} \hat{y}_{\mathbf{s}}(x) s_c$$
(pri tem je s_c tista komponenta stolpca $\mathbf{s}$, ki se nanaša na razred c).
4 Napovedi primerjamo s pravo razporeditvijo primerov po
razredih in izračunamo oceno ansambla kot celote.

Časovno zahtevnost tega postopka bi se dalo še malo izboljšati z dinamičnim programiranjem, vendar za ceno precej večje porabe pomnilnika, če bi upoštevali, da imajo vsote $\sum_{\mathbf{s}\in S} \hat{y}_{\mathbf{s}}(x) s_c$ pri različnih množicah S lahko veliko skupnih členov, če imajo tiste množice S veliko skupnih elementov. Če bi torej na primer najprej pregledali vse množice $S' \subseteq \mathcal{S}$ velikosti $m-1$ in za vsako od njih izračunali (za vsak c) $\sum_{\mathbf{s}\in S'} \hat{y}_{\mathbf{s}}(x) s_c$, potem bi lahko pri pregledu množic $S \subseteq \mathcal{S}$ velikosti m do vsake vsote $\sum_{\mathbf{s}\in S} \hat{y}_{\mathbf{s}}(x) s_c$ prišli tako, da iz S vzamemo poljuben c, vzamemo že prej izračunano vsoto za $S' := S - \{c\}$ in ji prištejemo le še novi člen $\hat{y}_{\mathbf{s}}(x) s_c$. Še ena možnost je, da si zapomnimo napovedi za vse množice S' velikosti $m/2$ in nato pri vsaki množici S velikosti m izračunamo njeno vsoto $\sum_{\mathbf{s}\in S} \hat{y}_{\mathbf{s}}(x) s_c$ tako, da množico S razdelimo na dve disjunktni podmnožici velikosti $m/2$, njuni vsoti imamo že izračunani in ju moramo le še sešteti.

2.3 Poskusi

Za sistematične poskuse v tem poglavju smo uporabili dve majhni hierarhiji s po tremi nivoji in sedmimi razredi. Vzeli smo naslednje kategorije iz ontologije ODP (Open Directory Project, `dmoz.org`):

- Hierarhija 1:

  Top/Science
  Top/Science/Math
  Top/Science/Math/Geometry
  Top/Science/Math/Algebra
  Top/Science/Physics
  Top/Science/Physics/Quantum_Mechanics
  Top/Science/Physics/Relativity

- Hierarhija 2:

  Top/Science
  Top/Science/Biology
  Top/Science/Biology/Botany
  Top/Science/Biology/Immunology
  Top/Science/Social_Sciences
  Top/Science/Social_Sciences/Economics
  Top/Science/Social_Sciences/Archaeology

Izmed dokumentov, ki so bili v posamezni kategoriji prisotni v ODP, smo naključno izbrali za vsako kategorijo po 100 učnih in 100 testnih primerov.

Za ocenjevanje napovedi ansambla smo uporabili Jaccardovo mero podobnosti med množico prednikov napovedane in prave kategorije. Naj bo A_c množica vseh prednikov kategorije c v naši hierarhiji (vključno s c samo); potem, če nek primer x v resnici pripada razredu c, ansambel pa je zanj napovedal, da pripada razredu c', je Jaccardova ocena te napovedi definirana kot $|A_c \cap A_{c'}|/|A_c \cup A_{c'}|$. Kot končno oceno ansambla (in s tem tudi kodne matrike, iz katere je bil pridobljen) pa smo vzeli povprečje Jaccardove ocene po vseh testnih primerih.

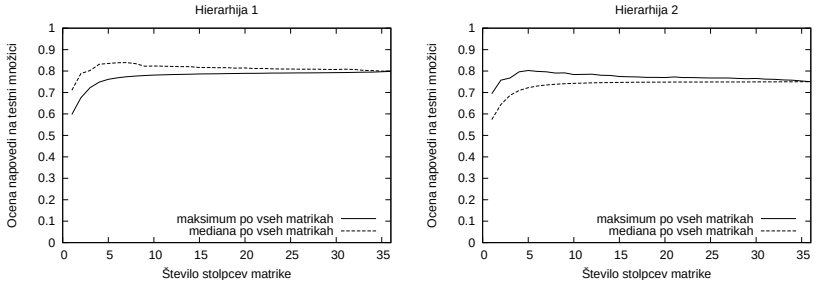
2.3.1 Distribucija ocen matrik

Kot smo videli v razdelku 2.2, ima posamezni stolpec kodne matrike pri naši hierarhiji le 36 možnih vrednosti, zato obstaja $\binom{36}{m}$ izborov m različnih stolpcev. Če pregledamo vseh $\binom{36}{m}$ matrik širine m in pri vsaki izračunamo povprečno Jaccardovo oceno njenih napovedi, se lahko vprašamo, kakšna je distribucija teh ocen. S preučevanjem te distribucije si pravzaprav odgovarjamo na vprašanje, kako dober model lahko pričakujemo, če bi matriko izbrali naključno (in prav to je eden od znanih pristopov h gradnji kodnih matrik).

Tabela 2.1 in graf na sliki 2.1 kažeta mediano in maksimum Jaccardove ocene po vseh matrikah z m stolpci (za m od 1 do 36). Mediana nam pove, da če naključno izberemo matriko z m stolpci, imamo 50 % možnosti, da bo njena ocena višja ali enaka od mediane; maksimum pa nam pove, kakšna je najboljša ocena, ki bi jo lahko dosegli, če bi pregledali vse matrike z m stolpci. V naših poskusih za $9 \leq m \leq 30$ nismo pregledali vseh matrik

Št. stolpcev (m)	Št. matrik z m stolpci	Hierarhija 1 Mediana ocene	Hierarhija 1 Maksimum ocene	Hierarhija 2 Mediana ocene	Hierarhija 2 Maksimum ocene
1	36	0,5972	0,7111	0,5803	0,6943
2	630	0,6753	0,7881	0,6433	0,7566
3	7 140	0,7221	0,8016	0,6861	0,7669
4	58 905	0,7475	0,8313	0,7089	0,7962
5	376 992	0,7607	0,8341	0,7221	0,8019
6	1 947 792	0,7685	0,8385	0,7297	0,7982
7	8 347 680	0,7735	0,8395	0,7346	0,7960
8	30 260 340	0,7765	0,8345	0,7379	0,7896
9	94 143 280	0,7784	0,8289	0,7403	0,7913
10	$2{,}54 \cdot 10^8$	0,7803	0,8267	0,7421	0,7834
15	$5{,}57 \cdot 10^9$	0,7859	0,8246	0,7465	0,7740
20	$7{,}31 \cdot 10^9$	0,7886	0,8173	0,7480	0,7691
25	$6{,}01 \cdot 10^8$	0,7904	0,8099	0,7485	0,7667
30	1,947,792	0,7923	0,8081	0,7489	0,7648
35	36	0,7967	0,7996	0,7499	0,7535
36	1	0,7976	0,7976	0,7496	0,7496

Tabela 2.1. Za vsako širino matrike je prikazano število matrik te širine ter mediana in maksimum Jaccardove ocene po matrikah te širine. (Pri $9 \leq m \leq 30$ so prikazani rezultati, dobljeni s pregledom 10^6 naključno izbranih matrik z m stolpci, pri ostalih m pa rezultati, dobljeni s pregledom vseh matrik z m stolpci.)



Slika 2.1. Pregledali smo kodne matrike določene širine (od 1 do 36 stolpcev) in pri vsaki matriki izračunali oceno pravilnosti napovedi ansambla klasifikatorjev, dobljenega iz te matrike. Na grafih sta prikazana mediana in maksimum teh ocen po matrikah posamezne širine (levi graf kaže rezultate za hierarhijo 1, desni pa za hierarhijo 2). (Pri $9 \leq m \leq 30$ so prikazani rezultati, dobljeni s pregledom 10^6 naključno izbranih matrik z m stolpci, pri ostalih m pa rezultati, dobljeni s pregledom vseh matrik z m stolpci.)

(ker jih je preveč), pač pa le po 10^6 naključno izbranih matrik za vsak m. Za primerjamo kaže tabela 2.2 tudi ocene matrik s tradicionalno regularno strukturo (eden proti enemu, eden proti ostalim).

Vidimo lahko, da se maksimum od $m = 4$ naprej povečuje zelo počasi, od $m = 8$ naprej pa začne celo rahlo upadati. Z drugimi besedami, naj-

Opis matrike	Št. stolpcev (m)	Jaccardova ocena Hierarhija 1	Hierarhija 2
1-proti-1	11	0.8143	0.7582
1-proti-ostalim	6	0.7959	0.7662
1-proti-1 (le za liste)	6	0.7737	0.7214
1-proti-ostalim (le za liste)	4	0.7812	0.7401

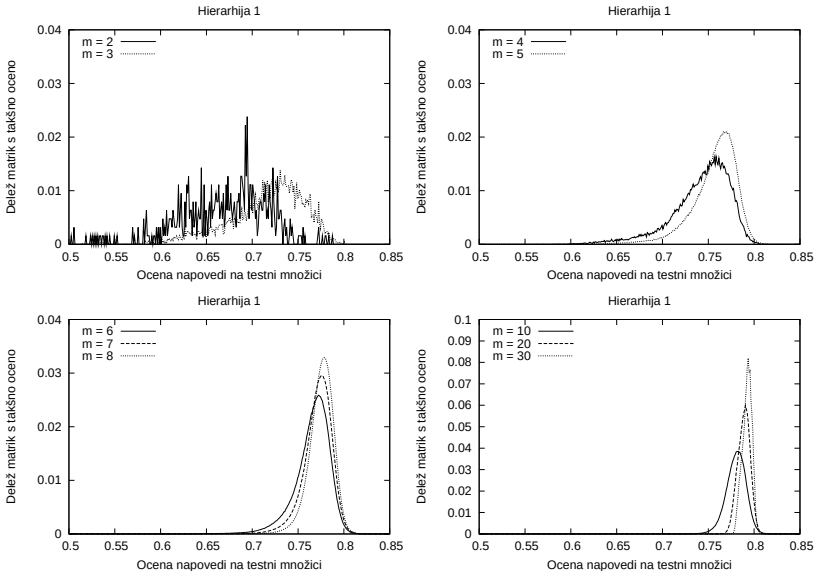
Tabela 2.2. Jaccardove ocene kodnih matrik z regularno strukturo. Matrike 1-proti-1 imajo po en stolpec za vsak par razredov (razen če je eden nadrejen drugemu), pri čemer je en razred pozitiven, drugi pa negativen; matrike 1-proti-ostalim pa imajo po en stolpec za vsak razred, pri čemer je ta razred pozitiven, ostali (če mu niso podrejeni) pa negativni.

boljša matrika s štirimi stolpci je že skoraj tako dobra kot najboljša matrika s sedmimi stolpci (ki je na teh podatkih tudi najboljša matrika sploh); če povečujemo število stolpcev še naprej, pa se ocena najboljše najdene matrike celo poslabša.[1] Po drugi strani pa se mediana tudi od $m = 4$ naprej ves čas še povečuje, čeprav vse počasneje. Ti rezultati kažejo, da obstajajo dobre matrike (torej take z visoko Jaccardovo oceno) tudi pri majhnem številu stolpcev, vendar jih je malo, pri večjem številu stolpcev pa je dobrih matrik več in jih je lažje najti. Če dovolimo res veliko število stolpcev, je dobre matrike vse lažje najti (mediana distribucije ocen matrik je tem bližje maksimumu, čim širše matrike opazujemo), vendar tako dobljene matrike niso tako dobre kot najboljše matrike z bolj zmernim številom stolpcev (najboljši rezultat je bil dosežen pri $m = 7$ stolpcih).

Distribucijo ocen matrik lahko prikažemo tudi s pomočjo histograma. Jaccardova mera nam vedno dá oceno z intervala $[0, 1]$; ta interval smo v naslednjem poskusu razdelili na 1000 enako širokih podintervalov in za vsak podinterval prešteli, pri kolikšnem deležu matrik pade ocena na ta podinterval. Dobljene histograme (ločene po širini matrik, od $m = 2$ do $m = 8$) kažejo grafi na sliki 2.2. Tudi na teh grafih se vidi, da maksimum ostaja pri večjih m približno enak, medtem ko se vse večji delež distribucije pomika k višjim ocenam.

Distribucije, ki jih vidimo na sliki 2.2, lahko poskusimo modelirati z beta

[1] Deloma je lahko to poslabšanje tudi posledica dejstva, da pri večjih m nismo pregledali vseh $\binom{36}{m}$ možnih matrik z m stolpci, pač pa le 10^6 naključno izbranih matrik; torej je delež pregledanih matrik v primerjavi s številom vseh matrik tem manjši, čim večji m gledamo (vse do $m = 18$). Toda na ta pomislek lahko navedemo vsaj dva protiargumenta. (1) Če pogledamo maksimum ocene pri pregledu 10^6 naključnih matrik z m stolpci in nato še maksimum pri pregledu 10^6 naključnih matrik s $36 - m$ stolpci (obojih matrik je enako veliko, torej smo v obeh primerih pregledali enak delež vseh možnih matrik tiste širine), se pri vsakem $m \geq 3$ izkaže, da so matrike z manj stolpci dale višji maksimum. (2) Če pogledamo m-je od 18 naprej in pri vsakem poiščemo maksimum ocene po 10^6 naključnih matrikah, vidimo, da se ta maksimum zmanjšuje, ko m narašča, čeprav je po drugi strani delež pregledanih matrik vse večji (ker vsakič pregledamo 10^6 matrik, vseh matrik širine m pa je vse manj).



Slika 2.2. Interval $[0, 1]$ smo razdelili na 1000 enako širokih podintervalov in za vsak podinterval izračunali, kolikšen delež vseh kodnih matrik določene širine doseže oceno s tega podintervala. Grafi kažejo dobljeno porazdelitev za m (število stolpcev) od 2 do 30 pri poskusih na hierarhiji 1.

distribucijo $B(\alpha, \beta)$. Pri slednji ima funkcija gostote verjetnosti obliko

$$F_X(x; \alpha, \beta) = x^{\alpha-1}(1-x)^{\beta-1}/B(\alpha, \beta),$$

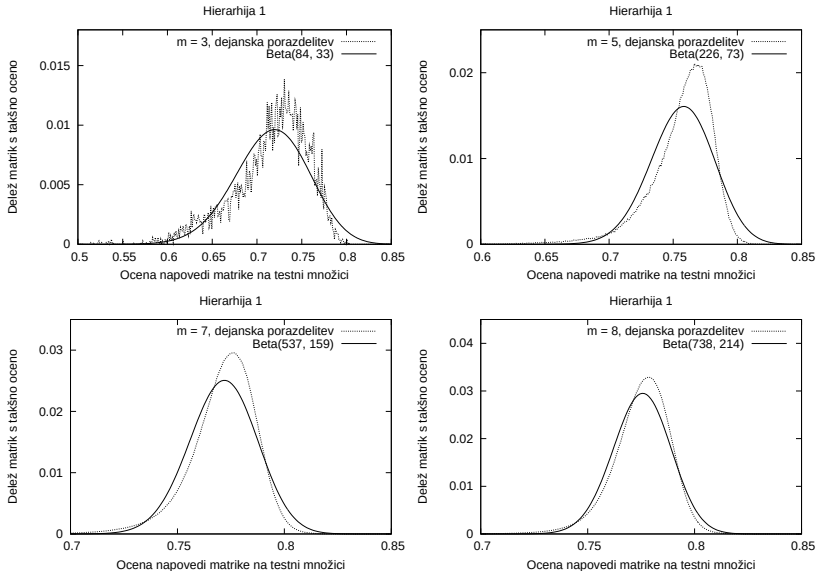
pri čemer je imenovalec funkcija beta: $B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt = \Gamma(\alpha)\Gamma(\beta)/\Gamma(\alpha+\beta)$. Tako porazdeljena slučajna spremenljivka ima upanje $E[X] = \alpha/(\alpha+\beta)$ in varianco $D[X] = \alpha\beta/[(\alpha+\beta)^2(\alpha+\beta+1)]$. Po metodi momentov sledita iz tega naslednji cenilki za parametra α in β:

$$\hat{\alpha} = \bar{x}(\bar{x}(1-\bar{x})/s^2 - 1) \text{ in } \hat{\beta} = (1-\bar{x})(\bar{x}(1-\bar{x})/s^2 - 1),$$

pri čemer je $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$ vzorčno povprečje, $s^2 = \frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2$ pa vzorčna disperzija.

V našem primeru bomo za vrednosti $x_1, \ldots, x_n$ vzeli Jaccardove ocene posameznih kodnih matrik (za neko fiksno število stolpcev m); pri vsakem m tako dobimo po en nabor parametrov (α_m, β_m). Primerjavo med tako dobljeno beta porazdelitvijo in med dejansko distribucijo Jaccardovih ocen vidimo na sliki 2.3 (za $m = 3, 5, 7, 8$). Vidimo lahko, da je odstopanje predvsem v tem, da je prava distribucija nagnjena malo bolj na desno in ima višji modus od beta distribucije, s katero smo jo skušali modelirati. Izkaže se, da ko m narašča, postaja beta distribucija vse boljši približek prave; α_m in β_m pa naraščata približno kot eksponentni funkciji m-ja.

Distribucija ocen matrik v odvisnosti od velikosti testne množice. Na tem mestu se lahko pojavi pomislek, da distribucija ocen matrik, kot smo jo

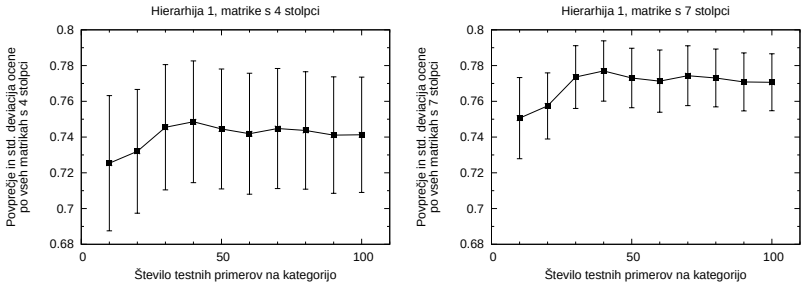


Slika 2.3. Porazdelitev Jaccardovih ocen vseh matrik z m stolpci ima približno obliko beta porazdelitve. Parametra te porazdelitve lahko ocenimo po metodi momentov; vsak graf kaže tako dobljeno beta porazdelitev skupaj s prvotno porazdelitvijo ocen matrik. Zgornji levi graf je za $m = 3$, zgornji desni za $m = 5$, spodnji levi za $m = 7$ in spodnji desni za $m = 8$.

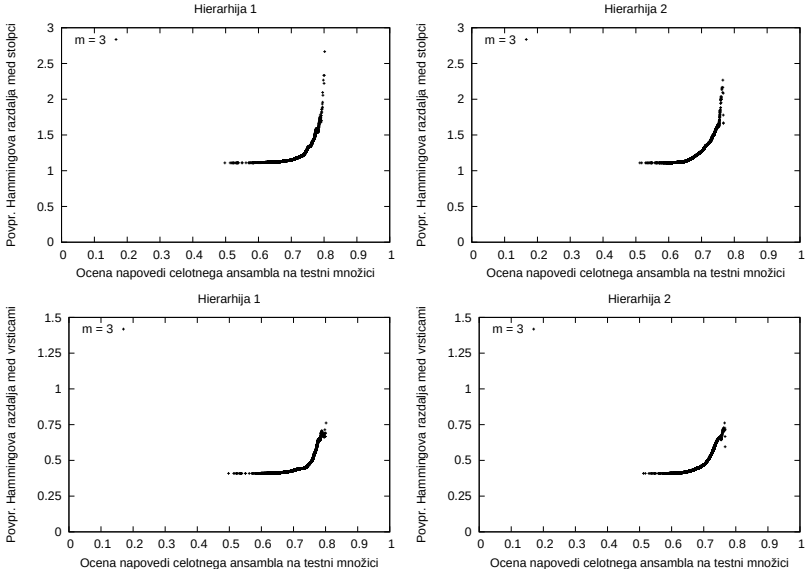
doslej videli v tem razdelku, ni nujno posledica tega, da so nekatere matrike boljše od drugih, ampak bi lahko bila posledica golega naključja: lahko bi imeli več enako dobrih klasifikatorjev, vendar bi na neki konkretni testni množici nekateri od njih pač dajali boljše rezultate kot drugi.

Recimo, da imamo klasifikator z verjetnostjo napake p in ga poženemo na t testnih primerih; število napak pri takem testiranju ima potemtakem upanje pt in standardno deviacijo $\sqrt{p(1-p)t}$. Če pogledamo veliko takšnih (enako dobrih) klasifikatorjev, bi torej pričakovali, da bodo njihove ocene porazdeljene po beta porazdelitvi s tem upanjem in standardno deviacijo. Toda to bi pomenilo, da je standardna deviacija ocen naših klasifikatorjev tem večja, čim večji je t (velikost testne množice).

Zato smo nekaj poskusov iz tega razdelka ponovili še tako, da smo velikost testne množice spreminjali (od 10 do 100 testnih primerov iz vsake kategorije). Kot vidimo iz grafov na sliki 2.4, je standardna deviacija ocen matrik ves čas približno enaka, ne glede na to, kako se povečuje testna množica. Ta rezultat govori v prid domnevi, da je naša distribucija ocen matrik posledica tega, da so nekatere res boljše od drugih in da ni le plod naključja oz. tega, da bi med več enako dobrimi matrikami nekatere slučajno dosegle boljše rezultate na naši konkretni (in relativno majhni) testni množici.



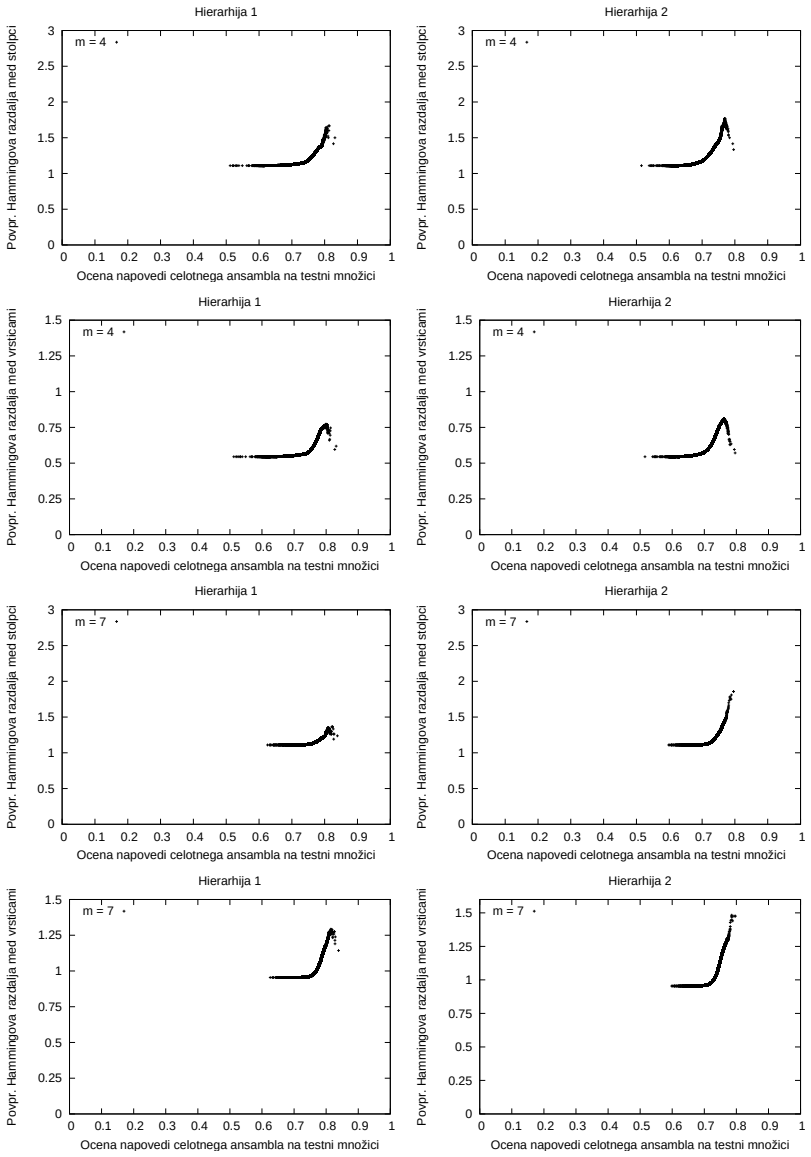
Slika 2.4. Grafa kažeta povprečje in standardno deviacijo Jaccardovih ocen matrik v odvisnosti od velikosti testne množice. Levi graf kaže rezultate za vse matrike s 4 stolpci, desni graf pa za naključno izbranih 10^6 matrik s 7 stolpci. Vidimo lahko, da se standardna deviacija ne spreminja v odvisnosti od velikosti testne množice.



Slika 2.5. Grafi vsebujejo po eno točko za vsako kodno matriko s 3 stolpci. Za x-koordinato je pri vseh vseh grafih uporabljena ocena napovedi ansambla, dobljenega iz matrike. Pri zgornjih dveh grafih je y-koordinata povprečna Hammingova razdalja med stolpci matrike, pri spodnjih pa povprečna Hammingova razdalja med vrsticami matrike. Levi par grafov se nanaša na poskuse s hierarhijo 1, desni pa na poskuse s hierarhijo 2.

2.3.2 Povezava med lastnostmi matrike in njeno oceno

Zanimivo vprašanje pri kodnih matrikah je, ali je uspešnost matrike pri klasifikaciji (kot jo merimo npr. z Jaccardovo oceno) v kakšni zvezi s kakšnimi preprosteje merljivimi lastnostmi matrike; to bi lahko koristilo pri usmerjanju algoritmov za iskanje oz. gradnjo kodnih matrik. Še posebej zanimive so lastnosti, ki jih lahko izračunamo iz matrike same, brez učenja ali testiranja



Slika 2.6. Grafi vsebujejo po eno točko za vsako kodno matriko s 4 ali 7 stolpci. Za x-koordinato je pri vseh vseh grafih uporabljena ocena napovedi ansambla, dobljenega iz matrike. Pri grafih v prvi in tretji vrstici je y-koordinata povprečna Hammingova razdalja med stolpci matrike, pri ostalih pa povprečna Hammingova razdalja med vrsticami matrike. V posamezni vrstici se levi graf se nanaša na poskuse s hierarhijo 1, desni pa na poskuse s hierarhijo 2. Pri matrikah s 7 stolpci so prikazani rezultati za naključno izbranih 10^5 matrik, ker je vseh matrik s 7 stolpci preveč, da bi lahko na grafu prikazali vse.

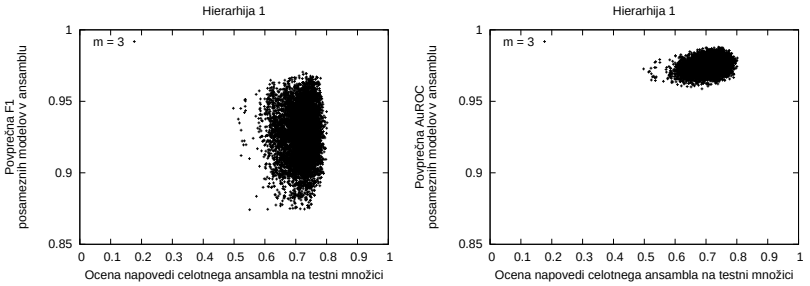
klasifikatorjev, ki izhajajo iz nje.

Primer takšne lastnosti je povprečna Hammingova razdalja med vrsticami oz. med stolpci matrike. Za oboje se da sestaviti neformalne argumente, zakaj sta to lastnosti, ki si ju od kodnih matrik želimo. Vsaka vrstica kodne matrike pove, kako so nek razred pri učenju uporabljali posamezni dvorazredni klasifikatorji; če torej vzamemo nek primer iz tega razreda in izračunamo zanj napovedi teh klasifikatorjev, bomo dobili vektor napovedi, ki bo enak tej vrstici v matriki, razen tam, kjer se kak klasifikator zmoti. Vektor napovedi primerjamo z vrsticami matrike in kot končno napoved ansambla uporabimo tisti razred, čigar vrstica se najbolj ujema z vektorjem napovedi. Če se je nekaj dvorazrednih klasifikatorjev pri napovedi zmotilo, se naš vektor napovedi na tistih mestih razlikuje od prave vrstice matrike. Bolj ko so si vrstice različne (to pa lahko merimo npr. s povprečno Hammingovo razdaljo med njimi), bolj je verjetno, da bo kljub temu naš vektor napovedi še vedno bolj podoben pravi vrstici kot pa kakšni drugi.

Če si je več stolpcev matrike preveč podobnih, rešujejo njihovi dvorazredni klasifikatorji precej podobne probleme in dajejo od sebe precej podobne napovedi; če se bo zmotil eden od njih, se bodo najbrž zmotili tudi drugi. Hitreje se bo torej zgodilo, da se bo vektor napovedi močno razlikoval od prave vrstice v matriki, to pa poveča tveganje, da bo na koncu bolj podoben neki drugi vrstici kot pa tisti pravi. Zato si načeloma želimo, da je tudi Hammingova razdalja med stolpci čim večja.

Empirični preizkus teh razmislekov kažejo grafi na slikah 2.5 in 2.6. Na grafih je za vsako matriko po ena točka, pri čemer je kot x-koordinata uporabljena Jaccardova ocena matrike (ki nam torej pove, kako dobro se matrika obnese z vidika klasifikacije), kot y-koordinata pa povprečna Hammingova razdalja med vrsticami oz. stolpci matrike. Prikazani so ločeni grafi za matrike širine 3, 4 in 7 stolpcev. Vidimo lahko, da v splošnem res velja, da imajo boljše matrike (torej take z višjo Jaccardovo oceno) tudi večjo povprečno Hammingovo razdaljo med vrsticami oz. stolpci. Vendarle pa rezultati za $m = 4$ in $m = 7$ kažejo, da zgolj maksimizacija obeh razdalj še ne zagotavlja, da bomo dobili najboljšo matriko, saj ima najboljših nekaj matrik obe razdalji manjši od maksimalne.

Zanimivo vprašanje je tudi, v kakšni zvezi je kakovost celotnega ansambla (ki izvira iz neke matrike) s kakovostjo posameznih dvorazrednih klasifikatorjev v njem. Tudi to si lahko ogledamo na grafu. Za vsak dvorazredni klasifikator smo izračunali oceno njegovih napovedi (na primer F_1 ali površino pod krivuljo ROC) s stališča dvorazrednega problema, za katerega je bil sploh naučen. Za vsako kodno matriko lahko izračunamo povprečje takih ocen po vseh klasifikatorjih v ansamblu, dobljenem iz tiste matrike. Rezultate tega poskusa kažeta grafa na sliki 2.7; kot vidimo, takšne povprečne ocene posameznih dvorazrednih modelov v ansamblu niso zares korelirane z oceno ansambla kot celote (korelacijski koeficient je približno 0,015, če dvorazredne modele ocenimo z F_1-mero, in približno 0,25, če jih ocenimo s



Slika 2.7. Grafa vsebujeta po eno točko za vsako kodno matriko s 3 stolpci. Za x-koordinato je pri vseh obeh grafih uporabljena ocena napovedi ansambla, dobljenega iz matrike. Pri levem grafu je y-koordinata povprečna vrednost F_1-mere po vseh dvorazrednih klasifikatorjih v ansamblu, pri desnem grafu pa povprečna vrednost površine pod krivuljo ROC po vseh dvorazrednih klasifikatorjih v ansamblu. Ocene posameznih dvorazrednih klasifikatorjev so bile dobljene s prečnim preverjanjem na učni množici.

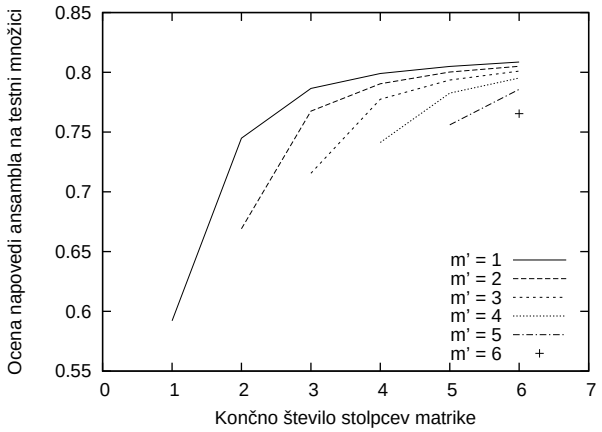
površino pod krivuljo ROC). Tudi poskusi z drugimi merami za ocenjevanje dvorazrednih klasifikatorjev (natančnost, priklic, klasifikacijska točnost, BEP) niso pripeljali do nobenih boljših korelacij.

2.3.3 Požrešni sprehodi skozi prostor matrik

Problem gradnje čim boljše kodne matrike si lahko predstavljamo kot problem preiskovanja prostora vseh možnih kodnih matrik. Ena od možnih tehnik priskovanja prostora je tudi lokalna optimizacija oz. požrešni algoritem:

```
1    naj bo M matrika z m' naključnimi stolpci;
2    while ima matrika M manj kot m stolpcev:
3        za vsak stolpec x, ki ga še ni v matriki M:
4            ocenimo matriko [M|x];
5        dodajmo v M tisti x, pri katerem je bila dosežena
             v vrstici 4 najvišja ocena;
```

Matrika, ki jo dobimo na koncu tega postopka, je odvisna od tega, s kakšno matriko smo začeli v vrstici 1. Pri naših sistematičnih poskusih na majhnem problemu si lahko privoščimo, da ta postopek izvedemo za vse možne izbore začetne matrike z m' stolpci (v vrstici 1) in izračunamo povprečno oceno končne matrike po vseh možnih izborih začetne matrike. Privoščimo si lahko tudi, da v vrstici 3 res pregledamo vse možne stolpce, ki jih še ni v matriki, medtem ko pri večjem problemu tega ne bi mogli, ker je možnih stolpcev preveč. Na ta način lahko dobimo nek (sicer optimističen) občutek za to, kako se obnese požrešni algoritem kot način za preiskovanje prostora kodnih matrik.



Slika 2.8. Recimo, da začnemo z m' naključno izbranimi stolpci, nato pa tako dobljeno matriko dopolnimo do m stolpcev s požrešnim postopkom (na vsakem koraku dodamo v matriko tak stolpec, ki najbolj poveča uspeh našega ansambla klasifikatorjev). Graf na sliki kaže oceno napovedi tako dobljenih ansamblov kot funkcijo m. Za vsak m' je na grafu po ena linija (oz. točka, če je $m' = m$), prikazani rezultati pa so povprečja po vseh možnih začetnih matrikah z m' stolpci.

Rezultate tega poskusa kaže graf na sliki 2.8. Vidimo lahko, da so (pri fiksnem m) matrike, do katerih pride požrešno preiskovanje prostora, v povprečju tem boljše, čim večji del matrike je bil dobljen s požrešnim iskanjem (torej: čim manjši je m'). Ta poskus torej kaže, da lokalna optimizacija s požrešnim algoritmom načeloma pripelje do boljših matrik kot popolnoma naključno generiranje celotne matrike (slednje bi ustrezalo primeru, ko je $m' = m$). Zato se bomo tudi v naslednjem poglavju ukvarjali s požrešno gradnjo kodnih matrik.

2.4 Zaključki

Prostor vseh možnih kodnih matrik je zelo velik. Če je hierarhija razredov v našem večrazrednem klasifikacijskem problemu dovolj pravilno zgrajena, lahko število možnih različnih stolpcev matrike izračunamo analitično, prek tega pa izračunamo tudi število vseh matrik določene širine. Če je število razredov v našem prvotnem večrazrednem problemu dovolj majhno, je prostor kodnih matrik še obvladljiv in lahko sistematično pregledamo in ocenimo vse matrike do neke želene širine. (Pri tem „ocena matrike“ pomeni v resnici oceno pravilnosti napovedi ansambla klasifikatorjev, ki izhajajo iz te matrike.)

Ti poskusi so pokazali, da obstajajo dobre matrike (torej take, katerih ansambel doseže visoko oceno) tudi pri majhnem številu stolpcev (torej z majhnim številom dvorazrednih klasifikatorjev v ansamblu), le da je

delež takšnih matrik manjši kot pri matrikah z več stolpci. Videli smo, da ima porazdelitev ocen vseh možnih matrik določene širine približno obliko beta-porazdelitve, pri čemer vrednosti parametrov α in β te porazdelitve naraščata približno kot eksponentni funkciji v odvisnosti od števila stolpcev matrike.

Klasifikacijska ocena matrike kot celote (oz. ansambla, dobljenega iz nje) ni korelirana s povprečno oceno posameznih dvorazrednih klasifikatorjev v njenem ansamblu. Obstaja pa povezava med povprečno Hammingovo razdaljo med vrsticami oz. stolpci matrike in klasifikacijsko oceno matrike; tiste matrike, pri katerih so si stolpci oz. vrstice v povprečju bolj različni (torej imajo večjo povprečno Hammingovo razdaljo med njimi), dosegajo višje ocene. Vendar pa ta povezava ni povsem monotona; ansambli, ki dosežejo najvišjo oceno pri klasifikaciji, imajo sicer veliko povprečno Hammingovo razdaljo med vrsticami oz. stolpci, vendar ne prav najvišje, in obratno.

Prostor matrik je primeren tudi za preiskovanje s požrešnim algoritmom, pri katerem se na vsakem koraku doda v matriko nov stolpec tako, da se ocena celotnega ansambla pri tem čim bolj poveča. Pri poskusih z majhnimi matrikami se je pokazalo, da so matrike, dobljene s požrešnim postopkom, boljše od naključno zgrajenih matrik in to tem bolj, čim večji delež matrike je bil zgrajen s požrešnim postopkom.

Poglavje 3

Postopna gradnja kodnih matrik

Obstoječe tehnike za gradnjo kodnih matrik imajo nekaj slabosti, zlasti če jih hočemo uporabiti na problemih z veliko razredi in v primerih, ko so razredi organizirani v hierarhijo. Ena od slabosti je, da večina teh metod za obravnavo k-razrednega problema potrebuje vsaj $O(k)$ modelov, kar je neugodno, če je število razredov veliko in če je tudi učenje posameznega modela relativno drago (na primer z metodo podpornih vektorjev). Lepo bi bilo torej imeti metodo, ki se osredotoča na gradnjo matrike z manjšim (sublinearnim) številom stolpcev. Še ena slabost obstoječih metod je, da ne upoštevajo hierarhičnih odnosov med razredi. Na te slabosti skuša odgovoriti naša metoda, ki jo predstavljamo v tem poglavju.

3.1 Omejitve, ki izvirajo iz hierarhije med razredi

Struktura kodne matrike mora upoštevati hierarhične odnose med k razredi prvotnega večrazrednega klasifikacijskega problema. Konkretno to pomeni, da če je razred c v hierarhiji prednik razreda c' in če nek model j vidi enega od teh dveh razredov kot pozitivnega, drugega pa kot negativnega (npr. $M_{cj} = 1$, $M_{c'j} = -1$), bi to pomenilo, da so z vidika modela j primeri razreda c' hkrati pozitivni in negativni (saj je c' podrazred razreda c in je zato vsak primer razreda c' hkrati tudi primer razreda c). Iz tega sledi omejitev, da če je c prednik razreda c', mora pri vsakem j veljati $M_{cj} = 0$ ali pa $M_{cj}M_{c'j} > 0$.

Te omejitve ni pretežko vgraditi v algoritem za gradnjo naključnih matrik. Tako dobimo:

ALGORITEM A (gradnja j-tega stolpca matrike):
1 naj bo $M_{cj} := 0$ za $c = 1, \ldots, k$;
2 naj bosta $Anc_1 := \{\}$ in $Anc_{-1} := \{\}$;

3 **while** ni v j-tem stolpcu dovolj neničelnih elementov in
še nismo pregledali vseh možnih parov (c, c'):
4 naključno izberi razreda c in c' tako, da nobeden od njiju
ni prednik drugega v hierarhiji in da para (c, c') še nismo
obravnavali v kakšni prejšnji iteraciji te zanke;
5 **if** $c \in Anc_1 \cap Anc_{-1}$ **then** $A_c := \{\}$;
else if $c \in Anc_1$ **then** $A_c := \{1\}$;
else if $c \in Anc_{-1}$ **then** $A_c := \{-1\}$;
else $A_c = \{-1, 1\}$;
6 enako kot v koraku 5 iniciraliziraj zdaj še $A_{c'}$ s pomočjo $M_{c'j}$;
7 $A := \{(a, a') : a \in A_c, a' \in A_{c'}, a \cdot a' < 0\}$;
8 če je A prazna množica, se vrni na korak 3;
9 naj bo (a, a') poljuben element množice A;
za vse c'', ki so potomci c (tudi $c'' = c$) postavi $M_{c''j} := a$;
za vse c'', ki so potomci c' (tudi $c'' = c'$) postavi $M_{c''j} := a'$;
10 dodaj c in vse njegove prednike v Anc_a,
dodaj c' in vse njegove prednike v $Anc_{a'}$;
11 **end while**;

V množicah Anc_1 in Anc_{-1} torej hranimo vse razrede, ki so v trenutnem stolpcu že dobili neničelno oznako, in vse njihove prednike. Tidve množici prideta prav pri preverjanju, kakšne omejitve nam doslej postavljene neničelne vrednosti v stolpcu nalagajo glede vrednosti ostalih elementov stolpca. Če na primer za nek razred c velja $c \in Anc_1$, to pomeni, da je bil bodisi c bodisi nek njegov potomec v trenutnem stolpcu že uporabljen kot pozitiven, tako da v nadaljevanju gradnje trenutnega stolpca razreda c vsekakor ne smemo uporabiti kot negativnega. Analogen razmislek velja, če je $c \in Anc_{-1}$. Na podlagi teh omejitev korak 5 gornjega algoritma sestavi množico A_c možnih vrednosti, ki jih še lahko dobi element M_{cj} trenutnega stolpca. Podobno lahko nato sestavimo še množico $A_{c'}$ z možnimi vrednostmi za element $M_{c'j}$ (korak 6).

Teoretično bi se lahko zgodilo, da bi bila časovna zahtevnost tega algoritma $O(k^2)$, torej sorazmerna s kvadratom števila razredov, vendar bi do tega lahko prišlo le v patoloških primerih. Pri realističnih, običajno razvejenih hierarhijah razredov bi bilo število iteracij glavne zanke (koraki 3–11) linearno v odvisnosti od števila želenih neničelnih elementov v novem stolpcu matrike, kar je torej v najslabšem primeru $O(k)$, v praksi pa še manj, saj želimo, da bi bila matrika redka.

Opisani pristop se opira na predpostavko, da hierarhični odnosi med razredi (koncepti ontologije) tvorijo drevo, torej da v hierarhiji ne prihaja do ciklov in da razred nima več kot enega neposrednega nadrazreda. V primerih, ko ima lahko posamezni razred v hierarhiji več neposrednih nadrazredov, bi doslej opisani postopek sicer deloval, vendar bi lahko zaradi tega trpela njegova učinkovitost in skalabilnost.

3.2 Požrešna gradnja kodne matrike

V tem razdelku opisujemo požrešno metodo, ki dodaja stolpce v kodno matriko enega po enega. Uveljavljeni pristopi h gradnji kodnih matrik definirajo bodisi celo matriko naenkrat (npr. eden proti enemu, eden proti ostalim, kode za odpravljanje napak) ali pa sicer gradijo matriko po stolpcih, vendar tako, da je vsak stolpec popolnoma neodvisen od drugih (npr. popolnoma naključne kodne matrike, kjer je posameznim stolpcem skupno kvečjemu to, da se vsi skupaj podrejajo neki morebitni globalni omejitvi glede gostote matrike).

Za razliko od teh pristopov izhaja naša metoda iz domneve, da utegne biti koristno, če matriko gradimo postopoma (stolpec za stolpcem, ne cele matrike naenkrat) in pri tem vsakič upoštevamo obnašanje tistega dela matrike, ki je bil doslej že zgrajen. Naš cilj pri tem je obdržati ali izboljšati uspeh matrike kot celote pri klasifikaciji (torej uspeh ansambla dvorazrednih klasifikatorjev, kakršnega definirajo stolpci matrike) in se pri tem izogibati potrebam po neobvladljivo velikem številu modelov. Požrešnost metode se kaže v tem, da skušamo vsak naslednji stolpec matrike sestaviti tako, da se bo uspeh ansambla kot celote čim bolj povečal, ko bomo vanj dodali nov dvorazredni klasifikator, naučen na podlagi novega stolpca matrike.

Požrešne gradnje kodne matrike se lahko lotimo takole:

ALGORITEM B (požrešna gradnja kodne matrike):
1 prvih nekaj stolpcev matrike inicializiraj naključno,
kot to opisuje algoritem A iz razdelka 3.1;
2 za vsak nadaljnji stolpec j:
3 naj bo $M_{cj} := 0$ za vse $c = 1, \ldots, k$;
4 naj bosta $Anc_1 := \{\}$ in $Anc_{-1} := \{\}$;
5 ovrednoti klasfikacijski uspeh dosedanjega ansambla
dvorazrednih klasifikatorjev (ki izhajajo iz stolpcev
$1, 2, \ldots, j-1$ naše kodne matrike) na učni ali
še raje na validacijski množici;
6 naj bo $E = (E_{cc'})$ matrika zamenjav, v kateri torej
element $E_{cc'}$ pove, koliko je primerov, ki pripadajo
razredu c in jih je ansambel klasificiral v razred c';
7 **while** ni v j-tem stolpcu dovolj neničelnih elementov in
še nismo pregledali vseh možnih parov (c, c'):
8 izberi naslednji par (c, c'), gledano po padajočem
vrstnem redu vrednosti $E_{cc'} + E_{c'c}$;
9 za ta par (c, c') izvedi korake 5–10 algoritma A;
10 **end while**;
11 **end for**;

Ta algoritem uporablja matriko zamenjav, da identificira posebej zahtevna območja odločitvenega problema in sestavi naslednji stolpec matrike

tako, da se bo naslednji dvorazredni klasifikator osredotočil na tista zahtevna območja in s tem mogoče popravil napake, ki so jih naredili dosedanji klasifikatorji. Ideja tega postopka je podobna tisti, ki se uporablja pri boostingu, le da slednji deluje na nivoju posameznih primerov in za učenje vsakega novega klasifikatorja spremeni učno množico, število razredov pa ohrani nespremenjeno in ne poskuša na primer definirati novih razredov s kombiniranjem ali spreminjanjem razredov prvotnega problema. Za razliko od tega pa si lahko naš postopek predstavljamo kot neke vrste boosting na nivoju razredov.

Če je imel prvotni klasifikacijski problem k razredov, je matrika zamenjav načeloma velikosti $k \times k$, kar je lahko pri večjih k že neugodno veliko. Vendar pa lahko vsak primer iz učne (ali validacijske) množice prispeva le omejeno število zamenjav, zato je tudi skupno število neničelnih elementov v matriki zamenjav omejeno. To torej pomeni, da je lahko matrika zamenjav (pri velikem k) precej redka in njena pomnilniška zahtevnost postane precej bolj obvladljiva, če eksplicitno hranimo le njene neničelne elemente. Pri izračunu matrike zamenjav moramo upoštevati tudi hierarhične odnose med razredi: če nek primer pripada razredu c, potem pripada tudi vsem prednikom tega razreda; podobno, če ga je naš ansambel klasificiral v razred c, se šteje to tudi kot napoved njegove pripadnosti vsem prednikom tega razreda. Za gradnjo matrike zamenjav uporabljamo naslednji postopek:

```
1   postavi E_{cc'} := 0 za vse c od 1 do k in vse c' od 1 do k;
2   za vsak primer x iz validacijske množice:
3       naj bo c pravi razred primera x;
        naj bo c' razred, v katerega je x klasificiral dosedanji ansambel;
4       naj bo A_c množica vseh prednikov razreda c (vključno s c samim);
        naj bo A_{c'} množica vseh prednikov razreda c' (vključno s c' samim);
5       za vsak c_1 ∈ A_c − A_{c'} in za vsak c_2 ∈ A_{c'} − A_c:
6           povečaj E_{c_1 c_2} za 1;
```

Ko dodamo v matriko nov stolpec in s tem v ansambel dodamo nov dvorazredni klasifikator, se lahko zgodi, da novi klasifikator ne bo dovolj spremenil napovedi ansambla kot celote (ker ga bodo ostali klasifikatorji preglasovali). Zato se matrika zamenjav ne bo dovolj spremenila in pri gradnji naslednjih stolpcev bo vrstni red parov (c, c'), ko jih postopek B gleda po padajočem številu zamenjav, ostal zelo podoben kot prej. To pa pomeni, da utegne nastati več podobnih stolpcev v matriki, iz njih dobljeni klasifikatorji pa si bodo torej precej podobni in tudi precej korelirani v svojih napovedih (npr. ko se bodo motili, se bodo motili vsi skupaj), kar je slabo za uspeh ansambla kot celote. Tej težavi se lahko poskusimo izogniti po zgledu iskanja s tabuji, ki je znano v postopkih lokalne optimizacije. Pri lokalni optimizaciji se lahko zgodi, da se pri preiskovanju prostora prevečkrat vrnemo v stanje, kjer smo v nedavni preteklosti že bili, ali pa se celo čisto zaciklamo. Iskanje s tabuji je tehnika, ki skuša ta problem omiliti tako, da

nedavno obiskana stanja razglasi za tabu in jih hrani v posebnem seznamu (seznam tabu); pri preiskovanju prostora se moramo nato držati dodatne omejitve, da se ne premaknemo v stanje, ki se trenutno nahaja na seznamu tabu. Po določenem številu korakov lahko stare elemente iz seznama tabu tudi pobrišemo. V našem primeru lahko seznam tabu uporabljamo na naslednji način: ko v postopku B za nek par (c, c') spremenimo eno ali obe izmed vrednosti M_{cj} in $M_{c'j}$ z 0 na neko neničelno vrednost, dodajmo par (c, c') na seznam tabu. Zanka v korakih 7–9 algoritma B naj pare, ki se trenutno nahajajo na seznamu tabu, preskoči. Par ostane na seznamu tabu omejeno dolgo časa, dokler matrika ne pridobi določenega števila novih stolpcev.

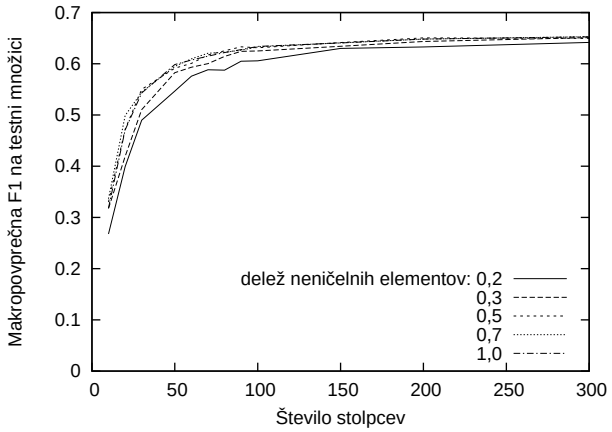
3.3 Poskusi s požrešnim algoritmom in seznamom tabu

Za poskuse v tem poglavju smo uporabili srednje veliko ontologijo, dobljeno iz ontologije *dmoz* oz. ODP (Open Directory Project). Iz slednje smo izbrali kategorijo *Top/Science*, sedem njenih neposrednih podkategorij in 72 podpodkategorij. Iz vsake od tako dobljenih 80 kategorij smo izbrali 100 dokumentov za v učno množico in 100 za v testno množico.

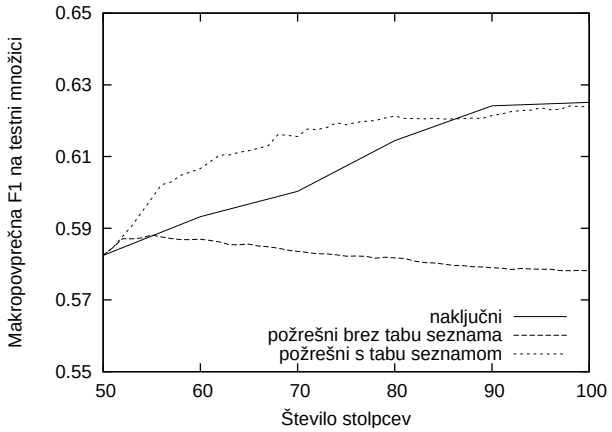
Za začetek si oglejmo obnašanje ansamblov, dobljenih na podlagi naključnih kodnih matrik, ki smo jih sestavili z algoritmom A iz razdelka 3.1. Graf na sliki 3.1 kaže uspešnost ansambla v odvisnosti od števila stolpcev v kodni matriki (in s tem števila dvorazrednih klasifikatorjev v ansamblu) in v odvisnosti od gostote naključne matrike (deleža neničelnih elementov glede na vse elemente matrike). Posamezne črte na grafu ustrezajo različnim gostotam matrike, neodvisna spremenljivka na abscisni osi pa je število stolpcev matrike. Vse prikazane ocene uspešnosti ansamblov so povprečja treh ponovitev po desetih naključnih kodnih matrikah.

Kot vidimo iz rezultatov tega poskusa, se je pretirano redkim matrikam bolje izogniti (spodnji dve liniji na grafu), še posebej, če je število modelov veliko, saj je takrat dekodiranje napovedi (iz vektorja napovedi dvorazrednih modelov v končno napoved za prvotni večrazredni problem) problematično in uspešnost ansambla kot celote se poslabša. Za ocenjevanje napovedi smo uporabili makropovprečje mere F_1, ker je ta mera standardna na področju poizvedovanja po tekstovnih podatkih (*information retrieval*); so pa vsekakor še druge možnosti, na primer mera za primerjanje ontologij, ki jo opisujemo v poglavju 5, ali metodologija iz [49].

Graf na sliki 3.2 kaže prednosti požrešnega algoritma za gradnjo matrik (algoritem B iz razdelka 3.2) v primerjavi s popolnoma naključnimi matrikami. Začeli smo s 50 naključnimi stolpci (dobljenimi z algoritom A), nato pa smo jim dodali še do 50 novih stolpcev s požrešno metodo (algoritem B). Za primerjavo kaže graf 3.2 tudi uspešnost popolnoma naključne matrike (vsi stolpci dobljeni z algoritmom A). Vidimo lahko, da če prvotni ansambel 50



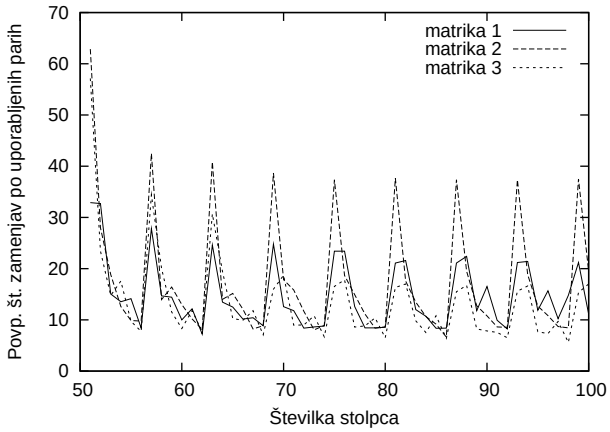
Slika 3.1. Graf kaže makropovprečno F_1 za ansamble, dobljene iz naključnih kodnih matrik, sestavljenih z algoritmom A iz razdelka 3.1, v odvisnosti od števila stolpcev in gostote matrike. Pri tem gostota matrike pomeni delež neničelnih komponent v njej in je pri teh poskusih variiral od 0,2 do 1. Prikazani rezultati so povprečja po treh ponovitvah algoritma.



Slika 3.2. Primerjava matrik, dobljenih v celoti naključno, s takimi, kjer začnemo s 50 naključnimi stolpci in jim nato dodamo še do 50 stolpcev s požrešno metodo (algoritem B iz razdelka 3.2).

modelov dopolnimo z le 20 dodatnimi modeli, dobljenimi z algoritmom B, dosežemo enak uspeh, kot če uporabimo 50 dodatnih naključnih stolpcev.

Vidimo pa lahko tudi, da se je uporaba seznama tabu v tem primeru izkazala za ključno, saj je brez nje matrika, dobljena s požrešno metodo, delovala celo slabše od popolnoma naključne. Dogaja se namreč — kot smo posumili že na koncu prejšnjega razdelka — da ko v ansambel dodamo nov klasifikator, se napovedi ansambla kot celote ne spremenijo dovolj, zato je nova matrika zamenjav zelo podobna prejšnji in požrešni algoritem sestavi

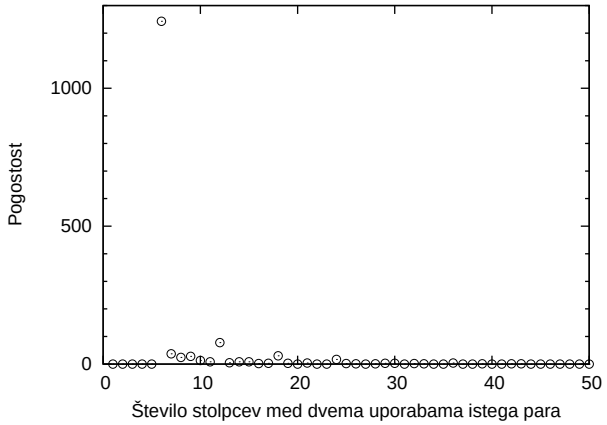


Slika 3.3. Analiza obnašanja algoritma B iz razdelka 3.2 z uporabo seznama tabu. Algoritem sestavi nov stolpec matrike tako, da vzame nekaj parov razredov s čim večjim številom zamenjav in od vsakega para uporabi en razred kot pozitivnega in drugi razred kot negativnega. Graf kaže skupno število zamenjav po vseh uporabljenih parih pri posameznem stolpcu, za tri ponovitve poskusa (začetni del matrike, sestavljen naključno, je bil vsakič drugačen). Periodičnost na grafu je posledica tega, da se pridejo isti pari razredov v obravnavo po večkrat znova, čim izstopijo iz seznama tabu.

še en stolpec, zelo podoben prejšnjemu. Zato so si stolpci matrike preveč podobni, kar pa je, kot vemo, slabo.

Izkaže pa se, da te težave tudi uporaba seznama tabu ne odpravi dovolj dobro. Videli smo, da gradnja posameznega stolpca matrike z algoritmom B poteka tako, da pregledujemo pare razredov po padajočem številu zamenjav in prvih nekaj parov uporabimo pri pripravi novega stolpca; pri tem pa preskočimo tiste pare (c, c'), ki so na seznamu tabu ali pa bi povzročili konflikte z že postavljenimi vrednostmi v trenutnem stolpcu (vrstica 8 algoritma A, ki jo uporabimo v sklopu vrstice 9 algoritma B). Recimo zdaj, da bi si pri vsakem uporabljenem paru (torej takem, ki ga nismo preskočili) zapomnili skupno število zamenjav, $E_{cc'} + E_{c'c}$, in na koncu izračunali zaporedje prek vseh uporabljenih parov pri tem stolpcu matrike. To povprečje je seveda od stolpca do stolpca različno in ga lahko narišemo na grafu; pove nam nekaj o tem, kako zahtevne pare razredov je poskušal razvozlati posamezni model našega ansambla. Ta poskus smo pognali trikrat, vsakič z različno naključno postavitvijo prvih 50 stolpcev, ki smo jim potem s požrešnim algoritmom dodali še 50 novih stolpcev; rezultate kaže graf na sliki 3.3.

Eden od parametrov poskusa je bil, da par razredov ostane na seznamu tabu $T = 5$ iteracij; če smo ga na primer uporabili pri gradnji stolpca i, ga ne smemo uporabiti pri stolpcih $i+1, \ldots, i+5$, lahko pa ga spet uporabimo pri stolpcu $i+6$. Vidimo pa lahko, da tudi naš graf „niha“ s periodo 6, na vsakih 6 stolpcev pride lokalni maksimum in od drugega maksimuma naprej se ti



Slika 3.4. Analiza obnašanja algoritma B iz razdelka 3.2 z uporabo seznama tabu. Ko algoritem uporabi nek par razredov pri gradnji trenutnega stolpca, ga v naslednjih T stolpcih ne sme ponovno uporabiti, kasneje pa ga lahko. Graf kaže v odvisnosti od x, kolikokrat se je zgodilo, da je bil nek par razredov ponovno uporabljen točno x stolpcev po svoji zadnji predhodni uporabi. Pri teh poskusih je bil $T = 5$ in vidimo lahko, da so bili pari razredov velikokrat ponovno uporabljeni takoj, ko je bilo to mogoče ($T + 1$ stolpcev po svoji predhodni uporabi).

maksimumi ne spreminjajo več kaj dosti. Dogaja se torej naslednje: prvih nekaj modelov se osredotoči na pare razredov z največjim številom zamenjav in nekaj teh zamenjav celo uspe odpraviti (zato so kasnejši maksimumi manjši od prvega); pri naslednjih nekaj modelih so najbolj problematični pari razredov na seznamu tabu, zato se ti modeli ukvarjajo z manj problematičnimi pari razredov (torej takimi, ki imajo manj zamenjav); ko mine 6 iteracij, pa pridejo tisti pari, ki smo jih uporabili pri prvem modelu, že iz seznama tabu in zdaj mnoge od njih uporabimo ponovno, saj so še vedno med prvimi po številu zamenjav. Pričakujemo torej lahko, da bo sedmi stolpec precej podoben prvemu, osmi drugemu in tako naprej. O tem se lahko prepričamo še drugače: vsakič, ko pri nekem stolpcu uporabimo tak par (c, c'), ki smo ga že uporabili pri nekem zgodnejšem stolpcu, si zapomnimo, koliko stolpcev nazaj je prišlo do zadnje predhodne uporabe tega para. Ta razlika je seveda nujno vsaj $T + 1$, v našem primeru torej 6. Graf na sliki 3.4 kaže, kako pogosto se pojavljajo posamezne možne razlike.

Iz njega vidimo, da če nek par razredov uporabimo po večkrat (pri različnih stolpcih matrike), je daleč najbolj verjetno, da smo ta par nazadnje uporabili pred šestimi stolpci; z drugimi besedami, par smo ponovno uporabili takoj, ko je bilo to mogoče (čim je padel iz seznama tabu). Iz tega lahko sklepamo, da seznam tabu ni res prava rešitev naših težav; vpeljali smo ga zato, da ne bi bil i-ti stolpec preveč podoben $(i - 1)$-vemu, dosegli pa smo le to, da je zdaj i-ti stolpec preveč podoben $(i - T - 1)$-vemu.

3.4 Uteževanje modelov

Naš dosedanji postopek gradnje matrike se je z vsakim novim stolpcem osredotočil na nekaj parov razredov, ki jih je dosedanji ansambel klasifikatorjev pogosto zamenjeval med sabo, in poskusil izboljšati razločevanje med razredi v teh parih. Videli smo, da se pogosto zgodi, da novi model števila zamenjav pri parih razredov, na katere se je osredotočil, ne zmanjša dovolj, zato se mnogi od teh parov razredov pojavijo tudi v naslednjih modelih (bodisi takoj ali pa ko padejo iz seznama tabu, če ga imamo). Na misel nam torej lahko pride, da bi modele utežili z nekimi utežmi, tako da bi imel novi model dovolj veliko težo pri glasovanju v ansamblu in bi potem mogoče dovolj uspešno zmanjšal število zamenjav pri svojih parih razredov.

Naj bo torej $u_j \in \mathbb{R}$ utež j-tega modela v ansamblu. Če je $y_j(x)$ napoved tega modela na primeru x in $m_{cj} \in \{-1, 0, 1\}$ element kodne matrike, ki pove, kako je model j videl primere razreda c med učenjem, je zdaj teža glasov ansambla za razred c enaka

$$z_c(x) := \sum_j u_j m_{cj} y_j(x).$$

Končna napoved ansambla za primer x pa je potem $z(x) := \arg\max_c z_c(x)$. Razmislimo, kaj se zgodi, če dodamo nov model r (s pripadajočim stolpcem v kodni matriki) in mu želimo izbrati utež. Teža glasov ansambla za razred c je zdaj pri primeru x enaka

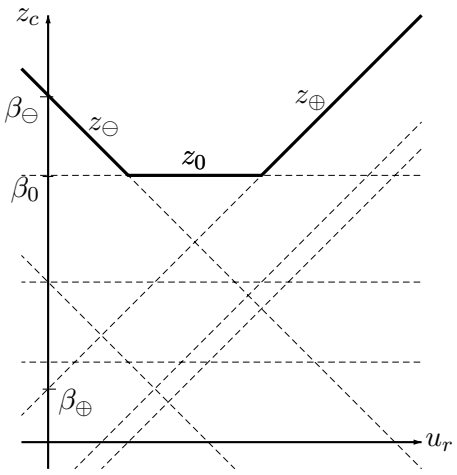
$$\begin{aligned} z_c(x) &= \textstyle\sum_{j=1}^{r} u_j m_{cj} y_j(x) \\ &= \textstyle\sum_{j=1}^{r-1} u_j m_{cj} y_j(x) + u_r m_{cr} y_r(x). \end{aligned}$$

Napovedi posameznih binarnih klasifikatorjev so (pri tem konkretnem x) take, kakršne pač so; enako velja tudi za elemente kodne matrike; odločimo se še, da uteži ostalih modelov trenutno ne bomo spreminjali; iz vsega tega pa sledi, da je v gornji enačbi za $z_c(x)$ na desni strani konstantno vse razen vrednosti u_r. Teža glasov ansambla za razred c je torej linearna funkcija uteži u_r; njen naklon je $\alpha_c := m_{cr} y_r(x)$, konstantni člen pa $\beta_c := \sum_{j=1}^{r-1} u_j m_{cj} y_j(x)$.

Končna napoved ansambla za x je odvisna od tega, pri katerem razredu c je dosežena največja vrednost $z_c(x)$. Vse $z_c(x)$ so linearne funkcije uteži u_r, njihovi konstantni členi so lahko zelo različni, glede naklona pa so le tri možnosti: naklon vsake od teh funkcij je namreč oblike $m_{cr} y_r(x)$, pri čemer $y_r(x)$ ni odvisna od razreda c, za m_{cr} pa so možne vrednosti le -1, 0 in 1. Označimo absolutno vrednost $|m_{cr} y_r(x)|$ z α; pri vsakem konkretnem c je torej $\alpha_c \in \{\alpha, 0, -\alpha\}$. Funkcije $z_c(x)$ lahko torej razdelimo v tri skupine glede na vrednost m_{cr}: v eni skupini so padajoče funkcije z naklonom $-\alpha$, v eni so konstantne funkcije (z naklonom 0), v eni pa naraščajoče funkcije z naklonom α.

Očitno je med več linearnimi funkcijami z enakim naklonom vedno največja tista, ki ima najvišji konstantni člen. Za napoved $z(x) = \arg\max_c z_c(x)$ pridejo v poštev torej le trije razredi c: namreč tisti, ki ima največjo vrednost β_c pri $\alpha_c = \alpha$, tisti z največjo β_c pri $\alpha_c = 0$ in tisti z največjo β_c pri $\alpha_c = -\alpha$. Označimo te tri razrede s $c_\oplus$, c_0 in $c_\ominus$, pripadajoče konstantne člene pa z $\beta_\oplus$, β_0 in $\beta_\ominus$. Imamo torej funkcije $z_\oplus(u_r) = \alpha u_r + \beta_\oplus$, $z_0(u_r) = \beta_0$ in $z_\ominus(u_r) = -\alpha u_r + \beta_\ominus$.

Namesto $\arg\max_c z_c(x)$ (pri čemer je šel maksimum po vseh možnih razredih c) je torej dovolj gledati maksimum teh treh funkcij: $z(u_r) = \max\{\alpha u_r + \beta_\oplus, \beta_0, -\alpha u_r + \beta_\ominus\}$. Ta funkcija je odsekoma linearna in sestavljata jo dva ali trije odseki. Pri $u_r = (\beta_\ominus - \beta_\oplus)/(2\alpha)$ velja $z_\oplus(u_r) = z_\ominus(u_r) = (\beta_\oplus + \beta_\ominus)/2$; pri manjših u_r je $z_\ominus$ večja od $z_\oplus$, pri večjih u_r pa manjša od $z_\oplus$. Če je $(\beta_\oplus + \beta_\ominus)/2 \geq \beta_0$, ima torej naša $z(u_r)$ le dva odseka (saj je z_0 takrat vedno $\leq \max\{z_\oplus, z_\ominus\}$); v nasprotnem primeru pa imamo tri odseke: na intervalu $(\beta_\ominus - \beta_0)/\alpha < u_r < (\beta_\oplus - \beta_0)/\alpha$ je maksimum dosežen pri z_0, levo od njega pri $z_\ominus$, desno od njega pa pri $z_\oplus$.



Primer funkcij $z_c(x)$ (ujemanje med napovedmi ansambla in c-to vrstico kodne matrike) v odvisnosti od uteži u_r pri fiksnem x. Vse funkcije so linearne in imajo le tri možne naklone: α, $-\alpha$ in 0. Končna napoved ansambla je $\arg\max_c z_c(x)$; vidimo, da je $\max_c z_c(x)$ (na grafu je narisana z debelo črto) odsekoma linearna funkcija z največ tremi odseki, tako da lahko pri tem za x kot napoved dobimo le enega od treh razredov, ostalih razredov pa zagotovo ne, ne glede na to, kako postavimo utež u_r.

V vsakem primeru torej vidimo, da če bi u_r počasi povečevali od $-\infty$ proti $+\infty$, bi le pri eni ali dveh vrednostih u_r prišlo do spremembe v napovedi ansambla za naš konkretni primer x; in vidimo tudi, da ni težko izračunati, pri katerih u_r pride do tega in kakšne so napovedi na vsakem intervalu (namreč $c_\oplus$, c_0 in $c_\ominus$). Ta razmislek lahko ponovimo za vsak x iz učne (ali pa, še bolje, validacijske) množice in dobimo seznam vseh možnih u_r-jev, pri katerih pride do kakšne spremembe v napovedi ansambla. Če ta seznam uredimo, lahko sistematično pregledamo vse relevantne vrednosti u_r, sproti popravljamo napovedi ansambla in tako poiščemo najboljši možni u.

naj bo L prazen seznam;
za vsak x iz učne (ali validacijske) množice:
    določi $c_\ominus$, c_0 in $c_\oplus$;
    napoved$[x] := c_\ominus$;

če je $(\beta_\oplus + \beta_\ominus)/2 \geq \beta_0$:
dodaj v L trojico $\langle(\beta_\ominus - \beta_\oplus)/(2\alpha), x, c_\oplus\rangle$;
sicer:
dodaj v L trojico $\langle(\beta_\ominus - \beta_0)/\alpha, x, c_0\rangle$
in trojico $\langle(\beta_\oplus - \beta_0)/\alpha, x, c_\oplus\rangle$;
tabela *napoved* vsebuje za vsak x napoved ansambla pri $u_r = -\infty$;
za ta nabor napovedi izračunaj kontingenčne tabele po razredih in
iz njih mikro- in/ali makropovprečne ocene ansambla
(na primer F_1);
uredi seznam L;

Seznam L zdaj vsebuje trojice oblike $\langle u, x, c\rangle$, urejene naraščajoče po u, ki nam povedo, da ko u_r doseže vrednost u, se napoved ansambla na primeru x spremeni v c. Zdaj se lahko sprehodimo po tem seznamu in računamo, kako se spreminja skupna ocena napovedi ansambla v odvisnosti od u:

$u' := -\infty$;
za vsak $\langle u, x, c\rangle \in L$, po naraščajočih u:
dosedanje agregatne ocene ansambla (npr. mikro- in
makropovprečja) veljajo za vse $u_r \in (u', u)$;
če je izbrana ocena (npr. makropovprečna F_1) najboljša doslej,
si jo zapomnimo (skupaj s tem, na katerem intervalu
u_r-jev smo jo dosegli);
$c' :=$ *napoved*$[x]$;
(* *ko* u_r *doseže* u*, se napoved primera* x *spremeni iz* c' *v* c *)
popravi kontingenčno tabelo razreda c':
če x ne pripada razredu c', zmanjšaj FP in povečaj TN,
sicer zmanjšaj TP in povečaj FN;
popravi kontingenčno tabelo razreda c:
če x pripada razredu c, povečaj TP in zmanjšaj FN,
sicer povečaj FP in zmanjšaj TN;
popravi agregatne ocene ansambla (mikro- in makropovprečja)
zaradi sprememb v kontingenčnih tabelah razredov c' in c;
$u' := u$;
na koncu gornje zanke imamo agregatne ocene ansambla, ki veljajo
za interval $u_r \in (u', +\infty)$; če je izbrana ocena najboljša
doslej, si jo zapomnimo;

Ko se spremeni kontingenčna tabela posameznega razreda, lahko v času $O(1)$ popravimo tudi agregatne ocene, kot so mikro- in makropovprečja. Zato je časovna zahtevnost te zanke linearna v odvisnosti od števila primerov v učni (ali validacijski) množici. Najpočasnejši del celotnega postopka je določanje $c_\ominus$, c_0 in $c_\oplus$ za vsak x, saj moramo v ta namen pri vsakem x izračunati napovedi vseh modelov in jih primerjati z vsemi vrsticami matrike. Toda

izračun vseh napovedi tako ali tako potrebujemo v vsakem primeru, tudi če posameznih modelov ne bi uteževali, saj napovedi potrebujemo za izračun nove matrike zamenjav, to pa potrebujemo pri tvorbi naslednjega stolpca kodne matrike. Zato v asimptotičnem smislu uvedba uteževanja modelov pravzaprav ni povečala časovne zahtevnosti našega postopka.

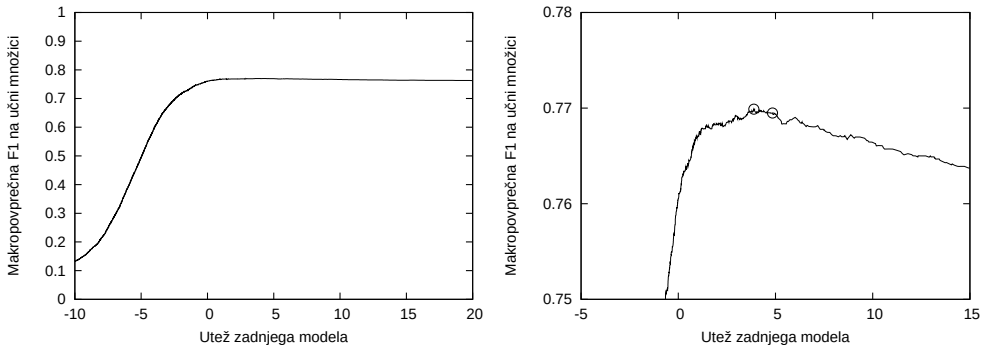
3.5 Poskusi z uteževanjem modelov

Postopek iz razdelka 3.4 se načeloma da uporabiti na katerem koli modelu v ansamblu — r si lahko izberemo poljubno in popravimo utež modela r, medtem pa držimo uteži ostalih modelov nespremenjene. Vendar pa smo pri naslednjih poskusih uteževanje uporabljali na požrešen način: ko dodamo v matriko nov stolpec (v ansambel pa nov dvorazredni klasifikator), mu s postopkom iz razdelka 3.4 določimo utež, kasneje pa je ne spreminjamo več.

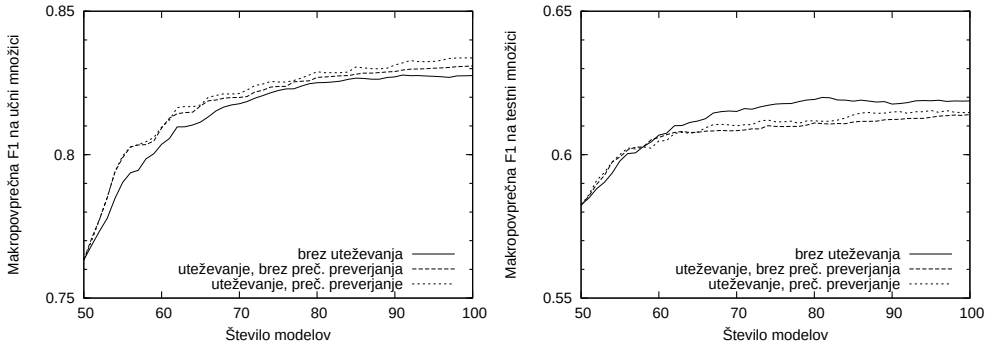
Graf na sliki 3.5 kaže primer, kako spreminjanje uteži enega modela vpliva na obnašanje ansambla. Vzeli smo naključno matriko s 50 stolpci in ji z algoritmom B dodali še en stolpec, nato pa opazovali makropovprečno F_1 v odvisnosti od uteži tega novega stolpca (oz. njemu pripadajočega modela). Izsek iz levega grafa je podrobneje prikazan na desnem grafu (ta torej temelji na istih podatkih kot levi, le razpon koordinat na oseh je drugačen). Utež 0 pomeni, da novega modela sploh ne uporabimo; utež 1 pomeni, da ima enako težo kot vsak od dosedanjih modelov; na desnem grafu pa sta označeni s krožcema še dve zanimivi vrednosti uteži: pri uteži $u \approx 3{,}88$ je dosežena največja makropovprečna F_1, pri uteži $u \approx 4{,}84$ pa najmanjše število zamenjav (vsota vseh nediagonalnih elementov v matriki zamenjav). Slednja mera, torej skupno število zamenjav, je zanimiva s tega vidika, ker se tudi naš postopek za gradnjo matrike (algoritem B) osredotoča na zmanjševanje števila zamenjav. Izkaže pa se, da je skupno število zamenjav zelo tesno korelirano z makropovprečjem F_1 (korelacijski koeficient je pribl. $-0{,}9988$); zato smo v nadaljevanju poskusov kot kriterij za določanje uteži vedno gledali le maksimizacijo F_1.

Oglejmo si zdaj, kako uteževanje modelov vpliva na obnašanje ansambla pri večjem številu modelov, dobljenih z algoritmom B. Grafa na sliki 3.6 kažeta makropovprečno vrednost mere F_1 v odvisnosti od števila stolpcev v matriki (enako kot pri dosedanjih poskusih smo začeli s 50 naključnimi stolpci in jim nato dodali še 50 stolpcev z algoritmom B iz razdelka 3.2); levi graf kaže rezultate na učni množici, desni pa na testni množici. Vsi prikazani rezultati so povprečja po treh ponovitvah poskusa (prvih 50 stolpcev, ki so dobljeni naključno, je torej pri vsaki ponovitvi drugačnih).

Vidimo lahko, da je učinek uteževanja modelov majhen, predvsem pa vodi v pretirano prilagajanje. Na učni množici pride do izboljšanja mere F_1, na testnih podatkih pa do poslabšanja. Da bi se pretiranemu prilagajanju izognili, smo izvedli tudi poskus s petkratnim prečnim preverjanjem na učni



Slika 3.5. Primer, kako spreminjanje uteži enega modela vpliva na obnašanje ansambla. Matriki s 50 naključnimi stolpci smo dodali še en stolpec s požrešnim algoritmom, naučili za vse stolpce pripadajoče dvorazredne klasifikatorje in opazovali obnašanje ansambla v odvisnosti od uteži zadnjega klasifikatorja (vsi ostali so imeli utež 1). Grafa kažeta makropovprečno F_1 celotnega ansambla (na učnih podatkih) v odvisnosti od uteži zadnjega modela; desni graf je le izsek iz levega, krožca na njem pa označujeta vrednost uteži, pri kateri je dosežena največja F_1 oz. najmanjše skupno število zamenjav.



Slika 3.6. Vpliv uteževanja posameznih modelov na makropovprečno F_1 celotnega ansambla. Levi graf kaže rezultate na učni, desni pa na testni množici. Vidimo lahko, da uteževanje privede do pretiranega prilagajanja učnim podatkom.

množici; prečno preverjanje smo uporabili tako za gradnjo novih stolpcev (matrika zamenjav izhaja iz napovedi na validacijskem delu učne množice) kot za njihovo uteževanje (pri vsakem stolpcu tako ob petkratnem prečnem preverjanju dobimo 5 uteži, na koncu pa vzamemo njihovo povprečje in ga uporabimo kot utež tega stolpca v končnem ansamblu, naučenem na celi učni množici). Kot vidimo iz grafov na sliki 3.6, je prečno preverjanje sicer zmanjšalo problem pretiranega prilagajanja, ni pa pripeljalo do boljših rezultatov, kot če modelov sploh ne bi uteževali.

3.6 Uteževanje kot vodilo pri gradnji matrike

Videli smo, da z uteževanjem modelov povzročimo predvsem pretirano prilagajanje učnim podatkom. Poskusimo drugačen pristop k sestavljanju naslednjega stolpca matrike. Za nek konkreten x vemo, kateri razred $c^\star$ je pravi; toda ansambel ga bo imel možnost napovedati le, če bo $c^\star$ eden od razredov $c_\ominus$, $c_\oplus$ in c_0 za ta x. Z drugimi besedami, med vsemi c, ki bodo imeli v novem stolpcu matrike enako vrednost kot $c^\star$, mora imeti $c^\star$ večjo konstanto β kot katerikoli drug c. Konstante β pa za vse razrede že poznamo, saj so odvisne le od napovedi in uteži dosedanjih modelov.

Torej imamo za vsak x in vsak c takšno omejitev:

$$\text{če je } \beta_c(x) > \beta_{c^\star}(x), \text{ potem bi si želeli, da bi bila } m_{cr} \neq m_{c^\star r}.$$

Pri tem pomeni r številko novega stolpca, ki bi ga radi dodali v matriko.

Razrede si lahko predstavljamo kot točke grafa:

za vsak primer x:
    naj bo $c^\star$ pravi razred primera x;
    za vsak c:
        if $\beta_c(x) > \beta_{c^\star}(x)$ **then** dodaj povezavo $(c, c^\star)$ v graf;

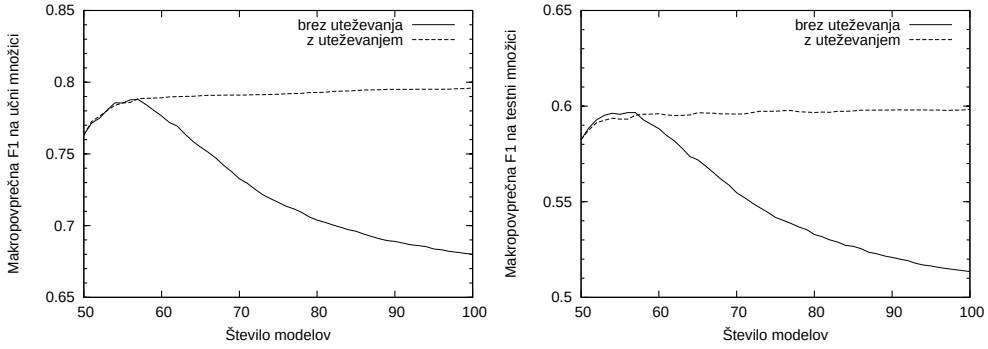
Če se je neka povezava dodala večkrat, ji pač povečajmo težo. Graf bi radi zdaj načeloma razbili na tri dele ($+1$, 0 in -1) in to tako, da bo vsota tež povezav znotraj vsakega dela čim manjša.

Dobili smo torej problem, ki je v tesni zvezi z dobro znanim problemom barvanja grafov [23, 44]. Običajna formulacija problema barvanja grafa je, da bi radi vsaki točki pripisali neko barvo, pri čemer nobena povezava ne sme imeti dveh krajišč enake barve; v okviru te omejitve pa bi želeli porabiti čim manj barv. V našem primeru bi radi graf pobarvali s tremi barvami, ki bodo ustrezale trem možnim vrednostim v novem stolpcu kodne matrike ($+1$, 0 in -1). Naš problem se od običajnega barvanja grafov razlikuje po tem, da je število barv predpisano vnaprej, graf pa bi radi pobarvali tako, da bo prekršenih čim manj omejitev (saj se najbrž ne bo mogoče izogniti temu, da bodo nekatere povezave vendarle imele obe krajišči enake barve).

Lotimo se ga lahko s hevrističnimi postopki, ki se uporabljajo tudi pri običajnem barvanju grafov. Ena možnost je na primer požrešni algoritem:

1 na začetku so vse točke nepobarvane;
2 za vsako točko $u \in V$:
3 pobarvaj u s tisto barvo (izmed treh možnih), s katero je pobarvanih najmanj njenih sosed;

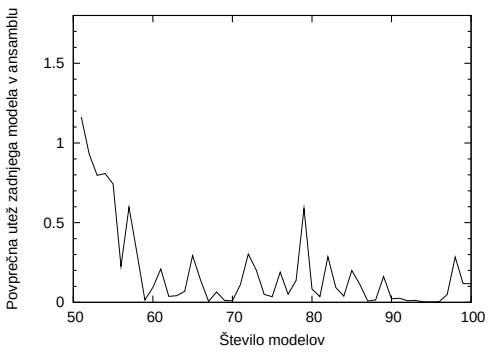
Rezultati algoritma so odvisni od tega, v kakšnem vrstnem redu pregledujemo točke; lahko tudi preizkusimo več vrstnih redov in si zapomnimo najboljši dobljeni rezultat. Še ena možnost je, da začnemo z neomejenim številom barv in jih nato počasi zmanjšujemo:



Slika 3.7. Gradnja kodnih matrik s hevristiko, ki temelji na barvanju grafov. Levi graf kaže rezultate na učni, desni pa na testni množici. Vidimo lahko, da je bilo uteževanje v tem primeru koristno, vendar se izkaže, da predvsem zaradi tega, ker je preprečilo, da bi ponavljanje preveč podobnih stolpcev v matriki naredilo preveč škode (glej sliko 3.8).

```
1  na začetku so vse točke nepobarvane;
2  za vsako točko u ∈ V:
3      za barvo u uporabi najmanjše naravno število, ki še ni barva
       nobene njene sosede;
4  dokler so v grafu več kot tri barve:
5      naj bo b barva z najmanj točkami;
6      vsako točko barve b prebarvaj v eno od ostalih barv tako,
           da se pri tem pojavi čim manj novih kršitev omejitev;
7      število kršitev lahko poskušamo zmanjšati tudi z lokalno
           optimizacijo in/ali simuliranim ohlajanjem;
```

Opisani pristop smo tudi preizkusili in to na isti ontologiji kot v prejšnjih razdelkih tega poglavja. Podobno kot v dosedanjih poskusih smo začeli s 50 naključnimi stolpci in nato dodali še 50 stolpcev s požrešnim algoritmom, le da smo zdaj vsakega od teh stolpcev sestavili s pomočjo barvanja grafa. Rezultate tega poskusa kažeta grafa na sliki 3.7. Žal se je tudi tokrat pojavil podoben problem kot pri hevristiki, ki je temeljila na matriki zamenjav (razdelek 3.3): še vedno se je dogajalo, da so si bili stolpci, ki smo jih dodali s požrešnim algoritmom, med seboj preveč podobni, zato se je uspeh napovedi ansambla kot celote že po prvih nekaj novih stolpcih prenehal povečevati. Pri poskusu brez uteževanja modelov se je ansambel zaradi preveč podobnih stolpcev sčasoma celo znatno poslabšal; če pa smo vključili tudi uteževanje modelov, so kasnejši stolpci večinoma dobivali uteži vse bližje 0. Tako od novih stolpcev sicer ni bilo posebne škode, vendar tudi koristi ne.



Slika 3.8. Graf kaže uteži posameznih modelov v ansamblu; prikazano je povprečje po treh ponovitvah poskusa, vsakič z drugačnim naključnim delom matrike (prvih 50 stolpcev). Če primerjamo ta graf s sliko 3.7, vidimo, da je bil učinek uteževanja v tem primeru predvsem ta, da so uteži zmanjšale škodo, ki bi jo matriki sicer povzročilo ponavljanje preveč podobnih stolpcev.

3.7 Zaključki in možnosti za nadaljnje delo

V tem poglavju smo si ogledali pristope za postopno gradnjo kodnih matrik, pri katerih ne sestavimo cele matrike naenkrat, pač pa vanjo postopno dodajamo nove stolpce enega za drugim. Naš pristop deluje kot požrešni algoritem, saj ko enkrat v matriko doda nov stolpec, ga kasneje ne poskuša več spreminjati. Ker je vseh možnih stolpcev neobvladljivo veliko (razen če je ontologija res izredno majhna), si ne moremo privoščiti, da bi na vsakem koraku preizkusili vse možne stolpce, zato potrebujemo neko hevristiko za izbor naslednjega stolpca.

Predlagali smo hevristiko, ki temelji na opazovanju matrike zamenjav. To je matrika, v kateri imamo za vsak par razredov podatek o tem, koliko primerov enega razreda je naš dosedanji ansambel dvorazrednih klasifikatorjev pomotoma klasificiral v drugi razred. Pri tvorbi novega stolpca si izberemo nekaj takih parov razredov, pri katerih je nastalo največ zamenjav. Od vsakega para uporabimo v novem stolpcu en razred kot pozitivnega, drugega pa kot negativnega; namen te hevristike je, da bi se novi stolpec osredotočil na razločevanje tistih razredov, na katerih se dosedanji ansambel največ moti.

Ta pristop smo preizkusili na srednje veliki hierarhiji (delu imenika ODP). Poskusi so pokazali, da po eni strani požrešni algoritem za postopno gradnjo matrike daje v nekem smislu boljše rezultate od popolnoma naključne gradnje matrik, saj z manj stolpci doseže enako dobre napovedi, za kakršne pri popolnoma naključnih matrikah potrebujemo več stolpcev.

Po drugi strani pa smo tudi pokazali tudi na pomembno težavo, na katero naletimo pri postopni gradnji matrike. Zaradi načina, kako se pri uporabi ansambla kombinirajo napovedi posameznih modelov v ansamblu, je vpliv posameznega modela na napovedi celotnega ansambla majhen (celo če dopustimo, da imajo različni modeli različno težo). Ker se po dodajanju novega stolpca napovedi ansambla ne spremenijo dovolj, se zelo lahko zgodi, da bo požrešen postopek za postopno gradnjo matrike tudi v naslednjem koraku sestavil zelo podoben stolpec in tako naprej. S tem pridemo do matrik, ki imajo preveč podobnih stolpcev, kar je, kot smo videli že v drugem poglavju, neugodno.

V konkretnem primeru hevristike, ki gradi stolpce s pomočjo matrike zamenjav, se ta problem odraža v dejstvu, da se matrika zamenjav po dodajanju novega stolpca ne spremeni dovolj, zato se lahko zgodi, da se bo tudi naslednjih nekaj stolpcev matrike (ki jih bomo sestavili v nadaljevanju postopka) ukvarjalo s približno istimi pari razredov kot prejšnji stolpec. Tako nastane matrika, katere stolpci so si med seboj preveč podobni.

Težavam s preveč podobnimi stolpci v matriki se lahko skušamo izogniti z uteževanjem posameznih stolpcev (oz. njim pripadajočih modelov v ansamblu), vendar se pri naših poskusih uteževanje ni posebej obneslo in je predvsem povečalo tveganje pretiranega prilagajanja.

Predstavili smo še eno hevristiko za gradnjo stolpcev, ki izhaja iz opažanja, da ko v matriko dodamo nov stolpec, pridejo pri vsakem primeru v poštev kot napoved ansambla le trije razredi (to, kateri bo izbran, pa je odvisno od uteži novega stolpca). Želimo si torej, da bi bil med temi razredi tudi tisti, ki mu ta primer dejansko pripada; to je kriterij, ki ga skušamo optimizirati pri gradnji novega stolpca. S to hevristiko se problem tvorbe novega stolpca izkaže za soroden problemu barvanja grafov. Poskusi pa so pokazali, da ima tudi ta hevristika podobne težave kot prejšnja, torej da gradi stolpce, ki so si med seboj preveč podobni.

Iz teh ugotovitev sledi več možnosti za nadaljnje delo. Ideja o čisto postopni in požrešni gradnji matrike stolpec za stolpcem se je izkazala za problematično; lahko bi razmišljali o hibridnem pristopu, ki matriko sicer gradi postopno, vendar po več stolpcev naenkrat in ob tem lažje pazi na to, da si stolpci med seboj niso preveč podobni. Še ena možnost je, da v matriko sicer dodajamo stolpce enega po enega, vendar v hevristiki za izbor novega stolpca tako ali drugače kot dodaten kriterij upoštevamo še Hammingovo razdaljo od novega stolpca do dosedanjih stolpcev matrike.

Na problem izbora naslednjega stolpca lahko pogledamo kot na optimizacijski problem iskanja po prostoru vseh možnih stolpcev. Namesto da nov stolpec izbiramo z nekimi zunanjimi hevristikami (kot smo to počeli v tem poglavju), bi ga lahko izbrali s sprehodom po prostoru stolpcev in pri tem uporabili kakšnega od znanih postopkov za optimizacijo in preiskovanje prostora.

Poglavje 4

Napovedovanje strukturnih sprememb v ontologijah

4.1 Uvod

Uveljavljen opis pojma ontologije je, da gre za konceptualizacijo nekega področja in da je ta konceptualizacija praviloma namenjena temu, da jo skupaj uporablja več ljudi.[1] Zato ni presenetljivo, da se morajo ontologije včasih tudi spreminjati, bodisi zato, ker se spreminja področje, ki ga opisujejo, bodisi ker se spreminja naše razumevanje tega področja, bodisi ker se spreminjajo nameni, zaradi katerih smo se konceptualizacije sploh lotili. Pri ontologijah, ki se spreminjajo skozi čas, se pojavi zanimivo vprašanje, ali lahko spremembe v njih avtomatsko napovedujemo vnaprej; s tem bi lahko na primer pomagali ljudem, ki ontologijo vzdržujejo, in jim predlagali ali priporočili morebitne spremembe.

V tem poglavju analiziramo praktičen primer velike ontologije, pri kateri je mogoče spremljati njen razvoj skozi več let. To je tematska hierarhija spletnega kazala Open Directory Project (ODP; dostopen na naslovu `http://www.dmoz.org`). Identificirali smo različne vrste strukturnih sprememb, do katerih je prihajalo v tej ontologiji, in analizirali njihovo pogostost. Na podlagi teh opažanj se v nadaljevanju osredotočamo na napovedovanje ene izmed teh strukturnih sprememb: dodajanja novega koncepta, ki postane podkoncept nekega že obstoječega koncepta in od njega tudi prevzame nekaj instanc. Izkaže se, da je to ena od najpogostejših vrst strukturnih sprememb v hierarhiji ODP, hkrati pa je tudi primerna za avtomatsko napovedovanje.

Ogledali smo si, kako lahko problem napovedovanja dodajanja podkonceptov formuliramo kot problem strojnega učenja. Glavni izziv pri tem je, kako opisati nek koncept z atributi in to na tak način, da bo nek napove-

[1] "A specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects — is called an ontology. [...] An ontology is an explicit specification of a conceptualization." [29], str. 199.

dni model (pridobljen s strojnim učenjem) lahko iz teh atributov napovedal, kdaj bi bilo treba pod opazovanim konceptom dodati nov podkoncept. Naš pristop izhaja iz predpostavke, da ontologija poleg konceptov vsebuje tudi instance in da se potreba po novem podkonceptu pojavi takrat, ko med instancami nekega obstoječega koncepta obstaja neka dovolj velika podskupina ali gruča instanc, ki so si med seboj dovolj blizu skupaj, da je smiselno zanje uvesti nov podkoncept. Pomagali smo si z razvrščanjem instanc v skupine (*clustering*) in za tako dobljene skupine izračunali razne statistične lastnosti, ki jih potem uporabimo kot atribute pri strojnem učenju. Pri hierarhiji ODP so instance kar besedila, tako da je mogoče za potrebe predstavitve instanc pri razvrščanju v skupine uporabiti tehnike s področja poizvedovanja po tekstovnih podatkih (*information retrieval*).

Predstavili bomo tudi eksperimentalno ovrednotenje opisanega pristopa. Poskusi na hierarhiji ODP kažejo, da je s tem pristopom res mogoče do neke mere napovedovati dodajanje podkonceptov. Na koncu tega poglavja opisujemo še nekaj možnosti za nadaljnje delo na tem področju, še zlasti v smeri napovedovanja drugih vrst strukturnih sprememb.

4.1.1 Dosedanje delo na tem področju

V delih [42] in [75] so definirali tri pristope k odkrivanju sprememb (*change discovery*): na podlagi strukture (*structure-driven*: predlagane spremembe napoveduje na podlagi analize strukture same ontologije), na podlagi uporabe (*usage-driven*: spremembe se predlaga na podlagi opazovanja, kako se skozi čas spreminjajo vzorci uporabe te ontologije) in na podlagi podatkov (*data-driven*: spremembe se predlagajo kot posledica sprememb v podatkih, ki opisujejo problemsko področje).

Z napovedovanjem sprememb na podlagi uporabe so se na primer ukvarjali Haase in sod. [31]. Cimiano in Völker [14] pa opisujeta pristop za odkrivanje sprememb na podlagi podatkov, kar sta vključila v okolje za gradnjo ontologije iz zbirke tekstovnih dokumentov.

Nedaven pregled področja sprememb v ontologijah in njegove povezave s sorodnimi področji (evolucija, združevanje in integracija ontologij) je na primer Flouris in sod. [20].

Zelo podrobno tipologijo sprememb v ontologijah so definirali Maynard in sod. [45] v okviru evropskega projekta NeOn. Njihov pregled se osredotoča na zelo drobne, nizkonivojske operacije. Tipično strukturno spremembo, kot je dodajanje nove podkategorije v hierarhijo, karšna je na primer ODP, bi pri njihovi metodologiji razbili na zaporedje več zelo elementarnih korakov: najprej se ustvari novi koncept; nato se ustvari povezava starš-otrok (neke vrste relacija *is-a*) med konceptom in njegovim staršem (torej enim od že prej obstoječih konceptov); in končno se v novi koncept (če je to potrebno) prenese nekaj instanc, ki so doslej pripadale staršu. Poleg tega opisujejo ti avtorji tudi precej operacij, ki pridejo v poštev le pri semantično bogatejših

ontologijah, torej takšnih, pri katerih obstaja več vrst relacij, atributi so bolj jasno strukturirani (npr. določeni so njihovi tipi in morebitne druge omejitve glede njihove vrednosti), nabor možnih atributov neke instance je odvisen od tega, kateremu razredu ta instanca pripada (pri ODP imajo vse instance ene in iste atribute: URL, naslov in opis).

Naš nabor strukturnih operacij je manjši in preprostejši od njihovega, to pa iz več razlogov. (1) Pri semantično preprostejši ontologiji, kakršna je ODP (takšne ontologije pa so tudi v praksi najbolj razširjene), mnoge od operacij iz [45] sploh ne pridejo v poštev. (2) Omejiti se moramo na operacije, ki jih je mogoče opaziti in napovedovati na podlagi tistih podatkov, ki so o razvoju ontologije ODP skozi čas na voljo. Na primer, v ODP se nove kategorije nikoli ne ustvari, ne da bi ob tem ta nova kategorija postala podkategorija neke že obstoječe. Zato dodajanja nove kategorije ne moremo, kot so to naredili Maynard in sod., razdeliti na ustvarjanje nove kategorije in na ustvarjanje povezave starš-otrok med njo in neko že obstoječo; naša operacija dodajanja nove kategorije nujno vsebuje oba tadva koraka. (3) Pri napovedovanju strukturnih sprememb gre v bistvu za problem modeliranja človeškega znanja, ki je sicer v veliki meri implicitno: za hierarhijo, kot je ODP, skrbijo človeški uredniki, ki se po svoji presoji odločajo, kdaj dodati ali zbrisati nek koncept ali izvajati druge podobne strukturne operacije. Smiselno je torej, da skušamo njihovo delo modelirati na približno takem nivoju, na kakršnem si ga tudi oni v mislih konceptualizirajo; in tipičen urednik ODP verjetno razmišlja bolj v smislu „premaknil bom kategorijo c_1 tako, da bo njen starš po novem kategorija c_2“, ne pa v smislu „pobrisal bom povezavo starš-otrok med c_3 in c_1, nato pa bom dodal povezavo starš-otrok med c_2 in c_1“. Pri našem napovedovanju strukturnih sprememb bomo torej izhajali iz operacij, o kakršnih verjetno razmišljajo tudi človeški skrbniki ontologije ODP.

4.2 Zaznavanje strukturnih sprememb s primerjavo dveh stanj ontologije

4.2.1 Open Directory Project in njegova ontologija

Za preučevanje strukturnih sprememb v ontologijah si je koristno ogledati primer ontologije iz stvarnega življenja, pri kateri lahko opazujemo, kako se je spreminjala skozi čas. Poleg tega si želimo tudi, da bi bila ontologija dovolj velika, tako da bomo iz nje lahko pridobili dovolj podatkov za učenje napovednih modelov. Zato smo se odločili uporabiti ontologijo spletnega imenika Open Directory Project (ODP), ki jo je moč dobiti na naslovu `http://www.dmoz.org/`.

V ontologiji ODP kot koncepti nastopajo tematske kategorije; s pomočjo relacije starš-otrok so organizirane v drevo. Takšni relaciji ponavadi pravimo tudi *is-a*, vendar ni odveč poudariti, da je pri tematskih hierarhijah, kakršna

je ODP, takšna interpretacija smiselna šele, če si predstavljamo, da govori o množicah dokumentov, ne pa o konceptih samih. Na primer, v ODP je kategorija "Open Source" podkategorija kategorije "Computers". Očitno je, da snovalci ontologije ODP s tem niso hoteli reči, da je odprta koda poseben primer računalnika; pač pa je ta povezava smiselna, če si jo razlagamo kot trditev „dokumenti, ki govorijo o odprti kodi, s tem neizogibno govorijo tudi o računalništvu". Povezava starš-otrok od c_2 do c_1 v tematski hierarhiji torej pomeni, da je vsak dokument, ki pripada tematiki c_1, hkrati tudi dokument, ki pripada tematiki c_2. V splošnem bi lahko takšna relacija tvorila poljuben usmerjen acikličen graf, v primeru ODP pa je relacija starš-otrok kar drevo in bomo zato tudi v nadaljevanju tega poglavja govorili o njej kot o drevesu.

Poleg relacije starš-otrok obstaja v ODP še ena relacija, namreč „simbolične" ali navzkrižne povezave. Te kažejo iz ene kategorije na drugo, ki ni njena podkategorija, ji je pa vsebinsko sorodna. Če bi upoštevali te povezave kot enakovredne povezavam starš-otrok, bi namesto drevesa dobili nek precej dobro povezan graf, v katerem je (zaradi povezav, ki kažejo navzgor po drevesu) iz skoraj vsakega koncepta dosegljiva v enem ali več korakih skoraj polovica vseh konceptov v drevesu. Teh povezav pri našem opazovanju strukturnih sprememb v ODP nismo upoštevali.

Vsaka kategorija v ODP ima ime in kratek opis. Običajni uporabnik pri brskanju po ODPjevem spletnem imeniku sicer opisa kategorije praviloma ne vidi, pač pa je ta opis shranjen v opisu ontologije v formatu RDF, ki ga je mogoče prenesti z ODPjeve spletne strani in je primeren za nadaljnjo strojno obdelavo. Ime kategorije v ožjem smislu ni enolično; enolično postane šele, če kot del imena štejemo tudi imena njenih prednic v drevesu, vse do korena. Tako na primer obstaja več različnih kategorij z imenom "Accounting", njihova polna imena pa so različna: "Business/Accounting", "Computers/Software/Accounting" idr.

Poleg kategorij vsebuje ontologija ODP tudi instance; to so pravzaprav kar povezave na spletne strani zunaj ODPja. Vsaka instanca vsebuje poleg samega naslova (URLja) tiste zunanje spletne strani tudi naslov in kratek tekstovni opis strani, na katero ta povezava kaže. Naslov je ponavadi dolg le nekaj besed, opis pa mogoče en ali dva stavka. Tako smo lahko pri našem delu vsako instanco v ODPju obravnavali kot kratko besedilo in si pomagali s tehnikami s področja analize besedil (*text mining*).

Dejstvo, da ontologija vsebuje poleg konceptov tudi instance, je za naš pristop zelo pomembno; to, da so instance ravno besedila, pa ni tako kritično. Kot bomo videli v nadaljevanju, je predpostavka pri našem pristopu zgolj ta, da je mogoče na instancah izvajati razvrščanje v skupine, to pa praviloma pomeni, da znamo med njimi definirati neko mero podobnosti (ali pa mero razdalje).

Ontologija ODP je za naše namene še posebej zanimiva zato, ker je njen razvoj skozi čas dobro dokumentiran. V ODPjevem arhivu (`http://rdf.dmoz.org/rdf/archive/`) obstajajo posnetki (*snapshots*) stanja cele

ontologije v različnih trenutkih v preteklosti. Vsega skupaj je na voljo približno 50 takšnih posnetkov, med drugim po en na mesec za obdobje od julija 2003 naprej, nekaj posnetkov v večjih časovnih razmikih pa iz še starejšega obdobja (od januarja 2001 naprej).

Ena od slabosti ODPja kot vira podatkov za napovedovanje strukturnih sprememb pa je žal ta, da je med dvema zaporednima posnetkoma starega stanja ontologije običajno časovni razmik približno enega meseca. V tem času se seveda v veliki in aktivno vzdrževani ontologiji, kakršna je ODP, zgodi veliko strukturnih sprememb. Lahko se tudi zgodi, da več sprememb zadeva isti predel ontologije in je zgolj s primerjavo starega in novega stanja nemogoče enolično ugotoviti, katere spremembe so se zgodile in v kakšnem vrstnem redu. V nadaljevanju opisujemo nabor hevristik, s katerimi lahko primerjamo dve stanji ontologije in sestavimo zaporedje operacij, ki bi lahko pripeljale ontologijo od enega stanja do drugega; pri tem pa seveda nimamo zagotovila, da je to ravno tisto zaporedje operacij, ki so ga v resnici izvedli ODPjevi uredniki. (V splošnem je takšno zaporedje operacij odvisno tudi od tega, kakšen nabor osnovnih operacij smo pripravljeni uporabljati v njem.)

4.2.2 Elementarne strukturne spremembe

Spremembe v (sematično preprostejši) ontologiji lahko v grobem razdelimo na tiste, ki zadevajo koncepte (t.j. kategorije), in tiste, ki zadevajo instance (t.j. dokumente). Med slednje lahko pri ODP štejemo dodajanje novih dokumentov (npr. novih spletnih strani, ki so jih opazili ODPjevi uredniki) v ontologijo, brisanje starih (npr. ker povezava na neko spletno stran ne deluje več) in premeščanje obstoječih dokumentov iz ene kategorije v drugo. Z napovedovanjem teh sprememb se ne bomo ukvarjali, prvič zato, ker to pravzaprav niso strukturne spremembe, drugič pa zato, ker bi bili za njihovo napovedovanje potrebni dodatni zunanji podatki, torej podatki, ki niso del ontologije, pač pa so jih imeli na voljo ODPjevi uredniki, ko so se o posamezni spremembi odločali. Do takšnih podatkov za nazaj niti ni nujno mogoče priti; na primer, nek dokument so mogoče zbrisali iz ODPja zato, ker je kazal na neko spletno stran, ki je prenehala obstajati; podatkov o tem, kaj se je v preteklosti dogajalo z neko spletno stranjo in kdaj je začela ali prenehala delovati, pa ni moč dobiti v takšnem obsegu in s takšno natančnostjo, da bi bili zanimivi za avtomatsko obdelavo.[2] Podobno bi za napovedovanje dodajanja novih dokumentov v ODP potrebovali podatke o tem, kakšne strani so obstajale na spletu v določenem trenutku v preteklosti (da so jih potem lahko ODPjevi uredniki odkrili in razmislili o tem, ali bi jih dodali v ontologijo), kar je ravno tako nerealistično.

[2]Do neke mere je takšne podatke mogoče dobiti v internetnem arhivu na `http://www.archive.org/`, vendar v njem marsikakšna stran manjka ali pa je arhivirana zelo poredko in torej arhiv nudi premalo podatkov o njenem razvoju skozi čas.

Zato se bomo v nadaljevanju osredotočili na spremembe na nivoju kategorij. Načeloma lahko, kar se tiče kategorij, en posnetek ontologije vedno predelamo v drugega z uporabo le dveh vrst elementarnih strukturnih operacij: dodajanja in brisanja kategorij. Če primerjamo množico kategorij v nekem posnetku ontologije z množico kategorij v posnetku, ki je nastal mesec dni prej, lahko vidimo, katere kategorije so medtem izginile in katere so se pojavile na novo. Tole je na primer odlomek iz seznama sprememb, ki so se zgodile znotraj poddrevesa "Top/Computers" v času med 3. aprilom in 1. majem 2007. Pri tem znak „⊖“ označuje izbrisane kategorije (torej take, ki so bile 3. aprila prisotne v ontologiji, 1. maja pa ne), znak „⊕“ pa nove kategorije (torej take, ki so bile prisotne v ontologiji 1. maja, ne pa 3. aprila):

⊖ Top/Computers/Open_Source/Software/Games/FPS
⊖ Top/Computers/Software/Internet/Servers/Directory/LDAP
⊖ Top/Computers/Software/Internet/Servers/Directory/LDAP/Products
⊖ Top/Computers/Software/Internet/Servers/Directory/LDAP/Standards_and_Organizations
⊖ Top/Computers/Software/Internet/Servers/Directory/LDAP/Products/Related_Middleware
⊖ Top/Computers/Software/Internet/Servers/Directory/LDAP/Products/Related_Client_Apps
⊕ Top/Computers/Open_Source/Software/Games/Shooter
⊕ Top/Computers/Programming/Languages/Smalltalk/Squeak/Croquet
⊕ Top/Computers/Programming/Languages/Smalltalk/Squeak/Croquet/News_and_Media
⊕ Top/Computers/Internet/Protocols/LDAP
⊕ Top/Computers/Internet/Protocols/LDAP/Standards_and_Organizations
⊕ Top/Computers/Internet/Protocols/LDAP/Software/Client
⊕ Top/Computers/Internet/Protocols/LDAP/Software/Server
⊕ Top/Computers/Internet/Protocols/LDAP/Software

Kot lahko vidimo iz tega seznama, bi bilo neugodno, če bi skušali spremembo iz enega stanja ontologije v drugega opisati le s pomočjo takšnih elementarnih operacij. Res je sicer, da lahko aprilski posnetek ontologije predelamo v majskega tako, da pobrišemo prvih šest kategorij z gornjega seznama in dodamo preostalih osem; vendar pa ni dvoma o tem, da so si ODPjevi uredniki, ko so aprila 2007 urejali ta del ontologije, svoje delo zamišljali kot zaporedje bolj abstraktnih, višjenivojskih strukturnih sprememb. Vsaka od teh sprememb pa se je potem izrazila kot zaporedje ene ali več elementarnih dodajanj ali brisanj kategorij.

Pri našem gornjem primeru na primer vidimo, da sta brisanje kategorije ".../FPS" in dodajanje ".../Shooter" pravzaprav tesno povezani operaciji: urednik je v bistvu le preimenoval kategorijo "FPS" v "Shooter" (FPS je namreč kratica za "first-person shooter" in se mu je verjetno zdela premalo razumljiva, da bi bila primerna za ime kategorije).

Podobno pri brisanjih in dodajanjih kategorij, povezanih z LDAP, vidimo, da je bilo celotno LDAPovo poddrevo, ki je bilo prej poddrevo kategorije ".../Internet/Servers/Directory", premaknjeno pod kategorijo ".../Internet/Protocols"; poleg tega pa so znotraj LDAPa poddrevo "Products"

preimenovali v "Software" in ga malo preuredili. Drugo LDAPovo poddrevo, "Standards and Organizations", pa se ni spremenilo.

Poddrevo, ki ga sestavljata kategorija ". . . /Squeak/Croquet" in njena podkategorija "News and Media" pa je, kot se izkaže, res povsem novo. Novi v posnetku z dne 1. maja 2007 nista le tidve kategoriji, ampak tudi vsi njuni dokumenti (3. aprila v ontologiji ni bilo še nobenega od njih). Očitno to pomeni, da so uredniki opazili neko novo skupino spletnih strani, ki prej bodisi niso obstajale ali pa nanje niso bili pozorni, in so zdaj ne le dodali v svoj imenik povezave nanje, ampak so zaradi teh povezav tudi ustvarili dve novi kategoriji.

Poleg strukturnih sprememb, ki jih ilustrira gornji primer, so v ODP pogoste še nekatere druge. Značilen pojav, na katerega odpade precejšen delež novih kategorij, je ustvarjanje novih kategorij „na zalogo", torej ne da bi imeli že pripravljene tudi neke dokumente, ki bi jih želeli dodati vanje. Uredniki dodajajo včasih cele skupine praznih kategorij, očitno v pričakovanju, da se bodo dokumenti zanje sčasoma začeli pojavljati, strukturo tega novega dela drevesa pa da je mogoče dovolj dobro definirati že vnaprej. Tako so na primer v mesecu med 1. aprilom in 4. majem 2004 dodali 36 novih kategorij pod kategorijo "Top/Computers/Programming/Internet/ASP/ASP.NET/Web_Hosting". Imena novih kategorij so "A", . . . , "Z", "0", . . . , "9", torej je bil namen očitno ta, da se obstoječi dolgi spisek ponudnikov spletnega gostovanja razdeli na krajše dele glede na prvo črko imena. Pri tem so zaradi doslednosti ustvarili tudi nekaj kategorij, ki so še nekaj časa ostale prazne, brez dokumentov (ker se pač nobenemu ponudniku ime ni začelo s tisto črko, npr. J in Q).

Podobno se včasih ustvari skupina sorodnih podkategorij (od katerih so mnoge sprva prazne), ki ustrezajo neki že znani skupini konceptov iz stvarnega sveta, npr. iz geografije. Pogost primer je, da neka kategorija nenadoma dobi 51 novih podkategorij, po eno za vsako državo znotraj ZDA. Spet drugi deli ODPjeve hierarhije se zgledujejo po že obstoječih ontologijah, kot so taksonomske enote v botaniki in naravoslovju. Nenadoma se lahko pojavi celo poddrevo (razvejeno in globoko več nivojev, vendar s pretežno praznimi kategorijami), ker se je nek urednik odločil v ODP uvoziti nek del obstoječe hierarhije taksonomskih enot (redov, družin, rodov ipd.). Tovrstnih dodajanj celih skupin kategorij na zalogo ne bomo skušali napovedovati, ker so preveč odvisna od predznanja urednikov ODPja, da bi jih lahko računalniški model napovedal zgolj iz podatkov o sami ontologiji. Zato se bomo osredotočili le na napovedovanje strukturnih sprememb v primerih, ko je bila sprememba res urednikov odziv na obstoječe podatke v ontologiji (dosedanje koncepte in instance) in ko je torej smiselno razmišljati o tem, da bi se dalo to spremembo na podlagi teh podatkov tudi avtomatsko napovedati. Primer takšne spremembe je, če se urednik odloči, da ima neka kategorija preveč dokumentov in/ali da so po vsebini preveč raznoliki, zato se odloči dodati eno ali več podkategorij in nekaj dokumentov prerazporediti mednje.

4.2.3 Hevristike za prepoznavanje višjenivojskih strukturnih sprememb

Kot smo videli v prejšnjem razdelku, lahko s primerjanjem dveh zaporednih posnetkov ontologije trivialno sestavimo zaporedje elementarnih brisanj in dodajanj, ki predelajo staro stanje v novo, vendar pa so bolj zanimive operacije abstraktnejše in sestavljene iz več elementarnih brisanj in/ali dodajanj. Poleg tega se lahko v mesecu dni med dvema posnetkoma ontologije zgodi več takšnih operacij; če te operacije vplivajo na isti del drevesa, ni očitno, kako iz elementarnih operacij, ki jih vidimo ob primerjavi dveh posnetkov ontologije, sestaviti neko primerno zaporedje višjenivojskih strukturnih sprememb. Zato moramo razviti nek dovolj robusten nabor hevristik, s katerimi bo mogoče identificirati vsaj nekatere tovrstne višjenivojske strukturne operacije, pri čemer pa se moramo zavedati, da v nekaterih primerih pač ne bodo rekonstruirale pravega zaporedja višjenivojskih operacij.

Izkaže se, da je največji delež elementarnih dodajanj in brisanj posledica preimenovanja kategorij (kot npr. preimenovanje "FPS" v "Shooter" v prejšnjem razdelku). Če se preimenuje neka kategorija z veliko potomkami v drevesu, se bo to odrazilo v velikem številu elementarnih dodajanj in brisanj (kot da bi bila vsaka od potomk izbrisana in nato dodana na novem mestu v drevesu). Čeprav je ta operacija pogosta, pri njej v bistvu ne gre za strukturno operacijo, zato bi radi takšna preimenovanja odkrili in jih izločili iz nadaljnje obravnave. V ta namen lahko uporabimo naslednjo hevristiko, ki temelji na merah natančnosti (*precision*) in priklica (*recall*). Naj bo C neka kategorija (iz starega stanja ontologije), ki je v novem stanju ontologije ni; množico vseh dokumentov v kategoriji C in njenih potomkah v drevesu označimo s S. Podobno naj bo C' neka kategorija iz novega stanja ontologije in S' množica vseh dokumentov, ki se v novem stanju nahajajo v C' ali v kateri od njenih potomk. Priklic kategorije C' glede na kategorijo C lahko zdaj definiramo kot $|S \cap S'|/|S|$, njeno natančnost pa kot $|S \cap S'|/|S'|$. Če je C' res le kategorija C pod novim imenom, bi pričakovali, da sta tako natančnost kot priklic visoka. Argument za visok priklic je očiten; če se je C preimenovala v C', so dokumenti iz C zdaj v C'. Vendar pa visok priklic sam po sebi ni dovolj, saj bo ta praviloma tem višji, čim višje po drevesu gremo, in najvišji priklic bi dobili, če bi za C' vzeli kar koren nove ontologije. Če torej poleg visokega priklica zahtevamo od C' tudi visoko natančnost, se bomo lažje izognili temu, da bi za C zmotno dobili vtis, da se je preimenoval v neko zelo splošno kategorijo. Na področju poizvedovanja po tekstovnih podatkih (*information retrieval*) priklic in natančnost pogosto skombinirajo v mero F_1, ki je definirana kot harmonična sredina priklica in natančnosti. Ta mera je za naš namen prav primerna, saj je harmonična sredina dveh števil ponavadi bližja manjšemu od njiju, torej bo F_1 visoka le, če sta visoka tako priklic kot natančnost.

Lahko se zgodi, da so bili v ontologijo med starim in novim stanjem

dodani tudi novi dokumenti in da so se nekateri od njih znašli v kategoriji C'; ti dokumenti bi torej povečali množico S', ne pa tudi $S \cap S'$ (ker jih v starem stanju ontologije ni, torej tudi v množici S ne), zato bi bila zaradi njih natančnost nižja. Ker ne bi bilo pošteno, da bi takšni novi dokumenti vplivali na našo oceno tega, ali se je C preimenovala v C' ali ne, bomo v množico S' vzeli le tiste dokumente, ki so obstajali v ontologiji že v njenem starem stanju.

Zdaj znamo torej za vsako izbrisano kategorijo C iz starega stanja poiskati, katera kategorija v novem stanju se z njo najbolje ujema (torej ima najvišjo vrednost F_1, merjeno na množicah njunih dokumentov); recimo ji $m(C)$. Načeloma se lahko zgodi, da pri nobeni kategoriji novega stanja ne pride do kaj posebej dobrega ujemanja, npr. če je bila kategorija C s svojimi dokumenti vred resnično izbrisana iz ontologije. Vendar pa so po naših opažanjih takšna brisanja redka; se pa lahko zgodi, če npr. zaradi hitrejšega izvajanja poskusov opazujemo le nek del celotne ontologije (neko poddrevo, npr. vse pod kategorijo "Top/Computers") , da je neka kategorija premaknjena zunaj opazovanega poddrevesa, kar je torej (če opazujemo le to poddrevo) videti enako, kot da bi bila popolnoma izbrisana.

Zato za odkrivanje premikov in preimenovanj kategorij upoštevamo le takšna ujemanja $(C, m(C))$, pri katerih je priklic $m(C)$ glede na C enak vsaj 90 %; recimo jim „ujemanja z visokim priklicem“. Naslednji korak je, da ujemanja na ravni posameznih kategorij skombiniramo v ujemanja celotnih poddreves. Na primer, če se izbrisana kategorija "Top/A/B/C" (ki ima podkategoriji "Top/A/B/C/D_1" in "Top/A/B/C/D_2") ujema (z visokim priklicem) z neko novo kategorijo "Top/E/C′" v novem stanju ontologije in če se njeni podkategoriji ujemata (z visokim priklicem) z novima kategorijama "Top/E/C′/D'_1" in "Top/E/C′/D'_2", potem je razumno o vsem tem govoriti kot o premiku celega poddrevesa s korenom v "Top/A/B/C", ne pa kot o množici sprememb, ki so se ločeno dogajale kategorijam C, D_1 in D_2 vsaki posebej. V splošnem je lahko poddrevo z začetkom pri C tudi globlje (ima C poleg otrok še vnuke in druge potomce) in takrat uporabljamo opisano hevristiko na naslednji način: rečemo, da se poddrevo s korenom C (v starem stanju ontologije) ujema s poddrevesom s korenom C' (v novem stanju ontologije), če sta izpolnjena naslednja dva pogoja: (1) za vsako potomko D kategorije C (v starem stanju ontologije) mora obstajati ujemanje $m(D)$ z visokim priklicem in kategorija $m(D)$ mora biti potomka kategorij C; (2) ta ujemanja morajo upoštevati tudi hierarhične odnose med kategorijami, torej mora za vsako tako D veljati $nad(m(D)) = m(nad(D))$, pri čemer $nad(\cdot)$ označuje nadkategorijo (torej starša v drevesu). Z drugimi besedami, o ujemanju dveh poddreves govorimo šele, če obstajajo ujemanja na nivoju posameznih kategorij in če ta ujemanja upoštevajo tudi hierarhično strukturo obeh poddreves. Ta definicija je v nekaterih pogledih ugodno robustna, saj na primer dopušča, da se v poddrevesu kategorije C' (v novem stanju ontologije) pojavi tudi kakšna nova kategorija ali pa se več obstoječih

kategorij zlije v eno.

Ko na opisani način odkrijemo ujemanja med celotnimi poddrevesi, je to ugodna podlaga za identifikacijo več vrst višjenivojskih sprememb v ontologiji:

- Če se C-jevo poddrevo (v starem stanju ontologije) ujema s poddrevesom C' (v novem stanju) in se nobeno drugo poddrevo (iz starega stanja) ne ujema s poddrevesom C' in če C' v starem stanju še ni obstajala in imata C in C' istega starša, potem rečemo, da se je kategorija C preimenovala v C'.

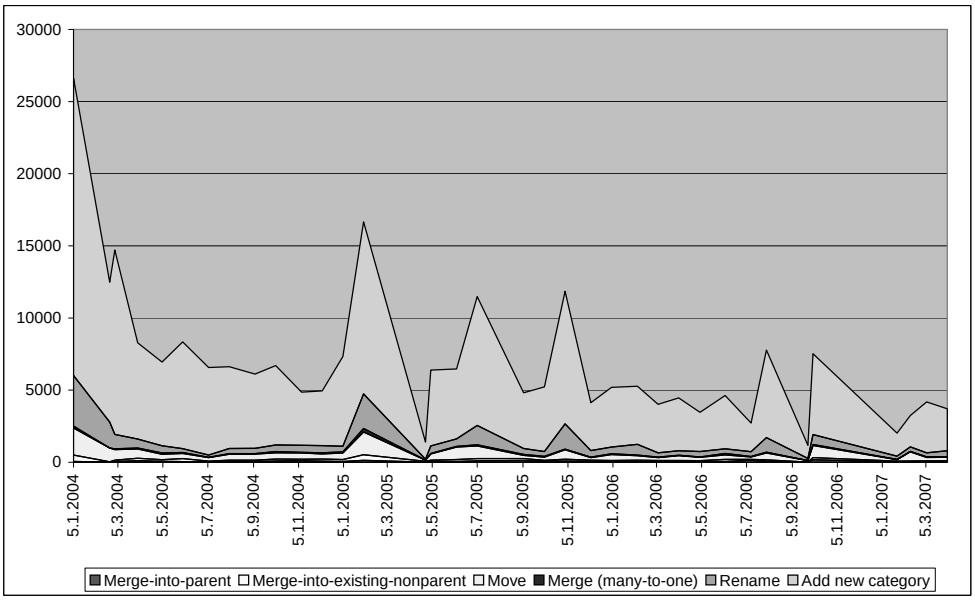
- Če so izpolnjeni opisani pogoji razen tistega, da imata C in C' istega starša, potem rečemo, da se je C premaknila na mesto C'.

- Če pa je C' že obstajala v starem stanju ontologije ali pa je sicer nova, vendar se je s poddrevesom C' ujemalo poleg C-jevega še kakšno drugo poddrevo iz starega stanja ontologije, potem rečemo, da se je C zlila v kategorijo C'. Zlivanja so lahko različnih vrst. Včasih se na primer kategorija zlije v svojega starša, če so se npr. uredniki odločili, da je bilo prejšnje razbitje starša na podkategorije nepotrebno ali preveč podrobno; včasih pa se tudi zgodi, da se kategorija zlije s kakšno bolj oddaljeno sorodnico, ne pa s staršem; lahko se tudi več kategorij zlije v eno.

Za ilustracijo si lahko ogledamo sliko 1, ki kaže pogostost različnih tu opisanih vrst višjenivojskih sprememb v ontologiji znotraj poddrevesa "Top/Computers" kategorije ODP prek obdobja treh let. Kot smo opisali zgoraj, se je dalo vsa elementarna brisanja kategorij pojasniti kot preimenovanja, premike ali zlivanja (pri čemer lahko zlivanja še podrobneje razdelimo na zlivanja v starša, zlivanja v ne-starša in zlivanja več kategorij v eno). Kar še ostane, so dodajanja resnično novih kategorij (za razliko od tistih elementarnih dodajanj, ki so posledica preimenovanj ali premikov obstoječih kategorij). Opazimo lahko, da so dodajanja najpogostejša vrsta strukturnih sprememb, sledijo pa jim preimenovanja in premiki. Zlivanja so v primerjavi z ostalimi operacijami redka. Ker je vprašljivo, v kolikšni meri lahko primenovanje štejemo za strukturno spremembo v ontologiji, in ker so zlivanja razmeroma redka, smo se v nadaljevanju osredotočili na dodajanja kot na najpomembnejšo in najpogostejšo vrsto strukturni sprememb v ontologiji ODP.

4.2.4 Različne vrste dodajanja kategorij

Kot smo videli na sliki 4.1, je dodajanje novih kategorij najpogostejša vrsta strukturnih sprememb, tudi po tistem, ko odmislimo kategorije, ki so navidez nove, v resnici pa so posledica preimenovanja, premikanja ali zlivanja starih kategorij. V tem razdelku si dodajanje novih kategorij oglejmo



Slika 4.1: Pogostost različnih vrst strukturnih sprememb v ontologiji.

podrobneje. V troletnem obdobju, ki ga pokriva slika 4.1, je bilo znotraj drevesa "Top/Computers" skupaj 1115 dodajanj kategorij. Slika 4.2 kaže, kako lahko ta dodajanja razdelimo na več vrst.

Včasih se zgodi, da nova kategorija ne postane list v drevesu, ampak se naenkrat doda celo poddrevo, ki ga poleg kategorije sestavljajo še ena ali več podkategorij in mogoče še njihove potomke. Približno 20 % na novo dodanih kategorij je tako imelo v drevesu starša, ki je bil tudi sam novo dodana kategorija (torej dodana v istem mesecu — spomnimo se, da smo imeli pri teh poskusih na voljo le približno en posnetek stanja ontologije na mesec). Dodajanja takšnih kategorij nismo poskušali napovedovati, saj je dovolj velik izziv že napovedati dodajanje posamezne kategorije, ne pa celega poddrevesa. Ostala dodajanja, ki si jih bomo ogledali v tem razdelku, so torej dodajanja kategorij, ki so postale podkategorije neke že od prej obstoječe kategorije.

Približno 10 % novih kategorij je bilo praznih, torej niso v prvem stanju ontologije, v katerem so se pojavile, vsebovale še nobenega dokumenta. Kot smo opisali v razdelku 4.2.2, nastopijo ti primeri večinoma zaradi sistematičnega dodajanja večjega števila kategorij naenkrat, npr. po ene kate-

gorije za vsako ameriško državo ali pa po ene za vsako črko abecede.

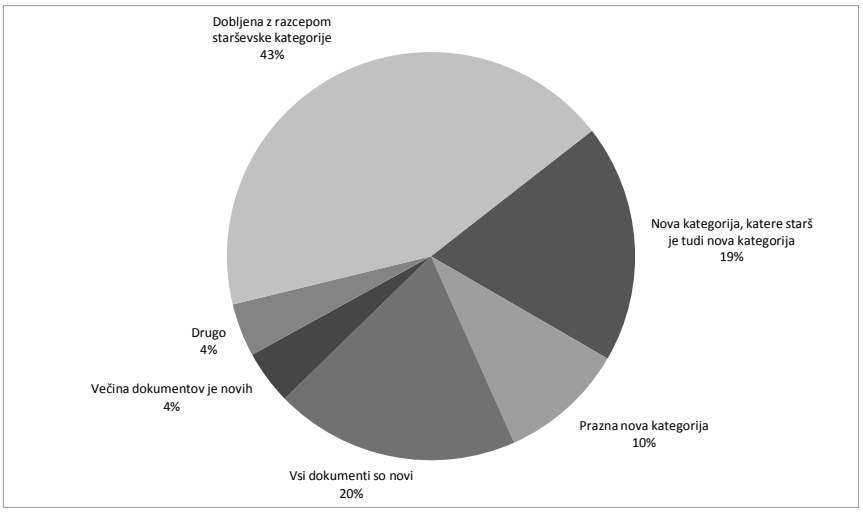
Približno 20 % novih kategorij sicer ni praznih, pač pa vsebujejo le nove dokumente, torej take, ki jih v starem stanju ontologije še ni bilo. To lahko pomeni, da je urednik opazil neko skupino spletnih strani o neki tematiki, ki v ontologiji predtem še ni bila pokrita, in je zdaj dodal te strani v ontologijo kot nove dokumente in ustvaril zanje tudi novo kategorijo (primer je kategorija "Croquet" iz razdelka 4.2.2). Če ne opazujemo cele ontologije, ampak le neko njeno poddrevo (kot na primer "Top/Computers" znotraj ontologije ODP pri naših poskusih), pa lahko ta pojav (nova kategorija s popolnoma novimi dokumenti) nastane tudi v primerih, ko je bila neka kategorija premaknjena v opazovani del ontologije od zunaj (torej iz nekega drugega dela celotne ontologije ODP).

Za približno 44 % novih kategorij se zdi razumno trditi, da so nastale z razdelitvijo neke že obstoječe kategorije (ki je postala roditelj nove kategorije v drevesu). To vrsto dodajanj smo definirali takole: nova kategorija (recimo ji C) mora biti otrok neke že obstoječe kategorije (recimo ji P); vsebovati mora vsaj en dokument, ki je bil prisoten že v starem stanju ontologije (poleg njega lahko sicer vsebuje tudi še kakšne nove dokumente), in od teh dokumentov, ki so bili prisotni že v starem stanju ontologije, jih mora večina priti iz kategorije P ali njenih potomk (ne pa iz ostalih delov ontologije, ki ne ležijo pod P). Z drugimi besedami, o razcepu obstoječe kategorije govorimo v primerih, ko nova podkategorija prevzame več dokumentov od starša kot iz ostalih delov stare ontologije. Pri naših prizadevanjih napovedovanja strukturnih sprememb smo se osredotočili prav na ta tip dodajanj. Žal se izkaže, da je večina kategorij, ki so bile v ontologijo dodane na ta način, precej majhnih; le približno tretjina (16 % vseh dodajanj) vsebuje vsaj pet dokumentov iz P.

Preostala dodajanja so tista, pri katerih vsebuje nova kategorija neko mešanico starih dokumentov iz P, starih dokumentov iz drugih delov ontologije in popolnoma novih dokumentov. V približno polovici teh primerov (4 % vseh dodajanj) prevladujejo novi dokumenti; pri ostalih pa je večina dokumentov sicer bila prisotnih že v stari ontologiji, vendar so med njimi tisti iz P (ali njenih potomk) v manjšini. Tako nastale nove kategorije so torej nastale s kombinacijo dodajanja novih podatkov (spletnih strani, ki so bile na novo vključene v kazalo ODP) in premeščanja starih podatkov (dokumentov, ki so bili v ODP že od prej), tako da ni očitno, da bi se jih dalo okarakterizirati na kakšen kratek in jedrnat način.

4.3 Napovedovanje dodajanja kategorij kot učni problem

Napovedovanje dodajanja kategorij lahko obravnavamo kot problem strojnega učenja. Posamezen primer tega učnega problema je kategorija na neki



Slika 4.2: Pogostost različnih vrst dodajanja kategorij.

točki v času; vprašanje, na katerega bi radi odgovorili, je, ali je pametno zdaj pod to kategorijo ustvariti kakšno novo podkategorijo. To je torej binarni (dvorazredni) klasifikacijski problem, pri čemer sestavljajo pozitivni razred tisti primeri, pri katerih je novo podkategorijo treba dodati, negativni razred pa tisti, pri katerih je ni treba.

Glavno vprašanje tu je, kako opisati vsak tak primer z množico atributov na tak način, da bo dobljena predstavitev primeren vhod za kakšnega od algoritmov strojnega učenja. Ti atributi bi morali zajemati podatke, pomembne za odločitev o tem, ali je dodajanje nove podkategorije pomembno ali ne. Kot smo opisali v razdelku 4.2.4, nismo napovedovali tistih dodajanj podkategorij, ki očitno izhajajo iz predznanja oz. informacij zunaj same ontologije; ostanejo torej dodajanja, ki vsaj delno temeljijo na obstoječi vsebini ontologije, torej na dokumentih v kategoriji, za katero se sprašujemo, ali naj dobi kakšno podkategorijo ali ne. Naš pristop temelji na domnevi, da uredniki ontologije ODP o ustvaritvi nove podkategorije verjetno razmišljajo takrat, ko v neki obstoječi kategoriji (recimo ji C) opazijo neko skupino dokumentov, ki govorijo o neki dovolj dobro definirani ožji tematiki (znotraj širše tematike, ki jo pokriva kategorija C); v tem primeru je smiselno to ožjo tematiko predstaviti z (novo) podkategorijo, ki postane otrok kategorije C v drevesu in prevzame tudi tiste C-jeve dokumente, ki so govorili o tej ožji tematiki. Če torej več takšnih dokumentov govori o isti ožji tematiki, lahko pričakujemo, da so si med seboj v nekaterih pogledih sorodni, da uporabljajo podobno terminologijo ipd. Če jih torej predstavimo kot točke v večrazsežnem prostoru, bi pričakovali, da bodo te točke ležale relativno blizu skupaj, torej da si bodo bližje, kot so si med seboj na splošno oddaljene točke iz C (ki pokriva širšo tematiko in bi zato pričakovali, da so dokumenti

iz C po prostoru malo bolj razpršeni). Pričakovali bi torej, da dokumenti za potencialno novo kategorijo tvorijo neko gručo ali podskupino (*cluster*) znotraj množice vseh dokumentov v kategoriji C. Zato smo si za ugotavljanje tega, ali tovrstne skupine dokumentov res obstajajo, pomagali s tehnikami s področja razvrščanja v skupine (*clustering*).

4.3.1 Predstavitev dokumentov v vektorskem prostoru

Za začetek predstavimo vsak dokument z vektorjem po postopku, ki je znan tudi kot „vreča besed“ (*bag of words*) in je uveljavljen pristop pri poizvedovanju po in strojnem učenju na tekstovnih podatkih [50]. Dokument d predstavimo z vektorjem $\mathbf{x}$, ki ima toliko komponent, kolikor je vseh različnih besed v celotni učni množici dokumentov, s katerimi se ukvarjamo. Če označimo te besede z $t_1, \ldots, t_n$, ima vektor $\mathbf{x}$ torej n komponent in posamezna vrednost v njem je določena s formulo

$$x_i = \textsc{tf}(t_i, d) \cdot \textsc{idf}(t_i).$$

Pri tem je $\textsc{tf}(t_i, d)$ (*term frequency*) število pojavitev besede t_i v dokumentu t; $\textsc{idf}(t_i)$ *inverse document frequency* pa definiramo kot kot $\log(|D|/\textsc{df}(t_i))$, pri čemer je $\textsc{df}(t_i)$ (*document frequency*) število učnih dokumentov, ki vsebujejo besedo t_i, množica D pa je množica vseh učnih dokumentov. Namen faktorja IDF je, da zmanjša vpliv tistih besed, ki so prisotne na splošno v velikem deležu dokumentov in zanje torej ne pričakujemo, da bi bile koristne pri ločevanju dokumentov različnih razredov med sabo. Na koncu vektor $\mathbf{x}$ ponavadi še normaliziramo, torej ga pomnožimo s takšnim skalarjem, da ima potem evklidsko dolžino 1, s čimer poskrbimo, da dolžina dokumenta nima prehudega vpliva na njegovo vektorsko predstavitev. O „vreči besed“ govorimo pri tej predstavitvi zato, ker se ne ozira na vrstni red besed v dokumentu, pač pa le na njihovo prisotnost in število pojavitev.

Predstavitev dokumentov z vektorji po metodi vreče besed je koristna osnova za merjenje podobnosti dokumentov. Če govorita dva dokumenta o podobni tematiki, bi pričakovali, da uporabljata tudi podobne besede in zato njuna vektorja kažeta v podobne smeri. To lahko merimo s kosinusom kota med njima, tega pa lahko računamo s skalarnim produktom.

$$\cos(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}/(||\mathbf{x}|| \cdot ||\mathbf{y}||),$$

kar je v našem primeru enako $\mathbf{x}^T\mathbf{y}$, ker smo naše vektorje normalizirali na evklidsko dolžino 1 ($||\mathbf{x}|| = ||\mathbf{y}|| = 1$). Bolj ko sta si dva dokumenta podobna, manjši bo kot med njima in večji bo kosinus tega kota.

4.3.2 Razvrščanje v skupine (clustering)

Mislimo si množico vseh dokumentov, ki so v ontologiji uvrščeni v kategorijo C ali kakšno od njenih potomk v drevesu. Za to množico nas zdaj zanima, ali

obstaja znotraj nje kakšna kompaktna podskupina dokumentov; to bi nam pomagalo pri odločanju o tem, ali bi bilo dobro pod C dodati še kakšno novo podkategorijo.

Za razvrščanje dokumentov v skupine smo uporabili metodo voditeljev (*k-means*), in sicer hierarhično bisekcijsko različico te metode [74]. Na začetku si predstavljamo celotno množico dokumentov kot eno skupino; nato pa na vsakem koraku izberemo eno od skupin in jo razcepimo na dve manjši skupini. Ustavimo se, ko se nam nabere 10 skupin, poleg tega pa tudi ne poskušamo razcepiti skupin z manj kot petimi dokumenti ali pa takih skupin, pri katerih bi ob razcepu nastala kakšna skupina z enim samim elementom. Ti ustavitveni pogoji izhajajo iz dejstva, da so množice dokumentov, s katerimi imamo opravka, pogosto precej majhne in z razbijanjem v skupine ne smemo pretiravati.

Za neko skupino vektorjev A lahko zdaj definiramo njen centroid, in sicer kot povprečje vektorjev iz A, normalizirano na enotsko dolžino:

$$\mathit{centroid}(A) = \textstyle\sum_{\mathbf{x}\in A} \mathbf{x} / || \sum_{x\in A} \mathbf{x}||.$$

Poleg tega definirajmo še varianco, in sicer kot povprečno razdaljo vektorjev iz A do centroida te množice. Ker nam kosinusna mera računa podobnost med dokumenti, ne pa razdalje, lahko za razdaljo vzamemo $1 - \cos$:

$$\mathit{var}(A) = \tfrac{1}{|A|} \textstyle\sum_{\mathbf{x}\in A}(1 - \cos(\mathbf{x}, \mathit{centroid}(A))).$$

Na vsakem koraku poskušamo na dve manjši skupini razbiti tisto skupino, ki ima največjo varianco.

Za razbitje skupine A na dve manjši skupini A_1 in A_2 uporablja metoda voditeljev naslednje korake:

1 Pripravi začetno razbitje A na dve skupini, A_1 in A_2.
2 Naj bo $\mathbf{c}_1 := \mathit{centroid}(A_1)$ in $\mathbf{c}_2 := \mathit{centroid}(A_2)$.
3 Naj bo $B_1 := \{\mathbf{x} \in A : \cos(\mathbf{x}, \mathbf{c}_1) > \cos(\mathbf{x}, \mathbf{c}_2)\}$ in $B_2 := A - B_1$.
4 Naj bo $A_1 := B_1$ in $A_2 := B_2$.
5 Ponavljaj korake 2–4, dokler ni izpolnjen ustavitveni pogoj.

Ustavitveni pogoj se ponavadi nanaša na število opravljenih iteracij ali pa na to, koliko dokumentov se je v zadnji iteraciji premaknilo iz ene skupine v drugo (torej $|A_1 \cap B_2| + |A_2 \cap B_1|$). V našem primeru smo postopek ustavili po petih iteracijah ali pa v primeru, da ni šel v zadnji iteraciji noben dokument iz ene skupine v drugo.

Za izbiro začetnega razbitja množice A na podskupini A_1 in A_2 (v koraku 1) obstajajo različni pristopi [76]. Lahko se na primer uporabi enak pristop kot v koraku 3, le da za voditelja c_1 in c_2 namesto centroidov vzamemo dva naključno izbrana dokumenta. Mi pa smo uporabili malo boljši pristop iz [76], ki najprej z analizo glavnih komponent (PCA) poišče glavno komponento množice A, torej tisto smer, v kateri ima množica A največjo

varianco. Naj bo $\mathbf{p}$ enotski vektor v tej smeri; za poljuben vektor $\mathbf{x}$ je potem $\mathbf{x}^T\mathbf{p}$ pravokotna projekcija vektorja $\mathbf{x}$ na smer $\mathbf{p}$. Izračunamo lahko povprečje teh projekcij:

$$\mu = \frac{1}{|A|} \sum_{\mathbf{x} \in A} \mathbf{x}^T \mathbf{p}$$

in nato za začetno razbitje vzamemo kar množici

$$A_1 = \{\mathbf{x} \in A : \mathbf{x}^T\mathbf{p} < \mu\} \quad \text{in} \quad A_2 = A - A_1.$$

Pri naših poskusih se je ta hevristika dobro obnesla in v kasnejših iteracijah korakov 2–4 ni prihajalo do velikega števila premikov dokumentov iz ene podskupine v drugo.

4.3.3 Od razbitja na skupine do vektorja atributov za kategorijo

Naj bo A začetna množica dokumentov v kategoriji C (in njenih potomkah) in naj bo $P = \{B_1, \ldots, B_k\}$ razbitje množice A na k disjunktnih skupin, ki smo jih dobili po metodi voditeljev iz prejšnjega razdelka. Kot smo rekli, nas za potrebe odločanja o tem, ali bi bilo dobro pod C ustvariti še kakšno podkategorijo, zanima predvsem to, ali obstaja znotraj A kakšna kompaktna podskupina elementov. Zato lahko iz razbitja P izračunamo na primer takšne atribute:

(1) Povprečna podobnost vsakega dokumenta s centroidom skupine, v katero je bil razvrščen. Ta mera nam pove nekaj o tem, kako tesne so na splošno skupine v razbitju P:

$$\frac{1}{|A|} \sum_{B \in P} \sum_{\mathbf{x} \in B} \cos(\mathbf{x}, \mathit{centroid}(B)).$$

(2) Naj bo B skupina z najmanjšo varianco: $B := \arg\min_{B' \in P} \mathit{var}(B')$. To je torej v nekem smislu najbolj kompaktna, najtesneje povezana podskupina, kar smo jih našli v razbitju P. V opisu kategorije C smo uporabili naslednje lastnosti te skupine:

- Velikost te skupine. Namesto velikosti $|B|$ same po sebi smo raje uporabili $\log |B|$, da ne bi imele velike skupine pretiranega vpliva na interval vrednosti, ki jih ta atribut zajema.
- Relativno velikost te skupine, torej $|B|/|A|$.
- Varianco te skupine, torej $\mathit{var}(B)$.
- Varianco te skupine, merjeno relativno glede na celotno množico A, torej $\mathit{var}(B)/\mathit{var}(A)$.

(3) Podobno kot varianco skupine lahko opazujemo tudi povprečno podobnost po vseh parih dokumentov v skupini (*average intra-cluster similarity*):

$$\text{AICS}(B') = \frac{1}{|B'|(|B'|-1)} \sum_{\mathbf{x},\mathbf{y}\in B', \mathbf{x}\neq\mathbf{y}} \cos(\mathbf{x},\mathbf{y}).$$

Naj bo zdaj $\tilde{B}$ skupina z največjo *AICS*, torej $\tilde{B} = \arg\max_{B'\in P} \text{AICS}(B')$. Za tako izbrano skupino $\tilde{B}$ lahko sestavimo atribute, analogne tistim iz točke (2), le da namesto variance uporabljamo AICS. Namen te skupine atributov je, da najbolj kompaktno skupino dokumentov poiščemo in opišemo še na malo drugačen način kot pri točki (2). Treba pa je priznati, da sta obe meri tako ali tako v tesni povezavi. Pišimo $s(A) := \sum_{x\in A} x$; potem je $\textit{centroid}(A) = s(A)/||s(A)||$. Varianca je zdaj enaka

$$\begin{aligned} \textit{var}(A) &= \frac{1}{|A|}\sum_{\mathbf{x}\in A}(1-\cos(\mathbf{x}, \textit{centroid}(A))) \\ &= 1 - \frac{1}{|A|}\sum_{\mathbf{x}\in A}\mathbf{x}^T s(A)/||s(A)|| \\ &= 1 - \frac{1}{|A|}s(A)^T s(A)/||s(A)|| \\ &= 1 - ||s(A)|| \,/\, |A|, \end{aligned}$$

AICS pa je enaka

$$\begin{aligned} \text{AICS}(A) &= \frac{1}{|A|(|A|-1)}\sum_{\mathbf{x},\mathbf{y}\in A,\mathbf{x}\neq\mathbf{y}}\cos(\mathbf{x},\mathbf{y}) \\ &= \frac{1}{|A|(|A|-1)}\left(\sum_{\mathbf{x},\mathbf{y}\in A}\mathbf{x}^T\mathbf{y} - |A|\right) \\ &= \frac{1}{|A|(|A|-1)}(s(A)^T s(A) - |A|) \\ &= -\frac{1}{|A|-1}(1 - ||s(A)||^2 \,/\, |A|). \end{aligned}$$

Tako smo razbitje P opisali z devetimi atributi. Vsakič, ko metoda voditeljev razbije neko skupino na dve manjši, se razbitje spremeni (eno od skupin v njem zamenjata dve manjši skupini) in za tako dobljeno novo razbitje lahko izračunamo novih devet atributov po enakem postopku. Z razbijanjem skupin smo nadaljevali, dokler ni nastalo razbitje z desetimi skupinami, tako da na koncu dobimo vektor 90 atributov. (Mogoče je tudi, da se postopek razbijanja skupin ustavi že pri manj kot desetih skupinah, če je že prej izpolnjen kakšen izmed ustavitvenih pogojev iz razdelka 4.3.2. V tem primeru po večkrat uporabimo deveterico atributov iz zadnjega tako dobljenega razbitja, tako da na koncu vendarle dobimo vektor 90 atributov.)

4.3.4 Učenje klasifikatorjev

Zdaj smo napovedovanje strukturnih sprememb formulirali kot učni problem; za učenje klasifikatorjev pri njem bomo uporabili metodo podpornih vektorjev (SVM) [15]. Ta metoda se je v zadnjih letih dobro obnesla na mnogih področjih, tudi v domenah z velikim številom atributov in učnih primerov. Model, ki ga zgradi metoda podpornih vektorjev, je oblike

$$f(\mathbf{x}) = b + \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}),$$

pri čemer x_i označuje i-ti učni primer; y_i je oznaka razreda, ki mu x_i pripada ($y_i = 1$ za pozitivne primere in $y_i = -1$ za negativne); funkcija K pa je jedro (*kernel*), ki si ga moramo izbrati pred začetkom učenja. Števila $b, \alpha_1, \ldots, \alpha_n$ poišče metoda podpornih vektorjev med učenjem. Vrednosti $f(x)$ lahko uporabimo za urejanje primerov — večja ko je $f(x)$, bolj verjetno je, da je primer x pozitiven. Konkretne binarne napovedi pa lahko dobimo, če si izberemo nek prag (ponavadi kar 0) in za pozitivne razglasimo tiste primere, pri katerih je $f(x)$ nad tem pragom, ostale pa za negativne.

Ker uporablja naša predstavitev razmeroma majhno število atributov („le“ 90, za razliko od npr. tipičnih predstavitev pri obdelavi besedil in slik, kjer je atributov pogosto več tisoč), smo se odločili uporabiti jedro, ki temelji na radialnih baznih funkcijah (RBF):

$$K(\mathbf{x}, \mathbf{x}_i) = e^{-\gamma||\mathbf{x}-\mathbf{x}_i||^2}.$$

Pri tem je $\gamma > 0$ parameter, ki si ga moramo izbrati pred začetkom učenja z metodo podpornih vektorjev. Če si predstavljamo to jedro kot funkcijo primera $\mathbf{x}$ pri konstantnem $\mathbf{x}_i$, nam K opisuje funkcijo zvonaste oblike z maksimumom ($K = 1$) v točki $\mathbf{x} = \mathbf{x}_i$, drugod pa vrednost funkcije z rastočo oddaljenostjo od $\mathbf{x}_i$ pada proti 0. Parameter γ uravnava širino zvona. Učinek tega jedra na obnašanje klasifikatorja (naučenega po metodi podpornih vektorjev) je, da vsak učni primer $\mathbf{x}_i$ glasuje za svoj razred y_i s težo, ki je odvisna od naučene vrednosti α_i in od oddaljenosti primera $\mathbf{x}$ (ki ga skušamo klasificirati) od učnega primera $\mathbf{x}_i$. SVM tako postane v nekem smislu izboljšana različica metode najbližjih sosedov, pri kateri je učni algoritem previdno izbral vpliv posameznih sosedov (prek uteži α_i). Pri takšnem razredu klasifikatorjev majhno število atributov ni nujno prehud problem, če imamo le dovolj učnih primerov.

4.4 Poskusi

4.4.1 Uporabljena domena

V tem razdelku opisujemo naše eksperimentalno ovrednotenje predlaganega pristopa za napovedovanje dodajanja kategorij v ontologijo. Uporabili smo poddrevo *Computers* iz ontologije ODP. Opazovali smo obdobje od januarja 2004 do oktobra 2006 (v tem času je na voljo po en posnetek stanja ontologije na mesec; za november in december 2006 pa tovrstnih posnetkov ni); v tem času se je poddrevo *Computers* povečalo od 7.732 na 8.309 kategorij, število dokumentov v njem pa se je v povprečju bolj zmanjševalo kot povečevalo in se je od 143.760 dokumentov v januarju 2004 zmanjšalo na 133.595 dokumentov v oktobru 2006.

V tem obdobju je bilo v omenjem delu ontologije dodanih 964 kategorij, 198 se jih je preimenovalo, 134 premaknilo, 153 pa je bilo raznih vrst zlivanj. Slika 4.1 prikazuje število različnih vrst operacij po posameznih mesecih.

Izmed na novo dodanih kategorij je bilo 482 primerov takih, pri katerih je bila nova kategorija dodana kot neposredna podkategorija neke že obstoječe kategorije in je iz tega svojega starša (ali pa njegovih že prej obstoječih potomcev v drevesu) prevzela več dokumentov, kot pa jih je dobila iz drugih delov ontologije. To je, kot smo opisali v razdelku 4.2.4, tista vrsta dodajanj kategorij, ki smo jo poskušali napovedovati. Vendar pa se izkaže, da je tudi pri teh dodajanjih število dokumentov, ki so se iz starša premaknili v (na novo dodanega) otroka, pogosto zelo majhno. Naš pristop k napovedovanju dodajanj temelji na hipotezi, da do (vsaj nekaterih) dodajanj pride zato, ker uredniki ontologije ODP opazijo v neki že obstoječi kategoriji skupino dokumentov, ki so tematsko povezani z nekim ožjim področjem, in se zato odločijo dodati za to področje novo podkategorijo in vanjo preseliti primerne dokumente. Zato ne moremo pričakovati, da se bo naš pristop obnesel v primerih, ko sta bila iz starševske kategorije v novo podkategorijo premaknjena le en ali dva dokumenta, saj v tem primeru v obstoječi starševski kategoriji očitno ni neke strnjene skupine dokumentov, ki bi tvorili jedro nove podkategorije; in če take skupine dokumentov ni, naš pristop ne bo imel podlage za napoved dodajanja nove podkategorije. Zato se bomo za potrebe definicije našega klasifikacijskega problema omejili na tiste primere dodajanj, pri katerih je nova kategorija prevzela vsaj neko minimalno število (v našem primeru pet) dokumentov iz svoje starševske kategorije (poleg njih pa lahko seveda nova kategorija dobi še dokumente iz drugih delov ontologije in dokumente, ki so zdaj šele prvič prišli v ontologijo; na število teh dokumentov ne postavljamo nobenih dodatnih omejitev). Tako nam ostane 107 primerov dodajanja nove kategorije, ki so temelj našega klasifikacijskega problema.

Naš napovedni model poskuša odgovoriti na vprašanje „ali naj se pod dano kategorijo doda v naslednjem mesecu dni kakšno novo podkategorijo?“. Ker sta možna dva odgovora, da in ne, gre torej za binarni oz. dvorazredni klasifikacijski problem. Kategorija C ob času t je z vidika tega problema pozitiven primer, če je v mesecu dni po času t (torej v obdobju do naslednjega datuma, za katerega imamo na voljo posnetek stanja ontologije) dobila kakšno podkategorijo, ki ustreza zgoraj opisanim zahtevam. V skladu s to definicijo nam iz prej omenjenih 107 dodajanj novih kategorij nastane 98 pozitivnih primerov (manj kot 107 jih je zato, ker včasih ista kategorija v istem mesecu dobi več primernih podkategorij).

Kdaj pa je kategorija negativen primer? Na primer, recimo, da neka kategorija C v mesecu marcu 2005 ni dobila nobene primerne podkategorije (kar ugotovimo s primerjavo posnetkov ontologije z začetka marca in z začetka aprila 2005), v aprilu 2005 pa je tako podkategorijo dobila (kar vidimo iz primerjave posnetkov ontologije z začetka aprila in začetka maja 2005). Za potrebe našega učnega problema je torej C v začetku aprila pozitiven učni primer; kaj pa naj rečemo o C v začetku marca? C se v marcu mogoče ni kaj dosti spremenila, kar bi pomenilo, da če je bila v začetku aprila primerna za to, da se ji doda novo podkategorijo, je bila za to skoraj

prav toliko primerna že tudi v začetku marca. Zato ni čisto upravičeno, da C v začetku marca razglasimo za negativen primer namesto za pozitivnega. Ontologijo ODP urejajo človeški uredniki, ki lahko kaj tudi spregledajo; oni bi mogoče dodali pod C primerno podkategorijo že v marcu, če bi že takrat opazili, da obstaja znotraj C neka manjša gruča dokumentov, ki je primerna za novo podkategorijo. Da torej ne bomo imeli preširoke množice negativnih primerov, smo se odločili definirati negativne primere takole: kategorijo C bomo v času t obravnavali kot negativni primer le, če ni dobila nobene primerne kategorije v treh mesecih pred in po času t. Kljub tej definiciji pa večina kategorij večino časa obvelja za negativne primere, saj je kategorij veliko, primernih dodajanj podkategorij pa razmeroma malo. Vsega skupaj smo iz poddrevesa *Computers* za obdobje 2004–2006 dobili čez 168.000 negativnih primerov. Da pospešimo eksperimente in da ne bodo pozitivni primeri čisto utopljeni v ogromni množici negativnih, smo za potrebe poskusov naključno izbrali manjšo podmnožico negativnih primerov, in sicer tako, da jih je trikrat toliko kot pozitivnih. Podatke iz let 2004 in 2005 smo uporabili za učenje, tiste iz leta 2006 pa za testiranje. Tako dobimo učno množico s 74 pozitivnimi in 222 negativnimi primeri ter testno množico z 24 pozitivnimi in 72 negativnimi primeri.

4.4.2 Zasnova poskusov

Kot smo omenili že v razdelku 4.3.4, smo za učenje klasifikatorjev uporabljali metodo podpornih vektorjev. Implementacija, ki smo jo uporabili, je iz programa SVMlight, ki ga je napisal Thorsten Joachims [34]. SVM poišče klasifikator tako, da reši optimizacijski problem, ki maksimizira kombinacijo dveh ciljev: klasifikator naj ima širok rob (*margin*) in naj naredi čim manj klasifikacijskih napak na učni množici. Ta dva cilja povežemo prek parametra za ceno napake, ki ga običajno označujemo s C. Večji ko je C, bolj se bo učni algoritem trudil najti klasifikator, ki naredi malo napak na učni množici; prevelik C pa lahko pripelje do pretiranega prilagajanja učnim podatkom in s tem do slabše generalizacije na še nevidene primere (npr. iz testne množice).

Pri domenah z razmeroma neuravnoteženo porazdelitvijo razredov, npr. takih, v katerih je negativnih primerov veliko več kot pozitivnih, je pogosto koristno, če obravnavamo napake na pozitivnih primerih kot bolj problematične od napak na negativnih primerih. Tako človek pravzaprav uporablja dve različni ceni napak: običajno ceno C za napake na negativnih primerih in nek njen večkratnik $j \cdot C$ za napake na pozitivnih primerih.

Poleg parametrov C in j moramo izbrati še parameter γ iz definicije radialnega jedra, ki smo ga uporabljali pri metodi podpornih vektorjev (razdelek 4.3.4). Ker je za te tri parametre težko vnaprej zanesljivo reči, katere vrednosti so najprimernejše, smo jih izbrali s petkratnim prečnim preverjanjem na učni množici. Preizkusili smo naslednje množne vrednosti: $C \in$

$\{0{,}1, 1, 10, 100, 1000\}$; $j \in \{1, 2, 3, 5, 10, 20, 50, 100\}$; in $\gamma \in \{0{,}0001, 0{,}0002, 0{,}0005, 0{,}001, 0{,}002, 0{,}005, 0{,}01, 0{,}02, 0{,}05, 0{,}1, 0{,}2, 0{,}5, 1, 2, 5\}$. Za večino kombinacij vrednosti teh treh parametrov je učenje modela trajalo manj kot sekundo, tako da je šla pri poskusih večina procesorskega časa pravzaprav za računanje vrednosti atributov (po postopku iz razdelkov 4.3.1, 4.3.2 in 4.3.3).

4.4.3 Evaluacijske mere

Napovedi naših klasifikatorjev smo ocenjevali z uveljavljenimi merami s področja poizvedovanja po podatkih (*information retrieval*) [62]: prelomno točko (*breakeven point*, BEP) in površino pod krivuljo ROC (*receiver operating characteristic*).

Klasifikator prek svojih napovedi $f(x)$ (kot smo jih videli npr. v razdelku 4.3.4) vpelje med primere (npr. tiste iz testne množice) nek vrstni red. Recimo, da bi si izbrali nek prag θ in za primere, ki ta prag presežejo (torej imajo $f(x) > \theta$), napovedali, da pripadajo pozitivnemu razredu, za ostale primere pa, da pripadajo negativnemu razredu. Glede na pravi in napovedani razred lahko razdelimo primere v štiri skupine in za vsako preštejemo, koliko primerov je v njej. Rezultate lahko uredimo v matriko, ki ji pravimo „kontingečna tabela“:

Pravi razred	Napovedani razred: Pozitivni	Napovedani razred: Negativni
Pozitivni	TP (pravi pozitivni)	FP (lažni pozitivni)
Negativni	FP (lažni pozitivni)	TN (pravi negativni)

Iz podatkov v kontingenčni tabeli lahko izračunamo med drugim naslednje koristne mere:

$$\begin{aligned} &\mathit{precision} = TP/(TP + FP) && \text{(natančnost)} \\ &\mathit{recall} = TP_{rate} = TP/(TP + FN) && \text{(priklic)} \\ &FP_{rate} = FP/(FP + TN) \end{aligned}$$

Natančnost je torej delež pravilnih napovedi med primeri, za katere je klasifikator napovedal pozitivni razred; priklic pa je delež pravilnih napovedi med vsemi primeri, ki res pripadajo pozitivnemu razredu. Ker gre pri priklicu za delež resnično pozitivnih (TP) glede na vse pozitivne ($TP+FN$), ga označujejo včasih tudi TP_{rate}, po analogiji z njim pa lahko zato definiramo tudi FP_{rate} kot delež lažnih pozitivnih (FP, ki so v resnici negativni) glede na vse negativne ($FP + TN$).

Če prag θ počasi znižujemo, dobi vse več primerov pozitivno napoved, zato se TP in FP povečujeta, TN in FN pa zmanjšujeta. Zato se priklic dviguje, natančnost pa se praviloma znižuje (čeprav ne nujno monotono). Nekje se torej njuni vrednosti srečata in to vrednost imenujemo „prelomna točka“ (BEP) [40]. BEP je koristna mera za ocenjevanje klasifikatorja, ker

nam pove nekaj o tem, kako lahko pri tem klasifikatorju uskladimo želje po čim višji natančnosti in želje po čim višjem priklicu (načeloma si namreč za obe meri želimo, da bi bili čim višji): če hočemo imeti natančnost nad BEP, bomo prisiljeni sprejeti priklic pod BEP in obratno. Večji ko je BEP, boljša je naša razvrstitev primerov (ki nam jo je s svojimi napovedmi priskrbel klasifikator), saj nas ne sili v tako neugodne kompromise kot razvrstitev z manjšim BEP.

Kot alternativo priklicu in natančnosti lahko uporabljamo TP_{rate} in FP_{rate}, torej delež pozitivno napovedanih primerov v vsakem od obeh razredov. Tidve meri pogosto narišemo na graf s FP_{rate} na vodoravni in TP_{rate} na navpični osi. Ko se prag θ znižuje, se vrednosti TP_{rate} in FP_{rate} obe spuščata, vse od $TP_{rate} = FP_{rate} = 0$ (če je prag tako visok, da nobenega primera ne napovemo kot pozitivnega) do $TP_{rate} = FP_{rate} = 1$ (če je prag tako nizek, da napovemo kot pozitivne kar vse primere). Tako nastane naraščajoča krivulja, znana tudi kot krivulja ROC (*receiver operating characteristic*) [57]. Ploščina lika pod to krivuljo (*area under ROC*, AuROC), torej med krivuljo in vodoravno osjo, je koristna mera za ocenjevanje vrstnega reda primerov (oz. klasifikatorja, na čigar napovedih ta vrstni red temelji); pokazati je mogoče, da je ta ploščina enaka verjetnosti, da za naključno izbran pozitiven primer x in naključno izbran negativen primer x' naš klasifikator pripiše prvemu višjo oceno kot drugemu (torej $f(x) > f(x')$. Popolnemu klasifikatorju bi to uspelo vsakič in bi imel ploščino pod krivuljo ROC enako 1. Ena od lepih lastnosti ploščine pod ROC je tudi ta, da ta mera ni odvisna od porazdelitve primerov med oba razreda (torej od tega, kolikšen delež primerov pripada pozitivnemu, kolikšen pa negativnemu razredu).

Za primerjavo z rezultati naših klasifikatorjev lahko razmislimo, kakšne ocene bi po tu opisanih merah dobil model, ki bi primere razvrstil v naključnem vrstnem redu brez kakršnekoli povezave z njihovim pravim razredom. Priklic je v tem primeru ne glede na višino praga ves čas približno enak deležu pozitivnih primerov glede na celotno testno množico, zato je takšna tudi vrednost BEP; v našem primeru je to 0,25. Ploščina pod krivuljo ROC pa je pri takem klasifikatorju enaka 1/2. Popoln klasifikator, ki bi oddal pri vseh primerih pravilno napoved, bi pri obeh merah (BEP in AuROC) dosegel vrednost 1.

4.4.4 Rezultati

Za ugotavljanje vpliva parametrov C, j in γ na obnašanje učnega algoritma smo uporabljali stratificirano 5-kratno prečno preverjanje na učni množici. Kot smo opisali v razdelku 4.4.2, smo opazovali 5 vrednosti parametra C, 8 vrednosti parametra j in 15 vrednosti parametra γ. Tako dobimo 600 kombinacij vseh treh parametrov. Izmed teh 600 smo izbrali tisto, ki doseže najboljši rezultat pri prečnem preverjanju na učni množici, nato pa smo s temi nastavitvami parametrov naučili še končni model na celotni učni

množici in ga preizkusili na testni množici. Rezultate prikazuje naslednja tabela:

Opis modela	Rezultat na učni množici pri preč. preverjanju		Rezultat na testni množici	
	BEP	AuROC	BEP	AuROC
Najvišji BEP pri preč. prev.	0,5148	0,7796	0,7083	0,8893
Najvišja AuROC pri preč. prev.	0,5021	0,7850	0,6667	0,8738
Najvišji BEP na testni mn.	0,4717	0,7436	0,7500	0,9011
Najvišja AuROC na testni mn.	0,4768	0,7495	0,7500	0,9155
Naključne napovedi	0,2500	0,5000	0,2500	0,5000

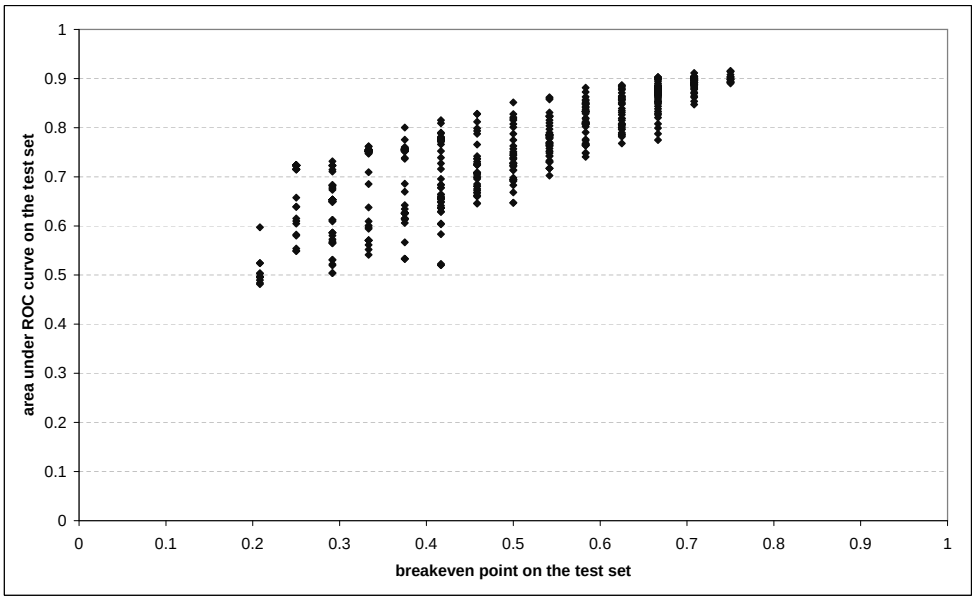
Rezultati v prvi vrstici se nanašajo na modele, naučene pri takšnih nastavitvah parametrov C, j in γ, ki so med prečnim preverjanjem dosegle najvišjo vrednost BEP. V tem primeru se je izkazalo, da so to najvišjo BEP dosegle tri različne kombinacije vrednosti parametrov, zato so v ostalih stolpcih (AuROC ter rezultati na testni množici) tu prikazana povprečja po vseh treh kombinacijah parametrov. Enak pristop smo uporabili tudi v preostanku tabele.

Vrstici, ki se nanašata na najvišjo BEP in AuROC na testni množici, kažeta, česa so načeloma sposobni najboljši izmed tu preizkušanih modelov, seveda s pridržkom, da brez uporabe testnih podatkov ne moremo vedeti, kateri modeli to so. Če primerjamo rezultate teh modelov s tistimi, ki jih lahko izberemo s prečnim preverjanjem na učni množici (prvi dve vrstici tabele), vidimo, da razlike niso zelo velike in da smo lahko s prečnim preverjanjem izbrali modele, ki se dobro obnesejo tudi na testni množici.

Za primerjavo kaže zadnja vrstica vrednosti BEP in AuROC, kakršne bi dosegel klasifikator, ki bi oddajal naključne napovedi. Te vrednosti lahko izpeljemo iz porazdelitve razredov v naših podatkih s pomočjo teoretičnega razmisleka s konca razdelka 4.4.3).

Kriteriji za izbor modela

Zanimivo vprašanje je, ali je vrednosti parametrov bolje izbrati tako, da dosežemo (pri prečnem preverjanju) čim večjo BEP, ali tako, da dosežemo čim večjo AuROC. Na sliki 4.3 je vsak izmed 600 modelov, ki jih dobimo pri različnih vrednostih parametrov v naših poskusih, predstavljen s točko, pri kateri smo za x-koordinato vzeli BEP tega modela, za y-koordinato pa njegovo AuROC. Kot vidimo, sta si obe meri precej tesno korelirani, še posebej pri najboljših modelih. Na problemih z razmeroma majhnim številom pozitivnih primerov (tak je tudi naš) ima BEP razmeroma majhno zalogo možnih vrednosti in zato kot mera za ocenjevanje na testni množici ni najbolj primerna.

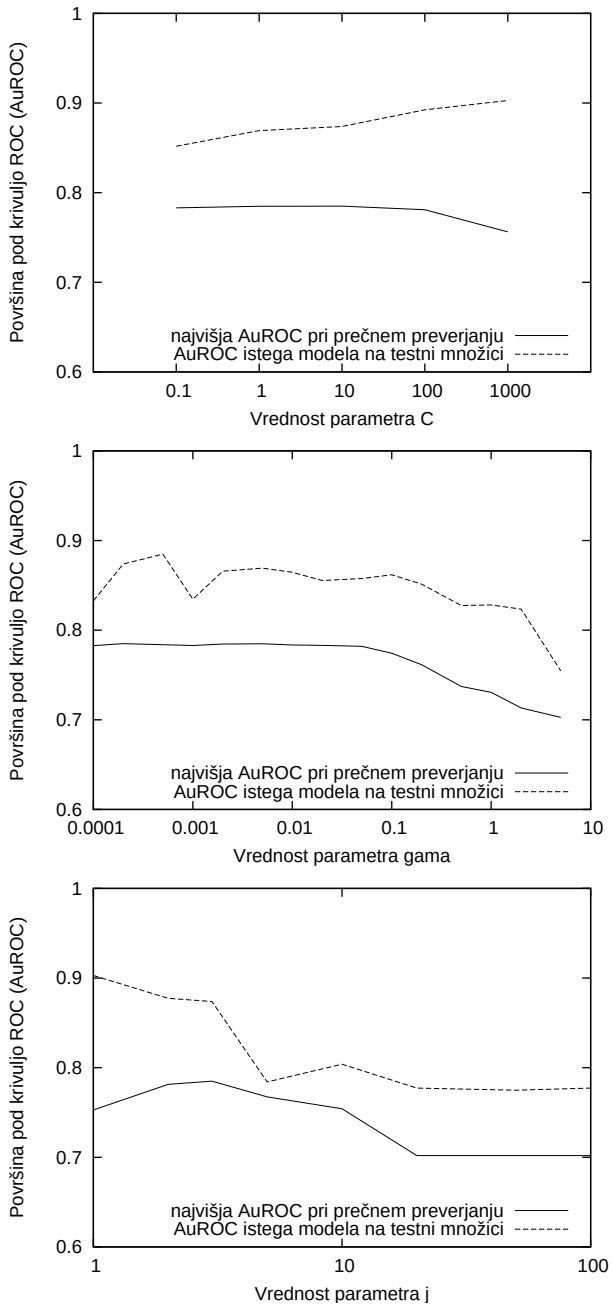


Slika 4.3: BEP in AuROC na testni množici za vseh 600 tu preizkušenih modelov (dobljenih pri različnih nastavitvah parametrov).

Vpliv posameznih parametrov

Doslej smo iskali najboljšo kombinacijo vseh treh parametrov, C, j in γ; zanimivo pa si je tudi ogledati, kakšen je vpliv vsakega parametra posebej. V tem razdelku opisujemo naslednji poskus: enega od teh treh parametrov fiksiramo na neko konkretno vrednost, nato pa ostala dva parametra postavimo na takšne vrednosti, da bo dosegel dobljeni model pri prečnem preverjanju na učni množici čim večjo AuROC. Na grafih na sliki 4.4 prikazujemo (kot funkcijo tistega parametra, čigar vrednost je bila fiksirana) to AuROC iz prečnega prevrejanja, pa tudi vrednost AuROC, ki jo isti nabor parametrov doseže na testni množici.

Pri parametru j so bili najboljši rezultati pri prečnem preverjanju doseženi pri $j = 3$. Ker je v naši učni (in testni) množici negativnih primerov trikrat več kot pozitivnih, se zdi smiselno domnevati, da bomo dobre rezultate dosegli, če napake na pozitivnih primerih obravnavamo kot trikrat resnejše od napak na negativnih primerih. To intuicijo deloma potrjujejo tudi naši poskusi, saj vidimo, da je predvsem neugodno, če je j prevelik (večji od 3), medtem ko po drugi strani zmanjševanje j-ja pod 3 ne poslabša rezultatov



Slika 4.4: najboljši rezultat, ki ga dosežemo, če je vrednost enega od parametrov učenja (C, γ ali j) vnaprej fiksirana in lahko izbiramo le vrednosti ostalih dveh parametrov.

na testni množici, ampak jih celo malo izboljša.

Opažanja o parametrih C in γ so v splošnem manj koristna, ker je pri učenju klasifikatorjev po metodi podpornih vektorjev najprimernejša vrednost teh parametrov močno odvisna od podatkov, s katerimi delamo, od števila atributov v vektorjih, od njihove zaloge vrednosti ipd. (to, kateri je najprimernejši C, pa je poleg tega odvisno tudi od izbora jedra in morebitnih parametrov v njem), zato izkušenj iz enega konkretnega nabora podatkov ne moremo preprosto prenesti na druge. Pri našem problemu se je izkazalo, da tako za C kot za γ obstaja precej širok interval, s katerega lahko zajamemo vrednosti teh dveh parametrov, ne da bi dobili klasifikator, ki bi bil veliko slabši od tistega pri najboljših vrednostih parametrov. Presenetljivo pri parametru C pa je, da nas prečno preverjanje po vsem videzu sodeč rahlo zavaja: večje vrednosti C so dajale pri prečnem preverjanju slabše rezultate, pri testiranju pa boljše.

4.5 Zaključki in možnosti za nadaljnje delo

V tem poglavju smo opisali pristop za napovedovanje nekaterih strukturnih sprememb v hierarhiji kot posebnem primeru ontologije, ki se omeji na relacijo *is-a*. Naš pristop poskuša napovedati dodajanje novih kategorij ob predpostavki, da je nova kategorija neposredna podkategorija neke že obstoječe kategorije in da se je iz slednje v novo kategorijo premaknilo nekaj dokumentov. Opisali smo, kako lahko to nalogo predstavimo kot problem strojnega učenja, in predstavili poskuse, ki kažejo, da je napovedovanje tovrstnih sprememb do neke mere možno.

S tem delom bi se dalo nadaljevati v različnih smereh. Koristno bi bilo razmisliti o še kakšnih atributih, s katerimi lahko opišemo kategorijo, poleg dosedanjih atributov. Naš dosedanji pristop poskuša poiskati v kategoriji že obstoječe dokumente, ki tvorijo razmeroma strnjene skupine, ne upošteva pa tega, da kakšna taka skupina mogoče ustreza kakšni od že obstoječih podkategorij in je torej mogoče ni pametno obravnavati kot znak, da je potrebna na tem mestu v hierarhiji še ena nova podkategorija.

Še ena zanimiva razširitev bi bila, da bi poskusili napovedovati ne le to, kdaj in kje bi bilo dobro dodati novo podkategorijo, pač pa tudi to, katere dokumente bi bilo dobro premakniti vanjo in s katerimi ključnimi besedami bi jo bilo dobro opisati. To bi bil lahko drugi korak po napovedovanju dodajanja nove podkategorije: ko bi dosedanji klasifikator priporočil, naj se na pod neko kategorijo C doda novo podkategorijo, bi lahko v nadaljevanju pregledali dokumente kategorije C, poiskali tiste, ki se ne ujemajo dovolj dobro z nobeno od že obstoječih C-jevih podkategorij, in te dokumente predlagali kot elemente nove podkategorije.

Poleg tega bi bilo zanimivo obravnavati tudi druge tipe sprememb v ontologiji. Doslej smo napovedovali dodajanje novih kategorij, pa še to le pod

določenimi pogoji. Med dodajanji, ki jih nismo pokrili, so na primer tista, pri katerih nova kategorija ne vsebuje (skoraj) nobenih takih dokumentov, ki so bili že od prej v ontologiji, pač pa vsebuje nekaj popolnoma novih dokumentov. Takšna dodajanja bi lahko poskušali napovedati, če bi si ogledali tudi zaporedje novih dokumentov, ki prihajajo v ontologijo v času od enega do naslednjega posnetka; med temi dokumenti bi skušali poiskati tiste, ki se ne prilegajo dovolj dobro nobeni od obstoječih kategorij (npr. s pomočjo klasifikatorjev za obstoječe kategorije); z razvrščanjem takšnih dokumentov v skupine bi lahko poskusili odkriti kandidate za nove kategorije.

Druga glavna skupina strukturnih sprememb (poleg dodajanj), ki smo jih opazili v ontologiji ODP, so premiki in zlivanja kategorij. Te bi lahko poskušali napovedovati tako, da obstoječe kategorije primerjamo med seboj (preprost in mogoče preveč naiven primer takega pristopa bi bil npr. računanje kosinusa kota med centroidi kategorij). S tem bi lahko ugotavljali, če se neka kategorija dovolj dobro prilega svoji sedanji nadkategoriji ali bi jo bilo bolje premakniti kam drugam v drevesu.

Še ena operacija, ki je aktualna vsaj za ontologijo ODP, pa je preimenovanje kategorij. To sicer ni zares strukturna sprememba, je pa v hierarhiji ODP razmeroma pogosta. Možen pristop za ta problem bi deloval na nivoju posameznih besed: vlogo posamezne besede v neki kategoriji lahko poskusimo opisati z množico atributov in nato naučiti nek napovedni model, ki bi iz takšnih atributov napovedoval, ali bi bila neka beseda primerna kot ena od ključnih besed v imenu kategorije. Atributi bi morali upoštevati ne le prisotnost oz. pogostost opazovane besede v dokumentih dane kategorije, pač pa tudi v dokumentih njenih otrok, sester in starša v drevesu — ime kategorije naj bi jo namreč pomagalo ločiti od vseh teh sosednjih kategorij. S takšnim modelom bi lahko za vse besede ocenili, kako primerne se zdijo za ključne besede pri dani kategoriji; če bi videli, da se v dosedanjem imenu kategorije pojavljajo besede z izrazito nizko oceno (oz. jim ta ocena skozi čas pada), bi lahko priporočili, naj se jih zamenja s kakšnimi drugimi besedami, ki dobijo v napovedih našega modela višjo oceno. Ima pa takšen pristop pomembno slabost: imena kategorij, sploh takih blizu vrha hierarhije, lahko vsebujejo kakšne bolj abstraktne besede, ki v konkretnih dokumentih iz te kategorije (in njenih podkategorij) niso nujno zelo pogoste. Mogoče bi bilo treba uporabiti WordNet ali kakšno podobno zbirko podatkov o nad- in podpomenkah, da bi odkrili povezave med takšnimi bolj abstraktnimi besedami in tistimi besedami, ki se dejansko pojavljajo v dokumentih.

O problemu napovedovanja strukturnih sprememb lahko govorimo tudi v semantično bogatejših ontologijah. Naš sedanji pristop predpostavlja, da je ontologija zelo preprosta, saj vsebuje le eno hierarhijo razredov (kategorij), v vsakem od njih pa je nič ali več instanc (dokumentov). Nekatere ontologije pa vsebujejo tudi druge relacije, atribute, logične stavke (ki opisujejo instance in razrede) in tako naprej. Pri takšni ontologiji je nabor možnih strukturnih sprememb precej večji, pa tudi napovedovati takšne spremembe

je veliko težje. Vprašljivo je, če lahko pristope, podobne temu, ki smo ga tukaj uporabili za napovedovanje dodajanja kategorij v preprosto hierarhijo, uporabimo tudi za napovedovanje strukturnih sprememb v semantično bogatejših ontologijah.

Poglavje 5

Mera za primerjanje ontologij

5.1 Uvod

V sodobnih informacijskih sistemih se poudarek marsikje pomika od klasične obdelave podatkov proti obdelavi „konceptov“, pri čemer torej osnovna enota obdelave ni več nek atomarni kos podatkov, pač pa bolj in bolj nek semantično opremljen koncept, ki mu pripada tudi neka interpretacija in ki obstaja v nekem kontekstu, torej v povezavah z drugimi koncepti. Ontologija je pri takšnih sistemih struktura, ki zajema znanje o nekem problemskem področju in hrani za to področje relevantne koncepte in relacije med njimi. Pri gradnji, vzdrževanju in uporabi ontologij ima pomembno vlogo tudi avtomatska analiza tekstovnih podatkov, sploh glede na naraščajočo priljubljenost polavtomatske gradnje ontologij (oz. učenja ontologij, kot jo tudi imenujejo). Za polavtomatsko gradnjo ontologij so predlagali že različne postopke odkrivanja zakonitosti v podatkih, med drugim [28]: nenadzorovano, pol-nadzorovano in nadzorovano učenje na zbirkah besedil; vizualizacijo besedil; uporabo tehnik obdelave naravnega jezika za gradnjo semantičnega grafa posameznega besedila; uporabo ekstrakcije informacij za odkrivanje relevantnih konceptov; in vizualizacijo konteksta posamezne imenske entitete v neki zbirki besedil.

Ontologije so temeljna podatkovna struktura za konceptualizacijo znanja o nekem področju, to pa je v praksi „mehko“ in ga je mogoče zajeti in izraziti na različne načine. Zato je v splošnem mogoče za neko problemsko področje zgraditi veliko različnih ontologij in koristno je, če lahko ugotovimo, katera od njih bolje ustreza nekemu vnaprej izbranemu kriteriju. Zato je evaluacija ontologij pomemben problem, o katerem moramo razmišljati, če hočemo, da se bodo ontologije na široko uveljavile na semantičnem spletu in v drugih semantično usmerjenih aplikacijah. Uporabniki, ki se znajdejo pred množico ontologij, morajo imeti možnost oceniti te ontologije in ugotoviti, katera naj-

bolje ustreza njihovim potrebam. Tudi ljudje, ki gradijo ontologijo, potrebujejo način, kako oceniti dobljeno ontologijo in s takšnimi ocenami mogoče tudi usmerjati sam postopek gradnje in izboljševanja ontologije. Še posebej pa efektivne mere za ocenjevanje ontologij potrebujejo avtomatski in polavtomatski postopki učenja ontologij, saj lahko s pomočjo takšne mere poiščejo „najboljšo“ izmed več možnih ontologij, nastavijo vrednosti parametrov kakšnega učnega algoritma ali pa usmerjajo postopek učenja, če je ta zastavljen kot proces preiskovanja nekega prostora možnosti.

V nadaljevanju tega poglavja najprej predstavljamo dosedanje delo na področju evaluacije ontologij (razdelek 5.2) z glavnimi pristopi k ocenjevanju ontologij in različnimi tehnikami za ocenjevanje različnih vidikov ali nivojev ontologije; v razdelku 5.3 kažemo, kako lahko ocenjevanje ontologije vključimo v enega od formalnih ogrodij za definicijo ontologije; v razdelku 5.4 opisujemo naš pristop za merjenje podobnosti med dvema hierarhičnima ontologijama; v razdelku 5.5 predstavljamo poskuse, ki ugotavljajo, kako naša mera podobnosti odreagira na različne spremembe v ontologiji, in to na primeru velike tematske ontologije iz prakse.

5.2 Pregled pristopov za evaluacijo ontologij

V literaturi so opisali že precej pristopov k evaluaciji ontologij, razlikujejo pa se po tem, kakšne vrste ontologij poskušajo ocenjevati in s kakšnim namenom. V grobem lahko večino teh pristopov razvrstimo na naslednje skupine:

- pristopi, ki primerjajo ontologijo z „zlatim standardom“, torej z nečim, kar za potrebe te primerjave obravnavamo kot „pravilno“ ali „idealno“ konceptualizacijo problemskega področja (zlati standard je lahko tudi sam podan v obliki ontologije) [43];

- pristopi, ki ontologijo uporabijo v neki aplikaciji in nato ocenijo izhode oz. rezultate te aplikacije [56];

- pristopi, pri katerih se uporablja nek dodaten vir podatkov (na primer zbirko besedil) o problemskem področju, ki naj bi ga ontologija pokrivala (*data-driven approaches*) [11];

- pristopi, pri katerih ontologijo ocenjujejo ljudje, ki gledajo, kako ontologija ustreza nekim vnaprej izbranim merilom, standardom, zahtevam ipd. [41].

Poleg gornje delitve pristopov k evaluaciji ontologij jih lahko delimo tudi na podlagi tega, na katerem nivoju ocenjujejo ontologijo, kot opisujemo v naslednjih podrazdelkih.

5.2.1 Evaluacija ontologij na različnih nivojih

Ontologija je precej kompleksna struktura in pogosto je bolj praktično, če se osredotočimo na evaluacijo ontologije na posameznih nivojih ali plasteh, ne pa da poskušamo nekako oceniti ontologijo kot celoto. To je še posebej pomembno v primerih, ko želimo evaluacijo izvajati čim bolj avtomatsko in s čim manj udeležbe človeških uporabnikov ali ekspertov. Še en razlog za evaluacijo na posameznih nivojih pa je, da če se pri gradnji ontologije uporablja tehnike strojnega učenja, so te tehnike praviloma precej različne za različne nivoje ontologije, zato je smiselno tudi evaluacijo obravnavati ločeno po posameznih nivojih. Nivoje ontologij so različni avtorji definirali različno [25, 26, 12, 56, 19], vendar pa so si te definicije v grobem podobne in običajno zajemajo naslednje nivoje:

- *Leksikalna, slovarska ali podatkovna plast.* Tu je poudarek na tem, kateri koncepti, instance, dejstva ipd. so bili vključeni v ontologijo in kakšno besedišče se uporablja za predstavitev ali označitev teh konceptov. Pri evaluaciji ontologije na tem nivoju se praviloma uporablja primerjave z raznimi viri podatkov o problemskem področju (npr. korpusi besedil o področju, ki ga ontologija pokriva), pa tudi mere podobnosti med nizi (npr. urejevalniška razdalja) in podobne tehnike.

- *Hierarhija ali taksonomija.* Ontologije zelo pogosto vsebujejo hierarhično relacijo „je“ (*is-a*, subsumpcija) med koncepti. Ontologija lahko sicer vsebuje tudi druge relacije, vendar je relacija is-a pogosto še posebej pomembna in se lahko evaluacija posebej osredotoči nanjo.

- *Druge semantične relacije.* Ontologija lahko poleg is-a vsebuje še druge relacije in te relacije lahko ocenjujemo posebej. Pri tem se pogosto uporabljajo mere s področja iskanja po tekstovnih podatkih, na primer natančnost in priklic.

- *Kontekst.*

  (1) Ontologija je lahko del neke večje zbirke ontologij in se lahko sklicuje na koncepte ali relacije, definirane v drugih ontologijah iz te zbirke, ali pa se one na podoben način sklicujejo na našo opazovano ontologijo. V tem primeru je lahko pomembno, da pri ocenjevanju ontologije upoštevamo tudi ta kontekst [77, 12, 54].

  (2) Še ena oblika konteksta pa je aplikacija, v okviru katere se ontologijo uporablja. Namesto da razmišljamo o tem, kako ocenjevati ontologijo samo po sebi, je lahko bolj praktično, da jo ocenjujemo v kontekstu neke konkretne aplikacije in opazujemo, kako uporaba te ali one ontologije vpliva na kakovost delovanja oz. izhodov neke aplikacije. Podobna možnost je, da se evaluacija namesto na neko posamezno aplikacijo osredotoča na zorni kot posameznih uporabnikov ali pa organizacije (npr. podjetja), ki bo ontologijo uporabljalo [21].

- *Sintaktična plast.* Evaluacija na tem nivoju je lahko še posebej zanimiva pri ontologijah, ki so bile pretežno sestavljene ročno. Takšna ontologija je običajno zapisana v nekem formalnem jeziku in mora ustrezati sintaktičnim pravilom tega jezika (uporabljati prave ključne besede ipd.). Pri evaluaciji je mogoče upoštevati tudi razna druga sintaktična merila, npr. prisotnost dokumentacije v naravnem jeziku, izogibanje krožnim referencam med definicijami ipd. [25]. Izmed vseh vidikov evaluacije ontologij je tega verjetno še najenostavneje avtomatizirati.

- *Struktura, arhitektura, design.* Za razliko od prvih treh plasti s tega seznama, pri katerih je poudarek na dejanski množici konceptov, instanc, relacij ipd. v opazovani ontologiji, se evaluacija na tej plasti osredotoča na načrtovalske odločitve, ki so bile sprejete v procesu razvoja ontologije. To je zanimivo predvsem pri ročno sestavljenih ontologijah. Če smo pred razvojem ontologije sprejeli določena načela ali merila glede snovanja in arhitekture ontologije, lahko na koncu ontologijo ocenjujemo po tem, kako dobro se ujema s temi načeli in merili. Med strukturne kriterije štejemo na primer organizacijo ontologije in njeno prikladnost za nadaljnji razvoj (npr. dodajanje novih konceptov, spreminjanje ali odstranjevanje obstoječih) [25, 26]. Za nekatere namene uporabe ontologij je pomembno tudi, da so formalne definicije in stavki v ontologiji primerno dokumentirani z opisi v naravnem jeziku, ta dokumentacija pa mora biti smiselna, koherentna, dovolj podrobna, aktualna in usklajena s formalnimi definicijami. Teh vidikov ontologije več ali manj ni mogoče ocenjevati drugače kot ročno, pri čemer sodelujejo ontološki inženirji ter strokovnjaki za problemsko področje, ki ga ontologija pokriva.

Tabela 5.1 kaže, katere vrste pristopov z začetka razdelka 5.2 se pogosto uporablja za katere izmed zgoraj navedenih nivojev. V naslednjih nekaj podrazdelkih podrobneje predstavljamo različne pristope in nivoje ocenjevanja ontologije.

5.2.2 Evaluacija na nivoju besedišča, konceptov in podatkov

Primer postopka za ocenjevanje ontologij na nivoju besedišča je naslednji pristop, ki sta ga predlagala Maedche in Staab [43]. Podobnost med dvema nizoma lahko merimo z Levenshteinovo urejevalniško razdaljo [39], ki jo še normaliziramo z dolžino nizov, tako da vrača vrednosti z intervala $[0, 1]$. (Namesto klasične urejevalniške razdalje lahko uporabimo tudi kakšno prilagojeno različico, ki upošteva posebne lastnosti problemskega področja, o katerem govori ontologija. Na primer, če imamo opravka z osebnimi imeni, lahko upoštevamo, da je ime pogosto predstavljeno le z začetno črko [19].) Dve množici nizov lahko zdaj primerjamo tako, da za vsak niz iz

Tabela 5.1: Pregled pristopov k ocenjevanju različnih nivojev ontologije.

Plast	Pristop k ocenjevanju ontologije			
	Zlati standard	S pomočjo aplikacije	S pomočjo podatkov	Človeški ocenjevalci
Leksika, slovar, koncepti, podatki	×	×	×	×
Hierarhija, taksonomija	×	×	×	×
Druge semantične relacije	×	×	×	×
Kontekst (zbirka ontologij, aplikacija)		×		×
Sintaksa	×[1]			×
Struktura, arhitektura, design				×

[1] „Zlati standard“ v tem primeru pomeni, da sintakso definicije ontologije primerjamo s specifikacijo sintakse formalnega jezika, v katerem je ontologija zapisana (npr. RDF, OWL ipd.).

prve množice poiščemo najpodobnejšega (merjeno z urejevalniško razdaljo) v drugi množici; za vsak tako dobljeni par si zapomnimo podobnost med njima in na koncu izračunamo povprečje. Za potrebe primerjanja dveh ontologij lahko kot množici nizov vzamemo imena vseh konceptov v eni in drugi ontologiji, lahko pa je ena množica nizov pridobljena iz ontologije, druga pa kako drugače, npr. s statistično obdelavo nekega korpusa besedil ali pa jo pripravijo strokovnjaki za problemsko področje; tako lahko opisani pristop uporabimo tudi za ocenjevanje ontologije prek primerjave z „zlatim standardom“.

Besedišče ontologije lahko ocenjujemo tudi s pomočjo mer s področja iskanja po tekstovnih podatkih, na primer natančnost in priklic. Natančnost (*precision*) v tem primeru pove, kolikšen delež leksikalnih elementov ontologije (npr. nizov, ki se uporabljajo za imena konceptov) se pojavlja tudi v zlatem standardu (delež računamo glede na celotno besedišče opazovane ontologije). Podobno pa priklic (*recall*) pove, kolikšen delež leksikalnih elementov iz zlatega standarda se pojavlja tudi v opazovani ontologiji. V svoji osnovni obliki sta lahko tidve meri neugodno togi, če ne dopuščata manjših odstopanj v zapisu besed in fraz (npr. uporabe vezajev v večbesednih imenih konceptov, ipd.). Še en način za fleksibilnejše primerjanje besedišča ontologije z zlatim standardom pa je, da vsako besedo predstavimo z množico, ki poleg te besede vsebuje še njene nadpomenke (iz WordNeta ali kakšnega drugega podobnega vira); namesto enakosti dveh leksikalnih enot lahko zdaj merimo prekrivanje med dvema tako dobljenima množicama (ki poleg osnovnega niza vsebujeta še nadpomenke) [11].

Takšne pristope bi lahko uporabili tudi za ocenjevanje besedišča ontologije na drugih plasteh, npr. nize, ki označujejo relacije ali instance.

Velardi in sod. [81] opisujejo pristop za ocenjevanje sistema za gradnjo ontologije, ki poskuša iz zbirke besedil v naravnem jeziku izluščiti koncepte (besede in fraze), pomembne za dano problemsko področje, nato pa poskusi s pomočjo WordNeta in iskanja po spletu nekatere od teh konceptov povezati

z relacijo is-a. Za večbesedne koncepte nato zgenerirajo opise v naravnem jeziku: "x y = a kind of y, ⟨definicija y⟩, related to the x, ⟨definicija x⟩", pri čemer je y ponavadi samostalnik, x pa je nek njegov prilastek (samostalnik ali pridevnik). Takšne opise so potem pregledovali strokovnjaki za problemsko področje ontologije in ugotavljali, ali je sistem za gradnjo ontologije izbral pravega izmed več možnih pomenov besed x in y. Dobra stran takšnega pristopa je, da strokovnjakom za problemsko področje ni treba pregledovati ontologije, zapisane v nekem formalnem jeziku, ki ga mogoče ne poznajo, pač pa imajo opravka z opisi fraz v bolj ali manj običajni angleščini. Slabost takšnega pristopa pa je, da zahteva razmeroma veliko ročnega pregledovanja.

5.2.3 Evaluacija taksonomskih in drugih semantičnih relacij

Brewster in sod. [11] so predlagali na podatkih temelječ pristop za ocenjevanje strukturnega ujemanja med ontologijo in korpusom besedil. (1) Na danem korpusu besedil s področja, ki naj bi ga pokrivala ontologija, uporabimo postopek EM za razvrščanje v skupine (clustering) in tako dobimo probabilistični model tematik, s pomočjo katerega lahko ugotavljamo, da je vsak dokument nastal kot mešanica več tematik v nekih znanih razmerjih. (2) Vsak koncept c naše ontologije predstavimo z množico izrazov, ki poleg imena koncepta vsebuje še njegove nadpomenke (iz WordNeta). (3) Probabilistični modeli iz točke (1) lahko za vsako tematiko ocenijo, kako dobro se ji prilega koncept c. (4) Zdaj lahko ugotavljamo, ali se vsak koncept dobro prilega vsaj eni od tematik, in tako dobimo pristop za ocenjevanje ontologije na leksikalnem nivoju. Lahko pa namesto tega ugotavljamo, ali so si koncepti, ki se prilegajo isti tematiki, tudi v naši ontologiji blizu skupaj, pri čemer to bližnjost merimo z relacijo is-a in mogoče tudi drugimi relacijami v ontologiji. Če to slednje drži, lahko zaključimo, da se struktura ontologije razmeroma dobro prilega strukturi skritih tematik, kot jih je za opazovano problemsko področje odkril algoritem EM. Slabost tega pristopa je, da je z njim težko upoštevati usmerjenost relacij; za dva koncepta c_1 in c_2 nam primerjava s probabilističnimi modeli, ki smo jih našli v koraku (1), mogoče lahko pove, da bi morala biti v neki zvezi, le težko pa nam pove, ali naj bi veljalo "c_1 is-a c_2" ali mogoče "c_2 is-a c_1" ali pa bi morala biti celo povezana s kakšno drugo relacijo, ne pa is-a.

Če imamo zlati standard, lahko relacijski nivo ontologije ocenjujemo tudi prek natančnosti in priklica. Spyns [73] je opisal pristop, ki iz besedila v naravnem jeziku avtomatsko izlušči množico „leksonov", torej trojic oblike $\langle pojem_1, vloga, pojem_2 \rangle$. Ta rezultat si lahko predstavljamo kot ontologijo, v kateri izrazi predstavljajo koncepte, vloge pa so (nehierarhične) relacije med njimi. Za ocenjevanje teh rezultatov je meril natančnost in priklic, ontologijo pa je primerjal bodisi z zlatim standardom (ki so ga pripravili ljudje ročno) ali pa s seznamom izrazov, ki naj bi bili relevantni glede na

neko statistično oceno.

Malo drugačen pogled na ocenjevanje ontologije sta opisala Guarino in Welty [30], ki navajata več filozofskih pojmov (nujnost, togost, enost, itd.), s pomočjo katerih lahko bolje razumemo naravo raznih vrst semantičnih relacij, ki utegnejo nastopati v ontologiji, in odkrijemo morebitne problematične odločitve v strukturi ontologije. Na primer, neka lastnost je „nujna“ za neko entiteto, če zanjo nujno vedno velja. Lastnost, ki je nujna za vse entitete, ki to lastnost imajo, je „toga“ (npr. „biti oseba“: nobene entitete ni, ki bi lahko bila oseba, vendar ni; vsaka entiteta, ki je oseba, je nujno vedno oseba). Lastnost, ki ne more biti nujna za nobeno entiteto, pa je „anti-rigidna“ (npr. „biti študent“: vsakdo, ki je študent, bi lahko tudi ne bil študent). Razred, ki je definiran na podlagi neke toge lastnosti, ne more biti podrazred razreda, ki je definiran na podlagi neke anti-rigidne lastnosti. Na podlagi tega opažanja lahko na primer ugotovimo, da če neka ontologija vsebuje koncept „oseba“ kot podrazred koncepa „študent“, je to napaka. S podobnimi razmisleki lahko odkrijemo še več drugih vrst zmotne uporabe relacije is-a (npr. ljudje jo včasih pomotoma uporabljajo za povezovanje sinonimov ali meronimov ali pa za navajanje metapodatkov o nekem konceptu). Slabost tega pristopa je, da zahteva precej ročnega dela; če drugega ne, mora strokovnjak, ki razume nujnost, togost in podobne pojme, označiti, kateri koncepti ontologije temeljijo na togih lastnostih ipd.; od tam naprej je potem mogoče vsaj nekatere vrste napak odkrivati avtomatsko. Primeri, v katerih je evaluacija tega tipa res potrebna in upravičuje njene stroške, so razmeroma redki [32]. Bili pa so tudi že poskusi, da bi označevnanje konceptov (z oznakami za togost ipd.) avtomatizirali [83].

Več mer za primerjavo dveh ontologij na nivoju relacij sta predlagala tudi Maedche in Staab [43]. V dani hierarhiji je „semantična kotopija“ koncepta c množica vseh njegovih nad- in podkonceptov. Če imamo dve hierarhiji H_1 in H_2, lahko nek niz t predstavlja v prvi hierarhiji nek koncept c_1, v drugi pa nek koncept c_2. Oglejmo si množici izrazov (termov), ki v H_1 (oz. H_2) predstavljajo koncepte iz semantične kotopije koncepta c_1 (oz. c_2). Prekrivanje med tema dvema množicama nam pove nekaj o tem, ali igra pojem t v obeh hierarhijah podobno vlogo. Izračunamo lahko povprečje tega po vseh pojmih t, ki se pojavljajo v naših dveh hierarhijah, in tako dobimo neko mero podobnosti med H_1 in H_2.

Na podoben način lahko primerjamo tudi druge relacije, ne le is-a. Recimo, da je R_1 neka dvomestna relacija v prvi ontologiji z domeno $d(R_1)$ in zalogo vrednosti $r(R_1)$; in podobno naj bo R_2 neka dvomestna relacija v drugi ontologiji. Domnevamo lahko, da sta si relaciji podobni, če je $d(R_1)$ podobna $d(R_2)$ in če je $r(R_1)$ podobna $r(R_2)$. Ker sta $d(R_1)$ in $d(R_2)$ preprosto dve množici konceptov, ju lahko primerjamo na podoben način kot v prejšnjem odstavku: poiščemo množico izrazov, ki v eni oz. drugi ontologiji nastopajo kot imena kakšnega od konceptov iz $d(R_1)$ oz. $d(R_2)$ ali kakšnega od njegovih hipernimov, nato pa izračunamo kakšno mero prekrivanja med

tema dvema množicama. Na analogen način lahko primerjamo tudi zalogi vrednosti $r(R_1)$ in $r(R_2)$. Če imamo v opazovanih ontologijah več takšnih parov relacij, lahko izmerimo podobnost za vsakega od njiju in uporabimo povprečje teh podobnosti kot neko splošno oceno podobnosti obeh ontologij na nivoju relacij.

5.2.4 Evaluacija s pomočjo konteksta

Včasih je ontologija del neke večje zbirke ontologij, ki se sklicujejo druga na drugo (npr. ena ontologija uporablja razred ali koncept, ki je deklariran v neki drugi ontologiji), na primer na spletu ali pa v knjižnici ontologij, ki jo vzdržuje neka inštitucija. V tem primeru torej ontologija obstaja v nekem kontekstu, ki ga lahko tudi uporabimo za evaluacijo te ontologije. Tako na primer iskalnik Swoogle [18] navzkrižne povezave med ontologijami uporabi tako, da definira graf in v tem grafu izračuna oceno vsake ontologije po podobnem postopku, kot je PageRank [53] za običajne spletne strani. Tako dobljeno mero („ontology mark“) uporablja Swoogle za razvrščanje rezultatov iskanja. Podoben pristop so uporabili Patel in sod. [54] pri portalu OntoKhoj. V obeh primerih obstaja pomembna razlika v primerjavi z običajnim PageRankom, namreč ta, da vseh „povezav“ ali navzkrižnih referenc med ontologijami ne obravnavamo kot enako pomembne. Na primer, če ena ontologija definira nov podrazred nekega razreda iz druge ontologije, je ta povezava med ontologijama mogoče pomembnejša, kot če bi prva ontologija uporabila razred iz druge ontologije le kot zalogo vrednosti neke svoje relacije.

Kontekst za evaluacijo ontologije lahko prispevajo tudi človeški eksperti; tako je na primer Supekar [77] predlagal, naj se ontologije dopolni z metapodatki o tem, po kakšnih načelih je bila zasnovana, kako se jo uporablja in kaj o njej menijo uporabniki. Primeren iskalnik bi lahko potem izvajal poizvedbe po teh metapodatkih in uporabniku pomagal pri odločitvi o tem, katero izmed mnogih ontologij v zbirki naj uporabi. Slabost tega pristopa je, da se tako pri pripravi metapodatkov o ontologijah kot pri ocenjevanju ontologij na osnovi teh metapodatkov opira skoraj povsem na ročno delo.

5.2.5 Evaluacija s pomočjo aplikacije

Ontologijo se običajno uporabi v neki aplikaciji oz. pri reševanju nekega problema. Rezultati te aplikacije oz. njen uspeh pri reševanju danega problema je lahko deloma odvisen tudi od ontologije, ki se uporablja v njej. Lahko bi torej rekli, da je dobra ontologija tista, ki pomaga opazovani aplikaciji doseči dobre rezultate pri reševanju danega problema. Ontologije bi torej načeloma lahko evaluirali preprosto tako, da jih vključimo v neko aplikacijo in ovrednotimo rezultate te aplikacije. To je lahko precej elegantno, če so rezultati takšne aplikacije nekaj, za kar že obstaja uveljavljen način ocenje-

vanja. Tako na primer Porzel in Malaka [56] opisujeta pristop, pri katerem se ontologijo in njene relacije (tako is-a kot druge) uporablja predvsem za ugotavljanje, kako tesno povezan je pomen dveh pojmov. To sta uporabljala pri prepoznavanju govora, kjer lahko nastopa več domnev o tem, kaj posamezna beseda v stavku zares pomeni; bolj verjetna naj bi bila tista domneva, pri kateri so si pomeni posameznih besed čim tesneje povezani. Ontologijo sta uporabljala za ocenjevanje razdalje med pomeni, s tem pa za odločanje o tem, za katero izmed več domnev o pomenu posameznih besed naj se sistem odloči. Ocenjevanje končnih rezultatov aplikacije je v tem primeru preprosto, saj je treba njene predlagane interpretacije stavkov zgolj primerjati s pravimi interpretacijami, ki so jih pripravili ljudje.

Takšen pristop se lahko elegantno izogne raznim zapletom v zvezi z evaluacijo ontologij in jih prevede na problem evaluacije rezultatov neke aplikacije, kar je pogosto preprosteje. Ima pa ta pristop k evaluaciji ontologij tudi več slabosti: (1) po takšni evaluaciji lahko sicer zaključimo, da je neka ontologija dobra ali slaba za reševanje določenega problema na določen način; ni pa očitno, ali velja ta ugotovitev tudi splošneje (če uporabimo ontologijo kako drugače ali pa za reševanje kakšnega drugega problema); (2) evaluacija je lahko premalo občutljiva, ker je ontologija mogoče le manjši del aplikacije in je tudi njen vpliv na kakovost rezultatov aplikacije mogoče razmeroma majhen; (3) če hočemo evaluirati več ontologij, si morajo biti med seboj dovolj kompatibilne, da jih lahko vse uporabimo v neki aplikaciji (oz. mora biti ta aplikacija dovolj fleksibilna, da lahko dela z vsemi temi ontologijami); kompatibilnost je potrebna glede formata, v katerem so ontologije zapisane, glede prisotnosti in imen semantičnih relacij in podobno. Če je treba za vsako ontologijo posebej prilagajati tudi aplikacijo, se lahko evaluacija ontologij hitro izkaže za nesprejemljivo drago.

5.2.6 Evaluacija s pomočjo podatkov

Ontologijo lahko ocenjujemo tudi tako, da jo primerjamo z že obstoječimi podatki (običajno neko zbirko besedil) o problemskem področju, ki ga ontologija pokriva. Tako so na primer Patel in soavtorji [54] predlagali pristop, ki ugotovi, ali se ontologija nanaša na neko tematiko, in klasificira ontologijo v nek katalog tematik: iz ontologije izluščijo tekstovne podatke (npr. imena konceptov in relacij ter druge koščke besedila v naravnem jeziku) in jih uporabijo kot vhod v klasifikacijski model (pridobljen s strojnim učenjem na zbirki besedil, ki se nanašajo na problemsko področje, ki nas zanima).

Še en na podatkih temelječi pristop pa so predlagali Brewster in sod. [11]. Iz zbirke besedil so najprej ekstrahirali množico izrazov (terminov), relevantnih za opazovano problemsko področje; to lahko naredimo npr. z latentno semantično analizo. Količina prekrivanja med tako dobljeno množico relevantnih izrazov ter množico izrazov, ki se pojavljajo v ontologiji (npr. kot imena konceptov), nam lahko pove nekaj o tem, kako dobro se onto-

logija prilega uporabljeni zbirki besedil. Pri tem lahko uporabljamo tudi uveljavljene mere, kot sta natančnost in priklic.

Pri obširnejših in semantično bogatejših ontologijah, ki vsebujejo veliko faktografskih podatkov (npr. Cyc), lahko zbirko besedil uporabimo tudi kot vir „dejstev“ o zunanjem svetu, ontologijo pa ocenjujemo po tem, kolikšen delež teh dejstev je mogoče izpeljati tudi iz podatkov, ki jih vsebuje ontologija.

5.2.7 Večkriterijski pristopi

Še ena družina pristopov k evaluaciji ontologij pa se osredotoča na vprašanje, kako izbrati dobro ontologijo (ali pa nek ožji izbor obetavnih ontologij) iz neke dane množice ontologij. Na to vprašanje lahko gledamo kot na klasičen odločitveni problem in si pri njem pomagamo s tehnikami, znanimi s področja sistemov za podporo odločanju. Pri teh pristopih običajno vnaprej definiramo nekaj odločitvenih kriterijev ali atributov; ontologijo ocenimo glede na vsakega od teh kriterijev in ji pripišemo številsko oceno. Vsakemu kriteriju vnaprej pripišemo tudi neko težo in na koncu za ontologijo izračunamo skupno oceno kot uteženo vsoto njenih ocen po posameznih kriterijih. Ta pristop se zgleduje po strategijah, s katerimi tudi v mnogih drugih situacijah izbiramo med več kandidati oz. možnostmi. Koristen je lahko predvsem v primerih, ko imamo pred sabo večje število ontologij, ki so vsaj potencialno relevantne za problemsko področje, ki nas zanima. Slabost tega pristopa je, da že večkrat omenjenih težav pri evaluaciji ontologij (npr. da v mnogih primerih zahteva preveč ročnega dela ali pa se zanaša na prisotnost „zlatega standarda“) v bistvu ni rešil, ampak jih je le odložil na fazo, v kateri ocenjujemo ontologijo z vidika posameznega kriterija.

Primer tovrstnega pristopa so predlagali Burton-Jones in sod. [12], ki so uporabili deset preprostih kriterijev: (1) sintaktično pravilnost (število sintaktičnih napak); (2) semantično bogastvo (kolikšen delelež sintaktičnih elementov, ki jih formalni jezik ontologije omogoča, ta ontologija tudi res izkorišča); (3) interpretabilnost (ali se izrazi, uporabljeni v ontologiji, pojavljajo tudi v WordNetu in jim torej lahko pripišemo pomen); (4) doslednost (koliko izmed konceptov v ontologiji je vpletenih v kakšno nedoslednost); (5) jasnost (ali imajo izrazi, uporabljeni v ontologiji, več pomenov v WordNetu in so torej dvoumni); (6) izčrpnost (število konceptov v ontologiji, relativno glede na druge ontologije v opazovani zbirki); (7) točnost (delež napačnih trditev v ontologiji); (8) relevantnost (koliko trditev v ontologiji uporablja le take sintaktične elemente, ki so za aplikacijo sprejemljivi); (9) avtoritativnost (koliko drugih ontologij uporablja koncepte iz te ontologije); (10) zgodovina (koliko so uporabniki dostopali do te ontologije v primerjavi z drugimi ontologijami iz opazovane zbirke).

Kot lahko vidimo iz tega seznama, obsega ta metodologija kriterije z večine nivojev, ki jih omenja razdelek 5.2.1. Njena slabost pa je, da nam

ne pomaga kaj dosti pri ugotavljanju, kako dobro se ontologija ujema z resničnim stanjem problemskega področja, ki nas zanima (in če se sploh nanaša na to problemsko področje, ne pa na kaj čisto drugega; za preverjanje tega bi morali vključiti še kakšne druge tehnike, npr. s področja klasifikacije besedil [54]). Z vidika točnosti (ki je eden od kriterijev v gornjem seznamu) je ontologijo težko oceniti drugače kot z ročnim pregledom trditev v njej. Dobra stran opisane metodologije pa je, da lahko ocene ontologije z vidika drugih uporabljenih kriterijev računamo ročno (se pa nekateri od njih zanašajo na domnevo, da je ontologija del neke knjižnice ontologij in da imamo o njej na voljo tudi nekatere metapodatke, npr. zgodovino dostopov do posameznih ontologij).

Malo drugačen nabor kriterijev so predlagali Fox in sod. [21], usmerjen pa je bolj k ročnemu vrednotenju in ocenjevanju ontologij. Njihovi kriteriji so med drugim funkcijska popolnost (ali vsebuje ontologija dovolj informacij za potrebe dane aplikacije), splošnost (ali je dovolj splošna, da bi si jo delilo več uporabnikov, inštitucij ipd.), učinkovitost (ali ontologija podpira učinkovito sklepanje), razumljivost (ali jo uporabniki razumejo), natančnost oz. granularnost (ali podpira več nivojev abstrakcije oz. podrobnosti), minimalnost (ali vsebuje le toliko konceptov, kolikor jih je res treba).

Spet drugi avtorji so predlagali še podrobnejši nabor 117 kriterijev [41] in ga organizirali v tronivojsko hierarhijo. Ti kriteriji pokrivajo razne vidike formalnega jezika, v katerem je ontologija opisana, vsebino ontologije (koncepte, relacije, taksonomijo, aksiome), metodologijo izdelave ontologije, stroške uporabe (strojna in programska oprema, licence itd.) in razpoložljivost orodij za delo z ontologijo. Mnogi od kriterijev so dovolj preprosti, da bi se dalo ontologijo glede teh kriterijev ocenjevati avtomatsko oz. vsaj z minimalno količino človeškega dela. Obstajajo tudi podobni zgodnejši poskusi z manjšim številom kriterijev.

5.3 Teoretično ogrodje za evaluacijo ontologij

V tem razdelku opisujemo, kako lahko evaluacijo ontologij vključimo v eno od možnih formalnih definicij ontologije.

Eno od možnih in dobro premišljenih formalnih definicij ontologije so na primer opisali Ehrig in soavtorji [19]. Po njih je ontologija (s podatkovnimi tipi) definirana kot matematična struktura $O = (C, T, R, A, I, V, \leq_C, \leq_T, \sigma_R, \sigma_A, \iota_C, \iota_T, \iota_R, \iota_A)$. V njej nastopajo (disjunktne) množice konceptov (C), tipov (T), relacij (R), atributov (A), instanc (I) in vrednosti (V). Relaciji delne urejenosti $\leq_C$ (na množici C) in $\leq_T$ (na množici T) določata hierarhijo konceptov in hierarhijo tipov. Funkcija $\sigma_R : R \to C \times C$ podaja za vsako relacijo omejitev glede tega, iz katerih konceptov smejo biti instance, ki so povezane s to relacijo. Podobno funkcija $\sigma_A : A \to C \times T$ pove, kateremu konceptu pripada nek atribut in kakšnega tipa so njegove

vrednosti. Na koncu so tu še instanciacijske funkcije $\iota_C : C \to 2^I$ (za vsak koncept navaja njegove instance), $\iota_T : T \to 2^V$ (za vsak tip navaja njegove možne vrednosti), $\iota_R : R \to 2^{I \times I}$ (za vsako relacijo navaja, katere instance so v tej relaciji) in $\iota_A : A \to 2^{I \times V}$ (za vsak atribut navaja, kakšno vrednost ima pri kateri instanci). Ta formalizacija ontologije seveda ni edina možna; primer še ene formalizacije, ki pa temelji na podobnih načelih, so predlagli na primer Bloehdorn in sod. [4].

Opisano formalizacijo je koristno za nekatere vrste ontologij še razširiti, npr. z atributi konceptov poleg doslej omenjenih atributov instanc. Atributi konceptov bi tvorili še neko novo množico A' s pripadajočo funkcijo $\sigma_{A'} : A' \to T$ (ki bi navajala tip posameznih atributov) in instanciacijsko funkcijo $\iota_{A'} : A' \to 2^{C \times V}$ (ki bi za vsak atribut povedala, kakšno vrednost ima pri posameznem konceptu). Vrednost takšnega atributa torej ni vezana na posamezno instanco nekega koncepta, pač pa na tisti koncept kot celoto. Kot bomo videli v nadaljevanju, je ta razširitev koristna pri nekaterih scenarijih evaluacije ontologij. Možne druge razširitve, ki pa bi bile v praksi manj zanimive, so relacije med koncepti (namesto med instancami), vpeljava metarazredov in vpeljava večmestnih relacij (poleg doslej uporabljenih binarnih).

Takšno formalno ogrodje je precej fleksibilno in lahko opiše razne pogosto uporabljane vrste ontologij:

- *Terminološke ontologije*, pri katerih so instance besede, koncepti pa pomeni besed. Primer takšne ontologije je WordNet (`http://www.cogsci.princeton.edu/~/wn/`. Atributi konceptov so na primer opisi pomena besede v naravnem jeziku, atributi instanc pa tekstovne predstavitve besed (z nizi znakov).

- *Tematske ontologije*, pri katerih so instance dokumenti, koncepti pa so tematike. Znani primeri takšnih ontologij so spletni imeniki, kot sta Open Directory (`http://www.dmoz.org/`) in Yahoo! (`http://dir.yahoo.com/`). Atributi konceptov so na primer ime in kratek opis tematike, atributi instanc pa naslov dokumenta, njegov opis, URL in končno tudi vsebina samega dokumenta (ki je za mnoge praktične namene lahko predstavljena z vektorjem uteži TF-IDF ali v kakšni podobni obliki).

- *Ontologije podatkovnih modelov*, v katerih so koncepti tabele neke podatkovne baze, instance pa so posamezni zapisi v tabelah. V tem primeru tipi in atributi iz zgoraj omenjene formalne definicije ontologije neposredno ustrezajo tipom in atributom (stolpcem v tabelah) iz podatkovne sheme v relacijski bazi.

Ocenjevanje ali evaluacijo ontologije lahko v to formalizacijo vključimo kot funkcijo, ki posamezni ontologiji O pripiše neko realno število, npr. z intervala $[0, 1]$. Vendar pa, kot smo videli že v razdelku 5.2, je ponavadi bolj

praktično, da se evaluacija osredotoči na posamezne komponente ontologije O (kar približno ustreza posameznim nivojem ali plastem iz razdelka 5.2.1). Rezultate evaluacije posameznih komponent je mogoče kasneje združiti v skupno oceno ontologije [19].

- Podatkovnih tipov in njihovih vrednosti (torej T, V, $\leq_T$ in ι_T) v evaluacijo običajno ne bi vključili, saj predstavljajo le podlago, na kateri je zgrajena ontologija.
- Evaluacija na leksikalnem oz. konceptnem nivoju se lahko osredotoči na C, I, ι_C in mogoče tudi na nekatere atribute iz ι_A.
- Evaluacija hierarhije konceptov (relacije is-a) bi se osredotočila na relacijo delne urejenosti $\leq_C$.
- Evaluacija drugih semantičnih relacij bi se ukvarjala z R, ι_R ter atributi konceptov in instanc.
- Zamisliti si je mogoče tudi evaluacijo, ki bi se ukvarjala s posameznimi atributi, na primer s tem, ali je bilo za nek koncept izbrano primerno ime v naravnem jeziku. Takšna evaluacija bi kot vhodne podatke vzela ι_C in atribute ter ugotavljala, če so atributi konceptov primerni glede na dano ι_C in dane atribute instanc.
- Evaluacijo s pomočjo aplikacije bi lahko vključili v ta formalizem tako, da aplikacijo definiramo kot funkcijo $A(D, O)$, ki vrača nek rezultat v odvisnosti od vhodnih podatkov D in ontologije O. Če vhodne podatke D fiksiramo, postane vsaka evaluacijska funkcija na izhodih aplikacije A v bistvu tudi evaluacijska funkcija ontologije O.
- Evaluacijo prek primerjave z zlatim standardom lahko v ta formalizem vključimo kot funkcijo dveh ontologij, ki računa neko obliko podobnosti ali razdalje med njima. Podobno lahko evaluacijo, temelječo na podatkih, predstavimo kot funkcijo $f(O, D)$, pri čemer so D podatki o problemskem področju, ki nas zanimajo; lahko bi jo celo zapisali probabilistično kot $P(O|D)$.

5.4 Indeks OntoRand

Razvili smo pristop k evaluaciji ontologij, ki je namenjen predvsem avtomatski evaluaciji tistih ontologij, ki vsebujejo poleg konceptov tudi instance. Uporabljeni pristop temelji na načelu zlatega standarda in se osredotoča na ugotavljanje, kako dobro se dana ontologija ujema z zlatim standardom glede razporeditve instanc med koncepte in oblikovanja hierarhije med samimi koncepti. Naš pristop je sicer podoben drugim pristopom k evaluaciji ontologij z zlatim standardom (razd. 5.2), od njih pa se razlikuje predvsem

po tem, da se opira predvsem na razporeditev instanc med koncepte, ne pa npr. na opise konceptov (in/ali instanc) v naravnem jeziku (kot to počnejo pristopi, ki temeljijo na urejevalniški razdalji med nizi [43]). Naš pristop o samih instancah in njihovi predstavitvi ne dela nobenih drugih predpostavk kot te, da se instance pač ločijo med sabo (vsaka ima svojo identiteto) in da tako ontologija, ki nas zanima, kot zlati standard uporabljata enako množico instanc.

5.4.1 Opis problema

Naš pristop smo preizkusili na konkretnem primeru evaluacije tematske ontologije, ki temelji na poddrevesu "Science" internetnega imenika ODP (Open Directory Project, `dmoz.org`). Imenik ODP je tematska ontologija, ki jo sestavlja hierarhija tematik ali kategorij, vsaka tematika pa lahko (poleg podkategorij) vsebuje tudi nič ali več povezav na spletne strani zunaj imenika. Koncepti te ontologije so torej tematike, instance pa so povezave na zunanje strani. Ob vsaki povezavi sta tudi naslov in kratek opis zunanje strani v naravnem jeziku, zato si lahko instanco predstavljamo kot kratek tekstovni dokument. Z vidika učenja ontologij je poleg problema klasifikacije novih dokumentov v že obstoječe tematike [51] zanimiv tudi problem gradnje ontologije od začetka, pri čemer je vnaprej dana neka množica instanc (dokumentov), ki bi jih radi uredili v ontologijo. V ODPjevem poddrevesu "Science" je na primer približno 100 000 dokumentov in zanimivo vprašanje je, ali jih lahko nek avtomatski postopek nenadzorovanega učenja dobro organizira v tematsko hierarhijo. To je pravzaprav problem hierarhičnega razvrščanja instanc v skupine ali gruče; pri tem nastane hierarhija skupin, ki približno ustrezajo nekakšnim tematikam in si lahko to hierarhijo zato predstavljamo tudi kot tematsko ontologijo. Ena od možnosti za ocenjevanje tako naučene hierarhije je, da jo poskušamo primerjati s prvotnim ODPjevim poddrevesom "Science" in ugotavljati, ali so dokumenti v obeh primerih organizirani oz. razporejeni podobno. Imenik ODP, ki so ga ročno pripravili človeški uredniki, nam je torej za potrebe te evaluacije rabil kot zlati standard.

Pri tem evaluacijskem problemu je vsaka instanca predstavljena s kratkim dokumentom v naravnem jeziku (sestavljata ga naslov in opis strani, kot sta navedena v imeniku ODP). Za koncepte v naučeni ontologiji pa ne predpostavljamo, da so opisani eksplicitno s kakšnimi termini, frazami ali drugimi tekstovnimi opisi. Izbor takšnega tekstovnega opisa za nek koncept je sicer tudi lahko zanimiv problem pri učenju ontologij, vendar pa se z njegovo evaluacijo tukaj nismo ukvarjali. Še en kriterij, na katerega moramo biti pri razvoju naše evaluacijske mere pozorni, pa je ta, da imamo opravka s precej velikim številom tako instanc kot konceptov, zato mora biti evaluacija povsem avtomatizirana in razumno hitra.

5.4.2 Mere podobnosti med razbitji

Naš pristop k evaluaciji ontologij se opira na podobnosti med tu opisanim problemom učenja ontologij in tradicionalnim nenadzorovanim učenjem z razvrščanjem v skupine (*clustering*). Pri razvrščanju v skupine je naš cilj razbiti množico primerov (instanc) v družino disjunktnih podmnožic. Raziskovalci na področju razvrščanja v skupine so predlagali razne tehnike za primerjanje dveh razbitij (particij) iste množice primerov in te tehnike lahko uporabimo za ocenjevanje rezultatov razvrščanja tako, da primerjamo avtomatsko dobljeno razbitje z drugim razbitjem, ki predstavlja naš zlati standard. Če razširimo kakšno od teh mer razdalje med tradicionalnimi „ploskimi“ razbitji na hierarhična razbitja, jih lahko uporabimo tudi za primerjavo naučene ontologije z zlatim standardom (ker bosta pri definiciji učenja ontologije iz razdelka 5.4.1 obe ontologiji pravzaprav hierarhični razbitji iste množice primerov).

Primer pogosto uporabljane mere podobnosti med dvema ploskima razbitjema iste množice je Randov indeks [59]. Naj bo $O = \{o_1, \ldots, o_n\}$ naša množica instanc (v tem razdelku bo O pomenila le množico instanc, ne cele ontologije kot v razdelku 5.3; za množico instanc uporabljamo oznako O namesto I, da ne bo zmede z uporabo črke i kot indeksom) in naj bosta $U = \{U_1, \ldots, U_m\}$ in $V = \{V_1, \ldots, V_k\}$ dve razbitji te množice (na disjunktne podmnožice, torje $O = \cup_{i=1}^{m} U_i = \cup_{j=1}^{k} V_j$ in $U_i \cap U_{i'} = \emptyset$ za vse $1 \leq i < i' \leq m$ in $V_j \cap V_{j'} = \emptyset$ za vse $1 \leq j < j' \leq k$). Ena od možnosti, kako primerjati razbitji U in V, je to, da pogledamo, kako pogosto se ujemata glede razporeditve primerov med množice v razbitju. Če dva primera, $o_i, o_j \in O$, pri razbitju U ležita v isti množici, pri razbitju V pa ne (ali pa obratno), je to znak neujemanja med razbitjema U in V; več ko je takšnih neujemanj, bolj sta si U in V različni. Randov indeks med razbitjema U in V je delež parov primerov, pri katerih do neujemanja ne pride, relativno glede na število vseh možnih parov (torej $n(n-1)/2$).

5.4.3 Mera podobnosti med ontologijami

Mero podobnosti med dvema ontologijama (zgrajenima nad isto množico primerov) lahko definiramo zelo elegantno, če definicijo Randovega indeksa prilagodimo takole. Za vsak $o \in O$ definirajmo $U(o)$ (oz. $V(o)$) kot tisto množico iz razbitja U (oz. V), ki vsebuje primer o. Naj bo $\delta_X(X_i, X_j)$ neka mera podobnosti med množicama X_i in X_j nekega razbitja X. Zdaj lahko definiramo Randov indeks za ontologije s takšno formulo:

$$\mathit{OntoRandIdx}(U,V) = 1 - \frac{\sum_{1 \leq i < j \leq n} |\delta_U(U(o_i), U(o_j)) - \delta_V(V(o_i), V(o_j))|}{n(n-1)/2}. \tag{5.1}$$

Če za mero podobnosti med množicami v razbitju vzamemo kar tradicionalno Kroneckerjevo δ-funkcijo, $\delta_U(U_i, U_j) = 1$ za $U_i = U_j$ in $\delta_U(U_i, U_j) = 0$

za $U_i \neq U_j$ (in analogno v razbitju V), je indeks OntoRand natančno enak običajnemu Randovemu indeksu. Takrat je namreč izraz v $|\ldots|$ v formuli (5.1) enak 1, če se U in V ne ujemata glede razporeditve primerov o_i in o_j, sicer pa je enak 0; vsota tega prek vseh i in j torej prešteje pare, pri katerih se U in V ne ujemata.

Če pa hočemo indeks OntoRand uporabiti za primerjavo dveh ontologij, moramo pri njem upoštevati hierarhično razporeditev konceptov. Pri navadnem Randovem indeksu je za posamezni par instanc pomembno zgolj to, ali sta (v nekem razbitju) pristala v isti podmnožici ali ne; pri primerjavi ontologij pa imamo namesto podmnožic (v ploskem razbitju množice O) koncepte, organizirane v drevesasto hierarhijo. Zato zdaj nista vsaka dva različna koncepta enako različna. Na primer, dva koncepta s skupnim nadkonceptom v hierarhiji sta si verjetno precej podobna, četudi ne gre za en in isti koncept; po drugi strani pa med dvema konceptoma, ki nimata nobenega skupnega prednika razen korena drevesa, verjetno res ni kakšne posebej tesne zveze. Če torej na primer ena ontologija nek par instanc uvrsti v isti koncept, druga pa tidve instanci uvrsti v dva različna koncepta s skupnim nadkonceptom, je to sicer neujemanje med obema ontologijama, vendar je to neujemanje majhno. Če pa bi druga ontologija uvrstila tisti dve instanci v dva koncepta, ki nimata nobenega skupnega prednika kot koren drevesa, bi bilo (za ta par instanc) neujemanje s prvo ontologijo veliko. Ta intuitivni razmislek smo zajeli s funkcijo $\delta_\bullet$, na katero se opira zgoraj omenjena definicija funkcije $\mathit{OntoRandIdx}(U, V)$. Za primerjavo ontologij torej naši funkciji δ_U in δ_V ne bosta vračali le 0 ali 1 (odvisno od tega, če sta dana koncepta enaka ali ne), pač pa morata vračati tudi druga realna števila z intervala $[0, 1]$, pri čemer večje vrednosti pomenijo, da sta si koncepta v hierarhiji bolj oddaljena.

Če v formuli (5.1) uporabimo različne definicije funkcij δ_U in δ_V, lahko dobimo celo družino mer za primerjavo ontologij (oz. za primerjavo ontologije z zlatim standardom za potrebe ocenjevanja avtomatsko zgrajenih ontologij, kot je bilo to opisano v razdelku 5.4.1). V nadaljevanju predlagamo dve konkretni družini funkcij δ_U in δ_V. Ker bo δ_V vedno definirana analogno funkciji δ_U (od katere se razlikuje le po tem, da se nanaša na hierarhijo V namesto na U), bomo v nadaljevanju govorili le o funkciji δ_U. Mimogrede tudi omenimo, da je za potrebe formule (5.1) pravzaprav vseeno, ali funkciji δ_U in δ_V merita podobnost ali razdaljo med koncepti, saj ta formula gleda le na to, ali sta si koncepta, ki vsebujete opazovani dve instanci, v obeh ontologijah enako podobna oz. enako oddaljena. Če namesto δ_U in δ_V uporabimo $-\delta_U$ in $-\delta_V$, se vrednost izraza $|\delta_U(\ldots) - \delta_V(\ldots)$ v (5.1) sploh ne spremeni.

Podobnost glede na skupne prednike

Ta definicija δ_U se zgleduje po pristopu, ki se včasih uporablja za ocenjevanje klasifikacije v hierarhije [49]. Za koncept U_i ontologije U naj bo $A(U_i)$

množica vseh prednikov tega koncepta, torej množica vseh konceptov, ki v drevesu ležijo na poti od korena do koncepta U_i (vključno z U_i samim). Če imata dva koncepta U_i in U_j na primer skupnega očeta v drevesu, bosta imeli množici $A(U_i)$ in $A(U_j)$ veliko skupnih elementov; po drugi strani pa, če U_i in U_j nimata nobenega skupnega prednika razen korena, bo tudi v preseku množic $A(U_i)$ in $A(U_j)$ le koren. Velikost tega preseka lahko torej uporabimo kot mero tega, kako tesno sorodna sta si oba koncepta.

$$\delta_U(U_i, U_j) = |A(U_i) \cap A(U_j)|/|A(U_i) \cup A(U_j)|. \tag{5.2}$$

Ta mera je torej Jaccardov koeficient množic $A(U_i)$ in $A(U_j)$ in ima tudi to lepo lastnost, da je uporabna tudi v primerih, ko hierarhija U ni drevo, ampak nek splošnejši usmerjen graf (niti acikličnost ni zares nujna). Če si mislimo v tem grafu povezave usmerjene od staršev proti otrokom, moramo množico $A(U_i)$ definirati preprosto kot množico vseh točk, iz katerih je dosegljiva točka U_i.

Podobnost glede na razdaljo v drevesu

Druga pot do primerne definicije funkcije δ_U pa je, da delamo neposredno z razdaljami med konceptoma U_i in U_j v drevesu U. Naj bo h globina najglobljega skupnega prednika konceptov U_i in U_j v drevesu, l pa naj bo razdalja med U_i in U_j (torej število korakov na poti od U_i do skupnega prednika ter od tam naprej do koncepta U_j). Če je l velik, je to znak, da si koncepta nista tesno sorodna; podobno, če je h majhen, je to znak, da koncepta nimata skupnih prednikov razen mogoče kakšnega zelo splošnega koncepta blizu korena, torej si U_i in U_j spet nista tesno sorodna. Tidve intuiciji je mogoče na različne načine zajeti v formulo, ki definira δ_U kot funkcijo vrednosti l in h. Tako so na primer Rada in sod. [58] predlagali mero razdalje takšne oblike:

$$\delta(l, h) = e^{-\alpha l} \tanh(\beta h) \tag{5.3}$$

Tu sta α in β dve nenegativni konstanti, tanh pa je hiperbolični tangens

$$\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x}) = 1 - 2/(1 + e^{2x}).$$

Če je torej h majhen, je $\tanh(\beta h)$ blizu 0, pri vse večjih h pa se $\tanh(\beta h)$ približuje 1. Primer, ko sta koncepta enaka, torej $U_i = U_j$ in zato $l = 0$, je koristno obravnavati posebej in zanj definirati $\delta(0, h) = 1$, tako da razdalja $\delta_U(U_i, U_i)$ ne bo odvisna od globine koncepta U_i v drevesu.

Mimogrede, če nastavimo α na 0 (ali blizu 0) in β na neko veliko vrednost, bo $\delta(l, h)$ približno enaka 0 za $h = 0$ in približno enaka 1 za $h > 0$. V vsoti (5.1), s katero smo definirali indeks OntoRand, vsak par instanc prispeva vrednost (približno) 1, če imata njuna koncepta v eni ontologiji še kakšnega skupnega prednika razen korena, v drugi pa ne; sicer pa tak par

instanc pripeva v vsoto vrednost blizu 0. V tem primeru torej naš indeks OntoRand postane več ali manj enakovreden običajnemu Randovemu indeksu, če le-tega računamo na razbitju množice O, ki ga definirajo koncepti en nivo pod korenom. To lahko vzamemo kot svarilo, da vrednost α ne sme biti premajhna in β ne prevelika, drugače bo indeks OntoRand ignoriral globlje ležeče dele hierarhij.

Tudi mero razdalje (5.2), ki temelji na prekrivanju, lahko definiramo s pomočjo h in l. Če za koren vzamemo, da je na globini 0, sledi, da presek $A(U_i) \cap A(U_j)$ vsebuje $h + 1$ konceptov, unija $A(U_i) \cup A(U_j)$ pa vsebuje $h + l + 1$ konceptov. Formula (5.2) je tedaj enakovredna definiciji

$$\delta(l, h) = (h + 1)/(h + l + 1). \tag{5.4}$$

Če zdaj primerjamo formuli (5.3) in (5.4), opazimo zanimivo razliko med obema definicijama funkcije δ. Pri $h = 0$, torej če koncepta opazovanih dveh instanc nimata nobenega skupnega prednika razen korena drevesa, vrača formula (5.3) vrednost $\delta = 0$, formula (5.4) pa $\delta = 1/(l + 1) > 0$. Ko primerjamo dve ontologiji, se bo najbrž pri marsikakšnem paru instanc zgodilo, da ležita tidve instanci tako v eni kot v drugi ontologiji v konceptih, ki razen korena nimata skupnih prednikov (torej je za tidve instanci $h_U = h_V = 0$), razdalja med konceptoma pa je v eni in v drugi ontologiji lahko različna (torej $l_U \neq l_V$). V takih primerih bi formula (5.3) dala rezultat $\delta_U = \delta_V = 0$, po formuli (5.4) pa bi bili δ_U in δ_V različni (ker sta l_U in l_V različna). Ko s tako dobljenimi vrednostmi računamo (za različne pare instanc) $|\delta_U - \delta_V|$ in jih seštevamo (v formuli (5.1)), bodo pri uporabi (5.3) mnogi členi vsote enaki 0 in dobljena vrednost *OntoRandIdx* bo zato blizu 1. Na primer, pri naših poskusih s poddrevesom "Science" ontologije ODP (razdelek 5.5.3), četudi sta bili v dveh opazovanih ontologijah razporeditvi instanc med koncepte precej drugačni, je približno 81 % parov instanc imelo $h_U = h_V = 0$ (le 3,2 % od njih pa je imelo poleg tega tudi $l_U = l_V$). Če za funkcijo δ uporabimo definicijo (5.3), se moramo sprijazniti s tem, da bo večina členov v vsoti (5.1) enakih 0 in da bo zato indeks OntoRand blizu 1. To ne pomeni, da dobljene vrednosti indeksa OntoRand niso uporabne za ugotavljanje tega, ali je neka ontologija zlatemu standardu bližja kot neka druga, je pa vseeno lahko za uporabnika moteče, da so vrednosti OntoRand vedno tako blizu 1. V tem primeru je možna alternativa definicije (5.3) takšna:

$$\delta(l, h) = e^{-\alpha l \tanh(\beta h + 1)} \tag{5.5}$$

Družino δ-funkcij, ki jih določa formula (5.5), si lahko predstavljamo kot nekakšno posplošitev funkcije δ iz formule (5.4). Na primer, primerjali smo vrednosti δ po obeh definicijah za 106 naključno izbranih parov dokumentov iz poddrevesa "Science" v ontologiji ODP. Pri primerno izbranih vrednostih α in β lahko definicija (5.5) vrača vrednosti δ, ki so zelo tesno korelirane s tistimi, ki jih vrača definicija (5.4) (npr. pri $\alpha = 0{,}15$ in $\beta = 0{,}25$ je bil

korelacijski koeficient 0,995). Podobno, če gledamo $|\delta_U - \delta_V|$ za različne pare dokumentov (takšne vrednosti bomo morali seštevati v formuli (5.1)), se izkaže, da lahko z definicijo (5.5) dobimo vrednosti, tesno korelirane s tistimi po definiciji (5.4) (npr. korelacijski koeficient 0,981 pri $\alpha = 0{,}8$ in $\beta = 1{,}5$). Opozoriti pa velja na to, da če so si vrednosti δ po definicijah (5.5) in (5.4) tesno korelirane za neki vrednosti α in β, to še ne pomeni, da si bodo pri istih α in β tesno korelirane tudi razlike $|\delta_U - \delta_V|$ (in obratno).

Ena od slabosti definicij (5.3) in (5.5) je vsekakor ta, da si moramo pri njej izbrati uteži α in β); če funkcijo δ_U definiramo na podlagi prekrivanja, torej po formulah (5.2) ali (5.4), se tovrstnemu izbiranju parametrov izognemo.

Nadaljnje posplošitve. Mero (5.3) lahko še posplošimo in si mislimo družino funkcij oblike $\delta(l, h) = f(l)g(h)$, pri čemer je f padajoča funkcija l-ja, g pa je naraščajoča funkcija h-ja. Ker sta vrednosti l in h vedno celi števili in sta omejeni z globino drevesa (oz. l z dvakratno globino drevesa), bi lahko f in g (ali pa kar $\delta(l, h)$ sámo) definirali celo s tabelo, ki bi podajala njuni vrednosti za vse l oz. h.

Omenimo lahko še to, da si lahko glavni del formule (5.1) za indeks OntoRand, torej vsoto $\sum_{1 \le i < j \le n} |\delta_U(U(o_i), U(o_j)) - \delta_V(V(o_i), V(o_j))|$, razlagamo kot manhattansko razdaljo (L_1-normo) med dvema vektorjema s po $n(n-1)/2$ komponentami, pri čemer je eden od teh dveh vektorjev odvisen le od ontologije U, drugi pa le od V. Tako smo pravzaprav predstavili ontologijo U (in na podoben način V) z „vektorjem značilk“, v katerem (i, j)-ta komponenta z vrednostjo $\delta_U(U(o_i), U(o_j))$ pove, kako blizu skupaj ležita v tej ontologiji instanci o_i in o_j. Ta interpretacija odpira pot do še nadaljnjih posplošitev, npr. z uporabo evklidske razdalje namesto manhattanske ali pa celo z uporabo jeder [33]; vendar pa se tu s tovrstnimi posplošitvami nismo ukvarjali.

5.4.4 Aproksimacijski algoritmi

Kot lahko vidimo iz formule (5.1), je za izračun naše mere podobnosti načeloma potrebna vsota po vseh parih dokumentov, torej po vseh (i, j) za $1 \le i < j \le n$. Ta kvadratna časovna zahtevnost je lahko neugodna, če hočemo primerjati ontologije z večjim številom dokumentov (npr. reda velikosti 100 000 dokumentov, kolikor jih je v poddrevesu “Science” ontologije ODP; gl. razd. 5.4.1). Ena od možnosti, kako pospešiti izračun naše mere podobnosti, je, da namesto računanja vsote po vseh parih dokumentov uporabimo naključno izbrano množico parov, kar bo pripeljalo do približne vrednosti naše mere podobnosti. V formuli (5.1) bi torej uporabili povprečje vsote $|\delta_U(U(o_i), U(o_j)) - \delta_V(V(o_i), V(o_j))|$ po samo nekaterih parih (i, j) namesto po vseh takih parih.

Še en pristop k približnemu izračunu naše mere podobnosti pa je, da

poskušamo poiskati pare (i, j), pri katerih razlika

$$|\delta_U(U(o_i), U(o_j)) - \delta_V(V(o_i), V(o_j))|$$

ni blizu 0. Če namreč obe ontologiji uvrstita instanci o_i in o_j v dva zelo nesorodna koncepta, bosta vrednosti $\delta_U(U(o_i), U(o_j))$ in $\delta_V(V(o_i), V(o_j))$ obe blizu 0, torej bo tudi njuna razlika blizu 0 in na vsoto (teh razlik po vseh (i, j)) ne bo preveč vplivala. (V ontologiji, kot je ODP, lahko pričakujemo, da bo ta pogoj izpolnjen pri velikem deležu parov (i, j). Recimo na primer, da δ_U definiramo po formuli (5.3). Če ležita dve instanci v konceptih, ki nimata nobenega skupnega prednika razen korena, bo $h = 0$ in zato tudi $\delta_U = 0$. Če se to zgodi tudi v drugi ontologiji, bo razlika $|\delta_U - \delta_V|$ za ta dva dokumenta 0.) Zato bi bilo koristno, če lahko na kakšen cenejši način (kot s pregledom vseh parov) poiščemo tiste pare (i, j), za katere dokumenta o_i in o_j v vsaj eni od opazovanih ontologij ležita v tesno sorodnih konceptih; za te pare bi razliko $|\delta_U - \delta_V|$ izračunali natančno, po definiciji, prispevek ostalih parov k vsoti $\sum_{i,j} |\delta_U - \delta_V|$ pa bi bodisi ignorirali bodisi ocenili z vzorčenjem po neki podmnožici teh parov (kot je opisano v prejšnjem odstavku). Recimo na primer, da je δ_U definirana po formuli (5.4) kot $\delta(l, h) = (h+1)/(h+l+1)$. Tedaj bi radi torej našli pare konceptov, za katere je $(h + 1)/(h + l + 1)$ nad nekim vnaprej izbranim pragom ε. (Potem bi vedeli, da je podrobna obdelava koristna za tiste pare instanc, ki padejo v kakšnega od teh parov konceptov.) Pogoj $(h+1)/(h+l+1) > \varepsilon$ lahko zapišemo kot $l < (h+1)(1/\varepsilon-1)$. Primerne pare konceptov lahko zdaj iščemo s takšnim algoritmom:

Inicializiraj $P := \{\}$.
Za vsak koncept c:
  Naj bo h globina c-ja v drevesu in naj bo $L = \lfloor (h + 1)(1/\varepsilon - 1) \rfloor$.
  Označimo c-jeve otroke (neposredne podkoncepte) s $c_1, \ldots, c_r$.
  Za vsak l od 1 do L, za vsak i od 1 do r, naj bo $S_{l,i}$ množica
    tistih podkonceptov c-ja, ki so tudi podkoncepti c_i-ja
    in so v drevesu l nivojev pod c-jem.
  Za vsak l od 1 do L, za vsak i od 1 do r,
    dodaj v P vse pare iz $S_{l,i} \times (\cup_{l' \leq L-1} \cup_{i' \neq i} S_{l',i'})$.

V vsaki iteraciji zunanje zanke si algoritem ogleda nek koncept c in poišče vse take pare konceptov c', c'', pri katerih je c najgloblji skupni prednik konceptov c' in c'' in velja $\delta_U(c', c'') > \varepsilon$. Za učinkovitejše delo z množicami $S_{l,i}$ je koristno obdelovati koncepte c od globljih nivojev hierarhije proti vrhu, saj lahko množice $S_{l,i}$ za nadkoncept dobimo z zlivanjem primernih množic za njegove otroke.

5.5 Poskusi

Predlagani pristop za primerjanje ontologij (oz. za evaluacijo ontologije na podlagi primerjave z zlatim standardom) smo preizkusili tako, da smo opa-

zovali, kako naša mera podobnosti med ontologijami odreagira na nekatere dobro definirane razlike med dvema ontologijama. Namesto da bi iz množice dokumentov zgradili ontologijo in jo nato ocenjevali, smo vzeli zlati standard in ga na določen način preoblikovali oz. pokvarili, nato pa tako dobljeno ontologijo primerjali s prvotnim zlatim standardom. Definirali smo več preprostih in intuitivnih operacij, s katerimi lahko pokvarimo prvotno ontologijo (zlati standard); njihov namen je ilustrirati primere neujemanj, do katerih lahko pride med naučeno ontologijo in zlatim standardom, in ugotoviti, kako takšna neujemanja vplivajo na vrednost indeksa OntoRand. V naših poskusih smo uporabljali naslednje operacije:

- Brisanje spodnjih nivojev ontologije (pobrišemo torej vse koncepte pod določeno globino v drevesu konceptov; gl. razd. 5.5.1).
- Zamenjava koncepta in njegovega starša (neposrednega nadkoncepta; gl. razd. 5.5.2).
- Prerazporeditev instanc (dokumentov) med koncepte na podlagi tekstovne vsebine teh dokumentov (razd. 5.5.3).

Za poskuse smo uporabljali ontologijo ODP z dne 21. oktobra 2005. Vsebuje 687 333 konceptov in 4 381 225 instanc. Koncepti so organizirani v drevo, ki je v najglobljih predelih sicer globoko do 15 nivojev, večinoma pa je plitvejše (85 % vseh konceptov je na nivojih od 5 do 9, povprečna globina koncepta pa je 7,13). Ker bi bilo računati indeks OntoRand prek vseh parov instanc preveč zamudno (vseh parov je namreč približno $9{,}6 \cdot 10^{12}$), smo uporabili naključno izbranih 10^6 parov instanc.[1]

Če uporabimo mero podobnosti (5.3), ki temelji na razdalji v drevesu med dvema konceptoma, moramo izbrati vrednosti parametrov α in β. Spomnimo se, da se α uporablja v členu $e^{-\alpha l}$, pri čemer je l dolžina poti med konceptoma. Ker ima naša hierarhija 15 nivojev, vemo, da bo za vsak par konceptov veljalo $l \leq 28$; ker pa je večina konceptov na globini od 5 do 9, lahko pričakujemo, da bo za tipičen naključno izbran par konceptov dolžina poti l nekje od 10 do 15. Zato smo se odločili uporabiti $\alpha = 0{,}3$, kar nam da vrednosti $e^{-\alpha l}$ od 0,74 (pri $l = 1$) prek 0,22 (pri $l = 5$) do 0,05 (pri $l = 10$) in 0,01 (pri $l = 15$).

S podobnim razmislekom lahko izberemo tudi parameter β. Mera ga uporablja v členu $\tanh(\beta h)$, pri čemer je h globina, na kateri leži v drevesu

[1] Za oceno stabilnosti smo za nekaj poskusov iz razdelka 5.5.1 izračunali indeks OntoRand 30-krat, vsakič z drugim naključnim izborom 10^6 parov instanc; nato smo izračunali standardno deviacijo tako dobljenih vrednosti indeksa OntoRand. Pri poskusu, kjer smo obdržali zgornje 4 nivoje hierarhije, je bila standardna deviacija pri meri s prekrivanjem $1{,}1 \cdot 10^{-4}$, pri meri z razdaljo v drevesu pa $1{,}3 \cdot 10^{-4}$. Pri poskusu, kjer smo obdržali zgornjih 10 nivojev hierarhije, sta bili tidve standardni deviaciji $4{,}0 \cdot 10^{-6}$ in $3{,}1 \cdot 10^{-6}$. V vsakem primeru torej vidimo, da je standardna deviacija zelo majhna v primerjavi z vrednostjo $1 - \mathit{OntoRandIdx}$, torej je 10^6 naključnih parov primerno velik vzorec za približen izračun indeksa OntoRand.

najgloblji skupni prednik opazovanih dveh konceptov. V našem primeru bo torej h med 0 in 14; za dva naključno izbrana nesorodna koncepta bo h blizu 0. Za dva koncepta, ki sta si v drevesu zelo blizu skupaj, pa bo h le malo manjši od globine, na kateri ležita, torej bo v povprečju (kot smo videli zgoraj) okoli 7. Uporabili smo vrednost $\beta = 0{,}4$, kar nam za $\tanh(\beta h)$ da vrednosti od 0 (pri $h = 0$) in 0,20 (pri $h = 1$) do 0,76 (pri $h = 5$), 0,89 (pri $h = 7$) in 0,96 (pri $h = 10$).

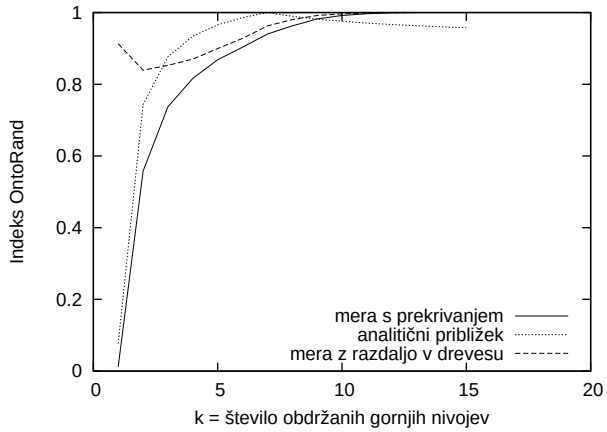
V splošnem je izbira vrednosti α in β odvisna od značilnosti ontologij, s katerimi se ukvarjamo. Bolj načelen pristop k izboru vrednosti α in β bi lahko temeljil na tem, da bi vnaprej postavili eksplicitne zahteve o tem, kakšno vrednost $e^{-\alpha l}$ hočemo v povprečju dobiti za dva naključno izbrana (in torej običajno ne preveč sorodna) koncepta in kakšno vrednost $\tanh(\beta h)$ hočemo dobiti za dva zelo sorodna koncepta.

5.5.1 Odstranjevanje spodnjih plasti drevesa

Pri tem scenariju obdržimo le gornjih k nivojev drevesa (za različne vrednosti k), torej nivoje od 0 (koren) do $k - 1$. Koncepte na nivojih od vključno k naprej zavržemo; instance, ki so pripadale kakšnemu od teh konceptov, pa premestimo v najglobljega nepobrisanega prednika tega koncepta (torej tistega na nivoju $k-1$). Tako dobljeno ontologijo nato primerjamo s prvotno ontologijo. Ta scenarij ponazarja primer, ko se ontologijo gradi avtomatsko od zgoraj navzdol (npr. s hierarhičnim razbijanjem množice instanc) in se nek ustavitveni pogoj odloči gradnjo ontologije ustaviti prezgodaj; rezultat je hierarhija, v kateri manjkajo globlje ležeči koncepti. Graf na sliki 5.1 kaže, kako se mera na podlagi skupnih prednikov (5.2) in mera na podlagi razdalje v drevesu (5.3) odzivata na takšno postopno brisanje spodnjih delov hierarhije.

Opazimo lahko, da mera na podlagi prekrivanja med predniki narašča monotono s številom obdržanih nivojev. Ta rast je na začetku hitra, proti koncu pa počasna, kar je razumljivo glede na prej omenjeno dejstvo, da je v najglobljih nivojih drevesa razmeroma malo konceptov (in instanc), torej se ontologija ne spremeni preveč, če jih pobrišemo. Če bi gradili ontologijo od zgoraj navzdol in se ustavili po največ sedmih nivojih, bi indeks OntoRand ocenil podobnost med dobljeno ontologijo in prvotno (pri kateri je, kot smo omenili zgoraj, povprečna globina koncepta približno 7) na 0,94. Po drugi strani pa, če bi se ustavili že po največ treh nivojih, bi bil indeks OntoRand med to ontologijo in prvotno okoli 0,74.

Po svoje je malo presenetljivo, da je mera podobnosti spremenjene in prvotne ontologije še vedno tako visoka (0,74), četudi smo od prvotne ontologije obdržali le zgornje tri nivoje. To lahko bolje razumemo s pomočjo naslednjega razmisleka. Mislimo si dva naključno izbrana koncepta; v prvotni ontologiji sta z veliko verjetnostjo razmeroma nesorodna (njun najgloblji skupni prednik je visoko v drevesu, najverjetneje kar koren drevesa)



Slika 5.1. Evaluacija ontologij, ki jim manjkajo spodnje plasti, na podlagi indeksa OntoRand. Mera podobnosti na podlagi prekrivanja prednikov uporablja za δ_U definicijo (5.2), mera na podlagi razdalje v drevesu pa definicijo (5.3). Pikčasta črta kaže analitično pridobljeni približek indeksa OntoRand z mero na podlagi prekrivanja prednikov.

in ležita na globini nekje okrog 7; njuni množici prednikov (ki ju uporablja formula (5.2) imata presek 1 in unijo okrog 13, torej je δ_U za tadva koncepta okoli $1/13$. V spremenjeni ontologiji, ki vsebuje le zgornjih k nivojev, bi bili dokumenti iz takšnih dveh konceptov preseljeni v njuna prednika na nivoju $k-1$ in δ_U bi bila zdaj okrog $1/(2k-1)$. Takšni pari dokumentov (ki jih je veliko) bi torej vrednost indeksa OntoRand premaknili v bližino $1-|1/13-1/(2k-1)|$. Kot kaže krivulja „analitični približek“ na grafu, ta formula dejansko lepo opiše obliko krivulje indeksa OntoRand na podlagi mere s skupnimi predniki.

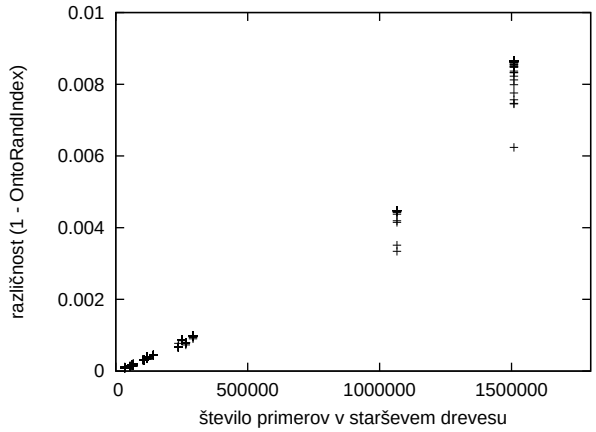
Mera podobnosti na podlagi razdalje v drevesu je pri tem scenariju malo bolj neugodna. V prvotnem drevesu tipičen naključno izbran par instanc leži v nesorodnih konceptih, ki razen korena nimata skupnih prednikov, torej imata $h=0$ in zato $\delta=0$ (ali pa δ blizu 0, če je h pozitiven, a majhen). Če porežemo nekaj spodnjih plasti drevesa in instance iz njih preselimo v prednike na najglobljem obdržanem nivoju, bo imel vsak par instanc, ki je imel prej $h=0$, še vedno $h=0$, torej se δ (po formuli (5.3)) zanj ne bo spremenila in ne bo prispevala k zmanjšanju mere podobnosti med prvotno in spremenjeno ontologijo. Zato pri tej meri podobnosti ostane vrednost indeksa OntoRand visoka ves čas. K zmanjšanju indeksa prispevajo le tisti pari konceptov (oz. instanc iz njih), pri katerih je $h>0$, ker se za takšne pare razdalja med konceptoma (l v formuli (5.3)) po rezanju spodnjih plasti drevesa zmanjša. Ker se l uporablja v členu $e^{-\alpha l}$, se vrednost δ za tak par konceptov po rezanju spodnjih plasti poveča in to tem bolj, čim več plasti smo porezali; to pa prispeva k zmanjšanju indeksa OntoRand. Do še enega čudnega pojava pa pride čisto na koncu, ko ostane le še en nivo

in torej h za čisto vse pare instanc pade na 0; torej, ko se z dveh nivojev premaknemo na enega, se δ ne poveča več, pač pa pade na 0 za vse pare instanc (spomnimo se, da je bila dotlej $\delta = 0$ le za tiste pare instanc, ki ležijo v konceptih, katerih skupni prednik je le koren; takšnih parov je sicer večina, ne pa vsi), tako da se indeks OntoRand pri tem premiku poveča, ne pa zmanjša. To nemonotonost na koncu bi se dalo odpraviti z manjšim popravkom formule (5.3), kar pa v praksi tako ali tako ne bi imelo velikega učinka, saj bi ontologija, ki bi jo primerjali z zlatim standardom, gotovo imela več kot en nivo.

5.5.2 Zamenjava koncepta z njegovim staršem

Tej operaciji na drevesih pogosto pravijo tudi „rotacija“. Mislimo si nek koncept c in označimo s c' njegovega starša (neposredni nadkoncept) v drevesu. Rotacija koncepta c je v bistvu zamenjava konceptov c in c': po rotaciji postane c' otrok koncepta c; vsi ostali otroci koncepta c' (ki so bili pred rotacijo sorojenci koncepta c) ostanejo otroci c' (in so po novem vnuki koncepta c); vsi otroci koncepta c (ki so bili pred rotacijo vnuki koncepta c') pa po rotaciji ostanejo otroci c (in so zdaj sorojenci koncepta c'). Če je imel c' pred rotacijo nekega starša c'', je po rotaciji koncept c'' roditelj koncepta c, ne koncepta c'. Posledica te rotacije je podobna, kot če bi se nek postopek, ki avtomatsko gradi ontologijo od spodaj navzgor, zmotil pri razbijanju množice instanc nekega koncepta na podmnožice za posamezne podkoncepte; na primer, namesto da bi množico dokumentov s področja znanosti razbil na podmnožice za fiziko, kemijo, biologijo itd., bi jih razbil na podmnožice za mehaniko, termodinamiko, jedrsko fiziko in „razno“, v to zadnjo podmnožico pa bi uvrstil vse dokumente s področja kemije, biologije itd.

Kako vpliva ta operacija na vrednosti h in l, ki ju uporabljamo v formulah (5.2) in (5.3)? Če sta bila oba opazovana koncepta prej v poddrevesu, ki se začne pri c, se vrednost h zmanjša za 1; če sta bila oba v poddrevesu, ki se začne pri c', ne pa v tistem, ki se začne pri c, se h poveča za 1; če je bil eden od konceptov v c-jevem poddrevesu, drugi pa zunaj c'-jevega poddrevesa, se l poveča za 1; če je bil eden od konceptov v c'-jevem poddrevesu, ne pa tudi v c-evem poddrevesu, drugi koncept pa je bil sploh zunaj c'-jevega poddrevesa, se l zmanjša za 1; v ostalih primerih pa se ne spremeni nič (niti h niti l). V to zadnjo skupino, torej med primere, ko se ne spremeni niti h niti l, sodijo še posebej vsi tisti pari konceptov, pri katerih noben od obeh konceptov ne leži v c'-jevem poddrevesu; to pokrije veliko večino vseh parov instanc (razen če je bilo c'-jevo poddrevo res zelo veliko). Zato je pri takšni rotaciji neujemanj glede medsebojnega položaja dveh instanc pred in po rotaciji ponavadi razmeroma malo in vrednost indeksa OntoRand, če z njim primerjamo ontologijo pred in po rotaciji, je pogosto precej blizu 1. Ta pojav je še posebej izrazit, če namesto mere podobnosti na podlagi razdalje



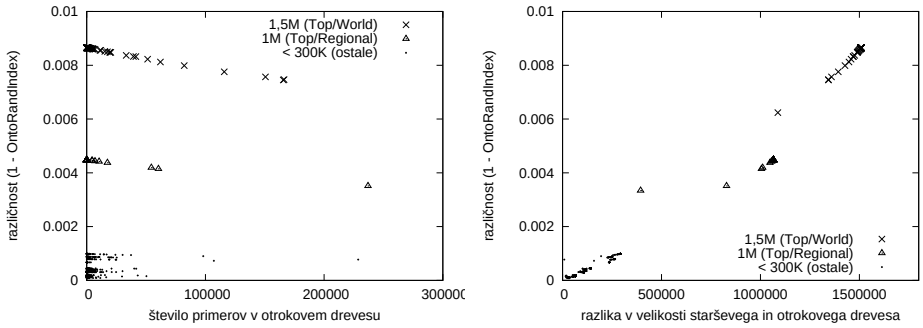
Slika 5.2. Primerjava ontologij, pri katerih je bil nek koncept c zamenjan s svojim staršem c'. Vsak simbol na grafu predstavlja rezultat poskusa za enega od možnih konceptov c. Za x-koordinate smo uporabili število instanc v poddrevesu starševskega koncepta (torej c'), za y-koordinato pa mero različnosti ($1-$OntoRand) med ontologijo po zamenjavi in prvotno ontologijo.

Vidimo lahko, da se mera različnosti povečuje približno linearno z velikostjo starševega poddrevesa. Skupine simbolov na desni strani se nanašajo na poskuse, pri katerih je bil c otrok ene od dveh največjih kategorij z drugega nivoja (Top/World in Top/Regional).

v drevesu (formula (5.3)) uporabimo tisto na podlagi prekrivanja prednikov (formula (5.2)). Zato v spodnjih grafih (sliki 5.2 in 5.3) prikazujemo le rezultate poskusov z mero podobnosti na podlagi prekrivanja med predniki, namesto vrednosti indeksa OntoRand pa smo prikazali razliko $1-$OntoRand.

V ontologiji ODP ima koren 640 vnukov in za vsakega od njih smo izvedli po en poskus, v katerem smo ta koncept zamenjali z njegovim staršem (npr. Top/Science/Physics z Top/Science ipd.).

Slika 5.2 kaže, da je razlika med ontologijo po rotaciji in prvotno ontologijo tem večja, čim večje je poddrevo c-jevega starša c', slika 5.3 pa kaže, da je ta razlika tem manjša, čim večje je c-jevo poddrevo samo. To je razumljivo; več ko je instanc v c-jevem poddrevesu, manj se c razlikuje od svojega starša in manj se torej ontologija spremeni, če zamenjamo c z njegovim staršem. Na primer, zgornja skupina simbolov „$\times$“ na sliki 5.3 se nanaša na poskuse, pri katerih je bil c eden od otrok kategorije Top/World, ki je največja kategorija na drugem nivoju drevesa. Kot kaže desni graf na sliki 5.3, je mera razlike med prvotno in spremenjeno ontologijo skoraj linearno sorazmerna z razliko v velikosti med drevesom c-jevega očega in c-jevim drevesom samim.

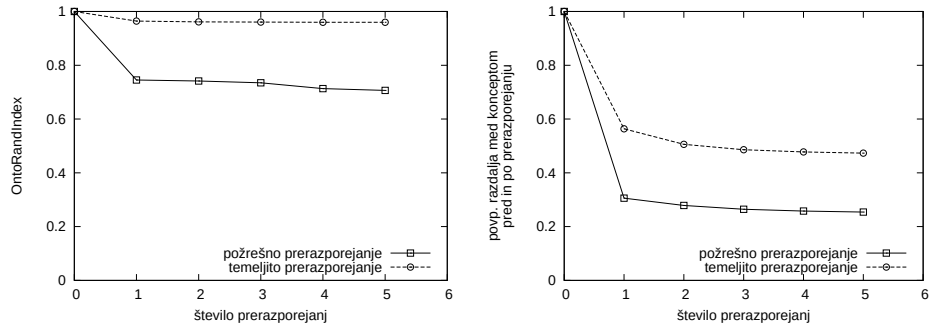


Slika 5.3. Primerjava ontologij, pri katerih je bil nek koncept c zamenjan s svojim staršem c'. Ta grafa kažeta odnos med različnostjo ontologij (torej $1 - \mathit{OntoRand}$) in številom instanc v c-jevem poddrevesu. Za vsako možno izbiro koncepta c je prikazan po en simbol (oblika simbola je odvisna od števila instanc v poddrevesu c-jevega starša c'). Na levem grafu smo za x-koordinate uporabili število instanc v c-jevem poddrevesu, na desnem pa razliko v številu instanc med c'-jevim in c-jevim poddrevesom. Za y-koordinate pa smo na obeh grafih uporabili mero različnosti ($1 - \mathit{OntoRand}$) med ontologijo pred in po zamenjavi.

5.5.3 Prerazporejanje instanc med koncepte

V ontologiji ODP je vsaka instanca pravzaprav kratek dokument v naravnem jeziku (v večini ontologije je to angleščina), ki ga sestavljata naslov in kratek opis (skupaj običajno okoli 10–20 besed) neke zunanje spletne strani. V tem razdelku smo sledili običajni praksi s področja poizvedovanja po tekstovnih podatkih in predstavili vsak tak dokument z normaliziranim vektorjem uteži TF-IDF. Iz teh vektorjev lahko za vsak koncept izračunamo centroid (torej povprečje vektorjev za vse instance, ki pripadajo temu konceptu ali njegovim potomcem v hierarhiji). Nato lahko poskusimo vsak dokument prerazporediti v tisti koncept, čigar centroid je najbližji vektorju opazovanega dokumenta. Po takšni prerazporeditvi so torej hierarhični odnosi med koncepti ostali nespremenjeni, razporeditev instanc med koncepte pa se lahko precej spremeni. Takšno prerazporejanje instanc med koncepte je podobno operacijam, ki bi se jih lahko uporabljalo pri avtomatski gradnji in polnjenju ontologij (npr. podobno kot pri razvrščanju v skupine s postopkom k-means). Po prerazporejanju instanc izračunamo mero podobnosti med novo in prvotno ontologijo.

Zaradi učinkovitosti poskusov v tem razdelku nismo izvedli na celotni ontologiji ODP, pač pa le na poddrevesu "Science". V njem je 11 624 konceptov in 104 853 instanc (dokumentov). Preizkusili smo dve strategiji prerazporejanja dokumentov: „temeljito prerazporejanje“ primerja vsak dokument s centroidi vseh konceptov, „požrešno prerazporejanje“ pa začne v korenu drevesa, v vsakem koraku primerja trenutni dokument le s centroidi otrok trenutnega koncepta in se nato premakne v tistega otroka, čigar centroid



Slika 5.4. Primerjava ontologije po tistem, ko smo njene instance prerazporedili med koncepte na podlagi opisov instanc v naravnem jeziku. Za x-koordinato na grafu je uporabljeno število prerazporeditev (v vsaki prerazporeditvi na novo razporedimo vse instance in nato izračunamo nove centroide vseh konceptov, ti centroidi pa so podlaga za naslednjo prerazporeditev). Levi graf kaže podobnost (indeks OntoRand) med prvotno ontologijo in ontologijo po prerazporeditvi instanc. Na desnem grafu pa smo za vsako instanco izračunali razdaljo med njenim konceptom v prvotni in v spremenjeni ontologiji (z mero δ_U po formuli (5.2)) in prikazali povprečje te razdalje po vseh instancah.

je trenutemu dokumentu najbližji; postopek se ustavi, ko pride do lista ali pa do koncepta, čigar centroid je dokumentu bolj podoben kot centroid kateregakoli izmed otrok tega koncepta. Požrešno prerazporejanje je veliko hitrejše od temeljitega, je pa izpostavljeno tveganju, da ga bodo neugodne odločitve na zgornjih nivojih zapeljale v napačen del drevesa.

Po tistem, ko vse dokumente prerazporedimo med koncepte, lahko na novo izračunamo centroide vseh konceptov (na podlagi nove množice instanc, ki pripada posameznemu konceptu po prerazporeditvi) in ti novi centroidi so lahko podlaga za ponovno prerazporeditev vseh dokumentov. Grafa na sliki 5.4 kažeta rezultate za do pet prerazporeditev; v obeh grafih smo uporabili mero podobnosti na podlagi prekrivanja prednikov (formula (5.2)).

Levi graf na sliki 5.4 kaže podobnost med ontologijo po določenem številu prerazporeditev in prvotno ontologijo. V skladu s pričakovanji požrešno prerazporejanje instanc ontologijo spremeni precej bolj kot temeljito prerazporejanje. Večina razlike glede na prvotno ontologijo nastane že ob prvi prerazporeditvi; tudi to je po svoje razumljivo, saj bi težko pričakovali, da bo preprosta klasifikacija na podlagi centroidov in 10–20 besed dolgih opisov dajala podobno dobre rezultate, kot jih ročno dosežejo človeški uredniki ontologije ODP. Izkaže se, da pri prvi prerazporeditvi kar 93 % dokumentov pristane v drugem konceptu, če uporabljamo požrešno prerazporejanje (pri temeljitem prerazporejanju je ta delež okrog 66 %). Mera podobnosti med prvotno in spremenjeno ontologijo pa je vseeno precej visoka, okrog 0,74. Razlogi za to so: (1) spremenila se je le razporeditev instanc med koncepte, ne pa tudi hierarhični odnosi med koncepti; (2) če se dokumente

prerazporedi med koncepte na konsistenten način, se lahko za mnoge pare instanc vrednost δ_U spremeni le malo in zato indeks OntoRand ostane visok; (3) četudi se je 93 % dokumentov premaknilo v drug koncept, je novi koncept v mnogih primerih precej blizu prvotnemu. To ilustrira desni graf na sliki 5.4, pri katerem smo za vsako instanco izračunali δ_U med konceptoma, v katerih leži ta instanca pred oz. po prerazporejanju; graf kaže povprečje te razdalje po vseh instancah. Kot vidimo na tej sliki, čeprav je le 7 % instanc ostalo v istem konceptu, povprečna podobnost δ_U med starim in novim konceptom ni npr. 0,07, pač pa je precej večja, okoli 0,31.

5.6 Zaključki in možnosti za nadaljnje delo

Glavna značilnost pristopa, ki smo ga opisali v tem poglavju, je, da se osredotoča na popolnoma avtomatsko evaluacijo ontologij na podlagi primerjave z zlatim standardom (gold standard); o predstavitvi instanc in konceptov ne predpostavlja ničesar, zahteva le, da vsebujeta obe ontologiji enako množico instanc. Predlagali smo novo mero podobnosti med ontologijami, indeks OntoRand, ki je definiran po zgledu Randovega indeksa, ki ga pogosto uporabljajo za primerjavo dveh razbitij (particij) neke množice. Tako dobljeno mero bi se dalo uporabljati tudi za ocenjevanje postopkov za hierarhično razvrščanje v skupine (*hierarchical clustering*). Opisali smo več različic indeksa OntoRand, ki temeljijo na različnih merah razdalje med koncepti v ontologiji. Predlagani pristop smo preizkusili na veliki ontologiji, pobrani iz spletnega imenika ODP. Naši poskusi so temeljili na več operacijah, ki spreminjajo prvotno ontologijo (zlati standard) in s tem posnemajo možna odstopanja, do katerih lahko pride med zlatim standardom in ontologijo, ki bi jo avtomatsko zgradili iz podatkov (npr. dane množice instanc). Poskusi so pokazali, da je mera, ki temelji na podobnosti množic prednikov (razd. 5.4.3), prikladnejša od mere, ki temelji na razdalji v drevesu (razd. 5.4.3), ker mora pri slednji uporabnik izbrati vrednosti dveh parametrov in je to težko narediti na primerno načelen način. Poleg tega je mera, ki temelji na razdalji v drevesu, pogosto stlačila vrednosti indeksa OntoRand v precej manjši del intervala $[0, 1]$ kot mera, ki temelji na skupnih prednikih; za rešitev tega problema smo predlagali popravek formule (5.5) in nadaljnji poskusi s to formulo bi bili zanimiva možna smer nadaljnjega dela. Pri obeh obravnavanih merah podobnosti ima indeks OntoRand tudi to slabost, da je včasih premalo občutljiv na spremembe, ki se dogajajo v zgornjih nivojih ontologije (razd. 5.5.2). Poskusi s prerazporejanjem instanc med koncepte ob nespremenjeni hierarhiji konceptov (razd. 5.5.3) so pokazali tudi, da takšne spremembe ontologije na indeks OntoRand včasih ne vplivajo toliko, kot bi človeški opazovalec mogoče pričakoval.

Zanimiva smer nadaljnjega dela je tudi vprašanje, ali lahko indeks OntoRand, kot smo ga definirali v razdelku 5.4.3, računamo točno (ne apro-

ksimacijsko) z manj kot kvadratno časovno zahtevnostjo (v odvisnosti od števila instanc).

Poskuse iz razdelka 5.5 bi lahko razširili še z raznimi drugimi operacijami. Obstoječe liste drevesa bi lahko na primer poskusili razdeliti na podkoncepte, bodisi naključno ali pa z uporabo tehnik razvrščanja v skupine (clustering). To bi bil v nekem smislu inverz operacije iz razdelka 5.5.1, v katerem smo spodnje nivoje hierarhije opuščali. Zanimiva operacija bi bilo tudi združevanje dveh ali več sorojencev, torej konceptov z istim staršem v drevesu.

Poskusi z zamenjavo koncepta s staršem (razd. 5.5.2) so pokazali, da ima preurejanje konceptov v gornjih nivojih drevesa (v našem primeru je bila to zamenjava koncepta na tretjem nivoju z njegovim staršem na drugem nivoju) precej majhen vpliv na mero podobnosti. Z vidika človeškega uporabnika je ta lastnost lahko neugodna, ker razlike v zgornjih nivojih ontologije predstavljajo pomembne razlike v odločitvah o tem, kako strukturirati in konceptualizirati problemsko področje, na katerega se ontologija nanaša. Uporabnik bi si zato lahko želel, da bi bila mera podobnosti med ontologijami bolj občutljiva na take razlike in koristno bi bilo, če bi lahko tudi naš indeks OntoRand prilagodili v tej smeri.

5.6.1 Primerjava ontologij brez predpostavke o enaki množici instanc

Pristop, ki smo ga predlagali v razdelku 5.4.3, predpostavlja, da primerjamo dve ontologiji, zgrajeni nad isto množico instanc (lahko pa imata različni množici konceptov, različno hierarhično relacijo nad koncepti in različno razporeditev instanc med koncepte). Zanimiva razširitev tega pristopa bi bila, da bi podprli tudi primerjavo ontologij z različnimi množicami instanc. V takem primeru ne moremo več za vsak par instanc gledati njun relativni medsebojni položaj v eni in v drugi ontologiji, ker se instance ne pojavljajo več v obeh ontologijah. Če so instance predstavljene s tekstovnimi dokumenti (ali pa s čim drugim, kar nam jih omogoča primerjati med sabo), bi lahko za vsako instanco ene ontologije poiskali nekaj najbolj podobnih instanc v drugi ontologiji; ko bi se potem pri računanju mere podobnosti med ontologijama U in V ukvarjali z instancama o_i in o_j iz U, bi primerjali razdaljo δ_U med njima v U z vrednostjo δ_V za razne pare njima najbolj podobnih instanc v V. Vendar pa bi takšno iskanje podobnih instanc močno povečalo časovno zahtevnost računanja mere podobnosti med ontologijama.

Primerjava dveh ontologij bi lahko temeljila tudi na načelu urejevalniške razdalje. Tak pristop bi iskal zaporedje urejevalniških operacij, ki lahko preoblikujejo eno ontologijo v drugo, pri tem pa bi minimiziral neko skupno ceno (npr. število operacij). Vendar pa, če imata ontologiji vsaka svoj nabor konceptov (in mogoče celo vsaka svojo množico instanc), bi bilo težko najti najcenejše zaporedje urejevalniških operacij. Učinkoviti algoritmi za

računanje urejevalniške razdalje med drevesi, v katerem je vrstni red otrok pomemben (*ordered trees*), sicer obstajajo (npr. [13]), tu pa bi imeli opravka z drevesi brez vrstnega reda otrok, kar je težje.

Mere podobnosti med ontologijami bi lahko temeljile tudi na načelih iz teorije informacije. Za primerjavo nehierarhičnih razbitij iste množice instanc obstaja na primer mera, ki temelji na variaciji informacije [46] in za katero se da pokazati, da ima več zaželenih in teoretično privlačnih lastnosti [47]. Ta mera gleda na pripadnost skupini (npr. konceptu) kot na naključno spremenljivko; dve različni razbitji iste množice sta torej dve naključni spremenljivki, podobnost med razbitjema pa lahko merimo z medsebojno informacijo med njunima spremenljivkama. To mero bi lahko razširili tudi na hierarhična razbitja, pri čemer bi morala mera zdaj odgovoriti na približno takšno vprašanje: koliko bitov informacije moramo prenesti, da lahko za vsako instanco opišemo, kje v drugi hierarhiji leži, če za vse instance že vemo, kje v prvi hierarhiji ležijo? Potrebno bi bilo vpeljati primerno kodno shemo; na primer, za vsak koncept c iz prve ontologije poiščimo najbolj podoben koncept c' iz druge ontologije; potem pa za vsako instanco o iz c opišemo njen položaj v drugi ontologiji tako, da opišemo pot v njej od c' do tistega koncepta, ki v tej ontologiji zares vsebuje instanco o.

5.6.2 Evaluacija brez zlatega standarda

Prvotna motivacija za to poglavje je bilo razmisliti o merah za evaluacijo ontologij, torej za ocenjevanje ali vrednotenje ontologije, po vsej verjetnosti take, ki je bila pridobljena z nekim avtomatskim procesom in bi si zato želeli, da je tudi evaluacija avtomatska. Pristop, s katerim smo se potem ukvarjali, ima to pomembno omejitev, da primerja ontologijo z zlatim standardom in torej odpove v primerih, ko takšnega standarda nimamo. Ena možnost je, da je prisoten vsaj delni zlati standard, npr. seznam pomembnih konceptov s problemskega področja, ki pa niso urejeni v hierarhijo; v tem primeru bi lahko evaluacija temeljila na natančnosti in priklicu ter drugih podobnih merah s področja poizvedovanja po podatkih. Še ena možnost je, da imamo zlati standard za neko drugo problemsko področje, ne pa tisto, ki nas zanima; uporabimo ga lahko tako, da ugotovimo, kateri učni algoritem in njegovi parametri se najbolje obnesejo na tistem drugem problemskem področju, potem pa iste nastavitve uporabimo tudi na našem področju in upamo, da bodo dale dober rezultat tudi na njem.

Zanimivo pa bi bilo razmisliti tudi o pristopih, ki se v celoti izognejo potrebi po kakršnem koli zlatem standardu. Pri „ploščatih“, nehierarhičnih razbitjih, s kakršnimi se tradicionalno srečujemo pri razvrščanju v skupine (clustering), se pogosto uporabljajo mere, kot sta kompaktnost skupin in povprečna razdalja med skupinami; te mere odražajo željo, naj si bodo instance iz iste skupine čim bližje skupaj, instance iz različnih skupin pa čim dlje narazen. Tovrstne mere bi lahko poskušali posplošiti na hierarhična

razbitja. Še en in precej drugačen pristop pa bi bil, da bi za ocenjevanje razbitij uporabili strojno učenje: vsako skupino instanc v našem razbitju si predstavljajmo kot razred in poskušajmo zanj zgraditi nek klasifikacijski model. Če je bilo razbitje na skupine razumno, bi pričakovali, da bo točnost dobljenih modelov višja, kot če je razbitje na primer čisto naključno oz. nima zveze z atributi naših instanc.

Poglavje 6

Odkrivanje zanimivih trojic v besedilu

6.1 Uvod

Mnogo potencialno zanimivih informacij o svetu se nahaja v nestrukturiranih besedilih v naravnem jeziku, ne pa v strojno berljivi obliki, kot so ontologije ali RDF-jevske trojice. Zato je vse bolj zanimiv problem avtomatske ekstrakcije ontoloških podatkov iz besedila. Dosedanja dela na tem področju so se na primer ukvarjala z ekstrakcijo trojic iz n-gramov [71] in z ekstrakcijo semantično opremljenih dejstev iz novičarskih člankov [79].

Opis problema. Ontologijo ali bazo znanja si lahko predstavljamo kot graf, v katerem točke ustrezajo konceptom, povezave (ki so označene tudi z imeni ali kakšnimi drugimi atributi) pa odražajo neke relacije nad temi koncepti. Enakovreden pogled pa je, da si vsako povezavo predstavljamo kot trojico $\langle koncept_1, relacija, koncept_2 \rangle$. Podobno je pri besedilu v naravnem jeziku, ki je sestavljeno iz stavkov, posamezni stavek pa ima podobno zgradbo: $\langle osebek, povedek, predmet \rangle$. Tu sta osebek in predmet dva koncepta, povedek pa odraža relacijo med njima. Zato se zdi naravno, da želimo iz besedila v naravnem jeziku izluščiti trojice koncept-relacija-koncept.

Problem ekstrahiranja takšnih trojic za namen polnjenja ontologije ali baze znanja lahko razdelimo na dva glavna dela. Prvi del temelji na sintaktični analizi besedila, s pomočjo katere lahko razbijemo večstavčne povedi na posamezne stavke, identificiramo v njih osebke, povedke in predmete, ugotovimo, na kaj se nanašajo zaimki ipd.

Drugi del problema pa je povezan z dejstvom, da dokumenti v naravnem jeziku ponavadi vsebujejo specifične podatke o zelo konkretnih konceptih. Če je naš cilj dopolniti neko precej splošnonamensko ontologijo, kot je na primer Cyc [38], z dejstvi, ki bodo pomagala pri raznih vrstah sklepanja nad to ontologijo, pa bi nas precej verjetno še bolj kot takšna konkretna dejstva o konkretnih konceptih zanimala abstraktnejša in splošnejša dej-

stva. Na primer: namesto (ali pa poleg) dejstva, da John Smith živi v Kanadi $\langle \textit{John Smith}, \textit{live}, \textit{Canada} \rangle$, in dejstva, da Jane Jones živi v Avstraliji $\langle \textit{Jane Jones}, \textit{live}, \textit{Australia} \rangle$, bi utegnilo biti še bolj koristno vedeti, da ljudje živijo v državah (oz. z drugimi besedami, da je država nekaj, v čemer lahko ljudje prebivajo). Tako je torej $\langle \textit{person}, \textit{inhabit}, \textit{country} \rangle$ primer takšne abstraktnejše, splošnejše trojice, ki se v korpusu besedil v naravnem jeziku najbrž ne bo neposredno pojavila (vsaj ne velikokrat), ki pa bi jo bilo zanimivo odkriti in dodati v našo ontologijo. Izziv je torej v tem, kako odkriti zanimive in koristne trojice, ki niso nujno „konkretne“ (se ne pojavljajo neposredno v besedilu), pač pa jih lahko iz konkretnih trojic dobimo z nekakšnim procesom abstrakcije. Pri našem delu smo se odločili za abstrakcijo uporabljati podatke o nadpomenkah (hipernimih) iz WordNeta: posamezni korak abstrakcije se sestoji iz tega, da eno od komponent v trojici zamenjamo z enim od njenih hipernimov iz WordNeta.

V tem delu se ukvarjamo le z opisanim drugim delom problema, za prvi del (ekstrakcijo osebkov, povedkov in predmetov iz naravnega besedila) pa uporabljamo rezultate del Rusu in sod. [63, 64].

6.2 Odkrivanje pogostih trojic s posploševanjem

6.2.1 Od fraz do trojic konceptov

Vhodni podatki, s katerimi smo začeli našo obdelavo, so bili seznam trojic oblike $\langle \textit{osebek}, \textit{povedek}, \textit{predmet} \rangle$, ki so jih Rusu in sod. [63] pridobili iz korpusa Reutersovih novic [61]. Vsaka od teh trojic približno ustreza enemu stavku vhodnega besedila. Ker se naš pristop k odkrivanju pogostih in zanimivih trojic močno opira na WordNet kot vir podatkov o nadpomenkah in podpomenkah, je pomemben prvi korak to, da ugotovimo, kateri koncept (ali več koncepvo) iz WordNeta ustreza posamezni komponenti vsake od vhodnih trojic.

Komponente vhodnih trojic so pogosto večbesedne zveze, ki se v WordNetu ne pojavljajo neposredno. Primer take trojice je $\langle \textit{European Union finance ministers}, \textit{approved}, \textit{convergence plans} \rangle$. V WordNetu ni koncepta “European Union finance ministers”; pač pa je v njem koncept “finance minister”, ki bi ga radi prepoznali kot osebek v naši trojici (in ga uporabljali pri nadaljnjem delu z njo). Naš postopek za povezovanje besedne zveze iz vhodne trojice s koncepti iz WordNeta deluje takole:

- Za začetek pretvorimo samostalnike, ki so bili v množini, v ednino, glagole pa v nedoločnik.

- Naj bo zdaj $(w_1, \ldots, w_n)$ zaporedje besed v opazovani besedni zvezi. Za vsako podzaporedje oblike $(w_i, w_{i+1}, \ldots, w_{i+d-1})$ pogledamo, če se pojavlja kot ime kakšnega koncepta v WordNetu.

- Med podzaporedji, ki se tako pojavljajo v WordNetu, vzamemo najdaljše podzaporedje (tisto z največjim d); če je takih več, pa med njimi vzamemo najbolj desno (tisto z največjim i).

Namen hevristik v tretji točki je naslednji: s tem, ko maksimiziramo d, skušamo najti čim bolj specifičen koncept (npr. "finance minister" namesto splošnejšega "minister"); s tem, ko maksimiziramo i, pa skušamo najti koncept, ki predstavlja jedro besedne zveze, saj se v angleščini to ponavadi pojavlja na desni (v zgornjem primeru, "European Union finance ministers", tako na primer prepoznamo WordNetov koncept „finance minister" namesto npr. „European Union").

V WordNetu je vsak koncept označen kot ena od naslednjih besednih vrst: samostalnik, glagol, pridevnik in prislov. Ko obdelujemo prvo in tretjo komponento naših vhodnih trojic (osebke in predmete), gledamo v WordNetu le tiste koncepte, ki so označeni kot samostalniki; ko obdelujemo drugo komponento (povedek), pa gledamo le koncepte, ki so označeni kot glagoli. Kljub tej dodatni omejitvi pa se lahko zgodi, da neka beseda nastopa kot ime več konceptov v WordNetu. Ker v naših vhodnih podatkih manjka kontekst, v katerem so opazovane besedne zveze prvotno nastopale, na tem mestu ne poskušamo izvajati disambiguacije pomena posameznih besed, pač pa takšno trojico obravnavamo kot dvo- oz. več-umno; takšna vhodna trojica torej ne podpira le ene konceptne trojice (torej trojice konceptov iz WordNeta), pač pa več konceptnih trojic.

Bolj formalno lahko to opišemo takole. Označimo naš vhodni korpus s $\mathcal{C} = \{\tau_i : i \in I\}$, posamezne vhodne trojice pa s $\tau_i = \langle \tilde{s}_i, \tilde{v}_i, \tilde{o}_i \rangle$. Množico WordNetovih konceptov, ki smo jih po gornjem postopku priredili posamezni komponenti trojice, označimo z $m_S(\tilde{s}_i)$, $m_V(\tilde{v}_i)$ oz. $m_O(\tilde{o}_i)$. (Idealno bi bilo, če bi vsaka od teh množic vsebovala natanko en koncept, zaradi dvoumnosti pa jih v praksi lahko vsebuje tudi več.) Vhodna trojica τ_i tako zdaj podpira naslednjo množico konceptnih trojic: $m_S(\tilde{s}_i) \times m_V(\tilde{v}_i) \times m_O(\tilde{o}_i)$; to označimo z $m(\tau_i)$.

6.2.2 Pogoste posplošene trojice

Kot smo že omenili, uporabljamo WordNetovo relacijo nadpomenka/podpomenka za posploševanje konkretnih konceptnih trojic v splošnejše in abstraktnejše. V naših konceptnih trojicah sta prva in tretja komponenta samostalnika, druga pa je glagol. Za vsako od teh besednih vrst ima WordNetu svoje drevo nadpomenk in podpomenk. Pri glagolih je pravzaprav več ločenih in precej plitvih dreves, katerih koreni nad seboj nimajo neke še bolj abstraktne skupne nadpomenke; samostalniki pa so vsi povezani v eno samo drevo (v korenu je koncept "entity"), ki je marsikje globoko tudi čez deset nivojev. Zato se lahko zgodi, da iz neke konkretne konceptne trojice s posploševanjem nastane več deset različnih bolj ali manj posplošenih trojic.

Na primer: v WordNetu ima koncept "finance minister" nadpomenko "minister", ta pa ima nadpomenko "executive"; podobno ima koncept "plan" nadpomenko "idea". Tako bo vsak stavek, ki podpira konceptno trojico $\langle$*finance minister, approve, plan*$\rangle$, podpiral tudi malo splošnejši konceptni trojici $\langle$*executive, approve, plan*$\rangle$, $\langle$*minister, approve, idea*$\rangle$ in še precej drugih.

Naj bo $h(c)$ množica neposrednih nadpomenk koncepta c v WordNetu, $h^\star(c)$ pa množica neposrednih in posrednih nadpomenk, vključno s samim c. To definicijo lahko razširimo tudi na trojice: $h^\star(s,v,o) = h^\star(s) \times h^\star(v) \times h^\star(o)$; in na množice (konceptov ali trojic): $h^\star(A) = \cup_{x\in A} h^\star(x)$. Podporo konceptne trojice $\langle s,v,o\rangle$ lahko zdaj definiramo kot

$$sup(s,v,o) := |\{i \in I : \langle s,v,o\rangle \in h^\star(m(\tau_i))\}|.$$

Naš cilj pri tem koraku je identificirati konceptne trojice (lahko bolj ali manj splošne), katerih podpora presega nek izbrani prag (in določiti dejansko podporo teh trojic). Načeloma bi se dalo ta problem prevesti na problem pogostih podmnožic (*frequent itemset problem*), za katerega obstajajo dobro znani algoritmi, zlasti Apriori [1]. Iz vsake vhodne trojice $\tau_i = \langle \tilde{s}_i, \tilde{v}_i, \tilde{o}_i\rangle$ lahko sestavimo množico konceptov $C(\tau_i)$, v kateri naj bodo vsi koncepti, ki smo jih po postopku iz razdelka 6.2.1 pripisali besednim zvezam iz te vhodne trojice, in vse (neposredne ali posredne) nadpomenke teh konceptov; torej $C(\tau_i) = h^\star(m_S(\tilde{s}_i)) \cup h^\star(m_V(\tilde{v}_i)) \cup h^\star(m_O(\tilde{o}_i))$. Nad tako dobljenim seznamom množic konceptov $C(\tau_i)$ (za vse vhodne trojice τ_i) lahko poženemo Apriori in z njim poiščemo vse podmnožice, katerih podpora presega zahtevani prag. Težava tega pristopa je, da bi v rezultatih poleg množic, ki ustrezajo pogostim (posplošenim) konceptnim trojicam, našli tudi mnoge množice, ki so sicer z vidika problema, ki ga rešuje Apriori, pogoste, vendar ne ustrezajo nikakršni trojici (torej niso oblike $C(\tau)$ za nobeno konceptno trojico τ, npr. ker ne vsebujejo nobenega povedka ali pa vsebujejo več različnih povedkov ali pa vsebujejo nek koncept in ne vsebujejo njegove nadpomenke).

Obstaja sicer tudi različica Apriorija, prilagojena za probleme, pri katerih je nad elementi množic definirana hierarhija [65], vendar za naše namene tudi ta ni neposredno uporabna. Ta različica temelji na opažanju, da če neka množica vsebuje tako nek element x kot nekega njegovega prednika v hierarhiji, recimo x' (v našem primeru bi to bila nadpomenka), se podpora te množice nič ne spremeni, če iz nje x' pobrišemo. Na podlagi tega je mogoče pokazati, da če v množici pogostih skupin (*itemsets*) velikosti k ni nobene take, ki bi vsebovala tako nek element kot nekega njegovega prednika, potem tudi v množici kandidatk za pogoste skupine velikosti $k+1$, ki jo bo zgeneriral Apriori pred naslednjim prehodom čez podatke, ne bo nobene take množice. Torej, če na to posebej popazimo pri pripravi kandidatov za pogoste skupine velikosti 2 (in zavržemo tiste, v katerih je en element prednik drugega), bo od tam naprej Apriori že sam od sebe zagotavljal, da v

pogostih skupinah, ki jih bo našel, ne bo takih, v katerih bi bil en element prednik kakšnega drugega. Vendar pa za naše potrebe to še ne zadošča, saj nam pri takšnem postopku še vedno lahko nastanejo množice, ki vsebujejo npr. več glagolov (lahko celo iz čisto različnih delov drevesa), a nobenega predmeta ipd. To težavo bi lahko odpravili z dodatnimi omejitvami, npr. da v množico kandidatk vključimo le skupine, ki vsebujejo natanko en osebek, en povedek in en predmet. Tudi s takšnimi dodatnimi omejitvami pa nam ostane precej resnejši problem, namreč ta, da bi s takšnim Apriorijem pri prehodu s pogostih skupin velikosti 2 na kandidatke za pogoste skupine velikosti 3 teh kandidatk nastane preveč; pri naših poskusih jih je bilo približno 502 milijona, od katerih je bilo zares pogostih le približno 40 milijonov. Od osnovne ideje Apriorija, torej da prostor možnih skupin pregleduje postopoma, od manjših proti večjim, in uporablja podatke o pogostih manjših skupinah za pripravo kandidatk za pogoste večje skupine, tu pravzaprav ni ostalo več veliko, saj gremo le do skupin velikosti 3, te pa poskušamo obdelati praktično vse na en mah. Potrebujemo torej način, kako razdeliti prostor možnih skupin velikosti 3 (torej potencialnih pogostih konceptnih trojic) in ga preiskovati po delih, to pa tako, da si bomo lahko s prej odkritimi pogostimi trojicami pomagali pri omejevanju iskanja v nadaljevanju. Koristna opora pri takšni razdelitvi prostora možnih trojic je globina pojmov v WordNetovih hierarhijah.

Za posamezni koncept c označimo z $d(c)$ njegovo globino v Wordnetovi hierarhiji nadpomenk in podpomenk. Primer: $d(\mathit{mammal}) = 9$ zaradi zaporedja nadpomenk *mammal* → *vertebrate* → *chordate* → *animal* → *organism* → *living thing* → *whole* → *object* → *physical entity* → *entity*. Koncepti brez nadpomenk imajo $d = 0$ (npr. "entity", ki je v korenu hierarhije samostalnikov). Pojem globine lahko razširimo tudi na konceptne trojice z definicijo $d(\langle s, v, o\rangle) := d(s) + d(v) + d(o)$.

Naš algoritem temelji na dejstvu, da če je npr. s' nadpomenka koncepta s, potem vsaka vhodna trojica, ki podpira konceptno trojico $\langle s, v, o\rangle$, podpira tudi konceptno trojico $\langle s', v, o\rangle$. Enako velja tudi za posplošitve (nadpomenke) drugih dveh komponent. Z drugimi besedami, če je trojica pogosta (torej če njena podpora dosega nek želeni minimalni prag), so pogoste tudi njene posplošitve. Zato velja tudi obratno: če kakšna posplošitev neke trojice ni pogosta, potem tudi ta trojica sama ne more biti pogosta in nam je ni treba vzdrževati v množici kandidatk za pogoste trojice. Trojice $\langle s, v, o\rangle$ bomo pregledovali po naraščajoči globini $d(s) + d(v) + d(o)$. V nadaljevanju prikazani algoritem pri vsakem prehodu čez vhodne podatke pripravi množico kandidatk za pogoste trojice na globini D in izračuna njihovo podporo, na koncu pa v množico F doda tiste, ki dosegajo predpisani prag minimalne podpore. Nato povečamo D za 1 in v naslednjem prehodu iščemo malo bolj specializirane trojice (z globino, večjo za 1). Ta postopek se nadaljuje, dokler še odkrivamo nove pogoste trojice.

algoritem PogosteTrojice:
vhod: zaporedje vhodnih trojic in minimalni prag za podporo, *MinSup*.
izhod: množica F vseh trojic, katerih podpora dosega prag *MinSup*.

$F := \{\}; D := 0;$
ponavljaj:
  $C := \{\};$
  za vsako vhodno trojico besednih zvez $\langle subj, pred, obj \rangle$:
    naj bodo S, V in O množice WordNetovih konceptov, ki jih
      lahko povežemo z besednimi zvezami *subj*, *pred* in *obj*
      (s hevristikami iz razdelka 6.2.1), in vseh
      njihovih (posrednih ali neposrednih) nadpomenk;
    za vsako $s \in S, v \in V, o \in O$:
      if $d(s) + d(v) + d(o) \neq D$ **then continue**;
      if obstaja neka nadpomenka $s' \in h(s)$, tako da je $\langle s', v, o \rangle \in F$,
          ali neka nadpomenka $v' \in h(v)$, tako da je $\langle s, v', o \rangle \in F$,
          ali neka nadpomenka $o' \in h(o)$, tako da je $\langle s, v, o' \rangle \in F$:
          ali pa je D =0:
        if $\langle s, v, o \rangle \notin C$
        then dodaj $\langle s, v, o \rangle$ v C s podporo 1
        else povečaj podporo $\langle s, v, o \rangle$ v C za 1;
  iz C skopiraj v F tiste trojice $\langle s, v, o \rangle$, ki imajo podporo vsaj *MinSup*;
  če ni bilo nobene take trojice, se postopek ustavi;
  $D := D + 1;$

Koristno je, če skušamo čim več pogojev preverjati čim bolj zgodaj in tako prihraniti čas, ki bi ga sicer izgubljali s pregledovanjem tistih trojic $\langle s, v, o \rangle$, ki bodisi nimajo prave vsote globin bodisi nimajo možnosti, da bi na koncu dosegle zahtevani prag minimalne podpore. V ločenem prehodu skozi vhodne podatke (še pred izvedbo gornjega algoritma) izračunajmo podporo vseh posameznih konceptov in parov konceptov; potem lahko trojno zanko, ki je zgoraj zapisana preprosto kot „**za vsako** $s \in S, v \in V, o \in O$“, razdelamo takole:

za vsako $s \in S$:
  if $sup(s) < MinSup$ **or** $d(s) > D$
    or $d(s) + \max_{v \in V} d(v) + \max_{o \in O} d(o) < D$ **then continue**;
  za vsako $v \in V$:
    if $sup(s, v) < MinSup$
      or $d(s) + d(v) + \max_{o \in O} d(o) < D$ **then continue**;
    za vsako $o \in O$:
      if $sup(s, o) < MinSup$ **or** $sup(v, o) < MinSup$ **then continue**;

6.2.3 Odkrivanje zanimivih trojic

Postopek iz prejšnjega razdelka odkrije vse trojice (ne glede na njihov nivo abstrakcije), ki dosegajo želeni prag minimalne podpore. Ni pa nujno, da so vse te trojice zanimive za uporabnika (npr. za namene polnjenja ontologije). Med drugim smo na primer že zgoraj videli, da se podpora trojice poveča (ali pa ostane enaka), če kakšno od njenih komponent zamenjamo z njeno nadpomenko, tako da so najpogostejše trojice tiste, ki so najbolj splošne in zato ne preveč koristne (na primer $\langle$*entity, move, entity*$\rangle$).

Zahteva, da mora biti trojica pogosta, je potreben pogoj, ni pa zadosten. V tem razdelku opisujemo mero za „zanimivost“ trojic, ki si pomaga s hierarhijo nadpomenk. Naj bo s nek koncept, s' pa njegova nadpomenka v WordNetu. Vsaka vhodna trojica, katere osebek podpira s, torej podpira tudi s', saj je ta s-jeva nadpomenka. Obratno pa ne drži nujno, saj lahko s' podpirajo tudi nekatere trojice, ki ne podpirajo koncepta s (pač pa kakšno drugo podpomenko koncepta s'). Tako lahko definiramo nekakšno „pogojno verjetnost“ s pri danem s':

$$P(s|s') := sup(s, s')/sup(s') = sup(s)/sup(s'),$$

pri čemer drugi enačaj velja zato, ker je s' nadpomenka s-ja in je zato $sup(s, s') = sup(s)$. Mislimo si zdaj trojico $\langle s, v, o\rangle$, v kateri se s pojavlja kot osebek. Vsaka vhodna trojica, ki podpira $\langle s, v, o\rangle$, podpira tudi malo splošnejšo trojico $\langle s', v, o\rangle$, ker je s' nadpomenka koncepta s. Kot smo videli, lahko verjetnost, da vhodna trojica, ki podpira s', podpira tudi s, ocenimo kot $P(s|s') = sup(s)/sup(s')$. Če torej $\sup(s', v, o)$ vhodnih trojic podpira konceptno trojico $\langle s', v, o\rangle$ in če je $P(s|s')$ delež teh trojic, ki podpirajo tudi koncept s, bi pričakovali, da bo konceptno trojico $\langle s, v, o\rangle$ podpiralo $P(s|s')sup(s', v, o)$ trojic. V resnici pa je podpora te trojice, $\sup(s, v, o)$ lahko drugačna (večja ali pa manjša). Če se $\langle s, v, o\rangle$ pojavlja v vhodnih podatkih precej pogosteje, kot bi pričakovali na podlagi takšne ocene, si lahko to razlagamo kot znak, da je ta trojica zanimiva. Tako smo dobili mero zanimivosti glede na osebek:

$$int_S(s, v, o) := \frac{sup(s, v, o)}{sup(s', v, o)} P(s|s') = \frac{sup(s, v, o)sup(s')}{sup(s', v, o)sup(s)}.$$

Na analogen način lahko definiramo tudi int_V in int_O, pri katerih gledamo nadpomenke drugih dveh komponent trojice, v in o. Vse tri mere lahko združimo v eno:

$$int(s, v, o) := \max\{int_S(s, v, o), int_V(s, v, o), int_O(s, v, o)\}.$$

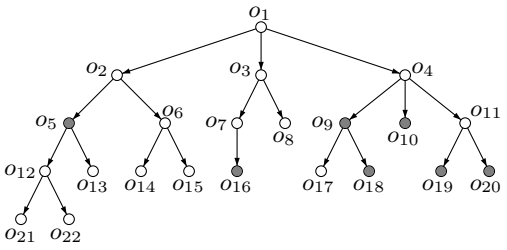
To mero zanimivosti lahko uporabimo za dodatno filtriranje trojic, ki jih bomo pokazali uporabniku kot rezultat naše analize vhodnih podatkov. Preprost in neposreden način bi bil, da bi poleg minimalne podpore predpisali

tudi minimalno zanimivost in vrnili le tiste trojice, ki dosegajo oba pragova. Pri naših preliminarnih poskusih se je pokazalo, da je takšna preprosta in direktna uporaba mere zanimivosti pregroba, zato smo uporabili naslednji postopek filtriranja, ki upošteva skupine sorodnih trojic. Za dano pogosto trojico $\langle s, v, o \rangle$ definirajmo njeno sv-okolico kot množico vseh tistih pogostih trojic, ki se z njo ujemajo v komponentah s in v. Rekli bomo, da je trojica sv-lokalno najzanimivejša, če ima največjo zanimivost izmed vseh trojic v svoji sv-okolici. Na analogen način lahko definiramo tudi so- in vo-okolici in lokalni zanimivosti. S pomočjo teh pojmov lahko definiramo nove kriterije filtriranja trojic: za zanimive štejemo le tiste trojice, ki so lokalno najzanimivejše v dveh ali pa celo vseh treh okolicah (sv, so in vo), lahko pa poleg tega štejemo za lokalno najzanimivejše le tiste trojice, pri katerih je okolica dovolj velika, s čimer se lahko izognemo osamelcem (outlierjem).

6.2.4 Zanimive trojice kot problem pokrivanja v drevesu

Identifikacija zanimivih trojic je zahteven problem in je kot tak primernejši za polavtomatsko kot povsem avtomatsko obdelavo. Predstavljamo si lahko scenarij, ko uporabnik določi dve komponenti trojice, sistem pa naj bi mu povedal, katere so možne vrednosti tretje komponente, pri katerih nastane trojica zanimiva. V preostanku tega razdelka smo predpostavili, da je uporabnik izbral osebek s_0 in povedek v_0, mi pa zdaj iščemo možne vrednosti predmeta o. (V primeru, ko sta fiksirani kakšni drugi dve komponenti, ne pa ravno osebek in povedek, lahko uporabimo povsem analogen razmislek.)

Predmeti v naših trojicah so samostalniki iz WordNeta, ti pa so povezani v drevesasto hierarhijo prek relacije nadpomenka/podpomenka. Predstavljajmo si, da v tem drevesu označimo tiste predmete o, za katere je trojica $\langle s_0, v_0, o \rangle$ prisotna v našem korpusu. Hipotetičen primer tega prikazuje slika 6.2.4:



Slika 6.2.4. Hipotetičen primer delno pokritega drevesa predmetov, ki bi ga dobili, če bi bile v korpusu prisotne (z zadostno podporo) trojice $\langle s_0, v_0, o_5 \rangle$, $\langle s_0, v_0, o_9 \rangle$, $\langle s_0, v_0, o_{10} \rangle$, $\langle s_0, v_0, o_{16} \rangle$, $\langle s_0, v_0, o_{18} \rangle$, $\langle s_0, v_0, o_{19} \rangle$ in $\langle s_0, v_0, o_{20} \rangle$, ne pa tudi trojice $\langle s_0, v_0, o \rangle$ za druge o iz tega drevesa.

Pri stanju, kot ga kaže drevo na sliki 6.2.4, bi si lahko rekli, da se predmet o_4 lepo ujema z osebkom s_0 in glagolom v_0, saj se v kombinaciji z njima pojavljajo mnoge njegove podpomenke (če se že o_4 sam ne). Podobno lahko

rečemo za o_5, ki se celo sam pojavlja skupaj s s_0 in v_0 (če se že njegove podpomenke ne), in za o_{16}. Želeli bi si torej postopek, ki bi obdelal takšno drevo in nam na koncu priporočil, da so zanimive trojice z osebkom s_0 in glagolom v_0 tiste, pri katerih je predmet eden od $\{o_4, o_5, o_{16}\}$. Problem, ki smo si ga zastavili, je torej približno tak: mislimo si, da je (pri izbranih, fiksiranih s_0 in v_0) naš korpus označil nekatere dele drevesa predmetov (namreč vozlišča, ki so na gornji sliki osenčena), mi pa bi radi izbrali neko manjše število vozlišč, ki pokrijejo (s svojimi poddrevesi) čim več označenih vozlišč in čim manj neoznačenih.

Za začetek se je koristno vprašati, kaj sploh mislimo s tem, da je korpus označil neko vozlišče našega drevesa. Če bi rekli preprosto, da je vozlišče o označeno natanko tedaj, ko je trojica $\langle s_0, v_0, o\rangle$ prisotna v našem korpusu (z neko dovolj veliko podporo), bi bilo to neprikladno, saj bi pomenilo, da bodo za vsako označeno vozlišče zagotovo označeni tudi vsi njegovi predniki v drevesu (saj imajo njim pripadajoče trojice zgolj še večjo podporo). To bi bilo nerodno za naš problem pokrivanja drevesa, saj bi bil najboljši način za pokritje tako označenega drevesa ta, da izberemo le korensko vozlišče. Zato bomo definirali vozlišče za označeno malo drugače: rekli bomo, da je o označeno, če ga podpira dovolj takih trojic, pri katerih se o pojavlja neposredno, ne le prek kakšne svoje podpomenke. Kot mero za označenost vozlišča torej vzemimo

$$dsup(o|s_0, v_0) := |\{i \in I : s_0 \in h^\star(m_S(\tilde{s}_i)), v_0 \in h^\star(m_V(\tilde{v}_i)), o \in m_O(\tilde{o}_i)\}|.$$

Če si zdaj ogledamo vozlišče o_4 na sliki 6.2.4, lahko najprej pomislimo, da je dober kriterij za to, kdaj izbrati neko vozlišče, ta, da mora biti v njegovem poddrevesu veliko označenih vozlišč (in malo neoznačenih). Vendar pa tak kriterij ni prikladen za vozlišče o_5 na tisti sliki, saj je označeno le tisto vozlišče samo, njegovi potomci v drevesu pa ne. Pri njem bi bilo torej koristno, če bi rekli, da če je neko vozlišče označeno, veljajo za označene tudi njegovi potomci v drevesu. V ta namen lahko poskusimo direktno podporo *dsup* propagirati od zgoraj navzdol po drevesu (*top-down support*):

$$\begin{aligned} tds(o|s_0, v_0) := \max\{ \quad & dsup(o|s_0, v_0), \\ & \max\{ dsup(o'|s_0, v_0) \tfrac{sup(o)}{sup(o')} : o' \in h(o)\} \}. \end{aligned}$$

Pri tej definiciji si torej predstavljamo, da podpora od vozlišča o' prehaja tudi na njegove potomce o, vendar zmanjšana za faktor $sup(o)/sup(o')$, ki odraža dejstvo, da je o sam po sebi redkejši od splošnejšega pojma o.

Zdaj lahko definiramo še vsoto te podpore po poddrevesu; ta vsota nam pove nekaj o tem, kako dobro je pokrito to poddrevo, bodisi tako, da so vozlišča pokrita neposredno, bodisi zaradi tega, ker so pokriti splošnejši pojmi nad njimi.

$$bus(o|s_0, v_0) := \sum_{o' : o \in h^\star(o')} tds(o'|s_0, v_0).$$

Poleg mer, kot sta *tds* in *bus*, ki merita pokritost poddrevesa, smo v merah za ocenjevanje posameznega vozlišča želeli upoštevati tudi velikost poddrevesa. Ena možnost je, da bi delili s številom vozlišč v poddrevesu; težava pri tem pa je, da je WordNetovo drevo precej neenakomerno v tem, kako temeljito razvejeno je. Ponekod ima nek koncept veliko število podkonceptov, ki pa se v praksi le redko pojavljajo (primer: koncept *domestic cat* ima 16 podkonceptov, večinoma za različne pasme mačk). Namesto preprostega štetja vozlišč lahko posamezna vozlišča utežimo z utežjo, ki je odvisna od podpore konceptov v njih:

$$nw(o) := 1 + \log sup(o),$$

teža poddrevesa pa je potem

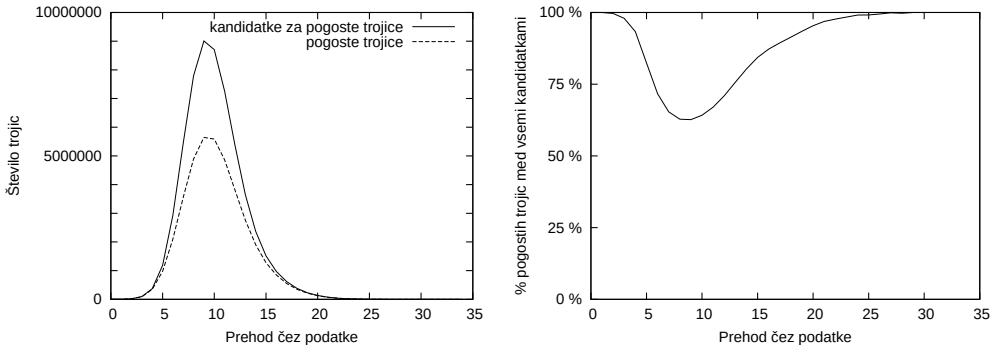
$$sw(o) := \sum_{o':o \in h^\star(o')} nw(o).$$

6.3 Poskusi

Poskuse smo izvedli na zaporedju 15,88 milijona trojic oblike $\langle$*osebek, povedek, predmet*$\rangle$, ki so jih pridobili Rusu in sod. [63] iz korpusa RCV1 (zbirka Reutersovih novic iz let 1996–1997, [61]). Za približno 4,1 milijona teh trojic postopek iz razdelka 6.2.1 ni našel primernih konceptov iz WordNeta (vsaj ne za vse tri komponente trojice; v mnogih primerih so težave na primer povzročala lastna imena, ki jih v WordNetu ni bilo), zato smo jih iz nadaljnje obdelave izločili. Vhod za nadaljnjo obdelavo torej predstavlja seznam približno 11,78 milijona trojic.

6.3.1 Pogoste trojice

Za preizkus algoritma za odkrivanje pogostih posplošenih trojic iz razdelka 6.2.2 smo postavili minimalni prag podpore na 400. Pri tem pragu je glavna zanka našega algoritma izvedla 35 prehodov skozi vhodno zaporedje; nobena trojica z globino 34 ali več ni bila pogosta (torej ni imela podpore vsaj 400). Slika 6.3.1 kaže za vsakega od teh prehodov, koliko se je pri njem nabralo kandidatov (v množici C) in koliko od teh kandidatov se je na koncu prehoda izkazalo za zares pogoste (in smo jih torej prenesli v izhodno množico F). Slika kaže za vsak prehod tudi delež pogostih trojic relativno glede na vse kandidate. Vidimo lahko, da je bil na vsakem prehodu delež pogostih kandidatov vsaj 60 %; kriteriji, ki jih naš algoritem uporablja za čim zgodnejše izločanje neobetavnih kandidatov, se torej dobro obnesejo in količina pomnilnika, porabljena za hranjenje neobetavnih kandidatov (ki se na koncu ne izkažejo za pogoste), je relativno majhna v primerjavi z velikostjo celotne množice pogostih trojic. Skupno število pogostih trojic, ki jih je ta postopek odkril (na vseh stopnjah posplošenosti skupaj), je približno 40,04 milijona.



Slika 6.3.1. Delovanje algoritma za odkrivanje pogostih trojic. Algoritem ob vsakem prehodu čez vhodne podatke identificira množico kandidatk za pogoste trojice, izračuna njihovo podporo in nato zavrže tiste, ki niso dosegle minimalnega praga podpore. Levi graf za vsak prehod čez podatke kaže število nastalih kandidatk in število trojic, ki so se na koncu res izkazale za pogoste. Desni graf kaže odstotek pogostih trojic glede na vse kandidatke. Vidimo lahko, da je delež pogostih trojic med kandidatkami pri vsakem prehodu vsaj 62 %, pri večini prehodov pa je še višji.

Skupni čas izvajanja tega postopka je bil približno 60,8 ur na računalniku z 2-GHz dual core opteronom. Ker imajo nekatere besede v WordNetu veliko možnih pomenov in ker pomanjkanje konteksta v naših vhodnih podatkih onemogoča disambiguacijo, lahko posamezna vhodna trojica podpira veliko število konceptnih trojic, še preden sploh upoštevamo posploševanje po WordNetu.

6.3.2 Zanimive trojice

V tem razdelku opisujemo poskuse z našim pristopom za odkrivanje potencialno zanimivih trojic. Pri tem poskusu smo se osredotočili na trojice, ki so specializacije trojice ⟨*person, inhabit, location*⟩. Takih trojic je bilo 5350, med njimi pa jih je 1321 imelo podporo 20 ali več. Teh 1321 trojic je bilo podlaga našega poskusa.

Vseh 1321 trojic smo ročno pregledali in ocenili njihovo zanimivost po naslednji lestvici, od manj do bolj zanimivih:

(1) *Napačna.* Do tega pride največkrat v primerih, ko si je program neko večpomensko besedo razlagal v pomenu, ki v tistem kontekstu ni pravi. Tako imamo na primer trojico ⟨*Edwin Herbert Land, occupy, location*⟩, ker je v WordNetu navedena beseda "Land" kot ena od možnih oblik koncepta *Edwin Herbert Land*; zato si je postopek lahko pojavitve besede "land" v pomenu *dežela* razlagal kot omembe izumitelja E. H. Landa.

(2) *Zelo pretirano specializirana.* Trojica je načeloma resnična in smiselna, vendar uporablja pojme, ki so precej preveč specifični. Primer: ⟨*litigant, inhabit, location*⟩.

(3) *Pretirano specializirana.* Trojica je smiselna, vendar uporablja pojme,

Opis	Število trojic				Delež
	Vseh	(4)	(5)	(4)+(5)	(4)+(5)
Vse trojice	1321	324	63	387	29,3%
so-lokalno najzanimivejše	5		5	5	100,0%
vo-lokalno najzanimivejše	34	17	1	17	52,9%
sv-lokalno najzanimivejše	98	24	3	27	27,6%
sv- in *vo*-lokalno najzanimivejše	5	2		2	40.0%
sv- ali *vo*- ali *sv*-lokalno najzanimivejše	137	41	9	50	36.5%

Tabela 6.3.2. Prikazan je delež trojic z najvišjimi ročnimi ocenami zanimivosti (4 in 5) v različnih skupinah trojic, izbranih na podlagi avtomatsko izračunane mere zanimivosti iz razdelka 6.2.3. Vrstica z oznako „*sv*- in *vo*-lokalno najzanimivejše“ pomeni trojice, ki so bile hkrati najzanimivejše v obeh okolicah (*sv* in *vo*), zadnja vrstica pa pomeni trojice, ki so bile najzanimivejše v vsaj eni od svojih treh okolic (*sv*, *so* ali *vo*).

ki so verjetno bolj specifični, kot bi bilo treba. Primer: ⟨*man, inhabit, location*⟩, pri čemer je prva komponenta tisti koncept v WordNetu, ki ima pomen *moški*, ne pa *človek*.

(4) *Specializirana, vendar na primeren način.* Trojica uporablja pojme, ki so sicer mogoče bolj specifični, kot bi bilo nujno treba, vendar so te specializacije smiselne glede na ostale komponente trojice. Primera: ⟨*Hungarian, inhabit location*⟩, ⟨*person, inhabit, Cyprus*⟩. Tidve trojici povesta nekaj zanimivega, namreč da izraz “Hungarian” pomeni prebivalce neke lokacije in da izraz “Cyprus” pomeni nekaj, kjer lahko ljudje prebivajo.

(5) *Zanimiva in uporablja primerne koncepte.* V tej skupini so trojice, ki bi jih najraje videli v ontologiji, ker so relevantne in ravno prav abstraktne. Primera: ⟨*inhabitant, inhabit, location*⟩, ⟨*Asian, populate, Asian country*⟩.

V tej lestvici so trojice iz skupin (1), (2) in (3) takšne, ki jih najverjetneje želimo zavreči, skupini (4) in (5) pa sestavljajo trojice, ki jih načeloma štejemo za relevantne.

Nato smo za vseh 1321 trojic izračunali njihovo zanimivost po postopku iz razdelka 6.2.3 in določili lokalno najzanimivejše trojice. Trojice nismo šteli za lokalno najzanimivejšo, če je bilo v njeni okolici manj kot pet trojic. Tabela 6.3.2 kaže povprečje ročne ocene zanimivosti za različne skupine trojic, izbranih na podlagi lokalne zanimivosti. Kot vidimo iz tabele 6.3.2, filtriranje trojic na podlagi kriterija lokalne zanimivosti res zavrže nekatere manj relevantne trojice in poveča delež relevantnih trojic.

6.3.3 Rezultati pristopa s pokrivanjem v drevesu

Pristop s pokrivanjem v drevesu smo preizkusili na treh testnih množicah, ki smo jih pripravili tako, da smo si izbrali nek osebek in povedek ter ročno ocenili trojice, v katerih nastopata ta osebek in povedek ter ki imajo dovolj visoko podporo. Kriteriji pri ročnem ocenjevanju so bili taki, kot smo jih

opisali v prejšnjem razdelku. Trojice z oceno (4) ali (5) smo šteli za pozitivne (relevantne), ostale pa za negativne (irelevantne). Podrobnosti o teh treh testnih množicah so opisane v naslednji tabeli:

			Min.	Število trojic		
	Osebek s_0	Povedek v_0	podpora	poz.	neg.	skupaj
1	person	inhabit, populate	100	28	52	80
2	company	sell	70	24	60	84
3	event	occur	100	65	498	563

Pri testni množici št. 2 smo postavili prag minimalne podpore nižje kot pri ostalih (70 namesto 100), da ne bi bila premajhna (pri pragu 100 bi ostalo le 55 trojic).

S temi ročnimi ocenami relevance nam vsaka od treh testnih množic predstavlja nek dvorazreden klasifikacijski problem. Na teh problemih smo preizkusili različne mere, ki temeljijo na definicijah iz razdelka 6.2.4. Vrednost posamezne mere, recimo ji $f(o|s_0, v_0)$, uporabimo kot oceno relevantnosti trojice (s_0, v_0, o). Ker te napovedi niso zgolj binarne, smo za ocenjevanje napovedi uporabili površino pod krivuljo ROC.

Mere, ki smo jih preizkusili za napovedovanje relevantnosti trojic (s_0, v_0, o), so sestavljene iz dveh delov: en del, ki meri pokritost poddrevesa s korenom pri o (*tds* ali *bus*) in en del, ki je namenjen normalizaciji in je povezan z velikostjo tega poddrevesa (nw = teža vozlišča o; sw = teža poddrevesa s korenom o; nn = število vozlišč v tem poddrevesu).

Kot vidimo iz rezultatov v tabeli 6.3.3, se mere, ki temeljijo na *tds*, obnesejo bolje od tistih, ki temeljijo na *bus*. Običajna podpora trojice, $sup(s_0, v_0, o)$, je dajala slabše rezultate kot direktna podpora *dsup* (in iz slednje izpeljana *tds*). Od raznih poskusov normalizacije ni bilo kakšne vidnejše koristi.

Za konec si oglejmo še primerjavo med pristopom na podlagi lokalne zanimivosti (razdelek 6.2.3) in pristopom na podlagi pokrivanja drevesa (razdelek 6.2.4). Slika 6.3.3 kaže krivuljo ROC za oba pristopa na prvi testni množici (80 pogostih trojic oblike $\langle$*person, inhabit*, $\star\rangle$). V enem primeru smo trojice uredili po tem, v koliko okolicah so lokalno najzanimivejše, znotraj tega pa še po zanimivosti *int*), v drugem pa po meri *tds*/*sw*. Vidimo lahko, da je mera s pokrivanjem (mera z drevesom) dosegla precej boljše napovedi (AuRoC = 0,7497) kot mera na podlagi lokalne zanimivosti (mera z okolicami; AuRoC = 0,6092).

6.4 Zaključki

Opisali smo pristop za odkrivanje pogostih in zanimivih trojic oblike $\langle koncept_1$, *relacija*, $koncept_2\rangle$ iz korpusa besedil. Naši poskusi so pokazali, da zmore opisani pristop uspešno in učinkovito odkriti pogoste trojice in pri tem

		Površina pod ROC		
Mera	Testna množica 1	2	3	Povprečje
tds	0,7919	0,6583	0,5497	0,6666
tds/sw	0,7479	0,6375	0,5144	0,6333
tds/nw	0,7775	0,6604	0,5397	0,6592
tds/nn	0,7404	0,6236	0,5213	0,6284
$tds/sup(o)$	0,7651	**0,6771**	0,4687	0,6370
$tds \cdot nw$	0,7926	0,6549	0,5598	0,6691
$tds \cdot nn$	**0,8111**	0,4917	0,5611	0,6213
$tds \cdot sup(o)$	**0,8207**	0,5243	**0,6721**	**0,6724**
$tds \cdot sw$	**0,8297**	0,5000	0,5892	0,6396
bus	0,7390	0,4278	0,5727	0,5798
bus/sw	0,7438	0,4660	0,5161	0,5753
bus/nn	0,7486	0,4632	0,5246	0,5788
$bus \cdot sw$	0,6793	0,3771	0,5569	0,5378
$bus \cdot nn$	0,6683	0,3875	0,5508	0,5355
$dsup(o\|s_0, v_0)$	0,6188	**0,6979**	0,6048	0,6405
$sup(s_0, v_0, o)$	0,5117	0,5604	**0,7215**	0,5979
$sup(o)$	0,2452	0,3007	0,6400	0,3953

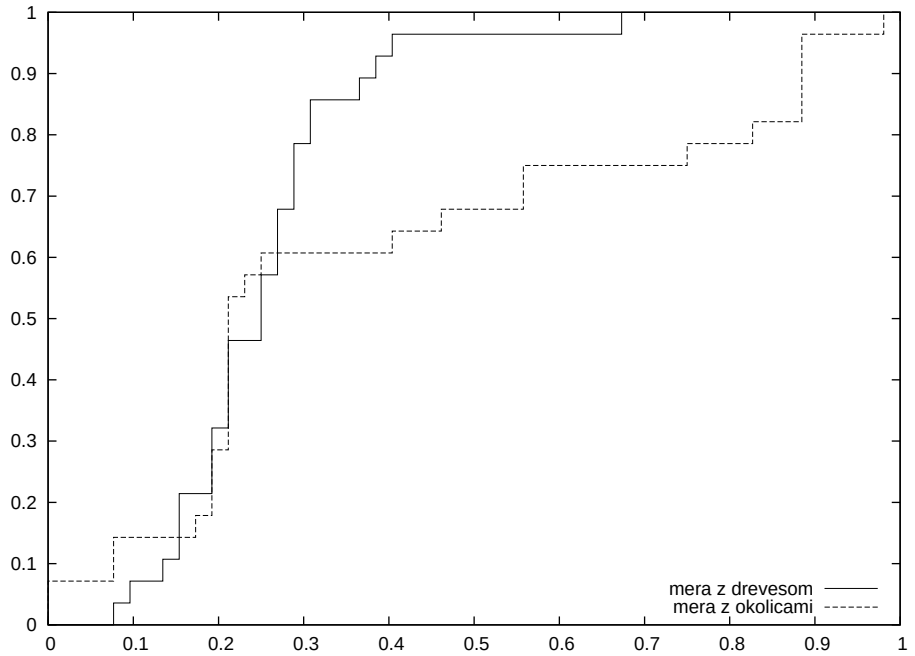
Tabela 6.3.3. Primerjava različnih mer za ocenjevanje zanimivosti na podlagi pristopa s pokrivanjem drevesa (razdelek 6.2.4). Napovedi vsake mere smo primerjali z ročnimi ocenami zanimivosti in jih ovrednotili s površino pod krivuljo ROC.

porabi le malo dodatnega pomnilnika (na primer, pri vsakem prehodu čez vhodne podatke se je vsaj 62 % kandidatk za pogoste trojice izkazalo za dejansko pogoste), tudi v primerih, ko zaradi večpomenskosti in posploševanja posamezna vhodna trojica (ki praviloma ustreza enemu stavku iz vhodnega besedila) podpira veliko število konceptnih trojic.

Opisali smo tudi več mer za avtomatsko ocenjevanje zanimivosti trojic ter definirali pojem lokalno najzanimivejših trojic, ki je lahko podlaga za filtriranje množice pogostih trojic, preden jih pokažemo uporabniku. Poskusi so pokazali, da je ločevanje zanimivih oz. relevantnih trojic od nezanimivih oz. nerelevantnih težak problem, ki ga nobeden od tu opisanih pristopov ne rešuje res dobro, je pa pristop s pokrivanjem drevesa konceptov dajal boljše rezultate od pristopa z lokalno najzanimivejšimi trojicami.

Ena od težav pri odkrivanju zanimivih trojic je pogosto pojavljanje večpomenskih besed v naših vhodnih podatkih, skupaj s pomanjkanjem možnosti za njihovo disambiguacijo. Pričakovali smo, da bodo zaradi velike količine vhodnega besedila na koncu pravilne interpretacije prevladale nad šumom, ki ga povzročajo napačne interpretacije večpomenskih besed, vendar se je izkazalo, da pri naših poskusih do tega ni prišlo in težave zaradi večpomenskosti so pogost vir irelevantnih trojic, ki jih je težko avtomatsko ločiti od relevantnih. Zanimiva smer nadaljnjega dela bi torej bila ponoviti tovrstne poskuse na kakšnem že disambiguiranem korpusu.

Videli smo, da lahko problem odkrivanja zanimivih trojic formuliramo



Slika 6.3.3. Krivulji ROC dveh pristopov za ocenjevanje zanimivosti trojic. Med trojicami oblike $\langle person,\ inhabit,\ \star \rangle$ je 80 takih, ki imajo podporo vsaj 100; te trojice smo ročno klasificirali na zanimive in nezanimive ter delovanje obeh pristopov primerjali s tako dobljenimi ročnimi oznakami.

kot problem pokrivanja, kar smo potem v tem poglavju skušali reševati z različnimi merami za pokritost posameznih delov drevesa konceptov (razdelek 6.2.4). Alternativni pogled, ki bi ga bilo tudi zanimivo preizkusiti, pa bi bil, da bi se zgledovali po algoritmih pokrivanja, ki se uporabljajo pri učenju klasifikacijskih pravil. V tem primeru bi pokrita vozlišča videli kot pozitivne primere, ostala vozlišča kot negativne, iskali pa bi nabor poddreves, ki vsebujejo čim več pokritih vozlišč in čim manj ostalih.

Poglavje 7

Zaključek

7.1 Prostor kodnih matrik

Večrazredni klasifikacijski problem lahko rešujemo tako, da ga prevedemo na več dvorazrednih problemov, pri čemer je v vsakem od njih pozitivni razred sestavljen iz nekaj razredov prvotnega problema, negativni pa iz nekaj drugih razredov prvotnega problema. To prevedbo lahko opišemo s kodno matriko, ki vsebuje po eno vrstico za vsak razred prvotnega problema in po en stolpec za vsakega od novih dvorazrednih problemov; posamezni element matrike pove, kako je bil nek razred prvotnega problema uporabljen pri učenju nekega dvorazrednega klasifikatorja. Za vsakega od dvorazrednih problemov zgradimo klasifikacijski model in tako dobimo ansambel klasifikatorjev. Končno napoved za posamezen primer dobimo tako, da primerjamo vektor napovedi dvorazrednih klasifikatorjev z vrsticami kodne matrike in pogledamo, pri kateri vrstici (torej za kateri razred prvotnega problema) je bila dosežena največja podobnost.

Prostor vseh možnih kodnih matrik je zelo velik. Če je hierarhija razredov v našem večrazrednem klasifikacijskem problemu dovolj pravilno zgrajena, lahko število možnih različnih stolpcev matrike izračunamo analitično, prek tega pa izračunamo tudi število vseh matrik določene širine. Če je število razredov v našem prvotnem večrazrednem problemu dovolj majhno, je prostor kodnih matrik še obvladljiv in lahko sistematično pregledamo in ocenimo vse matrike do neke želene širine. (Pri tem „ocena matrike“ pomeni v resnici oceno pravilnosti napovedi ansambla klasifikatorjev, ki izhajajo iz te matrike.) S temi poskusi smo pokazali, da obstajajo dobre matrike (torej take, katerih ansambel doseže visoko oceno) tudi pri majhnem številu stolpcev (torej z majhnim številom dvorazrednih klasifikatorjev v ansamblu), le da je delež takšnih matrik manjši kot pri matrikah z več stolpci. Ugotovili smo, da ima porazdelitev ocen vseh možnih matrik določene širine približno obliko beta-porazdelitve, pri čemer vrednosti parametrov α in β te porazdelitve naraščata približno kot eksponentni funkciji v odvisnosti od števila

stolpcev matrike.

Zanimivo vprašanje pri kodnih matrikah je, ali lahko dobre matrike (dobre v smislu klasifikacije) prepoznamo po kakšnih statističnih ali algebraičnih lastnostih, ki bi se jih dalo dovolj poceni in enostavno računati in ki bi nam bile zato mogoče lahko tudi v pomoč pri sestavljanju primernih matrik. Primer take lastnosti je čim večja povprečna Hammingova razdalja med vrsticami (ali pa med stolpci) matrike. Kot smo videli pri naših poskusih, so matrike, pri katerih je ta povprečna razdalja večja (torej so si stolpci oz. vrstice v povprečju bolj različni), praviloma res tudi uspešnejše pri klasifikaciji. Vendar pa ta povezava ni povsem monotona; ansambli, ki dosežejo najvišjo oceno pri klasifikaciji, imajo sicer veliko povprečno Hammingovo razdaljo med vrsticami oz. stolpci, vendar ne prav najvišje, in obratno.

Možnosti za nadaljnje delo je v zvezi s temi vprašanji precej. Poskuse bi bilo zanimivo ponoviti na kakšni večji domeni, kjer pa si ne bi mogli več privoščiti izčrpnega pregledovanja vseh možnih matrik, saj bi jih bilo že neobvladljivo veliko, tako da bi se morali opirati na pregledovanje neke manjše množice naključno izbranih matrik.

Še eno zanimivo vprašanje za nadaljnje delo je, ali mere, kot je povprečna Hammingova razdalja med stolpci, postanejo boljše vodilo za tvorbo matrik, če v njih vključimo tudi kakšno znanje o ontologiji, ki stoji za matriko. Na primer, nek stolpec je sicer lahko za matriko koristen z vidika povprečne Hammingove razdalje med stolpci, vendar se mogoče izkaže, da je dvorazredni klasifikacijski problem, ki ga ta stolpec definira, pretežak in nam bo ta stolpec v naš ansambel prispeval klasifikator z veliko napakami. Vendar pa smo pri dosedanjih poskusih videli, da klasifikacijska ocena matrike kot celote (oz. ansambla, dobljenega iz nje) ni korelirana s povprečno oceno posameznih dvorazrednih klasifikatorjev v njenem ansamblu.

Videli smo tudi, da je prostor kodnih matrik načeloma primeren za preiskovanje s požrešnim algoritmom, pri katerem se na vsakem koraku doda v matriko nov stolpec tako, da se ocena celotnega ansambla pri tem čim bolj poveča. Pri poskusih z majhnimi matrikami se je pokazalo, da so matrike, dobljene s požrešnim postopkom, boljše od naključno zgrajenih matrik in to tem bolj, čim večji delež matrike je bil zgrajen s požrešnim postopkom. Zanimiva smer nadaljnjega dela bi bila, da bi namesto požrešnega algoritma uporabili kakšen drug algoritem za preiskovanje prostora oz. optimizacijo, na primer simulirano ohlajanje ali genetske algoritme. To bi bilo še posebej koristno v luči opažanja iz 3. poglavja, da je gradnja matrike po stolpcih pravzaprav precej problematična (glej spodaj).

7.2 Postopna gradnja kodnih matrik

Eden od pristopov za tvorbo kodne matrike je, da vsebino matrike zapolnimo naključno. V našem delu smo postopek za tvorbo naključnih kodnih matrik

prilagodili za probleme, pri katerih so razredi organizirani v hierarhijo, kar predstavlja dodatne omejitve za vsebino kodne matrike. Nato smo razvili požrešni pristop, ki kodno matriko tvori postopoma in vanjo dodaja stolpce enega po enega. Tudi pri poskusih na zmerno veliki ontologiji smo videli, da postopna gradnja matrike s požrešnim algoritmom načeloma daje boljše rezultate kot popolnoma naključna gradnja matrike (oz. doseže enako dobre rezultate pri manjšem številu stolpcev).

Razvili smo postopek za uteževanje modelov v ansamblu, ki lahko za posamezni model učinkovito poišče tako vrednost uteži, ki maksimizira uspeh ansambla na validacijski množici.

Pokazali smo tudi na pomembno težavo, na katero naletimo pri postopni gradnji matrike. Zaradi načina, kako se pri uporabi ansambla kombinirajo napovedi posameznih modelov v ansamblu, je vpliv posameznega modela na napovedi celotnega ansambla majhen (celo če dopustimo, da imajo različni modeli različno težo). Ker se po dodajanju novega stolpca napovedi ansambla ne spremenijo dovolj, se zelo lahko zgodi, da bo požrešen postopek za postopno gradnjo matrike tudi v naslednjem koraku sestavil zelo podoben stolpec in tako naprej. S tem pridemo do matrik, ki imajo preveč podobnih stolpcev, kar je neugodno za uspeh ansambla kot celote.

V luči teh ugotovitev se vsekakor odpira več smeri za nadaljnje delo. Ideja o čisto postopni in požrešni gradnji matrike stolpec za stolpcem se je izkazala za problematično; lahko bi razmišljali o hibridnem pristopu, ki matriko sicer gradi postopno, vendar po več stolpcev naenkrat in ob tem lažje pazi na to, da si stolpci med seboj niso preveč podobni. Še ena možnost je, da v matriko sicer dodajamo stolpce enega po enega, vendar v hevristiki za izbor novega stolpca tako ali drugače kot dodaten kriterij upoštevamo še Hammingovo razdaljo od novega stolpca do dosedanjih stolpcev matrike.

V drugem poglavju, pri poskusih na zelo majhni ontologiji, smo si pri požrešnem pregledovanju prostora matrik lahko privoščili, da smo na vsakem koraku pregledali vse možne stolpce, ki bi jih lahko dodali v matriko; pri večji ontologiji je možnih stolpcev neobvladljivo mnogo, zato se pojavi pomembno vprašanje, kako naj si požrešni algoritem izbere naslednji stolpec.

Opisali in preizkusili smo dve hevristiki za izbor naslednjega stolpca. Ena hevristika temelji na opazovanju matrike zamenjav: naslednji stolpec kodne matrike (in s tem naslednji klasifikator v ansamblu) poskusi razmejiti nekaj tistih parov razredov, ki jih je dosedanji ansambel dvorazrednih klasifikatorjev največkrat zamešal med sabo. Druga hevristika temelji na opažanju, da so kot napovedi ansambla na posameznem primeru možni največ trije različni razredi, zato lahko skušamo novi stolpec sestaviti tako, da bi imel z njim dopolnjeni ansambel vsaj načeloma možnost napovedati pravi razred pri čim večjem številu učnih primerov.

Zanimiva smer nadaljnjega dela bi torej bila tudi ta, da na problem izbora naslednjega stolpca pogledamo kot na optimizacijski problem. Podobno kot smo prej na gradnjo celotne matrike gledali kot na problem prei-

skovanja celotnega prostora vseh možnih matrik (do neke širine), lahko tudi na izbor enega samega naslednjega stolpce gledamo kot na problem iskanja po prostoru vseh možnih stolpcev in uporabimo zanj razne znane postopke za optimizacijo in preiskovanje prostora.

7.3 Strukturne spremembe v ontologijah

Pri ontologijah kot konceptualizacijah nekega problemskega področja se lahko pojavijo potrebe po tem, da se ontologija skozi čas spreminja, na primer zaradi sprememb na področju, ki ga ontologija pokriva. V našem delu smo analizirali, kako se je skozi čas razvijala tematska ontologija ODP (*Open Directory Project*, `dmoz.org`), ki jo sestavljajo hierarhično organizirane tematike in kazalci na dokumente (zunanje spletne strani). Ta ontologija se ves čas po malem spreminja, saj jo urejajo, spreminjajo in dopolnjujejo številni človeški uporabniki oz. uredniki.

Pri naši analizi smo se osredotočili na spremembe na nivoju posameznih kategorij v ontologiji. S primerjavo stanja ontologije v dveh zaporednih časovnih trenutkih lahko identificiramo primere elementarnih strukturnih sprememb: dodajanja in brisanja posameznih kategorij. Poleg njih smo definirali več vrst višjenivojskih strukturnih sprememb: preimenovanje kategorije, premik kategorije (tako da postane podkategorija nekega drugega starša kot prej), zlitje kategorij (pri čemer se ena kategorija zlije v drugo, slednja pa prevzame tudi njene podkategorije) ter dodajanje kategorij.

Poskusi so pokazali, da so poleg preimenovanj najpogostejša vrsta sprememb dodajanja, zato smo se v nadaljevanju osredotočili nanje. Dodajanja lahko razdelimo na več vrst: dodajanje praznih kategorij; dodajanje novega večnivojskega poddrevesa (ko obstajajo primeri novih kategorij, ki imajo tudi za starša neko novo kategorijo); dodajanje kategorije s popolnoma novimi dokumenti (ki jih v starem stanju ontologije sploh še ni bilo); dodajanje kategorij z razcepom neke stare kategorije. Slednja vrsta dodajanj je najpogostejša in zajema primere, ko nova kategorija postane otrok neke obstoječe kategorije in se vanjo premakne nekaj dokumentov iz te obstoječe kategorije.

Problem napovedovanja te vrste strukturnih sprememb v ontologiji (torej dodajanja nove kategorije z razcepom neke obstoječe kategorije) smo obravnavali kot problem strojnega učenja. Naš pristop temelji na zamisli, da do takšne cepitve obstoječe kategorije pride, če obstaja v njej skupina dokumentov, ki tvorijo „gručo“, torej so si tematsko med seboj bližje kot do drugih dokumentov v obstoječi kategoriji. Dokumente obstoječe kategorije razvrstimo v določeno število skupin (clusters) in dobljeno razbitje opišemo z nekaj atributi, ki se še posebej nanašajo na najtesneje povezano skupino v razbitju. Tako dobimo vektor atributov, ki opisuje kategorijo in je lahko podlaga za uporabo v strojnem učenju, na primer z metodo podpornih vektorjev.

Videli smo, da je pri napovedovanju strukturnih sprememb v ontologiji, kot je ODP, problematično zlasti to, da iz tega, da do neke spremembe ni prišlo, še ne smemo sklepati, da ne bi bila umestna, saj je mogoče, da je uredniki preprosto še niso opazili oz. pomislili nanjo. Ker je možnih mest, kjer bi do dodajanja podkoncepta načeloma lahko prišlo, zelo veliko, dejansko dodanih podkonceptov pa je v primerjavi s tem relativno malo, imamo opraviti s klasifikacijskim problemom, pri katerem je pozitivnih primerov izrazito malo v primerjavi z negativnimi.

Opisani pristop smo preizkusili na podatkih iz ontologije ODP. Dobljeni učni primer se izkaže za težavnega, še posebej zato, ker je delež pozitivnih primerov zelo majhen (v kateremkoli opazovanem trenutku velika večina kategorij ne bo predmet cepitve in dodajanja nove podkategorije). Če naš klasifikator ocenjujemo s kakšno od mer, ki niso odvisne od razmerja v velikosti pozitivnega in negativnega razreda (na primer površina pod krivuljo ROC), dosega dobre rezultate, v praksi pa bi bila problematična predvsem nizka natančnost (*precision*) in neugodno veliko število lažnih pozitivnih napovedi.

Zanimivo vprašanje za nadaljnje delo je, ali lahko podobno kot doslej dodajanje napovedujemo tudi kakšne druge strukturne spremembe, na primer združitve ali premike konceptov. Če imamo dokumente predstavljene z vektorji, vsakega od konceptov pa s centroidom vseh dokumentov v njem, se lahko vprašamo, ali so dokumenti nekega koncepta bližje centroidu kakšnega drugega koncepta, in če je to res, bi lahko napovedali premik.

Predlagani postopek za napovedovanje dodajanja podkonceptov bi se dalo tudi dopolniti s tem, da bi preskočil tiste skupine oz. gruče dokumentov, ki se lepo ujemajo s kakšnim od že obstoječih podkonceptov.

Če pa se vrnemo na izhodišče, da je ontologija skupna, deljena konceptualizacija nekega problemskega področja, si moramo predstavljati, da strukturne spremembe v ontologiji praviloma bodisi pomenijo vnos novega znanja o problemskem področju v ontologijo bodisi odražajo to, da so se spremenile potrebe uporabnikov ontologije ali pa njihov pogled na problemsko področje. V vsakem primeru vidimo, da gre za stvari, ki so z vidika ontologije izrazito zunanje, zato je ideja o napovedovanju strukturnih sprememb ontologije zgolj iz podatkov, ki so že v tej ontologiji, inherentno problematična. Pri nadaljnjem delu bi bilo torej koristno v napovedovanje strukturnih sprememb vključiti dodatne zunanje vire podatkov. Če bi na primer imeli tok dokumentov, ki na novo prihajajo v ontologijo ODP v času od enega posnetka do naslednjega, bi lahko iskali primere, ko se več novih dokumentov ne prilega dovolj dobro nobeni od obstoječih kategorij, kar lahko sproži dodajanje popolnoma nove kategorije (za razliko od cepitve obstoječe kategorije, s čimer smo se ukvarjali doslej).

7.4 Primerjanje in vrednotenje ontologij

Pri vrednotenju ali evaluaciji ontologij nas zanima, kako dobra je dana ontologija v luči določenih standardov ali zahtev. Sestavili smo tipologijo številnih pristopov, ki so se v literaturi uporabljali za primerjanje in vrednotenje ontologij: primerjava z zlatim standardom; vrednotenje na podlagi uporabe v aplikaciji; vrednotenje s pomočjo zunanjih podatkov o problemskem področju ontologije; in pristopi, pri katerih ontologijo ocenjujejo človeški eksperti. Posamezni pristopi za ocenjevanje ontologij se ločijo tudi po tem, na kateri vidik ontologije se osredotočajo: koncepte, hierarhijo, relacije, sintakso, vključenost hierarhije v nek zunanji kontekst. Pokazali smo tudi, kako lahko evaluacijo ontologij vključimo v formalne definicije ontologij, kot so poznane iz literature [19].

Razvili smo svoj pristop, namenjen vrednotenju ontologij na podlagi primerjave z zlatim standardom. Naš pristop se omejuje na primere, ko je ontologija sestavljena iz množice dokumentov, razvrščenih v hierarhijo. Definirali smo mero podobnosti OntoRand za primerjanje dveh različnih hierarhičnih razbitij iste množice dokumentov; lahko si jo predstavljamo kot razširitev Randovega indeksa na hierarhična razbitja. Ideja naše mere je, da če za dva dokumenta pogledamo, v katerih konceptih neke hierarhije ležita in kakšna je razdalja med tema dvema konceptoma, potem sta si dve hierarhiji (dve hierarhični razbitji iste množice dokumentov) tem bolj podobni, čim bolj podobne so tako dobljene razdalje pri eni in pri drugi hierarhiji.

Tako dobljeno mero podobnosti smo preizkusili na realnih podatkih iz ontologije ODP. Na delu ontologije smo izvajali različne vrste sprememb in merili podobnost med prvotno ontologijo in ontologijo po spremembi: brisanje spodnjih nivojev ontologije; zamenjava koncepta z njegovim staršem (rotacija); prerazporejanje instanc med koncepte (na podlagi njihove tekstovne vsebine). Videli smo lahko, da je ta odziv sicer v pravi smeri, vendar je v nekaterih primerih prešibek. Zanimivo vprašanje za nadaljnje delo bi bilo, kako se indeks OntoRand odziva še na nekatere druge vrste sprememb v ontologiji (na primer cepitev obstoječega koncepta na več podkonceptov; ali pa združitev oz. zlitje dveh konceptov, ki imata skupnega starša) in kako ga prilagoditi, da bo nekatere spremembe (na primer rotacije oz. zamenjave koncepta s staršem) zaznaval močneje kot doslej. Ena možna posplošitev je na primer ta, da ko merimo dolžino poti po hierarhiji, ne obravnavamo vseh korakov kot enako dolgih, ampak upoštevamo bodisi trenutno globino v hierarhiji bodisi razdaljo med centroidi konceptov, med katerimi se premikamo (ob predpostavki, da smo koncepte preslikali v nek primeren prostor, na primer v vektorski prostor z vrečami besed, če so instance naše ontologije dokumenti).

Še ena omejitev našega sedanjega indeksa OntoRand je njegova predpostavka, da imata opazovani ontologiji poleg konceptov tudi instance in to obe enako množico instanc. Indeks bi vsekakor postal širše uporaben, če bi

to omejitev omilili. En pristop bi bil na primer ta, da predpostavimo obstoj neke mere podobnosti med instancami ene in druge ontologije. Potem lahko za vsako instanco ene ontologije poiščemo eno ali več najpodobnejših instanc druge ontologije; ko bi se potem pri računanju mere podobnosti med ontologijama U in V ukvarjali z instancama o_i in o_j iz U, bi primerjali razdaljo δ_U med njima v U z vrednostjo δ_V za razne pare njima najbolj podobnih instanc v V. Še ena možnost bi bila, da iz vektorske predstavitve instanc (npr. če so le-te dokumenti) izračunamo centroide konceptov in našo mero računamo po parih konceptov namesto instanc: za vsak par konceptov $U_1, U_2 \in U$ in vsak par $V_1, V_2 \in V$ bi imeli člen $|d_U(U_1, U_2) - d_V(V_1, V_2)|$, utežen s podobnostjo med U_1 in V_1, podobnostjo med U_2 in V_2 ter velikostjo (številom instanc v poddrevesih) teh konceptov.

7.5 Odkrivanje zanimivih trojic v besedilu

Razvili smo pristop, katerega cilj je iz korpusa besedil ekstrahirati zanimive relacijske trojice oblike $\langle koncept_1,\ relacija,\ koncept_2 \rangle$, z namenom, da bi tako pomagali pri polnjenju splošnonamenske ontologije ali baze znanja, kot je na primer Cyc [38]. Osnovna ideja našega pristopa je, da so trojice, ki bi bile zanimive za takšno ontologijo in ki bi bile lahko koristne pri sklepanju v njej, praviloma sestavljene iz abstraktnih konceptov in se zato v običajnih besedilih ne pojavljajo neposredno; pač pa se v besedilih pojavljajo bolj konkretne trojice, iz katerih lahko z abstrakcijo (zamenjavo pojmov z njihovimi nadpomenkami) dobimo primerno abstraktne trojice.

Naša metoda temelji na tem, da iz besedila ekstrahira konkretne relacijske trojice, nato jih z uporabo podatkov o nadpomenkah predela v bolj abstraktne in iz njih na koncu naredi izbor zanimivih trojic. Razvili smo učinkovit postopek za sistematično odkrivanje vseh posplošenih (abstraktnih) trojic, ki presegajo nek predpisan prag minimalne podpore (pogostosti v korpusu), in to na na poljubni stopnji abstrakcije. Naš postopek se zgleduje po algoritmu Apriori [1], naredi več prehodov čez podatke in sistematično našteva trojice od bolj abstraktnih k bolj konkretnim. Na vsakem prehodu generira kandidatke za pogoste trojice na določeni stopnji abstrakcije in se pri tem sproti izogiba kandidatkam, za katere je mogoče iz dosedanjih rezultatov že vnaprej sklepati, da ne bodo presegle zahtevanega praga podpore. S poskusi na obsežnem korpusu besedil iz zbirke Reutersovih novic [61] smo pokazali, da je naš postopek učinkovit, še posebej v tem smislu, da se pri vsakem prehodu več kot polovica kandidatk izkaže za dejansko pogoste, torej je algoritem varčen pri porabi prostora.

Predlagali smo tudi več hevristik za identifikacijo potencialno zanimivih trojic. Ena hevristika temelji na tem, da v trojici eno od komponent posplošimo in pogledamo, kako to vpliva na podporo trojice. Druga hevristika temelji na ideji pokrivanja drevesa: če fiksiramo dve komponenti trojice in si

predstavljamo drevo nad- in podpomenk za možne vrednosti tretje komponente, si lahko predstavljamo, da korpus besedil pokrije (s trojicami, ki se v njem dejansko pojavljajo) nekatera vozlišča tega drevesa; mi pa poskušamo izbrati neko majhno število takih vozlišč, da se unija njihovih poddreves čim lepše ujema s pokritimi deli drevesa. Obe hevristiki smo ocenili tako, da smo njune rezultate primerjali z ročno ocenjenimi podatki o zanimivosti trojic.

Vprašanje, ki nas je pri tem predvsem zanimalo, je, ali lahko zanimive trojice prepoznamo zgolj na podlagi opazovanja pogostosti raznih trojic v korpusu besedil. Kot smo videli, je glavna težava takšnega odkrivanja zanimivih trojic v tem, da je možnih pogostih trojic zelo veliko in če hočemo pri napovedovanju doseči visok priklic, dobimo veliko lažnih pozitivnih primerov in s tem nižjo natančnost.

Ena od možnih smeri nadaljnjega dela je, da bi na besedilu pred nadaljnjo obdelavo izvedli disambiguacijo, ker se je pri dosedanjih poskusih izkazalo, da je večpomenskost nekaterih besed pomemben vir trojic, ki se na videz sicer zdijo pogoste, po vsebini pa niso relevantne. Pričakovali smo, da bodo zaradi velike količine vhodnega besedila na koncu pravilne interpretacije prevladale nad šumom, ki ga povzročajo napačne interpretacije večpomenskih besed, vendar se je izkazalo, da pri naših poskusih do tega ni prišlo in težave zaradi večpomenskosti so pogost vir irelevantnih trojic, ki jih je težko avtomatsko ločiti od relevantnih.

Videli smo, da lahko problem odkrivanja zanimivih trojic formuliramo kot problem pokrivanja točk v drevesu. Zanimiva smer nadaljnjega dela bi bila, da bi se zgledovali po algoritmih pokrivanja, ki se uporabljajo pri učenju klasifikacijskih pravil. V tem primeru bi pokrita vozlišča videli kot pozitivne primere, ostala vozlišča kot negativne, iskali pa bi nabor poddreves, ki vsebujejo čim več pokritih vozlišč in čim manj ostalih.

Literatura

[1] R. Agrawal, R. Srikant. Fast algorithms for mining association rules in large databases. *Proc. 20th International Conference on Very Large Data Bases*, Santiago, Chile, 1994, pp. 487–499.

[2] E. L. Allwein, R. E. Schapire, Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141 (2000).

[3] A. Berger. Error-correcting output coding for text classification. IJCAI*'99: Workshop on machine learning for information filtering*, Stockholm, 1999.

[4] S. Bloehdorn, P. Haase, Y. Sure, J. Voelker, M. Bevk, K. Bontcheva, I. Roberts. *Report on the integration of ML, HLT and OM.* SEKT Deliverable D.6.6.1, July 2005.

[5] A. Bordes, L. Bottou, P. Gallinari, J. Weston. Solving multiclass support vector machines with LaRank. In: *Proceedings of the 24th International Conference on Machine Learning*, ICML 2007, pp. 89–97.

[6] A. Bordes, S. Ertekin, J. Weston, L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579-1619, September 2005.

[7] B. E. Boser, I. M. Guyon, V. N. Vapnik. A training algorithm for optimal margin classifiers. *Proc. of the 5th* ACM *Workshop on Computational Learning Theory*, 1992.

[8] J. Brank, M. Grobelnik, D. Mladenić. A survey of ontology evaluation techniques. *Conference on Data Mining and Data Warehouses* (SiKDD), Ljubljana, October 2005.

[9] J. Brank, M. Grobelnik, D. Mladenić. Automatic evaluation of ontologies. In: A. Kao, S. Poteet (eds.), *Text Mining and Natural Language Processing*. Berlin: Springer, 2006.

[10] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140 (2006).

[11] C. Brewster, H. Alani, S. Dasmahapatra, Y. Wilks. Data driven ontology evaluation. In: *Proceedings of the Int. Conf. on Language Resources and Evaluation*, Lisbon, Portugal, 26–28 May 2004.

[12] A. Burton-Jones, V. C. Storey, V. Sugumaran, P. Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data and Knowledge Engineering*, 55(1):84–102 (2004).

[13] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom. Change detection in hierarchically structured information. In: *Proc. of the* ACM SIGMOD *Conference*, 1996, pp. 493–504.

[14] P. Cimiano, J. Völker. Text2onto — a framework for ontology learning and data-driven change discovery. *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems* (NLDB 2005).

[15] C. Cortes, V. Vapnik. Support-vector networks. *Machine Learning* 20(3):273–297, September 1995.

[16] O. Dekel, Y. Singer. Multiclass learning by probabilistic embeddings. *Advances in Neural Information Processing Systems 15*, NIPS 2002, pp. 945–952.

[17] T. G. Dietterich, G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.

[18] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, J. Sachs. Swoogle: A search and metadata engine for the semantic web. In: *Proc.* ACM *Conf. on Information and Knowledge Management*, 2004, pp. 652–659.

[19] M. Ehrig, P. Haase, M. Hefke, N. Stojanovic. Similarity for ontologies — a comprehensive framework. In: *Proc. 13th European Conf. on Information Systems*, 2005.

[20] G. Flouris, D. Plexousakis, G. Anto. A classification of ontology change. In: *Proceedings of* SWAP *2006, the 3rd Italian Semantic Web Workshop*, Pisa, Italy, December 18–20, 2006.

[21] M. S. Fox, M. Barbuceanu, M. Gruninger, J. Lin. An organization ontology for enterprise modelling. In: M. Prietula *et al.* (eds.), *Simulating organizations: Computational models of institutions and groups*, AAAI/MIT Press, 1998, pp. 131–152.

[22] Y. Freund, R. E. Schapire. Experiments with a new boosting algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*, 2006, Bari, Italy, pp. 148–156.

[23] P. Galinier, A. Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, September 2006.

[24] R. Ghani. *Using error-correcting codes for efficient text classification with a large number of categories*. M. Sc. Thesis, Carnegie Mellon University, 2001.

[25] A. Gómez-Pérez. *Some ideas and examples to evaluate ontologies*. Knowledge Systems Laboratory, Stanford University, 1994.

[26] A. Gómez-Pérez. Towards a framework to verify knowledge sharing technology. *Expert Systems with Applications*, 11(4):519–529 (1996).

[27] M. Grobelnik, P. Cimiano, E. Gaussier, P. Buitelaar, B. Novak, J. Brank, M. Sintek. *Evaluating Ontology Learning and Population from Text: Task Description for* PASCAL *Challenge*. Tech. Rept., 2006.

[28] M. Grobelnik, D. Mladenić. Automated knowledge discovery in advanced knowledge management. *Journal of Knowledge Management*, 9(5):132–149 (2005).

[29] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993.

[30] N. Guarino, C. Welty. Evaluating ontological decisions with OntoClean. *Communications of the* ACM, 45(2):61–65, February 2002.

[31] P. Haase, Y. Sure, J. Völker. Management of dynamic knowledge. *Journal of Knowledge Management*, 9(5):97–107, 2005.

[32] J. Hartmann, P. Spyns, A. Giboin, D. Maynard, R. Cuel, M. C. Suárez-Figueroa, Y. Sure. *Methods for ontology evaluation*. KnowledgeWeb Deliverable D1.2.3, January 2005.

[33] D. Haussler. *Convolution kernels on discrete structures*. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

[34] T. Joachims. Making large-scale support vector machine learning practical. In: B. Schölkopf, C. J. C. Burges, A. J. Smola (*eds.*), *Advances in Kernel Methods: Support Vector Learning*, MIT Press, 1998, pp. 169–184.

[35] T. Joachims. Optimizing search engines using clickthrough data. *Proceedings of the* ACM *Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, 2002.

[36] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Proceedings of the 10th European Conference on Machine Learning* (ECML-98), Chemnitz, Germany, April 21–23, 1998, pp. 137–42.

[37] I. Kononenko. *Strojno učenje*. Ljubljana: Založba FE in FRI, 2005.

[38] D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Commmunications of the* ACM 38:11, November 1995.

[39] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10(8):707–710 (1966).

[40] D. D. Lewis. *Representation and Learning in Information Retrieval*. Ph. D. Thesis, Univ. of Massachusetts, Amherst, MA, USA, 1991.

[41] A. Lozano-Tello, A. Gómez-Pérez. Ontometric: A method to choose the appropriate ontology. *Journal of Database Management*, 15(2):1–18 (2004).

[42] A. Maedche, B. Motik, L. Stojanovic, R. Studer, R. Volz. Ontologies for Enterprise Knowledge Management. IEEE *Intelligent Systems*, January/February 2003.

[43] A. Maedche, S. Staab. Measuring similarity between ontologies. In: *Proc. of the 13th Conf. on Information and Knowledge Management* (2002). LNAI vol. 2473.

[44] E. Malaguti, P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34 (January 2010).

[45] D. Maynard, W. Peters, M. d'Aquin, M. Sabou, N. Aswani. *Dynamics of Metadata*. Deliverable 1.5.1, NeOn Project (EU IST-2005-027595). March 30, 2007.

[46] M. Meila. Comparing clusterings by the variation of information. In: *Proc. of the 16th Annual Conference on Computational Learning Theory*, 2003.

[47] M. Meila. Comparing clusterings — an axiomatic view. In: *Proc. of the Int. Conference on Machine Learning*, 2005.

[48] T. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997.

[49] D. Mladenić. *Machine Learning on non-homogeneous, distributed text data*. Ph. D. thesis, University of Ljubljana, 1998.

[50] D. Mladenić. Text mining — machine learning on documents. In: John Wang (ed.), *Encyclopedia of data warehousing and mining*. Hershey [etc.]: Idea Group Reference, 2006, pp. 1109–1112.

[51] D. Mladenić, M. Grobelnik. Feature selection on hierarchy of web documents. *Journal of Decision support systems*, 35:45–87 (2003).

[52] D. Mladenić, M. Grobelnik. Mapping documents onto web page ontology. *Proceedings of the First European Web Mining Forum*, pp. 77–96. Berlin: Springer, 2004.

[53] L. Page, S. Brin, R. Motwani, T. Winograd. *The PageRank citation ranking: Bringing order to the web*. January 29, 1998. Published 11th November 1999, Digital Libraries Project report SIDL-WP-1999-0120, Stanford University.

[54] C. Patel, K. Supekar, Y. Lee, E. K. Park. OntoKhoj: a semantic web portal for ontology searching, ranking and classification. In: *Proc. of the 5th* ACM *Workshop on Web Information and Data Management*, New Orleans, USA, 2004.

[55] J. Platt, N. Cristianini, J. Shawe-Taylor. Large margin DAGs for multiclass classification. *Advances in Neural Information Processing Systems 12*, pp. 547–553. MIT Press, 2000.

[56] R. Porzel, R. Malaka. A task-based approach for ontology evaluation. In: *Proc.* ECAI *2004 Workshop on Ontology Learning and Population*, pp. 9–16.

[57] F. Provost, T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231, March 2001.

[58] R. Rada, H. Mili, E. Bicknell, M. Blettner. Development and application of a metric on semantic nets. IEEE *Trans. on Systems, Man, and Cybernetics*, 19(1):17–30 (1989).

[59] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850 (1971).

[60] J. Rennie, R. Rifkin. *Improving multiclass text classification with the support vector machine*. Massachusetts Institute of Technology, AI Memo AIM-2001-026, 2001.

[61] *Reuters Corpus, Volume 1*, English language, 1996-08-20 to 1997-08-19. Release date 2000-11-03, `http://trec.nist.gov/data/reuters/reuters.html`.

[62] C. J. van Rijsbergen: *Information Retrieval*. Butterworths, 1979.

[63] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, D. Mladenić. Triplet extraction from sentences. In *Proceedings of the 10th International*

Multi-conference on Information Society, Ljubljana, Slovenia, 2007, pp. 218–222.

[64] D. Rusu, B. Fortuna, M. Grobelnik, D. Mladenić. Semantic graphs derived from triplets with application in document summarization. *Informatica*, 33(3):357–362, 2009.

[65] R. Srikant, R. Agrawal. Mining generalized association rules. *Proceedings of 21th International Conference on Very Large Data Bases*, VLDB-1995, September 11–15, 1995, Zurich, Switzerland, pp. 407–419.

[66] R. E. Schapire. Using output codes to boost multiclass learning problems. *Proceedings of the 14th International Conference on Machine Learning*, 2007, pp. 313–321.

[67] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–71 (2001).

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

[69] S. Shalev-Shwartz, Y. Singer, N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *Proc. of the 24th International Conference on Machine Learning*, ICML 2007.

[70] J. Shawe-Taylor, N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge: Cambridge University Press, 2004.

[71] R. Sipoš, D. Mladenić, M. Grobelnik, J. Brank. Modeling common real-world relations using triples extracted from n-grams. *Proc. of the Asian Semantic Web Conference*, 2009, pp. 16–30.

[72] A. J. Smola, B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222 (2004).

[73] P. Spyns. *EvaLexon: Assessing triples mined from texts*. Technical Report 09, STAR Lab, Brussels, 2005.

[74] M. Steinbach, G. Karypis, V. Kumar. A comparison of document clustering techniques. *Proceedings of the 6th* KDD *Workshop on Text Mining*, Boston, MA, USA, August 20–23, 2000.

[75] L. Stojanović. *Methods and Tools for Ontology Evolution*. Ph. D. thesis, University of Karlsruhe, 2004.

[76] T. Su, J. G. Dy. In search of deterministic methods for initializing K-means and Gaussian mixture clustering. *Intelligent Data Analysis*, 11(4):319–338, 2007.

[77] K. Supekar. A peer-review approach for ontology evaluation. In: *Proceedings of the International Protégé Conference*, Madrid, Spain, 2005.

[78] J. A. K. Suykens, J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300 (1999).

[79] M. Trampuš, D. Mladenić. Constructing event templates from written news. *Proc. of the International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Milan, Italy, 2009, pp. 507–510.

[80] V. N. Vapnik. *Statistical Learning Theory*. New York: John Wiley & Sons, 1998.

[81] P. Velardi, R. Navigli, A. Cucchiarelli, F. Neri. Evaluation of OntoLearn, a methodology for automatic learning of domain ontologies. In: P. Buitelaar, P. Cimiano, B. Magnini (*eds.*), *Ontology Learning from Text: Methods, Evaluation and Applications*, IOS Press, 2005.

[82] J. Völker, Y. Sure. *Data-driven change discovery*. Deliverable 3.3.1, SEKT Project (EU IST-2003-506826), July 22, 2005.

[83] J. Völker, D. Vrandecic, Y. Sure. Automatic evaluation of ontologies (AEON). In: *Proceedings of the 4th International Semantic Web Conference*, 2005.

[84] J. Weston, C. Watkins. Support vector machines for multi-class pattern recognition. In: *Proceedings of the 7th European Symposium on Artificial Neural Networks*, 1999.

[85] B. Yildiz. *Ontology Evolution and Versioning: The state of the art*. Tech. Rept. Vienna University of Technology, 2006.

[86] F. Zablith, M. Sabou, M. d'Aquin, E. Motta. Ontology Evolution with Evolva. In: *Proc. of the 6th European Semantic Web Conference* (2009). LNCS 5554, pp. 908–912.

[87] P. Zhong, M. Fukushima. A new multi-class support vector algorithm. *Optimization Methods and Software*, 21(3):359–372 (2006).

Izjava o avtorstvu

Spodaj podpisani Janez Brank z vpisno številko 63960012 sem avtor doktorske disertacije z naslovom Obravnava učnih problemov na velikih hierarhijah razredov kot več dvorazrednih problemov. S svojim podpisom zagotavljam, da:

- sem doktorsko disertacijo izdelal samostojno pod vodstvom mentorja prof. dr. Ivana Bratka in somentorstvom izr. prof. dr. Dunje Mladenić;
- so elektronska oblika doktorske disertacije, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko doktorske disertacije
- in soglašam z javno objavo elektronske oblike doktorske disertacije v zbirki „Dela FRI“.

V Ljubljani, dne 28. avgusta 2012. Podpis avtorja: