TakeLab

Laboratorij za analizu teksta i inženjerstvo znanja
Text Analysis and Knowledge Engineering Lab
Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva
Unska 3, 10000 Zagreb, Hrvatska

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1323

Učenje pretraživanja za rješavanje zadataka obrade prirodnoga jezika

Vjeran Crnjak

Zagreb, srpanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 11. ožujka 2016.

Predmet: **Strojno učenje**

DIPLOMSKI ZADATAK br. 1323

Pristupnik: **Vjeran Crnjak (0036467899)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Učenje pretraživanja za rješavanje zadataka obrade prirodnoga jezika**

Opis zadatka:

Mnogi problemi u obradi prirodnoga jezika uključuju predviđanje strukture (npr. sintaktička analiza, ekstrakcija relacija i sl.) ili slijednu obradu kroz više razina (npr. označavanje imenovanih entiteta u kombinaciji s razrješavanjem koreferencije, semantičko parsanje, i sl). Jednostavni pristupi temeljeni na strojnom učenju strukturu izgrađuju na temelju lokalnih klasifikacijskih odluka, a višerazinsku obradu ostvaruju slijedom nezavisnih klasifikatora. Takvi pristupi međutim ne iskorištavaju ovisnosti između zadataka i skloni su propagiranju pogrešaka kroz razine. Alternativu predstavljaju pristupi temeljeni na združenom učenju.
U okviru diplomskoga zadataka potrebno je proučiti pristupe paradigme temeljene na združenome strojnom učenju. Posebnu pažnju posvetiti paradigmi "učenja pretraživanja" (engl. learning to search), kao što su SEARN (Daumé III i dr., 2006) i LOLS (Kai-Wei Chang i dr., 2015), te proučiti radni okvir Vowpal Wabbit. Proučiti tri zadatka obrade prirodnog jezika koji uključuju predviđanje strukture, višerazinsku obradu ili oboje. Razraditi prilagodbu tih zadataka paradigmi učenja pretraživanja. Izgraditi združeni model za svaki od zadataka te provesti iscrpno vrednovanje modela, uključivo analizu pogrešaka te usporedbu s referentnim modelima. Radu priložiti izvorni i izvršni kod razvijenog sustava, skupove podataka i programsku dokumentaciju te citirati korištenu literaturu.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 1. srpnja 2016.

Mentor:

Doc. dr. sc. Jan Šnajder

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Zahvaljujem svojoj supruzi i obitelji na mnoštvu obroka koje su mi pripremili dok sam se ja izležavao pišući ovaj diplomski rad, mentoru na tome što me usmjerio u ovo neiscrpno i zanimljivo područje obrade prirodnoga jezika i strojnog učenja, Mazi što mi je dala malo nevine sreće – neću te nikad zaboraviti.

Sadržaj

Popis slika

Popis tablica

Popis algoritama

1. Uvod

Metode učenja pretraživanja (engl. *learning to search*, L2S) novija su pojava u strojnom učenju. U zadnjih deset godina istraživači su metode primijenili na probleme u različitim granama računarske znanosti, uključujući i obradu prirodnog jezika. Vrlo uspješna primjena vezana je uz probleme gdje se učenje i zaključivanje odvija na primjerima koji sadrže određenu strukturu. Učenje pretraživanja spada u metode strukturnog učenja (engl. *structured learning*) ili predviđanja strukturiranog izlaza (engl. *structured output prediction*). U obradi prirodnog jezika neki od problema koji uključuju predviđanje strukturiranog izlaza su označavanje vrste riječi (engl. *part-of-speech tagging*), ovisnosno parsanje (engl. *dependency parsing*), prepoznavanje imenovanih entiteta (engl. *named entity recognition*), prepoznavanje i praćenje entiteta (engl. *entity detection and tracking*), razrješavanje koreference (engl. *coreference resolution*), izlučivanje relacija između entiteta (engl. *entity relation extraction*), inkrementalno prevođenje (engl. *incremental translation*), inkrementalno odgovaranje na pitanja (engl. *incremental question answering*) i mnoštvo drugih. Ostale primjene metoda učenja pretraživanja su aktivno i interaktivno učenje (engl. *active and interactive learning*), algoritmi grananja i granice (engl. *branch-and-bound*), problemi u robotici, segmentacija slike (engl. *image segmentation*), predviđanje sekundarne strukture proteina (engl. *protein secondary structure prediction*) i dr.

Postojanost razvijenog matematičkog formalizma za te metode jedan je od razloga raznolike primjene. Metode učenja pretraživanja vuku inspiraciju iz područja podržanog učenja (engl. *reinforcement learning*) (Sutton i Barto, 1998), gdje je zadatak sustava naučiti dobru *politiku*. Politika se uči opetovanim izvršavanjem zadatka i ažuriranjem modela s obzirom na pojedinačni ishod. Kod učenja pretraživanja cilj je dodatno iskoristiti prisutne podatke, dok su ti podaci zanemareni ili se postepeno izgrađuju kod primjene podržanog učenja. Matematički aparat koji je iskorišten da bi objasnio podržano učenje i redukcije u strojnom učenju (engl. *machine learning reductions*) omogućava otkriće dobrih teoretskih jamstava metoda učenja pretraživanja. U ovom radu obrađuje se pitanje kako reducirati združeno učenje i predviđanje

(engl. *joint learning and prediction*) na jednostavniji problem. Redukcije u području strojnog učenja slične su redukcijama u drugim područjima računarske znanosti. Naposljetku, glavni je razlog široke primjene vrlo jednostavna implementacija koja iskorištava svojstvo modularnosti koje je prirodno u procesu redukcija (za implementaciju višerazredne klasifikacije potreban je dobar binarni klasifikator). Implementacija je ostvarena u brzom sustavu za strojno učenje zvanom Vowpal Wabbit,[1] a radni okvir L2S omogućava da u vrlo malo linija programskog koda istraživač može napisati algoritam učenja i zaključivanja za specifičan problem. Slični sustavi koji implementiraju algoritme učenja i zaključivanja za vjerojatnosne grafičke modele zahtijevaju puno više linija programskog koda i vjerojatnija je pojava pogrešaka, a nemodularna priroda onemogućuje njihovo izbjegavanje. Pokušaj generalizacije i iskorištavanja programskog koda općenitijim metodama učenja i zaključivanja kod vjerojatnosnih grafičkih modela rezultirao je pojavom vjerojatnosnih programskih jezika (engl. *probabilistic programming languages*), ali brzina učenja i zaključivanja nije ni blizu algoritama L2S.

Ovaj rad nudi sažet pregled metoda učenja pretraživanja i njihovu primjenu na probleme obrade prirodnoga jezika. Uz njihovu matematičku podlogu obrađene su i usporedbe s ostalim metodama strojnog učenja koje se uspješno primjenjuju na istim problemima. Dani su i odgovori na prijašnja postavljena pitanja te opisi konkretne implementacije koji pažljivije argumentiraju prednosti nad ostalim pristupima. U poglavlju 2 dan je pregled glavnih pristupa strukturnom učenju i predviđanju te su opisane njihove prednosti i mane. U poglavlju 3 dan je pregled glavnih metoda učenja pretraživanja u koji je uključen osvrt na redukcije u strojnom učenju te su definirani potrebni pojmovi vezani uz okvir učenja pretraživanja. U poglavlju 4 dan je opis primjena metoda učenja pretraživanja na konkretne probleme u obradi prirodnog jezika. Odabrani problemi služe za razvijanje osjećaja o tome što je moguće raditi u okviru učenja pretraživanja. Dane su i usporedbe s drugim modelima koji su primijenjeni na iste zadatke da bi se istaknula superiornost metoda L2S. U poglavlju 5 dano je vrednovanje modela za zadatke razvijene u ovom radu te opis implementacije algoritama učenja i zaključivanja. Tri različita zadatka: (1) označavanje vrste riječi, (2) ovisnosno parsanje i (3) združeno označavanje i parsanje. Uz tri osnovna zadatka obrađene su njihove varijante i različiti pristupi na istom problemu.

[1] https://github.com/JohnLangford/vowpal_wabbit/wiki

2. Pregled pristupa problemu združenog predviđanja

U ovom poglavlju definiran je pojam strukturnog i združenog predviđanja i učenja. Dan je kratak pregled svih popularnih metoda koje su se primjenjivale na probleme strukturnog predviđanja u području obrade prirodnoga jezika:

1. vjerojatnosni grafički modeli (engl. *probabilistic graphical models*),
   (a) skriveni Markovljev model (engl. *hidden Markov model*, HMM),
   (b) Markovljev model maksimalne entropije (engl. *maximum entropy Markov model*, MEMM),
   (c) uvjetna slučajna polja (engl. *conditional random fields*, CRF),
2. strukturirani perceptron (engl. *structured perceptron*),
3. Markovljeve mreže maksimalne margine (engl. *maximum margin Markov networks*, M^4) i
4. strukturirani stroj potpornih vektora (engl. *structured support vector machine*, SSVM).

Napravljena je i detaljnija analiza sposobnosti modela koji slijede i detaljan opis mana koje objašnjavaju zašto metode nisu dovoljno generalne da bi bile dobre za široku primjenu na problemu strukturnog predviđanja. Poglavlje koje slijedi definira i opisuje problem združenog predviđanja na koji se gornji modeli primijenjuju. Sustavni pregled temeljen je na disertaciji (Daumé III, 2006).

2.1. Združeno predviđanje

Združeno učenje i predviđanje (engl. *joint learning and prediction*) naziv je za metode koje rade učenje i predviđanje jednog primjera mjereći ili minimizirajući funkciju gu-

bitka združeno. Naziv strukturnog učenja i predviđanja (engl. *structured learning and prediction*) implicira da se funkcija gubitka raspoređuje po komponentama strukture i da se na strukturi koja je razbijena u svoje dijelove predviđa i uči – rečenica u značke, slika u slikovne elemente i dr. Chang et al. (2014) daju jednostavnu definiciju strukturnog predviđanja kao donošenja niza združenih odluka računajući vrijednost funkcije gubitka združeno. Ne postoji pretpostavka o tome kakav je ulaz i ima li specifičnu strukturu. U (Chang et al., 2014; Daumé III et al., 2015) odustaju od naziva *strukturno predviđanje* i počinju probleme kategorizirati pod nazivom združeno predviđanje upravo zbog toga što se vrši združeni niz odluka računajući vrijednost funkcije gubitka združeno preko niza odluka i ona nema nužno dekompoziciju preko tog niza (ne može za pojedinačnu odluku znati koliki je njen doprinos u globalnom gubitku). Naziv onda može uključivati združene probleme poput istovremenog označavanja vrste riječi u rečenici i ovisnosnog parsanja rečenice, ali i svaki problem zasebno. Kako bi nastavak bio konzistentan sa starijom literaturom koristi se staro ime, a definicija 1 je zapravo definicija združenog učenja.

U literaturi koja je koristila naziv za problem strukturnog predviđanja on nije jasno bio definiran, nego je objašnjen kroz ilustrativne primjere (McCallum et al., 2000; Punyakanok i Roth, 2001; Lafferty et al., 2001; Collins, 2002; Taskar et al., 2003; McAllester et al., 2004; Tsochantaridis et al., 2005). Daumé III et al. (2009) definiraju jasnije problem, ali potrebno ga je pažljivije definirati razmatrajući dva uvjeta.

Uvjet 1 *U problemu strukturnog predviđanja izlazni elementi $y \in \mathcal{Y}$ dekomponiraju u vektore varijabilnih duljina preko konačnog skupa. Tj., postoji konačan $M \in \mathbb{N}$ tako da je svaki $y \in \mathcal{Y}$ moguće identificirati bar jednim vektorom $v_y \in M^{T_y}$, gdje je T_y duljina vektora.*

Daumé III navodi da ovaj uvjet nije dovoljan i da uključuje probleme koje inače ne bi smatrali strukturnim predviđanjima poput binarne klasifikacije. To vodi do drugog uvjeta koji počinje uključivati funkciju gubitka. Želja je da funkciju gubitka nije moguće dekomponirati preko vektorskih reprezentacija, a kako je uvijek moguće osmisliti vektorsku reprezentaciju preko koje će funkcija gubitka dekomponirati slijedi uvjet.

Uvjet 2 *U problemu strukturnog predviđanja funkcija gubitka nema dekompoziciju preko vektora v_y za $y \in \mathcal{Y}$. Tj., $l(x, y, y^*)$ nije invarijantna na identične permutacije y i y^*. Ne postoji preslikavanje $y \mapsto v_y$ takvo da funkcija ima dekompoziciju za koju je duljina vektora $|v_y|$ polinomijalna u broju izlaznih elemenata $|y|$.*

Ovaj uvjet uspješno isključuje binarnu klasifikaciju i ostale klasifikacijske probleme kao probleme strukturnog predviđanja, ali isključuje i problem označavanja nizova koristeći Hammingovu funkciju gubitka (potonja je invarijantna na permutacije). Razlika između ova dva uvjeta je taj što u prvom značajke diktiraju strukturu, a u drugom funkcija gubitka. Problem koji se rješava je uvijek definiran nekom funkcijom gubitka, a ne značajkama, stoga Daumé III tvrdi da je problem bolje definirati preko funkcije gubitka. Pristupi koji slijede u nastavku ovog poglavlja ne mogu riješiti generalan problem strukturnog predviđanja ako je za definiciju korišten drugi uvjet. Prihvaćena definicija slijedi.

Definicija 1 (Daumé III et al. (2009)) *Problem strukturnog predviđanja $\mathcal{D}$ je klasifikacijski problem osjetljiv na trošak (engl.* cost-sensitive classification problem*), gdje $\mathcal{Y}$ ima strukturu, a elementi $y \in \mathcal{Y}$ imaju dekompoziciju u vektore varijabilne duljine $(y_1, y_2, \ldots, y_T)$. $\mathcal{D}$ je distribucija preko ulaza $x \in \mathcal{X}$ i vektora troška $\mathbf{c}$, gdje je $|\mathbf{c}|$ varijabla u 2^T.*

Kao primjer može se uzeti označavanje vrste riječi koristeći Hammingovu funkciju gubitka. Hammingov bi gubitak za niz odluka kreirao vektor troška $\mathbf{c}$ koji u sebi sadrži za svaku odluku (oznaku vrste riječi) njen trošak (ako je riječ točno označena trošak je 0, inače 1). U ovom zadatku funkcija gubitka ima dekompoziciju preko vektorske reprezentacije strukture $y \in \mathcal{Y}$ (za svaku značku kojoj je dodijeljena oznaka moguće je zasebno odrediti gubitak).

Parsanje je primjer zadatka koji zadovoljava uvjet 2. Ako se koristi funkcija gubitka F_1 (harmonijska sredina između preciznosti i odziva), onda nije moguće za pojedinu odluku stvaranja brida stabla odrediti parcijalan gubitak jer se F_1 računa na potpunim stablima. U tom slučaju je $\mathcal{D}$ distribucija preko $(x, \mathbf{c})$, gdje je x ulazni niz i za sva stabla y s $|x|$ listova, c_y je funkcija gubitka F_1 izlaza y na točnom izlazu y^*. Cilj strukturnog predviđanja je pronaći funkciju $h : \mathcal{X} \to \mathcal{Y}$ koja minimizira očekivanu vrijednost funkcije gubitka 2.1.

$$L(\mathcal{D}, h) = \mathbb{E}_{(x,\mathbf{c})\sim\mathcal{D}}\{c_{h(x)}\} \tag{2.1}$$

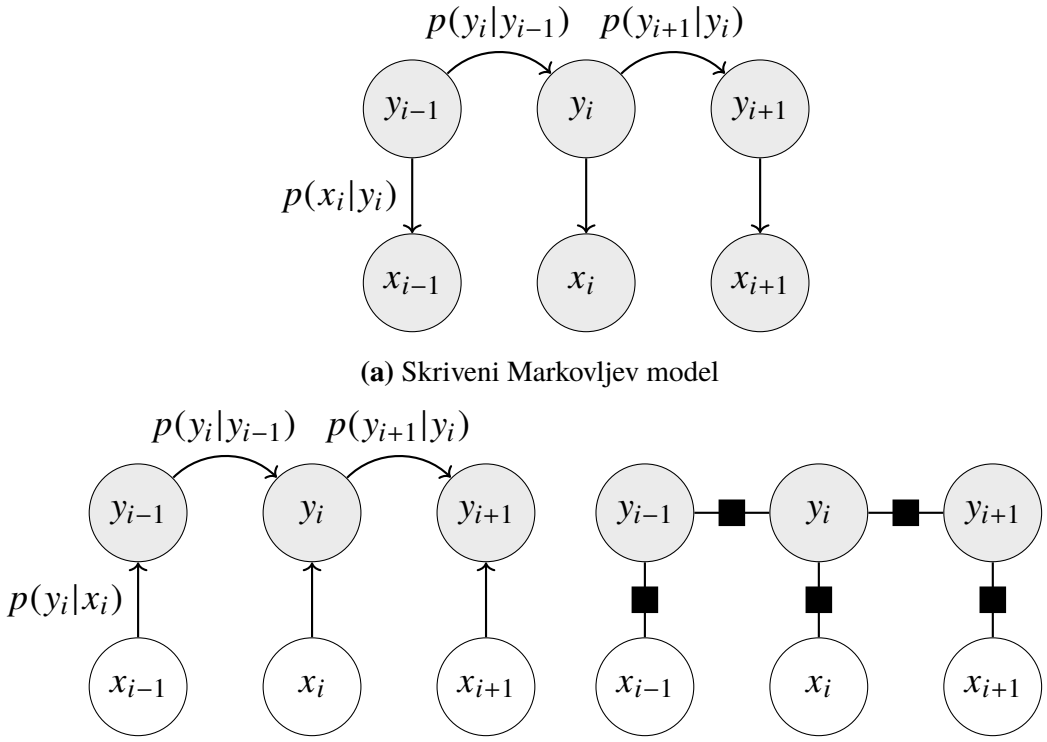
Sve metode strukturnog predviđanja u ovom poglavlju, nakon što je naučen vektor težina $\mathbf{w}$, za predviđanje strukture trebaju riješiti argmax-problem 2.2:

$$y^* = \underset{y\in\mathcal{Y}}{\operatorname{argmax}}\, \mathbf{w}^\top \mathbf{\Phi}(x, y)\ , \tag{2.2}$$

gdje je $\mathbf{w}$ naučen vektor težina i $\mathbf{\Phi}$ funkcija značajki za ulaz x i mogući izlaz y (ako se struktura predviđa inkrementalno, onda je moguće koristiti elemente y_i iz vektora dekompozicije kao značajke). Ovaj problem u generalnom slučaju nije izračunljiv u dovoljno kratkom vremenu. Ako postoji prikladan $\mathcal{Y}$ i prikladna funkcija značajki $\mathbf{\Phi}$, onda se mogu primijeniti dinamičko programiranje (engl. *dynamic programming*) ili cjelobrojno linearno programiranje (engl. *integer linear programming*) za pronalaženje dobrih rješenja. Recimo, ako $\mathbf{\Phi}$ ima takvu dekompoziciju preko vektorske reprezentacije $\mathcal{Y}$ da ni jedna značajka ne ovisi o elementima y koji su k ili više pozicija udaljeni, onda se može iskoristiti Viterbijev algoritam za rješenje 2.2, a složenost će biti $O(TM^k)$ – gdje je M broj mogućih oznaka, definiran u uvjetu 1, a T duljina vektora na koji se dekomponira y iz definicije 1.

2.2. Vjerojatnosni grafički modeli

Najpoznatiji vjerojatnosni grafički modeli korišteni za probleme strukturnog predviđanja su skriveni Markovljevi modeli (engl. *hidden Markov models*, HMM), Markovljev model maksimalne entropije (engl. *maximum entropy Markov model*, MEMM) i uvjetna slučajna polja (engl. *conditional random fields*, CRF). Svaki ima svoje prednosti. Učenje skrivenog Markovljevog modela moguće je izvesti vrlo brzo ako je na raspolaganju velika količina podataka. Prijelazne vjerojatnosti $p(x_i|y_i)$ i $p(y_i|y_{i-1})$ moguće je naučiti vrlo brzo prebrojavanjem supojavljivanja. Kod uvjetnih slučajnih polja lakše je opisati ulaz značajkama što omogućava bolju generalizaciju iz manje količine podataka, ali proces učenja je dugotrajniji. Markovljev model maksimalne entropije je po učinkovitosti između dva modela, omogućava brzo učenje i dobru generalizaciju, ali je primijećeno da način učenja dovodi do problema pristranosti oznakama (engl. *label bias*) (Lafferty et al., 2001), a problem je detaljnije obrađen u potpoglavlju 3.5.



(a) Skriveni Markovljev model

(b) Markovljev model maksimalne entropije

(c) Uvjetna slučajna polja

Slika 2.1: Prikaz predložaka grafičkih modela. Sivi čvorovi predstavljaju varijable koje model može generirati i opaziti, a bijeli koje model može samo opaziti.

2.2.1. Skriveni Markovljev model

Koristi se najčešće kod problema gdje prijelazne vjerojatnosti modeliraju niz – kao što je prikazano na slici 2.1. To čini model prikladnim za problem označavanja vrste riječi (Halácsy et al., 2007) i prepoznavanja imenovanih entiteta (Zhou i Su, 2002). Kako u većini slučajeva skup za učenje nije dovoljno velik, uobičajeno je dodati neke vanjske informacije – poput toga da se imenovani entitet koji nije bio u skupu za učenje pojavljuje u nekom rječniku koji je na raspolaganju ili da su riječi s određenim prefiksom su češće imenice nego glagoli – onda se algoritam učenja brojanjem supojavljivanja zamjenjuje Baum-Welch algoritmom koji ima složenost $O(TM^k)$ – gdje je T duljina niza, M broj oznaka koje je moguće predvidjeti za svaki izlaz i izlaz ne ovisi o elementima u y koji su k ili više pozicija udaljeni, ali pokazalo se da algoritam teško dolazi do dobrog lokalnog optimuma te se za navedene probleme ipak koriste drugi modeli (Johnson, 2007).

2.2.2. Markovljev model maksimalne entropije

Markovljev model maksimalne entropije (engl. *maximum entropy Markov model*) izravno su primijenili McCallum, Freitag, i Pereira (2000) koristeći model maksimalne entropije (logističke regresije) na probleme označavanja nizova. Model je vrlo sličan skrivenom Markovljevom modelu (engl. *hidden Markov model*), ali je fleksibilniji jer omogućava veću slobodu u odabiru značajki opaženih varijabli. Modelira se uvjetna vjerojatnost i-te oznake y_i izlaza y za dane x i prijašnje oznake y_{i-1} prikazana u jednadžbi 2.3.

$$\begin{aligned} p(y_i|x, y_{i-1}; \mathbf{w}) &= \frac{1}{Z(x, y_{i-1}; \mathbf{w})} \exp\left[\mathbf{w}^\top \mathbf{\Phi}(x, y_i, y_{i-1})\right] \\ Z(x, y_{i-1}; \mathbf{w}) &= \sum_{y' \in \mathcal{Y}} \exp\left[\mathbf{w}^\top \mathbf{\Phi}(x, y', y_{i-1})\right] \end{aligned} \tag{2.3}$$

Model se uči koristeći zlatne izlazne nizove kao skup podataka za učenje i koriste se zlatni y_{i-1} za generiranje primjera za učenje. Navedeni postupak proizvede primjere za učenje višerazredne klasifikacije. Broj primjera jednak je broju svih pojavljivanja oznaka u cijelom skupu za učenje (npr. za problem označavanja niza to bi bio zbroj duljina svih nizova). Taj generirani skup koristi se za učenje vektora $\mathbf{w}$ postupkom koji je identičan učenju modela maksimalne entropije – za učenje se može koristiti neka varijanta gradijentnog spusta.

Kod zaključivanja koristi se Viterbijev algoritam za rješavanje argmax-problema (2.2). Zbog ove kombinacije algoritama učenja i zaključivanja dolazi do problema pristranosti oznakama (potpoglavlje 3.5).

Cohen i Carvalho (2005) pokušali su smanjiti utjecaj pristranosti oznakama i uspješnost je usporediva s uvjetnim slučajnim poljima, ali sve prednosti vezane uz vremensku učinkovitost učenja iščezavaju. Unatoč ovim promjenama model se može primjenjivati samo na probleme označavanja nizova.

2.2.3. Uvjetna slučajna polja

Lafferty, McCallum, i Pereira (2001) uvode uvjetna slučajna polja kao rješenje problema pristranosti oznakama i eksperimentalno potvrđuju njegovo nepostojanje. Problem je poznat i ranije kod primjene neuronskih Markovljevih modela na slične probleme (Bottou, 1991). Uvjetna slučajna polja nemaju određenu funkciju gubitka, nego koriste log-gubitak kao aproksimaciju Hammingovog gubitka preko cijelog strukturiranog izlaza. Zbog toga ih je moguće primijeniti samo na probleme čija se funkcija gubitka

može dekomponirati po elementima. Modelira se uvjetna vjerojatnost prikazana u jednadžbi 2.4.

$$\begin{aligned} p(y|x;\mathbf{w}) &= \frac{1}{Z(x;\mathbf{w})} \exp\left[\mathbf{w}^\top \mathbf{\Phi}(x,y)\right] \\ Z(x;\mathbf{w}) &= \sum_{y' \in \mathcal{Y}} \exp\left[\mathbf{w}^\top \mathbf{\Phi}(x,y')\right] \end{aligned} \tag{2.4}$$

Particijska funkcija $Z(x;\mathbf{w})$ u zbroju koristi sve izlaze y' za koje model daje pogrešna predviđanja. Ako se koristi iterativni algoritam učenja, onda se za pravilnu procjenu parametara taj zbroj računa svaki put prije korigiranja težinskog vektora $\mathbf{w}$. Taj je zbroj u generalnom slučaju prevelik za efikasno računanje, ali ako se pretpostavi struktura linearnog lanca, onda ga je moguće izračunati koristeći dinamičko programiranje i svaka prilagodba na novi problem koji je predstavljen netrivijalnim linearnim lancem zahtijeva novu implementaciju algoritama učenja i zaključivanja (Lafferty et al., 2001; Sha i Pereira, 2003).

Algoritam za računanje zbroja $Z(x;\mathbf{w})$ ima složenost $O(TM^k)$ i gotovo je identičan algoritmu *forward-backward* (Baum i Petrie, 1966).

Kao što je slučaj kod modela maksimalne entropije, težine $\mathbf{w}$ moguće je regularizirati i koristi se log-posteriorna distribucija preko težina opisana u jednadžbi 2.5.

$$\log p(\mathbf{w}|\mathcal{D};\sigma^2) = -\frac{1}{\sigma^2}\|\mathbf{w}\|^2 + \sum_{(x_n,y_n)\in\mathcal{D}} \left[\mathbf{w}^\top \mathbf{\Phi}(x_n,y_n) - \log Z(x_n;\mathbf{w})\right] \tag{2.5}$$

Pronalaženje optimalnog vektora težina $\mathbf{w}$ rješava se na razne načine (Lafferty et al., 2001; Sha i Pereira, 2003; Sokolovska et al., 2010) i ovisno o problemu neke je moguće ili nemoguće primijeniti, a za više detalja čitatelja se upućuje na (Wallach, 2004; Sutton i McCallum, 2006). Zaključak je da se uvjetna slučajna polja mogu primjenjivati na probleme gdje se efikasno može riješiti argmax-problem (2.2) i izračunati particijska funkcija $Z(x;\mathbf{w})$. Funkcija gubitka mora imati takvu dekompoziciju preko izlaza $y \in \mathcal{Y}$ da je moguće log-gubitkom aproksimirati Hammingov gubitak. Unatoč navedenim uvjetima model je našao primjene u označavanju vrste riječi (Lafferty et al., 2001), prepoznavanju imenovanih entiteta (McCallum i Li, 2003; Settles, 2004), sažimanju dokumenata (engl. *document summarization*) (Shen et al., 2007), analizi sentimenta (McDonald et al., 2007) i dr. U svakoj primjeni funkcija gubitka imala je dekompoziciju preko niza odluka i algoritmi su se morali prilagoditi za pravilno zaključivanje ili se odbacivanjem točnog zaključivanja koristilo aproksimativno.

2.3. Strukturirani perceptron

Perceptron (Rosenblatt, 1958) je moguće proširiti tako da se može koristiti za rješavanje problema strukturnog predviđanja. Model je koji dozvoljava učenje primjer po primjer (engl. *online learning*), a najbrža varijanta s najboljom generalizacijom uzima prosjek težina koje su dobivene svakim primjerom (Freund i Schapire, 1999; Gentile, 2002) – dan je pseudokod 2.1. Algoritam bez usrednjavanja vraća samo $\mathbf{w}_0$ i b_0.

Algoritam 2.1 Perceptron algoritam s usrednjavanjem.

Potrebno: Skup podataka $\{x_i, y_i\}_{i=1}^N$ i broj prolaza I.

1: $\mathbf{w}_0 \leftarrow \langle 0, \ldots, 0 \rangle$, $b_0 \leftarrow 0$
2: $\mathbf{w}_a \leftarrow \langle 0, \ldots, 0 \rangle$, $b_a \leftarrow 0$
3: $c \leftarrow 1$
4: **za svaki** $i \in \{1, 2, \ldots, I\}$
5: | **za svaki** $n \in \{1, 2, \ldots, N\}$
6: | | **ako** $y_n \cdot (\mathbf{w}_0{}^\top \mathbf{\Phi}(x_n) + b_0) \leq 0$ **onda**
7: | | | $\mathbf{w}_0 \leftarrow \mathbf{w}_0 + y_n \mathbf{\Phi}(x_n)$, $b_0 \leftarrow b_0 + y_n$
8: | | | $\mathbf{w}_a \leftarrow \mathbf{w}_a + c y_n \mathbf{\Phi}(x_n)$, $b_a \leftarrow b_a + c y_n$
9: | | **kraj**
10: | | $c \leftarrow c + 1$
11: | **kraj petlje**
12: **kraj petlje**
13: **vrati** $(\mathbf{w}_0 - \mathbf{w}_a/c, b_0 - b_a/c)$

Proširenje potrebno za rješavanje problema strukturnog predviđanja prikazano je na pseudokodu 2.2. Collins (2002) uvodi model strukturnog perceptrona (engl. *structured perceptron*) kao zamjenu za uvjetna slučajna polja gdje je i dalje moguće imati vrlo bogatu reprezentaciju ulaza, ali je učenje puno jednostavnije i brže.

Jedini je zahtjev da se argmax-problem na liniji 6 može riješiti efikasno. U većini slučajeva rješava se Viterbijevim algoritmom ili aproksimativno. Postoji pretpostavka da je problem koji se rješava lako dekomponirati na Hammingov gubitak, što je malo bolje nego njegova aproksimacija log-gubitkom – kao što je slučaj kod uvjetnih slučajnih polja – jer se optimizira pogrešna dodjela oznake izlaza (0 ili 1), a ne stupanj dodjele.

Prva pojava pretraživanja u strukturnom predviđanju pojavljuje se u (Collins i Roark, 2004), gdje je strukturni perceptron korišten za ovisnosno parsanje. Jedina razlika je u tome što se argmax-problem rješava koristeći algoritam *beam* pretrage, stoga se linija 6 zamjenjuje s $\hat{y}_n \leftarrow \textsc{BeamPretraga}(x_n, \mathbf{w}_0)$. Algoritam, umjesto da ima samo jedno

Algoritam 2.2 Strukturirani perceptron algoritam s usrednjavanjem.

Potrebno: Skup strukturiranih podataka $\{x_i, y_i\}_{i=1}^{N}$ i broj prolaza I.

$\mathbf{w}_0 \leftarrow \langle 0, \ldots, 0 \rangle$
$\mathbf{w}_a \leftarrow \langle 0, \ldots, 0 \rangle$
$c \leftarrow 1$
za svaki $i \in \{1, 2, \ldots, I\}$
  za svaki $n \in \{1, 2, \ldots, N\}$
    $\hat{y}_n \leftarrow \text{argmax}_{y \in \mathcal{Y}} \mathbf{w}_0{}^{\top} \mathbf{\Phi}(x_n, y)$
    ako $y_n \neq \hat{y}_n$ **onda**
      $\mathbf{w}_0 \leftarrow \mathbf{w}_0 + \mathbf{\Phi}(x_n, y_n) - \mathbf{\Phi}(x_n, \hat{y}_n)$
      $\mathbf{w}_a \leftarrow \mathbf{w}_a + \mathbf{\Phi}(x_n, y_n) - \mathbf{\Phi}(x_n, \hat{y}_n)$
    kraj
    $c \leftarrow c + 1$
  kraj petlje
kraj petlje
vrati $\mathbf{w}_0 - \mathbf{w}_a / c$

stablo kao hipotezu, ima ih više i time pokušava izbjeći pristranost oznakama. Slična je ideja korištena u trenutno najtočnijem ovisnosnom parseru opisanom u (Andor et al., 2016). Uz *beam* pretragu koriste dvoslojnu neuronsku mrežu gdje svaki sloj ima 1024 neurona koju treniraju globalnom normalizacijom, ali umjesto Hammingovog gubitka koriste aproksimaciju log-gubitkom. Kao i kod prijašnjih modela, strukturirani perceptron ima ograničenje da funkcija gubitka mora imati dekompoziciju na izlaz $y \in \mathcal{Y}$.

2.4. Markovljeve mreže maksimalne margine

Markovljeve mreže maksimalne margine (engl. *maximum margin Markov networks*, M^4) dopuštaju formulaciju problema strukturnog predviđanja kroz problem kvadratnog programiranja (engl. *quadratic programming*) koristeći formalizam stroja potpornih vektora za binarnu klasifikaciju. Formulacija modela M^4 dana je u nastavku:

$$
\begin{aligned}
\min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\sum_{\hat{y}} \xi_{n,\hat{y}} \,, \\
\text{uz uvjete} \quad & \mathbf{w}^\top \mathbf{\Phi}(x_n, y_n) - \mathbf{w}^\top \mathbf{\Phi}(x_n, \hat{y}) \geq l(x_n, y_n, \hat{y}) - \xi_{n,\hat{y}} \qquad \forall n, \forall \hat{y} \in \mathcal{Y} \,, \\
& \xi_{n,\hat{y}} \geq 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \forall n, \forall \hat{y} \in \mathcal{Y} \,.
\end{aligned}
\tag{2.6}
$$

Bitno je naglasiti da formulacija M^4 ima previše uvjeta. Za svaki primjer (x_n, y_n) nad kojim se uči i za svaki netočni izlaz $\hat{y}$ potrebno je izgraditi uvjet. Moguće je uz određene pretpostavke o $\mathcal{Y}$ i $\mathbf{\Phi}$ zamijeniti eksponencijalan broj uvjeta polinomijalnim. Zamjena je moguća jedino u slučaju problema strukturnog predviđanja kod kojeg funkcija gubitka ima dekompoziciju preko niza – u praksi je to uvijek Hammingov gubitak.

Za efikasno učenje i zaključivanje funkcija gubitka mora imati dekompoziciju preko niza odluka. Prethodan uvjet ograničava M^4, ali postoje dvije prednosti nad uvjetnim slučajnim poljima. M^4 ne zahtijeva efikasan izračun normalizacijske particijske funkcije 2.4 i može se primijeniti na probleme koji imaju drugačiju funkciju gubitka od Hammingovog gubitka. U praksi je moguće primijeniti M^4 samo na probleme koji aproksimiraju Hammingov gubitak gubitkom zglobnice (engl. *(*)hinge loss).

Čitatelja se upućuje na (Taskar et al., 2003) za detaljniji pregled. S obzirom na to da je velike margine moguće aproksimirati jednostavnim tehnikama učenja pomoću gradijenta, M^4 se ne primjenjuje u svojoj originalnoj formulaciji (Daumé III i Marcu, 2005; Ratliff et al., 2006).

2.5. Strukturirani stroj potpornih vektora

Strukturirani stroj potpornih vektora (engl. *structured support vector machine*, SSVM) također se može primjenjivati na probleme strukturnog predviđanja (Tsochantaridis et al., 2005). Vidljiva je velika sličnost s modelom M^4. Razlika je u tome da M^4 skalira marginu koristeći gubitak, dok SSVM skalira *slack* varijable koristeći gubitak. Problem kvadratnog programiranja priložen je ispod.

$$
\begin{aligned}
\min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\sum_{\hat{y}} \xi_{n,\hat{y}} \,, \\
\text{uz uvjete} \quad & \mathbf{w}^\top \mathbf{\Phi}(x_n, y_n) - \mathbf{w}^\top \mathbf{\Phi}(x_n, \hat{y}) \geq 1 - \frac{\xi_{n,\hat{y}}}{l(x_n, y_n, \hat{y})} \qquad \forall n, \forall \hat{y} \in \mathcal{Y} \,, \\
& \xi_{n,\hat{y}} \geq 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \forall n, \forall \hat{y} \in \mathcal{Y} \,.
\end{aligned}
\tag{2.7}
$$

Funkcija koju je potrebno minimizirati identična je onoj u formulaciji M^4, a razlika je samo u prvom uvjetu. Intuitivnije je skalirati grešku tijekom učenja (rezervne (engl. *slack*) varijable) gubitkom nego skalirati marginu. Autori tvrde da je ovaj formalizam bolji od M^4 jer će potonji tjerati sustav da razdvoji hipoteze za koje je odabran velik gubitak čak i ako je te hipoteze teško zamijeniti istinom (npr. zbog toga što je za imenice određen velik gubitak – jer želimo imenice točnije označavati – formalizam M^4 će, unatoč tome što je teško zamijeniti imenice brojevima, pokušavati povećati marginu). Uz razlike u skaliranju, metode koriste drugačije algoritme učenja. Kod modela M^4 moguće je, u slučaju dekompozicije funkcije gubitka, uvelike smanjiti broj dodatnih uvjeta, dok kod modela SSVM to nije moguće. Tsochantaridis et al. (2005) navode da se uvjeti dodaju po potrebi i daju garancije da će takva strategija dovesti do dobrog rješenja u polinomnom broju koraka.

Glavna mana ovog formalizma je nemogućnost jednostavnog efikasnog učenja. Unatoč tome ovo je jedini formalizam u kojem nije potrebno da funkcija gubitka ima dekompoziciju preko strukture. Postoje gradijentne metode pomoću kojih je moguće učiti modele M^4 i SSVM, ali one zahtijevaju, uz rješavanje argmax-problema 2.2, rješavanje takozvane *pretrage proširene gubitkom*. Kod modela M^4 ta je pretraga rješiva zbog mogućnosti dekompozicije funkcije gubitka, dok kod modela SSVM nije (Ratliff et al., 2006).

3. Učenje pretraživanja

Učenje pretraživanja mlado je područje, ali u zadnjih deset godina pojavili su se pristupi koji su riješili mnoštvo otvorenih pitanja i učinili pristup konkurentnim uspoređujući ga s tad najboljim modelima u strojnom učenju. Za teorijska temelja i razvoj metoda L2S čitatelja se upućuje na (Collins i Roark, 2004; Daumé III i Marcu, 2005; Daumé III et al., 2009; Ross et al., 2011; Doppa et al., 2014; Ross i Bagnell, 2014; Chang et al., 2015b; Andor et al., 2016). Neki od navedenih izvora pretpostavljaju znanje o *redukcijama* u strojnom učenju, a za kratak uvod čitatelja se upućuje na (Beygelzimer et al., 2005a, 2015).

3.1. Redukcije u strojnom učenju

Redukcije u strojnom učenju postupak su za transformiranje složenijih problema u jednostavnije – kao što je primjer redukcije problema višerazredne klasifikacije u problem binarne klasifikacije gdje se može koristiti jedan-protiv-svih (engl. *one-against-all*, OVA) ili jedan-protiv-drugog (engl. *one-against-one*, OVO) pristup korištenja binarnih klasifikatora. Za pristup OVA potrebno je n klasifikatora gdje je n broj razreda, a OVO zahtijeva $\binom{n}{2}$ klasifikatora. Prvi ima prednost u vremenu ako je broj razreda velik, a potonji ako je vektor gust za OVA, a rijedak za OVO (ako je moguće izbaciti značajke tj. zadržati samo one koje su vezane za dva razreda koja uspoređujemo). Ostale prednosti ovise o samom problemu, a čak i o vrsti binarnog klasifikatora koji koristimo (Milgram et al., 2006). Zanimljiv je primjer redukcija problema višerazredne klasifikacije stablom odluka (engl. *decision tree*), gdje se svaki razred može binarno kodirati, a bitovi predstavljaju niz da/ne odgovora o tome kojim bridom krenuti gdje bi se nakon $O(\log n)$ koraka potpuno kodirao razred (Beygelzimer et al., 2009; Daumé III et al., 2016). Ovo je eksponencijalno ubrzanje s obzirom na prethodne redukcije u vremenu, a i broj klasifikatora je puno manji – $\lceil \log n \rceil$.

Pitanje je jesu li redukcije učinkovit pristup za rješavanje složenih problema strojnog učenja? Odgovor nije očit jer je teško opisati reprezentaciju na trivijalan matematički

način. Moguće je da redukcija stvori jednostavne probleme koje nije lako riješiti. Trivijalan je primjer trorazredne klasifikacije s jednom značajkom i linearnim binarnim klasifikatorom. Kod pristupa OVA nije moguće odvojiti razrede s dovoljno malom pogreškom, a kod ovo je to moguće.

Redukcije nisu toliko trivijalne kao ovi primjeri. Postoje tri potrebne komponente: (1) preslikavanje hipoteze, (2) preslikavanje primjera i (3) analiza ograde. Preslikavanje hipoteze opisuje postupak kako pretvoriti rješenje za jednostavan problem u rješenje za složen. Preslikavanje primjera opisuje postupak kako stvoriti skupove podataka za jednostavan problem koristeći skupove podataka za složen problem. Ograda daje jamstvo učinkovitosti na složenom problemu ako postoji dobra učinkovitost na jednostavnom problemu.

Postoje dvije vrste ograda koje se uzimaju u obzir: ograde na ograničenje pogreške (engl. *error bounds*) i ograde na ograničenje žaljenja (engl. *regret bounds*). U slučaju redukcije koja ograničava pogrešku, teoretsko jamstvo tvrdi da mala pogreška na jednostavnom problemu podrazumijeva malu pogrešku na složenom problemu. U slučaju redukcije koja ograničava žaljenje, ograda tvrdi da malo žaljenje na jednostavnom problemu podrazumijeva malo žaljenje na složenom.[1] Zadovoljavajuća je činjenica da ograde imaju svojstvo kompozicije (Beygelzimer et al., 2005a). Ako je moguće reducirati problem X u problem Y s ogradom f i problem Y u problem Z s ogradom g, onda je kompozicija $X \circ Y$ redukcija s ogradom $f \circ g$. Navedeno svojstvo pojednostavljuje matematičku analizu ograda redukcije. Bitna je razlika između navedenih ograda što, ako redukcija s ograničenjem pogreške kreira jednostavne probleme koji su šumoviti, onda su bilo kakve tvrdnje o težem problemu isprazne.

Potrebno je znati je li redukcija konzistentna (engl. *consistent*) da bismo se uvjerili da će redukcija raditi dobro (Beygelzimer et al., 2009, 2015). Neformalno, redukcija koja transformira bilo koje optimalno rješenje za osnovni jednostavni problem u optimalno rješenje za složen problem je *konzistentna*. Ispostavlja se da su gore navedene jednostavne redukcije stablom odluke i pristup OVA nekonzistentne. Navedene uvjete nekad je moguće zadovoljiti ne samo odabirom načina zaključivanja (OVO, OVA i dr.) nego i načinom učenja (Abe et al., 2004; Beygelzimer et al., 2005b) – način učenja glavni je razlog mogućnosti redukcije strukturnog predviđanja u višerazrednu klasifikaciju. Konzistentnost je osnovni uvjet za dobru redukciju, a redukcije s ograničenjem pogreške su, na žalost, uglavnom nekonzistentne. Čitatelja se upućuje na sustavni pregled zadnjih deset godina redukcija u strojnom učenju (Beygelzimer et al., 2015).

[1] Žaljenje hipoteze h na problemu $\mathcal{D}$ je razlika u pogrešci kod korištenja h i korištenja najboljeg mogućeg klasifikatora. Formalno, $R(\mathcal{D}, h) = L(\mathcal{D}, h) - \min_{h^*} L(\mathcal{D}, h^*)$.

Redukcije su bitna sastavnica u najuspješnijim metodama L2S jer bez njih ne bi postojala informacija o tome zašto su toliko uspješne, nego bi pretpostavljali da koristimo heurističke metode. Heurističke metode prisutne su u izobilju kod primjena na probleme združenog predviđanja i bez teorije postoji samo empirijska potvrda njihove uspješnosti.[2]

3.2. Politika i lokalna optimalnost

Politika (engl. *policy*) termin je za naučen model nakon postupka učenja ili referentni sustav ili algoritam koji vrši određeno ponašanje. U području podržanog učenja jedina politika koja postoji je ona koja je naučena. U području učenja oponašanjem (engl. *imitation learning*) postoji više vrsta politika. Politika koja je naučena, referentna politika koju se pokušava oponašati i prava optimalna politika (stručnjak). U nastavku slijedi nekoliko definicija koje bi trebale raščistiti koncept politika.

Definicija 2 (Politika) *Politika π je distribucija preko odluka koje su uvjetovane ulazom x i stanjem s.*

Skoro svi pristupi u potpoglavlju 3.4 zahtijevaju pristup efikasnoj i optimalnoj politici π^*. Bez takve politike učenje može biti dugotrajno ili redukcija nekonzistentna. Potrebno je definirati optimalnu politiku.

Definicija 3 (Optimalna politika) *Za par $(x, \mathbf{c})$ iz definicije 1 i stanje $s = \langle y_1, \ldots, y_t \rangle$ u prostoru pretraživanja optimalna politika $\pi^*(x, \mathbf{c}, s)$ je $\operatorname{argmin}_{y_{t+1}} \min_{y_{t+2},\ldots,y_T} c_{\langle y_1,\ldots,y_T \rangle}$. Tj. π^* odabire odluku (vrijednost za y_{t+1}) koja minimizira cjelokupni gubitak pretpostavljajući da će sve ostale buduće odluke biti donesene optimalno.*

Lokalno optimalno učenje pretraživanja – LOLS (potpoglavlje 3.4.4) – ima garancije o tome kakva će naučena politika biti, a definicija je dana ispod.

Definicija 4 (Lokalno optimalna politika) *Naučena politika je lokalno optimalna akko promjena bilo koje prijašnje odluke ne može poboljšati učinkovitost naučene politike.*

Potrebno je znati težinu učenja lokalno optimalne politike. Možda je problem lokalne optimalnosti toliko težak da bilo koji efikasan algoritam učenja neće moći naučiti lokalno optimalnu politiku u zadovoljavajućoj količini vremena. Teorem koji slijedi

[2]Dobar primjer je razvoj metoda za ovisnosno parsanje koje koriste *beam* pretragu i time eksperimentalno poboljšavaju performanse (Zhang i Nivre, 2011; Bohnet i Nivre, 2012).

govori o tome koliko je ažuriranja potrebno napraviti da bi naučena politika bila lokalno optimalna.

Teorem 1 *Pretpostavimo dostupnost algoritma koji ažurira politike koristeći odstupanje od jednog koraka od trenutne politike. Onda postoji problem s prostorom pretraživanja, razred politike i funkcija gubitka gdje takav algoritam mora napraviti* $\Omega(2^T)$ *ažuriranja prije nego što nauči lokalno optimalnu politiku.*

Konstrukcija problema koja dokazuje teorem 1 prisutna je u (Chang et al., 2015b). Naravno, kod problema gdje je moguće napraviti dekompoziciju prostora odluka i funkcije gubitka – kao što je slučaj kod označavanja vrste riječi – ne postoji konstrukcija koja bi zahtijevala eksponencijalno mnogo primjera. Ilustrativniji je primjer igranja šaha. Mnoštvo metoda koristi, uz bazu odigranih partija i dobro analiziranih otvaranja, pretraživanje tijekom igre za pronalazak najboljih poteza. Pitanje je može li se naučiti šah bez potrebe za pretraživanjem tijekom igre? Može li se naučiti politika koja će biti lokalno optimalna? Takva politika bi garantirala da nakon odigrane partije šaha ne postoji potez koji se može ispraviti i time promijeniti rezultat igre. Tj. ako bi postojao takav potez i ako bi se ažurirala naučena politika, onda ona ne bi mogla postati bolja. U slučaju šaha očito je da bi politika trebala postati bolja. Ako politika za svako stanje ploče može reći optimalnu vjerojatnost dobitka partije, onda bi otkriće boljeg poteza tu vjerojatnost trebalo promijeniti. Očigledno je da naučena politika za šah može biti lokalno optimalna jedino u slučaju da je odigran ogroman broj partija. Moguće je da šah koji igraju ljudi nije šah koji bi igrala sveznajuća računala. Potezi koje rade ljudi imaju određenu distribuciju gdje bi učenje lokalno optimalne politike koja igra protiv čovjeka zahtijevalo manje primjera. Dobar stvaran primjer je nedavna pobjeda AlphaGo protiv ljudskog igrača u igri Go. Silver et al. (2016) izgrađuju politiku koja za svako stanje vraća vjerojatnost pobjede uzevši u obzir sljedeći potez. S obzirom na broj mogućih igara ($\approx 2 \cdot 10^{170}$), vrlo je teško izgraditi takvu politiku čija će procjena vjerojatnosti biti lokalno optimalna te za poboljšanje učinkovitosti ipak koriste pretragu tijekom zaključivanja.

3.3. *Rollin* i *rollout*

Chang, Krishnamurthy, Agarwal, Daumé III, i Langford (2015b) uvode terminologiju *rollin* i *rollout* u kontekst metoda učenja pretraživanja. Termin *rollout* originalno je naziv za tehniku analize pozicija i poteza u igri *backgammon*.[3] Cilj je dovoljan broj

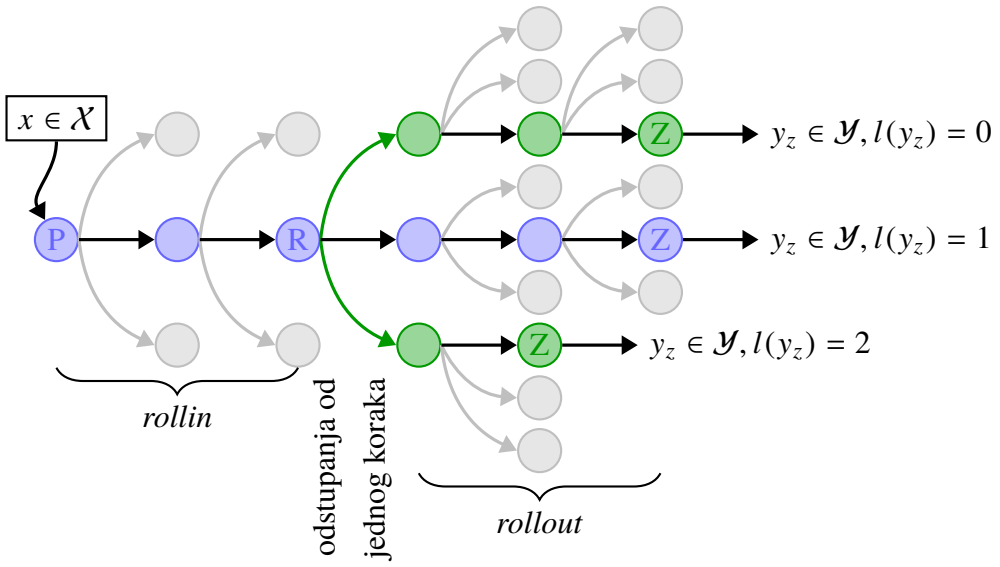
[3] https://en.wikipedia.org/wiki/Rollout_(backgammon)

puta krenuti od iste pozicije i odigravanjem (bacajući kocku) dolaziti do kraja igre i bilježiti rezultat svakog ishoda. Broj pobjeda i poraza daje procjenu o dobroti pozicije. U okviru podržanog učenja tehnika se koristi tako da se iz nekog stanja nasumično odabiru sljedeći potezi – odabir može biti baziran na do tada naučenoj politici. Nakon što se izvrši jedan niz poteza i dobije rezultat, cijeli postupak se ponavlja počinjanjem iz istog stanja tako da je procjena o dobroti pozicije što bolja.

Termin *rollin* nije bio prisutan kod podržanog učenja jer postoji samo naučena politika, ali se podrazumijeva da se do željenog stanja dolazi koristeći naučenu politiku – ako su neka stanja nedostižna trenutnoj naučenoj politici, onda bi bilo bespotrebno vrednovati odluke iz stanja do kojih se ne može doći. U kontekstu metoda učenja pretraživanja *rollin* je postupak dolaska do određenog stanja, a kako je na raspolaganju naučena i referentna politika dobro je vidjeti koja strategija vodi do brzog i konzistentnog učenja – hoće li se dolaziti do budućeg stanja referentnom ili naučenom politikom, a možda za svaki korak odabrati nekom vjerojatnošću jednu ili drugu. Na slici 3.1 prikazan je postupak korištenja politika u postupku *rollin* i *rollout* za vrednovanje odluka gdje su odluke predstavljene bridovima, a stanja čvorovima.

Chang et al. (2015b) pokazuju da korištenje višerazrednog klasifikatora osjetljivog na trošak, koji ima dobre ograde žaljenja (žaljenje mora biti konstantno tj. neovisno o broju binarnih klasifikatora u redukciji), ne garantira konzistentnu naučenu politiku za problem strukturnog predviđanja. Ovisno o načinu na koji se radi *rollin* i *rollout* i pretpostavci o tome kakva je referentna politika, moguće je dobiti nekonzistentnu redukciju. Rezultat je priložen na slici 3.2. Ovisno o tome koje se politike koriste u postupcima *rollin* ili *rollout*, moguće je dobiti naučenu politiku koja nije konzistentna (ima veliko strukturno žaljenje tj. žaljenje nije konstantno nego ovisi o duljini niza odluka) ili politiku koja nije lokalno optimalna. U slučaju korištenja naučene politike za *rollin* i *rollout*, problem strukturnog predviđanja reducira se na problem podržanog učenja koji je puno teži jer se za učenje uopće ne koristi znanje referentne politike.

Jedini dobar pristup kojim se postiže lokalna optimalnost i konzistentna redukcija je onaj u kojem se koristi mješovita politika za *rollout* – opis metode lokalno optimalnog učenja pretraživanja nalazi se u potpoglavlju 3.4.4. Ako se pretpostavi da je referentna politika optimalna i to je stvarno slučaj, onda će učenje koristeći *rollin* s naučenom, a *rollout* s referentnom rezultirati s konzistentnom i lokalno optimalnom naučenom politikom. Za mnoštvo problema nije lako definirati optimalnu politiku (nije lako iz podataka znati koji je sljedeći korak najbolji, npr. prisutnost prijevoda nekog dokumenta nije dovoljna da znamo, u slučaju da dobijemo parcijalan prijevod dokumenta, koja riječ slijedi) i zanimljiv je rezultat da upravo *rollout* s mješovitom politikom ima garancije



Slika 3.1: Prikaz postupka *rollin* i *rollout* kod učenja pretraživanja. Ulaz $x \in \mathcal{X}$ vodi pretragu svojim značajkama. Kreće se iz stanja P i dva se puta odabire srednja odluka od moguće tri koristeći odabranu politiku za *rollin*. Sivi čvorovi se ne pretražuju. U koraku R algoritam učenja odabire sve moguće odluke radeći odstupanje od jednog koraka i primjenjuje odabranu politiku u postupku *rollout* za svaku moguću odluku dovoljno puta dok ne dođe do kraja. Odstupanje od jednog koraka procjenjuje trošak za pojedinu odluku samo uzimajući tu odluku u obzir, nastavljajući niz odluka postupkom *rollout*. Odstupanje od dva koraka bi iz stanja R procjenjivalo trošak za odluku tako da nakon izvedene odluke dolaskom u novo stanje izvrši sve moguće odluke u tom stanju i onda za svaku izvrši ostatak postupkom *rollout*. Inačica s odstupanjem od dva koraka bi bolje aproksimirala trošak, ali bi vremenski bila zahtjevnija – odstupanje od jednog koraka zahtijeva izvršavanje postupka *rollout* za svaku odluku, odstupanje od dva koraka zahtijevalo bi izvršavanje postupka *rollout* za svaku moguću odluku nakon što se izvrši svaka moguća u stanju R, što vodi do kvadratne vremenske složenosti u ovisnosti o broju mogućih odluka. Na kraju izvršavanja postupka *rollout* pomoću potpunog niza odluka $y_z \in \mathcal{Y}$ dobiva se informacija o gubitku i ona se koristi kao procjena dobrote svake odluke. Nakon postupka moguće je zaključiti da se gornjom odlukom koja odstupa od odabranog srednjeg stanja može gubitak smanjiti za 1. Primjer preuzet iz (Chang et al., 2015b, str. 3)

Rollout → / ↓ Rollin	**Referentna**	**Mješovita**	**Naučena**
Referentna	Nekonzistentna redukcija		
Naučena	Nije lokalno optimalna	Dobra	Podržano učenje

Slika 3.2: Rezultati teoretske analize načina učenja koristeći *rollin* i *rollout*. Ovisno o odabiru *rollin* i *rollout* para naučena politika može biti nekonzistentna ili bez svojstva lokalne optimalnosti, problem učenja politike se može svesti na podržano učenje ili naučena politika ipak može biti konzistentna i lokalno optimalna. Mješovita politika je politika kod koje se nekom vjerojatnošću bira referentna ili naučena. Tablica preuzeta iz (Chang et al., 2015b, str. 4)

da će, ako je u prostoru odluka moguće raditi lokalno uspinjanje na vrh (engl. *local hill climbing*), naučena politika biti konstantno lošija od dane referentne politike ili će biti bolja od referentne politike. Takvu garanciju pristupi prije algoritma LOLS nisu imali. Problem postoji čak i kod metode učenja pretraživanja SEARN (Chang et al., 2015b). Navedena karakteristika nije pretjerano korisna ako je referentna politika loša, u tom slučaju učenje se odvija kao da referenta politika ne postoji (informacija koju ona daje je beskorisna). Ako je lako raditi uspinjanje na vrh u prostoru odluka, onda bi na problemu jednako dobro radilo i podržano učenje.

Da bi se izbjeglo ponavljanje nekih značajki, prednosti i ciljeva algoritama učenja pretraživanja njihove karakteristike detaljnije su analizirane u zasebnim potpoglavljima (3.4).

3.4. Pristupi

U nastavku slijedi opis metoda učenja pretraživanja. Pristupi opisani u ovom poglavlju zahtijevaju sljedeće.

- **Dobru referentnu politiku.** Svi pristupi – osim LOLS – zahtijevaju optimalnu referentnu politiku inače može doći do nakupljanja pogreške (engl. *compounding error*) i nekonzistentne redukcije;
- **Dobar binarni klasifikator.** Za učenje pretraživanja potrebno je efikasno rješenje za problem višerazredne klasifikacije osjetljive na trošak. Višerazredni klasifikator trebao bi za primjere kod kojih je definiran veliki trošak pokušati njih pravilno klasificirati. Ako se inače minimizira točnost klasifikatora, onda bi se kod klasifikatora osjetljivog na trošak trebao minimizirati ukupan zbroj troškova svih primjera. U tom slučaju moguće je da će se smanjiti točnost na primjerima

za koje je trošak mali, ali povećati na onima gdje je trošak velik. Koristeći težinsku redukciju OVO ili OVA (engl. *weighted all-pairs or one-against-all*) iz (Beygelzimer et al., 2005a,b) u kombinaciji s *costing* algoritmom iz (Zadrozny et al., 2003) moguće je koristiti bilo koji binarni klasifikator bez korigiranja izvornog koda klasifikatora. Moguće je koristiti i pravi višerazredni klasifikator osjetljiv na trošak bez navedenih tehnika redukcije;

– **Dobro definiranu funkciju gubitka.** Ona može biti definirana na cijeloj strukturi koju predviđamo. Na primjer, može se između predviđenog i pravog stabla računati F_1 gubitak. Između prevedenog i referentnog prijevoda može se koristiti BLEU mjera (engl. *bilingual evaluation understudy*, BLEU). Navedene funkcije gubitka nemaju dekompoziciju preko strukture odluka. Moguće je koristiti i funkcije poput Hammingovog gubitka.

3.4.1. SEARN

Daumé III (2006) u svojoj doktorskoj disertaciji daje doprinos otkrićem prve metode učenja pretraživanja SEARN (engl. *search* + *learn*, SEARN) – metoda kojom je problem strukturnog predviđanja i učenja uspješno reduciran na binarnu klasifikaciju. Metodu primjenjuje na mnoštvo problema u području obrade prirodnoga jezika:

– označavanje vrste riječi,

– prepoznavanje imenovanih entiteta,

– plitko parsanje (engl. *shallow parsing*),

– združeno označavanje vrste riječi i plitko parsanje,

– prepoznavanje i praćenje entiteta (uključuje razrješavanje koreferenci),

– sažimanje teksta koristeći više dokumenata i

– poravnavanje teksta na engleskom i španjolskom koristeći latentne varijable (engl. *text alignment with hidden variables*).

Predložio je primjenu na strojno prevođenje i ovisnosno parsanje koja je u okviru učenja pretraživanja nedavno ostvarena (He et al., 2015; Chang et al., 2015a).

Pseudokod 3.1 prikazuje algoritam SEARN. Algoritam radi više prolaza preko skupa za učenje određen brojem P. Za svaki primjer $\{x_i, y_i\}$ dolazi do nekog stanja s_t koristeći odabranu politiku za *rollin*. Ona je pri prvom prolazu jednaka referentnoj politici, a u ostalim prolazima ta politika je stohastička mješavina između politika naučenih u svakom prolazu i referentne politike – kod problema označavanja vrste riječi rezultat

Algoritam 3.1 Učenje + Pretraživanje (SEARN)

Potrebno: Skup podataka $\{x_i, y_i\}_{i=1}^{N}$ uzet iz distribucije $\mathcal{D}$, $\beta \geq 0$.

Inicijaliziraj politiku $\hat{\pi}_0 = \pi^*$.
za svaki $I \in [0, 1, 2, \ldots, P)$
  za svaki $i \in \{1, 2, \ldots, N\}$
    Inicijaliziraj $\Gamma = \emptyset$. ▷ skup primjera osjetljivih na trošak.
    za svaki $t \in \{0, 1, 2, \ldots, T_i - 1\}$
      Primijeniti t puta politiku $\pi^{\text{in}} = \hat{\pi}_I$ i stići do s_t. ▷ *Rollin*.
      za svaki $a \in A(s_t)$
        Neka je $\pi^{\text{out}} = \hat{\pi}_I$.
        Procjeni trošak $c_{i,t}(a)$ koristeći $T - t - 1$ puta π^{out}. ▷ *Rollout*.
      kraj petlje
      Generiraj vektor značajki $\mathbf{\Phi}(x_i, s_t)$.
      Postavi $\Gamma = \Gamma \cup \{\langle c_{i,t}, \mathbf{\Phi}(x_i, s_t)\rangle\}$.
    kraj petlje
  kraj petlje
  $\hat{\pi}' \leftarrow$ NAUČI(Γ).
  $\hat{\pi}_{I+1} \leftarrow \beta\hat{\pi}' + (1 - \beta)\hat{\pi}_I$.
kraj petlje
vrati $\hat{\pi}_P$ bez π^*

primjene odabrane politike za *rollin* je niz oznaka, a stanje s_t je riječ u rečenici na mjestu t koja još nije označena. Linija 16 pokazuje izračun nove politike. Izračun ne radi interpolaciju naučenih težina, nego samo prilagođava vjerojatnosti odabira do tada naučenih politika. Vjerojatnost je veća za najnoviju naučenu politiku i polako se smanjuje s brojem prolaza (množi se s $1 - \beta$ gdje je $\beta < 1$). Za procjenu troška donošenja odluke na stanju s_t (npr. odluka je određivanje vrste riječi na trenutnoj riječi) za svaku moguću odluku u postupku *rollout* koristi se politika koja je također stohastička mješavina (linija 7 i 8). Ta se politika primjenjuje do kraja strukture koja se obrađuje i na kraju se izračuna dobiveni gubitak (linija 9). Nakon prošlog postupka za svaku moguću odluku kreira se primjer za učenje osjetljiv na trošak i stavlja se u skup. Nakon što je postupak kreiranja primjera gotov nauči se nova politika (linija 15). Višerazredni klasifikator koji se koristi mora biti osjetljiv na trošak, a moguće je koristiti bilo koji binarni klasifikator uz nekoliko izmjena na skupu za učenje. Jedna od takvih izmjena može biti uzorkovanje primjera s obzirom na naveden trošak. Cilj je izgraditi skup za učenje koji će sadržavati manje primjera s malim troškom – davajući time veći naglasak na one s velikim troškom i indirektno minimizirati klasifikacijski trošak. Takav skup može se izgraditi vjerojatnosnom interpretacijom troška (primjeri s većim troškom bi trebali biti vjerojatniji u reduciranom skupu). Prvi vidljivi problem je taj što se skup za učenje smanjuje. Zadrozny et al. (2003) opisuju kako pravilno izvesti postupak za binarnu klasifikaciju, a Beygelzimer et al. (2005a,b) opisuju prilagodbu za višerazrednu klasifikaciju. Nakon učenja, zaključivanje se svodi na korištenje višerazrednog klasifikatora za svako stanje. Ako želimo da donošenje trenutne odluke ovisi o prethodnim odlukama i značajkama iz budućih stanja, onda je potrebno tijekom učenja i korištenja naučenih politika samo dodati te odluke i značajke u vektorsku reprezentaciju trenutnog stanja (tijekom primjene stohastičke politike za vrijeme *rollin* i *rollout* postupka ta reprezentacija mora biti prisutna jer se kod primjene koristi odabrani višerazredni klasifikator).

Ovaj algoritam ima vremensku složenost učenja $O(PNT^2Mk)$ – gdje je P broj prolaza, N broj primjera, T prosječna duljina niza odluka koji izgrađuju $y \in \mathcal{Y}$, M broj različitih odluka za svako stanje s_t i k broj odluka kojima se uvjetuje trenutna (svodi se samo na dodavanje k prijašnjih odluka u vektorsku reprezentaciju stanja s_t). Vremenska složenost zaključivanja za jedan primjer je $O(TMk)$. U procjenu učenja nije uključeno vrijeme učenja klasifikatora generiranim skupom (ako je moguće model učiti primjer po primjer (engl. *online*), onda trošak ovisi o broju primjera i njihovoj vektorskoj reprezentaciji). Kod procjene zaključivanja pretpostavilo se da klasifikatoru treba $O(M)$ vremena za određivanje točne odluke, a tipična složenost

klasifikatora u tom slučaju je $O(M \cdot d)$ – gdje je d duljina vektorske reprezentacije. Ako postoji jednostavna dekompozicija funkcije gubitka, onda složenost učenja može pasti na $O(PNTMk)$. Nije potrebno izvršavati *rollout* kad je parcijalan gubitak za trenutnu odluku na raspolaganju (npr. kod označavanja vrste riječi tijekom izvršavanja svih odluka u liniji 7 nije potrebno izvršiti *rollout* jer je gubitak na pojedinoj odluci 1 ako je netočna ili 0 ako je točna). Postoji niz implementacijskih detalja koji mogu, čak i kod nemogućnosti dekompozicije funkcije gubitka, smanjiti vremensku složenost učenja na $O(PNTMk)$ (Chang et al., 2014).

Za potrebe učenja, SEARN zahtijeva da je referentna politika optimalna (definicija 3 u potpoglavlju 3.2). Ako politika nije optimalna, onda će učenje biti otežano. U slučaju bolje procjene troška određene odluke, potrebno je izvršiti *rollout* nekoliko puta kako bi povećali vjerojatnost otkrića točnog niza odluka koji minimizira trošak. Primjer zadataka za koje nije lako izgraditi optimalnu politiku su ovisnosno parsanje i strojno prevođenje. Za ovisnosno parsanje nije jednostavno znati kako dovršiti stablo tako da se minimizira gubitak s obzirom na to da je došlo do niza pogrešaka u izgradnji parcijalnog stabla. Kod strojnog prevođenja teško je znati kako odrediti trenutnu riječ s obzirom na polovično preveden dokument. U slučaju da je dokument pogrešno preveden, ima više smisla staviti neku drugu riječ umjesto one koja je u zlatnom primjeru prisutna na toj poziciji. Kako bi se problem razriješio u okviru algoritma SEARN korišten je Monte-Carlo algoritam za procjenu troška. Taj postupak usporava učenje, ali omogućava učenje bez optimalne politike. Problem je riješen u (Chang et al., 2015b) i postupak je opisan u potpoglavlju 3.4.4. Posljednji nedostatak algoritma je taj što se moraju čuvati sve naučene politike, pristupi opisani u sljedećim potpoglavljima nemaju to ograničenje.

Daumé III za potrebe svoje disertacije isprobava različite binarne klasifikatore: (1) perceptron, (2) logističku regresiju, (3) SVM s linearnom jezgrom i (4) SVM s kvadratnom jezgrom. Izbor klasifikatora ovisi o zadatku. Ako je broj primjera ogroman, onda je bolje koristiti perceptron, logističku regresiju ili SVM s linearnom jezgrom. Ako je broj primjera mali, onda bi korištenje binarnog klasifikatora SVM omogućilo što veće margine između odluka i povećalo točnost (eksperimentalno utvrđeno u disertaciji). Moguće je koristiti i neuronske mreže s više slojeva, ali trenutno nije u literaturi zabilježeno korištenje. Odabire binarnih klasifikatora Daumé III uspoređuje sa strukturnim metodama opisanima u poglavlju 2 i pokazuje da SEARN ima istu ili bolju učinkovitost na odabranim zadacima.

Daumé III tvrdi da, u kontekstu algoritama za strukturno predviđanje, SEARN leži između algoritama koji koriste globalnu normalizaciju, kao što su M^4 i CRF, i onih

koji koriste lokalnu normalizaciju, opisani u (Punyakanok i Roth, 2001). Razlika između SEARN i globalnih algoritama je u načinu na koji rješavaju nesigurnost. Kod globalnih algoritama, algoritam pretraživanja koristi se za vrijeme zaključivanja kako bi se razriješila nesigurnost. Kod algoritma SEARN, trošak za svaku odluku koji je prisutan tijekom učenja razrješava nesigurnost. Oba pristupa razlikuju se od lokalnih algoritama kod kojih se nesigurnost uopće ne razrješava.

Sa šire perspektive strojnog učenja, SEARN je pokazao povezanost između podržanog učenja i strukturnog predviđanja. Strukturno predviđanje može se vidjeti kao problem podržanog učenja u kojem su sva opažanja (sve prave odluke) vidljive na početku.

Sve navedeno u zadnja dva paragrafa vrijedi i za ostale metode učenja pretraživanja navedene u ovom poglavlju. Daumé III postavlja otvorena pitanja vezana uz broj potrebnih primjera koji je potreban za učenje i što raditi bez prisustva optimalne politike. Odgovori na ta pitanja dani su algoritmom LOLS koji je opisan u poglavlju 3.4.4 (Chang et al., 2015b).

SEARN se može opisati, koristeći terminologiju uvedenu u potpoglavljima 3.1, 3.2 i 3.3, kao konzistentna (ako je referentna politika optimalna) redukcija problema združenog predviđanja na problem binarne klasifikacije, gdje se za postupke *rollin* i *rollout* koristi politika koja je stohastička mješavina između naučene i referentne politike.

3.4.2. DAGGER

Ross et al. (2011) predstavljaju metodu učenja pretraživanja DAGGER (engl. *dataset aggregation*, DAGGER), koja je po svojim karakteristikama bolja od algoritma SEARN. Demonstriraju uspješnost na dva zadatka:

- upravljanje automobilom u 3D trkaćoj igri (*Super Tux Kart*) i
- igranje igre *Super Mario*.

Kao optimalnu politiku za prvi zadatak koriste ljudskog stručnjaka, a za drugi koriste pretraživanje u širinu (engl. *breadth-first-search*, BFS) ograničeno dubinom. Algoritam ima nekoliko razlika u usporedbi s algoritmom SEARN:

- učenje je moguće raditi primjer po primjer (engl. *online*);
- za *rollin* se koristi naučena ili referentna politika (na prvom primjeru postoji samo referenta), dok SEARN koristi mješavinu referentne i svih naučenih kroz više prolaza;

– kod algoritma SEARN mješavina se primjenjuje za svako stanje, a kod algoritma DAGGER prije svakog primjera odabire se referentna ili naučena politika što rezultira time da se na nekim primjerima uči koristeći samo referentnu politiku za *rollin*, a na nekima samo naučenu. Unatoč tome što je algoritam moguće učiti primjer po primjer preporučuje se više prolaza kroz skup za učenje;

– *rollout* je eliminiran pretpostavljajući da se za svaku odluku može izračunati njen trošak, ali može se koristiti referentna politika ako se trošak ne može izračunati (koristeći postupak *rollout*);

– nije potrebno držati više naučenih politika, nego se koristi samo politika naučena do trenutnog primjera.

Autori su predstavili, uz eksperimentalnu potvrdu superiornosti nad algoritmom SEARN, i teoretsku analizu kojom pokazuju da je takva redukcija konzistentna.

DAGGER se može opisati, koristeći terminologiju uvedenu u potpoglavljima 3.1, 3.2 i 3.3, kao konzistentna (ako je referentna politika optimalna) redukcija problema združenog predviđanja na problem binarne klasifikacije, gdje se za *rollin* koristi referentna ili naučena politika (odabir se radi stohastički prije postupka *rollin*, a ne na svakom stanju kao kod algoritma SEARN), a za *rollout* referenta politika (ako postoji dekompozicija funkcije gubitka, onda se *rollout* ne bi izvršavao što je ekvivalentno kao da se koristi referentna politika kod koje funkcija gubitka ima dekompozciju na niz odluka).

3.4.3. AGGREVATE

Ross i Bagnell (2014) predstavljaju metodu učenja pretraživanja AGGREVATE (engl. *aggregate values to imitate*, AGGREVATE), koja je proširenje algoritma DAGGER. Proširenje se svodi na to da je moguće, uz referentnu politiku koja se koristi za odabir sljedećih poteza, tijekom učenja koristiti i stručnjaka koji želi što manje biti u interakciji s algoritmom učenja. Tj. odabir odluke u kojoj bi stručnjak trebao vrednovati preostali niz odluka ima određeni trošak. Cilj je istovremeno naučiti dobru politiku, ali pri učenju što manje koristiti stručnjaka. U stanju u kojem je skupo procijeniti gubitak odluke (ili informacija o trošku ne postoji) može se pozvati stručnjak koji će vratiti točnu odluku i njen trošak.

Dobar primjer koji opisuje primjenu ove metode je povećanje klikovnog postotka (engl. *click-through rate*, CTR). Stručnjak je u ovom slučaju dizajner koji pokušava prilagoditi web-stranicu posjetitelju, a cilj je povećati broj klikova na oglase koje korisnik vidi. Postupak kojim bi se potencijalno povećali klikovi je mijenjanje izgleda

web-stranice (pozicioniranje oglasa, veličina oglasa, sadržaj uz koji se oglas pojavljuje i dr.). Želi se izbjeći mogućnost da korisnik zbog ogromnog broja nasumičnih promjena odluči ne posjećivati stranicu. U tom je slučaju potrebno vršiti eksperiment promjene stranice što manje puta i pratiti kako promjene utječu na klikovni postotak (nakon nekog vremena od trenutka promjene se izračuna količina promjene CTR).

Drugi primjer je interaktivno označavanje skupa podataka. Pretpostavimo prisutnost malog označenog skupa podataka. Cilj je naučiti model koji će imati dobru generalizaciju na tom skupu. Za algoritam AGGREVATE odaberemo veći neoznačeni skup. Cilj algoritma je istovremeno minimizirati broj interakcija sa stručnjakom i gubitak na malom označenom skupu podataka (postojao bi i treći skup na kojem bi provjerili stvarnu mogućnost generalizacije).

Algoritam ima nedostatak u kojem je moguće da stručnjak daje preoptimistične procjene nakon izvršene odluke. Autori ilustriraju to primjerom gdje se uči vožnja automobila. Stručnjak bi vjerojatno odabrao užu, ali kraću cestu, za dolazak do cilja, unatoč postojanju duže, ali sigurnije ceste koja vodi do istog cilja. Ako naučena politika na užoj cesti dođe u stanje za koju ne radi dobro predviđanje troška sljedeće odluke, onda je moguće da će se vozilo skotrljati niz liticu. Navedeni problem može se riješiti tako da se poveća istraživanje dolazaka do svih stanja (problem je ekvivalentan pristranosti oznakama).

Ovaj algoritam je interaktivna verzija algoritma DAGGER, stoga je za učenje iz podataka dovoljan potonji. AGGREVATE je opisan radi usporedbe s algoritmom LOLS.

3.4.4. LOLS

Lokalno optimalno učenje pretraživanja (engl. *locally optimal learning to search*) najnoviji je pristup i primjer je redukcije združenog učenja koja je konzistentna s najmanje pretpostavki o raspoloživoj politici. Ovu metodu moguće je koristiti za sve zadatke koji su opisani u prethodnim poglavljima.

Chang et al. (2015b) daju teoretsku i empirijsku potvrdu te redukcije združenog pretraživanja. Izvedeni su pokusi na tri zadatka: višerazredna klasifikacija osjetljiva na trošak (engl. *cost-sensitive multiclass classification*), označavanje vrste riječi i ovisnosno parsanje. Zaključak je da algoritam LOLS bez obzira na kvalitetu referentne politike može postići rezultate koji su usporedivi s optimalnom politikom ili, u slučaju da referentna politika nije optimalna, bolji od referentne politike ako prostor pretraživanja dopušta uspinjanje na vrh.

Iz pseudokoda algoritma 3.2 može se primijetiti velika sličnost s algoritmom 3.1.

Algoritam 3.2 Lokalno optimalno učenje pretraživanja (LOLS)

Potrebno: Skup podataka $\{x_i, y_i\}_{i=1}^N$ uzet iz distribucije $\mathcal{D}$ i $\beta \geq 0$.

Inicijaliziraj politiku $\hat{\pi}_1$.
za svaki $i \in \{1, 2, \ldots, N\}$
    Generiraj referentnu politiku π^{ref} temeljenu na y_i.
    Inicijaliziraj $\Gamma = \emptyset$. ▷ skup primjera osjetljivih na trošak.
    za svaki $t \in \{0, 1, 2, \ldots, T_i - 1\}$
        Primijeniti t puta politiku $\pi_i^{\text{in}} = \hat{\pi}_i$ i stići do s_t. ▷ *Rollin.*
        za svaki $a \in A(s_t)$
            Neka je $\pi_i^{\text{out}} = \pi^{\text{ref}}$ s vjerojatnošću β, inače $\hat{\pi}_i$.
            Procjeni trošak $c_{i,t}(a)$ koristeći $T - t - 1$ puta politiku π_i^{out}. ▷ *Rollout.*
        kraj petlje
        Generiraj vektor značajki $\mathbf{\Phi}(x_i, s_t)$.
        Postavi $\Gamma = \Gamma \cup \{\langle c_{i,t}, \mathbf{\Phi}(x_i, s_t)\rangle\}$.
    kraj petlje
    $\hat{\pi}_{i+1} \leftarrow$ Naučи$(\hat{\pi}_i, \Gamma)$.
kraj petlje
Vrati usrednjenu politiku preko svih $\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N$.

Razlike su sljedeće:

- model u slučaju algoritma LOLS moguće je učiti primjer po primjer (*online*);
- SEARN zahtijeva čuvanje naučenih politika za svaki prolaz preko skupa za učenje jer se *rollin* i *rollout* izvršavaju koristeći mješavinu politika, dok LOLS koristi samo trenutno naučenu politiku i referentnu politiku;
- SEARN koristi mješovitu politiku za *rollin* i *rollout* te miješanje radi na svakom stanju (engl. *mix-per-state*). LOLS miješanje radi samo na početku postupka *rollout* (engl. *mix-per-roll*), a za *rollin* uvijek koristi naučenu politiku;
- u slučaju algoritma LOLS, nije potrebno da je referentna politika optimalna, a za SEARN je taj uvjet obavezan inače postoji mogućnost da naučena politika neće imati svojstvo generalizacije.

Ova varijanta učenja pretraživanja koristi se za rješavanje problema u okviru ovog rada.

Autori pokazuju i mogućnost interaktivnog učenja koje je slično metodi AGGREVATE. Razlika je u tome da za algoritam LOLS nije potreban stručnjak (optimalna politika). Jedini uvjet je da funkcija gubitka vraća pravu vrijednost s obzirom na odabrane odluke. U kontekstu problema povećanja CTR (potpoglavlje 3.4.3) nije potrebno imati dizajnera. Kod problema u kojem želimo interaktivno označavati nove podatke potrebno je stručnjaka pitati koliki je gubitak za označen primjer. U tom slučaju stručnjak može biti program koji vraća gubitak na malom skupu podataka. Očigledno je problem učenja, u slučaju bez stručnjaka, sveden na podržano učenje. Zanimljiv primjer je prisutnost referentne politike koja radi dobro, a cilj je naučiti novu politiku koja radi isto ili bolje. Algoritam LOLS vođen starom referentnom politikom može istu naučiti ili postati bolji. Čitatelja se upućuje na Chang et al. (2015b) za matematički dokaz.

LOLS se može opisati, koristeći terminologiju uvedenu u potpoglavljima 3.1, 3.2 i 3.3, kao konzistentna (bez obzira kakva je referenta politika) redukcija problema združenog predviđanja na problem binarne klasifikacije, gdje se za *rollin* koristi naučena politika, a za *rollout* stohastički odabir između naučene i referentne politike (odabir se radi prije postupka *rollout*, a ne na svakom stanju kao kod algoritma SEARN).

3.5. Pristranost oznakama

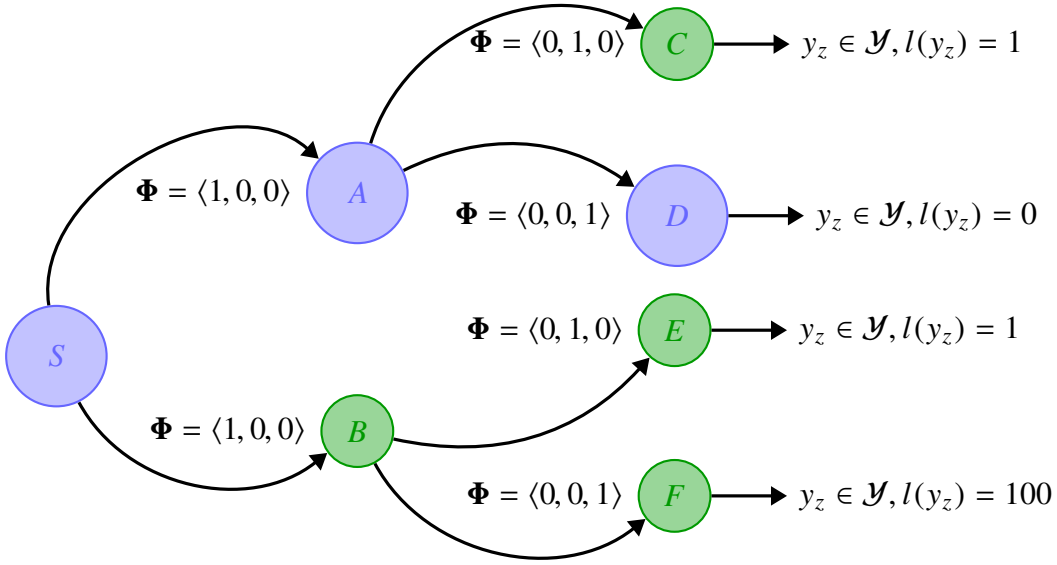
U strukturnom učenju moguća je pojava problema pristranosti oznakama (engl. *label bias*). Pojavljuje se kod modela koji moraju donositi niz odluka, ali zbog nedovoljno informacija prijašnje odluke donesene su pogrešno i uzrokuju akumulaciju pogreške

(engl. *compounding error*). Kako model dobro radi na skupu za učenje, problem koji se pojavljuje imenovan je zbog pristranosti prema oznakama u skupu za učenje. Modeli koji su lokalno normalizirani (Markovljev model maksimalne entropije) pate od pristranosti, dok globalno normalizirani modeli (uvjetna slučajna polja) nemaju taj problem. Bilo bi prikladno da je model u stanju ispraviti neku prijašnju pogrešku s obzirom na nove spoznaje o ulazu, ali to s lokalno normaliziranim modelima nije jednostavno. Postoji mogućnost dodavanja većeg broja značajki koje bi lokalne odluke uzimale u obzir i/ili *beam* pretrage. Eksperimentalno je utvrđeno da se mogu naučiti lokalno normalizirani modeli koji dobro rade, ali svejedno ne mogu dostići učinkovitost globalno normaliziranih modela (Liang et al., 2008). Razlika između globalne i lokalne normalizacije je ta što prethodni za pojedinu odluku koristi informaciju o globalnom gubitku preko cijele strukture, dok potonji samo informaciju o lokalnom gubitku – modeli koji se uče na jedan ili drugi način minimiziraju istu funkciju gubitka, ali pitanje je koja će normalizacija rezultirati konzistentnim modelom.

Lafferty, McCallum, i Pereira (2001) i Bottou (1991) definiraju i detaljno razrađuju problem pristranosti oznakama. Za noviji pregled o samom problemu čitatelja se upućuje na (Andor et al., 2016).

Lafferty, McCallum, i Pereira (2001) eksperimentalno potvrđuju pojavu pristranosti, a kao jedno od mogućih rješenja navode stapanje stanja u jedno. Navode da je takva operacija specijalni slučaj *determinizacije* koja, za slučaj težinskog stroja s konačnim brojem stanja, nije uvijek moguća, a kad je moguća, može dovesti do kombinatorijalne eksplozije. Za problem označavanja vrste riječi konkretan postupak determinacije bio bi odlučivanje u jednom koraku oznaku za trenutnu i sljedeću riječ (ili više sljedećih riječi). U tom slučaju bi se izbjegla situacija u kojem se pogreška na pojedinoj riječi propagira na odluke koje slijede. Postoji mogućnost da je za problem potrebno spojiti sva stanja u jednu diskretnu odluku i time izbjeći pristranost, ali tako nešto nije vremenski isplativo i zahtijeva veći skup za učenje. Drugo je rješenje početi treniranje s potpuno povezanim modelom gdje bi učenje trebalo otkriti temeljnu strukturu koja je prisutna u podlozi, ali šteta je unaprijed odbaciti znanje o samoj strukturi – kao što je lanac odluka kod označavanja vrste riječi.

Na slici 3.3 prikazan je jednostavan primjer donošenja odluka. Može se promatrati kao problem označavanja niza gdje je prva oznaka S dana, a za sljedeća dva koraka potrebno je odabrati između druge dvije oznake. Krenuvši iz početnog stanja S potrebno je odlučiti kojim bridom nastaviti dalje. Iz podataka točan niz odluka rezultira nizom $y_z = SAD$. Prikazani vektori značajki $\mathbf{\Phi}$ su reprezentacija stanja i opažanja pomoću koje bi klasifikator trebao napraviti odluku. Očito klasifikator ne može razlikovati



Slika 3.3: Prikaz problema pristranosti oznaka. Primjer preuzet iz (Daumé III, 2006, str. 31).

između stanja A i B te odluku bira nasumično. Da su odluke spojene (determinizacija) u AC, AD, BE i BF vjerojatno bi bilo moguće odabrati onu točnu. U ovom slučaju korisno je razmotriti kako bi Markovljev model maksimalne entropije naučio niz odluka. Prvo bi se konstruirao klasifikacijski problem između A i B, a nakon toga između C i D. Ako se koristi običan model maksimalne entropije za ove klasifikacijske probleme, onda bi naučen vektor težina bio $\mathbf{w} = \langle 0, 0, -1 \rangle$.

Ako se sada izvede pretraga koristeći naučene težine, moguće je da postupak neće otkriti točan niz, ali očekivanje je da barem proizvede rezultat koji minimizira očekivani gubitak. U prvom koraku potrebno je odabrati A ili B. Kako naučene težine ne omogućavaju razlikovanje između ta dva izbora – skalarni produkt $\mathbf{w} \cdot \mathbf{\Phi} = 0$ – potrebno je napraviti nasumičan odabir. Ako je odabran A, onda će u drugom koraku biti odabran E i gubitak će iznositi 0. Suprotno, ako je odabran B, onda će u drugom koraku biti odabran F. Očekivani gubitak tada iznosi 50.5, a ne 0.5 – toliko bi iznosio da se umjesto F odabere E. Ovaj primjer demonstrira razlog pojave pristranosti oznakama i nedostatak modela koji koriste lokalnu normalizaciju za učenje. Razlog zašto nije minimiziran očekivan gubitak je taj što se model trenira na optimalnom putu koji postoji u podacima. Ako nikad nije odabrano stanje B u podacima, onda se težine uopće neće ažurirati za razlikovanje stanja E i F. Kääriäinen (2006) pokazuje da donošenje krivih lokalnih odluka može akumulirati Hammingov gubitak otprilike jednak polovici duljine niza odluka. Očekivani gubitak koji ima linearnu ovisnost o duljini niza odluka nije zadovoljavajuć.

Korisno je pogledati kako metode učenja pretraživanja razrješavaju problem, ali bez potrebe dinamičkog programiranja ili aproksimacije particijske funkcije. Kod algoritma SEARN (alg. 3.1) prvi će prolaz rezultirati težinama kao i kod Markovljevog modela maksimalne entropije. Drugi prolaz preko podataka koristit će interpolaciju naučene politike i optimalne politike, što znači da postoji mogućnost da se odabere stanje B, a onda dodatno nauči razlikovanje između E i F. Kod algoritma LOLS (alg. 3.2) uči se koristeći naučenu politiku od samog početka (početna politika je nasumična), što garantira nailaženje na stanja koja ne postoje u podacima za učenje, a na kojima onda referentna politika može naučiti razlikovati bolje odluke od lošijih. Ako referentna politika nije dovoljno dobra, onda će sama procjena o gubitku tijekom *rollout* faze utjecati na razlikovanje između odluka. Detaljnije razmatranje postupka učenja izloženo je u potpoglavlju 3.3.

Andor et al. (2016) pretpostavljaju da je problem ovisnosnog parsanja vrlo težak upravo zbog pristranosti oznakama. Koristeći sve pristupe za koje se pokazalo da dobro rješavaju problem pristranosti oznakama – velika količina značajki, bogata vektorska reprezentacija koju omogućava obična višeslojna mreža bez povratnih veza, globalna normalizacija i *beam* pretraga – ostvarili su najbolji rezultat na tom zadatku.

4. Primjena na konkretan zadatak

U ovom poglavlju opisane su primjene metoda učenja pretraživanja na zadatke u obradi prirodnog jezika. Razmatrane probleme moguće je riješiti na više načina, gdje svaki ima svoje prednosti i mane. Dana je usporedba s metodama koje su korištene za rješavanje tih problema koja demonstrira nadmoć metoda učenja pretraživanja. Metode učenja pretraživanja omogućuju da istraživač u području obrade prirodnoga jezika ne treba biti istovremeno i istraživač u području strojnog učenja (ne treba znati kako efikasno implementirati algoritme učenja i zaključivanja kod raznih modela). Priloženi pseudokodovi koji slijede trebali bi potvrditi izjavu u prošloj rečenici.

4.1. Označavanje vrste riječi

Označavanje vrste riječi jedan je od jednostavnijih problema u obradi prirodnog jezika i shodno tome ima vrlo jednostavnu implementaciju u okviru učenja pretraživanja. Problem je ilustriran na slici 4.1. Za rečenicu u nekom jeziku (na slici je hrvatski) potrebno je svakoj znački dodijeliti oznaku vrste riječi kao što su imenica, glagol, pridjev i dr. Srednji redak pokazuje oznake koje daju više detalja o gramatičkim značajkama riječi kao što su rod i množina imenice, glagolsko vrijeme i dr. Algoritam 4.1 pokazuje jednostavnost zadatka u okviru učenja pretraživanja.

Algoritam je skoro jednak algoritmu koji provjerava koliko je riječi u rečenici točno označeno. Točna oznaka je prisutna za svaku riječ u rečenici i proslijeđena funkciji Odluči, koja vraća odluku trenutne politike – politika može biti naučena ili referentna. U slučaju da je politika referentna, funkcija vraća upravo proslijeđenu oznaku. Nakon

R	V	N	R	Z	C	C	N	V	R	Z
Rgc	Vmn3pf	Ncfpnn	Rgp	Z	Cs	Cs	Ncfpnn	Vmn3pf	Rgp	Z
Gore	gore	gore	gore	,	nego	što	gore	gore	dolje	.

Slika 4.1: Rečenica s oznakama vrste riječi. Za ovu rečenicu ovo nije jedini mogući niz oznaka. Oznake R, V, N, Z i C su za prilog, glagol, imenicu, znakove interpunkcije i veznike.

Algoritam 4.1 Označavanje vrste riječi u radnom okviru L2S.

```
1: funkcija IZVRŠI(riječi)
2:     izlaz ← []
3:     za n ← 1 do DULJINA(riječi)
4:         ref ← riječi[n].točna_oznaka
5:         izlaz[n] ← ODLUČI(riječi[n], ref, izlaz[:n-1])
6:     kraj petlje
7:     GUBITAK(# izlaz[n] ≠ riječi[n].točna_oznaka)
8:     return izlaz
9: kraj funkcije
```

što je cijeli izlaz predviđen objavljuje se gubitak koji je u ovom slučaju broj riječi koje su netočno označene. Algoritam tijekom učenja u pozadini mijenja politike i pokreće funkciju Izvrši dovoljno puta dok ne nauči referentnu politiku. Algoritam zaključivanja koristi istu funkciju, ali u tom trenutku ne postoji pristup točnoj oznaci tj. što god ona bila neće se uzeti u obzir nego će se zvati naučena politika. Jedna je funkcija dovoljna za algoritam učenja i zaključivanja, a to je slučaj za sve algoritme u L2S okviru.

Zanimljivo je usporediti vremenske složenosti uobičajenih pristupa s ovim. Svi pristupi opisani u poglavlju 2 za zaključivanje koriste Viterbijev algoritam, čija je vremenska složenost $O(TM^k)$, gdje je T duljina rečenice, M broj mogućih oznaka i oznaka ne ovisi o prijašnjim oznakama koje su k ili više pozicija udaljene. U okviru L2S složenost zaključivanja je $O(TMk)$ – ovdje je k broj prijašnjih odluka kojima uvjetujemo trenutnu (k je kod navedenih složenosti jednak, ali je kontekst drugačiji). Iz algoritma 4.1 može se primijetiti da se u funkciju ODLUČI prosljeđuju sve prijašnje oznake. One se pri izgradnji primjera osjetljivog na trošak u algoritmu LOLS dodaju kao značajke. Binarni klasifikatori koji omogućuju redukciju problema višerazredne klasifikacije osjetljive na trošak predviđaju trošak svake pojedine oznake kojih ima točno M. Složenost tog postupka je $O(Md)$ gdje je d duljina vektora značajki. U bilo kojem slučaju zaključivanje je puno brže od Viterbijevog algoritma.

Prednost je što algoritam 4.1 ne treba mijenjati ako se mijenjaju redukcije ispod apstrakcije. Recimo, moguće je koristiti redukciju višerazredne klasifikacije osjetljive na trošak opisanu u (Beygelzimer et al., 2009; Daumé III et al., 2016). Vremenska složenost određivanja razreda opada s $O(M)$ na $O(\log M)$. To bi automatski ubrzalo označavanje vrste riječi i vremenska složenost bi iznosila $O(kT \log M)$.

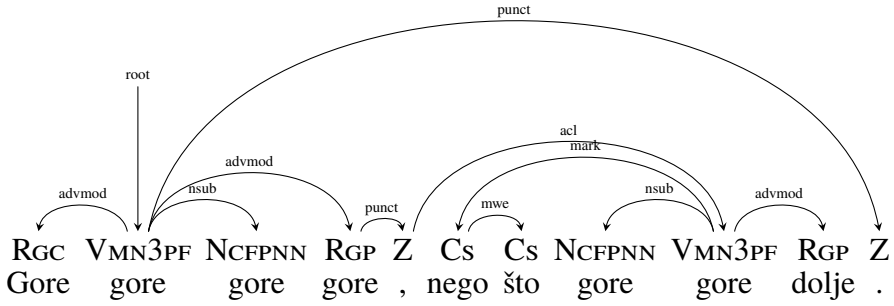
Vremenska složenost algoritma učenja ovisi o parametrima algoritma. U slučaju

algoritma LOLS (alg. 3.2), funkcija Izvrši zove se u liniji 6 i 9, što bi upućivalo da je složenost učenja jednog primjera $O(T^2Mk)$, ali raznim *trikovima* moguće je to smanjiti na $O(TMk)$ – ako funkcija gubitka ima dekompoziciju preko niza odluka, onda *rollout* nije potrebno izvršavati. Za detaljniji opis čitatelja se upućuje na (Chang et al., 2014).

Algoritam 4.2 Označavanje vrste riječi u radnom okviru L2S s više prolaza.

```
1:  funkcija IZVRŠI(riječi, P)
2:      izlaz ← []
3:      za p ← 1 do P
4:          za n ← 1 do DULJINA(riječi)
5:              ref ← PRAVAODLUKA(riječi[n].točna_oznaka, p, odluke[n,p−1])
6:              odluke[n,p] ← ODLUČI(riječi[i], ref, IZVUCISUSJEDE(odluke))
7:          kraj petlje
8:      kraj petlje
9:      pravi_izlaz ← REKONSTRUIRAJIZLAZ(odluke)
10:     GUBITAK(# pravi_izlaz[n] ≠ riječi[n].točna_oznaka)
11:     return pravi_izlaz
12: kraj funkcije
```

Algoritam 4.2 prikazuje alternativnu implementaciju označivača vrste riječi. U ovom slučaju za jednu se rečenicu radi više prolaza u nadi da će prvi prolaz otkriti novu informaciju kojom bi sljedeći mogao popraviti napravljene pogreške. U tom slučaju odluke koje referentna politika vrši nisu samo točne oznake nego i odluka promjene oznake dodijeljene u prošlom prolazu. Funkcija PravaOdluka vraća za novije prolaze odluku koju referentna politika vrši s obzirom na to koja je zapravo točna oznaka. U slučaju da je trenutnoj riječi dodijeljena točna oznaka, odluka bi trebala biti takva da očuva prijašnju odluku. Zbog više prolaza moguće je uključiti predviđanja koja su lijevo i desno od trenutne riječi, a logika je prepuštena funkciji IzvuciSusjede. Na samom kraju funkcija RekonstruirajIzlaz iz donesenih odluka rekonstruira pravi izlaz i ostatak je identičan originalnom algoritmu. Ovakav pristup gotovo je nemoguće napraviti bilo kojom drugom metodom, a da argmax-problem ostane izračunljiv. Sve odluke u navedenoj inačici algoritma i dalje se vrše združeno u slučaju da se susjedi dodaju kao značajke. Algoritam nije ekvivalentan pokretanju označivača vrste riječi više puta gdje drugo i sva sljedeća pokretanja imaju značajku vrste riječi prošlog prolaza.



Slika 4.2: Rečenica s oznakama vrste riječi i ovisnosnim stablom. Za ovu rečenicu ovo nije jedini mogući niz oznaka i jedino moguće stablo. Oznake R, V, N, Z i C su za prilog, glagol, imenicu, znakove interpunkcije i veznike. Oznake usmjerenih bridova ovisnosnog stabla označavaju veze o subjektu glagola ili objektu glagola (subjekt vrši radnju opisanu glagolom na neki objekt) i dr.

4.2. Parsanje ovisnosnog stabla

Na slici 4.2 ilustriran je problem parsanja ovisnosnog stabla. Za rečenicu koja ima označene vrste riječi potrebno je pronaći ovisnosno stablo. Trivijalno rješenje uključivalo bi učenje vjerojatnosti vezanih uz pojedinačne veze i pametnom enumeracijom svih mogućih stabala naći ono koje ima najveću vjerojatnost za tu rečenicu.

Cer et al. (2010) daju pregled metoda parsanja ovisnosnih stabala. Prvi način je iskoristiti postojeće konstituentske parsere i iz njih dobivati ovisnosna stabla, ali sam postupak dobivanja konstituentskog stabla je dugotrajan stoga je bolje razmotriti metode koje direktno izgrađuju ovisnosna stabla.

- Koristeći algoritam CYK za parsanje vremenska složenost iznosi $O(n^5)$, gdje je n duljina rečenice;
- Malo drugačija reprezentacija dopušta brži algoritam (Eisner i Satta, 1999). Vremensku složenost moguće je smanjiti na $O(n^4)$ i dodatno do $O(n^3)$, pretpostavljajući odvojeni korijen (engl. *root*) stabla, što je česta pretpostavka u parsanju ovisnosnog stabla;
- Koristeći programiranje s uvjetima (engl. *constraint programming*) i naivan pristup – pokušavajući između svih parova riječi odabrati odgovarajući brid ovisnosnog stabla – moguće je složenost spustiti na $O(n^2)$ (Covington, 2001);
- Najveći skok je otkriće postupka parsanja linearne $O(n + m)$ vremenske složenosti čiji je razvoj opisan u (Nivre, 2003; Zhang i Nivre, 2011; Bohnet i Nivre, 2012) – n je duljina rečenice, a m veličina stabla (prijašnje metode ovise o

veličini stabla, ali ona nije glavni faktor u vremenskoj složenosti). Ideja je inspirirana tranzicijskim parserom (engl. *transition parser*).[1] U slučaju parsanja stabla s usmjerenim bridovima (engl. *directed tree*) postoje tri odluke koje je potrebno izvršavati – pomak (engl. *shift*), redukcija na lijevo (engl. *reduce left*) i redukcija na desno (engl. *reduce right*). Za svaku odluku koristi se klasifikacijski postupak koji kao značajke koristi trenutno nedovršeno stablo i riječi u rečenici. Ako želimo označiti bridove, onda je potrebno koristiti klasifikacijski postupak kod odluka koje rezultiraju stvaranjem bridova. U okviru učenja pretraživanja implementiran je takav parser (Chang et al., 2015a), a i trenutno najbolji parser koristi identičan pristup (Andor et al., 2016).

Tranzicijski parser potrebno je pravilno implementirati u okviru učenja pretraživanja. Za okvir učenja pretraživanja potrebna je funkcija gubitka i referentna politika. Neki od pristupa ne zahtijevaju da je referentna politika optimalna, ali prisutnost optimalne politike omogućava bržu konvergenciju modela na dobro rješenje. Pitanje je koja je sljedeća odluka optimalna ako je prije izvršen niz odluka u kojima nisu sve odgovarale primjeru za učenje? Goldberg i Nivre (2013) opisuju kako učiniti referentnu politiku optimalnom s obzirom na neke prethodne pogreške u odlučivanju. Referentna politika ima mogućnost prilagodbe i vrlo mali vremenski trošak. Andor et al. (2016) argumentiraju da prisutnost optimalne politike ne rješava jaku prisutnost problema pristranosti oznakama kod parsanja. Slika 4.3 prikazuje kako tranzicijski parser izgrađuje stablo. Moguće je istovremeno donositi odluke o vrsti riječi i o vrsti usmjerenog brida za poboljšanje učinkovitosti modela. U tom slučaju bi operacija Pomak mogla imati pridruženu oznaku vrste riječi, a operacije RedukcijaDesno i RedukcijaLijevo vrstu veze. Naravno, postupak je moguće vršiti združeno tako da se prvo označe vrste riječi, a nakon toga stablo, ali moguće je da prepletenost odluka na sve zadatke dovodi do brže konvergencije.

4.3. Prepoznavanje imenovanih entiteta

Prepoznavanje imenovanih entiteta (engl. *named entity recognition*) jedan je od osnovnih problema u obradi prirodnog jezika. Cilj je u tekstu prepoznati imenovane entitete. Problem je ilustriran na slici 4.4. Rečenicu je potrebno segmentirati na riječi koje bi mogle sadržavati imenovane entitete i nakon toga te segmente prepoznati. Pristup koji se najčešće uzima je redukcija prepoznavanja imenovanih entiteta na označavanje niza.

[1] https://en.wikipedia.org/wiki/Shift-reduce_parser

Akcija	Stog	Međuspremink	Lukovi
	[**K**]	[Gore gore_1 gore_2 gore_3 , nego što gore_4 gore_5 dolje .]	{}
Pomak	[**K** Gore]	[gore_1 gore_2 gore_3 , nego što gore_4 gore_5 dolje .]	{}
RedukcijaLijevo	[**K**]	[gore_1 gore_2 gore_3 , nego što gore_4 gore_5 dolje .]	{(gore_1, Gore)}
Pomak	[**K** gore_1]	[gore_2 gore_3 , nego što gore_4 gore_5 dolje .]	{(gore_1, Gore)}
Pomak	[**K** gore_1 gore_2]	[gore_3 , nego što gore_4 gore_5 dolje .]	{(gore_1, Gore),}
RedukcijaDesno	[**K** gore_1]	[gore_3 , nego što gore_4 gore_5 dolje .]	{(gore_1, Gore), (gore_1, gore_2)}
Pomak	[**K** gore_1 gore_3]	[, nego što gore_4 gore_5 dolje .]	{(gore_1, Gore), (gore_1, gore_2)}
Pomak	[**K** gore_1 gore_3 ,]	[nego što gore_4 gore_5 dolje .]	{(gore_1, Gore), (gore_1, gore_2)}
Pomak	[**K** gore_1 gore_3 , nego]	[što gore_4 gore_5 dolje .]	{(gore_1, Gore), (gore_1, gore_2)}
Pomak	[**K** gore_1 gore_3 , nego što]	[gore_4 gore_5 dolje .]	{(gore_1, Gore), (gore_1, gore_2)}
RedukcijaDesno	[**K** gore_1 gore_3 , nego]	[gore_4 gore_5 dolje .]	{(gore_1, Gore), (gore_1, gore_2), (nego, što)}
Pomak	[**K** gore_1 gore_3 , nego gore_4]	[gore_5 dolje .]	{(gore_1, Gore), (gore_1, gore_2), (nego, što)}
RedukcijaLijevo	[**K** gore_1 gore_3 , nego]	[gore_5 dolje .]	{(gore_1, Gore), . . . , (gore_5, gore_4)}
RedukcijaLijevo	[**K** gore_1 gore_3 ,]	[gore_5 dolje .]	{(gore_1, Gore), . . . , (gore_5, nego)}
Pomak	[**K** gore_1 gore_3 , gore_5]	[dolje .]	{(gore_1, Gore), . . . , (gore_5, nego)}
Pomak	[**K** gore_1 gore_3 , gore_5 dolje]	[.]	{(gore_1, Gore), . . . , (gore_5, nego)}
RedukcijaDesno	[**K** gore_1 gore_3 , gore_5]	[.]	{(gore_1, Gore), . . . , (gore_5, dolje)}
RedukcijaDesno	[**K** gore_1 gore_3 ,]	[.]	{(gore_1, Gore), . . . , (, , gore_5)}
RedukcijaDesno	[**K** gore_1 gore_3]	[.]	{(gore_1, Gore), . . . , (gore_3, ,)}
RedukcijaDesno	[**K** gore_1]	[.]	{(gore_1, Gore), . . . , (gore_1, gore_3)}
Pomak	[**K** gore_1 .]	[]	{(gore_1, Gore), . . . , (gore_1, gore_3)}
RedukcijaDesno	[**K** gore_1]	[]	{(gore_1, Gore), . . . , (gore_1, .)}
RedukcijaDesno	[**K**]	[]	{(gore_1, Gore), . . . , (**K**, gore_1)}

Slika 4.3: Primjer parsanja ovisnosnog stabla koristeći tranzicijski pristup. **K** je korijen (engl. *root*). Korištena je rečenica prisutna na slici 4.2 i dobiven skup bridova odgovara prikazanom ovisnosnom stablu bez oznaka. Međuspremnik na početku sadrži cijelu rečenicu iz koje se izvlače značke i stavljaju na stog ako je to potrebno. U svakom koraku, prije donošenja jedne od moguće tri odluke, potrebno je izgraditi vektor značajki koji u sebi može sadržavati vektorske reprezentacije riječi na stogu, riječi međuspremnika i prethodnih odluka.

Koristeći oznaku B za početak imenovanog entiteta, oznaku I za nastavak i oznaku da riječ nije ime O, problem je jednostavno sveden na označavanje niza. Moguće je prvo samo označiti riječi entiteta, a kasnije prepoznati koji je tip, ali združeno prepoznavanje u obliku različitih oznaka za različite entitete (B-PER, B-LOC i dr.) bi trebalo imati bolju učinkovitost. Treba napomenuti da nastavak imenovanog entiteta uvijek prethodi nastavak ili početak imenovanog entiteta. To znači da prije nego što smo iskoristili oznaku B ne bi trebalo biti moguće koristiti oznaku I. Ova činjenica upućuje da je ipak potrebno prilagoditi označivač ako bismo htjeli konzistentan izlaz. Drugi pristup je segmentirati i prepoznavati entitete istovremeno. Sarawagi i Cohen (2004) opisuju prilagođen model prepoznavanja entiteta koji, umjesto da za entitet od tri riječi donese tri zasebne odluke (B, I i I), istovremeno odlučuje o duljini entiteta i njegovoj vrsti. Eksperimentalno potvrđuju da je taj pristup bolji od prvog navedenog.

B-PER	I-PER	O	O	O	O	B-LOC	O
Patrik	Baboumian	je	najjači	čovjek	u	Njemačkoj	.

Slika 4.4: Rečenica s označenim imenovanim entitetima. Oznake koje počinju sa B odnose se na početak imenovanog entiteta, a one koje počinju s I kao nastavak. Oznaka O je za značke koje nisu imenovani entitet. Moguće je imati oznake za osobna imena, imena privatnih organizacija, gradova, mjesta i dr.

Algoritam 4.3 pokazuje kako bi izgledala implementacija prepoznavanja imenovanih entiteta koristeći okvir učenja pretraživanja. Funkcija ODLUČI prima skup mogućih oznaka za trenutnu riječ radi održavanja konzistentnosti. Kao i kod označavanja niza, vremenska složenost zaključivanja je $O(TMk)$, što je poboljšanje s obzirom na vremensku složenost vjerojatnosnih grafičkih modela. Algoritam 4.4 pokazuje kako bi izgledala implementacija modela koji je ekvivalentan modelu opisanom u (Sarawagi i Cohen, 2004). Kod ovog modela nije potrebno ponuditi niz mogućih oznaka, nego je za odluku potrebno ponuditi dovoljno značajki (buduće riječi) kako bi odluke o entitetima koji imaju više od jedne riječi bile ispravne. Ako nas interesiraju samo lokacije duljine do tri riječi, onda skup oznaka može biti {LOC1, LOC2, LOC3, O}. Ako je ulaz rečenica, onda je potrebno samo preskočiti riječi za koje je već donesena odluka.

Sličan pristup ilustriran u algoritmu 4.3 mogli bismo iskoristiti za označavanje vrste riječi s morfosintaktičkim deskriptorom (oznaka vrste riječi koja uključuje atribute poput padeža, lica, roda i dr.). Umjesto da se cijeli deskriptor smatra potpunom oznakom, može se uvjetovati skup mogućih oznaka za trenutno donošenje odluke (poziv funkcije ODLUČI). Tako da, ako je riječ u prvom prolazu označena kao imenica,

onda će za drugi prolaz biti ponuđene samo odluke vezane uz rod imenice (muški, ženski ili srednji). Ovakva fleksibilnost čini bilo koji zadatak s dodatnim uvjetima (engl. *constraints*) rješivim (Chang et al., 2012).

Algoritam 4.3 Prepoznavanje imenovanih entiteta.

```
1:  funkcija IZVRŠI(riječi)
2:      izlaz ← []
3:      za n ← 1 do DULJINA(riječi)
4:          ref ← riječi[n].točna_oznaka
5:          skupOznaka ← {B, O}
6:          ako izlaz[n-1] = B ili izlaz[n-1] = I onda
7:              skupOznaka += {I}
8:          kraj
9:          izlaz[n] ← ODLUČI(riječi[n], ref, skupOznaka)
10:     kraj petlje
11:     GUBITAK(# izlaz[n] ≠ riječi[n].točna_oznaka)
12:     return izlaz
13: kraj funkcije
```

Algoritam 4.4 Prepoznavanje imenovanih entiteta sa segmentacijom.

```
1:  funkcija IZVRŠI(riječi)
2:      izlaz ← []
3:      prethodna ← 0
4:      za n ← 1 do DULJINA(riječi)
5:          ako (prethodna -= 1) > 0 onda
6:              nastavi                          ▷ continue
7:          kraj
8:          ref ← riječi[n].točna_oznaka
9:          izlaz[n] ← ODLUČI(riječi[n], ref)
10:         prethodna = izlaz[n].duljina
11:     kraj petlje
12:     GUBITAK(# izlaz[n] ≠ riječi[n].točna_oznaka)
13:     return izlaz
14: kraj funkcije
```

4.4. Od fine do grube analize sentimenta

Analiza sentimenta zadatak je u obradi prirodnoga jezika gdje je potrebno za dani dokument (rečenicu ili paragraf) odrediti njegov sentiment. Sentiment je vezan uz osjećaje i emocije prisutne u dokumentu, a najčešće se koriste oznake za pozitivan, neutralan i negativan sentiment (oznake mogu biti i za emocije poput ljutnje, tuge, sreće, ljubavi i dr.). McDonald et al. (2007) uvode jednostavan model za analizu sentimenta kroz više razina. Koristeći njihov model moguće je za jedan dokument združeno označiti njegov sentiment, ali i sentiment paragrafa i rečenica u dokumentu. Ako bi se sentiment označavao tako da se prvo označi sentiment dokumenta, pa se ta informacija propagira na niže razine, onda bi došlo do nakupljanja pogrešaka jer učenje nije provedeno združeno. Na slici 4.5 prikazana su dva modela, slika 4.5a pokazuje vjerojatnosni grafički model u slučaju označavanja sentimenta dokumenta i rečenica, a slika 4.5b dodatno označavanje i paragrafa. Ako se pretpostavi da je broj mogućih oznaka sentimenta jednak za sve razine, onda je vremenska složenost koraka učenja i predviđanja za model na prvoj slici $O(M^3T)$, a $O(M \cdot (M^2P + PM^2T)) = O(M^3PT)$ za model na drugoj – gdje je M broj mogućih oznaka sentimenta, P broj paragrafa u dokumentu i T broj rečenica u dokumentu u prvom problemu, a prosječan broj rečenica po paragrafu u drugom. Ako bismo povećali broj prijašnjih odluka koje utječu na trenutnu, onda bi eksponent na M rastao. Ako bi se broj mogućih oznaka povećao, onda bi se učenje i zaključivanje dodatno usporilo. Već sada se može zaključiti da učenje i zaključivanje rađeno u okviru vjerojatnosnih grafičkih modela nije dovoljno brzo.

Korisno je razmotriti isti problem u okviru učenja pretraživanja. U dodatku C prisutne su dvije moguće implementacije u okviru učenja pretraživanja za model označavanja sentimenta i rečenica. Moguće je donositi odluke na dva načina:

1. prvo odrediti sentiment na dokumentu, a nakon toga propagirati tu informaciju pri donošenju odluka na rečenicama;

2. prvo odrediti sentiment na svim rečenicama, a nakon toga propagirati cijeli određeni sentiment na odluku za dokument.

Algoritam 4.5 ilustrira prvi pristup, a ponašanje funkcija identično je kao i u algoritmima u potpoglavlju 4.1. Očigledno je postupak određivanja sentimenta samo na rečenicama jednak običnom označavanju niza, stoga se implementacija na ovom zadatku ne razlikuje previše od običnog označivača niza. Drugi pristup ima problem jer broj značajki za pojedinu odluku na razini dokumenta raste ovisno o broju rečenica

(svaka odluka na rečenici postaje značajka u odluci za dokument). Prvi pristup koristi se u (McDonald et al., 2007). Lakše je odrediti sentiment za cijeli dokument nego pojedinu rečenicu. Vremenska složenost ovog pristupa je iznenađujuće dobra. Zaključivanje ima vremensku složenost $O(M(P+T))$ za model na slici 4.5b, što je ogromna razlika u usporedbi s vjerojatnosnim grafičkim modelom.

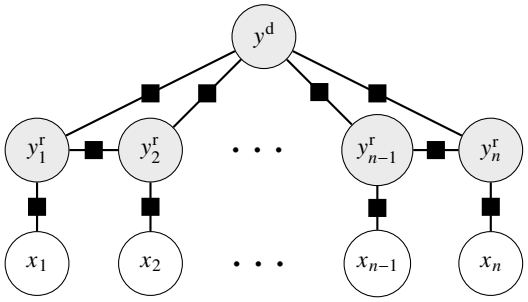
Algoritam 4.5 Združena analiza sentimenta dokumenta i rečenica.

```
1:  funkcija IZVRŠI(dokument)
2:      izlaz ← []
3:      dlabel ← ODLUČI(dokument, dokument.točna_oznaka)
4:      GUBITAK(dlabel ≠ dokument.točna_oznaka) ▷ Izvještaj o parcijalnom gubitku.
5:      za n ← 1 do DULJINA(dokument)
6:          ref ← dokument[n].točna_oznaka
7:          izlaz[n] ← ODLUČI(dokument[n], ref, izlaz[:n-1], dlabel)
8:      kraj petlje
9:      GUBITAK(# izlaz[n] ≠ dokument[n].točna_oznaka)
10:     return izlaz
11: kraj funkcije
```

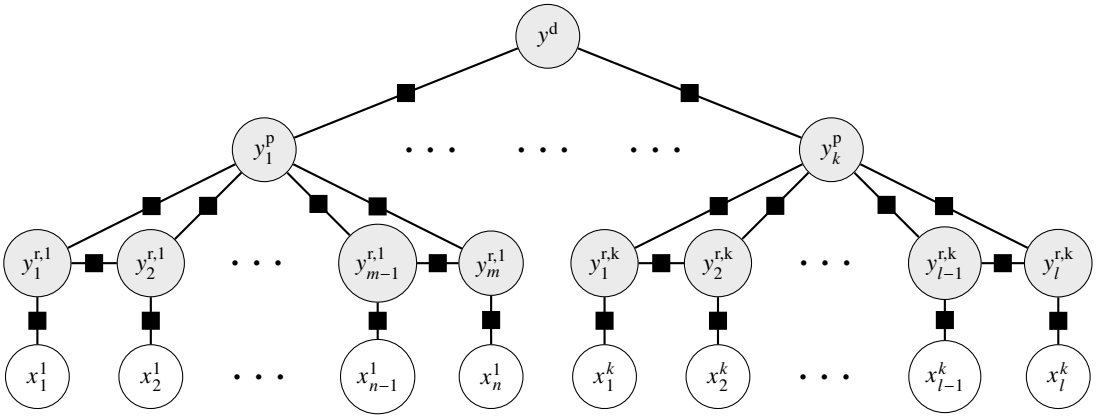
Ovaj pristup lako je primijeniti i na hijerarhijsko označavanje kategorije dokumenata. Prednost pristupa je ta što metoda učenja pretraživanja ima garancije da akumulirana pogreška ne ovisi o duljini niza odluka, a uvjetovanje na prošle odluke ostvaruje se samo dodavanjem dodatnih značajki. Vremenska složenost je dodatna prednost.

4.5. Združeno predviđanje i ostale primjene

Najmoćnija primjena metoda učenja pretraživanja je združeno predviđanje i učenje (engl. *joint prediction and learning*). Zbog činjenice da je moguće koristiti bilo koju funkciju gubitka, moguće je kombinirati više zadataka istovremeno. Trenutni označivači vrste riječi postigli su točnost od otprilike 97–98% (Manning, 2011) i ne mogu više od toga. Jedan od razloga je taj što neke oznake nije lako odrediti bez informacije o sintaksi rečenice. U okviru L2S procjenu troška pojedine oznake vrste riječi moguće je proširiti tako da se nakon označene cijele rečenice izvrši ovisnosno parsanje. Gubitak koji je dobiven nakon parsanja proslijedit će informaciju i poboljšati procjenu troška za pojedine oznake vrste riječi. Primjena združenog predviđanja na ovom konkretnom zadatku postoji, ali brzina predviđanja i učenja nije ni približno zadovoljavajuća (Bohnet i Nivre, 2012).



(a) Model za određivanje sentimenta rečenica i dokumenta.



(b) Model za određivanje sentimenta rečenica, paragrafa i dokumenta.

Slika 4.5: Prikaz hijerarhijskih modela za analizu sentimenta. Sivi čvorovi predstavljaju varijable koje model može generirati, a bijeli koje model može samo opaziti. Prikazana su dva modela gdje su ulazi modelirani značajkama, a izlazi su kategoričke varijable sentimenta (pozitivno, negativno i neutralno). Varijable dokumenta y^{d}, paragrafa y^{p} i rečenice y^{r} predstavljaju sentiment, a x_i ulaznu rečenicu.

Problem prepoznavanja imenovanih entiteta i izlučivanje relacija između entiteta u tekstu dva su zadatka koja bi bilo dobro raditi združeno. Očito učinkovitost izlučivanja relacija ovisi o tome koliko dobro su entiteti prepoznati. Propagiranje gubitka zadatka izlučivanja relacija bi trebalo dodatno poboljšati prepoznavanje entiteta. U okviru L2S moguće je, ako su problemi slijedno povezani (prvo se izvršava jedan, a nakon njega drugi), učiti model na jednom zadatku, a nakon toga učiti na oba istovremeno. Tako se može izbjeći potreba za ogromnom količinom označenih podataka koji imaju oznake za oba zadatka. Ako se uči prepoznavanje imenovanih entiteta na skupu koji nema označene relacije, onda je prisutna lošija procjena troška pojedine odluke – referentna politika nije optimalna, ali nije ni loša. Učenje na skupu podataka sa svim oznakama uvelo bi optimalnu politiku koja ne bi počinjala od nule za zadatak prepoznavanja imenovanih entiteta te bi onda bilo moguće popraviti krive procjene novim podacima na zadatku prepoznavanja entiteta.

Zbog činjenice da je za združeno predviđanje u okviru L2S potrebna samo združena procjena troška, za svaki se problem može koristiti zaseban model. Time je izbjegnuto korištenje značajki za probleme koji od tih značajki ne bi imali koristi. Ako se problemi rješavaju slijedno, kao što je slučaj kod označavanja vrste riječi s ovisnosnim parsanjem, onda je moguće stati sa zaključivanjem nakon izvršenog prvog zadatka. To omogućuje efikasno korištenje modela koji su združeni, a da se testiranje ne vrši združeno. Ovako nešto je vrlo rijetko u literaturi i mnoštvo modela koje koristi algoritme dinamičkog programiranja mora raditi *backtracking*. Bez potpune enumeracije ne postoji sigurnost da je pronađen najvjerojatniji niz odluka za prvi zadatak, stoga je potrebno dohvatiti sve optimalne odluke za združeni zadatak. Ovisnosno parsanje zahtijeva puno više značajki i najbolji parseri ne mogu parsati više od nekoliko tisuća riječi po sekundi – Andor et al. (2016) opisuju združeni označivač i parser koji postiže najbolje rezultate, ali mu je brzina oko 600 riječi po sekundi na osobnom računalu unatoč tome što je složenost algoritma koji je u pozadini $O(T)$, gdje je T duljina rečenice. Združeno učenje bi omogućilo bolje označavanje vrste riječi, ali bi očuvalo brzinu na prvom zadatku označavanja vrste riječi koja je za sustav Vowpal Wabbit preko nekoliko stotina tisuća znački u sekundi (Chang et al., 2014).

5. Vrednovanje

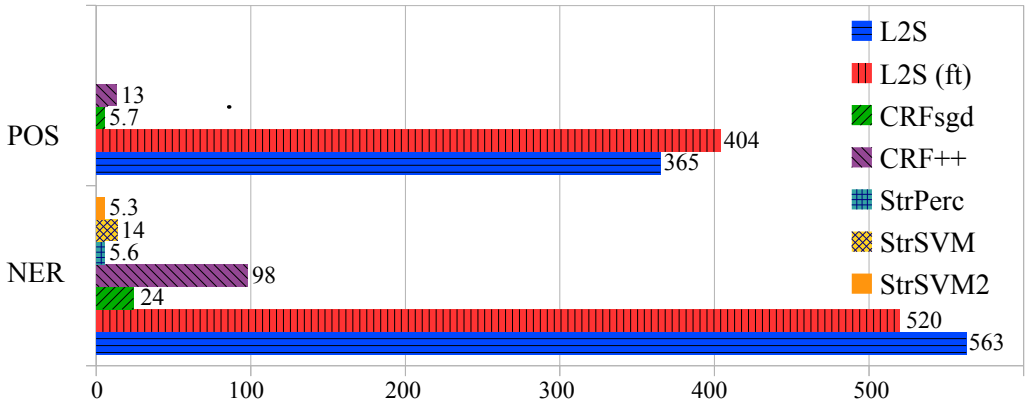
5.1. Implementacija

Svi zadaci ostvareni su u okviru učenja pretraživanja koji je prisutan u sustavu za strojno učenje zvanom Vowpal Wabbit. U dodacima A.1, A.2, A.3 prikazana je jednostavnost implementacije za zadatke označavanja vrste riječi. Korišten je programski jezik C++. Moguće je sve zadatke izvesti u jeziku PYTHON, ali čitljivost smanjuje vremensku učinkovitost. Uzevši u obzir da je okvir učenja pretraživanja poprilično modularan, tijekom implementacije greške su uglavnom bile prisutne u obradi podataka i odabiru hiperparametara. *Hashing trick* dozvoljava običan tekstni format za svaku značajku, stoga je ulazna datoteka vrlo jednostavna (dodatak B). *Hashing trick* je postupak izgradnje vektora značajki gdje se unaprijed zna dimenzionalnost vektora. Potrebna je *hash* funkcija koja će dane riječi pretvoriti u indeks za konkretnu poziciju u vektoru. Više riječi može se preslikati u istu poziciju što upućuje na neizravnu regularizaciju. Uobičajena je praksa odabrati ogromnu dimenziju za vektor značajki i spremati značajke u rijedak vektor (engl. *sparse vector*).

Obrada podataka i njihova pretvorba u prikladan oblik vektora značajki za potrebe sustava Vowpal Wabbit izvršena je koristeći alate ljuske (engl. *shell*). Neki od bitnijih su (1) SED, (2) CUT, (3) DIFF, (4) SHUF i (5) AWK. AWK je ubrzao kreiranje značajki i zbog prikladnog standardnog ponašanja sve napisane skripte rade za označene i neoznačene podatke. SHUF dopušta vrlo jednostavno miješanje rečenica i stvaranje drugačijih skupova za učenje, testiranje i razvoj. DIFF je korišten za računanje mjere točnosti. Zbog stupčastog formata podatkovnih skupova CUT i SED korišteni su za jednostavnu selekciju stupaca, laku izmjenu sadržaja i filtriranje komentara. Svi navedeni alati trebali bi biti prisutni na standardnoj distribuciji UBUNTU. Čitatelja se upućuje na dokumentaciju izvornog koda koja je priložena s elektronskom verzijom rada.

U okviru ovog rada, u sustavu Vowpal Wabbit, implementirana su rješenja za sljedeće probleme od kojih su neki obrađeni u prijašnjim poglavljima:

– označavanje vrste riječi,



Slika 5.1: Usporedba brzine označavanja metoda strukturnog predviđanja. Na horizontalnoj osi prikazani brojevi su u tisućama znački po sekundi (brzina označavanja). Slika se u izvornom obliku nalazi u (Daumé III i Langford, 2015).

- označavanje vrste riječi s više prolaza,
- označavanje vrste riječi koristeći morfosintaktičke deskriptore (engl. *morphosyntactic descriptor*, MSD) po atributima,
- označavanje vrste riječi koristeći MSD po razinama,
- ovisnosno parsanje,
- združeno ovisnosno parsanje i označavanje vrste riječi i
- združeno ovisnosno parsanje i označavanje vrste riječi koristeći MSD.

Rezultati za svaki pristup prikazani su u nastavku.

Chang et al. (2014) opisuju efikasnu implementaciju okvira učenja pretraživanja. Na slici 5.1 prikazana je vremenska učinkovitost sustava na zadacima označavanja vrste riječi i prepoznavanju imenovanih entiteta – broj označenih znački u tisućama po sekundi. Učinkovitost je prisutna i kod ostvarenih rješenja za probleme obrađene u okviru ovog rada.

5.2. Opis skupa podataka za učenje i testiranje

Podaci korišteni za vrednovanje modela zadataka koji slijede koriste univerzalan format ovisnosti (engl. *universal dependencies*, UD), opisan u (Nivre et al., 2015) – CoNLL-

U,[1] a preoblikovani su iz originalnog skupa podataka SETimes.HR (Agić i Ljubešić, 2014).[2] Popis svih raspoloživih jezika i poveznice za preuzimanje skupa podataka nalazi se na universaldependencies.org.[3] U slučaju podataka za označavanje vrste riječi korišten je stari SETimes.HR skup jer je u prikladnijem obliku za taj zadatak. Kod ovisnosnog parsanja korišteni su podaci u CoNLL-U formatu. U poglavlju dodataka B prikazani su sirovi podaci i obrađeni podaci prikladni za ulaz u Vowpal Wabbit. Broj mogućih oznaka vrste riječi je 13 bez atributa, 645 s atributima. Broj mogućih oznaka vrste bridova kod ovisnosnog parsanja je 39.

5.3. Označavanje vrste riječi

Za vrednovanje učinkovitosti koristi se mjera točnosti (engl. *accuracy*). Za usporedbu se koriste rezultati dobiveni u (Agić et al., 2013), koji su dobiveni HunPos označivačem. Vrednovanje se vrši na dva testna skupa (SETimes.HR i Wiki).[4] Vjerojatno su u međuvremenu postignuti bolji rezultati od navedenih, ali nisu zabilježeni u raspoloživoj literaturi.

U tablici 5.1 prikazani su rezultati. Uz prijašnje rezultate prikazana je uspješnost običnog označivača vrste riječi (Vwpos-1 – alg. 4.1) i označivača s više prolaza (Vwpos-2 – alg. 4.2). Korištene značajke su prefiksi i sufiksi do najveće duljine od pet znakova, oblik riječi u kojem su mala i velika slova zamijenjeni slovom l i u, dvije susjedne riječi lijevo i desno od trenutne i duljina riječi. Donošenje trenutne odluke uvjetuje se s tri prošle odluke. U slučaju algoritma Vwpos-2 korištena su tri prolaza. Kod HunPos označivača koristi se Markovljev lanac drugog stupnja, a za rezultate na Wiki testnom skupu i dodatni vanjski leksikon.

U tablici 5.2 prikazani su rezultati za označavanje koristeći morfosintaktičke deskriptore. Značajke korištene identične su kao i za prethodni zadatak. Pristupi Vwpos-1 i 2 koriste se tako da se svaka morfosintaktička oznaka gleda kao zasebna oznaka vrste riječi (takvih u korištenom skupu ima 645). Pristupi Vwmsd 1 i 2 gledaju morfosintaktičke odluke kao oznake vrste riječi s atributima. U svakom pristupu za svaku riječ donosi se niz odluka koje na kraju rezultiraju potpunim ispravnim morfosintaktičkim deskriptorom. Kod pristupa Vwmsd-1 trenutna odluka uvjetovana je svim prethodnim

[1] https://github.com/UniversalDependencies/UD_Croatian

[2] https://github.com/ffnlp/sethr

[3] http://universaldependencies.org/

[4] Datoteke `set.hr.test.conll` i `wiki.hr.test.conll` prisutne na poveznici https://github.com/ffnlp/sethr

Tablica 5.1: Rezultat označavanja vrste riječi. Koristi se mjera točnosti.

Metoda	SET	WIKI
HunPos (Agić et al., 2013)	97.04	94.62
VWPOS-1	98.18	96.20
VWPOS-2	**98.71**	96.24
VWMSD-1	98.31	**96.57**
VWMSD-2	98.23	96.41

odlukama za dvije prošle riječi i trenutnu riječ, a za VWMSD-2 korištene su sve donesene odluke na prethodne dvije i sljedeće dvije riječi.

U oba pristupa VWMSD za riječ se prvo odabire oznaka vrste riječi (jednu od trinaest mogućih). Nakon toga moguće je odabrati prvi atribut, ali ne bilo koji nego baš onaj koji odgovara za odabranu oznaku. To smanjuje broj binarnih odluka koje je potrebno izvršiti. Pretpostavimo da se označava rečenica od deset riječi. Kod pristupa VWPOS-1 za svaku se riječ mora pozvati 645 binarnih odluka (OVA), što na kraju rezultira s ukupno 6450 odluka. Zbog ograničenja mogućih vrijednosti atributa s obzirom na izabranu oznaku vrste riječi broj odluka bi trebao biti manji. Kod donošenja odluke na prvoj razini za riječ možemo izabrati jednu od 13 mogućih. Nakon toga će se za pojedinu riječ – pretpostavljajući da oznaka ima šest mogućih atributa (zamjenica ima toliko) i da svaki atribut ima sedam mogućih vrijednosti (padež kod imenice) – morati pozvati $13 + 6 \cdot 7 = 55$ binarnih odluka, što je ukupno 550 za cijelu rečenicu. U usporedbi sa 6450 to je puno manje, a u praksi broj odluka je još manji iz čega slijedi da je vrijeme učenja i testiranja puno manje. U tablici 5.1 prikazana je uspješnost pristupa VWMSD na običnom zadatku označavanja vrste riječi. Pristupi VWMSD imaju bolju generalizaciju od pristupa VWPOS-1, ali očito povećani broj značajki (svaka prijašnja odluka kojom uvjetujemo trenutnu je nova značajaka) zahtijeva više podataka za bolju generalizaciju. Moguće je da u lancu odluka koji je potreban za formiranje jednog morfosintaktičkog deskriptora dolazi lakše do grešaka nego u slučaju da se jedan morfosintaktički deskriptor promatra kao jedinstvena odluka. Ovaj bi slučaj upućivao na prisutnost pristranosti odlukama koja se može dogoditi ako klasifikator nije dobro naučen. Razlog lošijim rezultatima na testnom skupu WIKI je taj što skup sadrži više neviđenih riječi i nepoznate morfosintaktičke deskriptore. Zbog toga pristupi koji donose odluke po atributima imaju bolju generalizaciju na tom skupu. Ako se kreiraju novi skupovi za učenje i testiranje koji koriste sve rečenice iz svih skupova, onda VWPOS-2 još uvijek ima najbolju točnost.

Tablica 5.2: Rezultat označavanja vrste riječi koristeći morfosintaktičke deskriptore. Prikazana je točnost modela na dva testna skupa.

Metoda	SET	WIKI
HunPos (Agić et al., 2013)	87.11	80.83
VWPOS-1	89.94	83.27
VWPOS-2	**90.22**	**84.13**
VWMSD-1	89.19	82.22
VWMSD-2	89.06	81.90

5.4. Parsanje ovisnosnih stabala

Za vrednovanje učinkovitosti koristi se mjera točnosti (engl. *accuracy*). Koristi se mjera bez oznaka (engl. *unlabeled attachment score*, UAS) za stablo koje nema označene bridove. Ako su usmjereni bridovi označeni, onda se koristi mjera točnost uzimajući oznake u obzir (engl. *labeled attachment score*, LAS). Za usporedbu se koriste rezultati dobiveni u (Agić i Merkler, 2013), koji su dobiveni označivačem MSTParser. Vrednovanje se vrši na testnom skupu u formatu CoNLL-U.[5] Rezultati nisu usporedivi na zadatku gdje se označavaju bridovi jer se koristi drugi podatkovni skup za MSTParser. Taj podatkovni skup ima samo 15 oznaka dok korišteni skup ima 32. Zbog toga bi problem označavanja UD skupa trebao biti teži. Oba skupa imaju iste rečenice i ista ovisnosna stabla bez oznaka, a testni skup prikazan u tablicama je spojen SETIMES i WIKI.

Korištene značajke uključuju sufikse i prefikse do najveće duljine od pet znakova. To su jedine značajke koje je moguće kontrolirati izvan algoritma. Model nakon treniranja koristi više od milijardu zasebnih značajki (svaki jedinstveni sufiks i prefiks i interne značajke su u tom broju), dok je npr. za označavanje vrste riječi taj broj oko 40 milijuna. Za pregled ostalih internih značajki čitatelja se upućuje na (Chang et al., 2015a). Kako je model prevelik, koristi se neuronska mreža s jednim skrivenim slojem od pet čvorova, a za regularizaciju se koristi postupak prati-regulariziranog-vođu (engl. *follow the regularized leader*, FTRL), koji uključuje L1 i L2 regularizaciju. Algoritam FTRL postupak je regularizacije koji dopušta učenje primjer po primjer. Kao kod ostalih algoritama FTL, tijekom učenja nastaje više mogućih hipoteza (vektora težina za završni model) i vrši se odabir hipoteze koja za dani primjer daje najbolje predviđanje

[5]Datoteka `hr-ud-test.conllu` raspoloživa na poveznici https://github.com/UniversalDependencies/UD_Croatian

Tablica 5.3: Rezultat ovisnosnog parsanja.

Metoda	LAS	UAS
MSTParser (POS) (Agić i Merkler, 2013)	74.56	81.59
MSTParser (MSD) (Agić i Merkler, 2013)	77.49	83.58
VWDEP (POS)	74.91	83.17
VWDEP (MSD)	**79.22**	**86.44**

(skalarni produkt između težina hipoteze i vektora značajki koji daje najveću vrijednost). Nakon što je vođa odabran njegove se težine ažuriraju s regularizacijskim pravilom, gdje nastaje nova hipoteza, i postupak se ponavlja. Prikazani rezultati koriste navedenu konfiguraciju, inače bi rezultati bili lošiji jer je skup za učenje premali s obzirom na broj značajki. Chang et al. (2015a) ne koriste sufikse i prefikse jer rade označavanje na jezicima koji imaju dovoljno podataka – značajke su samo cijele riječi i oznaka vrste riječi.

U tablici 5.3 prikazani su rezultati. Kako morfosintaktički deskriptori sadrže informaciju o sintaksi očekivano je da njihovo korištenje poboljšava učinkovitost oba pristupa. Razlog zašto je VWDEP bolji jest zbog superiornijeg algoritma. Cer et al. (2010) dobivaju slične razlike između pristupa baziranog na minimalnom razapinjućem stablu i tranzicijskog parsera. Prikazani rezultati su optimistični jer pretpostavljaju točne oznake vrste riječi kao ulaz, ali označivač vrste riječi razvijen u okviru ovog rada dovoljno je točan da nema bitne razlike između parsera koji koristi zlatne oznake.

5.5. Združeno označavanje vrste riječi i parsanje

U nastavku je prisutno vrednovanje združenog zadatka označavanja vrste riječi i ovisnosnog parsanja. Model se uči na združenom gubitku, a ne odvojeno po zadatku. Prvo se označavaju vrste riječi gdje se odluke propagiraju za zadatak ovisnosnog parsanja. Nakon što je parsanje gotovo, gubitak na cijelom zadatku propagira se i do odluka za oznake vrste riječi. Moguće je model učiti združeno tako da odluka POMAK kod parsera istovremeno daje i vrstu riječi, ali to nije izvedeno u ovom radu. Koriste se iste mjere učinkovitosti za vrednovanje modela i iste značajke opisane u prijašnjim potpoglavljima. Podatkovni skup za učenje i testiranje je za ovaj združeni zadatak samo iz UD, a za potrebe usporedbe testni je skup razdvojen u dva skupa koji su identični SET i WIKI u rečenicama.

U tablici 5.4 prikazana je učinkovitost združenog modela samo na zadatku ozna-

Tablica 5.4: Rezultat označavanja vrste riječi sa združenim modelom. Prikazana je točnost modela na dva testna skupa.

Metoda	SET_{POS}	WIKI_{POS}	SET_{MSD}	WIKI_{MSD}
VWPOS-2	**98.71**	96.24	90.22	84.13
VWJOINT	98.69	**97.23**	**92.03**	**85.01**

Tablica 5.5: Rezultat ovisnosnog parsanja združenog modela.

Metoda	LAS	UAS
VWDEP (MSD)	79.22	86.44
VWJOINT (POS)	75.44	83.91
VWJOINT (MSD)	**81.01**	**88.02**

čavanja vrste riječi. Za usporedbu je prikazan najuspješniji model na tom zadatku VWPOS-2. Očito nije lako unaprijediti označavanje osnovnih oznaka vrste riječi, ali se vidi bitan skok u učinkovitosti kod morfosintaktičkih deskriptora – UD ne koristi sve atribute koje ima SETIMES.HR, ali učinkovitost bez tih atributa nije značajno promijenjena. Skok je bio očekivan jer postoje neki atributi koje je moguće jasnije označiti ako je prisutna informacija o sintaksi. Kako u okviru združenog učenja dolazi do propagacije informacije o grešci nakon ovisnosnog parsanja, onda će zadatak označavanja vrste riječi prilagoditi svoje težine tako da izvršavanje ovisnosnog parsanja bude točnije.

Rezultati u tablici 5.5 potvrđuju da združeno učenje može poboljšati rezultate na zadatku iskorištavanjem funkcije gubitka koja u ovom slučaju daje bolju procjenu greške. Moguće je prvo trenirati samo na jednom zadatku u kaskadi, a kasnije na oba, u slučaju da za prvi zadatak postoji više podataka. U prvotnom slučaju funkcija gubitka daje lošu procjenu prave greške (jer se združeni zadatak ne izvršava), a u naknadnom učenju cilj je pretrenirani model prilagoditi da ima dobre performanse na združenom zadatku.

Kao što je vidljivo iz prijašnjih rezultata, dodavanjem dodatne informacije (prelazak sa POS na MSD) poboljšava uspješnost na zadacima. Najveći skok događa se kod združenog učenja i predviđanja. Informacija u gubitku koju nosi združen zadatak dovoljna je za poboljšanje uspješnosti na oba zadatka. Rezultati za zadatak ovisnosnog parsanja približavaju se rezultatima na ostalim jezicima i vjerojatno bi bilo potrebno povećati korpus za hrvatski jezik da bi se uspješnost još više poboljšala. Rezultati na zadatku označavanja vrste riječi bolji su kod korištenja morfosintaktičkih deskriptora jer je za neke atribute potrebna informacija o sintaksi rečenice.

6. Zaključak

Metode učenja pretraživanja relativno su nov pristup za rješavanje problema združenog učenja i predviđanja. U zadnjih deset godina razvoj metoda omogućio je primjenu na zadatke koji se prije nisu mogli efikasno rješavati ni jednom od uobičajenih metoda strojnog učenja za strukturno predviđanje. Okvir učenja pretraživanja omogućava istraživačima da vrijeme troše na razrađivanje konceptualnih dijelova rješenja, umjesto da to vrijeme troše na ispravljanje grešaka u izvornom kodu algoritama za učenje ili zaključivanje.

U okviru ovog diplomskog rada razvijeni su i vrednovani označivači vrste riječi i ovisnosni parseri s razinama točnosti koje su usporedive s najboljim rezultatima za hrvatski jezik koji su prisutni u literaturi. Najbolji model postignut je koristeći združeno učenje i predviđanje na oba zadatka. Ovo potvrđuje da bi za poboljšanje uspješnosti na pojedinačnim zadacima ubuduće trebalo, ako je to moguće, zadatke rješavati združeno. Informacija koju zadaci dijele u svim uobičajenim pristupima nije iskorištena, a ako ju želimo iskoristiti, onda bi vjerojatno trebali prihvatiti povećanu vremensku složenost ili koristiti računalo s boljim karakteristikama. Kako u okviru učenja pretraživanja nema nedostataka u združenom učenju i predviđanju (vremenska složenost se ne povećava), šteta bi bilo ne iskoristiti bolju združenu procjenu gubitka.

U literaturi trenutačno nije prisutna široka primjena učenja pretraživanja na najteže probleme u obradi prirodnog jezika – strojno prevođenje, sažimanje teksta i ostalih problema koji vode do mogućnosti računala da stvarno razumije prirodni jezik, a informacija o strukturi bi sigurno za te zadatke bila korisna. Navedeni problemi zahtijevaju kvalitetna rješenja za osnovne zadatke (označavanje vrste riječi, ovisnosno parsanje), a u okviru učenja pretraživanja moguće je sve te zadatke vršiti združeno. Mogućnost zamjene osnovnog klasifikatora nekim modelom iz područja dubokog učenja (engl. *deep learning*) i uspješnost metoda učenja pretraživanja na zadacima strukturnog učenja, u obradi prirodnog jezika još nije razrađena i to bi mogao biti sljedeći korak primjene. U svakom slučaju, metode učenja pretraživanja omogućavaju rješavanje problema koji su prije bili nedostižni starijim metodama i ogromna su prilika za istraživanje i primjenu.

LITERATURA

Naoki Abe, Bianca Zadrozny, i John Langford. An iterative method for multi-class cost-sensitive learning. U *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, stranice 3–11. ACM, 2004.

Željko Agić i Nikola Ljubešić. The SETimes.HR Linguistically Annotated Corpus of Croatian. U Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, i Stelios Piperidis, urednici, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, stranice 1724–1727, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). ISBN 978-2-9517408-8-4. URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/690_Paper.pdf. ACL Anthology Identifier: L14-1542.

Željko Agić i Danijela Merkler. Three syntactic formalisms for data-driven dependency parsing of Croatian. U *Text, Speech, and Dialogue*, stranice 560–567. Springer, 2013.

Željko Agić, Nikola Ljubešić, i Danijela Merkler. Lemmatization and morphosyntactic tagging of Croatian and Serbian. U *4th Biennial International Workshop on Balto-Slavic Natural Language Processing (BSNLP 2013)*, 2013.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, i Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

Leonard E Baum i Ted Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

Alina Beygelzimer, Varsha Dani, Tom Hayes, John Langford, i Bianca Zadrozny. Error limiting reductions between classification tasks. U *Proceedings of the 22nd international conference on Machine learning*, stranice 49–56. ACM, 2005a.

Alina Beygelzimer, John Langford, i Bianca Zadrozny. Weighted one-against-all. U *AAAI*, stranice 720–725, 2005b.

Alina Beygelzimer, John Langford, i Pradeep Ravikumar. Error-correcting tournaments. U *Algorithmic Learning Theory*, stranice 247–262. Springer, 2009.

Alina Beygelzimer, Hal Daumé III, John Langford, i Paul Mineiro. Learning Reductions that Really Work. U *IEEE Proceedings*, 2015.

Bernd Bohnet i Joakim Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. U *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, stranice 1455–1465. Association for Computational Linguistics, 2012.

Léon Bottou. *Une Approche théorique de l'Apprentissage Connexioniste; Applications à la reconnaissance de la Parole*. Doktorska disertacija, University of Paris-Sud, 1991.

Daniel M Cer, Marie-Catherine De Marneffe, Daniel Jurafsky, i Christopher D Manning. Parsing to Stanford Dependencies: Trade-offs between Speed and Accuracy. U *LREC*, 2010.

Kai-Wei Chang, Hal Daumé III, John Langford, i Stéphane Ross. A Credit Assignment Compiler for Joint Prediction. *arXiv preprint arXiv:1406.1837*, 2014.

Kai-Wei Chang, He He, Hal Daumé III, i John Langford. Learning to search for dependencies. *arXiv preprint arXiv:1503.05615*, 2015a.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, i John Langford. Learning to search better than your teacher. U *Proceedings of the International Conference on Machine Learning (ICML)*, 2015b.

Ming-Wei Chang, Lev Ratinov, i Dan Roth. Structured learning with constrained conditional models. *Machine learning*, 88(3):399–431, 2012.

William W. Cohen i Vitor R. Carvalho. Stacked sequential learning. U *Proceedings of the IJCAI 2005*, 2005.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. U *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, stranice 1–8. Association for Computational Linguistics, 2002.

Michael Collins i Brian Roark. Incremental parsing with the perceptron algorithm. U *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, stranica 111. Association for Computational Linguistics, 2004.

Michael A Covington. A fundamental algorithm for dependency parsing. U *Proceedings of the 39th annual ACM southeast conference*, stranice 95–102. Citeseer, 2001.

Hal Daumé III. *Practical Structured Learning Techniques for Natural Language Processing*. Doktorska disertacija, University of Southern California, Los Angeles, CA, Kolovoz 2006.

Hal Daumé III i John Langford. Learning to Search ICML Tutorial. http://hunch.net/~l2s/, 2015. Pristupljeno: 2016-06-01.

Hal Daumé III i Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. U *Proceedings of the 22nd international conference on Machine learning*, stranice 169–176. ACM, 2005.

Hal Daumé III, John Langford, i Daniel Marcu. Search-based Structured Prediction. *Machine Learning Journal*, 2009.

Hal Daumé III, Nikos Karampatziakis, John Langford, i Paul Mineiro. Logarithmic Time One-Against-Some. *arXiv preprint arXiv:1606.04988*, 2016.

Hal Daumé III, John Lanford, Kai-Wei Chang, He He, i Sudha Rao. Hands-on Learning to Search for Structured Prediction, 2015. URL http://hunch.net/~l2s/. North American Chapter of the Association for Computational Linguistics.

Janardhan Rao Doppa, Alan Fern, i Prasad Tadepalli. HC-Search: A Learning Framework for Search-based Structured Prediction. *J. Artif. Intell. Res.(JAIR)*, 50: 369–407, 2014.

Jason Eisner i Giorgio Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. U *Proceedings of the 37th annual meeting of the*

Association for Computational Linguistics on Computational Linguistics, stranice 457–464. Association for Computational Linguistics, 1999.

Yoav Freund i Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

Claudio Gentile. A new approximate maximal margin classification algorithm. *The Journal of Machine Learning Research*, 2:213–242, 2002.

Yoav Goldberg i Joakim Nivre. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1:403–414, 2013.

Péter Halácsy, András Kornai, i Csaba Oravecz. HunPos: an open source trigram tagger. U *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, stranice 209–212. Association for Computational Linguistics, 2007.

He He, II Alvin Grissom, Jordan Boyd-Graber, Jordan Boyd Graber, i Hal Daumé III. Syntax-based rewriting for simultaneous machine translation. U *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal*, 2015.

Mark Johnson. Why doesn't EM find good HMM POS-taggers? U *EMNLP-CoNLL*, stranice 296–305, 2007.

Matti Kääriäinen. Lower bounds for reductions. U *Talk at the Atomic Learning Workshop (TTI-C)*, 2006.

John D. Lafferty, Andrew McCallum, i Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. U *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, stranice 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

Percy Liang, Hal Daumé III, i Dan Klein. Structure compilation: trading structure for features. U *Proceedings of the 25th international conference on Machine learning*, stranice 592–599. ACM, 2008.

Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? U *Computational Linguistics and Intelligent Text Processing*, stranice 171–189. Springer, 2011.

David McAllester, Michael Collins, i Fernando Pereira. Case-factor diagrams for structured probabilistic modeling. U *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, stranice 382–391. AUAI Press, 2004.

Andrew McCallum i Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. U *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, stranice 188–191. Association for Computational Linguistics, 2003.

Andrew McCallum, Dayne Freitag, i Fernando CN Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation. U *ICML*, svezak 17, stranice 591–598, 2000.

Ryan McDonald, Kerry Hannan, Tyler Neylon, Mike Wells, i Jeff Reynar. Structured models for fine-to-coarse sentiment analysis. U *Annual Meeting-Association For Computational Linguistics*, svezak 45, stranica 432. Citeseer, 2007.

Jonathan Milgram, Mohamed Cheriet, i Robert Sabourin. "One Against One" or "One Against All": Which One is Better for Handwriting Recognition with SVMs? U *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

Joakim Nivre. An Efficient Algorithm for Projective Dependency Parsing. U *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, stranice 149–160, 2003.

Joakim Nivre, Željko Agić, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Cristina Bosco, Sam Bowman, Giuseppe G. A. Celano, Miriam Connor, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Daniel Galbraith, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Yoav Goldberg, Berta Gonzales, Bruno Guillaume, Jan Hajič, Dag Haug, Radu Ion, Elena Irimia, Anders Johannsen, Hiroshi Kanayama, Jenna Kanerva, Simon Krek, Veronika Laippala, Alessandro Lenci, Nikola Ljubešić, Teresa Lynn, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso,

Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Shunsuke Mori, Hanna Nurmi, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Prokopis Prokopidis, Sampo Pyysalo, Loganathan Ramasamy, Rudolf Rosa, Shadi Saleh, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Kiril Simov, Aaron Smith, Jan Štěpánek, Alane Suhr, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Sumire Uematsu, Larraitz Uria, Viktor Varga, Veronika Vincze, Zdeněk Žabokrtský, Daniel Zeman, i Hanzhi Zhu. Universal Dependencies 1.2, 2015. URL http://hdl.handle.net/11234/1-1548. LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague.

Vasin Punyakanok i Dan Roth. The use of classifiers in sequential inference. *arXiv preprint cs/0111003*, 2001.

Nathan D Ratliff, J Andrew Bagnell, i Martin A Zinkevich. Maximum margin planning. U *Proceedings of the 23rd international conference on Machine learning*, stranice 729–736. ACM, 2006.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Stephane Ross i J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stéphane Ross, Geoffrey J Gordon, i Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. U *AISTATS*, svezak 1, stranica 6, 2011.

Sunita Sarawagi i William W Cohen. Semi-markov conditional random fields for information extraction. U *Advances in neural information processing systems*, stranice 1185–1192, 2004.

Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. U *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, stranice 104–107. Association for Computational Linguistics, 2004.

Fei Sha i Fernando Pereira. Shallow parsing with conditional random fields. U *Proceedings of the 2003 Conference of the North American Chapter of the Association*

for Computational Linguistics on Human Language Technology-Volume 1, stranice 134–141. Association for Computational Linguistics, 2003.

Dou Shen, Jian-Tao Sun, Hua Li, Qiang Yang, i Zheng Chen. Document Summarization Using Conditional Random Fields. U *IJCAI*, svezak 7, stranice 2862–2867, 2007.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Nataliya Sokolovska, Thomas Lavergne, Olivier Cappé, i François Yvon. Efficient learning of sparse conditional random fields for supervised sequence labeling. *Selected Topics in Signal Processing, IEEE Journal of*, 4(6):953–964, 2010.

Charles Sutton i Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, stranice 93–128, 2006.

Richard S Sutton i Andrew G Barto. *Reinforcement learning: An introduction*, svezak 1. MIT press Cambridge, 1998.

Ben Taskar, Carlos Guestrin, i Daphne Koller. Maximum-margin markov networks. *Advances in neural information processing systems*, 16:25, 2003.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, i Yasemin Altun. Large margin methods for structured and interdependent output variables. U *Journal of Machine Learning Research*, stranice 1453–1484, 2005.

Hanna M Wallach. Conditional random fields: An introduction. *Technical Reports (CIS)*, stranica 22, 2004.

Bianca Zadrozny, John Langford, i Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. U *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, stranice 435–442. IEEE, 2003.

Yue Zhang i Joakim Nivre. Transition-based dependency parsing with rich non-local features. U *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, stranice 188–193. Association for Computational Linguistics, 2011.

GuoDong Zhou i Jian Su. Named entity recognition using an HMM-based chunk tagger. U *proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, stranice 473–480. Association for Computational Linguistics, 2002.

Dodatak A
Označivači vrste riječi

A.1. Običan označivač vrste riječi

```
void run(Search::search& sch, vector<example*>& ec)
{ Search::predictor P(sch, (ptag)0);
  for (size_t i=0; i<ec.size(); i++)
  { action oracle     = ec[i]->l.multi.label;
    size_t prediction = P.set_tag((ptag)i+1)
                         .set_input(*ec[i])
                         .set_oracle(oracle)
                         .set_condition_range((ptag)i, sch.get_history_length(), 'p')
                         .predict();

    if (sch.output().good())
      sch.output() << sch.pretty_label((uint32_t)prediction) << ' ';
  }
}
```

A.2. MSD slijedni označivač po atributima

```
void run(Search::search& sch, vector<example*>& ec)
{ Search::predictor P(sch, (ptag)0);
 task_data * data = sch.get_task_data<task_data>();
 resize_tags(data, ec.size());
 size_t pass = 0, shift = 1;
 for (size_t i=0; i<ec.size(); i++)
 {
   while (has_positional_decision(data, i, pass)) {
     ptag p = (ptag)shift;
     action oracle     = get_label(ec[i], pass);
     v_array<action> & allowed = get_allowed(data, pass, i);
     P.set_tag(p).set_input(*ec[i]).set_oracle(oracle).set_allowed(allowed);

     if (i > 0) {
       size_t start = i < sch.get_history_length() ? 0 : i - sch.get_history_length();
       char name = 'a', prevn='a'; // for erasing previous conditions
       for(size_t j = start; j < i; ++j) {
         for(tag & t : get_tags(data, j)) {
           if (name == 'a') P.set_condition(t.i, name++);
           else             P.add_condition(t.i, name++);
         }
         name = prevn + 10; prevn = name;
       }
     }

     size_t prediction = P.predict();
     push_tag(data, i, p, prediction, pass);

     sch.loss(prediction != oracle);
     if (sch.output().good()){
       sch.output() << sch.pretty_label((uint32_t)prediction);
       if (has_positional_decision(data, i, pass+1)) {sch.output() << '␣';}
       else                                          {sch.output() << '\n';}
     }
     shift += 1;
     pass += 1;
   }
   pass = 0;
 }
}
```

A.3. MSD označivač po slojevima

```
void run(Search::search& sch, vector<example*>& ec)
{ Search::predictor P(sch, (ptag)0);
  task_data * data = sch.get_task_data<task_data>();
  resize_tags(data, ec.size());
  size_t pass = 0, shift = 1; bool touched = true;
  while (touched) {
    touched = false;
    for (size_t i=0; i<ec.size(); i++)
    { if (has_positional_decision(data, i, pass)) {
        ptag p = (ptag)shift;
        action oracle     = get_label(ec[i], pass);
        v_array<action> & allowed = get_allowed(data, pass, i);
        P.set_tag(p).set_input(*ec[i]).set_oracle(oracle).set_allowed(allowed);

        size_t start = i < sch.get_history_length() ? 0 : i - sch.get_history_length()/2;
        size_t end = pass == 0 ? i : std::min(i+1 + sch.get_history_length()/2, ec.size());
        char name = 'a', prevn = 'a'; // for erasing previous conditions
        for(size_t j = start; j < end; ++j) {
          if (j == i) continue;
          for(tag & t : get_tags(data, j)) {
            if (name == 'a') P.set_condition(t.i, name++);
            else             P.add_condition(t.i, name++);
          }
          name = prevn + 10; prevn = name; // no msd has more than 10 labels
        }
        size_t prediction = P.predict(); push_tag(data, i, p, prediction, pass);
        shift += 1; touched = true;
      }
    }
    pass += 1;
  }
  int loss = 0;
  for(int j = 0; j < ec.size(); ++j){
    pass = 0;
    for(tag & t : get_tags(data, j)) {
      if (t.a !=  get_label(ec[j], pass++)) {
        loss += 1; break;
      }
    }
  }
  sch.loss(loss);
  if (sch.output().good()){
    for(int j = 0; j < ec.size(); ++j){
      pass = 0;
      for(tag & t : get_tags(data, j)) {
        sch.output() << sch.pretty_label((uint32_t)t.a);
        if (has_positional_decision(data, j, ++pass)) {sch.output() << '␣';}
        else                                          {sch.output() << '\n';}
      }
    }
  }
}
```

Dodatak B
Podaci

B.1. SETimes.HR

```
1 Proces  proces  N       Ncmsn   Type=common|Gender=masculine|Number=singular|Case=nominative    0
Elp     6:nsubj UposTag=NOUN|Case=Nom|Gender=Masc|Number=Sing
2 privatizacije   privatizacija   N       Ncfsg   Type=common|Gender=feminine|Number=singular|Case=genitive       1
Obj     1:nmod  UposTag=NOUN|Case=Gen|Gender=Fem|Number=Sing
3 na      na      S       Sl      Case=locative   1       Prep    4:case  UposTag=ADP|Case=Loc
4 Kosovu  Kosovo  N       Npnsl   Type=proper|Gender=neuter|Number=singular|Case=locative 3
Adv     6:nmod  UposTag=PROPN|Case=Loc|Gender=Neut|Number=Sing
5 pod     pod     S       Si      Case=instrumental       0       Prep    6:case  UposTag=ADP|Case=Ins
6 povećalom       povećalo        N       Ncnsi   Type=common|Gender=neuter|Number=singular|Case=instrumental     5
Elp     0:root  UposTag=NOUN|Case=Ins|Gender=Neut|Number=Sing
```

B.2. UD

```
1 Proces  proces  NOUN    _       Case=Nom|Gender=Masc|Number=Sing        6       nsubj   _       _
2 privatizacije   privatizacija   NOUN    _       Case=Gen|Gender=Fem|Number=Sing 1       nmod    _       _
3 na      na      ADP     _       Case=Loc        4       case    _       _
4 Kosovu  Kosovo  PROPN   _       Case=Loc|Gender=Neut|Number=Sing        6       nmod    _       _
5 pod     pod     ADP     _       Case=Ins        6       case    _       _
6 povećalom       povećalo        NOUN    _       Case=Ins|Gender=Neut|Number=Sing        0       root    _       _
```

B.3. vw - označavanje vrste riječi

```
2 |w Proces proces 0:6 ullll
2 |w privatizacije privatizacija 0:13 lllllllllllll
1 |w na na 0:2 ll
2 |w Kosovu kosovu 0:6 ullll
1 |w pod pod 0:3 lll
2 |w povećalom povećalom 0:9 lllllllll
```

B.4. vw - ovisnosno parsanje

```
6 1 |w Proces |p NOUN  NOUNCase=Nom NOUNGender=Masc NOUNNumber=Sing
1 2 |w privatizacije |p NOUN  NOUNCase=Gen NOUNGender=Fem NOUNNumber=Sing
4 3 |w na |p ADP  ADPCase=Loc
6 2 |w Kosovu |p PROPN  PROPNCase=Loc PROPNGender=Neut PROPNNumber=Sing
6 3 |w pod |p ADP  ADPCase=Ins
0 4 |w povećalom |p NOUN  NOUNCase=Ins NOUNGender=Neut NOUNNumber=Sing
```

Dodatak C
Analiza sentimenta

C.1. Dokument prije rečenica

```
void run(Search::search& sch, vector<example*>& ec)
{ Search::predictor P(sch, (ptag)0);
  action oracle = ec[0]->l.multi.label;
  size_t global_prediction = P.set_tag(1 /* global tag */)
                              .set_input(*ec[0])
                              .set_oracle(oracle)
                              .predict();
  if (sch.output().good())
      sch.output() << sch.pretty_label((uint32_t)global_prediction) << ' ';
  for (size_t i=1; i<ec.size(); i++)
  { oracle     = ec[i]->l.multi.label;
    size_t prediction = P.set_tag((ptag)i+1)
                         .set_input(*ec[i])
                         .set_oracle(oracle)
                         .set_condition_range((ptag)i, sch.get_history_length(), 'p')
                         .add_condition(1 /* global tag */, 'a')
                         // adds the global as condition on each sequence element
                         .predict();

    if (sch.output().good())
      sch.output() << sch.pretty_label((uint32_t)prediction) << ' ';
  }
}
```

C.2. Rečenice prije dokumenta

```
void run(Search::search& sch, vector<example*>& ec)
{ Search::predictor P(sch, (ptag)0);
  for (size_t i=0; i<ec.size()-1; i++)
  { action oracle     = ec[i]->l.multi.label;
    size_t prediction = P.set_tag((ptag)i+1)
                         .set_input(*ec[i])
                         .set_oracle(oracle)
                         .set_condition_range((ptag)i, sch.get_history_length(), 'p')
                         .predict();

    if (sch.output().good())
      sch.output() << sch.pretty_label((uint32_t)prediction) << '␣';
  }
  action oracle = ec.back()->l.multi.label;
  size_t global_prediction = P.set_tag(ec.size())
                              .set_input(*ec.back())
                              .set_oracle(oracle)
                              .set_condition_range(ec.size()-1, ec.size()-1, 'a')
                              // conditioning on all previous sequence decisions
                              .predict();
  if (sch.output().good())
      sch.output() << sch.pretty_label((uint32_t)global_prediction) << '␣';
}
```

Učenje pretraživanja za rješavanje zadataka obrade prirodnoga jezika

Sažetak

Strukturno predviđanje i učenje sveprisutno je u problemima obrade prirodnoga jezika. Metode učenja pretraživanja pružaju okvir u kojem je te probleme moguće efikasno rješavati. U ovom radu dan je pregled povijesti metoda učenja pretraživanja i osvrt na područja u strojnom učenju koja su omogućila njihov razvoj. Pokazana je i primjena na razne probleme u obradi prirodnog jezika. Istaknute su i bitne razlike između ostalih pristupa strukturnom učenju koje utvrđuju nadmoć i prilagodivost metoda učenja pretraživanja. Ostvaren je sustav koji zadatke označavanja vrste riječi i ovisnosnog parsanja vrši združeno te su istaknute razlike i prednosti na pristupe koji zadatke ne gledaju združeno. Vrednovanje sustava izvršeno je na podacima za hrvatski jezik.

Ključne riječi: učenje pretraživanja, obrada prirodnog jezika, strojno učenje, združeno učenje i predviđanje, hrvatski jezik

Learning to Search for Solving Natural Language Processing Tasks

Abstract

Structured prediction and learning is omnipresent for problems in natural language processing. Learning to search (L2S) methods provide a framework in which these problems can be efficiently solved. This thesis gives an overview of L2S methods and their theoretical basis that allowed their development. Their application to various problems of natural language processing is described. Important differences that show the superiority and flexibility of L2S methods are pointed out in the context of previous solutions for structured and joint prediction. Description of the development of a system that considers part-of-speech tagging and dependency parsing as a joint task is provided and detailed analysis of differences and advantages to various approaches is given. Evaluation of the system is done on Croatian corpus.

Keywords: learning to search, natural language processing, machine learning, joint prediction and learning, Croatian language