Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matic Di Batista

Označevanje imenskih entitet v pravnih besedilih

diplomsko delo

na univerziteznem študijskem programu
računalništva in informatike

Mentor: izr. prof. dr. Marko Bajec

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

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

Št. naloge: 01961/2013
Datum: 24.09.2013

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

Kandidat: **MATIC DI BATISTA**

Naslov: **OZNAČEVANJE IMENSKIH ENTITET V PRAVNIH BESEDILIH**
NAMED ENTITY RECOGNITION IN LEGAL DOCUMENTS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Odkrivanje podatkov iz besedil velja za eno izmed aktualnih podpodročij v okviru obdelave tekstovnih podatkov. Za slovenski jezik še nimamo dovolj prilagojenih pristopov ali ogromnih podatkovnih množic iz katerih bi lahko zgradili praktično uporabne metode za odkrivanje entitet. Namen diplomske naloge ja zato izdelava orodja, ki bo znalo odkrivati imenske entitete v slovenskih besedilih.

Kandidat naj pregleda obstoječe metode za odkrivanje entitet v besedilih in jih prilagodi za delo s slovenskim jezikom. Pri tem naj primerja njihovo delovanje in razišče morebitne probleme, ki so posledica sintakse in pravil v slovenščini. Nazadnje naj predlaga nov nabor značilk za učenje modelov in razvito metodo testira nad lastno izdelano podatkovno množico.

Mentor:

izr. prof. dr. Marko Bajec

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matic Di Batista, z vpisno številko **63070093**, sem avtor diplomskega dela z naslovom:

Iskanje imenskih entitet v pravnih besedilih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marka Bajca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 01. september 2013 Podpis avtorja:

Zahvaljujem se sodelavcem IUS SOFTWARE, še posebej vodji razvoja Romanu Zvonarju za pomoč in nasvete pri izdelavi diplomske naloge.

Prav tako bi se zahvalil Slavku Žitniku za nasvete in napotke pri obravnavani tematiki.

Zahvala gre tudi staršem in sestri za spodbude in podporo skozi ves čas študija.

We will never know our full potential unless we push ourselves to find it. It's this self discovery that inevitably takes us to the wildest places on Earth. - Travis Rice

Kazalo

Povzetek

Pridobivanje podatkov iz besedil postaja vse bolj pomembno, saj na ta način lahko uporabniku olajšamo delo z besedili. Tehnologije razvite v zadnjih desetletjih pa omogočajo, da lahko dosežemo zelo dobre rezultate.

V diplomskem delu je predstavljenih nekaj obstoječih rešitev, za iskanje imenskih entitet v pravnih besedilih. Namen iskanja imenskih entitet, je prepoznavanje določenih entitet, s katerimi lahko nato zgradimo povezave na dokumente, ki pripadajo tem entitetam. S kombinacijo različnih entitet pa pridobimo še dodatno informacijo o samem dokumentu.

Opisane so lastnosti posameznih rešitev za prepoznavanje imenskih entitet. Samo delovanje sistema v slovenskih pravnih besedilih smo testirali pa obstoječih rešitvah - Stanford CoreNLP, in lastni rešitvi - aplikacija NERInLaw, z uporabo CRFsuite za namene samega označevanja imenskih entitet.

Delovanje smo preverili na ročno označenih besedilih, kjer smo iskali določene imenske entitete. Besedila smo razdelili na učno in testno množico, da smo uspešnost lahko tudi ocenili. Preizkusili smo uspešnost delovanja z različnimi nabori atributskih funkcij in ugotovili, katere so bolj in katere manj pomembne za uspešnost. Hkrati smo se ozirali tudi na hitrost delovanja. Ta je v primeru velikega nabora besedil, kjer želimo imenske entitete označiti, tudi zelo pomembna.

Sama implementacija že sedaj dosega zelo dobre rezultate, v primeru nadaljnjega razvoja in izboljšav pa verjetno lahko to uspešnost še izboljšamo. Prav tako je trenutna implementacija z manjšimi spremembami primerna za uporabo v drugih jezikih in ne le za slovenščino.

Ključne besede: iskanje imenskih entitet, oblikoskladenjske oznake, pogojna naključna polja, Stanford CoreNLP, CRFsuite.

Abstract

Named entity recognition from natural language texts is getting more important every day, because it helps user with text manipulation. Technologies developed in last decades are able to produce really good result with information retrieval from natural texts.

In this diploma thesis we made brief representation of available solutions for named entity recognition in law texts. We want to recognize as many Named entities as possible so we can use them to make hyperlinks to referring documents. In combination of multiple named entities we can get additional information of observed document.

We described properties of available solutions for named entity recognition. Afterwards we tested named entity recognition on Slovenian law texts with two solutions – Stanford CoreNLP, and our own solution - application NERInLaw, with the use of CRFsuite.

We tested both solutions on hand marked law texts, where we marked multiple named entities. We divided the texts into learning set and test set, so we were able to evaluate the results. Tests were made with the use of different set of attribute functions, so we could see the difference in results and see which functions are more important for the system. Another important property of testing was the speed of tested solutions. With large dataset, it is important that we get results as fast as possible.

Our implementation got really good results with some basic settings. We are sure that with the future work, we could get even better results. Another good thing is, that current implementation could be easily used for other

languages than Slovenian with some minor changes.

Key words: named entity recognition, part of speech, conditional random fields, Stanford CoreNLP, CRFSuite

Poglavje 1

Uvod

Razvoj informacijskih tehnologij je v današnjem svetu omogočil hrambo in dostop do velike količine podatkov. Podatki, ki jih hranimo, so v veliki meri tudi precej neurejeni ter ne nudijo dostopa do informacije, ki jo v resnici vsebujejo. Če vzamemo za primer bazo besedilnih dokumentov, ki hrani osnovno informacijo o dokumentu, se v resnici v vsakem dokumentu hrani še veliko dodatne informacije, ki je človeku dostopna šele ob prebiranju besedila. Če dokumente analiziramo in v njih odkrijemo določene entitete, lahko le-te še lažje klasificiramo in pridobimo dodatno znanje, ki prej ni bilo dostopno.

Pri iskanju informacij v besedilih si pomagamo z raznimi analizami besedila, kot so: iskanje imenskih entitet, iskanje ko-referenc ter iskanje odvisnosti v besedilih. Naštete metode so del področja računalništva, imenovanega procesiranje naravnega jezika (angl. Natural language processing - NLP) [3]. Omenjeno področje v zadnjih letih doživlja močan razvoj. NLP temelji na uporabi statističnih metod in metod strojnega učenja za preiskovanje besedil. NLP področje se je v zadnjih letih precej razvilo, predvsem zaradi povečanja računske moči današnjih računalnikov.

Zgodovina NLP sega vse tja do 50 let 20 stoletja, ko je Alan Turing objavil članek z naslovom "Computing Machinery and Intelligence" [19], ki je predstavljal Turingov test kot kriterij inteligence.

Raziskave na tem področju so se nadaljevale predvsem v smeri avtomat-

skih prevajalnikov besedila, ter robotov za pogovore. Rešitve, razvite do 1980, so večinoma temeljile na ročno vpisanih pravilih in posledično niso bile preveč uspešne. Po letu 80 je NLP doživel velikanski napredek, saj so bili predstavljeni novi načini z uporabo metod strojnega učenja. V začetku so raziskovalci uporabljali predvsem odločitvena drevesa, ki v resnici prav tako predstavljajo nekakšen nabor if-then pravil. To pa za procesiranje besedil ni najboljša metoda, predvsem zaradi nepravilnih vhodnih podatkov, ki so v besedilih pogosto zaradi človeških napak.

Razvoj algoritmov za NLP se je zato nadaljeval v smeri uporabe statističnih modelov, ki z verjetnostnimi pravili postavljajo uteži za določene značilke vhodnih podatkov. Modeli delujejo veliko bolje, kadar vhodni podatki vsebujejo veliko šuma in nepravilnosti. To je pogosto prisotno v besedilih, saj besedila kreirajo ljudje.

Algoritmi strojnega učenja, ki se uporabljajo za NLP, so večinoma algoritmi nadzorovanega učenja. Raziskave zadnjega obdobja pa poskušajo tudi z algoritmi nenadzorovanega učenja. Le-ti so precej težji ter dajejo nekoliko slabše rezultate, vendar pa so primerni zaradi ogromnih količin neoznačenih podatkov, ki so na voljo.

Načini pridobivanja informacij iz besedil so se torej do današnjega časa razvili do te mere, da dosegajo precej dobre rezultate in omogočajo uporabo na različnih področjih. V nadaljevanju bomo predstavili eno izmed rešitev za iskanje imenskih entitet v pravnih besedilih. To področje je zelo zanimivo, saj nam pridobivanje imenskih entitet omogoča boljši vpogled v informacijo, ki jo besedilo vsebuje. V našem primeru predvsem za povezovanje besedil med seboj.

V prvem delu si bomo pogledali nekaj sorodnih del oziroma obstoječih rešitev. Kasneje pa še uporabo nekaterih rešitev pri oblikovanju lastne implementacije za reševanje problema iskanja imenskih entitet.

Poglavje 2

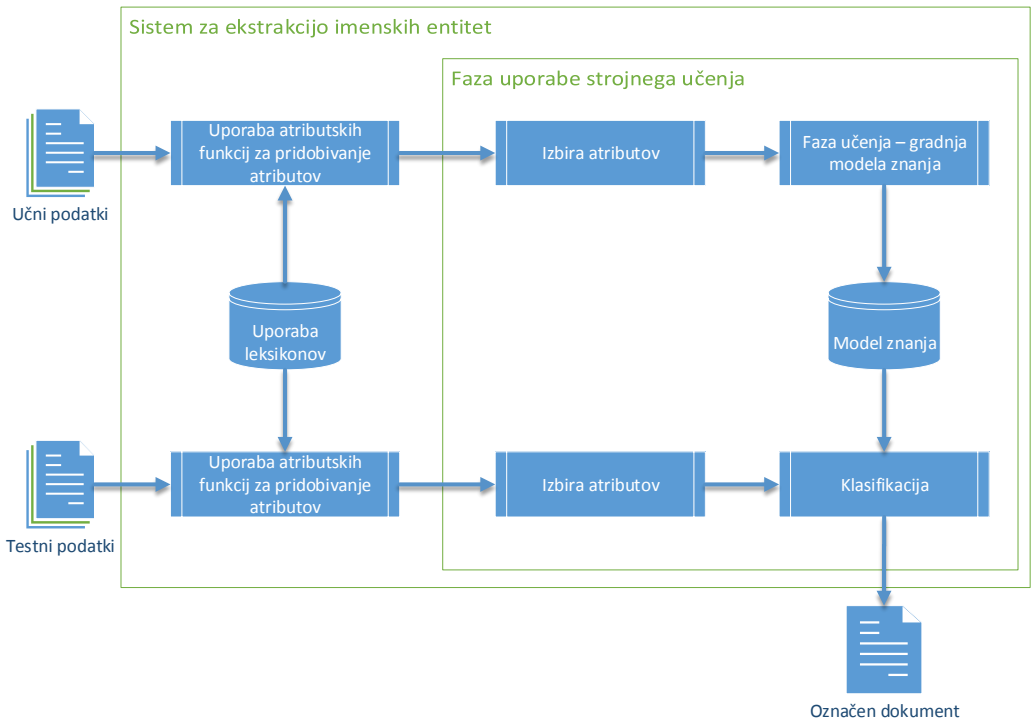
Pregled sorodnih del

Pri izdelavi diplomskega dela smo opravili pregled že obstoječih rešitev, ki omogočajo iskanje imenskih entitet v besedilih. Primer, kako je sestavljen klasičen sistem za iskanje imenskih entitet, je prikazan na diagramu2.1.

Pregledana orodja so bila večinoma razvita za angleški jezik. Za uporabo iskanja imenskih entitet v slovenskih besedilih je bilo potrebno najprej zgraditi napovedni model, za kar smo potrebovali dovolj velik nabor ročno označenih besedil, na katerih bi orodje zgradilo model znanja.

2.1 Pregled algoritmov za ekstrakcijo podatkov (entitet) iz besedil

Za ekstrakcijo podatkov iz besedil (v našem primeru iskanje imenskih entitet in iskanje oblikoskladenjskih oznak) se v praksi uporablja različne označevalne metode. Najbolj znani pristopi so z uporabo skritih markovskih modelov (angl. Hidden Markov Model – HMM) [8], ki implementirajo uporabo Viterbijevega algoritma [14]. V praksi se lahko uporabi tudi različne znane implementacije algoritmov strojnega učenja, kot so: metoda podpornih vektorjev (angl. Support vector machinge - SVM) [7], perceptron [12], algoritem k-najbližjih sosedov ... Pri vseh so bili doseženi rezultati uspešnosti nad



Slika 2.1: Diagram, ki prikazuje klasično zgradbo sistema za odkrivanje imenskih entitet.

95%.

Veliko število znanih rešitev za ekstrakcijo podatkov iz besedil uporablja pristop z uporabo sekvenčnih naključnih polj – CRF. Omenjeni pristop smo uporabljali tudi v naši rešitvi, zato sledi podroben opis v naslednjem razdelku.

2.1.1 Pogojna naključna polja (angl. CRF – Conditional random fields)

Pogojna naključna polja (angl. Conditional random fields – CRF) [21], je pristop, ki se pogosto uporablja pri razpoznavanju imenskih entitet. Gre za verižni model, ki obravnava in označuje besede zaporedno. To pomeni, da pri odločitvi klasifikacije upošteva tudi klasifikacijo prejšnje besede.

V primeru iskanja imenskih entitet so te v modelu CRF predstavljene kot stanja, ki imajo za oznake imenske entitete. Te so lahko na primer: *Zakon, Člen, Odločba, UradniList, ...*

V našem primeru ob obravnavi določenega stavka določimo te imenske entitete besedam glede na oznake prejšnjih besed ter glede na značilke posamezne besede, ki jih pridobimo v koraku predpriprave besedila. CRF pa omogoča tudi odvisnosti med besedami višjih redov, vendar za besedila to ni preveč primerno, ker lahko pride do prevelikega prilagajanja ter upočasnitve procesa učenja zaradi povečanja števila značilk. Priporočeno je uporaba modela, ki je odvisen le od lokalnih značilk in od razredov bližnjih besed.

Način delovanja CRF lahko najlažje predstavimo z grafom $G = (V, E)$, kjer je $Y = (Y_v)_{(v \in V)}$, tako da posamezne dimenzije X predstavljajo vozlišča G. X predstavlja v primeru uporabe posamezne primere - v našem konkretnem primeru besede skupaj z značilkami, Y pa predstavlja posamezne ciljne razrede, oziroma v našem konkretnem primeru imenske entitete.

Zato CRF spada v družino verjetnostnih grafičnih modelov (angl. Probabilistic graphical models [2]), kar predstavlja diagramično predstavitev porazdelitve verjetnosti. V takih grafih so vse naključne spremenljivke predstavljene z vozliščem. Pomanjkanje povezave med dvema spremenljivkama pa

X_0	X_1	X_2	Y
Meni	meniti	Ggnste	O
,	,	,	O
da	da	Vd	O
sta	biti	Gp-std-n	O
bili	biti	Gp-d-dz	O
prekršeni	prekršen	Pdnzdi	O
določbi	določba	Sozdi	O
86.	86.	Kav	CLEN
člena	člen	Somer	CLEN
Obligacijskega	obligacijski	Ppnmer	ZAKON
zakonika	zakonik	Somer	ZAKON
(	(	(	O
OZ	oz	Slmer	ZAKON
)	)	)	O

Tabela 2.1: Tabela prikazuje primer X in Y, kjer so vrednosti X beseda, lema in POS oznaka.

predstavlja pogojno neodvisnost med tema dvema spremenljivkama.

Če imajo naključne spremenljivke Y_v markovsko lastnost glede na sosednost, potem je (X, Y) naključno polje. Markovska lastnost glede na sosednost pomeni, da je novo stanje odvisno le od prejšnjega stanja: $p(Y_v \mid X, Y_w, w \neq v) = p(Y_v \mid X, Y_w, w \backsim v)$, kjer $w \backsim v$ pomeni, da sta w in v soseda. V našem primeru to pomeni, da je oznaka (imenska entiteta) opazovane besede odvisna le od značilk trenutne besede in oznake (imenske entitete) prejšnje besede.

Med spremenljivkama X in Y torej lahko opišemo pogojno verjetnost in sicer z množico funkcij značilk oblike $f_k(y, y', x_t)_{(k=1)}^{K} \in \Re^K$. V našem konkretnem primeru je lahko taka funkcija $f_{(rimska-odlocba)}$, ki bo vračala 1, kadar se bo v trenutni besedi pojavila rimska številka, hkrati bo pa predhodna beseda prav tako označena kot "ODLOČBA", sicer bo funkcija vračala vrednost 0.

$$f_{(stevilo-clen)}(y, y', x_t) = \begin{cases} 1 & if y = \text{CLEN } and x_1 = \text{člen} \\ 0 & \text{drugače} \end{cases} \tag{2.1}$$

Linearno verižno naključno polje – Linear chain CRF, se zato predstavi kot porazdelitev $p(y \mid x)$, ki jo opišemo z množico parametrov $\Lambda = \lambda_k \in \Re^K$:

$$p(y \mid x) = \frac{1}{(Z(x))} \exp \sum_{k=1}^{K} \lambda_k * f_k(y_t, y_{(t-1)}, x_t) \tag{2.2}$$

$Z(x)$ je normalizacijska funkcija.

Da zgradimo model znanja, s katerim bomo napovedovali določene imenske entitete, je nato potrebno oceniti vrednost parametrov Λ. Te nam povedo povezanost med določeno značilko in določenim ciljnim razredom. CRF po navadi za ocenjevanje parametrov uporablja maksimizacijo regulariziranega pogojnega log-verjetja (angl. Conditional log-likelihood) glede na učno množico primerov. Uporabo pogojnega log-verjetja lahko predstavimo s sledečo enačbo:

$$l(\Lambda) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k * f_k(y_t^{(i)}, y_{t-1}^{(i)}, x_t^{(i)}) - \sum_{i=1}^{N} \log Z(x^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2} \quad (2.3)$$

Prvi člen po vseh primerih izračuna porazdelitev, zadnji člen pa potrebujemo za regularizacijo, ki prepreči prekomerno prilagajanje našega modela znanja podatkom. Kazni za previsoke uteži λ_k dobijo kazen, ki je določena z izrazom $\frac{1}{2\sigma^2}$ – moč regularizacije. Izraza $l(\Lambda)$ ni moč maksimizirati v zaprti obliki, yato se uporablja numerična optimizacija s pomočjo odvodov. Rešitve, ki se uporabljajo za ta namen, so različni optimizacijski algoritmi. CRFsuite nam ponuja več različnih algoritmov. Ti so: Limited-memory BFGS [5], Othant-wise Limited-memory Quasi-Newton [4], Stochastic Gradient Descent (SGD) [15], Averaged Perceptron [10], Passive Aggressive [6] in Adaptive Regularization of Weight vectors [1].

Proces poteka v sledečem vrstnem redu. Predvidimo, da imamo množico učnih primerov – nabor stavkov s pripadajočimi oznakami imenskih entitet. Modelu na začetku priredimo naključne uteži posameznim značilkam. Če uporabimo za učenje princip gradientnega spusta, je postopek sledeč:

Najprej gremo čez vse atributske funkcije f_i in izračunamo gradient log verjetnosti učnih primerom glede na uteži λ_i:

$$\frac{\delta}{\delta * \omega_j} * log(p(y \mid x)) = \sum_{j=1}^{m} f_i(y_j, y_{j-1}, x_j) - \sum_{l'} p(y' \mid x) \sum_{j=1}^{m} f_i(y'_j, y'_{j-1}, x_j) \quad (2.4)$$

Prvi izraz v gradientu je dodatek značilke f_i pri dejanski oznaki, drugi izraz pa dodatek značilke pri oznaki, napovedani s trenutnim modelom. To je način, ki ga uporablja gradientni spust.

Da popravimo utež λ_i v smeri spusta, uporabimo naslednjo enačbo:

$$\lambda_i = \lambda_i + \alpha[\sum_{j=1}^{m} f_i(y_j, y_{j-1}, x_j) - \sum_{y'} p(y' \mid x) \sum_{j=1}^{m} f_i(y'_j, y'_{j-1}, x_j)] \quad (2.5)$$

α je stopnja hitrosti učenja.

Ti koraki se nato ponavljajo, dokler ne dosežemo nekega vnaprej določenega kriterija, ko proces ustavimo. Na primer spremembe α padejo pod neko mejo.

Če posplošimo, se v vsakem koraku računa razlika med tem, kaj želimo, da se model nauči, in kaj dejansko trenutni model napove. Ta razlika pa premika uteži α v smeri, da popravijo napoved modela.

Ko zaključimo s procesom učenja, tj. se naučimo parametrov našega modela znanja, lahko te parametre uporabimo za napovedovanje imenskih entitet novim primerom.

2.2 Pregled knjižnic za ekstrakcijo podatkov (entitet) iz besedil

2.2.1 NLTK

NLTK je kratica za "Natural Language Toolkit" [11] in predstavlja močno zbirko knjižnic za simbolično in statistično procesiranje naravnega jezika. Napisana je za programski jezik Python, ki v zadnjem času postaja vse bolj uporabljen jezik pri podatkovnem rudarjenju in strojnem učenju. Prav NLTK vsebuje poleg osnovnih funkcionalnosti tudi velik nabor knjižnic za grafično predstavitev podatkov, kar pride prav pri analizi rezultatov.

Kot večina večjih platform za NLP tudi NLTK vsebuje velik nabor označenih korpusov, s katerimi si lahko pomagamo pri učenju in testiranju lastnih rešitev. Vsebuje knjižnice za klasifikacijo besedila, razbitje besedila na osnovne enote (tu uporablja sistem svojo hevristiko), stemming, označevanje imenskih entitet, parsanje dokumentov in semantično analizo.

Med algoritmi strojnega učenja, ki so na voljo, sta med drugim tudi CRF in skriti Markovski model (angl. Hidden Markov model – HMM) [8], kar je bila sprva dodatna prednost. Prav tako omogoča tudi interakcijo s knjižnicami za NLP, razvitimi na univerzi Stanford.

Namestitev je precej preprosta in dobro dokumentirana, kot vse druge funkcionalnosti. Dokumentacija je dostopna na njihovi spletni strani ter

v knjigi Natural Language Processing with Python [13], ki je tudi prosto dostopna na internetu.

Za našo rešitev NLTK nismo izbrali iz dveh razlogov: rešitev je v programskem jeziku Python, kar zaradi slabšega poznavanja ni bilo sprejemljivo, ter sam proces učenja se je zdel pri drugih rešitvah precej bolj jasen in enostaven.

2.2.2 OpenNLP

OpenNLP je knjižnica skupine Apache Software Foundation in je prav tako prosto dostopna na internetu. Vsebuje podobno kot NLTK funkcionalnosti za NLP, kot so razbitje besedila na osnovne enote (besede, ločila, itd.), segmentacija stavkov, iskanje oblikoslovnih oznak, ekstrakcija imenskih datotek ter druge. Ta je napisana v Javi. Algoritmi strojnega učenja, ki jih uporablja OpenNLP so princip maksimalne entropije (angl. Maximum entropy) in perceptron.

API (angl. Application Program Interface) je precej intuitivno sestavljen in omogoča ogromen nabor lastnih nastavitev in sprememb. Tudi dokumentacija je na uradni strani OpenNLP zelo dobro sestavljena. Omogoča tudi uporabo javno dostopnih korpusov za označevanje imenskih entitet ter oblikoskladenjskih oznak, ki pa so namenjene angleškim besedilom.

Zanimiv je pristop pri rešitvi za razbitje besed na osnovne enote ter orodju za prepoznavanje stavkov. Oba je namreč moč naučiti z algoritmi strojnega učenja in pravilno označenim tekstom. Za razbijanje besedila na stavke in osnovne enote torej ne potrebujemo vnaprej določenih hevristik, ampak potrebujemo zgolj korpus, kjer je besedilo že razbito na osnovne enote z oznakami stavkov. Takšni korpusi za slovenščino so opisani v enem od naslednjih razdelkov.

2.2.3 Stanford CoreNLP

Stanford CoreNLP je zbirka knjižnic, ki omogočajo delo s tekstovnim rudarjenjem. Zbirka knjižnic omogoča razbitje besedila na osnovne enote (sicer narejeno za angleščino, vendar se z nekaj dodane logike obnese tudi za slovenščino), razbijanje na stavke, vsebuje oblikoskladenjski označevalnik, lematizator, prepoznavanje imenskih entitet, uporabo regularnih izrazov za prepoznavanje imenskih entitet ter iskanje odvisnosti med besedami v stavkih.

Iskalec imenskih entitet, ki ga uporablja Stanford CoreNLP, vsebuje osnovno implementacijo pogojnih naključnih polj (angl. Conditional random fields – CRF), ki se je v praksi izkazal kot odličen algoritem za grajenje sekvenčnih modelov za naloge, kot so NLP in sekvenciranje genoma v genetiki.

Celoten sistem deluje na principu objekta CoreMap, ki vsebuje stavke in objekte tipa CoreLabel, ki predstavljajo posamezne besede v stavku. Te nato vsebujejo dodatne atribute, ki jih določimo glede na želene rezultate. Če se odločimo, da bomo uporabili oblikoskladenjske oznake, jih sistem določi za vse objekte CoreLabel. Podobno je z lemami in drugimi potrebnimi atributi. V zadnji fazi naredimo še prepoznavanje imenskih entitet, ki vzame atribute, ki smo jih določili v predprocesiranju, in s pomočjo atributskih funkcij zgradi model za učenje, kadar smo v fazi učenja. Kadar smo v fazi klasifikacije, pa tako zgradi atributske funkcije, vendar jih uporabi za prepoznavanje entitet na podlagi pridobljenega znanja.

Glavni problem, na katerega smo naleteli pri testiranju Stanford CoreNLP sistema, je naraščanje časovne zahtevnosti zaradi prevelikega nabora oblikoskladenjskih oznak v primerjavi z angleščino. V primeru, ko nismo uporabili oblikoskladenjskih oznak, pa rezultati niso bili visoki. Kot bomo videli kasneje tudi v primeru uporabe oblikoskladenjskih oznak, rezultati niso dosegli rezultatov naše rešitve.

2.2.4 CRFSuite

CRFSuite je implementacija Pogojnih naključnih polj - CRF, napisana v programskem jeziku C++. Poleg različnih implementacij CRF omogoča kar nekaj dodatnih možnosti. Prva prednost je zelo hitra faza učenja in označevanja teksta, ki je posledica dobre implementacije v programskem jeziku C++. Predprocesiranje podatkov je povsem neodvisno od samega sistema CRFSuite. CRFSuite namreč uporablja preprost format podatkov, na katerih izvaja učenje oziroma klasifikacijo. Podatki so predstavljeni v vrsticah, kjer prva beseda predstavlja iskano entiteto, naslednje besede pa atribute, ki jih dobimo z atributskimi funkcijami, kar je potrebno implementirati posebej.

CRFsuite ponuja tudi precej velik nabor metod za učenje. Lahko izbiramo med Limited-memory L-BFGS, Orthant-Wise Limited-memory Quasi-Newton OWL-QN, Stohastično gradientno spuščanje SGD, Povprečeni perceptron, Pasivno agresivno učenje ter Adaptivno regularizacijo uteženih vektorjev AROW.

Omogoča tudi način evalvacije rezultatov, ki jih dobimo z učenjem in jih lahko preverimo s testno množico. CRFsuite nam izračuna precision, recall, F1 oceno, kar lahko uporabimo za predstavitev uspešnosti rezultatov. Sistem zna sam razbiti vhodne podatke na grupe, ki jih lahko nato uporabimo za različne načine evalvacije: 10-kratno prečno preverjanje, deljenje na učno in testno množico. Hkrati pa te evalvacije izvaja med samim procesom učenja, tako da je možen sproten vpogled v napredovanje sistema.

Na koncu pa vsebuje tudi precej dober API za uporabo CRFsuite v drugih aplikacijah napisanih v C++ in Python.

2.2.5 Obeliks

Kot pomoč pri predprocesiranju teksta smo v nekaterih primerih poskusili tudi z uporabo knjižnic, ki so bile razvite za slovenska besedila. Tak sistem je sistem Obeliks [9], ki omogoča lematizacijo in označevanje oblikoskladenjskih oznak za slovenska besedila. Napisan je v programskem jeziku C# in

uporablja knjižnice .NET Framework 2.0.

Poleg prej omenjenih funkcionalnosti, zna sistem prav tako razbiti besedilo na odstavke, stavke in besede. To ga deli na tri dele: segmentacijski del, modul za razbitje besedila na osnovne enote in oblikoslovni označevalnik skupaj z lematizatorjem. Obeliks uporablja specifikacije, ki so bile definirane v okviru projekta Jezikoslovno označevanje slovenščine JOS.

Oblikoskladenjsko označevanje temelji na statistični verjetnosti izbire med več možnimi oznakami. Uspešnosti označevanja so objavljene na spletni strani označevalnika. Obeliks določi kategorije in lastnosti besed (sklon, spol, število itd.) z 91,34% natančnostjo. Besedne vrste (samostalnik, glagol, pridevnik itd.) pa določi z 98,30% natančnostjo. Natančnost lematizatorja se giblje nekje od 97-98% natačnosti, odvisno od upoštevanja velike začetnice ali neupoštevanja velikih in malih črk.

Poglavje 3

Opis rešitve

Pri izdelavi lastne rešitve za iskanje imenskih entitet smo najprej potrebovali dovolj velik nabor vhodnih podatkov. Ti podatki so predstavljeni kot nabor besedil, ki smo jih naključno izbrali iz baze.

V nadaljevanju si bomo ogledali nekaj pomembnih razlik med slovenskim in angleškim jezikom, ki močno vplivajo na razvoj rešitev, kasneje pa natančneje preučili razvoj celotne rešitve in probleme, na katere smo naleteli pri izdelavi le-te.

3.1 Primerjava slovenskega in angleškega jezika

V preteklosti je bilo razvitih ogromno rešitev za pridobivanje informacij iz angleških besedil. Glavni razlog za to je razširjenost angleškega jezika ter dostopnost označenih besedil, ki služijo kot učna množica. Na internetu je prosto dostopnih kar nekaj tekstovnih zbirk, kjer so označene tako oblikoslovne oznake kot imenske entitete.

V tem razdelku je razloženih nekaj glavnih razlik med jezikoma, ki najbolj vplivajo na proces iskanja imenskih entitet v besedilih. Pogledali si bomo, kaj so oblikoskladenjske oznake (angl. POS – part of speech tags) in kakšen vpliv imajo ter katere zbirke podatkov za učenje že obstajajo.

Tudi na področju pridobivanja informacij iz slovenskih besedil je opazen velik napredek v zadnjih letih. Veliko je k temu pripomogla tudi baza 1.2 milijona označenih besed z oblikoskladenjskimi oznakami JOS1M. Omenjena baza besedil omogoča, da lahko naše aplikacije naučimo prepoznavanja imenskih entitet s pomočjo dodatne informacije o besedilu, ki jo predstavljajo oblikoskladenjske oznake.

Oblikoskladenjske oznake so specificirane v tabeli [16]. Primer take oznake je Sometn - kratica nosi sledečo informacijo o izbrani besedi prikazano v seznamu 3.1

Seznam 3.1: Prikaz pomena POS oznake Sometn

```
1 S − samostalnik
2 o − obcno ime
3 m − spol = moski
4 e − stevilo = ednina
5 t − sklon = tozilnik
6 n − zivost = ne
```

Primer označenega besedila je v tabeli 3.1.

Beseda	Meni	,	da	sta
POS oznaka	Ggnste	,	Vd	Gp-std-n
Beseda	bil	prekršeni	določbi	
POS oznaka	Gp-d-dz	Pdnzdi	Sozdi	
Beseda	86.	člena	Obligacijskega	zakonika
POS oznaka	Kav	Somer	Ppnmer	Somer
Beseda	(	OZ	)	
POS oznaka	(	Slmer	)	

Tabela 3.1: Tabela prikazuje primer besedila s pripadajočimi oznakami POS.

Po tabeli, ki vsebuje vse oznake in njihov pomen, lahko pridobimo torej vso potrebno informacijo.

Vendar pa tu pridemo do problema, če v naši rešitvi uporabljamo knjižnice, ki so bile razvite za angleška besedila. Glavni problem je število oblikoskladenjskih oznak. V slovenščini poznamo okoli 2000 oblikoskladenjskih oznak, med tem ko jih je v angleščini po navadi uporabljenih le 36, to povzroči kar nekaj nevšečnosti, saj se rado zgodi, da algoritmi niso napisani optimalno za tako veliko število različnih oznak. Omenjeno pripelje do problemov z naraščanjem časovne zahtevnosti.

Posledično postane ozko grlo celotne rešitve označevanje oblikoskladenjskih oznak, ki niti niso najbolj pomembne značilke za prepoznavanje imenskih entitet, kar bomo videli v nadaljevanju. V primeru, da iščemo hitro rešitev, je torej potrebno oblikoskladenjske oznake odstraniti iz nabora značilk, s čimer izgubimo nekaj informacije, ali pa poiskati drug način iskanja oblikoskladenjskih oznak, ki je hitrejši za slovenski jezik, oziroma razviti lastno rešitev.

3.2 Uporaba knjižnic

3.2.1 Stanford CoreNLP

Prva rešitev, ki smo jo preizkusili na naših podatkih je bila uporaba knjižnic Stanford CoreNLP. Knjižnice za procesiranje naravnega jezika smo uporabili preko API-ja, ki je na voljo. Rešitev je bila razvita v programskem jeziku Java, saj je za ta jezik API tudi napisan.

Prva faza te rešitve zajema razbitje besedila na osnovne enote (tekst razbijemo na besede, ločila, itd.), učenje označevalnika oblikoskladenjskih oznak, preprost sistem za iskanje lem določenih besed, grajenje atributskih funkcij ter učenje označevalnika imenskih entitet. Druga faza pa vsebuje razbitje besedila na osnovne enote, označevanje oblikoskladenjskih oznak, določanje lem besedam, grajenje atributskih funkcij ter prepoznavanje imenskih entitet.

Kot je bilo že omenjeno, rešitev, ki jo ponuja Stanford, vsebuje velik nabor različnih atributskih funkcij, ki jih lahko poljubno izbiramo za uporabo v našem označevalniku imenskih entitet. Določanje uporabe atributskih

funkcij ter določanje osnovnih podrobnosti poteka preko datoteke, ki v vsaki vrstici vsebuje ime atributske funkcije, ki jo želimo uporabiti. Prav tako v tej datoteki določimo pomen stolpcev, ki jih uporabimo v naši vhodni datoteki z besedilom, ki se uporablja.

Za vhodno datoteko, ki vsebuje 4 stolpce, ki predstavljajo besedo, lemo, oblikoskladenjsko oznako in oznako imenske entitete, je primer datoteke z navodili za označevalnik v seznamu 3.2.

Seznam 3.2: Našteti uporabljeni atributi v drugem poskusu

```
1  trainFile = data/ner/STANFORD-OUT-SLOPOSTAGGER-FROMJOS-28.3.2013-in.tsv
2  serializeTo = data/ner/testi/ner-model-slopos-28032013-tri.ser.gz
3  map = word=0,lemma=1,tag=2,answer=3
4  useNGrams=true
5  useDisjunctive=true
6  usePrev=true
7  useEntityTypes=true
8  useNext=true
9  useSequences=true
10 useLongSequences=true
11 useIfInteger=true
12 maxLeft=2
13 maxRight=2
14 useWordPairs=true
15 useLC=true
16 wordShape=chris2useLC
17 usePosition=true
18 useGazettes=true
19 gazette = data/ner/GazeteZakoniTrue.txt;data/ner/zakonigazeetkratice.txt
20 cleanGazette=true
21 type=crf
```

Pomeni posameznih atributov so sledeči:

- *trainFile*: S tem atributom definiramo datoteko, v kateri so shranjeni podatki za učenje. Datoteka predstavlja v stolpcih ločene podatke, ki jih določimo z atributom map.

- *serializeTo*: Določimo ime datoteke, v katero se serializira model znanja, ki ga generira sistem.

- *Map*: Nam pove pomene posameznih stolpcev v vhodni datoteki. $Word = 0$ nam pove, da je v prvem stolpcu beseda, $lemma = 1$ nam pove, da se v drugem stolpcu nahajajo leme, $tag = 2$ pove, da se v tretjem stolpcu nahajajo oblikoskladenjske oznake in $answer = 3$ pove, da se v četrtem stolpcu nahaja imenska entiteta za določeno besedo.

- *useNGrams*: Uporabimo, če želimo pri grajenju atributskih funkcij uporabiti $n - grame$. To so sekvence n-instanc, ki jih uporabljamo, v našem primeru je to n sosednjih besed.

- *useDisjunctive*: Ta značilka se uporablja za uporabo disjunkcij med besedami.

- *usePrev*: Nam določa, da se poleg trenutne besede kot posebna značilka uporabi tudi predhodna beseda.

- *useEntityTypes*: S tem določimo, da se v procesu učenja uporabijo tudi tipi entitet kot značilke.

- *useNext*: Podobno kot *usePrev* se ta značilka uporablja za uporabo naslednje besede za trenutno.

- *useSequences*: Če želimo *usePrev* in *useNext* še nadgraditi, uporabimo to značilko, ki uporabi sekvenco besed, koliko besed se bo uporabil z zastavicama $maxLeft$ in $maxRight$.

- *useIfInteger*: Zastavica, ki uporabi atributsko funkcijo, ki preverja, če je trenutna beseda celo število.

- *maxLeft*: Služi za parameter pri uporabi zastavice *useSequence*.
- *maxRight*: Služi za parameter pri uporabi zastavice *useSequence*.
- *useWordPairs*: S to zastavico uporabimo atributsko funkcijo, ki uporabi pare besed kot posebno značilko.
- *useLC*: Sistem uporabi besede brez velikih začetnic.
- *wordShape*: S to zastavico določimo, kateri način se bo uporabljal za predstavitev oblik besed. Stanford CoreNLP vsebuje kar nekaj različnih predstavitev. Različni načini so bili razviti predvsem zaradi različnih področij uporabe. Lahko izbiramo med parametri: chris1, dan1, dan2, dan2useLC, dan2bio, dan2bioUseLC, jenny1, jenny1useLC, chris2, chris2useLC, chris3, chris3useLC, chris4. Iz imen lahko vidimo, da so nekatere predstavitve namenjene tudi biološkim analizam tekstov.
- *usePosition*: S to zastavico uporabimo podatek o poziciji opazovane besede v stavku.
- *useGazettes*: S to zastavico uporabimo atributsko funkcijo, ki za učenje uporablja leksikone.
- *Gazette*: Če smo za učenje uporabili leksikone, uporabimo to zastavico, da naštejemo datoteke, v katerih so v vrsticah naštete besedne zveze, oziroma besede, ki jih želimo uporabiti, ter njihove entitete.
- *cleanGazette*: Z uporabo te zastavice besede iz uporabljenih leksikonov uporabimo v isti obliki kot smo jo navedli v datotekah. V primeru dolgih besednih zvez je to dobrodošlo, saj besede, kot so o, v, na, za, zmanjšajo uspešnost, ki jo dobimo z uporabo leksikonov.

Glavni problemi, na katere smo naleteli pri tej rešitvi, so bili predvsem s časi izvajanja. Kot je bilo že prej omenjeno, je čas sistema za označevanje oblikoskladenjskih oznak začel močno naraščati v primeru, ko se je pojavila beseda, ki se prej ni pojavila v učni množici.

Vzrok tega problema je prišel iz implementacije označevalnika, ki je bila poenostavljena (najverjetneje zaradi angleškega jezika) na ta način, da je v primeru nove besede večkrat uporabila prehode preko gnezdenih programskih zank, kjer se je računala verjetnost uporabe določene oblikoskladenjske oznake za to besedo. Teh prehodov je bilo toliko, kolikor je na voljo vseh oblikoskladenjskih oznak. V primeru slovenščine je to pomenilo več zank z okoli 1900 prehodi. Takih problemov v angleščini seveda ni bilo, ker je maksimalno število oznak le okoli 36.

Rešitev tega problema bi bila posplošitev slovenskih oblikoskladenjskih oznak, s čimer bi lahko zmanjšali nabor oznak, vendar bi izgubili veliko dodatne informacije. V literaturi [18] avtorji omenjajo ravno informacijo, ki jo dobimo iz oblikoskladenjskih oznak, kot zelo primerno za izboljšanje rezultatov označevanja imenskih entitet. V literaturi avtorjev uporabljene rešitve pa za angleška besedila velja ravno obratno. Oblikoskladenjske oznake v angleščini torej niso prinesle velikega povečanja uspešnosti.

Druga rešitev tega problema je bila rešena z uporabo zunanjih knjižnic za označevanje oblikoskladenjskih oznak ter iskanje lem besed. Uporabili smo knjižnici Obeliks in LemmaGen [20], razviti v okviru Inštituta Jožef Štefan. Poleg problema naraščanja časovne zahtevnosti pri označevanju oblikoskladenjskih oznak je bil dodaten problem tudi sama hitrost učenja oblikoskladenjskega označevalnika, označevalnika imenskih entitet ter poraba prostora v delovnem pomnilniku.

Za učenje oblikoskladenjskega označevalnika je sistem porabil 23GB delovnega pomnilnika ter 7 dni in 16 ur. Sicer je bil problem prisoten tudi pri učenju s sistemom CRFSuite, vendar več o tem v naslednjem razdelku.

Pri učenju označevalnika imenskih entitet je bil problem podoben, saj je sistem za učenje na datoteki, veliki 7MB, porabil ravno toliko delovnega pomnilnika ter 3 ure časa. Isti postopek je z uporabo CRFSuite trajal precej manj.

Faza označevanja imenskih entitet je bila glede na čas delovanja odvisna predvsem od hitrosti oblikoskladenjskega označevalnika, o čemer smo pisali

nekaj odstavkov prej. Samo označevanje imenskih entitet ni zahtevalo veliko časa. Drugih problemov pri uporabi Stanford CoreNLP nismo opazili. Morda bi bilo vredno le še omeniti, da z uporabo različnih atributskih funkcij, ter različnih kombinacij z upoštevanjem oblikoskladenjskih iznak ter brez njih, nismo dosegli prepričljivih rezultatov prepoznavanja imenskih entitet. Več o tem pa si lahko preberete v razdelku z rezultati.

3.2.2 CRFSuite

CRFSuite predstavlja le implementacijo pogojnih naključnih polj za označevanje sekvenčnih podatkov. Poleg implementacije CRF vsebuje še nabor učnih algoritmov in nekaj dodatnih opcij za testiranje rezultatov.

Atributske funkcije in generator ustreznih vhodnih podatkov je bilo potrebno implementirati posebej. Ta del implementacije smo razvili s programskim jezikom C# in uporabno knjižnic .Net.

Tudi ta rešitev je razdeljena v dva koraka. V prvem delu sistem vhodni tekst razbije na osnovne enote, tem določi lemo in oblikoskladenjsko oznako ter zgradi atributske funkcije. Z uporabe teh podatkov nato zgradi vhodno datoteko za proces učenja s sistemom CRFSuite. Rezultat učenja je nato datoteka modela, ki ga lahko uporabimo za napovedovanje imenskih entitet novim besedilom.

Drugi korak predstavlja vnos novega besedila, ki mu želimo označiti imenske entitete. Tega nato razbijemo na osnovne enote, ki jim določimo lemo in oblikoskladenjske oznake ter zgradimo atributske funkcije. Z dobljenimi rezultati zgradimo datoteko, ki jo sprejme sistem CRFSuite in na njej, z znanjem iz prejšnjega koraka, označi imenske entitete.

Problemi, na katere smo naleteli v tem primeru, so bili vezani predvsem na označevalnik oblikoskladenjskih oznak. Uporabili smo namreč dva označevalnika. Prvi je bil Obeliks v kombinaciji z LemaGen sistemom, drugi pa lasten označevalnik oblikoskladenjskih oznak, ki je deloval s pomočjo sistema CRFSuite.

Pri uporabi oblikoskladenjskega označevalnika smo naleteli predvsem na

problem v fazi učenja, saj je učenje trajalo 41 dni in 10 ur. Verjetno je bil glavni problem oziroma ozko grlo moč procesorja na računalniku, kjer se je sistem učil.

Drugih problemov tu ni bilo. Tudi samo označevanje imenskih entitet je bilo precej hitro in ni predstavljalo ovir pri doseganju čim hitrejših časov. Glavna ovira je bil zopet oblikoskladenjski označevalnik.

Tudi komunikacija med sistemom napisanim v C# in sistemom CRFsuite je potekala brez problemov. V ta namen smo uporabili pomožne datoteke za učenje/testiranje, kamor smo najprej zapisali besede z vsemi značilkami. Nato smo preko programskega vmesnika pognali zagonsko datoteko sistema CRFSuite, ki je vrnil dve datoteki z označenimi imenskimi entitetami in verjetnostmi za posamezne entitete. Slednja je koristila predvsem v primeru, ko za določeno entiteto osnovna verjetnost ni bila dovolj velika in smo nato z uporabno nižjega praga za sprejem imenskih entitet še izboljšali rezultate.

Sama uporaba CRFsuite je zelo preprosta in precej obsežno dokumentirana na uradni strani. Za učenje sistema lahko izberemo več načinov izvajanja. Lahko izvajamo le učenje iz učne datoteke s klicem ukaza:

```
crfsuite learn -m CRF.model train.txt
```

Ta nam zgradi model z imenom CRF in za učenje uporabi datoteko train.txt. Lahko pa uporabimo kar zajeten nabor zastavic, ki omogočajo različne nastavitve.

Seznam zastavic in kratka obrazložitev:

- t: Ta zastavica določa grafični model, ki se ga uporabi za generiranje značilk. Privzeta vrednost je “1d”, ki pomeni uporabo Markovega CRF prvega reda. V trenutni implementaciji je ta algoritem tudi edini možen za uporabo.

- a: S to zastavico določimo algoritem učenja. Privzeta vrednost je “lbfgs”. Druge možnosti so še:

1. l2sgd – Stohastični gradientni spust z L2 regularizacijo
2. ap – povprečni perceptron
3. pa – Pasivno agresiven algoritem
4. arow – Adaptivna regularizacija vektorjev uteži

- p: Ta zastavica nam omogoča le informacijo sistemu, da bomo uporabili določene parametre za proces učenja, če ga ta potrebuje.
- m: S to zastavico določimo ime, pod katerim želimo model znanja shraniti na disk.
- g: Če želimo učno množico razbiti na več grup, ki jih lahko uporabimo pri učenju, potem uporabimo to zastavico in zraven število grup, ki jih želimo uporabiti.
- e: Če smo z zastavico g razbili vhodne podatke na več grup, potem s to zastavico določimo, koliko od teh grup želimo uporabiti za testiranje. V primeru, da imamo ogromno učno množico, lahko razbijemo podatke na 5 grup (-g5) in nato uporabimo eno za testiranje (-e4).
- x: Če za testiranje želimo uporabiti prečno preverjanje, potem uporabimo zastavico x. Za (-x10) uporabimo 10x prečno preverjanje. Še prej pa potrebujemo razbitje na 10 grup z –g10.
- l: Zastavica –l omogoča zapisovanje izhodnih podatkov v procesu učenja v posebno "log"datoteko.
- L: Potrebujemo za določitev imena log datoteke.
- h: Ta zastavica služi kot pomoč pri uporabi učenja oziroma označevanja.
- H: Tudi ta služi kot pomoč, vendar nam poda informacije o posameznih parametrih, ki jih lahko uporabimo, in njihove opise.
- p: Ta zastavica služi za določanje parametrov, ki jih potrebujemo pri učenju.

Preprosti primeri klicev procesa učenja so sledeči:

```
crfsuite learn -g10 -x -l train.txt
```

S tem klicem dosežemo, da se v procesu učenja izvaja še 10-kratno prečno preverjanje.

```
crfsuite learn -m CRF.model -p feature.minfreq=2 train.txt
```

S tem klicem iz modela odstranimo značilke, ki se pojavijo manj kot dvakrat.

3.3 Učna množica vhodnih podatkov

Kot smo že omenili v enem od prejšnjih razdelkov, smo za samo označevanje imenskih entitet potrebovali tudi oblikoslovne oznake ter leme. Za označevanje oblikoslovnih oznak je bilo potrebno naš opraviti učenje na označenem besedilu, ki je dostopno na strani oddelka za tehnologije znanja, inštituta Jožefa Štefana.

Uporabili smo korpus jos1M [16], ki vsebuje 1 milijon besed, ki imajo delno ročno določene oblikoslovne oznake ter leme. Korpus je delno ročno označen zato, ker resursi projekta niso omogočali, da bi celoten korpus preverili ročno. Zato so celoten korpus najprej označili z avtomatskimi označevalniki, nato pa ročno preverili sumljive oznake. To so bile takšne oznake, kjer so se avtomatski označevalniki razlikovali. Eksperiment uspešnosti takega označevanja so preverili z 10-kratnim prečnim preverjanjem na korpusu jos100k, ki vsebuje 100000 ročno označenih besedil.

Testi so pokazali, da je potrebno od 1 milijona besed približno 190000 besed ročno preveriti, da bi dosegli natančnost 96,8%.

Korpus jos1M je kodiran z XML oznakami. Korpus sestavljajo serije odstavkov, ki vsebujejo enega ali več stavkov. Vsi stavki so označeni kot

samostojni XML elementi, kjer so nato besede označene z oblikoslovnimi oznakam, lemami ter besedo samo.

Druga zbirka besedil, ki smo jih uporabili pri naši rešitvi, je predstavljala korpus pravnih besedil, v katerih smo ročno označili entitete, ki so nas zanimale za pridobivanje dodatnih informacij o besedilih.

Besedila so predhodno razbita in shranjena v bazo po vnaprej določenih pravilih. Večinoma se ta besedila razbija s pravili, predstavljenimi z regularnimi izrazi. Gre predvsem za razbijanje na odstavke, glavo in nogo dokumenta, meta podatke itd.

Iz baze smo nato potegnili segmente, ki so vsebovali tekst besedila, ki nas je najbolj zanimal, ter nad njim izvedli ročno označevanje imenskih entitet. Ker je bil del besedil že delno avtomatsko pregledan in je vseboval del imenskih entitet, ki smo jih iskali, je bilo potrebno tudi predprocesiranje besedila. V tem koraku smo poiskali označene entitete ter jih zapisali v zapisu, ki se je uporabljal pri označevanju. S tem smo olajšali delo označevalcu, saj je bil del besedila vnaprej označen.

Preostanek označevanja je potekal ročno z delno pomočjo avtomatskega označevanja. Ko označevalec označi v tekstu del besedila, ki predstavlja imensko entiteto, se mu pojavi izbirno okence, kjer določi, kateri imenski entiteti pripada del besedila, le-to se nato shrani v kolekcijo, ki služi kot pomoč pri naslednjih besedilih. V primeru, da se v naslednjem besedilu pojavi že označena entiteta, jo avtomatska logika prepozna in označi. Označevalec pa lahko v realnem času preveri, če je bila označba morda napačna.

S tem se označevanje močno pohitri, saj ni potrebe po večkratnem označevanju istih entitet. Prav tako pa do neke mere popravimo točnost označevanja, saj je mogoče, da bo označevalec določeno entiteto, spregledal.

V končnem korpusu je bilo označenih 85096 besed, ki so vsebovale 6 različnih entitet, ki so se pojavile 5790-krat.

Rezultat označevanja je bila kolekcija tekstov, ki so vsebovali dodatne XML oznake, ki so predstavljale imenske entitete. Ta korpus smo nato uporabili v naslednjem koraku, ki služi za grajenje atributskih funkcij, ki so

služile za boljšo uspešnost algoritmov strojnega učenja. O tem bomo več razložili v naslednjem razdelku.

3.4 Opis sistema

Kot smo omenili v razdelku opisa uporabljenih knjižnic, smo za to diplomsko delo uporabljali knjižnice .NET, kajti aplikacija za označevanje imenskih entitet je bila v večini razvita z jezikom C#. Algoritem CRF, ki se je uporabljal za strojno učenje, smo pa klicali iz dodatne knjižnice CRFSuite, ki je napisana v C++.

Za komunikacijo med tema aplikacijama smo uporabljali tekstovne datoteke, ki so vsebovale besedilo, namenjeno učenju oziroma označevanju.

3.4.1 NERInLaw

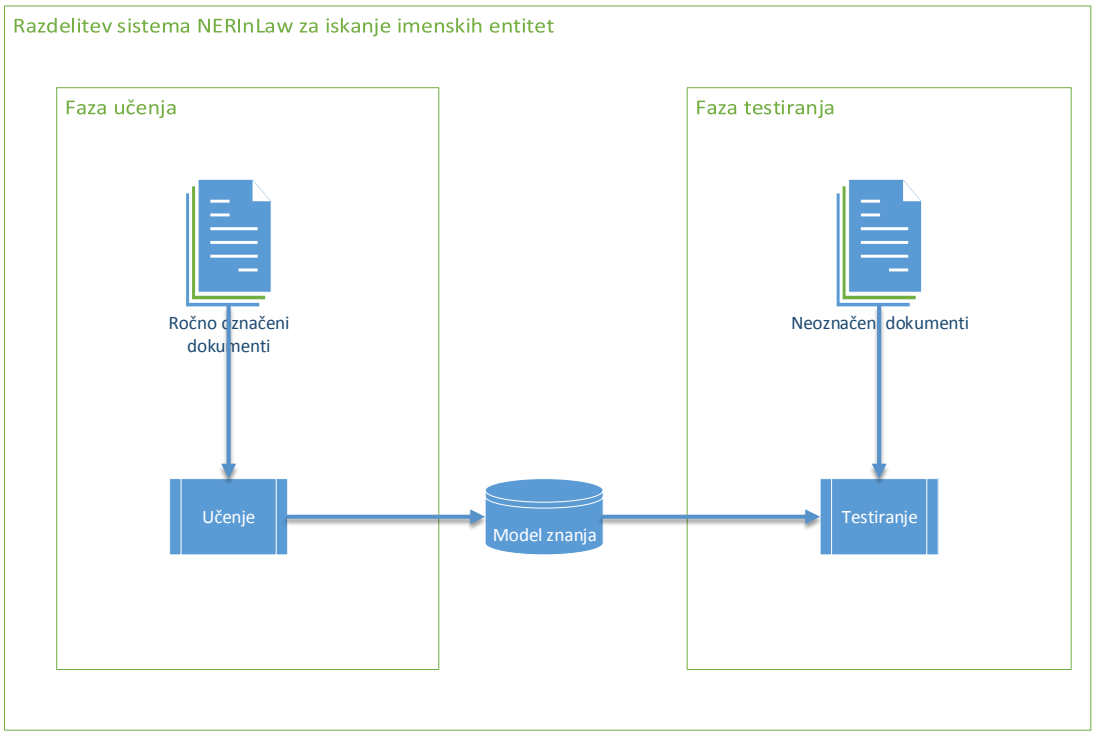
Aplikacija NERInLaw, ki je napisana v C# in .NET frameworku, služi za komunikacijo z bazo, v kateri so shranjena vsa ciljna besedila, ter omogoča uporabniški vmesnik, s katerim je mogoče ročno označiti imenske entitete za proces učenja.

Iz diagrama na sliki 3.1 je razviden grob prikaz celotnega sistema za označevanje imenskih entitet v pravnih besedilih. Glavni dve komponenti sta komponenta za učenje in komponenta za testiranje zgrajenega modela znanja.

3.4.2 Komponenta za učenje

Ta komponenta (slika 3.2) vsebuje sistem za ročno označevanje imenskih entitet, ki prejme nabor besedil iz baze. Ta besedila so "surova"besedila brez oblikoskladenjskih oznak ter oznak imenskih entit.

Komponenta za učenje vsebuje grafični vmesnik, kjer se besedila, ob klicu posebne metode, prikazujejo. Uporabnik nato s kurzorjem izbere besedo ali



Slika 3.1: Diagram prikazuje grobo razdelitev sistema za iskanje imenski entitet.

besedno zvezo, ki predstavlja določeno entiteto. Ko je beseda označena, se pojavi novo okno, kjer uporabnik izbere, kateri entiteti pripada označena beseda oziroma besedna zveza.

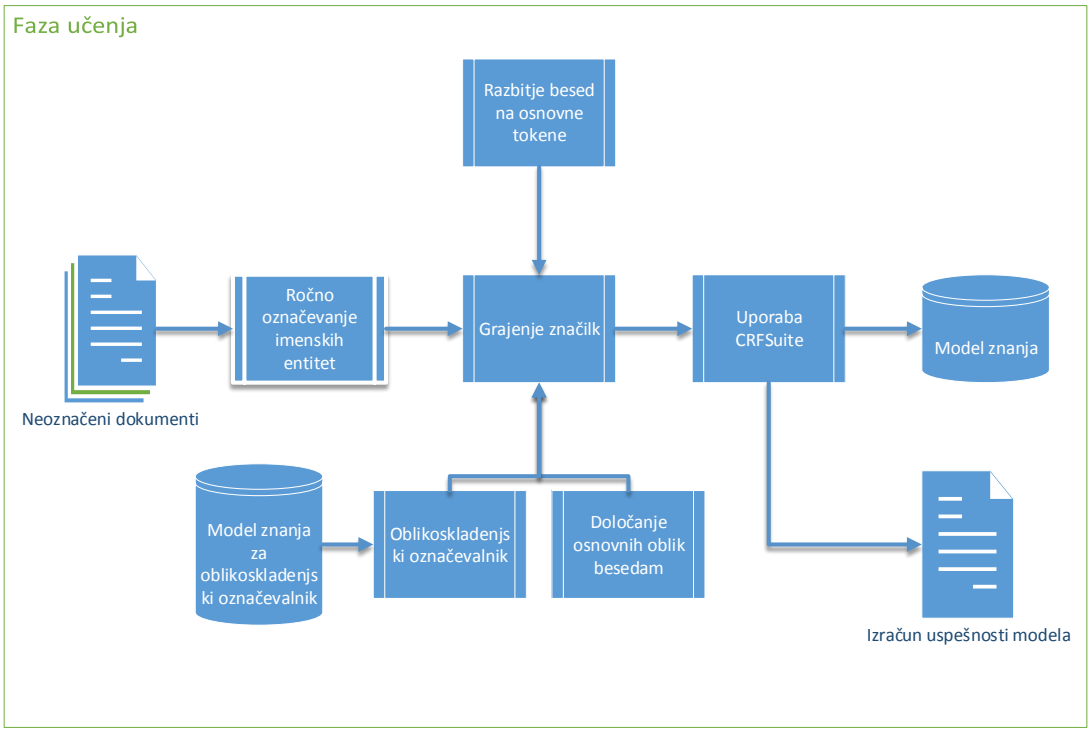
Pri označevanju imenskih entitet se vsaka beseda ali besedna zveza, ki dobi oznako posebne entitete, shrani v kolekcijo. Pri vsakem novem besedilu se nato opravi predprocesiranje, ki poišče, s pomočjo regularnih izrazov, v besedilu že znane entitete in jih označi. S tem močno pohitrimo proces ročnega označevanja, saj se izognemo večkratnemu ročnemu označevanju pojavitev istih besednih zvez.

Primer ročno označenega besedila je spodaj:

> Tožena stranka je vložila predlog za dopustitev revizije. Meni, da sta bili prekršeni določbi ⟨*ENTITETAtype = CLEN*⟩ 86. člena⟨/*ENTITETA*⟩ ⟨*ENTITETAtype = ZAKON*⟩ Obligacijskega zakonika (OZ) ⟨/*ENTITETA*⟩ in ⟨*ENTITETAtype = CLEN*⟩8. člena⟨/*ENTITETA*⟩ ⟨*ENTITETAtype = ZAKON*⟩ Zakona o zemljiški knjigi (ZZK-1)⟨/*ENTITETA*⟩ ter ustavne določbe o enakem varstvu pravic in o pravici do sodnega varstva. V predlogu postavlja pravno vprašanje: ali je za ugotovitev ničnosti pogodbe, katere predmet je nepremičnina, treba ugotoviti ne dobrovernost obeh pogodbenih strank. Trdi, da odločitev odstopa od sodne prakse Vrhovnega sodišča.

Ko z ročnim označevanjem zaključimo, sistem pošlje v korak za grajenje značilk. V procesu se zgradijo vse značilke, ki so bile določene za ta tip dokumenta.

Prvi korak, ki se nato izvede, je razbitje besedila na osnovne enote (besede, ločila, ...), kajti takšno obliko potrebujemo za grajenje korpusa, ki ga sprejme CRFSuite. Besede in ločila so razbiti v vrstice. Besedilo se prevede v objekte, kjer je osnovni objekt seznam vseh odstavkov. Objekt odstavkov vsebuje stavke, saj z razbitjem stavka razbremenimo sistem za strojno



Slika 3.2: Diagram, ki prikazuje potek delovanja faze učenja.

učenje, ki za učenje sprejme vsak stavek kot ločeno celoto. Objekt stavek nato vsebuje vsako besedo, ki jo dobimo z razbitjem besedila na osnovne enote, posebej.

Zgrajena datoteka, ki se uporabi za proces gradnje modela znanja s CRFsuite, je sestavljena iz vrstic, kjer vsaka vrstica predstavlja na prvem mestu posamezno entiteto, ki ji sledijo s tabulatorjem ločene značilke. Stavki so med seboj ločeni s prazno vrstico. Primer take datoteke je prikazan v tabeli 3.2.

O	f[W]=da:1.0	f[WL]=da:1.0	f[WLP]=,:1.0
O	f[W]=je:1.0	f[WL]=je:1.0	f[WLP]=da:1.0
O	f[W]=treba:1.0	f[WL]=treba:1.0	f[WLP]=je:1.0
O	f[W]=določbo:1.0	f[WL]=določbo:1.0	f[WLP]=treba:1.0
ODSTAVEK	f[W]=drugega:1.0	f[WL]=drugega:1.0	f[WLP]=določbo:1.0
ODSTAVEK	f[W]=odstavka:1.0	f[WL]=odstavka:1.0	f[WLP]=drugega:1.0
CLEN	f[W]=160.:1.0	f[WL]=160.:1.0	f[WLP]=odstavka:1.0
CLEN	f[W]=člena:1.0	f[WL]=člena:1.0	f[WLP]=160.:1.0
ZAKON	f[W]=ZPPSL:1.0	f[WL]=zppsl:1.0	f[WLP]=člena:1.0

Tabela 3.2: Atributi, dobljeni z atributskimi funkcijami in pripadajoče labele za posamezne besede.

Zaradi preprostosti je v primeru prikazanih le prvih nekaj značilk za vsako besedo.

Glavni problem tega koraka je uporaba določenih atributskih funkcij za gradnjo značilk. Če se odločimo, da bomo za naš problem uporabili označevanje oblikoskladenjskih oznak, to zahteva uporabo izbranega modula za to označevanje. Prav ta korak pa zahteva zaradi zahtevnosti označevanja največ časa, kar močno podaljša sam proces označevanja.

Logika aplikacije omogoča, da za vsak korpus določimo, katere vnaprej definirane značilke bomo uporabili. Tako lahko za besedila, kjer z analizo spoznamo, da določenega atributa ne bomo potrebovali, prihranimo na zah-

tevnosti učenja s tem, da zmanjšamo učno množico. Vendar pa je pri tem potrebno paziti, da s tem ne zmanjšamo informacijskega prispevka.

Atributske funkcije

- Beseda: Atribut beseda je kar obravnavana beseda sama. Torej za vsako besedo kot atribut uporabimo kar besedo samo. To nam poda nekaj informacije, vendar pa zaradi pojavitev iste besede v različnih npr. sklonih, sistem ne opazi, da gre za isto besedo.

- Beseda brez velikih začetnic: Atribut besede zapisane z malimi črkami služi predvsem za izogibanje ločevanja med isto besedo, ki se enkrat pojavi na začetku stavka, drugič pa na nekje vmes. Če tega ne bi uporabili, bi program razumel obe besedi kot drugačni, s tem pa se temu izognemo in mu pozicijo v stavku predstavimo z drugimi atributi.

- Beseda pred izbrano: Besedo pred trenutno besedo uporabimo za prikaz interakcij med različnimi besedami. S tem prepoznamo določene besedne zveze, ki se pogosto pojavljajo in morda vplivajo na pojavitev imenske entitete. S to atributsko funkcijo lahko pridobimo informacije o odvisnosti med določenimi besedami, ki se pogosto pojavljajo skupaj kadar tvorijo določeno entiteto. Primer: “13. člen Zakona o . . . ”. Besedi “člen”in “zakonše pogosto pojavita, kadar navajamo določene člene zakonov. Isto velja za besedi “zakon”in “o”. Ker funkcija vrača dve zaporedni besedi, lahko rečemo, da je funkcija tipa bi-gram.

- Beseda za izbrano: Podobno kot z besedo pred izbrano besedo, lahko uporabimo tudi besedo za izbrano besedo. Prav tako v tem primeru dobimo nekaj dodatne informacije o besednih zvezah, ki se pogosto pojavljajo v besedilih in morda nastopajo kot imenske entitete. Ker funkcija vrača dve zaporedni besedi, lahko rečemo, da je funkcija tipa bi-gram.

- Več sosednjih besed: Če predhodna atributa še nekoliko razširimo, do-

bimo interakcijo več kot dveh besed. V naši rešitvi smo uporabili največ tri besede pred izbrano in tri besede za izbrano. S tem tvorimo besedne zveze dolge 7 besed, kar predstavlja dovolj besed za pokritje večine besednih zvez, ki jih iščemo. Tu velja opozoriti, da lahko s prevelikim številom sosednjih besed močno povečamo število značilk, kar posledično upočasni učenje, hkrati pa poveča nevarnost prekomernega prilagajanja. Funkcija torej vrača tip 3-grama, ki predstavlja tri zaporedne besede.

- Lema: Atribut lema predstavlja lemo izbrane besede. Razlika z uporabo besede same je ta, da se tu izognemo razlikovanju besed v različnih oblikah ter na ta način prepoznamo, da gre za isto besedo. Sistem bo torej prepoznal besedno zvezo: "Zakona o varnosti v cestnem prometu"ter zvezo: "Zakonu o varnosti v cestnem prometu"kot isti, saj bo lematizirana različica te besedne zveze enaka: "zakon o varnost v cesten promet".

- Lema pred izbrano besedo: Podobno kot pri besedah samih, tudi tu na ta način pridobimo interakcijo med besedami in s tem odkrijemo morebitne besedne zveze. Razlika med interakcijami odkritimi z besedami samimi je ta, da na ta način lahko lažje najdemo interakcijo med istimi besedami, ki so v različnih oblikah. Hkrati pa dobimo tudi nekaj interakcij, ki morda niso enake drugi, vendar se v lematizirani obliki prepoznata kot isti.

- Lema po izbrani besedi: Podobno kot pri zgornjih atributih tudi tu iščemo interakcije, le z besedo za izbrano.

- Pozicija besede v stavku: Ta atribut nam pove, kje v stavku se beseda pojavlja. Atribut je informativen, kadar je beseda prva ali zadnja v stavku. Tu velja omeniti, da pri prejšnjih atributih, ker upoštevamo sosednje besede, v primeru začetka oziroma konca stavka, sosednje besede ignoriramo in jih predstavimo kot znak za prazen niz: _.

- Oblikoskladenjska oznaka: Atribut "Oblikoskladenjske oznaka" nam poda nekaj informacije o besedi sami ter njeni umestitvi v stavku. Pojavitve istih oblikoskladenjskih oznak nam lahko pomagajo poiskati nekatere besedne zveze, ki jih iščemo. Primer oblikoskladenjskih oznak, ki jih sistem določi za besedno zvezo "Zakona o varnosti v cestnem prometu", je prikazana v tabeli 3.3.

Zakon	o	varnosti	v	cestnem	prometu
Somer	Dm	Sozem	Dm	Ppnmem	Somem

Tabela 3.3: Primer označenega stavka s POS oznakami.

- Oblikoskladenjska oznaka besede pred izbrano: Ta atribut nam poda oblikoskladenjske oznake besed pred trenutno obravnavano besedo. Služi za iskanje odvisnosti med različnimi oblikoskladenjskimi oznakami. Ta funkcija vrača dve sosednji oblikoskladenjski oznaki, zato je tip funkcije bi-gram.

- Oblikoskladenjska oznaka besede za izbrano: Podobno kot zgornji atribut, tudi ta služi za iskanje odvisnosti med oblikoskladenjskimi oznakami, le da v tem primeru obravnavamo oznako za obravnavano besedo. Ta funkcija vrača dve sosednji oblikoskladenjski oznaki, zato je tip funkcije bi-gram.

- Oblikoskladenjske oznake daljših sekvenc: Če želimo iskati odvisnosti daljših besednih zvez, lahko uporabimo ta atribut, ki nam poda oblikoskladenjske oznake dveh besed pred in po obravnavani besedi. Vsega skupaj funkcija vrne 5 sosednjih oblikoskladenjskih oznak, torej je tip funkcije 5-gram.

- Ali je beseda število: S tem atributom preverimo, če je obravnavana beseda število. V primeru, da je, vrne "True", če beseda ni število vrne "False".

- Ali je beseda ločilo: Podobno kot pri prejšnjem atributu, tudi tu vrača funkcija "True/False", le da tu preverjamo ali je beseda ločilo oziroma vsebuje ločilo. Pri razbitju teksta na osnovne enote namreč v nekaterih primerih kako ločilo ostane skupaj z besedo, npr: "št.". Ločila v besedi iščemo z regularnim izrazom. Primer preverjanja ločila je prikazana v seznamu 3.3.

Seznam 3.3: Logika, ki pove, ali je obravnavana beseda ločilo

```
1 If(Regex("[+−/,.?!()*\;:]").IsMatch(w.GetWord()))
2         w.AddFeature("LOCILA", True);
3 else
4         w.AddFeature("LOCILA", False);
```

- Oblika besede: Ta atributska funkcija nam po določeni gramatiki pove kakšna je oblika besede. Način zapisa oblike besede je povzet po metodi, ki jo uporablja knjižnica StanfordNer. Gre za dokaj preprost način, ki nam z določenimi znaki pove obliko besede – ali ima velike začetnice, ali ima števila, ločila ... V primeru, da gre za število, vrnemo znak d, če gre za besedo, ki se začne z veliko začetnico, vrnemo: Ww. Če obravnavamo besedo z malo začetnico, vrnemo: w. Kadar besedo sestavljajo le besede z velikimi tiskanimi črkami, vrnemo: W. Za ločila vračamo le ločila.

  Za lažjo predstavitev je v seznamu 3.4 primer kode, ki skrbi za gradnjo atributa oblike besede.

  Da pridobimo še več informacije, zgradi atributska funkcija še dodatne atribute, ki predstavljajo oblike besed sosednjih besed. To koristi v primeru, ko imamo primer entitete $CLEN$, ki je v večini primerov sestavljena iz števila in besede člen - 14. člena.

- Ali je beseda rimsko število: Ta atributska funkcija se je pokazala za precej uporabna na področju obravnavanja pravnih besedil. V le-teh se

Seznam 3.4: Logika, ki skrbi za grajenje atributov za obliko besede.

```
1  StringBuilder sb = new StringBuilder("WT-");
2  char lastM = '~';
3  bool nonLetters;
4  int len = s.Length;
5  for (int i = 0; i < len; i++)
6  {
7          char c = s[i];
8          char m = c;
9          if (char.IsDigit(c))
10                 m = 'd';
11         else if (char.IsLower(c) OR c == '_')
12                 m = 'x';
13         else if (char.IsUpper(c))
14                 m = 'X';
15         if (m != 'x' AND m != 'X')
16                 nonLetters = true;
17         if (m != lastM)
18                 sb.Append(m);
19         lastM = m;
20 }
21 if (len <= 3)
22         sb.Append("\\:").Append(len);
```

Seznam 3.5: Logika, ki pove, ali je obravnavana beseda rimsko število

```
1 If(Regex("\b[IVXLCDM]+\b").IsMatch(w.GetWord()))
2         w.AddFeature("RIMST", True);
3 else
4         w.AddFeature("RIMST", False);
```

namreč pogosto pri notacijah in imenih raznih dokumentov uporabljajo rimska števila. Funkcija preveri z uporabo regularnih izrazov, če je trenutna beseda rimsko število. Primer uporabe te atributske funkcije je za boljšo predstavo prikazano v seznamu 3.5.

- Uporaba leksikonov: Kadar iščemo določene imenske entitete ter imamo seznam, ki vsebuje velik nabor teh entitet, lahko uporabimo to funkcijo. Funkcija preveri, če se trenutno obravnavana beseda nahaja v besedni zvezi, ki je vsebovana v seznamu oziroma v prej sestavljenem leksikonu. V primeru, da je to res, dobijo vse sosednje besede, ki se nahajajo v tej besedni zvezi, vrednost atributa "True". S to atributsko funkcijo smo močno izboljšali iskanje nekaterih daljših besednih zvez, ki vsebujejo veliko pogostih besed, kot so: in, na, od, o ... Besede, ki jih preverjamo, moramo prej obvezno lematizirati, saj so gesla v leksikonu prav tako sestavljena iz lem besed.

  Potek preverjanja vsebovanosti besedne zveze v leksikonu je zaradi optimizacije prirejen tako, da najprej sestavimo slovar, kjer ključ predstavlja prva beseda, vrednost pa vse besede, ki se pojavijo v leksikonu za prvo besedo. Isto naredimo tudi za drugo in tretjo besedo. S tem načinom takoj preskočimo iskanje dolgih besednih zvez, ko najdeno dve besedi, ki sta si sosednji, vendar taka kombinacija ne obstaja v leksikonu.

  Iskanje rekurzivno nadaljujemo, dokler ne pridemo do zadnje besede v besedni zvezi. Vse sosednje besede v besedni zvezi pa dobijo vrednost atributa "True"le v primeru, če ta besedna zveza obstaja v leksikonu.

Tak način je dobrodošel zato, ker je naša predstavitev nekega besedila v obliki stavkov, ki vsebujejo objekte besed. Za vsako besedo nato zgradimo posebej atributske funkcije.

- Razbitje oblikoskladenjskih oznak: Ker slovenščina vsebuje veliko več oblikoskladenjskih oznak kot na primer angleščina, se v primeru uporabe oblikoskladenjskih oznak za atribut pojavi problem, ko naš označevalnik ne prepozna podobnosti med dvema samostalnikoma, ki sta v drugem sklonu. Funkcija za razbitje oblikoskladenjskih oznak nam kratico *Somer* razbije na več atributov, kjer dobimo informacijo, da je beseda: samostalnik, vrsta je občno ime, spol je moški, število je ednina, sklon je dajalnik. Na ta način močno zmanjšamo razlike med besednimi vrstami. En atribut nam vedno pove, ali je beseda samostalnik, ne glede na sklon, spol, število.

  Ko zgradimo učno množico z vsemi pripadajočimi atributi, lahko začnemo s procesom učenja. Orodje CRFSuite je precej dobro dokumentiran. Omogoča velik nabor nastavitev, testiranja, prilagoditev učnih algoritmov ter algoritmov preiskovanja prostora. Zaradi uspešnosti z osnovnimi nastavitvami veliko sprememb nismo uporabili v naši rešitvi. Smo pa uporabili možnost vračanja verjetnosti za posamezne entitete pri vsaki besedi v besedilu.

  Ta funkcionalnost služi predvsem v primerih dolgih besednih zvez, kjer zaradi določenih razlogov orodje CRFSuite ne zna določiti pravilne entitete za besede na koncu besedne zveze. Vendar pa po pregledu verjetnosti za določene entitete ugotovimo, da je verjetnost za določeno entiteto še vedno dovolj velika, da lahko spremenimo prag, ki določa ali je beseda določena entiteta ali ne.

  V seznamu 3.6 je prikazan primer za besedo “Zakona”, ki prejme oblikoskladenjsko oznako “Somer”.

Seznam 3.6: Razlaga oblikoskladenjske oznake Somer

```
samostalnik
vrsta=obcno ime
spol=moski
stevilo=ednina
sklon=rodilnik
```

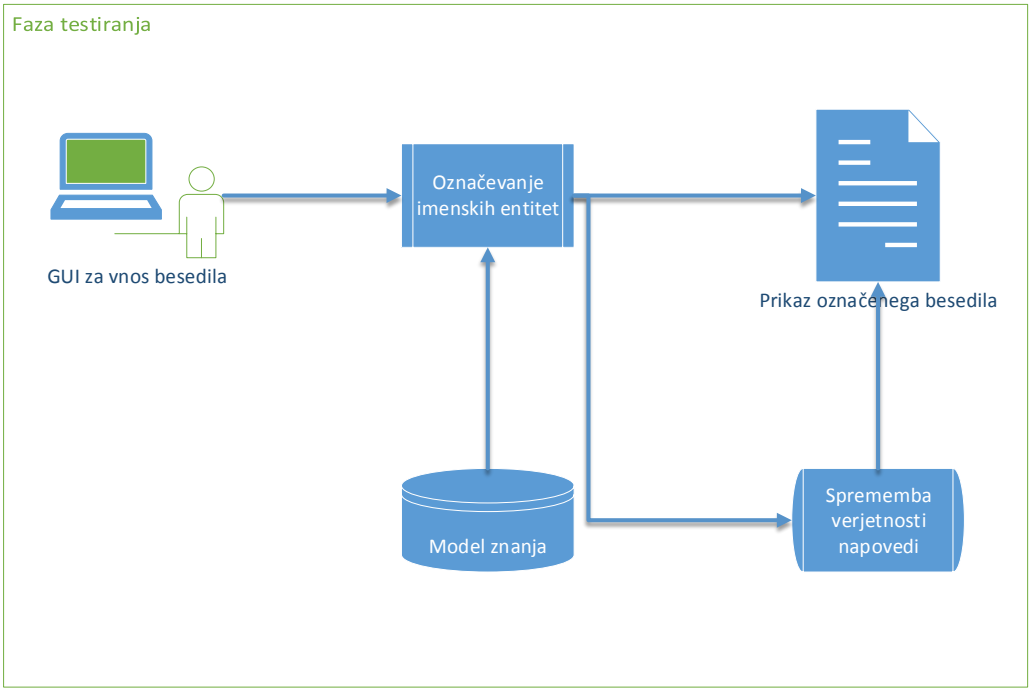
3.4.3 Komponenta za testiranje zgrajenega modela znanja

Ta komponenta (Slika 3.3) je precej preprosta in služi predvsem za vizualno testiranje zgrajenega modela znanja. Vsebuje okno za vnos poljubnega teksta, na katerem nato izvedemo označevanja imenskih entitet.

V ozadju se uporabi enak sistem za grajenje atributskih funkcij, kot je bil uporabljen v procesu učenja. Torej je tudi tu pomembno, ali smo izbrali časovno zahtevne atributske funkcije ali ne. Komponenta nam nato vrne označeno besedilo, kjer lahko vidimo našo uspešnost. Hkrati pa omogoča uporabo verjetnosti za napoved posamezne entitete, ki ji lahko nastavimo nižji prag in s tem prepoznamo še nekatere prej neoznačene entitete. To služi predvsem analizi, ki pokaže, da bi sistem z večjim naborom učnih primerov verjetno tudi bolj točno napovedal določene entitete oziroma dele besednih zvez, ki niso v celoti pravilno označeni.

Primer avtomatsko označenih imenskih entitet z zgrajenim modelom znanja je prikazan v 3.4.3:

Zoper navedeno sodbo je revident vložil revizijo . Navaja , da je vrednost spora 234.871,30 EUR . Zatrjuje zmotno uporabo materialnega prava . Meni , da izplačila nerezidentom nimajo vira v Sloveniji ter da je razlaga določbe 1. točke prvega odstavka 10. členaZDoh - 2 , kakršno



Slika 3.3: Diagram, ki prikazuje potek delovanja faze testiranja.

so sprejeli davčna organa in prvostopenjsko sodišče , napačna . Po tej določbi ima izplačan dohodek vir v Sloveniji le , če je bil izplačan iz virov dohodka izplačevalca znotraj Slovenije , v obravnavnem primeru pa je rezident dohodke nerezidentom izplačal iz dohodkov , ki jih je dosegel v oziroma preko svojih predstavništev v tujini . Dohodki , doseženi preko poslovne enote v tujini , imajo v skladu z določbami Zakona o davku od dohodkov pravnih oseb (v nadaljevanju ZDDPO - 2) vir v tujini . Pojasnjuje , kakšne posledice ima interpretacija 1. točke prvega odstavka 10. člena ZDoh - 2 , ki jo je potrdilo prvostopenjsko sodišče , in sicer dvojno oziroma celo trojno obdavčitev dohodkov . Napačno je bil uporabljenZPDDP . Analogna uporaba pravil o viru dohodka iz ZDoh - 2 je v nasprotju z ustavnim načelom zakonitosti . Poleg tega ZPDDP določa , da se ta davek plačuje od izplačil fizični osebi za opravljeno storitev na podlagi pogodbe o delu po zakonu o delovnih razmerjih in zakonu o obligacijskih razmerjih , v obravnavanem primeru pa so se pogodbe sklepale po tuji zakonodaji , kar je skladno tudi z določili Zakona o mednarodnem zasebnem pravu in postopku (v nadaljevanju ZMZPP) . Kršenih je bilo več načel davčnega postopka : načelo gotovosti (ker ni bilo sprejeto nobeno stališče v zvezi z obdavčevanjem nerezidentov za delo , ki ga opravijo v tujini) , načelo sorazmernosti (ker ni bila uporabljena razlaga sporne določbe , ki je za zavezanca manj obremenjujoča) , načelo materialne resnice (ni bil ugotovljeno , iz katerih dohodkov je rezident izplačal dohodke nerezidentom) . Kršenih je bilo tudi več določb Ustave Republike Slovenije : 2. člen (zaradi prekomernega posega v lastninski pravico) , 14. člen (zaradi neenakega obravnavanje nerezidentov , ki so zaposleni oziroma so samozaposleni , za katere velja oprostitev dohodnine po 33. členu ZDoh - 2) , 15. člen (zaradi prekomernega posega v lastninsko pravico) in 153. člen (prepoved analogne uporabe davčnih predpisov) . Če bi Vrhovno sodišče menilo , da so davčna organa in prvostopenjsko sodišče pravilno razlagali zakon

, predlaga , da zaradi neustavnosti predpisov prekine postopek in začne postopek za oceno ustavnosti . Vrhovnemu sodišču predlaga , da reviziji ugodi in izpodbijano sodbo spremeni tako , da tožbi ugodi , oziroma podrejeno , da izpodbijano sodbo v celoti razveljavi in zadevo vrne v novo sojenje , toženi stranki pa v vsakem primeru naloži plačilo stroškov tega postopka . Priglaša tudi stroške revizijskega postopka .

V nadaljevanju bomo analizirali rezultate, ki jih dobimo z uporabo določenih atributov oziroma brez uporabe le-teh . Z analizo lahko ugotovimo, katerim atributskim funkcijam se lahko izognemo in s tem zmanjšamo časovno zahtevnost celotnega procesa.

Poglavje 4

Rezultati

4.1 Opis rezultatov s prikazom izboljšav preko različnih atributskih funkcij

Rezultati naših rešitev so razdeljeni v dva dela. V prvem delu bomo predstavili rezultate dobljene z orodjem Stanford CoreNLP, kjer smo poizkušali z različnimi nabori atributskih funkcij ter z uporabo oblikoskladenjskih oznak in brez njih. V tem primeru je bil nabor imenskih entitet manjši kot v drugem delu, kjer so prikazani rezultati, dobljeni s sistemom CRFsuite, kjer smo zaradi boljših začetnih rezultatov poizkusili tudi z večjim naborom imenskih entitet.

Celotno besedilo, na katerem smo testirali uspešnost označevalcev, je vsebovalo 85096 besed. Besedilo je bilo sestavljeno iz več različnih besedil pravnih dokumentov.

V teh besedilih smo ročno označili naslednje entitete: Zakon, Člen, Odstavek, Odločba, UradniList. Število besed, ki so bile označene s posamezno entiteto, je sledeče:

- ZAKON: 1544 besed
- ČLEN: 1509 besed
- ODSTAVEK: 802 besedi

- ODLOČBA: 635 besed
- URADNILIST: 1300 besed

Ostale besede so bile označene z entiteto O ("Other"oziroma "Ostalo"). Pri računanju uspešnosti smo uporabili tri metrike (Tabela 4.1):

- Natančnost, s katero izvemo, koliko dobljenih entitet je pravilnih.
- Priklic, ki nam pove, koliko entitet, ki smo jih identificirali, je tudi znanih.
- Ocena F1, ki se izračuna kot geometrijsko povprečje preklica in natančnosti.

	Dejanski razredi	
Napovedani razredi	TP (angl. true positive)	FP (angl. false positive)
	FN (angl. false negative)	TN (angl. true negative)

Tabela 4.1: Razlaga priklica in točnosti za ocenjevanje uepešnosti.

Točnost in priklic računamo s spodnjimi formulami:

$$Tocnost = \frac{TP}{TP + FP} \tag{4.1}$$

$$Priklic = \frac{TP + TN}{TP + FN + FP + FN} \tag{4.2}$$

Oceno F1 pa s formulo:

$$F = 2 * \frac{Tocnost * Priklic}{Tocnost + Priklic} \tag{4.3}$$

4.2 Rezultati testiranja z orodjem Stanford CoreNLP

Rezultati, dobljeni z orodjem Stanford CoreNLP, so bili z izbranim naborom atributskih funkcij precej slabi. Prav tako je bilo samo izvajanje sistema

precej počasno, zato podrobnejših testiranj nismo izvajali. Naredili smo dva testa z dvema modeloma znanja, ki sta bila zgrajena z različnim naborom atributskih funkcij.

Pri obeh testih smo uporabili le vgrajene atributske funkcije ter slovenski označevalnik oblikoskladenjskih oznak. Poleg besede same in oblikoskladenjskih oznak smo uporabili tudi leme besed.

V prvem primeru so bile uporabljene atributske funkcije naštete v seznamu 4.1.

Seznam 4.1: Našteti uporabljeni atributi v prvem poskusu

```
1  useNGrams=true
2  useDisjunctive=true
3  usePrev=true
4  useEntityTypes=true
5  useNext=true
6  useSequences=true
7  useLongSequences=true
8  useIfInteger=true
9  maxLeft=2
10 maxRight=2
11 useWordPairs=true
12 useLC=true
13 wordShape=chris2useLC
14 useLemmas=true
15 usePrevNextLemmas=true
16 usePosition=true
17 useGazettes=true
18 gazette = data/ner/GazeteZakoniTrue.txt;data/ner/zakonigazeetkratice.txt
19 cleanGazette=true
20 type=crf
21 sigma = 20
22 featureDiffThresh=0.05
```

Rezultati za to testiranje so prikazani v tabeli 4.2.

Entiteta	Natančnost	Priklic	F1
CLEN	0.3636	0.0209	0.0396
ODSTAVEk	0.8333	0.3125	0.4545
ODLOČBA	0.0000	0.0000	0.0000
URADNILIST	0.0000	0.0000	0.0000
ZAKON	0.6783	0.5455	0.6047
Skupaj	0.6471	0.3167	0.4253

Tabela 4.2: Rezultati označevanja imenskih entitet, dobljeni s prvim poskusom.

V drugem primeru smo uporabili nekoliko drugačen nabor značilk, ki je prikazan v seznamu 4.2.

Glavna razlika je bila v uporabi druge funkcije za določanje oblik besed, ter neuporaba lem sosednjih besed. Rezultati tega testiranja so prikazani v tabeli 4.3.

Entiteta	Natančnost	Priklic	F1
ČLEN	0.4286	0.0314	0.0585
ODSTAVEK	0.7143	0.3125	0.4348
ODLOČBA	0.0000	0.0000	0.0000
URADNILIST	0.0000	0.0000	0.0000
ZAKON	0.6681	0.5280	0.5898
Skupaj	0.6378	0.3109	0.4181

Tabela 4.3: Rezultati označevanja imenskih entitet, dobljeni z drugim poskusom.

V obeh primerih je entiteta ZAKON dosegla okoli 60% oceno F1. Vzrok za to verjetno tiči v uporabi leksikonov, kjer so našteti vsi zakoni ter njihove

Seznam 4.2: Našteti uporabljeni atributi v drugem poskusu

```
1  useNGrams=true
2  useDisjunctive=true
3  usePrev=true
4  useEntityTypes=true
5  useNext=true
6  useSequences=true
7  useLongSequences=true
8  useIfInteger=true
9  maxLeft=2
10 maxRight=2
11 useWordPairs=true
12 useLC=true
13 wordShape=dan2useLC
14 usePosition=true
15 useGazettes=true
16 gazette = data/ner/GazeteZakoniTrue.txt;data/ner/zakonigazeetkratice.txt
17 cleanGazette=true
18 type=crf
19 sigma = 20
20 featureDiffThresh=0.05
```

kratice. Stanfordov CoreNLP vsebuje namreč zelo dobro logiko za grajenje značilk z uporabo leksikonov.

Po drugi stani je entiteta ČLEN, ki navadno predstavlja le število s piko in besedo "člen", dosegla zelo slabe rezultate. Morda bi z bolj intenzivnim testiranjem z različnimi nabori atributskih funkcij dobili dovolj dober rezultat tudi za druge entitete, vendar smo to zaradi pomanjkanja časa in boljših rezultatov opustili.

4.3 Rezultati testiranja s sistemom CRFsuite

Orodje CRFsuite smo uporabili za označevanje oblikoskladenjskih oznak ter imenskih entitet. Ker smo za označevanje oblikoskladenjskih oznak uporabili tudi orodje Obeliks, se bomo tu osredotočili na rezultate, dobljene pri označevanju imenskih entitet. Glavni razlog za to je pomanjkanje časa za več različnih testiranj orodja za označevanje POS oznak, kajti velikost učne množice je botrovala zelo dolgemu izvajanju koraka grajenja modela znanja.

Pri testiranju uspešnosti označevanja imenskih entitet smo uporabili vgrajen mehanizem 10-kratnega prečnega preverjanja v koraku grajenja modela znanja.

Samo testiranje smo razdelili v štiri dele. Testirali smo uspešnost sistema pri grajenju modela znanja z uporabo vseh atributskih funkcij. Nato pa še različne primere, ko smo izpustili določene atributske funkcije. Rezultati gradnje teh modelov znanja so navedeni v naslednjih razdelkih.

4.3.1 Grajenje modela znanja z uporabo vseh atributskih funkcij

V tem primeru smo zgradili sistem znanja z vsemi atributskimi funkcijami. Na uspešnost tega modela nato referenciramo vse naslednje dobljene rezultate, saj so bili tu rezultati najboljši. To smo tudi pričakovali, saj je sistem za grajenje modela v tem primeru dobil največ informacije.

Kakovost rezultatov smo merili enako kot v primeru Stanford CoreNLP z metrikami "priklic", "natančnost"in "ocena F1". V tabeli 4.4 so vidni rezultati za šest entitet.

Entiteta	Natančnost	Priklic	F1
O	0.99373	0.9973	0.99551
ZAKON	0.94997	0.8833	0.91265
ČLEN	0.98541	0.95462	0.96937
ODSTAVEK	0.98056	0.96999	0.97501
ODLOČBA	0,96649	0.94736	0.9546
URADNILIST	0.92913	0.91596	0.91628
Skupaj	0.96754	0.94476	0.953903

Tabela 4.4: Rezultati označevanja imenskih entitet z uporabo CRFsuite, dobljeni z uporabo vseh atributskih funkcij.

Iz rezultatov je razvidno, da se je sistem najbolje obnesel pri označevanju entitet ČLEN in ODSTAVEK (entitete O ne upoštevamo zaradi prepogoste pojavitve). Razlog za to je verjetno v preprostih pojavitvah teh entitet, saj se obe entiteti pojavljata v dokaj standardni obliki, ki se jo sistem lahko hitro nauči.

Rezultat označevanja entitet ODLOČBA je prav tako presenetljiv, saj gre za precej zapleteno entiteto, ki je sestavljena iz števil, črk in rimskih števil. Zato je zanimivo, da je dosegla prav tako precej veliko uspešnost.

Najbolj nas pa zanimata uspešnosti označevanja entitet ZAKON in URADNILIST, saj s kombinacijo teh dveh dobimo največ informacije, ki jo želimo iz teksta. Z uporabo vseh značilk smo pri odkrivanju obeh entitet dosegli uspešnost preko 91%, kar je precej dober rezultat.

Iz kasnejšega preverjanja rezultatov v besedilu smo tudi ugotovili, da verjetnost verjetno precej pokvari pojavljanje dolgih oblik teh entitet. Primer, ko imamo v besedilu navedbo zelo dolgega zakona, se pogosto zgodi, da sistem označi vse besede razen zadnje ali zadnjih dveh. To bi verjetno lahko

izboljšali z uporabo večje učne množice.

4.3.2 Grajenje modela znanja brez uporabe interakcij med besedami

Pri testiranju gradnje modela znanja brez uporabe atributskih funkcij, ki nam podajo interakcijo med besedama, smo izpustili uporabo funkcij: "Beseda pred izbrano", "Beseda za izbrano", "Več sosednjih besed", "Lema pred izbrano", "Lema po izbrani"ter iste funkcije za oblikoskladenjske oznake.

S tem smo izgubili nekaj informacije o tem, kako določene pojavitve besed v zaporedju določijo entiteto, vendar se vrednosti rezultatov niso občutno zmanjšale. Prikaz je v tabeli 4.5.

Entiteta	Natančnost	Priklic	F1
O	0.99391	0.99705	0.99548
ZAKON	0.94534	0.88315	0.91027
ČLEN	0.9847	0.95795	0.97082
ODSTAVEK	0.97887	0.96999	0.97416
ODLOČBA	0.9481	0.94348	0.94362
URADNILIST	0.92974	0.92218	0.92004
Skupaj	0.96344	0.94563	0.95239

Tabela 4.5: Rezultati označevanja imenskih entitet z uporabo CRFsuite, dobljeni brez uporabe atributske funkcije za iskanje interakcij.

Iz tabele rezultatov je predvsem zanimivo dejstvo, da se je uspešnost pri entitetah ČLEN in URADNILIST celo izboljšala. Verjetno je glavni razlog, ki je povzročil izboljšanje, pojavitev števil v teh dveh entitetah. Primer, če imamo dva primera entitete ČLEN: "13. člen"in "15. člena", nam konjunkcija teh dveh besed v obeh entitetah ne pove nič, saj sta si različni. Koristno informacijo bi dobili le v primeru istega člena. Podobno je tudi pri entiteti URADNILIST, kjer se prav tako pojavljajo števila oznak uradnega lista.

Uspešnosti pri iskanju drugih entitet se niso bistveno spremenile in še vedno ostajajo precej dobre. Pričakovali smo predvsem spremembo v iskanju entitete ZAKON, kjer bi interakcija sosednjih besed lahko prinesla nekaj koristne informacije.

4.3.3 Gradnja modela znanja brez uporabe oblikoskladenjskih oznak

Testiranje uspešnosti gradnje modela brez uporabe oblikoskladenjskih oznak nas je najbolj zanimal. Proces označevanja oblikoskladenjskih oznak je namreč najbolj zamuden v celotnem procesu označevanja imenskih entitet. V primeru, da uspešnost ne bi preveč padla brez informacije o POS oznakah, bi lahko ta proces tudi izpustili. Rezultati so prikazani v tabeli 4.6.

Entiteta	Natančnost	Priklic	F1
O	0.99363	0.99709	0.99537
ZAKON	0.94328	0.87565	0.90551
ČLEN	0.98542	0.95594	0.97012
ODSTAVEK	0.97643	0.96999	0.9729
ODLOČBA	0.968	0.91945	0.94039
URADNILIST	0.93107	0.93278	0.9255
Skupaj	0.966305	0.94182	0.951632

Tabela 4.6: Rezultati označevanja imenskih entitet z uporabo CRFsuite, dobljeni brez uporabe atributske funkcije za določanje oblikoskladenjskih oznak.

Rezultati označevanja imenskih entitet brez uporabe POS oznak so precej presenetljivi. Tudi tu se je pri dveh entitetah uspešnost izboljšala glede na rezultate uporabe vseh značilk pri označevanju. Entiteti URANDILIST in ČLEN sta dosegli boljši rezultat, URADNILIST celo skoraj za cel odstotek. Med tem je entiteta ZAKON izgubila ne uspešnosti za nekaj več kot pol odstotka. Na uspešnosti sta izgubili tudi entiteti ODSTAVEK in ODLOČBA,

vendar ne veliko.

Rezultati so pokazali, da morda uporaba oblikoskladenjskih oznak ni tako zelo pomembna v našem primeru odkrivanja imenskih entitet. V primeru, če bi se temu koraku izognili, bi hkrati pridobili precej na samem času izvajanja celotnega procesa.

4.3.4 Gradnja modela znanja brez uporabe leksikonov

Zadnji primer testiranja uspešnosti smo opravili na gradnji modelov znanja brez uporabe leksikonov. Zaradi uporabe leksikonov vseh zakonov smo pričakovali upad uspešnosti predvsem pri entiteti ZAKON ter entitetah, ki se pogosto pojavljajo v okolici entitet ZAKON. Rezultati so navedeni v tabeli 4.7.

Entiteta	Natančnost	Priklic	F1
O	0.99215	0.99732	0.99473
ZAKON	0.9534	0.81	0.87044
ČLEN	0.98273	0.95042	0.966
ODSTAVEK	0.97799	0.96761	0.97249
ODLOČBA	0.96328	0.94822	0.95368
URADNILIST	0.92913	0.91596	0.91628
Skupaj	0.96644667	0.93159	0.945603

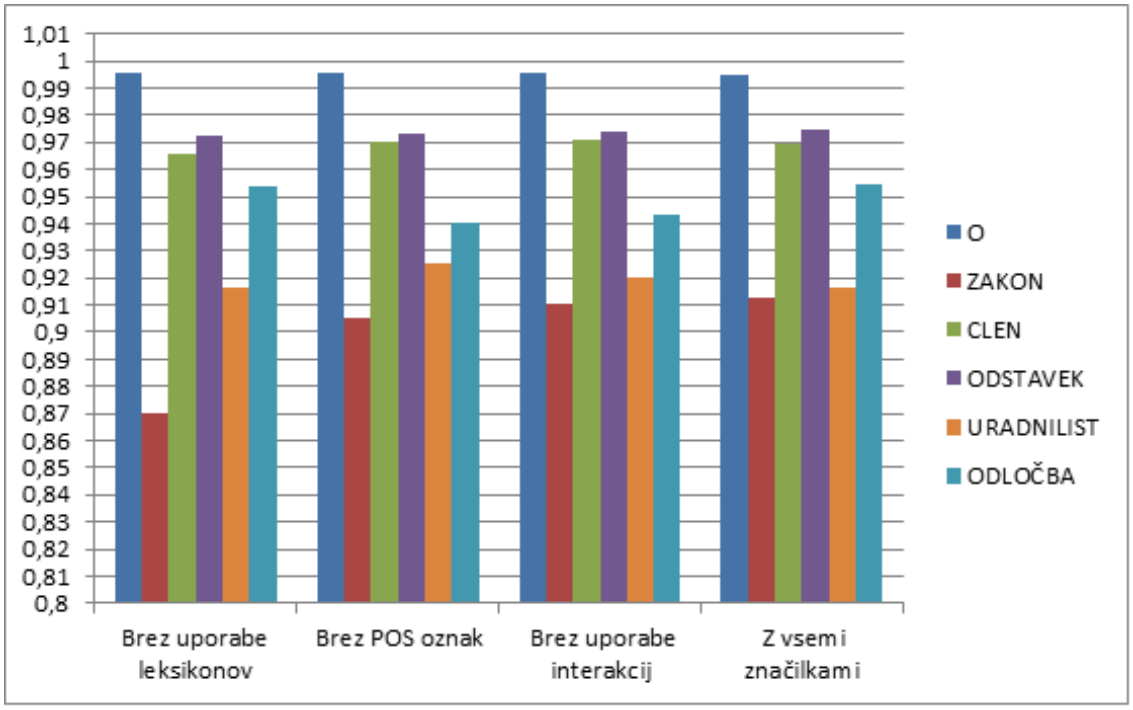
Tabela 4.7: Rezultati označevanja imenskih entitet z uporabo CRFsuite, dobljeni brez uporabe atributske funkcije, ki omogoča uporabo leksikonov.

Kot smo pričakovali, je viden močan upad uspešnosti pri entiteti ZAKON. Verjetno bi z izboljšavo same atributske funkcije, ki uporablja leksikone, to uspešnost še zmanjšali glede na referenčne rezultate, saj trenutna rešitev ne najde vseh entitet v procesu grajenja atributskih funkcij.

Druga zanimiva ugotovitev pri tem poskusu je, da ostale uspešnosti pri odkrivanju entitet ostanejo precej nespremenjene. Še največji upad uspešnosti

je prisoten pri entiteti CLEN, kar je bilo tudi pričakovano, saj se le-ta entiteta največkrat pojavi neposredno ob entiteti ZAKON.

S temi ugotovitvami lahko sklepamo, da je uporaba leksikonov pri odkrivanju imenskih entitet precej pomembna. Za konec si lahko ogledamo predstavitev vseh rezultatov še v obliki grafikona 4.1. Uspešnosti so razdeljene v skupine glede na to, katero značilko smo v testiranju izpustili.



Slika 4.1: Prikaz uspešnosti, glede na uporabo različnih atributskih funkcij. Rezultati so prikazani glede na uspešnost odkrivanje posamezne entitete. Za merilo je uporabljena ocena uspešnosti F1.

4.4 Primerjava s trenutnim načinom in prikaz boljših možnosti te metode

Sistem za označevanje imenskih entitet v pravnih besedilih nam lahko služi za pridobivanje dodatne informacije v besedilu. V našem primeru potrebujemo te entitete za iskanje in določanje dodatnih povezav na druge dokumente. Obstoječ sistem "DRP provider", ki je trenutno v uporabi, nam je v tem delu predstavljal veliko omejitev, saj zaradi nedostopnosti izvorne kode in omejitev, ki so jih razvijalci postavili, nismo mogli razširiti sistema.

Z novo rešitvijo, ki uporablja sistem CRFsuite, imamo sedaj vpogled in možnost dodajanja novih imenskih entitet, ki bodo pripomogle h graditvi dodatnih povezav.

Druga prednost je, da se zaradi transparentnosti rešitve lahko spreminja logika grajenja povezav glede na dodatne zahtevnosti. Dober primer za to je določanje pravilne povezave glede na verzijo zakona, ko gradimo povezavo za določen zakon. Gre namreč za to, da lahko pri odkrivanju navedb entitet tipa ZAKON dobimo dodatno informacijo, če zraven iščemo tudi entiteto URADNILIST, ki v kombinaciji z zakonom pove, o kateri verziji zakona govorimo. Z nekaj dodatne logike, ki nam poišče kombinacije teh dve entitet, lahko močno izboljšamo gradnjo povezav.

Poglavje 5

Zaključek in nadaljnje delo

S pregledom obstoječih rešitev za reševanje problema imenskih entitet smo ugotovili, da je na spletu dostopnih precej tako in drugače zelo dobro realiziranih rešitev. Nekatere so nekoliko boljše, druge so precej prilagojene za trenutno obravnavane jezike. Za naš problem smo ugotovili, da se najbolj obnese, če problem razbijemo na več problemov, ki jih delno realiziramo s svojimi rešitvami, za sam del strojnega učenja pa uporabimo že obstoječo rešitev.

CRFsuite, ki smo ga uporabili za grajenje modela znanja in samo označevanje imenskih entitet, se je obnesel boljše, kot smo pričakovali. V razdelku rezultati smo nato tudi naredili analizo, katere značilke pripomorejo k večji uspešnosti sistema. Ta del je pokazal, da trenutno najbolj časovno potraten del – označevanje oblikoskladenjskih oznak, niti ni tako zelo pomemben za uspešnost samega sistema. To je odlična iztočnica za nadaljevanja dela.

Za uspešno realizacijo iskanja imenskih entitet, ki bo prinesla dodatno informacijo, ki jo lahko uporabimo za grajenje povezav, torej ne potrebujemo koraka, ki nas je do sedaj najbolj oviral. Za doseganje boljših rezultatov se zato lahko usmerimo v izboljšavo sistema uporabe leksikonov, ki so se izkazali kot zelo pomembni pri odkrivanju najbolj pomembne entitete – ZAKON.

Druga stvar, ki bo verjetno pripomogla k izboljšavi rezultatov, pa je večja učna množica. Sedanja učna množica sama po sebi niti ni tako zelo majhna,

vendar pa predstavlja problem pri označevanju manj pogostih imenskih entitet. Entiteti ODSTAVEK in ODLOČBA se namreč pojavljata pri manj kot 1000 besedah, natančneje pri 0.9% in 0.7% besedah, kar je precej malo. Kljub temu rezultati niso tako zelo slabi.

Z delom na povečanju učne množice lahko pridobimo tudi na dodajanju novih entitet. V pravnih tekstih se pogosto pojavljajo tudi entitete USTAVA, UREDBA, itn., ki jih v naših poskusih nismo označevali zaradi premajhne učne množice in že tako omejenega časa za ročno označevanje entitet.

Trenutna realizacija sistema za odkrivanje imenskih entitet v pravnih besedilih nam omogoča implementacijo vseh naštetih izboljšav in precej enostavno dodatno označevanje novih entitet ter dodajanje primerov v učno množico.

Literatura

[1] A. Mejer and K. Crammer. "Confidence in Structured-Prediction using Confidence-Weighted Models". Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010). str. 971-981. 2010.

[2] C. M. Bishop Čhapter 8. Graphical Models". Pattern Recognition and Machine Learning. Springer. str. 359–422. 2006.

[3] C. D. Manning and H. Schütze. Foundations of Statistical Natural Language Processing. The MIT Press. 1999.

[4] G. Andrew and J. Gao. "Scalable training of L1-regularized log-linear models". Proceedings of the 24th International Conference on Machine Learning.

[5] J. Nocedal. "Updating Quasi-Newton Matrices with Limited Storage". Mathematics of Computation. 35. 151. str. 773-782. 1980.

[6] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. "Online Passive-Aggressive Algorithms". Journal of Machine Learning Research. 7. Mar. str. 551-585. 2006.

[7] Y. Lee; Y. Lin; and G. Wahba. "Multicategory Support Vector Machines". Computing Science and Statistics 33. 2001.

[8] L. Rabiner in B. Juang. An introduction to hidden Markov models. ASSP Magazine, IEEE, 3(1): str. 4–16. 1986.

[9] M. Grčar, S. Krek, K. Dobrovoljc: Obeliks: statistični oblikoskladenjski označevalnik in lematizator za slovenski jezik. V T. Erjavec, J. Žganec Gros (ur.): Zbornik Osme konference Jezikovne tehnologije. Ljubljana: Institut Jožef Stefan. 2012.

[10] M. Collins. "Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms". Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2002). str. 1-8. 2002.

[11] J. Perkins. Python Text Processing with NLTK 2.0 Cookbook. Packt Publishing. ISBN 1849513600. 2010.

[12] F. Rosenblatt, The Perceptron–a perceiving and recognizing automaton. Poročilo 85-460-1, Cornell Aeronautical Laboratory. 1957.

[13] S. Bird, E. Klein, and E. Loper. O'Reilly Media, 2009 Natural Language Processing with Python. Dostopno na: http://nltk.org/book/.

[14] R. Shinghal and G. T. Toussaint, "Experiments in text recognition with the modified Viterbi algorithm,"IEEE Transactions on Pattern Analysis and Machine Intelligence, Izdaja PAMI-l, pp. 184–193. April 1979.

[15] S. Shalev-Shwartz, Y. Singer, and N. Srebro. "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM". Proceedings of the 24th International Conference on Machine Learning (ICML 2007).str. 807-814. 2007.

[16] T. Erjavec, S. Krek, Š. Arhar, D. Fišer, N. Ledinek, A. Saksida, B. Sivec, in B. Trebar. Oblikoskladenjske specifikacije JOS. Dostopno na: http://nl.ijs.si/jos/msd/. 2010c.

[17] T. Erjavec, D. Fišer, S. Krek, N. Ledinek: The JOS Linguistically Tagged Corpus of Slovene. Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10), Malta. 2010.

[18] T. Štajner, T. Erjavec and S. Krek. Razpoznavanje imenskih entitet v slovenskem besedilu; In Proceedings of 15th Internation Multiconference on Information Society - Jezikovne Tehnologije, Ljubljana, Slovenia. 2012.

[19] A. Turing, Čomputing Machinery and Intelligence", Mind LIX (236): 433–460. October 1950.

[20] M. URŠIČ, I. MOZETIČ, T. ERJAVEC, N. LAVRAČ. LemmaGen : multilingual lemmatisation with induced Ripple-Down rules. J. univers. comput. sci. (Print), vol. 16, no. 9, str. 1190-1214. 2010.

[21] Wallach, H.M.: Conditional random fields: An introduction. Technical Zbornik MS-CIS-04-21, University of Pennsylvania. 2004.