Karakterizacija likova u dječjim pričama

Levačić, Gorana

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:747138

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Gorana Levačić

KARAKTERIZACIJA LIKOVA U DJEČJIM PRIČAMA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, studeni, 2016.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **v**

Uvod **1**

1 Obrada prirodnog jezika **3**
1.1 Opis područja . . . 3
1.2 Razumijevanje prirodnog jezika . . . 5
1.3 Prepoznavanje imenovanih entiteta . . . 7
1.4 Razrješavanje koreferencije . . . 8
1.5 Analiza sentimenta . . . 9

2 Modeli za prepoznavanje imenovanih entiteta **13**
2.1 Generativni i diskriminativni modeli . . . 13
2.2 Grafički modeli . . . 14
2.3 Klasifikacija . . . 16
2.4 Skriveni Markovljevi modeli . . . 18
2.5 Uvjetna slučajna polja . . . 20
2.6 Učenje i korištenje modela . . . 22

3 Modeli za razrješavanje koreferencije **25**
3.1 Model parova . . . 25
3.2 Model logike prvog reda . . . 29
3.3 Metoda višeprolaznog sita . . . 31

4 Modeli za analizu sentimenta **35**
4.1 Model "vreća riječi" . . . 35
4.2 Model "vreća stavova" . . . 36
4.3 Naivni Bayesov klasifikator . . . 38
4.4 Duboko učenje . . . 38

5 Razvoj aplikacije za karakterizaciju likova **45**
5.1 Korišteni alati . . . 45
5.2 Organizacija projekta . . . 46
5.3 Klasa `MainClass` . . . 48
5.4 Klasa `CharacterRecogniserTrainer` . . . 51
5.5 Klase `StoryData` i `CharacterName` . . . 54
5.6 Klasa `Solver` . . . 56
5.7 Klasa `CharacterRecognition` . . . 58
5.8 Klasa `CoreferenceResolution` . . . 61
5.9 Klasa `SentimentAnalysis` . . . 64
5.10 Klasa `Constants` . . . 67

6 Rezultati **69**
6.1 Pepeljuga . . . 69
6.2 Snjeguljica . . . 74
6.3 Vuk i sedam kozlića . . . 81
6.4 Pljačkaš zaručnik . . . 84

7 Zaključak **87**
7.1 Ocjena implementiranog rješenja . . . 87
7.2 Nastavak istraživanja . . . 88

Bibliografija **91**

Uvod

U ovom diplomskom radu bavit ćemo se problemom automatske karakterizacije likova u dječjim pričama. Cilj je implementirati jednostavno softversko rješenje koje će, za lika iz dječje priče, s nekom točnošću odrediti je li dobar ili zao. Taj problem pripada području **obrade prirodnog jezika**, konkretno, razumijevanju prirodnog jezika.

Problem ćemo pokušati riješiti kombiniranjem nekoliko već razrađenih metoda za obradu prirodnog jezika. Cilj je postojećim alatima analizirati i transformirati tekst u skup podataka koje će računalo znati obraditi. Prvi korak je samo prepoznavanje likova. Taj korak nije nužan, odnosno, možemo sami zadati koji su likovi u priči. Sljedeći korak je razrješavanje koreferencije, odnosno, zamjena svih zamjenica imenima likova na koje se odnose. Naposljetku, za svakog lika možemo automatski izdvojiti rečenice koje se odnose na njega i provesti analizu sentimenta nad njima. Analiza sentimenta će nam dati odgovor na pitanje je li dotičan lik dobar ili zao.

Drugi mogući pristup se sastoji u oblikovanju i treniranju nekog od modela strojnog učenja nad označenim podacima, odnosno, pričama. Problem ovakvog pristupa je u dostupnosti podataka za učenje. S obzirom da se radi o atipičnom problemu, ne postoji javno dostupan odgovarajući skup podataka. Stoga je, u ovom pristupu, potrebno samostalno stvoriti takav skup. Radi se o čitanju većeg broja priča te ručnom označavanju likova i dijelova rečenica koje se odnose na njihovu ličnost. To je dugotrajan, iscrpljujuć zadatak te postoji vjerojatnost da, na samom kraju, istrenirani model ne bi bio znatno uspješniji od modela dobivenog prvim pristupom.

Priče su prikupljene s Project Gutenberga, online repozitorija e-knjiga [12]. Radi se o dječjim bajkama, poput Crvenkapice ili Snjeguljice. U takvim pričama najčešće imamo svega nekoliko likova koji su strogo podijeljeni na dobre i zle. Stoga su one idealne za rješavanje našeg problema.

Kako gotovo svi dostupni alati za obradu jezika rade isključivo s engleskim jezikom, dječje priče koje ćemo obrađivati, kao i svi primjeri u ovom diplomskom radu, dani su na engleskom jeziku.

Poglavlje 1

Obrada prirodnog jezika

1.1 Opis područja

Obrada prirodnog jezika (eng. *natural language processing*) je znanstveno područje koje obuhvaća računarstvo, umjetnu inteligenciju i računalnu lingvistiku. Ovom području pripadaju neki od najvažnijih problema iz područja umjetne inteligencije, poput razumijevanja prirodnog jezika, strojnog prevođenja i prepoznavanja govora. S obzirom da je jezik osnovni način sporazumijevanja među ljudima, uspjeh u području obrade prirodnog jezika značio bi ostvarivanje uspješne komunikacije između ljudi i računala.

Prvi korak u obradi prirodnog jezika je njegovo modeliranje nekim formalnim skupom pravila. U tu svrhu dolazi do oblikovanja formalnih gramatika i semantika. **Formalna gramatika** definira skup pravila za generiranje valjanih rečenica (nizova riječi) danog jezika.

Opišimo ukratko formalnu gramatiku engleskog jezika. Prvo je potrebno definirati objekte (alfabet) formalne gramatike.

- *S (sentence)* — rečenica
- *NP (noun phrase)* — imenička fraza, skup riječi koji unutar rečenice čini subjekt ili objekt
- *VP (verb phrase)* — glagolska fraza, skup riječi koji se unutar rečenice odnosi na predikat
- *PP (prepositional phrase)* — prepozicijska fraza, skup riječi koji pobliže opisuje neku imenicu ili glagol
- *V* ili *Verb* — glagol
- *N* ili *Noun* — imenica

- *Proper-Noun* — vlastita imenica
- *Pronoun* — zamjenica
- *Det (determiner)* — riječ koja pobliže opisuje imenicu na koju se odnosi, na primjer *the, a, this, blue*
- *Prep (preposition)* — prijedlog

Slijedi nekoliko pravila gramatike engleskog jezika:

$$\begin{aligned}
S &\rightarrow NP \;\; VP \\
NP &\rightarrow Pronoun \\
&\quad \mid Proper\text{-}Noun \\
&\quad \mid Det \;\; Noun \\
VP &\rightarrow Verb \\
&\quad \mid Verb \;\; NP \\
&\quad \mid Verb \;\; NP \;\; PP \\
&\quad \mid Verb \;\; PP \\
PP &\rightarrow Prep \;\; NP \\
Prep &\rightarrow from \mid to \mid before \mid in \mid \ldots \\
Verb &\rightarrow do \mid eat \mid look \mid stay \mid \ldots \\
Noun &\rightarrow girl \mid cat \mid bird \mid forest \mid \ldots \\
Pronoun &\rightarrow I \mid she \mid they \mid someone \mid \ldots
\end{aligned} \tag{1.1}$$

Uočimo da se primjenom gornjih pravila rečenice mogu rastaviti na osnovne gramatičke dijelove, ali i obratno — iz riječi možemo generirati nove gramatički ispravne rečenice. Zbog toga se formalne gramatike zovu i **generativne gramatike**.

Nadalje, uočimo da gramatička ispravnost ne znači, ujedno, i semantičku ispravnost. Kao primjer, često se navodi rečenica iz [1],

Colorless green ideas sleep furiously.

koja je gramatički ispravna, ali nema nikakvo smisleno značenje. Iz tog razloga je važno proučavati i semantiku prirodnog jezika. Jedan pristup tome je Montagueva gramatika [10], koja semantiku prirodnog jezika dovodi u vezu s logikom višeg reda. Dodatno se različitim statističkim metodama modelira semantika prirodnog jezika. Takve metode daju modele koji, za neki niz riječi, predviđaju koliko je vjerojatno da se pojavi u prirodnom jeziku, odnosno, koliko je vjerojatno da je smislen.

Nažalost, dosad opisani pristupi ne daju sasvim uspješne modele. Razlog tome je izuzetna složenost prirodnog jezika, koja često ljudima nije očita — kompleksna gramatička pravila, veliki broj iznimaka od tih pravila, višeznačnost riječi i izraza predstavljaju računalima velik problem. Zapravo je nemoguće eksplicitno modelirati čitavu gramatiku i semantiku jednog jezika, te je potrebno promijeniti ili barem nadopuniti ovakav pristup. Kao rješenje se javlja **strojno učenje** (eng. *machine learning*), koje se krajem prošlog stoljeća počelo snažno razvijati. Svrha strojnog učenja je oblikovati algoritme koji mogu učiti iz podataka te davati predviđanja nad novim podacima. Treniranjem nad velikim skupom podataka računalo može oblikovati odličan model prirodnog jezika, odnosno, njegove strukture, pravila i semantike. Stoga se danas najveći uspjesi, ne samo u obradi prirodnog jezika, već u čitavom području umjetne inteligencije, postižu upravo korištenjem strojnog učenja.

U nastavku ćemo opisati nekoliko važnijih problema iz ovoga područja vezanih uz problem karakterizacije likova.

1.2 Razumijevanje prirodnog jezika

Razumijevanje prirodnog jezika (eng. *natural language understanding*) je jedan od najvažnijih i najkompleksnijih problema obrade prirodnog jezika. Da bi računalo moglo "razumjeti" tekst, potrebno ga je reprezentirati nekim formalnim jezikom, na primjer, logikom prvog reda. Rješavanje problema otežava činjenica da gotovo svaka rečenica može imati nekoliko mogućih značenja. Često nama (ljudima) neka rečenica ima samo jedno očito značenje, ali računalo može prepoznati i nekoliko desetaka različitih značenja i često neće uspješno odrediti točno značenje. Najčešći razlozi tome su višeznačnost riječi, postojanje frazema i različitih stilskih figura, te nejedinstvenost stabla parsiranja.

Višeznačnost riječi, pogotovo u engleskom jeziku, potječe iz više izvora. Mnoge engleske riječi u svom izvornom obliku ne pripadaju jednoj, već nekoliko vrsta riječi, te u svakoj od tih vrsta mogu imati više značenja. Na primjer, riječ *iron*, ovisno o kontekstu, može biti imenica, glagol ili pridjev. Značenja imenice *iron* su *željezo*, *glačalo*, ili pak *palica za golf*, dok kao glagol, riječ *iron* ima značenje *glačati*. Pridjev *iron* može značiti *željezni*, ali i *čvrst*, *odlučan*.

Uz višeznačnost riječi je vezan i pojam *frazema*. Frazemi, u engleskom jeziku poznatiji kao *idiomi*, su višečlane jezične jedinice koje se uvijek javljaju kao cjelina te im je stvarno značenje različito od doslovnog značenja tog niza riječi. Ispuštanje ili zamjena samo jedne riječi u frazemu dovodi do gubljenja njegovog značenja. Na primjer, engleski idiom *piece of cake* označava jednostavnu i laganu aktivnost, zadatak ili posao. Analogni hrvatski frazem bi bio *mačji kašalj*. U oba slučaja

očito je da frazem ima posve različito značenje od doslovnog značenja. U rečenici poput

The exam was piece of cake.

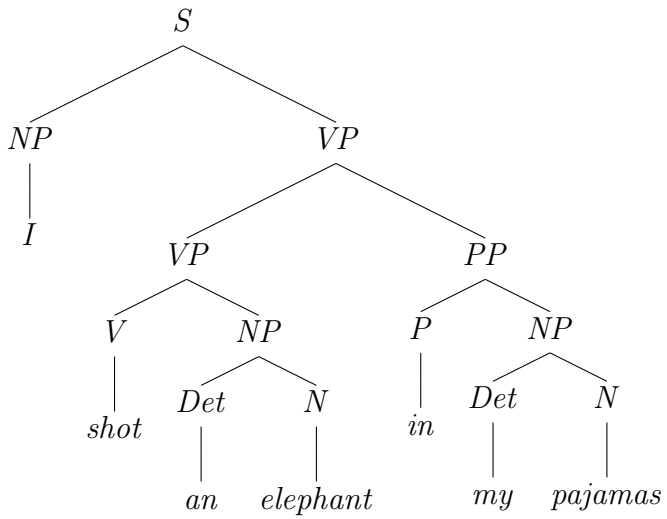
čovjeku je očito da se radi o figurativnom značenju. S druge strane, u nekim situacijama će riječi, koje inače čine frazem, imati svoje doslovno značenje, kao u rečenici:

I went on her birthday party and ate large piece of cake.

Posljednji razlog tomu što računala najčešće prepoznaju više značenja iste rečenice je taj, što se za istu rečenicu može izgraditi više *stabala parsiranja*. Stablo parsiranja predstavlja gramatičku strukturu rečenice. Pogledajmo dva moguća stabla parsiranja za poznatu rečenicu iz filma *Animal Crackers* braće Marx:

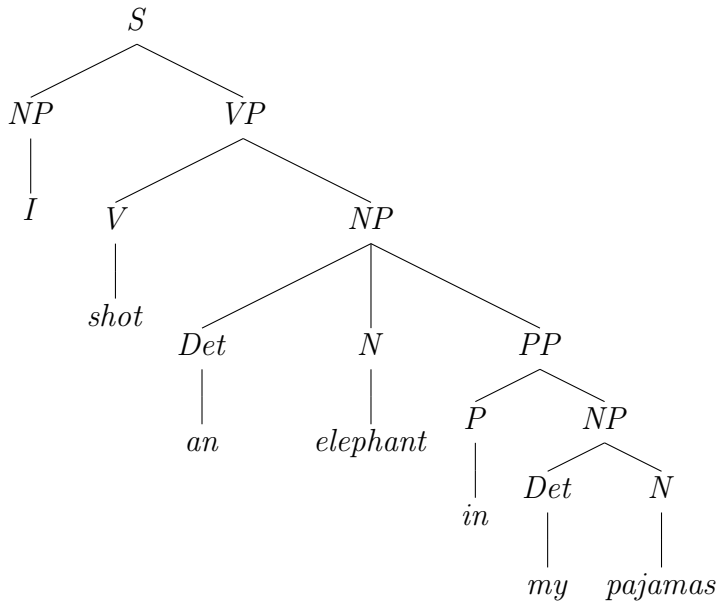
I shot an elephant in my pajamas.

Oznake čvorova su definirane u prethodnoj sekciji.



Ovo stablo parsiranja odgovara značenju

I was in my pajamas when I shot the elephant.



S druge strane, ovo stablo parsiranja odgovara značenju

> *I shot the elephant which was wearing my pajamas.*

Uočimo da još ne možemo prevesti ovu rečenicu na hrvatski jezik, jer nam nije poznato značenje riječi *shot*. Naime, možemo ju shvatiti kao *upucati* (puškom tijekom lova) ili kao *fotografirati*. Tek uz kontekst možemo točno interpretirati značenje:

> *While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know.*

1.3 Prepoznavanje imenovanih entiteta

Prepoznavanje imenovanih entiteta (eng. *named entity recognition*, skraćeno NER) je problem određivanja i klasifikacije onih fraza u tekstu koje se odnose na osobe, lokacije, organizacije i druge istaknute pojmove. Zajednički naziv za takve fraze je **imenovani entitet** (eng. *named entity*). Drugim riječima, imenovani entitet definiramo kao objekt stvarnog svijeta koji ima vlastito ime. Pogledajmo primjer:

> *The girl called* $\underbrace{\textit{Red Riding Hood}}_{\textit{Person}}$ *walked through the forest called* $\underbrace{\textit{Grünewald}}_{\textit{Location}}$.

Uočimo da se NER problem sastoji od dva problema — prepoznavanja samih entiteta (*Red Riding Hood*) i njihovog svrstavanja u unaprijed poznate kategorije (*Person*). Stoga se naivni pristup rješavanju problema sastoji od dva koraka. Prvi korak je označiti kao imenovani entitet sve riječi pisane velikim početnim slovom. Problem prvog koraka je u tome što prva riječ svake rečenice, također, započinje velikim slovom. S druge strane, često naziv entiteta sadrži riječi pisane malim početnim slovom, kao što je u slučaju *Hrvatske narodne banke*. Nadalje, arapski i kineski jezik uopće nemaju kapitalizaciju, dok se u njemačkom jeziku sve imenice pišu velikim početnim slovom, pa se ovakav pristup uopće ne može primijeniti.

Drugi korak naivnog pristupa bi se sastojao u definiranju skupa svih entiteta neke kategorije, npr. $location = \{Zagreb, Rijeka, Velebit, \ldots\}$, nakon čega bismo za pronađene entitete provjeravali nalaze li se u nekoj od kategorija. Osim što je nemoguće popisati sve moguće entitete (npr. sva ljudska imena), problem je i višeznačnost teksta — *Irska* može, ovisno o kontekstu, označavati državu ili otok.

Iz prethodno opisanih problema zaključujemo da je prepoznavanju entiteta potrebno pristupiti metodama strojnog učenja. U sljedećem poglavlju ćemo formalno opisati danas najčešće korištene metode rješavanja NER problema.

Za sam kraj ovog dijela opišimo vezu problema prepoznavanja imenovanih entiteta i karakterizacije likova u dječjim pričama. Svi dostupni sustavi, kao što je *Stanford NER*, prepoznaju barem kategorije *Person*, *Location*, *Organisation*. Kod problema karakterizacije dobrih i loših likova, u priči je dovoljno prepoznati samo jednu kategoriju entiteta — likove. Očito je da su, najčešće, sve (imenovane) osobe spomenute u priči, ujedno i likovi. Obrat ne vrijedi — nisu svi likovi (imenovane) osobe. Na primjer, uz Snjeguljicu se u istoimenoj bajki javljaju lovac i zla kraljica. Oni su očito likovi, ali kako nisu posebno imenovani, nijedan postojeći NER sustav ih neće prepoznati i klasificirati kao osobe (*Person*). Zato je potrebno istrenirati novi model koji će prepoznavati sve likove, neovisno o njihovom imenu.

1.4 Razrješavanje koreferencije

Problem **razrješavanja koreferencije** (eng. *coreference resolution*) čini prepoznavanje onih izraza u tekstu koji se odnose na isti izvanjezični entitet. Koreferentni izrazi mogu biti različitih oblika, kao što su vlastita imena, imeničke fraze ili zamjenice. Razlikujemo više vrsta koreferencije te ih dijelimo na zamjeničke i imeničke. Anafora, katafora i podijeljeni prethodnici su primjer zamjeničke koreferencije, dok koreferirajuće imeničke fraze pripadaju imeničkoj koreferenciji.

Anafora označava koreferenciju u kojoj se zamjenica (napisana plavom bojom) javlja nakon izraza na kojeg se referira (izraz je napisan crvenom bojom). Jasnije

je iz primjera:

> *Red Riding Hood was so happy that she wanted to dance through the forest.*

S druge strane, **katafora** označava koreferenciju u kojoj se zamjenica javlja prije izraza na kojeg se referira:

> *She always wore red cloak with a hood, so people called her Little Red Riding Hood.*

I anafora i katafora mogu uključivati **podijeljene prethodnike**. Radi se o obuhvaćanju više riječi jednom zamjenicom:

> *When Little Red Riding Hood and the wolf met, they spoke as old friends.*

Posljednja vrsta koreferencije koju ćemo navesti su **koreferirajuće imeničke fraze**. Ona označava referiranje na neku imeničku frazu drugom imeničkom frazom:

> *The girl in red riding hood was so happy on that sunny day. The child was singing merrily while walking through the forest.*

U zasebnom poglavlju opisat ćemo nekoliko metoda koje se koriste za razrješavanje koreferencije.

1.5 Analiza sentimenta

Također poznata i pod nazivom **rudarenje stavova** (eng. *opinion mining*), **analiza sentimenta** (eng. *sentiment analysis*) bavi se identificiranjem i ekstrakcijom subjektivnih informacija u tekstu. Najčešće se primjenjuje nad recenzijama, komentarima i kritikama objavljenim na različitim društvenim medijima, a u svrhu marketinga ili poboljšanja usluge.

Osnovni i najčešći zadatak analize sentimenta je određivanje polariteta danog teksta, odnosno, prepoznavanje stava izrečenog u tekstu kao pozitivnog, negativnog ili neutralnog. Osim toga, analiza sentimenta za cilj može imati i klasifikaciju teksta na objektivni ili subjektivni. Najkompleksniji oblik analize sentimenta se odnosi na više karakteristika objekta o kojemu se u tekstu govori. Tada je cilj raspoznati koje karakteristike su ocijenjene pozitivno, a koje negativno. Na primjer, korisnička recenzija nekog proizvoda može uključivati pohvale za dizajn, ali i negativne kritike o cijeni.

Polaritet teksta ili rečenice najčešće možemo prepoznati po prisutnosti nekih riječi. Na primjer, ako se u rečenici spominju negativne riječi, poput pridjeva *evil*, *selfish*, glagola *hate* ili imenice *junk*, sentiment je vjerojatno negativan. S druge strane, ako se koriste pozitivne riječi, kao što su pridjevi *great*, *nice*, glagoli *love*, *like* ili imenica *delight*, možemo zaključiti da je sentiment pozitivan.

This story is interesting and it has good characters and good ending.

Ipak, često nije dovoljno promatrati isključivo riječi u tekstu. Naime, u neutralnim rečenicama se mogu javiti i pozitivne riječi, kao u sljedećem slučaju:

Is this story interesting? Does it have good characters and good ending?

Nadalje, velik problem predstavlja oblik predikata, što može u potpunosti promijeniti značenje teksta. Jednak utjecaj ima i prisutnost nekih riječi ili izraza, poput *not*, *any*, *at all*. Na primjer, sljedeće rečenice su posve suprotnog polariteta, iako im je građa gotovo jednaka:

I heard this book was brilliant. In the end, I was disappointed.
I heard this book was brilliant. In the end, I wasn't disappointed at all.

U sljedećem primjeru oblik predikata određuje stav. U prvom slučaju radi se o negativnom stavu — knjiga je *trebala biti* odlična, ali očito nije. Druga rečenica je jasno pozitivnog stava:

This book was supposed to be brilliant.
This book is brilliant.

Još veći problem predstavlja sarkazam, odnosno, rečenica koja je po smislu pozitivna, ali ju treba shvatiti negativno. Često je i ljudima teško prepoznati sarkazam u tekstu. U sljedećem primjeru prvu rečenicu treba shvatiti sarkastično, odnosno, kao negativnu, dok druga ima doslovno, pozitivno značenje. Naglašene riječi su podcrtane.

I <u>can't wait</u> to see that old lady, she is <u>so</u> nice.
I can't wait to see that old lady, she is so nice.

Također je problematično prepoznati sentiment u rečenicama u kojima, naočigled, nema niti negativnih niti pozitivnih riječi. Na primjer, u sljedećem slučaju možemo se složiti da je vila učinila loše djelo:

Old fairy left the children to wander alone in endless forest.

Jedini način kako prepoznati sentiment takvog teksta je shvatiti njegovo značenje, odnosno, utvrditi bit tvrdnje koja se izriče. To je više problem razumijevanja prirodnog jezika, nego klasične analize sentimenta.

Neke od metoda koje se koriste u analizi sentimenta ćemo detaljnije opisati u zasebnom poglavlju.

Poglavlje 2

Modeli za prepoznavanje imenovanih entiteta

2.1 Generativni i diskriminativni modeli

Za početak, opisat ćemo dvije velike klase modela kojima pripadaju danas najčešće korišteni modeli za prepoznavanje imenovanih entiteta. To su generativni i diskriminativni modeli.

Neka je X skup ulaznih varijabli te Y skup izlaznih varijabli. U slučaju prepoznavanja imenovanih entiteta, X može uključivati podatke kao što su broj, rod i vrsta riječi, dok Y sadrži informaciju o kategorijama kojima pripada svaka riječ nekog izraza u tekstu. Varijable skupa X promatramo, dok varijable skupa Y želimo predvidjeti sa što većom točnošću.

Neka su dani vektor ulaznih varijabli $\mathbf{x} \in X$ i izlazni vektor $\mathbf{y} \in Y$. Prepoznati imenovane entitete u suštini znači odabrati niz riječi za koje je vjerojatnost $\mathbb{P}(\mathbf{x}, \mathbf{y})$ dovoljno visoka.

Prisjetimo se, za dvije varijable X i Y vrijedi Bayesovo pravilo:

$$\begin{aligned} \mathbb{P}(Y = y, X = x) &= \mathbb{P}(X = x) \cdot \mathbb{P}(Y = y | X = x) \\ &= \mathbb{P}(Y = y) \cdot \mathbb{P}(X = x | Y = y). \end{aligned} \tag{2.1}$$

Dakle, za modeliranje zajedničke distribucije varijabli X i Y je potrebno znati distribuciju $\mathbb{P}(X)$. Analogno, u slučaju vektora $\mathbf{x}$, radi se o distribuciji $\mathbb{P}(\mathbf{x})$. S obzirom da varijable vektora $\mathbf{x}$ mogu uključivati kompleksne međuzavisnosti, često nije moguće odrediti njegovu distribuciju. U slučaju kada je to moguće, dobiveni model može biti prekompleksan za izračunavanje. S druge strane, ignoriranje međuzavisnosti često rezultira lošim ili čak neupotrebljivim modelom.

Ovakvi modeli zajedničke distribucije se nazivaju **generativni modeli**. Naziv je intuitivan — oni izravno opisuju kako vrijednosti ulaznih varijabli (probabilistički) generiraju vrijednost izlaznih. Stoga se generativni modeli mogu koristiti i za generiranje novih podataka.

S druge strane, **diskriminativni modeli** su modeli uvjetne distribucije $\mathbb{P}(\mathbf{y}|\mathbf{x})$. Oni ne zahtijevaju modeliranje distribucije $\mathbb{P}(\mathbf{x})$. U praksi, $\mathbb{P}(\mathbf{x})$ često nije niti potrebna za rješavanje danog problema. U našem slučaju, nije bitno je li međusobno povezan rod imenice i njezino završno slovo, već samo koliko je vjerojatno da ta imenica pripada zadanoj klasi uz dani ulaz. Zato se često diskriminativnim modelima postižu puno bolji rezultati.

Zanimljiva je činjenica da postoji međusobna uparenost generativnih i diskriminativnih modela. To znači da su dva modela oblikovana nad istim prostorom hipoteza te da učenjem nad jednakim podacima daju jednaki rezultat. Iz toga slijedi da se generativni model može preinačiti u diskriminativni i obratno. O tome ćemo detaljnije reći nakon što damo neke primjere modela.

2.2 Grafički modeli

Grafički modeli su statistički modeli koji se uobičajeno koriste za prikaz ovisnosti među varijablama. Jedan model čini skup vjerojatnosnih distribucija koje se faktoriziraju prema definiranom grafu. Cilj grafičkih modela je prikazati distribuciju nad velikim brojem slučajnih varijabli kao produkt lokalnih funkcija manjeg broja varijabli.

Neka su $X = \{x_1, \ldots, x_n\}$ skup ulaznih varijabli koje promatramo i $Y = \{y_1, \ldots, y_m\}$ skup izlaznih varijabli koje želimo predvidjeti. Nadalje, neka je $V = X \cup Y$. Svaki $v \in V$ poprima vrijednosti iz nekog skupa $\mathcal{V}$. Neka su dani ulazni vektor $\mathbf{x} \in X$ i izlazni vektor $\mathbf{y} \in Y$. Za dani podskup $A \subseteq V$, s $\mathbf{x}_A$ označavamo vektor ulaznih varijabli takvih da je $x_i \in A$. Analogno, s $\mathbf{y}_A$ označavamo vektor izlaznih varijabli za koje je $y_j \in A$.

Definicija 2.2.1. *Neka je $\mathcal{F}$ familija nekih podskupova $A \subseteq V$.* ***Neusmjereni grafički model*** *je skup svih distribucija koje se mogu zapisati u obliku*

$$\mathbb{P}(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{A \in \mathcal{F}} \Psi_A(\mathbf{x}_A, \mathbf{y}_A), \tag{2.2}$$

za neki izabrani skup funkcija $F = \{\Psi_A\}$, pri čemu su $\Psi_A : \mathcal{V}^{|A|} \to \mathbb{R}^+$.

Elemente skupa F nazivamo ***faktorima*** *ili* ***lokalnim funkcijama****. Konstantu Z nazivamo* ***normalizacijski faktor*** *i definiramo kao*

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_{A} \Psi_A(\mathbf{x}_A, \mathbf{y}_A).$$

Normalizacijski faktor osigurava da distribucija ima sumu jednaku jedan, odnosno, da vrijedi

$$\sum_{\mathbf{x},\mathbf{y}} \mathbb{P}(\mathbf{x}, \mathbf{y}) = 1.$$

S obzirom da se u gornjem izrazu sumira po svim mogućim vrijednostima vektora $\mathbf{x}$ i $\mathbf{y}$, u praksi je, najčešće, faktor Z nemoguće izračunati. Stoga se koriste različite metode za njegovu aproksimaciju.

Faktorizaciju (2.2) grafički prikazujemo faktor grafom.

Definicija 2.2.2. ***Faktor graf*** *za distribuciju vjerojatnosti* (2.2) *je bipartitni graf $G = (V, F, E)$, pri čemu je čvor-varijabla $v \in V$ povezan s čvorom-faktorom $\Psi_A \in F$ ako i samo ako je v argument funkcije Ψ_A.*

Uz faktor grafove, postoje i druge vrste grafičkih modela. Najjednostavniji od njih su usmjereni grafički modeli. Oni se prikazuju usmjerenim acikličkim grafom $G = (V, E)$.

Definicija 2.2.3. ***Usmjereni grafički model*** *ili* ***Bayesova mreža*** *se definira kao zajednička distribucija vjerojatnosti koja se faktorizira kao*

$$\mathbb{P}(\mathbf{x}, \mathbf{y}) = \prod_{v \in V} \mathbb{P}(v|\pi(v)),$$

gdje su $\pi(v)$ roditelji čvora v u grafu G.

Usmjereni grafički modeli se mogu koristiti za prikazivanje generativnih modela. Konkretno, generativni modeli se prikazuju usmjerenim modelima u kojima izlazne varijable uvijek prethode ulaznima. Drugim riječima, nijedna ulazna varijabla $x \in X$ ne može biti roditelj izlaza $y \in Y$.

Prije nego damo primjere grafičkih modela, opisat ćemo dva jednostavna modela za klasifikaciju. Nakon toga ćemo pokazati kako se oni proširuju do modela koji se, između ostalog, koriste za rješavanje problema imenovanih entiteta.

2.3 Klasifikacija

Klasifikacija (eng. *classification*) je problem predviđanja klase varijable y na temelju ulaznog vektora značajki $\mathbf{x} = (x_1, \ldots, x_n)$. Na primjer, u slučaju određivanja spola nekog lika y u priči, (uobičajene) klase su *muško, žensko*. Neke od značajki mogu biti *završno slovo imena, završni slog imena, zamjenica koja se koristi pri referiranju na lika* i slično.

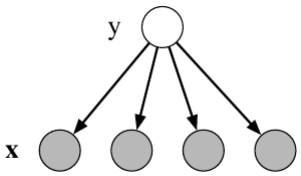
Dva poznata klasifikatora, odnosno, modela koji rješavanju ovaj problem, su naivni Bayesov klasifikator i logistička regresija.

Naivni Bayesov klasifikator (eng. *naive Bayes classifier*) koristi pretpostavku da su, ukoliko je poznata klasa, sve značajke međusobno nezavisne. Očito je da u stvarnom svijetu ne vrijedi nezavisnost. Unatoč tome, u praksi naivni Bayesov klasifikator pokazuje dosta dobre rezultate. Također, iz njegove jednostavnosti slijedi velika brzina, lakoća implementacije i niska potreba za računalnim resursima. Iz navedenih razloga se naivni Bayesov klasifikator često koristi za rješavanje mnogih problema, poput detekcije spam poruka, analize sentimenta ili automatske medicinske dijagnoze.

Definicija 2.3.1. ***Naivni Bayesov klasifikator*** *je zajednička distribucija vjerojatnosti koja se faktorizira kao*

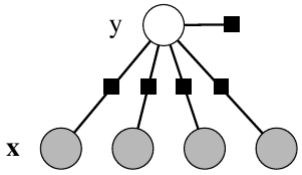
$$\mathbb{P}(y, \mathbf{x}) = \mathbb{P}(y) \prod_{i=1}^{n} \mathbb{P}(x_i|y). \tag{2.3}$$

Model je moguće prikazati usmjerenim grafičkim modelom ili faktor grafom.



Slika 2.1: Naivni Bayesov klasifikator kao usmjereni grafički model

Ukoliko model prikazujemo faktor grafom, faktori su dani s $\Psi(y) = \mathbb{P}(y)$ te $\Psi_i(y, x_i) = \mathbb{P}(x_i|y)$, za svaki x_i. Dogovorno, krugovi označavaju čvorove-varijable, a kvadrati čvorove-faktore.



Slika 2.2: Naivni Bayesov klasifikator kao faktor graf

Drugi često korišteni klasifikator je **logistička regresija** (eng. *logistic regression*). Klasifikator polazi od pretpostavke da je logaritam vjerojatnosti svake klase, $\log(\mathbb{P}(y|\mathbf{x}))$, linearna funkcija od $\mathbf{x}$ (uz normalizacijsku konstantu).

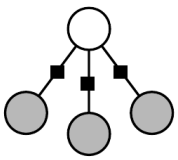
Definicija 2.3.2. ***Logistička regresija*** *je uvjetna distribucija vjerojatnosti koja se faktorizira kao*

$$\mathbb{P}(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{i=1}^{n} \theta_i f_i(y, \mathbf{x})\right),$$

pri čemu su $\theta_i \in \mathbb{R}$ parametri, f_i funkcije koje nazivamo ***funkcije značajki****, dok je $Z(\mathbf{x})$ normalizacijska konstanta definirana s*

$$Z(\mathbf{x}) = \sum_{\tilde{y},\tilde{\mathbf{x}}} \exp\left(\sum_{i=1}^{n} \theta_i f_i(\tilde{y}, \tilde{\mathbf{x}})\right).$$

Logistička regresija se prikazuje faktor grafom:



Slika 2.3: Prikaz logističke regresije kroz faktor graf

Naivni Bayesov klasifikator i logistička regresija čine generativno–diskriminativni par. Naime, ukoliko u logističkoj regresiji maksimiziramo zajedničku vjerojatnost

$$\mathbb{P}(y, \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{i=1}^{n} \theta_i f_i(y, \mathbf{x})\right),$$

dobivamo onaj klasifikator kojega generira naivni Bayes.

Obratno, upotrebom Bayesovog pravila (2.1) nad Bayesovim klasifikatorom (2.3) dobivamo diskriminativni model

$$\mathbb{P}(y|\mathbf{x}) = \frac{\mathbb{P}(y,\mathbf{x})}{\mathbb{P}(\mathbf{x})} = \frac{\mathbb{P}(y)\prod_{i=1}^{n}\mathbb{P}(x_i|y)}{\mathbb{P}(x_1,\ldots,x_n)}.$$

Ovakav klasifikator generira i logistička regresija.

2.4 Skriveni Markovljevi modeli

Skriveni Markovljevi modeli (eng. *hidden Markov models*, kraće HMM) čine svojevrsnu nadogradnju naivnog Bayesovog klasifikatora na model koji, umjesto za jednu varijablu, odlučuje za vektore varijabli. Objasnimo kako je došlo do te nadogradnje.

Neka je dan neki niz riječi (tekst) u kojem želimo prepoznati imenovane entitete nekih kategorija, na primjer, *Osoba*, *Lokacija*, *Organizacija*. Uobičajeno se uvodi oznaka *Ostalo*, koja označava da riječ nije dio imenovanog entiteta. Jedan od pristupa rješavanju NER problema je nezavisno klasificiranje svake riječi niza nekom od navedenih oznaka. Problem koji se pri tome javlja je upravo pretpostavljena nezavisnost. S obzirom da imenovani entiteti često sadrže više riječi, očito imamo zavisnost među tim riječima. Na primjer, *New York* označava lokaciju, dok je *New York Times* organizacija.

Skriveni Markovljevi modeli djelomično relaksiraju pretpostavku o neovisnosti. HMM modelira ulazni niz $\mathbf{x} = \{x_1,\ldots,x_T\}$ tako da pretpostavlja postojanje niza stanja $\mathbf{y} = \{y_1,\ldots,y_T\}$ koja odgovaraju nizu $\mathbf{x}$. U našem slučaju, x_t označava riječ na poziciji t, dok je y_t oznaka kategorije pridjeljene riječi x_t. Pri modeliranju zajedničke distribucije $\mathbb{P}(\mathbf{y},\mathbf{x})$ koriste se dvije pretpostavke o nezavisnosti.

Prvo pretpostavljamo da svaka oznaka y_t ovisi jedino o svom neposrednom prethodniku y_{t-1}, odnosno, da ne ovisi o ostalim prethodnicima $y_{t-2},\ldots,y_1$. Formalni zapis ove pretpostavke glasi:

$$\mathbb{P}(y_t = y'_t \mid y_{t-1} = y'_{t-1},\ldots,y_1 = y'_1) = \mathbb{P}(y_t = y'_t \mid y_{t-1} = y'_{t-1}),$$

pri čemu je y'_j vrijednost oznake y_j, odnosno, neka od vrijednosti *Osoba*, *Lokacija*, *Organizacija*, *Ostalo*. Ovo svojstvo se naziva **Markovljevo svojstvo**.

Druga pretpostavka je ta da svaka varijabla x_t ovisi isključivo o trenutnoj kategoriji y_t. Odavde potječe naziv *skriveni* Markovljevi modeli. Ulaz nije izravno vidljiv, već je vidljiv samo rezultat koji ovisi o tom ulazu.

Koristimo tri distribucije vjerojatnosti da bismo definirali skriveni Markovljev model. Prva je distribucija početnih oznaka $\mathbb{P}(y_0)$, odnosno, oznaka koje nemaju

prethodnika. Sljedeća je distribucija prijelaza $\mathbb{P}(y_t|y_{t-1})$. Osim što je t indeks pojedine riječi u nizu, on označava i indeks pojedinih prijelaza za zadani izlazni vektor. Posljednja distribucija, $\mathbb{P}(x_t|y_t)$, opisuje vezu ulazne i izlazne varijable.

Ove distribucije će biti jasnije iz primjera. Neka je dan ulazno-izlazni par vektora $\mathbf{x}$ i $\mathbf{y}$, odnosno, rečenice i ispravnih oznaka za tu rečenicu:

$$\mathbf{x} = \textit{Red}, \textit{Riding}, \textit{Hood}, \textit{walked}, \textit{through}, \textit{Grünewald},$$
$$\mathbf{y} = \textit{Osoba}, \textit{Osoba}, \textit{Osoba}, \textit{Ostalo}, \textit{Ostalo}, \textit{Lokacija}.$$

Distribucija početnih oznaka opisuje vjerojatnost da prva riječ u rečenici pripada klasi *Osoba*. Distribucija prijelaza opisuje vjerojatnost da, na primjer, nakon oznake *Lokacija* slijedi oznaka *Osoba* (niska vjerojatnost), odnosno, *Lokacija* ili *Ostalo* (visoka vjerojatnost). Posljednja navedena distribucija opisuje vjerojatnost da je ulazna riječ bila *Grünewald* ako je izlaz *Lokacija*.

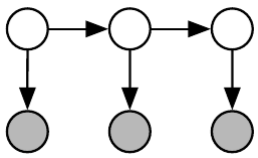
Sada možemo dati formalnu definiciju skrivenih Markovljevih modela.

Definicija 2.4.1. *Neka su* $\mathbf{x} = \{x_1, \ldots, x_T\}$ *i* $\mathbf{y} = \{y_1, \ldots, y_T\}$ *slučajni vektori.* ***Skriveni Markovljev model*** *je zajednička distribucija vjerojatnosti oblika*

$$\mathbb{P}(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{T} \mathbb{P}(y_t|y_{t-1})\mathbb{P}(x_t|y_t),$$

pri čemu smo za $\mathbb{P}(y_1)$ *koristili zapis* $\mathbb{P}(y_1|y_0)$.

Skriveni Markovljevi modeli se mogu prikazati usmjerenim grafom, kao na sljedećoj slici:



Slika 2.4: Grafički model skrivenih Markovljevih modela

Skriveni Markovljevi modeli se, osim za prepoznavanje imenovanih entiteta, uspješno koriste i za prepoznavanje govora, rukom pisanih znakova, za određivanje gramatičkog ustrojstva rečenice, te u bioinformatici.

Nedostatak skrivenih Markovljevih modela je to što promatraju samo jednu značajku ulaznih podataka. U slučaju prepoznavanja imenovanih entiteta to je identitet riječi. S obzirom da se većina osobnih imena neće pojaviti u skupu podataka za učenje, ne možemo se oslanjati samo na taj podatak. Odnosno, ukoliko

učimo iz novinskih članaka, malo je vjerojatno da će istrenirani model moći u bajci prepoznati Alladina.

2.5 Uvjetna slučajna polja

Uz skrivene Markovljeve modele, najčešće korištena metoda za prepoznavanje imenovanih entiteta su **uvjetna slučajna polja** (eng. *conditional random fields*, skraćeno CRF). Prvo ćemo opisati najjednostavniju vrstu uvjetnih slučajnih polja, **linearno ulančana uvjetna slučajna polja** (eng. *linear-chain conditional random fields*). Skriveni Markovljevi modeli i linearno ulančana uvjetna slučajna polja čine generativno–diskriminativni par. Opišimo njihovo generiranje iz skrivenih Markovljevih modela.

Za varijable y, y' vektora $\mathbf{y}$ definiramo $\theta_{ij} = \log \mathbb{P}(y' = i \mid y = j)$. Nadalje, uvodimo funkcije značajki oblika $f_k(y_i, y_{i-1}, x)$. Definiramo:

$$f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{y'=j\}},$$
$$f_{io}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{x=o\}}.$$

U modelu neće sudjelovati sve ovako dobivene funkcije značajki, već po jedna za svaki prijelaz $(t, t-1)$ te za svaki izlazno-ulazni par (t, o), pri čemu je ulaz riječ (opservacija), dok je izlaz stanje (kategorija). S $k \in \{1, \ldots, K\}$ indeksiramo sve takve funkcije značajki.

Konačno, skriveni Markovljevi model se može zapisati u obliku

$$\mathbb{P}(\mathbf{y}, \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right). \tag{2.4}$$

Uvjetna distribucija vjerojatnosti se dobije kao

$$\mathbb{P}(\mathbf{y}|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{y}, \mathbf{x})}{\mathbb{P}(\mathbf{x})} = \frac{\mathbb{P}(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}'} \mathbb{P}(\mathbf{y}', \mathbf{x})} = \frac{\prod_{t=1}^{T} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right)}{\sum_{\mathbf{y}'} \prod_{t=1}^{T} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t) \right)},$$

pri čemu $\sum_{\mathbf{y}'}$ označava sumiranje po svim mogućim vrijednostima vektora $\mathbf{y}$.

Gornji izraz definira linearno ulančana uvjetna slučajna polja koja koriste samo jednu ulaznu slučajnu varijablu. Ukoliko koristimo proizvoljne funkcije značajki, dobivamo općenita linearno ulančana uvjetna slučajna polja.

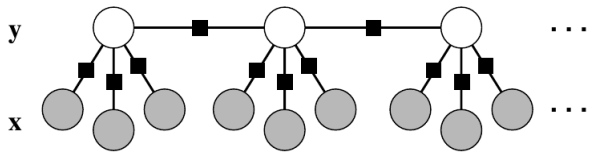
Definicija 2.5.1. *Neka su* $\mathbf{x} = \{x_1, \ldots, x_T\}$ *i* $\mathbf{y} = \{y_1, \ldots, y_T\}$ *slučajni vektori. Nadalje, neka je* $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_T\}$ *realan vektor parametara te neka su* $f_k(y, y', \mathbf{x}_t)$, *za* $k \in \{1, \ldots, K\}$, *realne funkcije koje nazivamo funkcije značajki.* ***Linearno ulančano uvjetno slučajno polje*** *je distribucija uvjetne vjerojatnosti oblika*

$$\mathbb{P}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right), \tag{2.5}$$

pri čemu je $Z(\mathbf{x})$ *normalizacijska konstanta definirana s*

$$Z(\mathbf{x}) = \sum_{\tilde{\mathbf{y}}} \prod_{t=1}^{T} \exp\left(\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right).$$

Ovi modeli se uobičajeno prikazuju linearno ulančanim faktor grafovima. Odatle, očito, potječe njihov naziv.



Slika 2.5: Grafički model linearno ulančanog uvjetnog slučajnog polja

S druge strane, distribucije nad proizvoljnim faktor grafovima, koje smo definirali u Definiciji 2.2.2, čine opća uvjetna slučajna polja.

Definicija 2.5.2. *Neka je* $G = (V, F, E)$ *faktor graf. Distribucija* $\mathbb{P}(\mathbf{y}|\mathbf{x})$ *je* ***uvjetno slučajno polje*** *ako se faktorizira prema faktor grafu* G *za svaki* $\mathbf{x}$.

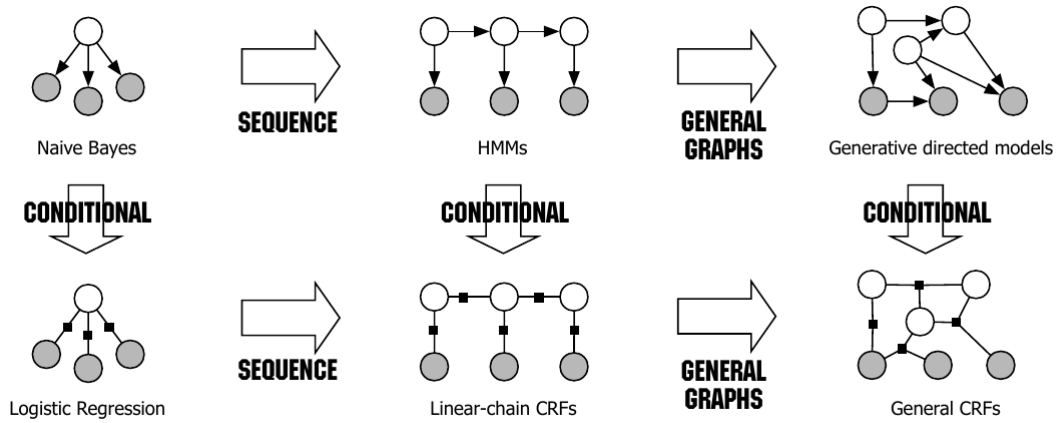
Drugim riječima, za skup faktora $F = \{\Psi_A\}$ iz G, definiranih s

$$\Psi_A(\mathbf{y}_A, \mathbf{x}_A) = \exp\left\{ \sum_k \theta_{Ak} f_{Ak}(\mathbf{y}_A, \mathbf{x}_A) \right\},$$

se uvjetno slučajno polje zapisuje kao

$$\mathbb{P}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\Psi_A \in G} \exp\left(\sum_{k=1}^{|A|} \theta_{Ak} f_{Ak}(\mathbf{y}_A, \mathbf{x}_A) \right).$$

U sljedećem dijagramu je jasan međusobni odnos naivnog Bayesovog klasifikatora, logističke regresije, skrivenih Markovljevih modela, te linearno ulančanih i općenitih uvjetnih slučajnih polja:



Slika 2.6: Dijagram veza dosad navedenih modela

S obzirom da uvjetna slučajna polja u obzir uzimaju "širu sliku" nego skriveni Markovljevi modeli, imaju i šire područje primjene. Uz obradu prirodnog jezika, uvjetna slučajna polja uspješno rješavaju mnoge probleme računalnog vida, kao što je prepoznavanje objekata na slikama. Također, ovim modelom se postižu daleko najbolji rezultati u prepoznavanju imenovanih entiteta. *Stanford NER*, jedan od najuspješnijih sustava, se temelji upravo na uvjetnim slučajnim poljima. Stanfordov sustav se sastoji od tri modela za engleski jezik, po dva za njemački i kineski, te jednog za španjolski jezik. Modeli za engleski jezik, ovisno o tome nad kojim skupom podataka su trenirani, prepoznaju:

- tri klase: *Location*, *Person*, *Organization*,
- četiri klase: *Location*, *Person*, *Organization*, *Misc*,
- sedam klasa: *Location*, *Person*, *Organization*, *Money*, *Percent*, *Date*, *Time*.

2.6 Učenje i korištenje modela

Za kraj ćemo opisati postupak određivanja, odnosno, učenja modela koji će se koristiti za prepoznavanje imenovanih entiteta.

Neka je dan skup podataka za učenje $D = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\}$. Pretpostavljamo da su elementi skupa D nezavisni i jednako distribuirani. Za svaki i

je $\mathbf{x}^{(i)} = \{x_1^{(i)}, \ldots, x_T^{(i)}\}$ T-dimenzionalni ulazni vektor, odnosno, niz riječi. S druge strane, $\mathbf{y}^{(i)} = \{y_1^{(i)}, \ldots, y_T^{(i)}\}$ je T-dimenzionalni vektor željenih izlaza. Željeni izlazi su, u ovom slučaju, točne oznake klasa niza riječi $\mathbf{x}^{(i)}$. Cilj je odrediti skriveni Markovljev model (2.4) ili linearno ulančani CRF (2.5), koji će nad novim podacima što manje griješiti, odnosno, što točnije određivati klase.

Učenje modela se sastoji u određivanju nepoznatih parametara ciljne funkcije za koje se postiže (po mogućnosti) globalni ekstrem. U ovom slučaju, nepoznati parametri su elementi vektora $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_K\}$.

U slučaju skrivenih Markovljevih modela, za $\mathbb{P}(\mathbf{y}^{(i)}, \mathbf{x}^{(i)})$ kao u (2.4), promatramo jednadžbu

$$l(\boldsymbol{\theta}) = l(\theta_1, \ldots, \theta_K) = \sum_{i=1}^{N} \log \mathbb{P}(\mathbf{y}^{(i)}, \mathbf{x}^{(i)}).$$

Analogno, u slučaju linearno ulančanih uvjetnih slučajnih polja, za $\mathbb{P}(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})$ kao u (2.5), imamo

$$l(\boldsymbol{\theta}) = l(\theta_1, \ldots, \theta_K) = \sum_{i=1}^{N} \log \mathbb{P}(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}).$$

Postoji više metoda za određivanje maksimuma funkcije l — egzaktno, metodom gradijentog penjanja, ili nekom drugom heurističkom metodom. Metoda koja će se koristiti ovisi o veličini skupa za učenje i broju nepoznatih parametara, odnosno, složenosti funkcija značajki. U praksi se koriste veoma veliki skupovi za učenje, čija veličina se mjeri u stotinama tisuća riječi, te prilično složene funkcije značajki. Stoga se treniranje modela vrši na snažnim računalima uz brojne optimizacije. Unatoč tome, postupak može trajati satima.

Nakon određivanja parametara, možemo koristiti model za određivanje klase niza riječi, odnosno, prepoznavanje imenovanih entiteta u tekstu. Za zadani niz riječi x, računa se svaka od vjerojatnosti $\mathbb{P}(\mathbf{y}^{(k)}|\mathbf{x})$, pri čemu $\mathbf{y}^{(k)}$ predstavlja jedan mogući raspored dodijeljenih klasa, na primjer, $y_1 = \textit{Osoba}$, $y_2 = \textit{Osoba}$, $y_3 = \textit{Ostalo}$. I u ovom koraku problem može biti visoka složenost postupka, odnosno, dugotrajno izračunavanje. Naime, za niz riječi proizvoljne duljine n i konstantan broj klasa m, potrebno je izračunati m^n različitih vjerojatnosti, odnosno, postupak je eksponencijalne složenosti. Stoga su i za ovaj korak razvijene brojne optimizacije.

Poglavlje 3

Modeli za razrješavanje koreferencije

3.1 Model parova

U ovom poglavlju ćemo opisati nekoliko modela koji se koriste za razrješavanje koreferencije u tekstu, te navesti njihove prednosti i nedostatke. Započet ćemo s najjednostavnijim modelom, a to je **model parova** (eng. *pairwise model*). Ovaj model razrješavanju koreferencije pristupa kao problemu binarne klasifikacije parova izraza. Definirajmo takav model.

Definicija 3.1.1. *Neka je za par izraza (slučajnih varijabli) $x_{ij} = \{x_i, x_j\}$ varijabla y_{ij} definirana s*

$$y_{ij} = 1 \iff x_i \text{ i } x_j \text{ su koreferentni.}$$

Nadalje, neka je $F = \{f_k(x_{ij}, y_{ij})\}$ skup funkcija značajki nad x_{ij}, te neka su $\theta_k \in \mathbb{R}$ parametri. ***Model parova*** *se definira kao distribucija vjerojatnosti oblika*

$$\mathbb{P}(y_{ij}|x_{ij}) = \frac{1}{Z_{x_{ij}}} \exp\left(\sum_k \theta_k f_k(x_{ij}, y_{ij})\right), \tag{3.1}$$

pri čemu je $Z_{x_{ij}}$ faktor normalizacije.

Na primjer, funkcija $f_k(x_{ij}, y_{ij})$ može označavati jesu li x_i i x_j istog roda, dok funkcija $f_l(x_{ij}, y_{ij})$ označava jesu li jednake brojnosti.

Ukoliko par $\{x_i, x_j\}$ smatramo uređenim parom, pri čemu izraz x_i u tekstu prethodi izrazu x_j, te se x_j odnosi na x_i, dobivamo asimetričnu interpretaciju problema razrješavanja koreferencije. Takva interpretacija je slična razrješavanju anafore. Jedina razlika je u tome što, kod anafore, izraz x_j mora biti zamjenica

koja se referira na izraz x_i. U ovom slučaju, x_i i x_j mogu biti bilo kakvi izrazi. Pogledajmo primjer:

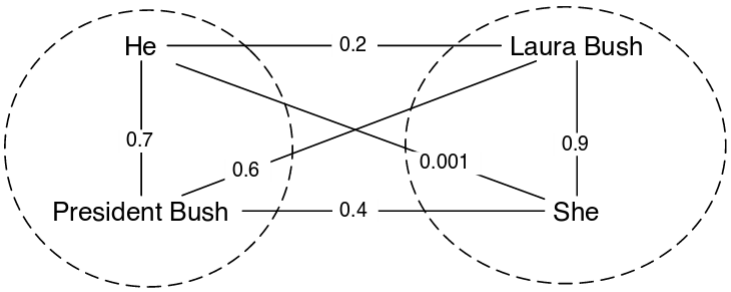
$$\underbrace{\textit{Red Riding Hood}}_{x_i} \textit{ was so happy. } \underbrace{\textit{Young girl}}_{x_j} \textit{ wanted to dance in the forest.}$$

Simetričnu interpretaciju dobivamo kada par $\{x_i, x_j\}$ interpretiramo kao neuređeni par, pri čemu su x_i i x_j u koreferenciji, ali nije definiran smjer koreferencije.

Prednost modela parova je što je $\mathbb{P}(y_{ij}|x_{ij})$ moguće izračunati isključivo iz danog teksta, odnosno, pomoću parova $\{x_i, x_j\}$. S druge strane, model implicira pretpostavku o snažnoj neovisnosti među parovima. Rezultat toga je nemogućnost reprezentacije čitavog skupa izraza koji se odnose na isti entitet. Drugim riječima, dobivamo isključivo skupove oblika $\{izraz_i, izraz_j\}$, a ne možemo dobiti skup izraza $\{izraz_1, \ldots, izraz_n\}$ koji se referiraju na isti entitet.

Slično kao i u prethodnom poglavlju, modele za razrješavanje koreferencije možemo prikazati grafički i to na više načina.

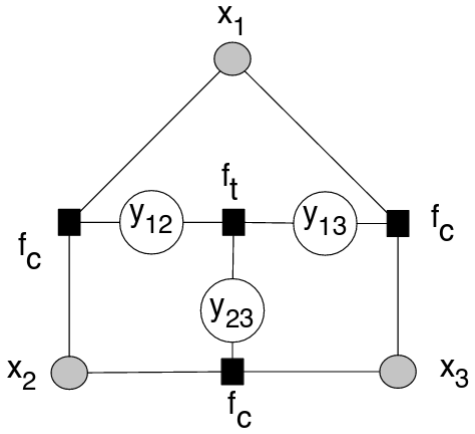
Prvi način prikaza je prilično intuitivan. U slučaju modela parova, kreiramo graf čiji vrhovi predstavljaju imeničke fraze, dok težine bridova odgovaraju vrijednostima $\mathbb{P}(y_{ij}|x_{ij})$. Sada se problem razrješavanja koreferencije svodi na particioniranje grafa u klastere s maksimalnim težinama bridova unutar pojedinog klastera te minimalnim težinama među klasterima. Jasnije je iz sljedeće slike:



Slika 3.1: Primjer klasteriranja grafa

Drugi način koristi faktor grafove koje smo definirali u prethodnom poglavlju. Model za razrješavanje koreferencije prikazuje se faktor grafom. U takvom prikazu je jednostavno model parova proširiti tranzitivnošću izraza, čime dobivamo mogućnost reprezentiranja skupova izraza.

Tranzitivnost izraza znači sljedeće: ako su $izraz_1$ i $izraz_2$, te $izraz_2$ i $izraz_3$ koreferentni, onda su $izraz_1$ i $izraz_3$, također, koreferentni. Na sljedećoj slici prikazan je primjer faktor grafa za model parova koji uključuje tranzitivnost.



Slika 3.2: Grafički prikaz modela parova

Faktori f_c modeliraju koreferenciju dvaju izraza, dok faktor f_t modelira tranzitivnost među izrazima. Sada možemo definirati grafički model parova za razrješavanje koreferencije.

Definicija 3.1.2. ***Grafički model parova*** *za razrješavanje koreferencije čini distribucija uvjetne vjerojatnosti koja se faktorizira kao*

$$\mathbb{P}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \prod_{y_{ij} \in \mathbf{y}} f_c(y_{ij}, x_{ij}) \prod_{y_{ij}, y_{jk} \in \mathbf{y}} f_t(y_{ij}, y_{jk}, y_{ik}, x_{ij}, x_{jk}, x_{ik}), \tag{3.2}$$

pri čemu je $Z_{\mathbf{x}}$ normalizacijski faktor, faktor f_c je definiran s

$$f_c(y_{ij}, x_{ij}) = \exp\left(\sum_k \theta_k f_k(y_{ij}, x_{ij})\right),$$

dok je faktor f_t definiran s

$$f_t(\cdot) = \begin{cases} -\infty, & \textit{ako je tranzitivnost zadovoljena,} \\ 1, & \textit{inače.} \end{cases}$$

Model parova se određuje analogno postupku opisanom na kraju prethodnog poglavlja. Neka je zadan skup nezavisnih jednako distribuiranih podataka $D =$

$\{(x_{ij}, y_{ij})\}$, pri čemu par izraza $\{x_i, x_j\}$ može, ali i ne mora biti koreferentan. Cilj je nad zadanim podacima maksimizirati funkciju

$$l(\boldsymbol{\theta}) = l(\theta_1, \ldots, \theta_K) = \sum_{ij} \log \mathbb{P}(y_{ij}|x_{ij}),$$

pri čemu je $\mathbb{P}(y_{ij}|x_{ij})$ kao u (3.1) ili (3.2).

Model se koristi na sljedeći način. Neka je zadan parametar $\delta \in [0, 1]$ kojeg nazivamo prag vrijednosti. Izračunati model, za svaki par $\{x_i, x_j\}$ takav da je $\mathbb{P}(y_{ij}|x_{ij}) \geq \delta$, zaključuje da je međusobno koreferentan. Uobičajeni izbor za prag δ je 0.5. Što je prag veći, to će model biti precizniji. **Preciznost** (eng. *precision*) se definira kao udio točno klasificiranih primjera u skupu pozitivno klasificiranih primjera. U ovom slučaju, visoka preciznost znači da su gotovo svi izrazi koje je model označio koreferentnima uistinu koreferentni. Uz preciznost, bitno svojstvo klasifikatora je i odziv. **Odziv** (eng. *recall*) se definira kao udio točno klasificiranih primjera u skupu svih pozitivnih primjera. U ovom slučaju, brojni koreferentni izrazi neće biti označeni pa će odziv modela biti nizak.

Obratno, za manji prag δ, model će veći broj parova označiti kao koreferentne te će vjerojatno označiti većinu doista koreferentnih izraza. Dakle, model će imati visok odziv. S druge strane, vjerojatno će i dosta parova krivo označiti, pa će preciznost biti niska.

Dakle, za uspješno razrješavanje koreferencije potrebno je da model ima što više i preciznosti i odziva. Te dvije mjere se objedinjuju u jednu mjeru kvalitete koju nazivamo F-mjera.

Definicija 3.1.3. ***F_β-mjera****, za $\beta \in \mathbb{R}$, se definira kao*

$$F_\beta = (1 + \beta^2) \frac{\mathit{preciznost} \cdot \mathit{odziv}}{\beta^2 \cdot \mathit{preciznost} + \mathit{odziv}}.$$

Posebno, ***F_1-mjera*** *se definira kao*

$$F_1 = 2 \cdot \frac{\mathit{preciznost} \cdot \mathit{odziv}}{\mathit{preciznost} + \mathit{odziv}}.$$

F-mjera poprima vrijednosti između 0 i 1, pri čemu je model kvalitetniji što je njegova F-mjera viša.

Parametar β određuje kojem svojstvu pridajemo veću važnost, preciznosti ili odzivu. Na primjer, za $\beta = 0.5$ dvostruko veću važnost pridajemo preciznosti nego odzivu, dok je za $\beta = 2$ obratno. Najčešće se koristi F_1-mjera koja za obje stavke pridjeljuje jednaku važnost.

3.2 Model logike prvog reda

Model logike prvog reda (eng. *First-order logic model*) za razrješavanje koreferencije se razvio s ciljem rješavanja problema reprezentacije skupa izraza. Model je analogan prethodno opisanom modelu.

Definicija 3.2.1. *Neka je* $\mathbf{x}^j = \{x_i\}$ *skup imeničkih fraza i neka je varijabla* y_j *definirana s*

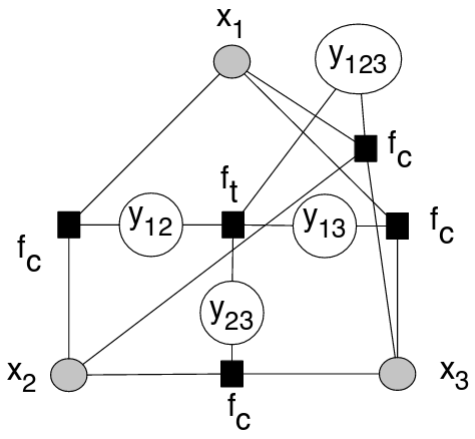
$$y_j = 1 \iff \textit{sve fraze } x_i \in \mathbf{x}^j \textit{ su međusobno koreferentne.}$$

Model logike prvog reda *definiramo kao distribuciju vjerojatnosti oblika*

$$\mathbb{P}(y_j|\mathbf{x}^j) = \frac{1}{Z_{\mathbf{x}^j}} \exp\left(\sum_k \theta_k f_k(\mathbf{x}^j, y)\right), \tag{3.3}$$

pri čemu su f_k *funkcije značajke nad čitavim skupom* $\mathbf{x}^j$*, a* $\theta_k \in \mathbb{R}$ *su pripadni parametri.*

I ovaj model možemo prikazati pomoću faktor grafova, kao na sljedećoj slici:



Slika 3.3: Grafički prikaz modela logike prvog reda

Faktori f_c i f_t imaju jednaku ulogu kao i u modelu parova. Tranzitivnost je nešto kompliciranija: ako je $y_j = 1$, za svaki podskup $\mathbf{x}^k \subseteq \mathbf{x}^j$ mora vrijediti $y_k = 1$. Nadalje, uvodi se dodatni čvor y_I, pri čemu je I neki skup indeksa, koji označava jesu li svi izrazi $\mathbf{x}^I$ međusobno koreferentni. Uočimo da broj čvorova y_I raste eksponencijalno u odnosu na broj ulaznih varijabli x.

Definicija 3.2.2. ***Grafički model logike prvog reda*** *za razrješavanje koreferencije čini distribucija uvjetne vjerojatnosti koja se faktorizira kao*

$$\mathbb{P}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \prod_{y_j \in \mathbf{y}} f_c(y_j, \mathbf{x}^j) \prod_{y_j \in \mathbf{y}} f_t(y_j, \mathbf{x}^j), \tag{3.4}$$

pri čemu je $Z_{\mathbf{x}}$ normalizacijski faktor, faktor f_c je definiran s

$$f_c(y_j, \mathbf{x}^j) = \exp\left(\sum_k \theta_k f_k(y_j, \mathbf{x}^j)\right),$$

dok je faktor f_t definiran s

$$f_t(\cdot) = \begin{cases} -\infty, & \textit{ako je tranzitivnost zadovoljena,} \\ 1, & \textit{inače.} \end{cases}$$

Posebnog ovog modela je u tome što, ako je potrebno, skup podataka za treniranje možemo jednostavno proširiti. Pozitivne (međusobno koreferentne) skupove $\mathbf{x}^j$ generiramo slučajnim, uniformnim uzorkovanjem nekog pozitivnog skupa $\mathbf{x}^k$, nakon čega uzorkujemo podskup tog skupa. S druge strane, negativne skupove dobivamo uzorkovanjem dva različita pozitivna skupa i spajanjem u jedan skup. Na primjer, neka su zadani koreferentni skupovi {*Red Riding Hood*, *young girl*, *granddaughter*, *child*} te {*evil wolf*, *beast*}. Pozitivan skup je bilo koji podskup prvog skupa, poput {*granddaughter*, *child*}. S druge strane, negativan skup je {*Red Riding Hood*, *evil wolf*}.

Učenje modela logike prvog reda je analogno učenju modela parova. Jedina razlika je u obliku podataka za učenje i ciljne funkcije $l(\boldsymbol{\theta})$, koja je sad oblika

$$l(\boldsymbol{\theta}) = l(\theta_1, \ldots, \theta_K) = \sum_{ij} \log \mathbb{P}(y_j|\mathbf{x}^j),$$

pri čemu je $\mathbb{P}(y_{ij}|\mathbf{x}^j)$ kao u (3.3) ili (3.4).

Model se primjenjuje u tekstu na sljedeći način. Nad imeničkim frazama se prvo odvija pohlepno aglomerativno klasteriranje, odnosno, nenadzirano grupiranje imeničkih fraza. Svako grupiranje ocjenjujemo proporcionalno vjerojatnosti takvog skupa u modelu. Postupak završava kada više ne postoji grupiranje koje bi povećalo ocjenu dobivenog skupa, odnosno, vjerojatnost da su svi elementi tog klastera međusobno koreferentni. Na primjer, neka se u tekstu fraze *Red Riding Hood*, *young girl*, *child*, *granddaughter*, *she* odnose na Crvenkapicu. Uz njih, algoritam je pronašao i fraze *evil wolf*, *old lady*. Postupak može izgledati ovako:

1: Grupiraju se {*Red Riding hood*, *young girl*}. Vjerojatnost dobivenog skupa je 0.64.

2: Grupiraju {*Red Riding hood*, *young girl*, *child*}. Vjerojatnost dobivenog skupa iznosi 0.7.

3: Ne grupiraju se {*Red Riding hood*, *young girl*, *child*} i {*old lady*}, jer bi vjerojatnost tako dobivenog skupa bila 0.2.

4: Grupiraju se {*granddaughter*} i {*she*}. Vjerojatnost dobivenog skupa je 0.4.

5: Grupira se {*Red Riding hood*, *young girl*, *child*} i {*granddaughter*, *she*} u jedan skup vjerojatnosti 0.8.

6: Ne grupira se {*Red Riding hood*, *young girl*, *child*, *granddaughter*, *she*} i {*evil wolf*}, jer bi vjerojatnost tako dobivenog skupa bila 0.1.

7: Postupak završava, jer su svi izrazi obuhvaćeni.

3.3 Metoda višeprolaznog sita

Metoda višeprolaznog sita (eng. *multi-pass sieve*) je modularna metoda koju koristi Stanford Deterministic Coreference Resolution System, jedan od danas najuspješnijih i najpoznatijih sustava za razrješavanje koreferencije.

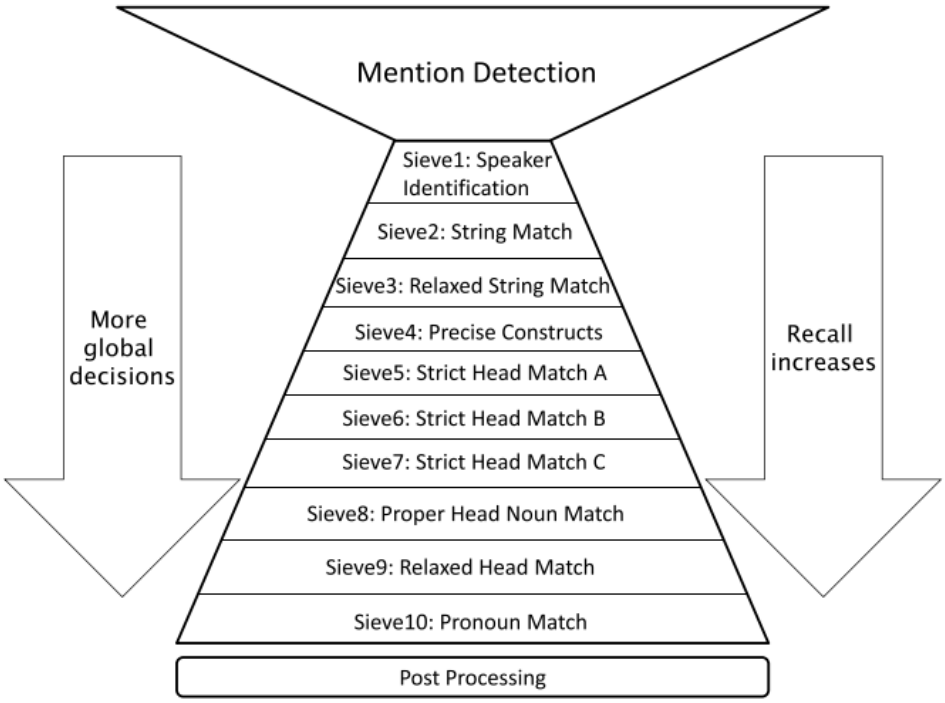
Većina pristupa rješavanju problema koreferencije se temelji na korištenju jedinstvene funkcije nad skupom ograničenja, odnosno, svojstava. Promatraju se lokalna leksička, sintaktička, semantička i/ili diskurzivna svojstva teksta. To nipošto nije loše — ovakav pristup je rezultirao veoma kvalitetnim rješenjima [5]. Unatoč tome, u praksi se javljaju određeni problemi.

Prvi problem ovog pristupa se javlja zbog prevelikog broja svojstava. Neka svojstva se u tekstu mogu javljati veoma često, ali sama po sebi nisu jako precizna. S druge strane, postoje i svojstva koja se javljaju rijetko, ali su vrlo precizna. Kako funkcija uzima u obzir sva svojstva, postoji mogućnost da u izračunu prevagnu češća, manje precizna svojstva, odnosno, da se zanemare one karakteristike koje daju točno rješenje. Drugi problem je u lokalnosti informacija. Često ne možemo iz neposredne okoline ispravno zaključiti s kojim entitetom je izraz u koreferenciji, već treba promatrati širu okolinu izraza.

Metoda višeprolaznog sita primjenjuje modele, počevši od najpreciznijeg, prema najmanje preciznome. Svaki model gradi nad prethodno dobivenim informacijama — u svakom se koraku koriste sve dotad poznate informacije. Metoda koristi isključivo determinističke modele, odnosno, pri svakoj primjeni neke komponente dobiva se jednak rezultat. Svaka od komponenti je oblikovana nenadziranim učenjem. To znači da nije potrebno imati velike skupove ručno označenih podataka za učenje

do kojih je, kao što smo ranije spomenuli, mukotrpno doći, već je dovoljno predati podatke modelu i on će ih automatski obraditi.

Stanford Deterministic Coreference Resolution System se sastoji primarno od dvije faze, otkrivanja navoda (eng. *mentions*) i razrješavanja koreferencije. Na slici 3.4 vidimo građu ovog sustava.



Slika 3.4: Građa Stanfordovog sustava za razrješavanje koreferencije

U prvoj fazi se algoritmom visokog odziva otkrivaju svi navodi — imeničke fraze, zamjenice, imenovani entiteti. Visoki odziv znači da su u tekstu označeni gotovo svi navodi entiteta. Potom se iz tog skupa izbacuju svi izrazi koji nisu navodi, poput numeričkih entiteta, pleonastičkog *it*[1], ili partitiva[2].

[1]Primjer pleonastičkog *it* je rečenica *"**It** rains."*. U hrvatskom jeziku su analogon besubjektne rečenice, poput *Kiši.*

[2]Partitivi određuju količinu nebrojivih imenica, poput ***half** of cookies*, ***bag** of flour*. U hrvatskom jeziku je takva struktura poznata pod nazivom dijelni genitiv.

U drugoj fazi se nad označenim tekstom primjenjuje deset nezavisnih modela za razrješavanje koreferencije. Svi modeli su testirani nad istim, unaprijed određenim skupom podataka, čime se utvrđuje njihova preciznost. Prednost nezavisnih modela je ta što, u bilo kojem trenutku, na bilo kojoj razini, možemo nadograditi sustav novim modelima. Ti modeli koriste sve informacije koje su dotad poznate sustavu — na primjer, u zadnjem koraku, razrješavanju zamjeničke koreferencije, osim okoline navoda, koristimo podatke kao što su vrsta entiteta, spol, broj (jednina/množina) i lema[3].

Upravo iz dvofazne arhitekture slijedi visoka preciznost i visok odziv sustava, odnosno, visoka F-mjera sustava. Konkretno, prema [7], prosječna F-mjera dobivena testiranjem nad više velikih skupova podataka iznosi 0.5956. To se naizgled ne čini mnogo, ali je vrlo dobar rezultat kad se u obzir uzme složenost problema.

[3]Rječnička natuknica ili lema je osnovna riječ u rječniku, leksikonu ili enciklopediji, koja se objašnjava, odnosno, definira i s kojom započinje članak takve knjige.

Poglavlje 4

Modeli za analizu sentimenta

4.1 Model "vreća riječi"

Model **"vreća riječi"** (eng. *bag-of-words model*) je najjednostavniji model koji se koristi u analizi sentimenta. Model se, također, može koristiti i za otkrivanje spam e-mailova. Tekst se prikazuje isključivo kao skup riječi koje ga čine. Preciznije, model promatra multiskup riječi, s obzirom da je važno koliko često se javlja pojedina riječ, ali se ignorira njihov poredak. Također je moguće ignorirati i oblik riječi, odnosno, promatrati isključivo infinitive glagola, jednine imenica i slično.

Na primjer, rečenicu *"This story is interesting and it has good characters and good ending."* možemo prikazati sljedećom "vrećom riječi":

Riječ	Frekvencija
this	1
story	1
is	1
interesting	1
and	2
it	1
has	1
good	2
characters	1
ending	1

Za određivanje sentimenta rečenice je dovoljno odrediti koliko ima pozitivnih riječi, a koliko negativnih. Ako ima više pozitivnih riječi nego negativnih, sentiment je pozitivan. U protivnom je negativan.

Preduvjet tom koraku je oblikovanje skupova svih pozitivnih i svih negativnih riječi. Takvi skupovi su obično dostupni na internetu, te su sortirani abecedno:

$$Positive = \{acclaim, \ldots, blessing, \ldots, captivate, \ldots, divine, \ldots\},$$
$$Negative = \{absurd, \ldots, betrayal, \ldots, cheating, \ldots, death, \ldots\}.$$

Za svaku riječ u vreći provjeravamo nalazi li se u nekom od ova dva skupa i, ovisno o tome, ju bodujemo pozitivno ili negativno, s onoliko bodova kolika je frekvencija te riječi. U gornjem primjeru imamo dvije pozitivne riječi, *interesting* i *good*, koje se ukupno javljaju tri puta. Kako nemamo nijednu negativnu riječ, konačno stanje bodova je 3, te zaključujemo da je rečenica pozitivnog stava.

S obzirom da metoda ignorira poredak riječi, gramatiku i bilo kakve dodatne informacije koje možemo izvući iz teksta, česte su pogreške. Na primjer, metoda jednako promatra sljedeće izjave:

I am disappointed. This book is not very good.
I am not disappointed. This book is good.

U oba slučaja rezultat će biti neutralan stav. Naime, po jednom se javljaju pozitivna riječ *good* i negativna riječ *disappointed*. Ostale riječi nisu prepoznate kao pozitivne ili negativne.

Ovaj problem možemo riješiti tako da, osim samih riječi, odnosno, unigrama, promatramo i bigrame ili trigrame. Na primjer, ukoliko se u rečenici javlja neka od riječi *not*, *just*, *very*, *no*, nju i njezinog sljedbenika promatramo kao cjelinu, dok sljedbenika više ne promatramo zasebno. Na primjer, izraz *not good* se promatra kao negacija pozitivne riječi *good*, pa zaključujemo da je izraz negativan. Samu riječ *good* više ne uzimamo u obzir za bodovanje.

4.2 Model "vreća stavova"

Model **"vreća stavova"** (eng. *bag-of-opinions*) je nastao kao odgovor na probleme modela "vreće riječi". Model, također, omogućuje relativno uspješnu stupnjevitu analizu sentimenta, na primjer, raspoznavanje vrlo pozitivnih od umjereno pozitivnih stavova.

Ovaj model promatra tekst kao skup stavova s proizvoljnim brojem modifikatora i negatora. Možemo ga opisati na sljedeći način.

Neka se tekst $\mathbf{x}$ sastoji od skupa stavova $O = \{\mathbf{o}_1, \ldots, \mathbf{o}_n\}$. Svaki stav $\mathbf{o}_i$ se sastoji od:

- korijena w_r, za neki $r \in R$, pri čemu je R skup svih indeksa korijena teksta $\mathbf{x}$,

- skupa modifikatora $\{w_m \mid m \in M'\}$, pri čemu je M' podskup skupa M svih indeksa modifikatora teksta $\mathbf{x}$,
- skupa negatora $\{w_z \mid z \in Z'\}$, pri čemu je Z' podskup skupa Z svih indeksa negatora teksta $\mathbf{x}$.

Za proizvoljni stav $\mathbf{o}$ u tekstu, definiramo funkciju $score : O \to \mathbb{R}$ na sljedeći način:

$$score(\mathbf{o}) = \text{sign}(r)\beta_r x_r + \sum_{m \in M} \text{sign}(r)\beta_m x_m + \sum_{z \in Z} \text{sign}(r)\beta_z x_z.$$

Funkcija $\text{sign} : R \to \{-1, 1\}$ označava polaritet stava i definira se kao

$$\text{sign}(r) = \begin{cases} 1, & \text{korijen } w_r \text{ je pozitivan,} \\ -1, & \text{korijen } w_r \text{ je negativan.} \end{cases}$$

Nadalje, β_i su težine svakog elementa stava $\mathbf{o}$. Na primjer, *extremely* ima značajno veću težinu od *slightly*. Iz samog modela slijedi da težine modifikatora moraju biti pozitivne, dok su negatori negativnih težina, s obzirom da mijenjaju stav u suprotan.

Također, x_z, x_m, x_r su binarne varijable definirane s

$$x_i = \begin{cases} 1, & w_i \text{ je prisutan u stavu } \mathbf{o}, \\ 0, & \text{inače.} \end{cases}$$

Konačno, funkcija f, koja za tekst $\mathbf{x}$ predviđa sentiment, definira se kao prosjek ocjena svih stavova pronađenih u tekstu:

$$f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} score(\mathbf{o}_i). \tag{4.1}$$

Slično kao i u "vreći riječi", ako je $f(\mathbf{x}) \geq 0$, zaključujemo da je stav pozitivan. Ukoliko je $f(\mathbf{x}) \leq 0$, stav je negativan. Dodatno, ovisno o vrijednosti $f(\mathbf{x})$, možemo stupnjevati stav — stav je vrlo negativan, umjereno negativan, umjereno pozitivan, ili vrlo pozitivan.

Parametri modela $\boldsymbol{\beta} = \{\beta_r, \beta_m, \beta_z \mid r \in R, m \in M, z \in Z\}$ se određuju učenjem nad označenim podacima, slično postupcima opisanim u prethodnom poglavlju. Ovdje je postupak nešto kompliciraniji pa ga nećemo detaljno opisivati. Također, nećemo opisivati postupak prepoznavanja stavova u tekstu, s obzirom da je to zaseban problem.

4.3 Naivni Bayesov klasifikator

U prethodnom poglavlju smo opisali naivni Bayesov klasifikator. Sada ćemo opisati kako se on može koristiti za analizu sentimenta.

Naivni Bayesov klasifikator, također, možemo promatrati kao svojevrsno unaprjeđenje modela "vreće riječi". Naime, prvi korak je izgraditi vreću riječi zadanog teksta, ali za određivanje sentimenta nećemo koristiti ranije opisani sustav "bodovanja", već sljedeći postupak.

Neka je dan tekst $\mathbf{x}$, duljine n, u kojem su označene riječi koje opisuju pozitivan, odnosno, negativan sentiment. Neka je c oznaka neke od klasa *pozitivno*, *negativno*. Za svaku riječ w_i u tekstu računamo sljedeću vjerojatnost:

$$\mathbb{P}(w_i|c) = \frac{count(w_i, c) + 1}{count(w, c) + n},$$

pri čemu $count(w_i, c)$ označava broj pojava riječi w_i koje su označene klasom c, dok $count(w, c)$ označava ukupan broj riječi u tekstu označenih klasom c.

Za čitavi tekst $\mathbf{x}$ potom računamo

$$\mathbb{P}(\mathbf{x}|c) = \mathbb{P}(c) \prod_{i=1}^{n} \mathbb{P}(w_i|c),$$

pri čemu je $\mathbb{P}(c)$ vjerojatnost pojedine klase. Konačno, zaključujemo da tekst pripada onoj klasi c za koju gornji izraz postiže maksimum.

4.4 Duboko učenje

Za sam kraj, opisujemo jednu od danas najuspješnijih metoda analize sentimenta koja je razvijena na Sveučilištu u Stanfordu. Radi se o **dubokom učenju** (eng. *deep learning*), konkretno, o **rekurzivnim dubokim modelima za semantičku djeljivost nad Sentiment Treebankom** (eng. *Recursive deep models for semantic compositionality over a Sentiment Treebank*).

Za početak, objasnimo uopće pojam dubokog učenja. **Duboko učenje** je vrlo mlada grana strojnog učenja, čiji cilj je podatke reprezentirati vrlo složenim modelima, koji su rezultat niza nelinearnih transformacija. Jedan takav model se sastoji od više slojeva koji su, najčešće, oblikovani nenadziranim učenjem. Svaki sloj za ulaz uzima izlaz prethodnog sloja i možemo ga shvatiti kao jednu razinu apstrakcije podataka. Prvi sloj, najčešće, za cilj ima prepoznati bitne karakteristike ulaznih podataka (eng. *feature extraction*). Na primjer, ukoliko je modelu za prepoznavanje lica predana fotografija, prvi sloj može prepoznati određene točke na fotografiji

(kutovi usana, očiju, krajevi obrva), dok neki kasniji sloj obrađuje isječke dane fotografije (usne, oči, obrve).

Modeli dobiveni dubokim učenjem se uspješno primjenjuju za rješavanje problemima u kojima je složenost podataka veoma visoka, poput obrade prirodnog jezika ili računalnog vida.

Duboko učenje se pretežno temelji na neuronskim mrežama. **Neuronske mreže** (eng. *neural networks*) su model strojnog učenja motiviran biološkim neuronima, odnosno, ljudskim mozgom. Biološki neuron se sastoji od tijela, ulaznih veza koje nazivamo dendriti i izlazne veze koju nazivamo akson. Ukoliko jačina ulaznih signala na dendritima prijeđe neku granicu, neuron se aktivira i šalje određenu vrijednost preko aksona. Sljedeći neuroni taj izlaz primaju kao svoj ulaz te se postupak ponavlja.

Shodno tome, umjetnu neuronsku mrežu čini slojevito organiziran skup umjetnih neurona. Veze među neuronima u slojevima k i $k+1$ imaju težine $w_{ij}^{(k)}$, koje određuju utjecaj određenog ulaza (s neurona i na sloju k) na rezultat neurona j na sloju $k+1$. Uobičajeno se u svakom sloju dodaje dodatni neuron koji određuje pristranost ili pomak (eng. *bias*) ulaza. Početni sloj neurona se naziva **ulazni sloj** (eng. *input layer*) i prima vektor ulaznih podataka $\mathbf{x}$. Ulaz se prosljeđuje sljedećem sloju neurona. U svakom od tih neurona se računa vrijednost **aktivacijske funkcije**. Ona može biti proizvoljnog oblika, ali najčešće se koristi sigmoidalna funkcija definirana s

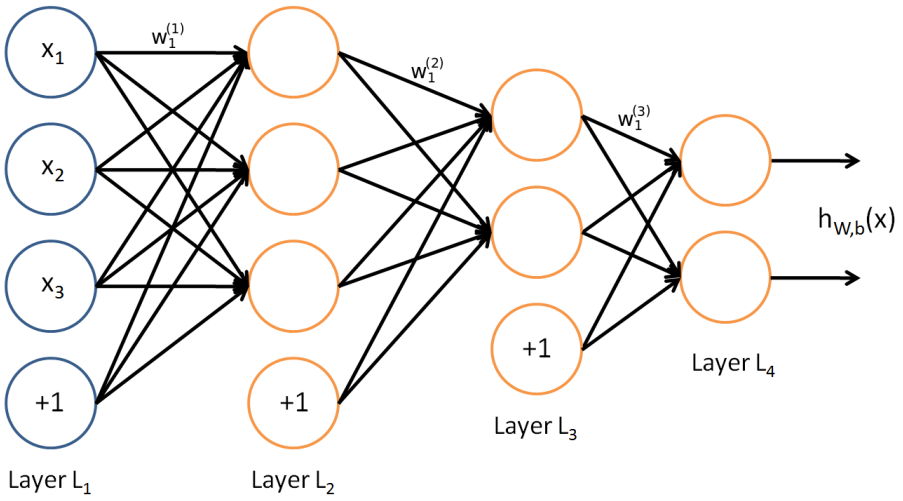
$$\psi(\mathbf{v}) = \frac{1}{1+e^{-\mathbf{v}}},$$

pri čemu je $\mathbf{v}$ težinska linearna kombinacija svih ulaza i pomaka, za odgovarajući neuron.

Neuron, kao izlaz, šalje izračunatu vrijednost aktivacijske funkcije za svoj ulaz. Sljedeći sloj neurona prima te vrijednosti kao ulaz i ponavlja računanje. Postupak završava u zadnjem, **izlaznom sloju** (eng. *output layer*). Svi slojevi neurona između ulaznog i izlaznog sloja se nazivaju **skriveni slojevi** (eng. *hidden layers*).

Uobičajeno se ulazni sloj ne uračunava u broj slojeva mreže, s obzirom da se u njemu ne odvija nikakvo računanje. Stoga, kad kažemo "jednoslojna mreža" mislimo na mrežu sastavljenu samo od ulaznog i izlaznog sloja.

Na sljedećoj slici prikazana je arhitektura jedne jednostavne neuronske mreže. S L_1 je označen ulazni sloj mreže. Slojevi L_2 i L_3 su skriveni slojevi, dok je L_4 izlazni sloj. Ulazni sloj L_1 prima zadani ulazni vektor $\mathbf{x} = (x_1, x_2, x_3)$. Na svakom sloju, s $+1$ su označeni dodatni neuroni koji generiraju pomak i time utječu na izlaz aktivacijske funkcije za zadani ulaz. Na slici su označene samo težine veza između prvih neurona na slojevima k i $k+1$, uz skraćenu oznaku $w_1^{(k)} = w_{11}^{(k)}$.



Slika 4.1: Primjer neuronske mreže

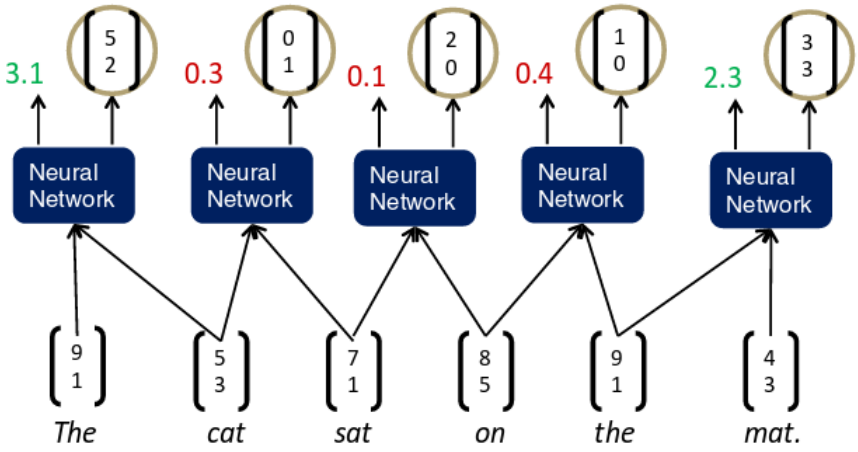
Neuronske mreže koje se koriste u dubokom učenju su vrlo kompleksne arhitekture. Konkretno, Stanfordov model za analizu sentimenta koristi **rekurzivne neuronske tenzorske mreže** (eng. *recursive neural tensor networks*).

Prvo opišimo općenite rekurzivne neuronske mreže. Ove mreže se koriste za određivanje strukture podataka i to pomoću stabala. Svaki čvor-roditelj u stablu se prikazuje kao nelinearna transformacija svojih čvorova-djece. Drugim riječima, svi čvorovi, osim listova, nastali su kao izlaz (iste) neuronske mreže, kojoj su kao ulaz dani čvorovi nastali u prethodnom koraku. Očito se radi o rekurzivnom postupku koji započinje s listovima, a završava s korijenom stabla.

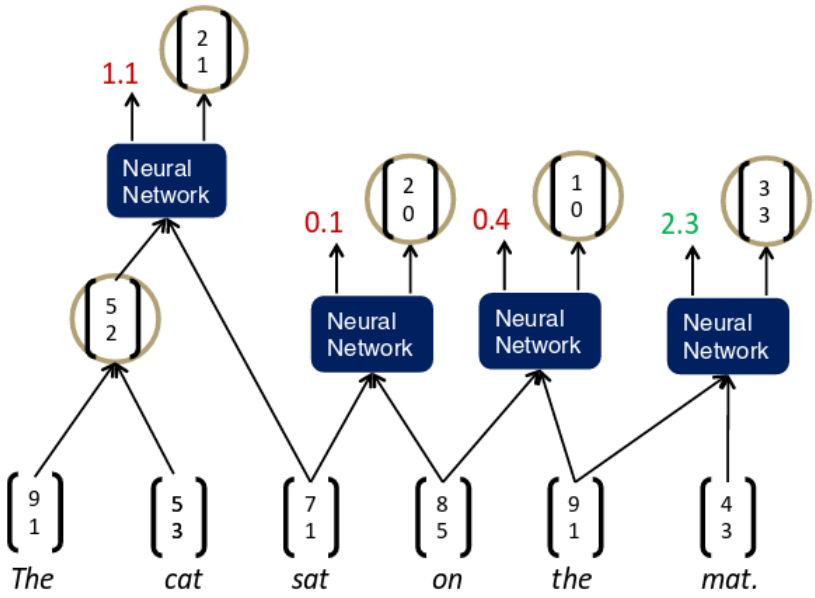
Rekurzivne neuronske mreže se najviše koriste u obradi prirodnog jezika i određivanju strukture slika (primjer toga je prepoznavanje ceste, stabla i neba, kao zasebnih cjelina na slici).

Na sljedećem nizu slika vidimo primjer određivanja gramatičke strukture rečenice rekurzivnom neuronskom mrežom. Na početku je svaka riječ predstavljena nekim vektorom. Te riječi čine listove budućeg stabla. Neuronska mreža računa moguće roditelje tih listova u stablu. Osim vrijednosti vektora, izračunava se i njegova ocjena. Što je ta ocjena viša, to je vjerojatnije da je dobiveni čvor ispravan. Na slikama su zelenom bojom označene visoke ocjene čvorova. Takve čvorove zadržavamo u stablu. S druge strane, crvenom bojom su označene loše ocjene čvorova. Te čvorove odbacujemo.

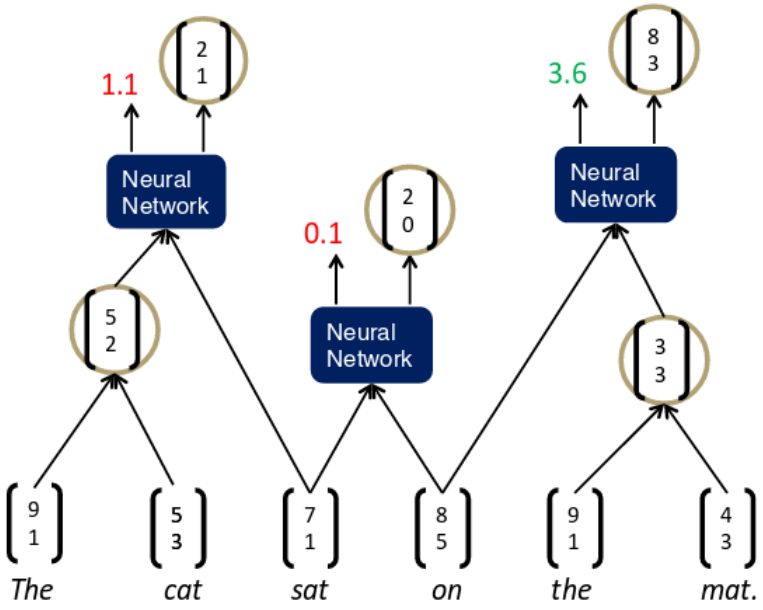
Za rečenicu *"The cat sat on the mat."*, na slikama su prikazana prva 3 koraka parsiranja i završno stablo parsiranja.



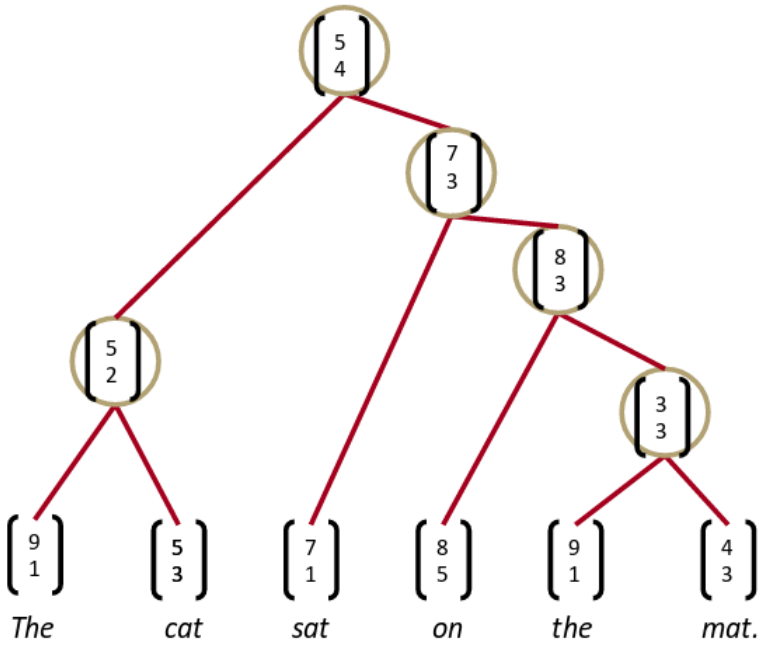
Slika 4.2: Prvi korak parsiranja — "the cat" i "the mat" su dobri čvorovi pa ih zadržavamo.



Slika 4.3: Drugi korak parsiranja — svi novi čvorovi su loši te ih odbacujemo.



Slika 4.4: Treći korak parsiranja — čvor "on the mat" je dobar te ga dodajemo u stablo.



Slika 4.5: Završno stablo parsiranja — s obzirom da je novi čvor korijen, postupak završava.

U osnovnim rekurzivnim neuronskim mrežama, vrijednost d-dimenzionalnog vektora čvora-roditelja se računa na sljedeći način. Neka su l, r oznake za, redom, lijevo i desno dijete čvora p. Označimo s h_p vrijednost vektora roditelja p. S h_l i h_r označimo, redom, vrijednosti vektora djece l i r. Tada je

$$h_p = f\left(W \begin{bmatrix} h_l \\ h_r \end{bmatrix} + b\right),$$

pri čemu je W matrica težina neuronske mreže, dok je b vektor pomaka koji omogućuje točniji model. Nadalje, kao f se uobičajeno koristi funkcija tangens hiperbolni, po svakoj komponenti vektora.

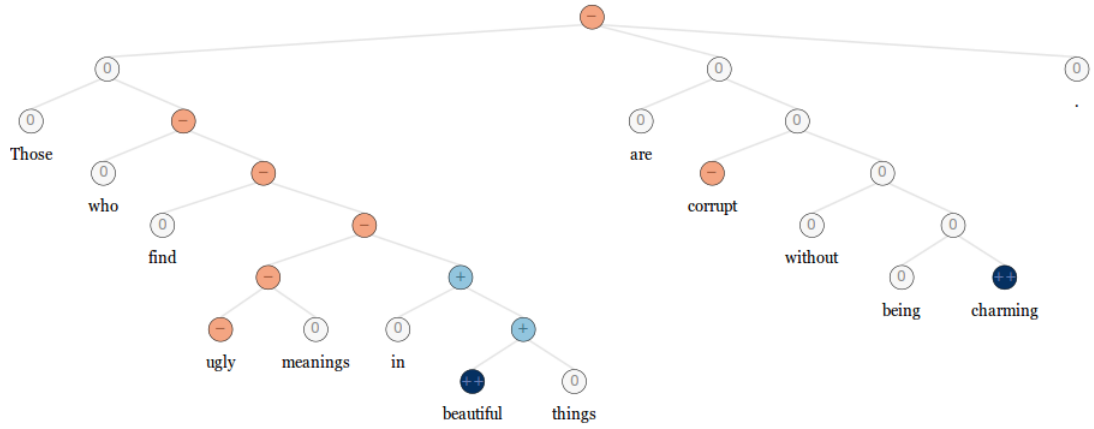
S druge strane, u **rekurzivnim neuronskim tenzorskim mrežama** se na kompleksniji način računa vrijednost vektora roditelja. Ranije su isključivo vrijednosti čvorova-djece utjecale na vrijednost roditelja, dok sada dodatan utjecaj ima i međusoban odnos djece. To je postignuto pomoću podizraza $(*)$ u sljedećem izrazu:

$$h_p = f\Bigg(\underbrace{\begin{bmatrix} h_l \\ h_r \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} h_l \\ h_r \end{bmatrix}}_{(*)} + W \begin{bmatrix} h_l \\ h_r \end{bmatrix} + b\Bigg),$$

pri čemu je V tenzor nad $\mathbb{R}^d \times \mathbb{R}^{2d} \times \mathbb{R}^{2d}$. Tenzor možemo jednostavno opisati kao višedimenzionalno poopćenje matrica. Matrica W i vektor b imaju jednaku ulogu kao ranije.

Sada kada nam je poznat Stanfordov model za analizu sentimenta, možemo opisati i podatke koji su se koristili za njegovo treniranje. Radi se o *Stanford Sentiment Treebanku*, u slobodnom prijevodu *banci sentimentnih stabala*, koje čini gotovo 12 000 rečenica. Svaka od tih rečenica je automatski parsirana, odnosno, računalnim modelima je generirano njezino stablo parsiranja. Nadalje, u tim rečenicama se nalazi oko 215 000 jedinstvenih fraza koje su trojica "sudaca" ručno označili kao vrlo negativne, negativne, neutralne, pozitivne ili vrlo pozitivne. Na slici 4.6 vidimo primjer parsirane rečenice s označenim čvorovima. Oznake su vezane uz boje na sljedeći način:

- vrlo negativno
- negativno
- neutralno
- pozitivno
- vrlo pozitivno



Slika 4.6: Rečenica *"Those who find ugly meanings in beautiful things are corrupt without being charming."* prikazana kroz stablo parsiranja s označenim čvorovima.

Točnost ovog modela je 80.7%, što je izuzetno visoko. Sasvim je moguće da je ovaj model statistički točniji od "ljudskog" modela, odnosno, analize sentimenta koju provode ljudi. Naime, izvedeno je više istraživanja točnosti "ljudskih" modela te je u njima točnost varirala od 70% do najviše 90% [11, 4]. To znači da se u 70–90% slučajeva ljudi slažu oko stava neke rečenice, dok se u ostalim slučajevima ne slažu.

Poglavlje 5

Razvoj aplikacije za karakterizaciju likova

5.1 Korišteni alati

Temelj nad kojim smo razvijali aplikaciju za karakterizaciju likova je *Stanford CoreNLP* skup alata za obradu prirodnog jezika. To je skup brojnih alata razvijen na Sveučilištu u Stanfordu. Alati su implementirani u programskom jeziku *Java*, ali ih je moguće koristiti na više načina. Najjednostavniji način je pokretanje tih alata kao samostalnih aplikacija kojima se pristupa preko web preglednika. U pregledniku predajemo tekst i odabiremo obrade koje želimo provesti. Osim toga, postoje sučelja prema brojnim programskim jezicima, poput *Pythona*, pomoću kojih možemo iz tog programskog jezika pozvati sučelja prema Stanfordovom alatu. Konačno, moguće je i izravno u *Javi* pristupati klasama iz Stanfordovih alata, čime dobivamo potpunu integraciju korisničkog koda i koda samog alata.

Od brojnih alata, izravno ćemo koristiti samo tri — alate za prepoznavanje imenovanih entiteta, za razrješavanje koreferencije te za analizu sentimenta. Prvi alat se temelji na uvjetnim slučajnim poljima, opisanim u poglavlju 2. Sljedeći alat, za razrješavanje koreferencije, koristi metodu višeprolaznog sita, opisanu na kraju poglavlja 3. Posljednji alat, za analizu sentimenta, se temelji na dubokom učenju, odnosno, rekurzivnim neuronskim mrežama, opisanim u zadnjoj sekciji poglavlja 4. Kao što smo već rekli, sva tri modela pripadaju danas najuspješnijim modelima za rješavanje danih problema.

Osim ovih alata, *Stanford CoreNLP* uključuje brojne druge, poput tokenizatora, koji rastavlja tekst na zasebne riječi (tokene), parsera gramatičke strukture rečenica ili označivača vrste riječi u rečenici. Mnoge od tih alata ćemo koristiti posredno. Na primjer, za analizu sentimenta dubokim učenjem je preduvjet odrediti gramatičku

strukturu rečenice, a jedan od preduvjeta za to je prepoznati sve tokene u danoj rečenici. Dovoljno je sustavu predati popis potrebnih alata, te će *Stanford CoreNLP* automatski provesti sve korake u traženom redoslijedu.

Koristimo posljednju verziju alata, a to je 3.6.0.

Rješenje je implementirano u programskom jeziku Java. Za izgradnju aplikacije iz izvornog koda smo koristili sustav *Gradle*. *Gradle* omogućuje prilično jednostavnu i brzu izgradnju. Naime, ukoliko je potrebno koristiti dodatne biblioteke, dovoljno ih je potražiti na web stranici *Maven Repository*, odabrati verziju te dobivene podatke navesti unutar `build.gradle` dokumenta u odgovarajućoj sekciji. "Osvježavanjem" *Gradle* sustava se automatski dohvaćaju *Stanford CoreNLP* biblioteka i njezini već istrenirani modeli.

Nadalje, korištena je biblioteka `sl4j` za logiranje, odnosno, ispis trenutnog stanja u konzolu, odnosno, tekstualnu (takozvanu *log*) datoteku. Biblioteka se koristi i unutar *Stanford CoreNLP* biblioteke za istu svrhu.

Slijedi isječak iz `build.gradle` datoteke:

```
dependencies {
   compile group: 'edu.stanford.nlp', name: 'stanford-corenlp',
      version: '3.6.0'
   compile group: 'edu.stanford.nlp', name: 'stanford-corenlp',
      version: '3.6.0', classifier: 'models'
    compile group: 'org.slf4j', name: 'slf4j-simple', version:
       '1.7.21'
}
```

Kao razvojno okruženje koristili smo *IntelliJ IDEA*, integrirano razvojno okruženje za programski jezik Java, koje podržava *Gradle* alat i omogućuje jednostavno pokretanje *Gradle* zadaća i same Java aplikacije.

5.2 Organizacija projekta

Projekt slijedi uobičajenu organizaciju gotovo svih Java projekata. Datoteke projekta su organizirane u dva temeljna direktorija, `main` i `test`. Prvi direktorij sadrži temeljni dio aplikacije nužan za rad, dok se u potonjem direktoriju nalaze podaci za testiranje rada aplikacije.

U direktoriju `main` se nalaze direktoriji `java` i `resources`. Njihova uloga jasna je iz naziva — `java` direktorij sadrži programski kod, odnosno, klase organizirane u pakete. S druge strane, u `resources` direktoriju se nalaze različiti dokumenti potrebni za rad aplikacije.

Direktorij `test` sadrži samo jedan direktorij, `resources`, u kojem se pak nalazi direktorij `stories`. Ondje se nalaze odabrane dječje priče nad kojima smo testirali rad aplikacije.

Programski kod je organiziran u klase. Klase su dalje grupirane po ulozi u različite pakete:

- paket `trainer`, u kojem se nalazi klasa `CharacterRecogniserTrainer` koja provodi treniranje novog NER modela,
- paket `solver`, u kojemu se nalazi istoimena klasa koja provodi analizu priče od početka do kraja,
- paket `phase`, u kojem se nalaze klase za svaku od tri faze našeg rješenja, `CharacterRecognition`, `CoreferenceResolution` i `SentimentAnalysis`,
- paket `model`, koji sadržava klase za čuvanje podataka, u svrhu lakše obrade teksta — `StoryData` i `CharacterName`,
- paket `util`, u kojem se nalazi klasa `Constants` u kojoj su pohranjene konstante, poput lokacije priča, istreniranih modela i slično.

Osim navedenih klasa, imamo i klasu `MainClass` koja se ne nalazi u niti jednom paketu. Ova klasa sadržava istoimenu metodu, pokretanjem koje započinjemo izvršavanje programa.

Nadalje, resursi su organizirani u dva direktorija — `stanford` i `stories`. U prvom direktoriju se nalaze različiti dokumenti potrebni za rad *Stanford CoreNLP* alata. To su:

- `coref-animate.txt`, koji sadržava listu svih entiteta koje će sustav za razrješavanje koreferencije smatrati živim entitetima i za koje će provoditi navedeni postupak,
- `ner-training-data.tsv` i `ner-training-data-no-punctuation.tsv`, koji čini skup podataka za treniranje novog NER modela koji će moći prepoznavati likove u tekstu,
- `ner-training.properties`, u kojem su zapisane postavke samog treniranja NER modela,
- `ner-model.ser.gz` i `ner-model-no-punctuation`, koji predstavljaju pohranjene NER modele, čime izbjegavamo potrebu da svaki put prilikom pokretanja aplikacije treniramo model otpočetka.

Kasnije ćemo detaljnije opisati značenje i upotrebu gore navedenih datoteka.

5.3 Klasa `MainClass`

Posebnost Java programskog jezika je u tome što sav kod mora biti organiziran u klase, te mora postojati metoda `main()` u nekoj od klasa. Pozivom ove metode započinje izvršavanje naše aplikacije. Nadalje, zbog preglednosti koda, sve važne obrade nad pričom su izdvojene u zasebne klase. U klasi `MainClass` je implementiran isključivo uvodni korak — treniranje NER modela ukoliko on ne postoji, rad s tekstualnim datotekama i komunikacija s korisnikom.

Prije samog koda, napomenimo da se nudi mogućnost treniranja NER modela s podacima koji sadržavaju interpunkcijske znakove, kao i s podacima iz kojih su takvi znakovi izbačeni. Detaljnije o tome ćemo reći kasnije.

```
public static void main(String... args) {

   Solver solver = new Solver();
   Scanner scanner = new Scanner(System.in);
   boolean usePunctuation;

   // first step - decide type of NER classifier to use
   if (!Files.exists(Paths.get(modelLocation))) {
      trainNewCharacterRecogniser(usePunctuation);
   } else {
      System.out.println("Existing NER model for character
          recognition found. \n" +
            "Do you want to train new model or use old model? \n" +
            "Please enter 'y' if you want to train new model or 'n'" +
               "if you want to skip training.\n" +
            "If you want to exit, type 'q'.");

      while (true) {
         String input = scanner.next();
         String trimmedInput = input.trim();
         if (trimmedInput.equalsIgnoreCase("q")) {
            return;
         } else if (trimmedInput.equalsIgnoreCase("y")) {
            trainNewCharacterRecogniser(usePunctuation);
            break;
         } else if (trimmedInput.equalsIgnoreCase("n")) {
            break;
         } else {
            System.out.println("Please enter 'y', 'n' or 'q'.");
         }
```

```
        }
    }

    String modelLocation;
    if (usePunctuation) {
        modelLocation = NER_MODEL;
    }
    else {
        modelLocation = NER_MODEL_NO_PUNCTUATION;
    }

    // second step - check if NER classifier exists
    if (Files.exists(Paths.get(modelLocation))) {
        System.out.println("Existing NER model for character
            recognition found. Do you want to use this model or train
            new model? Please enter 'u' (use), 't' (train) or 'q'
            (exit).");

        while (true) {
            String input = scanner.next();
            String trimmedInput = input.trim();
            if (trimmedInput.equalsIgnoreCase("q")) {
                return;
            } else if (trimmedInput.equalsIgnoreCase("u")) {
                break;
            } else if (trimmedInput.equalsIgnoreCase("t")) {
                trainNewCharacterRecogniser(usePunctuation);
                break;
            } else {
                System.out.println("Please enter 'u', 't' or 'q'.");
            }
        }
    } else {
        trainNewCharacterRecogniser(usePunctuation);
    }

    // third step - solve for story
    System.out.println("Please enter file name without extension from
        '/src/main/resources/stories/', e.g. 'snowwhite if file name is
        snowwhite.txt. File should contain only story text, no titles,
        footnotes etc.\n If you want to exit, type 'q'.");
```

```
        while (true) {
            String input = scanner.next();
            String trimmedInput = input.trim();
            if (trimmedInput.isEmpty()) {
                System.out.println("Filename must not be empty.");
                continue;
            }
            if (trimmedInput.equalsIgnoreCase("q"))
                return;

            String storyPath = STORIES_LOCATION + input + TXT;
            if (!Files.exists(Paths.get(storyPath))) {
                System.out.println("File does not exist! Please enter
                    existing filename.");
                continue;
            }

            try {
                solver.solveForStory(storyPath, usePunctuation);
            } catch (IOException e) {
                System.out.format("Error while reading file: %s",
                    e.getMessage());
            }
            System.out.println("\n\n");
            System.out.println("Enter next file name or 'q' to exit.");
        }
    }
```

Metoda `trainNewCharacterRecogniser(boolean usePunctuation)` je pomoćna metoda kojom započinjemo treniranje novog NER modela:

```
    private static void trainNewCharacterRecogniser(usePunctuation) {
        CharacterRecogniserTrainer trainer = new
            CharacterRecogniserTrainer();
        trainer.train(usePunctuation);
    }
```

5.4 Klasa CharacterRecogniserTrainer

Prva klasa koja će se pozvati je `CharacterRecogniserTrainer` unutar paketa `ner`. Klasa sadrži metodu `train(boolean usePunctuation)` u kojoj je implementirano treniranje novog NER modela nad zadanim podacima za učenje. Ti podaci se nalaze u direktoriju `resources/stanford` u datotekama `ner-training-data.tsv` i `ner-training-data-no-punctuation.tsv`. Prva datoteka označava podatke za učenje u kojima sam token može biti neki interpunkcijski znak. Druga datoteka pak označava iste podatke, ali iz kojih su izbačeni interpunkcijski znakovi.

Podaci su dobiveni ručnim označavanjem ukupno 69 dječjih priča, koje smo proveli kolega Tomislav Horina i ja, u sklopu projekta *Ekstrakcija likova iz kratkih priča* na kolegiju Strojno učenje. Cilj projekta je bio istrenirati više NER modela koji će uspješno prepoznavati (imenovane i neimenovane) likove u danim pričama — bajkama, basnama, mitovima i legendama. Modeli su se temeljili na skrivenim Markovljevim modelima, uvjetnim slučajnim poljima te na *Stanford NER* sustavu. Najkvalitetniji modeli su dobiveni upravo treniranjem vlastitog *Stanford NER* modela, te je njihova F_2 mjera iznosila oko 0.6. Prilikom testiranja, model treniran nad podacima koji ne uključuju interpunkcijske znakove je pokazivao neznatno bolje rezultate.

Prvi korak projekta je bila priprema podataka za učenje. S web-stranice *Project Gutenberg* je prikupljen veći broj priča. Za svaku priču je automatski generirana po jedna `tsv` (*tab separated value*) datoteka dvostupčanog formata, pri čemu su u drugom stupcu, redom, tokeni te priče, dok je u prvom stupcu oznaka klase kojoj pripada taj token. Oznaka je `C` (*character*) ukoliko se radi o liku, inače je oznaka `O` (*other*).

Sljedeći tekst je isječak iz datoteke `ner-training-data.tsv` koja sadrži interpunkcijske znakove:

```
O Long
O ago
O there
O lived
O a
O monarch
O ,
O who
O was
O such
O a
O very
O honest
```

```
O man
O that
O his
O subjects
O entitled
O him
O the
C Good
C King
O .
```

S druge strane, ovo je analogni isječak iz datoteke `ner-training-data.tsv`, u kojoj su regularnim izrazima izbačeni svi interpunkcijski znakovi:

```
O Long
O ago
O there
O lived
O a
O monarch
O who
O was
O such
O a
O very
O honest
O man
O that
O his
O subjects
O entitled
O him
O the
C Good
C King
```

U generiranoj datoteci su, u početku, svi tokeni nosili oznaku `O`. Čitanjem priča smo kolega i ja uočavali likove te oznake pripadnih tokena mijenjali u `C`. Potom su sve datoteke spojene u jedinstvenu datoteku čija duljina je preko dvjesto tisuća linija. Također je stvorena kopija te datoteke, iz koje su regularnim izrazima uklonjeni svi retci koji sadrže interpunkcijske znakove. Te datoteke su činile skup podataka za treniranje NER modela.

Pri treniranju novog *Stanford NER* modela, osim datoteke s podacima za učenje, potrebno je predati i `ner-trainer.properties` datoteku. U njoj su zadane postavke treniranja, poput poretka stupaca u tim podacima i duljine n-grama koji će se promatrati pri treniranju. Korištene su preporučene postavke:

```
map = word=1,answer=0
useClassFeature=true
useWord=true
useNGrams=true
noMidNGrams=true
maxNGramLeng=6
usePrev=true
useNext=true
useSequences=true
usePrevSequences=true
maxLeft=1
useTypeSeqs=true
useTypeSeqs2=true
useTypeySequences=true
wordShape=chris2useLC
useDisjunctive=true
saveFeatureIndexToDisk=true
```

Ova datoteka se, također, nalazi u direktoriju `resources/stanford`.

Slijedi programski kod klase `CharacterRecogniserTrainer`:

```
public class CharacterRecogniserTrainer {

   private Logger logger =
       LoggerFactory.getLogger(CharacterRecogniserTrainer.class);

   public void train(boolean usePunctuation) {

      logger.info("Started training of new NER model. Model
          includes punctuation: {}", includePunctuation);

      Properties properties =
          StringUtils.propFileToProperties(TRAINING_PROPERTIES);
      if (includePunctuation) {
         properties.setProperty(TRAIN_FILE, TRAINING_DATA);
         location = NER_MODEL;
      }
```

```
            else {
               properties.setProperty(TRAIN_FILE,
                   TRAINING_DATA_NO_PUNCTUATION);
               location = NER_MODEL_NO_PUNCTUATION;
            }
            SeqClassifierFlags flags = new SeqClassifierFlags(properties);
            CRFClassifier<CoreLabel> crf = new CRFClassifier<>(flags);
            crf.train();
            crf.serializeClassifier(location);
            logger.info("Training finished. Model saved to {}.",
                NER_MODEL);
        }
    }
```

Nakon učitavanja `training.properties` datoteke, njezine vrijednosti se postavljaju kao postavke treniranja. Također se, ovisno o korisnikovom odabiru modela, postavlja lokacija gdje će se pohraniti istrenirani model. To je jedna od sljedećih lokacija:

- `resources/stanford/ner-model.ser.gz`
- `resources/stanford/ner-model-no-punctuation.ser.gz`

Naredbom `crf.train(boolean usePunctuation)` započinjemo treniranje klasifikatora sa zadanim postavkama. Zatim, naredbom `crf.serializeClassifier (SERIALIZE_LOCATION)` pohranjujemo istrenirani klasifikator na zadanu lokaciju.

5.5 Klase `StoryData` i `CharacterName`

Ove dvije klase predstavljaju pomoćne klase za pohranu podataka o samoj priči i likovima. Nalaze se u paketu `model`.

Klasu `StoryData` čine varijable `characters`, odnosno, popis likova prepoznatih u priči, te sam tekst priče `story`. Osim njih, imamo i metode za postavljanje i dohvaćanje njihovih vrijednosti, takozvane *gettere* i *settere*.

```
public class StoryData {

    private List<CharacterName> characters;
    private String story;
```

```
    public List<CharacterName> getCharacters() {
        return characters;
    }

    public void setCharacters(List<CharacterName> characters) {
        this.characters = characters;
    }

    public String getStory() {
        return story;
    }

    public void setStory(String story) {
        this.story = story;
    }
}
```

Klasa `CharacterName` predstavlja nešto složeniji prikaz imena pojedinog lika. Naime, likovi često imaju složeno ime koje se sastoji od više riječi, na primjer *mother goat* u *Vuku i sedam kozlića*. Sasvim je moguće da se autor na takve likove ponekad referira punim imenom, a ponekad skraćenim imenom, na primjer *mother*. S obzirom da bismo htjeli prepoznati navedenog lika u svim situacijama, osim njegovog punog imena (`fullName`), čuvamo i skup samih riječi (`tokens`):

```
public class CharacterName {

    private String fullName;

    private List<String> tokens;

    public CharacterName(String fullName, List<String> tokens) {
        this.fullName = fullName;
        this.tokens = tokens;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }
```

```
    public List<String> getTokens() {
        return tokens;
    }

    public void setTokens(List<String> tokens) {
        this.tokens = tokens;
    }

    @Override
    public String toString() {
        return "Full name: " + fullName + ", tokens: " +
            tokens.toString();
    }
}
```

Metoda `toString()` omogućuje jasniji ispis objekata ove klase.

5.6 Klasa `Solver`

Klasa `Solver` čini srž naše aplikacije. U njoj se provodi analiza čitave priče, odnosno, tri prethodno opisana koraka.

Jedina metoda koju ova klasa sadrži je `solveForStory(String path, boolean usePunctuation)`, koja prima lokaciju priče koju želimo analizirati i podatak o tome koji NER model koristimo. Priča treba biti u tekstualnoj (`.txt`) datoteci, te bi trebala sadržavati isključivo tekst same priče, bez naslova, bilješki, komentara i sličnog dodatnog teksta.

```
public class Solver {

    private CharacterRecognition characterRecognition = new
        CharacterRecognition();

    private CoreferenceResolution coreferenceResolution = new
        CoreferenceResolution();

    private SentimentAnalysis sentimentAnalysis = new
        SentimentAnalysis();
```

```
    public void solveForStory(String path, boolean usePunctuation)
        throws IOException {

      StoryData storyData =
          characterRecognition.setUpStoryAndCharacters(path,
          usePunctuation);
      List<String> characterTextList =
          coreferenceResolution.recogniseCharacterSentences(storyData);
      List<String> sentiments =
          sentimentAnalysis.calculateCharacterSentiments(
      characterTextList);
      System.out.println("----------------------------------------");
      System.out.println("Results:");
      for (int i = 0; i < sentiments.size(); ++i) {
        System.out.format("%s: %s\n",
            storyData.getCharacters().get(i).getFullName(),
            sentiments.get(i));
      }
      System.out.println("----------------------------------------");
    }
  }
```

Kao što vidimo, prvi korak je postavljanje `storyData` varijable kroz metodu `setUpStoryAndCharacters(String path, boolean useInterpunction)` iz klase `CharacterRecognition`. U toj metodi se ekstrahiraju likovi i tekst priče. Sljedeći korak je prepoznavanje onih rečenica u priči koje se odnose na pojedinog lika. Ovaj postupak je implementiran u metodi `recogniseCharacterSentences(StoryData storyData)` iz klase `CoreferenceResolution`. Rezultat je lista tekstova koji su potom dani, kao ulaz, metodi `calculateCharacterSentiments(List<String> characterSentences)`, koja obavlja analizu sentimenta nad svakim od njih. Ta metoda se nalazi u klasi `SentimentAnalysis`. Sve tri navedene klase se nalaze u zasebnom paketu `phase`.

Napomenimo još samo da se razrješavanje koreferencije i analiza sentimenta ponašaju jednako, neovisno o tome koji NER model koristimo. NER model utječe samo na prvi korak, prepoznavanje samih likova u priči.

5.7 Klasa `CharacterRecognition`

U ovoj klasi implementiramo prepoznavanje likova u priči, odnosno, popunjavanje `StoryData` varijable. Glavna metoda u klasi je `setUpStoryAndCharacters`:

```
public StoryData setUpStoryAndCharacters(String path,
                            boolean modelWithPunctuation)
   throws IOException
{

   // load NER model
   NERClassifierCombiner classifier;
   if (modelWithPunctuation) {
      classifier = new NERClassifierCombiner(false, false, NER_MODEL);
   }
   else {
      classifier = new NERClassifierCombiner(false, false,
          NER_MODEL_NO_PUNCTUATION);
   }
   // read story text
   String storyText = IOUtils.slurpFile(path);

   // tag story with NER tags
   List<List<CoreLabel>> taggedText = classifier.classify(storyText);
   List<CharacterName> characters = new ArrayList<>();

   for (List<CoreLabel> sentence : taggedText) {
      for (int i = 0; i < sentence.size(); ++i) {
         CoreLabel firstLabel = sentence.get(i);
         String tag = firstLabel.ner();
         String name = firstLabel.value();
         if (tag.equals("O")) {
            continue;
         }

         List<String> tokens = new ArrayList<>();
         tokens.add(name.toLowerCase());
         CoreLabel nextLabel = sentence.get(i + 1);

         // read all neighbouring character tokens
         // they all belong to single character
```

```
            while (nextLabel.ner().equals("C")) {
               name += " " + nextLabel.value();
               tokens.add(nextLabel.value().toLowerCase());
               ++i;
               if (i >= sentence.size() - 1) {
                  break;
               }
               nextLabel = sentence.get(i + 1);
            }
            updateCharacterList(name, tokens, characters);
         }
      }

      StoryData storyData = new StoryData();
      storyData.setCharacters(characters);
      storyData.setStory(storyText);
      logger.info("File '{}' successfully read.", path);
      logger.info("Found characters:\n{}",
          buildNiceCharactersString(characters));
      return storyData;
   }
```

Dakle, prvi korak je učitavanje NER modela i njegovo pokretanje nad tekstom. Rezultat je lista rečenica, od kojih se svaka sastoji od liste `CoreLabel` objekata. `CoreLabel` klasa dolazi iz *Stanford CoreNLP* biblioteke, te označava jedan obrađeni token. Potom, iteriramo po označenim rečenicama i unutar njih tražimo tokene označene oznakom `C`. Kada naiđemo na takav token, varijabli `name` pridružujemo njezinu vrijednost. Također, dodajemo vrijednost tokena u skup `tokens`. Nakon toga provjeravamo je li sljedeći token, također, označen tom klasom. Ukoliko jest, proširujemo ime lika novim tokenom, te dodajemo novi token u skup. Postupak ponavljamo dokle god je oznaka jednaka `C`. S obzirom da većina likova ima imena od jedne ili dvije riječi, očekujemo jednako toliko ponavljanja.

Na kraju iteracije imamo puno ime pronađenog lika, kao i potpunu listu tokena. S obzirom da se imena likova ponavljaju u tekstu, te da se likovi mogu oslovljavati punim imenom, kao i skraćenim, potrebno je provjeriti možemo li dodati novog lika u popis likova ili se pak radi o postojećem liku. Za svakog lika provjeravamo sadrži li novodobivena lista tokena sve njegove tokene. Ako sadrži, nadopunjujemo ime postojećeg lika. Ako ne sadrži, provjeravamo obratnu situaciju, odnosno, sadrže li postojeći tokeni sve nove tokene. U tom slučaju, pronašli smo samo dio imena nekog lika i vraćamo se u glavnu metodu bez ikakvih promjena nad listom likova. U metodi su u komentarima dani primjeri iz kojih je jasnije.

Konačno, u trećem slučaju, kada su skupovi tokena svih dosad prepoznatih likova posve disjunktni s pronađenim tokenima, očito, imamo posve novog, nepoznatog lika kojeg je potrebno dodati u listu.

Gore opisan postupak je implementiran u zasebnoj metodi:

```
private void updateCharacterList(String name,
                                 List<String> tokens,
                                 List<CharacterName> characters) {
   for (CharacterName character : characters) {
      // already in list: Red Riding Hood
      // now found: Little Red Riding Hood
      // update character name
      if (tokens.containsAll(character.getTokens())) {
         character.setFullName(name);
         character.setTokens(tokens);
         return;
      }
      // opposite:
      // already in list: Little Red Riding Hood
      // now found: Red Riding Hood
      // skip
      if (character.getTokens().containsAll(tokens)) {
         return;
      }
   }
   // other situation - completely new character
   CharacterName newCharacter = new CharacterName(name, tokens);
   characters.add(newCharacter);
}
```

Na samom kraju metode `setUpStoryData` postavljamo varijablu `storyData`, ispisujemo poruku o uspješno završenom postupku, kao i popis prepoznatih likova. Metoda `buildNiceCharactersString` za cilj ima generirati pregledan zapis liste likova.

5.8 Klasa `CoreferenceResolution`

U ovoj klasi je implementiran sljedeći korak, razrješavanje koreferencije, odnosno, prepoznavanje onih rečenica u priči koje pripadaju pojedinom liku. Glavna metoda je `recogniseCharacterSentences`:

```
public List<String> recogniseCharacterSentences(StoryData storyData) {

   // set up annotator
   StanfordCoreNLP pipeline = setUpCoreferencePipeline();
   Annotation document = pipeline.process(storyData.getStory());

   int numberOfCharacters = storyData.getCharacters().size();
   List<String> characterTextList =
       initializeAllCharacterTexts(numberOfCharacters);
   List<CoreMap> sentences = document.get(SentencesAnnotation.class);

   for (CorefChain cc :
       document.get(CorefChainAnnotation.class).values()) {
      List<CorefChain.CorefMention> mentions =
          cc.getMentionsInTextualOrder();
      logger.info("Mentions in current chain: {}", mentions);

      int characterIndex =
          checkMentionChainForAnyCharacter(storyData, mentions);
      if (characterIndex == -1) {
         // character not found in current mention chain
         continue;
      }
      updateCharacterTextList(characterIndex, characterTextList,
          sentences, mentions);
   }
   return characterTextList;
}
```

U metodi prvo postavljamo objekt klase `StanfordCoreNLP`, kroz kojega ćemo pozvati razrješavanje koreferencije, i to kroz sljedeću metodu:

```
private StanfordCoreNLP setUpCoreferencePipeline() {
   Properties props = new Properties();
   props.setProperty("annotators", "tokenize, ssplit, pos, lemma,
       ner, parse, mention, dcoref");
   props.setProperty("dcoref.use.big.gender.number", "false");
```

```
    props.setProperty("dcoref.animate", ANIMATE_LIST);
    return new StanfordCoreNLP(props);
}
```

Postavljamo preporučene postavke, osim za popis živih entiteta, gdje zadajemo vlastitu listu, koju čine lista i živih i neživih entiteta. Bez te promjene, sustav za razrješavanje koreferencije ne provodi taj postupak za nežive entitete, poput mački, ogledala, patuljaka. Drugim riječima, u sljedećem primjeru, sustav ne bi prepoznao da se *she* odnosi na *cat*: *"Cat is on the mat. She is sleeping."*

U metodi `initializeAllCharacterTexts`, za početak, inicijaliziramo tekstove svih likova na prazne nizove.

Sljedeći korak je obrada teksta, odnosno, provođenje samog postupka razrješavanja koreferencije nad pričom. Time smo dobili listu rečenica `sentences`, ali i listu svih `CorefChain` objekata, koje čini niz prepoznatih međusobno koreferentnih izraza u tekstu. Jedan primjer tog objekta je:

```
[a huntsman" in sentence 15, "The huntsman" in sentence 17, "he" in
    sentence 17, "his" in sentence 17, "my" in sentence 17, "I" in
    sentence 17, "the huntsman" in sentence 18, "he" in sentence 18,
    "the huntsman" in sentence 49, "the huntsman" in sentence 57]
```

Iteriramo kroz sve `CorefChain` objekte. U svakoj iteraciji provjeravamo spominje li se neki od likova u tom lancu koreferentnih izraza. Ako se spominje, dohvaćamo njegov indeks:

```
private int checkMentionChainForAnyCharacter(StoryData storyData,
                                        List<CorefMention> mentions) {

    int numberOfCharacters = storyData.getCharacters().size();
    for (CorefMention mention : mentions) {
       for (int i = 0; i < numberOfCharacters; ++i) {
          for (String characterToken :
              storyData.getCharacters().get(i).getTokens()) {

             // get all words in single mention
             // example of mention:
             // "dwarves, which were living in the mountains"
             String[] mentionWords =
                 mention.mentionSpan.split("[\\.\\'\\,\\ ]+");
             for (String mentionWord : mentionWords) {
                if (equalsIgnoreCase(mentionWord, characterToken)) {
```

```
                    logger.info("Found character {} in mention '{}',
                        mention chain is '{}'",
                    storyData.getCharacters().get(i).getFullName(),
                    mention.mentionSpan,
                    mentions);
                    return i;
                }
            }
        }
    }
    return -1;
}
```

Dakle, ako je indeks različit od -1, nadopunjemo tekst koji se odnosi na tog lika:

```
private void updateCharacterTextList(int characterIndex,
                                     List<String> characterTextList,
                                     List<CoreMap> sentences,
                                     List<CorefMention> mentions) {

    for (CorefMention mention : mentions) {
        // check if character is subject in that sentence
        CoreMap coreMap = sentences.get(mention.sentNum - 1);
        String characterText = characterTextList.get(characterIndex);

        // there can be multiple mentions in same sentence
        // if that case, such sentences would be added multiple times,
            once for every mention
        if (characterText.contains(coreMap.toString())) {
            return;
        }

        SemanticGraph semanticGraph =
            coreMap.get(CollapsedCCProcessedDependenciesAnnotation.class);
        IndexedWord verb = semanticGraph.getFirstRoot();
        IndexedWord subject = semanticGraph.getChildWithReln(verb,
        GrammaticalRelation.valueOf(Language.UniversalEnglish,
            "nsubj"));
```

```
            // if subject is not found or if character is subject
            // add that sentence to text
            // very often, characters are both subjects and objects
            // we want only subjects - bad characters often do something
            // bad to another character
            // both bad and good characters would get that same bad sentence
            if (subject == null ||
                  containsIgnoreCase(mention.mentionSpan, subject.word())) {
               String updatedText = characterText + " " + coreMap;
               characterTextList.set(characterIndex, updatedText);
            }
         }
      }
```

U gornjoj metodi iteriramo po koreferentnim izrazima u lancu. U prvom koraku provjeravamo nalazi li se već u tekstu rečenica u kojoj se nalazi koreferentni izraz. Ako je odgovor potvrdan, nije ju potrebno dodavati, s obzirom da ne želimo da ista rečenica više puta sudjeluje u analizi sentimenta, već samo jednom.

U sljedećem koraku dohvaćamo subjekt dane rečenice i provjeravamo je li subjekt dio koreferentnog izraza. Ovaj korak je dodan kao neplanirana optimizacija, s obzirom se u istoj rečenici mogu spominjati dva lika, jedan kao subjekt, a drugi kao objekt. U tom slučaju, najčešće, želimo tu rečenicu pridružiti liku koji je subjekt, a ne objekt. Naime, često zločesti likovi, kao subjekti u rečenici, vrše neku negativnu radnju (na primjer, vrijeđanje) nad dobrim likom, koji je objekt. Bez prepoznavanja subjekta, negativnu rečenicu bismo pridružili i pozitivnom i negativnom liku. Ipak, ponekad analiza rečenice ne uspije pronaći subjekt. U toj situaciji rečenicu ipak pridružujemo liku.

5.9 Klasa `SentimentAnalysis`

Ova klasa čini zadnji korak analize priče, odnosno, čitavog algoritma. Temeljna metoda u njoj je `calculateCharacterSentiments`, koja, za sve likove, izračunava pripadni sentiment:

```
   public List<String> calculateCharacterSentiments(List<String>
       characterTextList) {

      StanfordCoreNLP pipeline = setUpSentimentPipeline();
      logger.info("Calculating sentiment for each character...");
      List<String> sentiments = new ArrayList<>();
```

```
    for (String text : characterTextList) {
        String sentiment = calculateSentiment(pipeline, text);
        sentiments.add(sentiment);
    }
    return sentiments;
}
```

Slično kao i u klasi `CoreferenceResolution`, prvi korak je postavljanje `Stanford CoreNLP` objekta, koji će provesti analizu sentimenta:

```
private StanfordCoreNLP setUpSentimentPipeline() {
    Properties sentProperties = new Properties();
    sentProperties.setProperty("annotators", "tokenize, ssplit, pos,
        parse, sentiment");
    return new StanfordCoreNLP(sentProperties);
}
```

U sljedećem koraku iteriramo po tekstovima pojedinih likova i provodimo analizu sentimenta nad njima. U metodi `calculateSentiment`, za dani tekst, računamo sentiment:

```
private String calculateSentiment(StanfordCoreNLP pipeline,
                                  String text) {

    if (text.length() == 0) {
        logger.info("Text is empty - skipping sentiment analysis.");
        return "insufficient data for sentiment analysis";
    }

    logger.info("Received text for sentiment analysis:\n{}",
        text.toString());

    double sum = 0;
    int count = 0;

    // predicted class can be one of:
    // 0 = very negative
    // 1 = negative
    // 2 = neutral
    // 3 = positive
    // 4 = very positive
```

```
        Annotation document = pipeline.process(text);

        for (CoreMap sentence :
            document.get(CoreAnnotations.SentencesAnnotation.class)) {
           Tree tree =
               sentence.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);
           int sentiment = RNNCoreAnnotations.getPredictedClass(tree);
           if (sentiment >= 0) {
              sum += sentiment;
              ++count;
           }
        }

        // return average of sentence sentiments
        double value = sum / count;
        logger.info("Sentiment numeric value: {}", value);

        // usually, neutral characters are considered good
        if (value <= 1.5)
           return "bad";
        else
           return "good";
    }
```

Uočimo da smo odabrali karakterizirati likove isključivo kao dobre ili loše. Razlog tome je što djeca često promatraju svijet crno-bijelo. Posebno, u dječjim lektirama, u nižim razredima, gotovo uvijek se na taj način karakteriziraju likovi. Nikad nije pitanje tko je u priči neutralan lik, već tko su dobri, a tko loši likovi.

Nadalje, Stanfordov sustav dodjeljuje sljedeće ocjene rečenicama:

- 0 — izrazito negativna rečenica,
- 1 — negativna rečenica,
- 2 — neutralna rečenica,
- 3 — pozitivna rečenica,
- 4 — izrazito pozitivna rečenica.

S obzirom na to, prag od 1.5 za lošeg lika znači da samo isključivo negativno ocijenjene likove smatramo zlima. S druge strane, neutralni likovi će često u dječjim pričama biti proglašeni dobrima. Razlog tome je crno-bijelo viđenje svijeta. Svatko, tko nije izrazito zločest, je dobar.

5.10 Klasa Constants

Ova klasa se nalazi u paketu `util` i sadržava tekstualne konstante:

```
public class Constants {

    public static final String TRAINING_PROPERTIES =
        "src/main/resources/stanford/ner-training.properties";
    public static final String NER_MODEL =
        "src/main/resources/stanford/ner-model.ser.gz";
    public static final String NER_MODEL_NO_PUNCTUATION =
        "src/main/resources/stanford/ner-model-no-punctuation.ser.gz";
    public static final String STORIES = "src/main/resources/stories/";
    public static final String ANIMATE_LIST =
        "src/main/resources/stanford/coref-animate.txt";
    public static final String TXT = ".txt";
    public static final String TRAINING_DATA =
        "src/test/resources/stanford/ner-training-data.tsv";
    public static final String TRAINING_DATA_NO_PUNCTUATION =
        "src/main/resources/stanford/ner-training-data-no-punctuation.tsv";
    public static final String TRAIN_FILE = "trainFile";

    private Constants() {}
}
```

Svrha izdvajanja konstanti u zasebnu klasu je veća preglednost koda te izbjegavanje definiranja iste konstante u različitim klasama.

Poglavlje 6

Rezultati

Aplikaciju smo testirali nad desetak dječjih priča. Za svaku od priča ćemo dati kratki sažetak s opisom likova. Nadalje, zapisat ćemo i komentirati dobivene rezultate.

Napomenimo samo da su neke od bajki dane u svom izvornom zapisu, koji je ponešto drukčiji od modernih, nježnijih verzija tih bajki.

6.1 Pepeljuga

Ova priča predstavlja varijantu poznate priče o Pepeljugi.

Sažetak

Pepeljuga je djevojka, čija je majka, dobra i pobožna žena, umrla. Pepeljuga je majci na samrti obećala da će biti dobra i pobožna. Otac se nakon nekog vremena ponovno oženio, i kako to obično biva u bajkama, maćeha je bila zla žena. Imala je dvije prelijepe, ali zločeste kćeri. Jednog dana je otac išao na sajam. Kad je pitao djevojke što žele da im donese, bahate kćeri su rekle da žele skupu odjeću, dok je Pepeljuga htjela prvu granu koja dodirne lice njenog oca dok bude putovao. Otac je tako i uradio. Kad se vratio, Pepeljuga je na majčin grob zarila tu granu, nad kojom je često plakala i koja je ubrzo izrasla u prekrasno stablo. Svaki put kad bi Pepeljuga plakala pred stablom, pojavila bi se bijela golubica i ispunila bilo koju njezinu želju.

Ubrzo je kralj odlučio organizirati trodnevni festival za sav narod, s ciljem da njegov sin nađe ženu. Pepeljugi je maćeha zabranila da ide na bal, jer je imala kućanskih poslova za obaviti, te nije imala lijepe odjeće. Pepeljuga je zazvala ptice da joj pomognu s kućanskim poslovima, te je plakala na majčinom grobu i

zazivala stablo da joj donese haljinu, što se i dogodilo. Pepeljuga je bila najljepša na balu. Princ je isključivo s njom plesao i nije htio da itko drugi pleše s njom. Po završetku bala, princ ju je htio otpratiti kući, ali je Pepeljuga odlučila pobjeći od srama. Sljedeći dan se sve ponovilo — Pepeljuga je zazvala još ljepšu haljinu, plesala s princem i pobjegla. Isto se dogodilo i posljednjeg dana, ali princ je odlučio stepenice premazati smolom, tako da se Pepeljugina cipela pri bijegu zalijepila.

Sljedećeg dana je princ otišao k dvjema sestrama da provjeri pripada li cipela nekoj od njih dvije. Starijoj kćeri je cipela bila prevelika, pa je poslušala majčin savjet i odrezala prst, te obukla cipelu. Princ je nasjeo na prevaru, te je krenuo sa starijom sestrom prema dvorcu. Putem je uočio da joj stopalo krvari, te se vratio kući i zahtijevao da mlađa kći proba cipelu. Mlađoj kćeri, s druge strane, peta nije mogla stati u cipelu, te si je ona odrezala petu. Ponovno se dogodila ista situacija — krenuli su prema dvorcu, ali je princ vidio da krvari i da je prevaren. Naposljetku je Pepeljuga probala cipelu te je princ shvatio da je ona njegova mladenka.

Na dan vjenčanja, dvije sestre su hodale uz kraljevski par. Za kaznu njihovoj bahatosti i zlobi, golubovi su ih napali i oslijepili.

Likovi

Možemo zaključiti da je u ovoj priči Pepeljuga dobar lik, kao i njezina pokojna majka, te princ i otac. S druge strane, maćeha i njezine kćeri su zli likovi.

Dobiveni rezultati uz upotrebu NER modela s interpunkcijskim znakovima

```
[main] INFO step.CharacterRecognition - Found characters:
princess
sisters
bird
King
Aschenputtel
mother
step-mother
prince
father
queen
```

Očito je da sustav nije posve uspješno prepoznao likove. Princeza se odnosi na Pepeljugu na balu, kada su smatrali da je ona nepoznata princeza daleke zemlje.

Ptica, odnosno, bijela golubica je, također, prepoznata kao lik. To možda i nije posve pogrešno, s obzirom da je ptica imala dosta veliku ulogu u priči — pomagala je Pepeljugi s kućanskim poslovima, ostvarivala želje. Nadalje, kraljica se u priči spominje u dvije rečenice, a to su savjeti maćehe kćerima: *Kada budeš kraljica, nećeš nikad morati pješačiti*. Vjerojatan razlog za prepoznavanje kraljice je taj što je NER sustav treniran nad većim brojem priča u kojima su kraljice bile likovi. Konačno, majka se ipak odnosi na maćehu, a ne na pokojnu majku.

```
Results:
princess: good
sisters: bad
bird: bad
King: bad
Aschenputtel: bad
mother: bad
step-mother: insufficient data for sentiment analysis
prince: good
father: bad
queen: insufficient data for sentiment analysis
```

Sustav je ispravno prepoznao sestre kao zločeste likove, kao i majku, odnosno, maćehu. Također, kralj, princ i princeza (Pepeljuga na balu) su okarakterizirani kao dobri likovi.

S druge strane, za pticu i samu Pepeljugu je sustav zaključio da su zle. Proučavanjem ispisa u konzoli uočavamo da se na Pepeljugu odnose sljedeće rečenice:

```
[main] INFO step.SentimentAnalysis -
Sentence: Then the two sisters were very glad, because they had
pretty feet.
Sentiment: 2

[main] INFO step.SentimentAnalysis -
Sentence: Besides that, the sisters did their utmost to torment
her,--mocking her, and strewing peas and lentils among the ashes,
and setting her to pick them up.
Sentiment: 1

[main] INFO step.SentimentAnalysis -
Sentence: The step-mother and the two sisters were thunderstruck,
and grew pale with anger; but he put Aschenputtel before him on
```

```
his horse and rode off.
Sentiment: 1

[main] INFO step.SentimentAnalysis -
Sentence: And as they passed the hazel bush, the two white pigeons
cried, "There they go, there they go!
Sentiment: 1

[main] INFO step.SentimentAnalysis -
Sentence: And when her wedding with the prince was appointed to
be held the false sisters came, hoping to curry favour, and to
take part in the festivities.
Sentiment: 1

[main] INFO step.SentimentAnalysis -
Sentence: Of Aschenputtel they never thought at all, and supposed
that she was sitting at home, and picking the lentils out of the
ashes.
Sentiment: 1

[main] INFO step.SentimentAnalysis - Final sentiment numeric value:
1.1666666666666667
```

Zaključujemo da je jedan od problema sustava taj što nije ispravno dodijelio rečenice likovima. Gotovo sve prethodne rečenice se odnose na Pepeljugu, ali opisuju postupke zlobnih sestara prema njoj. Dakle, bilo bi ispravno kad bi te rečenice bile dodijeljene sestrama, a ne Pepeljugi.

Nadalje, problem predstavlja i pogrešno zaključivanje sustava za analizu sentimenta. Na primjer, tužne rečenice su uglavnom ocijenjene negativno. U dječjim pričama to može dovesti do greške, s obzirom da takve rečenice zapravo definiraju dobre likove.

Konačno, za oca je utvrđeno da je zao:

```
[main] INFO step.SentimentAnalysis -
Sentence: The father thought to himself,
"It cannot surely be Aschenputtel," and called for axes and
hatchets, and had the pigeon-house cut down, but there was no
one in it.
Sentiment: 1
```

```
[main] INFO step.SentimentAnalysis -
Sentence: "The first twig, father, that strikes against your
hat on the way home; that is what I should like you to bring me."
Sentiment: 1

[main] INFO step.SentimentAnalysis - Final sentiment numeric value:
1.0
```

Dobiveni rezultati uz upotrebu NER modela bez interpunkcijskih znakova

Sustav je u ovom slučaju prepoznao sljedeće likove:

```
[main] INFO phase.CharacterRecognition - Found characters:
princess
sisters
mother
King
Aschenputtel
bird
prince
father
queen
wife
step-mother
```

Uočavamo da je ovaj model prepoznao dodatno i suprugu (*wife*). U tekstu se pod tom riječi referira i na Pepeljuginu pokojnu majku, kao i na njezinu maćehu.

S obzirom da su svi sljedeći koraci jednaki, jedina razlika u konačnim rezultatima je ocjena za novog lika. Supruga je karakterizirana kao dobar lik:

```
[main] INFO phase.SentimentAnalysis -
Sentence: When the
winter came the snow covered the grave with a white covering, and
when the sun came in the early spring and melted it away, the man
took to himself another wife.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
```

```
Sentence: "No," said the man, "only my dead wife left behind her a
little stunted Aschenputtel; it is impossible that she can be the
bride."
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: The new wife brought two daughters home with her, and they
were beautiful and fair in appearance, but at heart were black and
ugly.
Sentiment: 3

[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
1.6666666666666667
```

Vidimo da se druga po redu rečenica odnosi na pokojnu suprugu. Također, uočavamo da je posljednja rečenica ocijenjena pozitivno, što predstavlja priličnu grešku sustava.

6.2 Snjeguljica

Priča o Snjeguljici i sedam patuljaka je svima poznata. Ipak, prisjetimo se.

Sažetak

Kraljica je imala kćer čija koža je bila bijela poput snijega, usne crvene poput krvi, a kosa crna poput ugljena. Nažalost, ubrzo nakon što se Snjeguljica rodila, kraljica je umrla. Kralj je godinu dana tugovao, a potom se ponovno oženio. Nova kraljica je bila prekrasna, ali ohola i zla. Svakog dana se gledala u magično ogledalo i pitala ga tko je najljepša osoba na svijetu. Dok je Snjeguljica bila jako mlada, ogledalo je odgovaralo kraljici da je ona najljepša osoba na svijetu. Ipak, kada je Snjeguljica porasla, ogledalo je počelo govoriti da je kraljica prekrasna, ali je ipak Snjeguljica ljepša.

To je u kraljici izazvalo strašnu ljubomoru, te je naredila da joj dovedu lovca. Lovcu je zapovijedila da odvede Snjeguljicu u šumu i ubije ju. Lovac je isprva pristao, ali kad ju je trebao ubiti, Snjeguljica je zaplakala i zamolila ga da ju poštedi. Lovac se smilovao zbog njezine ljepote, ali je smatrao da će ju ubiti zvijeri u šumi.

Kraljica je povjerovala lovcu da je ubio Snjeguljicu, te je bila mirna neko vrijeme. U međuvremenu, Snjeguljica je lutala kroz šumu i naišla na kuću u kojoj su živjeli

patuljci. Ondje je ručala njihovu hranu te zaspala u njihovom krevetu. Patuljci su dopustili da Snjeguljica ostane živjeti s njima. Preko dana su patuljci odlazili u planine rudariti, dok je ona ostajala sama kod kuće. Upozorili su je da ne otvara vrata strancima, jer bi to mogla biti zla kraljica.

Kraljica je nakon nekog vremena saznala da ju je lovac prevario. Odlučila je potražiti Snjeguljicu i uspjela ju je pronaći u kući patuljaka. Više puta se prerušavala s ciljem da prevari Snjeguljicu i usmrti ju. Posljednji put je uspjela, otrovnom jabukom. Kad su patuljci došli kući, zatekli su mrtvu Snjeguljicu. Tužni i slomljeni, okupali su je, počešljali i polegli u stakleni lijes, te odnijeli na planinu.

Jednog dana je princ prolazio šumom, uočio kuću patuljaka i stakleni lijes na planini. Kad se približio lijesu, zaljubio se u Snjeguljicu i htio ju je odvesti kući u lijesu. Dok su njegovi sluge nosili lijes, spotaknuli su se, pa je Snjeguljica iskašljala komad jabuke koji joj je zapeo u grlu.

Princ i Snjeguljica su se zaručili, a kraljici je tada ogledalo poručilo da je mladenka ipak ljepša od nje. Kraljica je sumnjala da je to Snjeguljica, te je morala otići na vjenčanje. Ondje su je uhvatili u bijesu i mržnji, te kaznili plesom u užarenim željeznim cipelama.

Likovi

Prepoznajemo Snjeguljicu, patuljke i princa kao dobre likove, te kraljicu i lovca kao zle. Lovca smatramo zlim likom jer je prvo pristao ubiti Snjeguljicu, a potom joj se "smilovao" na način da ju je pustio da umre u šumi.

Dobiveni rezultati uz upotrebu NER modela s interpunkcijskim znakovima

Sustav je prilično točno prepoznao likove. Nažalost, nije prepoznao princa, u tekstu *king's son*, već je prepoznao kralja. Ne možemo biti sigurni je li prepoznat kralj — Snjeguljičin otac, ili kralj — princev otac. Nadalje, starica je prerušena kraljica, koju je sustav prepoznao zasebno, jer se javlja pod tim imenom u tekstu. Također je ogledalo prepoznato kao lik. To možda i nije pogrešno, s obzirom da kraljica razgovara s ogledalom. Ipak, ogledalo se u tekstu češće spominje kao *looking-glass* ili samo *glass*, a ti izrazi nisu prepoznati.

```
[main] INFO step.CharacterRecognition - Found characters:
Queen
king
Snow-white
```

```
mirror
huntsman
dwarfs
dwarf
old woman
```

Sustav je donio sljedeće zaključke za pojedine likove:

```
Results:
Queen: bad
king: good
Snow-white: bad
mirror: good
huntsman: bad
dwarfs: bad
dwarf: good
old woman: good
```

Rečenice koje se odnose na kraljicu su sljedeće:

```
[main] INFO phase.SentimentAnalysis -
Sentence: And as she worked, gazing at times out on
the snow, she pricked her finger, and there fell from it three
drops of blood on the snow.
Sentiment: 0

[main] INFO phase.SentimentAnalysis -
Sentence: And when she was born the queen died.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: So one day when the queen went to her mirror and said,

"Looking-glass upon the wall,
Who is fairest of us all?"
Sentiment: 2

[main] INFO phase.SentimentAnalysis -
Sentence: When she heard that she was so struck with surprise that
all the blood left her heart, for she knew that Snow-white must
```

```
still be living.
Sentiment: 2

[main] INFO phase.SentimentAnalysis -
Sentence: The looking-glass answered,

"O Queen, although you are of beauty rare,
The young bride is a thousand times more fair."
Sentiment: 1

[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
1.2
```

U priči se prve dvije rečenice zapravo odnose na prvu kraljicu, Snjeguljičinu majku, dok se preostale odnose na zlu kraljicu. Uočimo, da je sustav ispravno prepoznao samo rečenice koje se odnose na zlu kraljicu, ona bi bila karakterizirana kao dobar lik, s ocjenom 2.5.

Sljedeće rečenice se odnose na "kralja":

```
[main] INFO phase.SentimentAnalysis -
Sentence: After a year had gone by the king took another wife, a
beautiful woman, but proud and overbearing, and she could not bear
to be surpassed in beauty by any one.
Sentiment: 3

[main] INFO phase.SentimentAnalysis -
Sentence: When he so spoke the good little dwarfs had pity upon
him and gave him the coffin, and the king's son called his servants
and bid them carry it away on their shoulders.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: He saw on the mountain the coffin, and beautiful
Snow-white within it, and he read what was written in golden
letters upon it.
Sentiment: 3

[main] INFO phase.SentimentAnalysis -
Sentence: The king's son answered, full of joy,
"You are near me," and, relating all that had happened, he said,
```

```
"I would rather have you than anything in the world; come with me
to my father's castle and you shall be my bride."
Sentiment: 3

[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
2.5
```

Uočimo, zapravo, da se sve osim prve rečenice odnose na kraljevog sina. U posljednje dvije rečenice je sentiment pozitivan (ocjena 3, dok je 4 najviša ocjena). S druge strane, prva rečenica se više odnosi na zlu kraljicu nego na Snjeguljičinog oca.

Analizirajmo i neočekivanu odluku o Snjeguljici:

```
[main] INFO phase.SentimentAnalysis -
Sentence: Poor Snow-white, thinking no harm, let the old woman do as
she would, but no sooner was the comb put in her hair than the
poison began to work, and the poor girl fell down senseless.
Sentiment: 0

[main] INFO phase.SentimentAnalysis -
Sentence: The dwarfs, when they came home in the evening, found
Snow-white lying on the ground, and there came no breath out of her
mouth, and she was dead.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: It answered, "Queen, you are full fair, 'tis true,
But Snow-white fairer is than you."
Sentiment: 3

[main] INFO phase.SentimentAnalysis -
Sentence: And it was salted and cooked, and the wicked woman ate it
up, thinking that there was an end of Snow-white.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: Just at that moment a young wild boar came running by,
so he caught and killed it, and taking out its heart, he brought
it to the queen for a token.
```

```
Sentiment: 3

[main] INFO phase.SentimentAnalysis -
Sentence: Now, for a long while Snow-white lay in the coffin and
never changed, but looked as if she were asleep, for she was still
as white as snow, as red as blood, and her hair was as black as
ebony.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: Then he told the others, who came running up, crying out
in their astonishment, and holding up their seven little candles to
throw a light upon Snow-white.
Sentiment: 2

[main] INFO phase.SentimentAnalysis -
Sentence: He saw on the mountain the coffin, and beautiful
Snow-white within it, and he read what was written in golden
letters upon it.
Sentiment: 3

[main] INFO phase.SentimentAnalysis -
Sentence: In the morning the dwarfs went to the mountain to dig for
gold; in the evening they came home, and their supper had to be ready
for them.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: But Snow-white's wicked step-mother was also bidden to
the feast, and when she had dressed herself in beautiful clothes she
went to her looking-glass and said,
"Looking-glass upon the wall,
Who is fairest of us all?"
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: Snow-white, suspecting nothing, stood up before her, and
let her lace her with the new lace; but the old woman laced so
quick and tight that it took Snow-white's breath away, and she fell
```

```
down as dead.
Sentiment: 0

[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
1.4545454545454546
```

Uočavamo da se jedan dio rečenica ne odnosi izravno na Snjeguljicu, već pretežno na zlu kraljicu, dok je drugi dio rečenica ocijenjen negativno jer opisuju nesreće koje su zadesile Snjeguljicu.

Patuljci su, također, karakterizirani kao loši likovi:

```
[main] INFO phase.SentimentAnalysis -
Sentence: this time the dwarfs will not be able to bring you to life
again."
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: When he so spoke the good little dwarfs had pity upon him
and gave him the coffin, and the king's son called his servants and
bid them carry it away on their shoulders.
Sentiment: 1

[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
1.0
```

Prepoznate su samo dvije rečenice koje se odnose na patuljke. U prvoj je to rečenica koju izgovara kraljica, dok druga rečenica opisuje njihovu tužnu situaciju.

Dobiveni rezultati uz upotrebu NER modela bez interpunkcijskih znakova

Sustav je, koristeći ovaj model, prepoznao sve likove koji su prepoznati korištenjem NER modela treniranog uz interpunkcijske znakove. Stoga nećemo komentirati daljnje rezultate.

6.3 Vuk i sedam kozlića

Sažetak

Majka koza je imala sedam kozlića. Jednog dana je morala ići u šumu po hranu, pa je okupila kozliće kraj sebe i poručila im da se paze zločestog vuka. Upozorila ih je da će se vuk vjerojatno prerušiti, ali da ga mogu prepoznati po grubom glasu i crnim šapama.

I tako je majka otišla u šumu. Nedugo nakon, vuk je kucao na vrata i predstavio se kao majka. Kozlići su ga prepoznali po grubom glasu, te su mu poručili da ode. Vuk je otišao, ali nije odustao. Pošao je u trgovinu kupiti kredu koju je pojeo, od čega mu je glas postao nježniji. Potom se vratio kozlićima.

Kozlići ni ovaj put nisu nasjeli jer su vidjeli njegove crne šape. To su mu i poručili. Vuk je ponovno otišao, ali ovaj put k pekaru i mlinaru. Pekara je zamolio da mu da tijesto da bi od njega oblikovao stopala. Potom je vuk otišao mlinaru i naredio da mu zabijeli šape brašnom, što je ovaj isprva odbio. Vuk mu je potom priprijetio, pa ga je mlinar poslušao.

Vuk se sada treći put vratio pred kozliće. Ovaj put su kozlići nasjeli i otvorili vrata "majci". Kad su shvatili da je to ipak vuk, sakrili su se po kući. Vuk je pronašao i pojeo sve kozliće osim najmlađeg, te je zadovoljan pošao na livadu i zaspao pod stablom.

Kad se majka vratila kući, najmlađi kozlić joj je ispričao što se dogodilo. Majka je u tugi i bijesu krenula prema vuku. Kad ga je vidjela, uočila je da su kozlići još uvijek živi u njegovoj utrobi. Majka je poslala kozlića po škare, konac i iglu, te je razrezala vuka, oslobodila kozliće, i u njegovu utrobu stavila kamenja. Vuk je cijelo to vrijeme čvrsto spavao.

Kad se nakon nekog vremena vuk probudio, bio je žedan zbog kamenja, te je jedva odšetao prema potoku da se napije vode. Kako su kamenja bila teška, povukla su ga u vodu, te se naposlijetku utopio.

Likovi

Dobri likovi su kozlići i majka. Istina, možda majčina osveta nije karakteristika istinski dobrih likova, ali vjerujemo da u ovom slučaju to možemo previdjeti. S druge strane, vuk je očito vrlo zločest lik.

Što se tiče mlinara, za njega ne možemo donijeti neki zaključak. On jest počinio loše djelo, ali iz straha. Ipak, u tekstu se spominje rečenica *And that just shows what men are.*, koja predstavlja autorovo mišljenje o (kukavičkim) ljudima, koje nije baš pozitivno.

Dobiveni rezultati uz upotrebu NER modela s interpunkcijskim znakovima

Sustav je prepoznao majku, vuka i mlinara, ali začudo, nije prepoznao kozliće:

```
[main] INFO phase.CharacterRecognition - Found characters:
mother goat
wolf
miller
```

Također, nije prepoznao pekara, ali to ne trebamo smatrati velikom greškom, jer se pekar javlja u svega par kratkih rečenica.

Za prepoznate likove je donio sljedeće zaključke:

```
Results:
mother goat: good
wolf: bad
miller: bad
```

Možemo se složiti da je sustav ispravno odlučio, ako uzmemo u obzir prethodno rečenu napomenu o mlinaru. Ipak, analizirajmo rečenice koje ga opisuju.

```
[main] INFO phase.SentimentAnalysis -
Sentence: And when the baker had plastered his feet, he ran to the
miller.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: But the miller refused, thinking the wolf must be meaning
harm to some one.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: And the miller was afraid and did as he was told.
Sentiment: 1

[main] INFO phase.SentimentAnalysis -
Sentence: "Miller," said he, "strew me some white meal over my
paws."
Sentiment: 1
```

```
[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
1.0
```

Uočimo da je sustav prvu i posljednju rečenicu označio negativno, iako bi bilo točnije da su označene neutralno. Preostale dvije rečenice su označene ispravno.

Dobiveni rezultati uz upotrebu NER modela bez interpunkcijskih znakova

Ovaj model je prepoznao posebno i majku-kozu i samo majku. Također je prepoznao i pekara:

```
[main] INFO phase.CharacterRecognition - Found characters:
mother goat
wolf
mother
Baker
miller
```

Sustav je karakterizirao likove na sljedeći način:

```
Results:
mother goat: good
wolf: bad
mother: insufficient data for sentiment analysis
Baker: bad
miller: bad
```

Iako je majka prepoznata kao zaseban lik, nije pronađena niti jedna rečenica koja se odnosi na nju.

Nadalje, za pekara je pronađena samo jedna rečenica koja je označena negativno. Stoga je pekar karakteriziran kao loš lik:

```
[main] INFO phase.SentimentAnalysis -
Sentence: And when the baker had plastered his feet, he ran to the
miller.
Sentiment: 1

[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
1.0
```

6.4 Pljačkaš zaručnik

Pljačkaš zaručnik je jedna od manje poznatih priča braće Grimm.

Sažetak

U jednom mjestu je živio mlinar čija kći je bila prekrasna. Kad je odrasla, htio joj je naći dobrog muža. Odlučio je vjenčati ju za udvarača koji bude pristojan i dobar.

Nedugo nakon, naočigled pristojan i uspješan gospodin je posjetio mlinara i tražio ruku njegove kćeri. Otac je pristao, ali kćeri se mladoženja nikako nije svidio, nije mu vjerovala i činio joj se uznemirujuć.

Jednog dana, mladoženja je uspio nagovoriti mladu da ga posjeti u njegovoj kući u šumi. Poručio joj je da će prosuti pepeo da se ne izgubi putem. Ipak, mlada je odlučila putem prosipati grašak. Duboko u šumi je pronašla mračnu, odbojnu kuću. Kad je ušla, ptica ju je pjesmom upozorila da ode, ali je mlada ipak odlučila istražiti kuću. Na tavanu je našla staricu koja ju je upozorila da ju mladoženja i njegovi prijatelji razbojnici planiraju ubiti i skuhati. Potom ju je starica sakrila. Ubrzo su razbojnici stigli kući s nekom drugom djevojkom koju je zatekla opisana sudbina. Starica je otrovala vino razbojnika, pa su čvrsto zaspali nakon večeri. Mlada je u tom trenutku pobjegla kući, prateći grašak na putu, i ispričala sve ocu.

Na dan vjenčanja, mladoženja i njegovi prijatelji su došli kao da se nije ništa dogodilo. Svi uzvanici su trebali ispričati priču, pa je mlada ispričala svoje iskustvo. Na to su uzvanici uhvatili mladoženju i razbojnike i pogubili ih.

Likovi

U priči prepoznajemo mlinara, njegovu kćer, odnosno, mladu, te staricu, mladoženju i razbojnike. Sve likove osim mladoženje i njegovih prijatelja razbojnika smatramo dobrim likovima.

Dobiveni rezultati uz upotrebu NER modela s interpunkcijskim znakovima

Prepoznati likovi su:

```
[main] INFO phase.CharacterRecognition - Found characters:
daughter
old old woman
robbers
```

Sustav, nažalost, nije prepoznao mlinara niti mladoženju kao zasebne likove. Starica je prepoznata kao *old old woman*, jer se u tekstu spominje jednom pod tim imenom, a u ostalim slučajevima pod *old woman*. Sustav je automatski odabrao najdulju prepoznatu pojavu lika kao njegovo stvarno ime.

Dobiveni rezultati su sljedeći:

```
Results:
daughter: bad
old old woman: good
robbers: bad
```

Analizirajmo razloge za negativnu ocjenu kćeri, odnosno, mladenke:

```
[main] INFO phase.SentimentAnalysis -
Sentence: Soon after a suitor came forward who seemed very well to
do, and as the miller knew nothing to his disadvantage, he promised
him his daughter.
Sentiment: 1

[main] INFO phase.SentimentAnalysis - Final sentiment numeric value:
1.0
```

Čini se da je sustav isključivo gornju rečenicu pridružio kćeri. Ostale rečenice uopće nisu prepoznate.

Dobiveni rezultati uz upotrebu NER modela bez interpunkcijskih znakova

Sustav je, koristeći ovaj model, dodatno prepoznao i mlinara — mladenkinog oca:

```
[main] INFO phase.CharacterRecognition - Found characters:
daughter
miller
old old woman
robbers
```

Ipak, za mlinara sustav nije donio nikakav zaključak, jer nije pronađena niti jedna rečenica koja se odnosi na njega:

```
[main] INFO phase.SentimentAnalysis - Text is empty - skipping
sentiment analysis.
```

Poglavlje 7

Zaključak

7.1 Ocjena implementiranog rješenja

U prethodnom poglavlju smo analizirali dobivene rezultate za nekoliko klasičnih bajki. Iako nismo testirali nad velikim brojem priča, možemo donijeti neke zaključke o radu sustava.

U prvoj fazi, prepoznavanju samih likova u priči, uočavamo relativnu točnost sustava. Sustav je uspješno prepoznao većinu likova. Također, uočavamo razliku između dva modela. Model treniran nad podacima koji ne uključuje interpunkcijske znakove je pronašao više likova od modela koji je interpunkcijske znakove uzimao u obzir. Mogući razlog tome je taj što se prije pokretanja postupka prepoznavanja likova iz teksta izbacuju interpunkcijskih znakovi. Stoga je takav "očišćeni" tekst sličniji skupu podataka nad kojima je treniran potonji model.

U drugoj fazi, razrješavanju koreferencije, uočavamo neke probleme. Sustav za razrješavanje koreferencije radi prilično uspješno, ali to samo po sebi često nije dovoljno da bismo ispravno zaključili koje rečenice se odnose na kojeg lika. Kao što smo vidjeli, jedan od problema je referiranje jednog lika na drugoga. U tom slučaju bismo htjeli zadanom liku pridružiti samo one rečenice u kojima se on javlja kao subjekt, ili u kojima je subjekt neki pojam koji se odnosi na tog lika, poput *Snjeguljičine kose*. Ovo predstavlja zaseban problem kojeg nije jednostavno riješiti. Također, parser rečenica nije uspješno prepoznao subjekta u svakoj rečenici, pa smo takve rečenice ipak pridruživali liku. U protivnom bismo za pojedine likove prepoznali premali (ili čak prazan) skup rečenica.

Drugi problem ove faze su složene rečenice. Sustav nije uzimao u obzir surečenice unutar složenih rečenica. Pogledajmo sljedeću rečenicu:

> *Snow-white, suspecting nothing, stood up before her, and let her lace her with the new lace; but the old woman laced so quick and tight that it took Snow-white's breath away, and she fell down as dead.*

Ova rečenica se sastoji od više surečenica, od kojih se neke odnose na Snjeguljicu, a neke na zlu kraljicu. Naše rješenje je čitavu rečenicu pridružilo Snjeguljici, što je očito pogrešno.

Niti posljednja faza, analiza sentimenta, nije bez problema. Uočavamo da ova vrsta analize sentimenta nije posve prikladna za karakterizaciju likova. Naime, dobri likovi su često potlačeni, kažnjavani ili nesretni. Model za analizu sentimenta kojeg smo koristili je takve rečenice pretežno označavao negativnima. To nije neispravno — za sljedeći primjer se možemo složiti da je stav prilično negativan, ali to ipak ne govori ništa o Snjeguljičinom karakteru:

> *The dwarfs, when they came home in the evening, found Snow-white lying on the ground, and there came no breath out of her mouth, and she was dead.*

Također, problem vjerojatno predstavlja i to što je Stanfordov model za analizu sentimenta treniran nad filmskim kritikama, koje su prilično različite od dječjih bajki. Možemo pretpostaviti da je to dijelom razlog za neke neočekivane ocjene rečenica, poput sljedeće rečenice koja je ocijenjena negativno, ocjenom 1:

> *In the morning the dwarfs went to the mountain to dig for gold; in the evening they came home, and their supper had to be ready for them.*

Konačno, unatoč navedenim nedostacima sustava, možemo zaključiti da implementirano rješenje ipak pokazuje određene rezultate. Uočavamo da su svi loši likovi gotovo uvijek i karakterizirani kao takvi. S druge strane, brojni dobri likovi budu pogrešno označeni lošima, iz razloga što im se događaju loše stvari. Ako su dobri likovi sretni i zadovoljni (što ipak u bajkama nije često), vjerojatno će biti proglašeni dobrima.

7.2 Nastavak istraživanja

Za sam kraj ćemo dati nekoliko prijedloga koji bi mogli rezultirati kvalitetnijim sustavom za karakterizaciju likova u dječjim pričama. Ti prijedlozi se mogu podijeliti u dva pristupa.

Prvi pristup predstavlja poboljšanje trenutnog sustava. Kao što smo naveli u prethodnoj sekciji, drugi korak, razrješavanje koreferencije, samo po sebi nije dovoljan za određivanje onih dijelova teksta koji se odnose na lika. Ukoliko bismo u sustav dodali prilično uspješan parser rečenica, koji je u stanju prepoznati zasebne surečenice, te u kojima bi se, potom, ispravno prepoznavali subjekti i objekti, vjerujemo da bi drugi korak rezultirao točnijim skupom informacija o svakome liku. *Stanford CoreNLP* nudi takav parser, iako smo vidjeli da za subjekte ne radi uvijek ispravno.

Sljedeći korak kojeg je potrebno provesti je učenje novog modela za analizu sentimenta, koji bi iz rečenica prepoznavao karakter likova, s obzirom da postojeći nije posve primjenjiv. U tu svrhu je potrebno prikupiti veći broj podataka i ručno ih označiti. Na primjer, rečenice u kojima je lik izrazito nesretan i tužan se trebaju označiti pozitivno, jednako kao i rečenice u kojima je lik sretan, osim ako je sretan zbog tuđe nesreće. S druge strane, ako je lik ljubomoran, pohlepan ili izaziva nekom nesreću, tu rečenicu je potrebno označiti negativno.

Što se tiče prvog koraka, prepoznavanja likova, tu nisu potrebne velike promjene, s obzirom da taj korak radi bez posebno velikih problema.

Drugi pristup bi počeo iznova od samog početka problema. Kao što smo opisali u uvodu, problem karakterizacije likova možemo oblikovati i kao jedinstven problem strojnog učenja. Tada je cilj iskoristiti neku od metoda strojnog učenja nad skupom ručno označenih podataka. Nakon preciznog definiranja problema moguće je prepoznati koje metode strojnog učenja bi nam odgovarale. Također, s obzirom da bi dobiveni model morao objediniti sva tri opisana koraka, očekujemo da bi oblikovanje hipoteze bilo prilično složeno. Moguće je da bi neuralne mreže bile dobar izbor za metodu, s obzirom da se danas koriste u brojnim područjima znanosti i da postoji velika sloboda u njihovom oblikovanju. S druge strane, njihov nedostatak je taj što često zahtijevaju izuzetno velike skupove podataka za učenje.

Ukoliko bismo slijedili ovakav pristup, nužno bi bilo prikupiti velik broj podataka za učenje. Nad tim podacima bismo trebali provesti određene, trenutno nepoznate transformacije. Zasigurno bi jedan od koraka bio uvođenje oznaka u tekst. Primjer takvih oznaka bi bio par *lik, karakter* kojima bi se za određene dijelove teksta ukazalo da se odnose na određenog lika i da označavaju dobro, odnosno, loše djelo.

Bibliografija

[1] Noam Chomsky, *Syntactic Structures*, Mouton Publishers, The Hague / Paris, 1957.

[2] Jonathan H. Clark i José P. González-Brenes, *Coreference resolution: Current trends and future directions*, `https://www.cs.cmu.edu/~jhclark/pubs/clark_gonzalez_coreference.pdf`, 2008., pristupljeno 23. 10. 2016., str. 1–19.

[3] Jenny Rose Finkel, Trond Grenager i Christopher Manning, *Incorporating non-local information into information extraction systems by gibbs sampling*, Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, 2005., str. 363–370.

[4] Seth Grimes, *Expert analysis: Is sentiment analysis an 80% solution?*, `http://www.informationweek.com/software/information-management/expert-analysis-is-sentiment-analysis-an-80--solution/d/d-id/1087919?`, 2010., pristupljeno 23. 10. 2016.

[5] Aria Haghighi i Dan Klein, *Simple coreference resolution with rich syntactic and semantic features*, Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3, Association for Computational Linguistics, 2009., str. 1152–1161.

[6] Daniel Jurafsky i James H. Martin, *Speech and Language Processing (second edition)*, Pearson, 2009.

[7] Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu i Dan Jurafsky, *Deterministic coreference resolution based on entity-centric, precision-ranked rules*, Computational Linguistics **39** (2013), br. 4, 885–916.

[8] Bing Liu, *Sentiment analysis and opinion mining*, Synthesis lectures on human language technologies **5** (2012), br. 1, 1–167.

[9] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard i David McClosky, *The Stanford CoreNLP Natural Language Processing Toolkit.*, ACL (System Demonstrations), 2014, str. 55–60.

[10] Richard Montague, *Universal grammar*, Theoria **36** (1970), br. 3, 373–398.

[11] Maria Ogneva, *How companies can use sentiment analysis to improve their business*, `http://mashable.com/2010/04/19/sentiment-analysis/`, 2010., pristupljeno 23. 10. 2016.

[12] *Project Gutenberg*, `https://www.gutenberg.org/`, 2016., pristupljeno 12. 08. 2016.

[13] Lizhen Qu, Georgiana Ifrim i Gerhard Weikum, *The bag-of-opinions method for review rating prediction from sparse text patterns*, Proceedings of the 23rd International Conference on Computational Linguistics, Association for Computational Linguistics, 2010., str. 913–921.

[14] Stuart C. Shapiro, *Encyclopedia of Artificial Intelligence (second edition)*, A Wiley Interscience Publication, New Jersey, 1992.

[15] Milad Sharif i Hossein Karkeh Abadi, *Recursive Nested Neural Network for Sentiment Analysis*, `https://cs224d.stanford.edu/reports/SharifMilad.pdf`, 2015., pristupljeno 23. 10. 2016., str. 1–7.

[16] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng i Christopher Potts, *Recursive deep models for semantic compositionality over a sentiment treebank*, Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, 2013., str. 1631–1642.

[17] Charles Sutton i Andrew McCallum, *An introduction to conditional random fields for relational learning*, Introduction to Statistical Relational Learning (Lise Getoor i Ben Taskar, ur.), MIT Press, 2007, str. 92–128.

Sažetak

U ovom diplomskom radu rješavali smo problem automatske karakterizacije likova u dječjim pričama. Problem se sastoji od prepoznavanja samih likova u tekstu te, za svakog prepoznatog lika, odlučivanja radi li se o dobrom ili lošem liku.

U prvom dijelu rada obradili smo teorijsku podlogu ovoga problema. Započeli smo s opisom područja obrade prirodnog jezika, koje obuhvaća brojne probleme. Neke od tih problema smo potom prepoznali kao zasebne cjeline unutar problema karakterizacije likova. To su problemi prepoznavanja imenovanih entiteta, razrješavanja koreferencije te analize sentimenta. U nastavku smo, za svaki od ova tri problema, opisali nekoliko modela strojnog učenja kojima se oni rješavaju.

U drugom dijelu diplomskog rada smo implementirali softversko rješenje za problem karakterizacije likova. Kao temelj smo koristili skup alata *Stanford CoreNLP* razvijen na Sveučilištu u Stanfordu. U radu je dana opsežna dokumentacija rješenja, kao i detaljna analiza dobivenih rezultata za nekoliko dječjih priča. Na temelju njih smo donijeli zaključke o prednostima i nedostacima implementiranog rješenja. Naposljetku smo dali nekoliko prijedloga za poboljšanje postojećeg rješenja, kao i za implementaciju novoga.

Summary

In this thesis we solved the problem of automatic resolving of good and bad characters in children's stories. This problem is comprised of recognition of characters in the text, and deciding for each found character whether he/she/it is good or bad.

In the first part of the thesis we considered the theoretical background of this problem. We began by describing the field of natural language processing, which comprises many problems. Some of these problems we recognized as separate components of the character resolving problem. These are the problems of named entity recognition, coreference resolution and sentiment analysis. For each of these three problems, we described several machine learning models which are used to solve them.

In the second part of the thesis we implemented a software solution for the character resolving problem. As a basis we used *Stanford CoreNLP* toolkit developed at Stanford University. Comprehensive documentation for the solution is given in the thesis, as well as detailed analysis of obtained results for several children's stories. Based on them we came to the conclusion about advantages and disadvantages of the implemented solution. Lastly, we gave several recommendations for improvement of the current solution, as well as for implementation of a new solution.

Životopis

Rođena sam 21. rujna 1992. godine u Zadru. Pohađala sam Osnovnu školu Stanovi, nakon čega, 2007. godine, upisujem prirodoslovno-matematički smjer Gimnazije Jurja Barakovića u Zadru. Tijekom srednjoškolskog školavanja sam sudjelovala u natjecanjima iz informatike, robotike, geografije i hrvatskog jezika na županijskoj i državnoj razini.

Po završetku srednjoškolskog obrazovanja, 2011. godine, upisujem preddiplomski sveučilišni studij Matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu. 2014. godine nastavljam studij upisom diplomskog studija Računarstvo i matematika na istom fakultetu.

Za vrijeme diplomskog studija sam sudjelovala u radu udruge Primus koja se primarno bavi popularizacijom znanosti. U sklopu toga sam vodila matematičke i informatičke radionice za djecu i sudjelovala u organizaciji popularno-znanstvenih predavanja.

Također sam tijekom studija radila na poziciji mlađeg softverskog inženjera u tvrtki Corvsu Info d.o.o., te potom na istoj poziciji u tvrtki CROZ d.o.o., gdje trenutno radim.