Popović Zoran

Evaluacija programa za obeležavanje (etiketiranje) teksta na srpskom jeziku

Matematički fakultet
Univerzitet u Beogradu

Beograd, 2008.

Popović Zoran

Evaluacija programa za obeležavanje (etiketiranje) teksta na srpskom jeziku

Članovi komisije:

Mentor: prof dr Duško Vitas
prof dr Gordana Pavlović-Lažetić
prof dr Ivan Obradović

Sadržaj

1 Uvod

1.1 Obrada prirodnih jezika i etiketiranje

Problemi obrade prirodnog jezika (NLP, Natural Language Processing) kao jedne od oblasti računske lingvistike (Computational Linguistics) podrazumevaju najčešće veoma složen postupak kojeg čine pre svega: leksička analiza (segmentacija govora ili teksta, gde se najpre određuje početak i kraj reči i rečenica, kao i osnovne leksičke klase (lekseme, ili tokeni): brojevi, interpunkcija, reči, HTML etikete, i slično), morfo-sintaksna analiza (struktura reči, rečenica i teksta), i na kraju semantička (ili čak analiza pragmatike - način izgovora i naglasak nad određenom reči menja suštinsko značenje rečenice) analiza. Parseri, programi koji objedinjuju ceo ovaj postupak imaju nimalo lak zadatak da obuhvate sva pravila i osobine nekog prirodnog jezika, kao i jezičke nejasnoće i mnoge druge probleme. Tradicionalan pristup je od ,,vrha na dole", gde se složenim analizama u objedinjenom postupku dobijaju i jednostavnije osobine kao što su leksičke. Cilj ovih analiza su strukture i pravila po ugledu na formalne gramatike Noama Čomskog - koje odgovaraju tipovima jezika prema osobinama produkcionih pravila: regularnim, kontekstno-slobodnim (koji se najčešće eksploatišu u tom smislu), kontekstno-osetljivim i rekurzivno nabrojivim (bez ograničenja). Jezički modeli i odgovarajući algoritmi zbog toga u mnogim slučajevima nisu bili dovoljno praktično efikasni zbog velike složenosti takvog modela prirodnog jezika i cene rešenja. Nakon početnog entuzijazma nastalog pod uticajem tehnološkog razvoja računara do početka 70-tih brzo se došlo do zaključka o pravoj težini klase problema kojima pripada i automatsko prevođenje - nisu dovoljni samo sintaksna analiza, rečnik i dobri algoritmi pretrage već i znanje o semantici jezika (kontekst, npr. opisan semantičkim mrežama), pa na kraju i nekakvo opšte znanje i iskustvo ugrađeno u program takve vrste.

Jedan pristup ovim problemima koji polazi obratno od jednostavnijih struktura (od dole ka vrhu) u kojem se koriste efikasni (brzi) algoritmi statističke prirode (ne toliko egzaktni u tradicionalnom smislu) su dali neočekivano dobre rezultate i bitno doprineli novim NLP rešenjima. Jedna klasa takvih programa se bavi otkrivanjem klase reči u rečinici, programi za etiketiranje (**tagger**-i, otkrivaju složenije leksičke klase, odnosno gramatičke delove rečenice - PoS, Part-of-Speech), gde svaka reč prema morfološkoj, sintaksnoj i drugim ulogama dobija odgovarajuću etiketu (pripadnost klasi: imenica, zamenica, glagol, predlog, itd - broj klasa može biti veoma velik ako se uzmu u obzir i druge osobine pored tipa: rod, broj, lice, padež i slično). Zatvorene klase etiketa se ne menjaju (npr. brojevi, zamenice), a kod otvorenih se

očekuje često proširivanje. Broj klasa je tako određen složenošću modela jezika koji se na taj način gradi, ali i samim jezikom. Negde 80-tih godina prvi praktični uspesi evropskih istraživača ([5] - CLAWS, delom baziran na skrivenim lancima Markova, HMM) pokrenutli su dalje interesovanje za ove i slične probleme i programe, razvoj statističkog NLP (kao i Corpus based NLP), kao i primenu mašinskog učenja u NLP.

Ovaj rad pre svega ima za cilj uporednu evaluaciju nekoliko različitih postojećih rešenja i njihovih konkretnih implementacija - programa za etiketiranje teksta, koji predstavljaju referentna i akademska rešenja, i to primenjenih na tekst na srpskom jeziku. To su mahom nekomercijalni programi, ali to nipošto ne znači da programi za etiketiranje nisu i komercijalno priznati, naprotiv - postoje mnoge komercijalne primene, kao i veze sa drugim NLP problemima. Neki od primera primene su: lematizacija (ili stemming, utvrđivanje osnove reči za datu reč - grupisanje različitih oblika reči i morfoloških pravila - radi se automatski ali može i leksikonom), prepoznavanje i sinteza govora (PoS predstavlja nezamenljivu heuristiku, ali i automatsko ispravljanje reči koristi slične metode). Problemi koji su slični (mada ne isti) i rešavaju se sličnim metodama su: priključenje izraza sa predlogom (Prepositional Phraze Attachment - na koji deo rečenice se odnosi predlog ?), Word Sense Disambiguation (ista reč u tekstu može da ima različita značenja, npr. ,,znak" kao slovo, grafički objekat, ili kao deo broja - PoS i WSD programi iako veoma slični po načinu rešavanja problema nisu jednako uspešni - 75% WSD uspešnosti naspram 95% za PoS), prepoznavanje granica rečenice (ili reči u nekim jezicima) - a na ovo sve se oslanjaju: parsiranje rečenice (sintaksne strukture etiketirane čenice kao drvoidne strukture, uz razrešenje svih nejasnoća reči i prepoznate strukture), automatsko prevođenje, Information Retrieval (IR ili pretraživanje informacija kao oblast primene mašinskog učenja, od postavljanja upita nad velikim brojem dokumenata, kategorizacije teksta do semantičkog web-a), i drugo.

1.2 Osnovni pojmovi

Jezički model može biti opisan na različite načine, kako eksplicitno kroz gramatička, morfološka i druga pravila, tako i implicitno davanjem primera ispravnih jezičkih konstrukcija i njihovog značenja. Tako se izdvajaju pre svega dva osnovna pristupa: nestatistički i statistički. Ovde dalje u tekstu je opisan pristup (u kome se jezički model zadaje jezičkim korpusom) koji odgovara modelima statističke teorije učenja (početak još sa informacionom teorijom, najkraćom i najverovatnijom hipotezom, Shanon, Weaver, 1949, i čuvenim primerom Šenonove igre prepoznavanja sledeće reči u rečenici (cilj

je da bude što manja entropija reči), 1951; kasnije 1960-tih Ray Solomonoff, Kolmogorov, Chaitin, Gold i drugi postavljaju pojmove statističke indukcije, stohastičkih kontekstno-slobodnih gramatika i identifikacije jezika). Poznata je anegdotska izjava Freda Jelineka (istraživača u oblasti NLP i automatskog prepoznavanja govora, IBM 1988) koja ilustruje sukob racionalista i empirista u NLP (generalno, po prvima prirodni jezik je kod ljudi inherentna osobina, nekim delom ,,instinktivna" kao svojevrsni bias, dok je po drugima učenje jezika bez preduslova najvažnija osobina): ,,kada god lingvista napusti prostoriju poveća se uspešnost programa za automatsko prepoznavanje govora". Čomski je opet s pravom smatrao svojevremeno da nijedan poznat pojam verovatnoće rečenice nema smisla (nije bilo teoretskog osnova niti praktičnog opravdanja s obzirom na kompleksnost problema identifikacije jezika - njegov primer za to: ,,Colorless green ideas sleep furiously" ne znači ništa iako je gramatički ispravno; entropija reči definisana je najpre pokušajima pogađanja od strane ljudi, i pitanje je da li se semantički može povezati sa nekom teoretskom veličinom - slično problemu Tjuringovog testa ili kineske sobe, ova pitanja nemaju jednostavan odgovor). Napretkom statističke teorije mašinskog učenja u poslednje vreme postaje ponovo aktuelna veza između ova dva pristupa.

Ovaj drugi pristup je poznat u lingvistici i upotrebom jezičkog korpusa koji predstavlja seriju odlomaka različitih tekstova (novinski članci, knjige i slično) sa pomenutim etiketama (oznakama, anotacijama). Takav jezički korpus predstavlja osnovu daljeg proučavanja jezika, a za NLP probleme predstavlja nezamenljiv resurs. Korpus je određen ukupnim brojem reči čiji niz čini njegov tekst (dužina), leksikonom (rečnikom korpusa koji obično podrazumeva veze sa pomenutim morfo-sintaksnim klasama - leksemama, a ponekad i sa lemama i frekvencijama istih u korpusu - svaka reč ima osnovu, lemu, i svoje različite oblike za svaku od etiketa: ,,leksema = lema + PoS") i klasama etiketa (dužina leksikona je broj leksema u korpusu/leksikonu). Struktura klase etikete određuje bogatstvo korpusa, ali i otežava samo etiketiranje (kako ručno, tako i automatsko - programima za etiketiranje). Korpus služi kao ,,zlatni standard" kojim se obučava program za etiketiranje i mere njegova tačnost (performanse u smislu mašinskog učenja). Mogu posebno biti interesantni paralelni korpusi (i leksikoni) gde se isti sadržaj proučava na više različitih jezika (to je npr. važno za probleme automatskog prevođenja prirodnog jezika). Jezički korpus za srpsko-hrvatski jezik od 20-tak miliona reči postojao je još sredinom 70-tih godina u ne-digitalnom obliku, ali za primenu programa za etiketiranje je važan digitalni oblik korpusa. Jedan uobičajen tok razvoja digitalnog korpusa (uz sve nepohodne resurse: klasu etiketa, polazni ručno etiketirani korpus i leksikon) je:

1. akvizicija digitalnog izvora
2. konverzija u standardni format, problemi izbora karakter seta, struktura, navodnici i drugi znaci, zaglavlja, itd.
3. segmentacija: rečenice, tokeni(zacija) - tipovi, veličina slova, skraćenice, složenice, brojevi
4. morfološka analiza: leksikon, (pravila), skupovi etiketa
5. etiketiranje (obrada) programom za etiketiranje
6. iterativno poboljšavanje: interaktivno okruženje
7. eksploatacija (npr. konkordacija - konkordance su veze (poravnanja) između različitih delova teksta zadatih istom leksemom ili regularnim izrazom - korisno za proučavanje slaganja reči u rečenici prema osobinama u smislu ispravnih konstrukcija, ili čak paralelno između tekstova na različitim jezicima - primer: [24])

U ovom radu se koriste već pripremljeni korpusi o kojima će u daljem tekstu biti više detalja. Poznati primeri korpusa za engleski jezijk su Brown-ov (1000000 tj. 1M reči, nastao 60-tih), National British Corpus (BNC) i WSJ deo PTB (Wall Street Journal članci u okviru PTB - Penn Tree Bank), a primer klase etiketa je Penn Tree Bank Tagset, ICE ili WSJ TreeBank Tagset ([21]: TreeBank je, inače, posebna vrsta korpusa gde je svaka rečenica označena sintaksnom strukturom - drvetom, što je korisno za testiranje i obučavanje parsera). Postoji pristup integrisanog automatizovanog građenja korpusa gde je program za etiketiranje deo većeg NLP sistema i gde semantika etiketa nije od značaja - onda je moguće indukovati etikete automatski. Vredi takođe pomenuti da je lematizaciju moguće raditi i automatski - primera radi u okviru MULTEXT-East [1] se koristi program CLOG koji na osnovu datog korpusa generiše (uči) Perl kôd koji vraća lemu za datu leksemu za svaku etiketu posebno (svodi se na transformaciju ulazne niske).

Nekoliko korpusa (prema podacima iz [22], gde se takođe pominje i mogućnost nenadgledanog učenja programa za etiketiranje) datih ispod ilustruje neke osnovne njihove osobine:

Korpus	BNC	CLEF	Wortschatz
Jezik	Engleski	Holandski	Nemački
Dužina	100M	70M	755M
Broj etiketa	344	418	511
Broj leksema	25706	21863	74398

(Tabela 1 - nekoliko primera korpusa)

U programima koji koji koriste ovakve korpuse (gde spadaju i programi za etiketiranje) jezički model je određen pre svega leksikonom i klasama etiketa, a zatim i implicitno korpusom zadatim dodatnim podacima koji zavise od samog programa i njegovog algoritma (neka vrsta biasa ovakvih programa - to mogu biti datoteke n-grama kao kod TnT-a, pravila prezapisivanja kao kod Brilovog programa, i drugo). Tako shvaćen jezički model se razlikuje od tradicionalnog skupa formalnih gramatičkih i drugih pravila koja su u ovom slučaju donekle sadržana u nekom drugačijem obliku. N-grami kao podniske dužine N reči ili etiketa u korpusu (sa frekvencijama pojavljivanja u tekstu) sadrže implicitnu informaciju o ispravnim ili statistički očekivanim vezama među rečima (ali s druge strane zamagljuju sintaksnu strukturu rečenice, što za problem etiketiranja ipak nije presudno važno). Pomenuti primer semantički neispravne rečenice koji daje Čomski bi trebao da bude odbačen u savremenim statističkim NLP programima jer jednostavno nema primera koji ga podržava kao deo korpusa.

2 Različiti programi za etiketiranje i metode mašinskog učenja

Kako je već pomenuto, prve uspešne implementacije programa za etiketiranje su zasnovane na statističkim metodama, i to posebno na metodi skrivenih lanaca Markova (HMM, Hidden Markov Models). Neki od najpopularnijih i najefikasnijih rešenja (kao što je TNT) i danas koriste u suštini isti metod uz neka poboljšanja. Mnoga poboljšanja se tiču osetljivosti programa za etiketiranje na nepoznate reči (one koje se ne nalaze u leksikonu) gde se uvek pokazuje veliki pad tačnosti (i do 50%) bez dodatnih modifikacija. Mnogi programi koriste prepoznavanje dela reči ili sufiksa/prefiksa (sa određenim zadatim maksimalnim brojem slova) u leksikonu, ili se prate dodatne osobine leksema pored postojećih (koje često mogu biti i specifične za određeni jezički model ili jezik), ili kao u Brilovom slučaju, koriste se posebna kontekstna pravila kojima se određuje najverovatnija etiketa nepoznate reči. Pošto se etiketiranje može svesti na problem metoda mašinskog učenja i klasifikacije (date reči u odnosu na neki kontekst), vremenom su nastale implementacije zasnovane i na metodama koje nisu statističke prirode. Referentni primer je Brilov program zasnovan na učenju pravila prezapisivanja kojima se povećava tačnost etiketiranja (uveo je pravila prezapisivanja kojima obuhvata i reči pored etiketa u odnosu na standardni HMM), a dosad najbolje rezultate postigli su programi zasnovani na SVM klasifikaciji - kakav je i SVMTool, pored Stanford-ovog PoS Tagger-a ([11], zasnovanog na Bajesovim mrežama i varijanti MLE) i LTAG POS Tagger-a ([12], zasnovanog na varijanti neuronskih mreža), prema [10] - svi ovi programi koriste dvosmerno učenje u odnosu na tradicionalno ,,s leva na desno" čitanje tokom učenja. SVM nije metoda zasnovana na verovatnoći (iako jeste proizvod statističke teorije učenja) i statistici, i predstavlja veoma popularan i efikasan metod klasifikacije.

Postoje mnoge varijante statističkog učenja koje nisu zasnovane na HMM i sličnim modelima - na primer, program MXPOST koristi model maksimalne entropije. Statistički metod je donekle prisutan i u Brilovom programu (određivanje inicijalnog stanja), ali to nije njegova suština već učenje pravila prezapisivanja etiketa. Među ne-statistički orijentisanim metodama pre svega se izdvaja učenje drvetom odluke (Decision Tree Learning) koje koristi program Tree Tagger, i učenje memorijom (MBT - Memory-Based Taggger - TiMBL, Daelemans, 1999) koje se svodi na učenje izborom slučajeva (Case-Based Reasoning, Learning) i metodom najbližeg suseda (kNN).

Nova rešenja i realizacije programa za etiketiranje se često pojavljuju -

organizuju se takmičenja i konferencije gde se upoređuju desetine rešenja i trenutno se vodi ,,rat" gotovo za svaki promil tačnosti. Pored toga, brzina rada programa i njegovog obučavanja, kao i jednostavnost njegove upotrebe mnogima znače više nego nekoliko procenata tačnosti. U daljem tekstu će biti kratko opisane nekoliko najpopularnijih metoda mašinskog učenja (sve su to metode nadgledanog učenja - korpus kao skup obučavanja) i odgovarajućih programa za etiketiranje koji ih koriste.

2.1 Performanse programa za etiketiranje

Različiti kriterijumi mogu biti interesantni prilikom ocene rada nekog od programa za etiketiranje, ali najbitnija ocena performanse jeste tačnost etiketiranja (u funkcionalnom smislu, kao i kod svakog problema mašinskog učenja). U ovom tekstu, greška etiketiranja za dati niz rečenica i reči u njima biće definisana kao broj pogrešno etiketiranih reči u odnosu na ukupan broj reči. Pored ove osobine, za ovakve programe može biti važno i vreme izvršavanja učenja, a još više vreme etiketiranja, ali to u ovom tekstu ne razmatram detaljno. Važne su i dodatne mogućnosti i opcije učenja modela.

Testiranje performansi se radi desetostrukim unakrsnim testom (10-fold cross-validation) koji se često koristi i za finije određivanje optimalnih vrednosti parametara učenja. Sastoji se u tome da se korpus (skup obučavanja) izdeli na 10 delova, i da se onda obuka programa vrši nad 9/10 korpusa, a testiranje nad preostalih 1/10. Postupak se ponavlja (za svaku kombinaciju particija ,,9/10+1/10" za obuku i testiranje) i dobija se 10 vrednosti grešaka etiketiranja čija prosečna vrednost i varijansa odgovaraju oceni performansi programa. Finija analiza performansi bi se odnosila na statistike posebno za reči prepoznate u leksikonu, i posebno za one koje su nepoznate programu. U ovom drugom slučaju može nastati pad tačnosti 30-50% (svaki program ima svoja rešenja za nepoznate reči). Međutim, vrhunski (state-of-the-art, [10]) rezultati su trenutno oko 96-97% pri čemu se ne vodi računa o nepoznatim rečima kojih ima uvek oko 10% u korpusu za testiranje (tokom testiranja opisanog u četvrtom poglavlju se pokazuje da je tako). Svi programi su ovde u tom smislu ravnopravno testirani (istim korpusima za učenje i pomoću istih particija 10-fold testiranja). Statistike do kojih se dolazi u ovom tekstu služe pre svega kao pokazatelj odnosa različitih programa, a ne kao mera udaljenosti od vrhunskih rezultata. Ponašanje programa sa nepoznatim rečima je izraženo procentualnim udelom nepoznatih reči koje su pogrešno etiketirane, a na nivou celog korpusa pokazuje se da uspešnost varira 50-60% na poslednjem korpusu u slučaju nepoznatih reči - što je u granicama očekivanog.

2.2 Bajesovo učenje

Bajesovo učenje je osnovni primer algoritama mašinskog učenja koji koriste verovatnoću. Osnovna varijanta je izbor MAP hipoteze (MAP - Maximum a posteriori) najverovatnije uslovne verovatnoće u odnosu na dati skup primera (posmatranja) D i prostor hipoteza H (npr. u smislu mašinskog učenja hipoteza je vektor parametara, a u smislu problema etiketeiranja hipoteza je preslikavanje niza leksema u odgovarajuće etikete - niz leksema predstavlja jednu od instanci učenja, a skup primera je zadat korpusom). Ovo se može posmatrati i kroz opštiji statistički model najverovatnijeg ishoda:

$$\theta = argmax_{\theta} L(x_1, ..., x_n|\theta)$$

- Maximum Likelihood Estimation, gde se za date parametre traže vrednosti koje maksimizuju posmatranu uslovnu verovatnoću (dok kod MAP se dodatno posmatraju i parametri kao slučajne promenljive sa nekom raspodelom). Uz primenu Bajesovog pravila i izuzimanja $P(D)$ kao konstante za različite h:

$$h_{MAP} \equiv argmax_{h \in H} P(h|D) = argmax_{h \in H} \frac{P(D|h)P(h)}{P(D)} = argmax_{h \in H} P(D|h)P(h)$$

Može se i $P(h)$ izostaviti slično $P(D)$ ako se pretpostavi da je a priori verovatnoća hipoteze ista za sve hipoteze (uniformna, na primer, $P(h) = 1/|H|$ za sve $h \in H$).

Kod programa za etiketiranje baziranih na HMM i posebno n-gramima, uobičajeno je da se procenjuje verovatnoća etikete t_n u nizu $t_1, t_2, ..., t_{n-1}, t_n$ i odgovarajuće lekseme (reči) $w_1, w_2, ..., w_{n-1}, w_n$ isto principom maksimalnog ishoda za $p(t_n|t_{n-2}t_{n-1})$:

$$t_n = argmax_{t_n} p(t_n|t_{n-2}t_{n-1}) = argmax_{t_n} \frac{F(t_{n-2}t_{n-1}t_n)}{F(t_{n-2}t_{n-1})}$$

gde je F frekvencija n-grama u datom skupu obučavanja (frekvencija prelaska, ako postoji u leksikonu). Tada važi rekurzivna veza:

$$p(w_1...w_n, t_1...t_n) = p(t_n|t_{n-2}t_{n-1})p(w_n|t_n)p(w_1...w_{n-1}, t_1...t_{n-1})$$

gde je $p(w_n|t_n)$ određena tzv. frekvencijom emitovanja (para leksema-etiketa u leksikonu). Jedan od problema koji se javlja u takvom pristupu su frekvencije jednake nuli, kada program nema dobru informaciju o nepoznatom n-gramu (koji može biti gramatički neispravan ili jednostavno redak). Jedan od načina rešavanja ovog problema je dodeljivanje vrednosti bliskih nuli u takvim slučajevima, i normalizacija ostalih verovatnoća (tako da ukupna

suma bude 1).

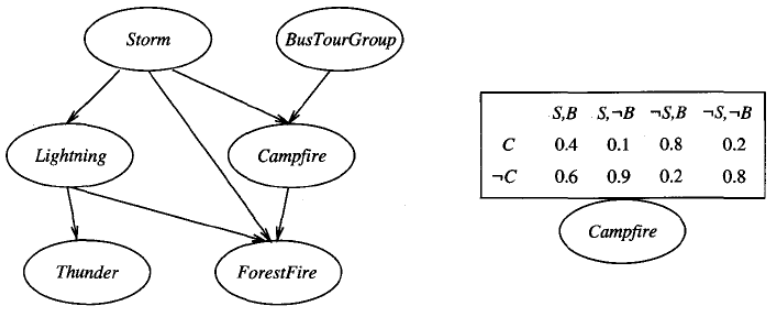
Nasuprot ovom pristupu (koji npr. koristi TnT), Tree Tagger verovatnoću $p(t_n|t_{n-2}t_{n-1})$ ocenjuje drvetom odluke za dati par prethodnih dveju etiketa (tag_{-2}, tag_{-1}) - ovaj metod će biti dodatno pojašnjen u nastavku.

2.2.1 Mreže uverenja

Ako je $P(X|Y,Z) = P(X|Z)$ kaže se da su X i Y uslovno nezavisno pod uslovom Z. Mrežama uverenja (belief networks, ili Bajesove mreže) se definišu višedimenzionalne (složene) raspodele koja se sastoji od n-torke promenljivih $(Y_1, ..., Y_n)$. Mreža uverenja je onda usmereni aciklični graf gde su čvorovi slučajne promenljive u kojem je svaki potomak čvora zavisan od njega, a onaj koji to nije je nezavisan (uz uslov prethodnih čvorova ako ih ima). Tako je rekurzivno definisana složena verovatnoća:

$$P(y_1, ..., y_n) = \prod_{i=1}^{n} P(y_i|Roditelji(Y_i))$$

gde je $Roditelji(Y_i)$ skup neposrednih roditelja čvora Y_i i kombinacija njihovih vrednosti kao u tabeli u primeru ispod:



Slika 1: primer preuzet iz [6]

gde je prvim slovom u tabli skraćeno označen odgovarajući čvor (tabela je vezana za čvor $Campfire$, a čvorovi imaju dve moguće vrednosti - logičke vrednosti). Mreža uverenja je potpuno zadata takvim uslovnim verovatnoćama za svaki čvor (ili npr. faktorima dovoljnosti i potrebnosti za svaki luk da bi se formirali logički iskazi kao pravila, kao što je to u ekspertnom sistemu PROSPECTOR) i početnim (a priori) verovatnoćama. Cilj je obično naći raspodelu verovatnoće n-torke promenljivih na osnovu zadatih raspodela ostalih promenljivih u mreži (i onda možda oceniti njihovu vrednost na osnovu nekih zadatih početnih verovatnoća (dokaza) ili vrednosti promenljivih). Ovakvo verovatnosno zaključivanje je u opštem slučaju pokazano kao

problem NP klase (Cooper, 1990), čak i u aproksimativnim metodama.

Učenje mreža uverenja (tj. njihovih uslovnih verovatnoća ili koeficijenata) na efikasan način se svodi na (naivni) Bajesov klasifikator ako su sve promenljive sa verovatnoćama poznate na osnovu primera, i ako je struktura mreže unapred poznata. Međutim, ako su vrednosti samo nekih promenljivih (ostale su ,,skrivene" - skriveni Markovljevi lanci) raspoložive tokom učenja (u primerima), onda nije tako jednostavno. Problem je donekle uporediv sa problemom obučavanja neuronskih mreža sa skrivenim slojevima (npr. povratnim propagiranjem), kao što se mreže zaključivanja mogu predstaviti kao vrsta neuronskih mreža u krajnjem slučaju. Jedan način je traženjem najverovatnije hipoteze gradijent metodom, odnosno maksimizovanjem $P(D|h)$. Ako je $w_{ijk} = P(Y_i = y_{ij}|U_i = u_{ik})$ uslovna verovatnoća da će promenljiva Y_i imati vrednost y_{ij} ako n-torka njenih direktnih roditelja U_i ima vrednost u_{ik}, pokazuje se da onda važi:

$$\frac{\partial \ln P(D|h)}{\partial w_{ijk}} \equiv \frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|d)}{w_{ijk}} \equiv \sum_{d \in D} \frac{P(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

Ovim se dobija pravilo učenja:

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

gde se mora dodatno voditi računa da je ispunjen preduslov da su w_{ijk} ispravne verovatnoće ($w_{ijk} \in [0, 1]$ i da je $\sum_j w_{ijk} = 1$ za sve i, k), što se radi renormalizacijom koeficijenata nakon svakog ciklusa promene koeficijenata.

Učenje strukture mreže je još teži problem. Jedan način je uvođenje metrike kojom se ocenjuju mreže alternativne strukture (u odnosu na početnu i kasnije radnu strukturu - algoritam K_2, Cooper, Herskovits, 1992), a drugi poznati način je konstruistanjem uslova (constraints) i otkrivanjem relacija zavisnosti na osnovu primera.

2.2.2 EM algoritam

EM algoritam je posebno koristan za primere mreža sa pomenutim skrivenim čvorovima. Primer problema ocene k-sredina (k-means) je jedan od osnovnih problema tog tipa - ako su ulazni podaci zadati kao niz brojeva gde je svaki element niza nasumice uniformno po jednoj od k normalnih raspodela poznatih jednakih disperzija σ^2 i nepoznatih sredina tj. proizvoljan element niza X ima vrednost po nekoj od k normalnih raspodela $\{N(\mu_j, \sigma^2)\}_{j=1}^k$, a

svaki takav element ima i ,,skrivenu" pridruženu vrednost j (indeks uniformno nasumično izabrane normalne raspodele sa odgovarajućom sredinom $\mu_j,\ 1 \leq j \leq k$). Cilj problema je otkriti k-torku $h = (\mu_1, ..., \mu_k)$ sredina kao ispravnu hipotezu za primer od m zadatih vrednosti. Da je skrivena vrednost poznata (da nije skrivena, zapravo) algoritam bi bio jednostavan - za svaki podniz y_i ovakvog niza x_i čijim je vrednostima pridružena samo jedna sredina (i odgovarajuća raspodela) traži se hipoteza μ_{ML} td. je

$$\mu_{ML} = argmin_{\mu} \sum_{i=1}^{m} (y_i - \mu)^2$$

za šta se pokazuje da je rešenje (koje minimizuje ovu sumu) $\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} y_i$ (recimo, kao da je dat niz $(k+1)$-torki $(x_i, z_{i_1}, ..., z_{i_k})$ gde $z_{i_j} = 1$ ukazuje da je za vrednost x_i korišćena normalna raspodela sa sredinom μ_j - ostale vrednosti $z_{i_n} = 0$). EM algoritam za ovaj problem incijalizuje $h = (z_{i_1}, ..., z_{i_k})$ (recimo prvih k vrednosti ulaznog niza ili njihova srednja vrednost) i onda ponavlja naredne korake do stabilnog h (po nekom kriterijumu):

1. izračunaj $E[z_{i_j}],\ j = \overline{1,k}$ uz pretpostavku trenutne h:

$$E[z_{i_j}] = \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{k} p(x = x_i | \mu = \mu_n)} = \frac{e^{-\frac{1}{2\sigma^2}} (x_i - \mu_j)^2}{\sum_{n=1}^{k} e^{-\frac{1}{2\sigma^2}} (x_i - \mu_n)^2}$$

2. izračunaj najverovatniju novu hipotezu $h' = (\mu'_1, ..., \mu'_k)$ uz pretpostavku da je trenutna vrednost $z_{i_j},\ j = \overline{1,k}$ je prethodno izračunata $E[z_{i_j}]$ i zameni h sa h':

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{i_j}] x_i}{\sum_{i=1}^{m} E[z_{i_j}]}$$

Suština algoritma su ova dva koraka, gde se najpre koristi trenutna hipoteza da bi se ocenile skrivene vrednosti, a onda se tako dobijenim skrivenim vrednostima računa iterativno bolja hipoteza. EM algoritam obavezno konvergira ka lokalno maksimalno verovatnoj ($P(D|h)$) hipotezi na taj način. Uobičejena uopštena formulacija je da se za skup posmatranih vrednosti (observed) $X = \{x_1, ..., x_m\}$ i skrivenih (hidden, unobserved) $Z = \{z_1, ..., z_m\}$ ($Y = X \bigcup Z$ je ukupan, pun skup podataka) posmatra skup parametara θ koji je od interesa za problem. Z se može posmatrati kao slučajna promenljiva (a time i Y) čija raspodela zavisi od nepoznatih parametara θ i poznatih podataka X. Traži se h' tako da maksimizuje $E[\ln P(Y|h')]$, tj. $Q(h'|h) = E[\ln P(Y|h')|h, X]$ (kao funkcija od h' pod pretpostavkom $\theta = h$). Tada se uopšteni EM algoritam svodi na sledeća dva koraka:

procena (estimation) (E): računanje $Q(h'|h)$

maksimizovanje (maximization) (M):

$$h \leftarrow argmax_{h'} Q(h'|h)$$

Ako je Q neprekidna onda EM konvergira ka stacionarnoj tački funkcije $P(Y|h')$ (slično gradijent metodama). U slučaju problema k-sredina važi: $\theta = (\mu_1, ..., \mu_k)$, $p(y_i|h') = p(x_i, z_{i_1}, ..., z_{i_k}|h') = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{i_j}(x_i - \mu'_j)^2}$, pa je $\ln P(Y|h') = \ln \prod_{i=1}^{m} p(y_i|h') = \sum_{i=1}^{m} p(y_i|h')$, i na osnovu toga je:

$$Q(h'|h) = E[\ln P(Y|h')] = \sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} E[z_{i_j}](x_i - \mu'_j)^2 \right)$$

pa je: $argmax_{h'} Q(h'|h) = argmin_{h'} \sum_{i=1}^{m} \sum_{j=1}^{k} E[z_{i_j}](x_i - \mu'_j)^2$.

2.3 Lanci Markova

Lanac Markova se definiše za sistem u kome se nizom slučajnih promenljivih X_n (u trenutku n, takav niz npr. obično opisuje evoluciju fizičkog sistema kroz vreme, ili u slučaju etiketiranja predstavlja nisku leksema kao deo datog teksta) sa diskretnim vrednostima $\{x_1, x_2, ...\}$ - *stanjima* sistema (koja se mogu poistovetiti sa etiketama u PoS). Niz je *lanac Markova prvog reda* (ili samo lanac Markova) ako postoji *zavisnost Markova*: ako za svaki konačan skup $\{m, n_1, ..., n_r\}$ trenutaka takvih da je $0 \leq n_r < n_{r-1} < ... < n_1 < m$ i za svaki skup stanja $\{x_{i_r}, x_{i_{r-1}}, ..., x_{i_1}, x_j\}$ važi:

$$P(\{X_m = x_j\}|\{X_{n_1} = x_{i_1}\} \cap ... \cap \{X_{n_r} = x_{i_r}\}) = P(\{X_m = x_j\}|\{X_{n_1} = x_{i_1}\})$$

Za kontinualni skup trenutaka (realan ili čak kompleksan) imamo *proces Markova*. Verovatnoća $p_{ij}(n, m) = P(\{X_m = x_j\}|\{X_n = x_i\})$ je *verovatnoća prelaza* iz stanja x_i u trenutku n u stanje x_j u trenutku m. Ako zavisi samo od $n - m$ onda je lanac Markova *homogen* i može se skraćeno zapisati:

$$p_{ij}(n) = P(\{X_{k+n} = x_j\}|\{X_k = x_i\}), \quad p_{ij} = p_i j(1)$$

(verovatnoća prelaska u n koraka i jednom koraku), i često se zapisuje u matričnom obliku $P_n = [p_{ij}(n)]$, $P = P_1$. Verovatnoća prelaska za više koraka $n = 2, 3, ...$ se određuje jednačinama Čepmen-Kolmogorova:
$p_{ij}(n) = \sum_k p_{ik}(m) p_{kj}(n - m)$, $1 \leq m < n$ - tj. $P_n = P_m P_{n-m}$ (specijalno za $m = 1$: $P_n = P^n$). Ako je početna raspodela verovatnoća stanja $p_i(0)$ onda

za naredne trenutke $p_i(n) = P(\{X_n = x_i\})$ važi: $p_i(n) = \sum_k p_k(n-1)p_{ki}$ (potpunom verovatnoćom). Ako se uvede zavisnost $p_{ij} = p_{ij}(d)$ za neki parametar $d \in D$, i za svaki vremenski korak (trenutak) i stanje važi $d = d(x_i, n)$ *funkcija odlučivanja*, dobija se *problem optimalnog upravljanja Markova*: naći funkciju optimalnog odlučivanja d^0 tako da je $P(d^0) = \max_n P(X_n \in S)$, tj. $d^0 = argmax_d P(d)$ tako da sistem u trenutku n s najvećom verovatnoćom uđe u zadati podskup stanja S. Funkcija optimalnog odlučivanja $d^0 = d^0(x_i, k)$ se određuje rekurzivno Belmanovom jednačinom:

$$P^0(x_i, k) = \max_d \sum_j p_{ij}(d) P^0(x_j, k+1), \quad k = 0, ..., n-1$$

... pošto je u koraku $k+1$ nađeno $d^0(x_j, k+1)$ i $P^0(x_j, k+1)$, gde je

$$P^0(x_i, n-1) = P(x_i, n-1, d^0) = \max_d P(x_i, n-1, d)$$

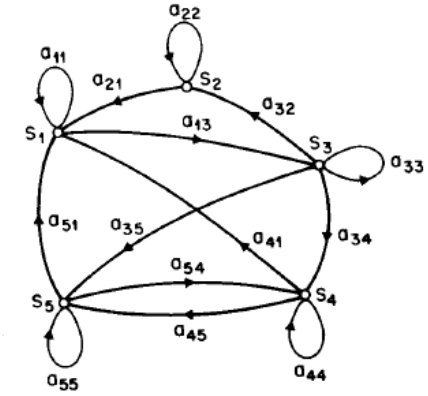
za neko $d = d^0$ i

$$P(x_i, n-1, d) = \sum_{j:\ x_j \in S} p_{ij}(d)$$

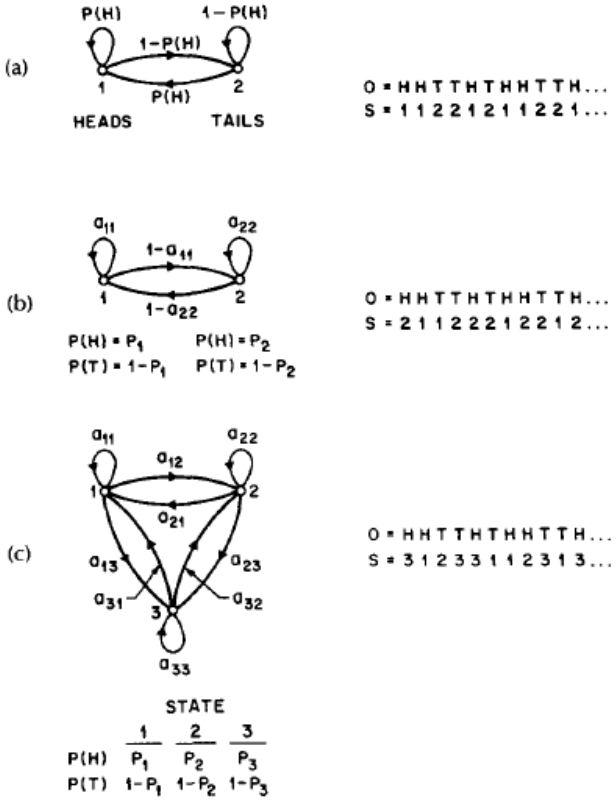
2.3.1 Skriveni lanci Markova

Skriveni lanac Markova (SLM, ili HMM u daljem tekstu) deli događaje, odnosno stanja i odgovarajuće promenljive, u skrivena X i posmatrana Y - na primer, skriveni događaj može biti fonema koja je izgovorena u govornom signalu ili neki od njegovih parametara, ili leksema i etiketa reči u rečenici tekstuelnog ulaza (odnosno etiketa), a posmatrani događaj je onda ili etiketa (za PoS), ili prepoznata reč u rečenici (za WSD). Model SLM $\theta = (X, Y, \pi)$ je tako određen raspodelom prelaska za X i Y (model se može shvatiti i kao niz parametara, odnosno ovih verovatnoća) i raspodelom početnih stanja π. Niz posmatranih stanja zavisi samo od trenutnog jednog stanja u nizu skrivenih stanja (u smislu uslovnih verovatnoća), a svako skriveno stanje zavisi samo od trenutnog i prethodnih u nizu. Primer: jedan čovek baca novčić ali posmatrač može samo da čuje od njega na koju stranu je pao, ne zna čak ni koliko različitih novčića može učestvovati u eksperimentu (i time rezličitih modela). Tako svakoj etiketi ili reči u rečniku može biti posvećen poseban lanac sa skrivenim stanjima koja odgovaraju prethodnim rečima ili etiketama (mogu se i paralelno obrađivati). Jezički model može tako biti zadat grafovima prelaska stanja u vremenu (slično grafovima kontekstno slobodnih gramatika) čiji su lukovi, tzv. *bigrami*, obeleženi verovatnoćom, odnosno trigrami ako

prethodna dva stanja utiču na izbor novog, itd. Uopšte, n-grami kao podniske niza leksema koje čine ulazni tekst sa svojim frekvencijama u korpusu su važan alat za dobijanje verovatnoća i efikasno obučavanje HMM. Ispod je primer modela među kojima su i modeli pomenutog eksperimenta bacanja novčića:



Slika 2a



Slika 2b

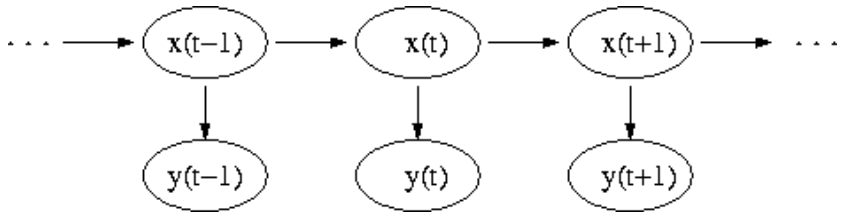
Osnovna pitanja su:

- Za dati niz posmatranja koja je njena verovatnoća u odnosu na dati model ?
- Kako odabrati tj. dekodirati najoptimalniji (najverovatniji) odgovarajući niz skrivenih stanja (objašnjenje) za dati niz posmatranja i model ?
- Kako menjati (učiti) parametre modela tako da se maksimizuje pomenuta verovatnoća niza posmatranja ?

Poslednje pitanje se tiče učenja, a pretposlednje problema PoS etiketiranja. Verovatnoća posmatranja je data sa $P(Y) = \sum_X P(Y|X)P(X)$ (gde su X svi nizovi skrivenih stanja). Ovo je osnova algoritma računanja unapred (*forward* algoritam) koji se može ovako opisati za dati HMM - ako je $(O_1, ..., O_M)$ niz posmatranja u vremenu, N broj stanja, $y_j(O_t) = P(Y_j = O_t)$ verovatnoća posmatranja O_t u stanju j, x_{ij} verovatnoća prelaska iz stanja i (u trenutku $t-1$) u stanje j, $\alpha_t(j) = P(O_1, ..., O_t, X_t = j)$ verovatnoća da se nakon t koraka HMM nalazi u stanju j:

1. incijalizacija: $\alpha_1(j) = x_{0j}y_j(O_1)$, $1 \leq j \leq N$
2. rekurzija: $\alpha_t(j) = \sum_{i=1}^{N-1} \alpha_{t-1}(i)x_{ij}y_j(O_t)$, $1 \leq j \leq N$, $1 \leq t \leq M$
3. rezultat: $P(Y) = \alpha_M(N) = \sum_{i=1}^{N-1} \alpha_M(i)x_{iN}$

Na druga dva pitanja ne postoji ovako egzaktna vrednost kao odgovor već procene koje se npr. dobijaju Viterbi algoritmom, odnosno Baum-Velčovim algoritmom, respektivno.



Slika 3

Više o konkretnim metodama i detaljima u [2].

2.3.2 Viterbi algoritam

Ovaj algoritam nalazi verovatnoću najverovatnijeg niza posmatranja, ali pre svega pronalazi niz odgovarajućih skrivenih stanja lanca Markova, tzv. Viterbijevu putanju - dinamičkim programiranjem. Postoje i mnoge gotove

biblioteke za rad sa HMM. Slično forward algoritmu, rekurzivno se računa najbolja ocena $v_t(j) = \max_{1 \leq i \leq N-1} v_{t-1}(i) x_{ij} y_j(O_t)$. Pseudo-kodom bi to moglo ovako da se predstavi - ulazni argumenti su: y niz posmatranja, X je niz skrivenih stanja, *sp* su početne verovatnoće (verovatnoće početnih skrivenih stanja), *tp* su verovatnoće prelaska, *ep* verovatnoće posmatranja. Niz T za dato stanje daje trojku koja se redom sastoji od ukupne verovatnoće svih putanja do trenutnog stanja, putanje i njene verovatnoće, a niz U sadrži narednu takvu trojku:

```
def forward_viterbi(y, X, sp, tp, ep):
   T = {}
   for state in X:
       T[state] = (sp[state], [state], sp[state])
   for output in y:
       U = {}
       for next_state in X:
           total = 0
           argmax = None
           valmax = 0
           for state in X:
               (prob, v_path, v_prob) = T[state]
               p = ep[state][output] * tp[state][next_state]
               prob *= p
               v_prob *= p
               total += prob
               if v_prob > valmax:
                   argmax = v_path + [next_state]
                   valmax = v_prob
           U[next_state] = (total, argmax, valmax)
       T = U
   ## primena sum/max nad stanjima na kraju:
   total = 0
   argmax = None
   valmax = 0
   for state in X:
       (prob, v_path, v_prob) = T[state]
       total += prob
       if v_prob > valmax:
           argmax = v_path
           valmax = v_prob
   return (total, argmax, valmax)
```

2.3.3 Baum-Velč-ov algoritam

Pored ovog algoritma se koristi Baum-Velčov algoritam (Baum-Welch) koji predstavlja primenu pomenutog EM algoritma koristi računanje napred-unazad (forward-backward) verovatnoća prelaska (u oba vremenska pravca) i primer je obučavanja parametara SLM modela. Određivanjem frekvencije parova prelaska i posmatranja u odnosu na ukupnu verovatnoću putanje određuju se verovatnoće prelaska i posmatranja metodom *maksimizovanja očekivanja*. Ukratko, u prvom koraku ove metode se računaju procene očekivanja verovatnoće prelaska skrivenog stanja uslovljenog posmatranjima

$$P(X|Y,\theta) = \frac{P(Y,X|\theta)}{P(Y|\theta)} = \frac{P(Y|X,\theta)P(X|\theta)}{\int P(Y|X,\theta)P(X|\theta)}$$

a u drugom se maksimizuju, i to se iterativno ponavlja:

$$\theta_{n+1} = argmax_\theta Q(\theta)$$

gde je:

$$Q(\theta) = E_x[\log P(Y,X|\theta)|Y] = \int_\infty^\infty p(x|y,\theta_n) \log p(y,x|\theta)dx$$

Praktični detalji se mogu naći npr. u [3], [2] i posebno u [4].

2.4 Statističko učenje, modeli najverovatnijeg ishoda i maksimalne entropije

Sve pomenute metode i poznate metode matematičke verovatnoće i statistike nude korisne alate u problemima mašinskog učenja. To se može i ovako posmatrati: ako je X slučajna promenljiva nad n-torkama gde je $p(X)$ (a priori) verovatnoća da je to n-torka koju tražimo da klasifikujemo, odnosno gustina raspodele slučajne promenljive X, onda se može sa $p(X|1)$ i $p(X|2)$ označiti gustina raspodele za kategorije 1 i 2 ($p(i)$ je a priori verovatnoća kategorije), redom (dalje se sve može uopštiti jednostavno za proizvoljan broj kategorija) - raspodele za te dve kategorije se mogu razlikovati. Dve osnovne greške se mogu napraviti: da je data n-torka (uzorak, šema) u kategorji 1 iako pripada kategoriji 2, i obratno. *Ozbiljnost* napravljene greške se ocenjuje *funkcijom gubitka* (loss function) $\lambda(i|j)$ za odabranu kategoriju i ako je kategorija j tačna ($\lambda(i|i) = 0$). Očekivana vrednost gubitka za dato x je $L_X(i) = \lambda(i|1)p(1|X) + \lambda(i|2)p(2|X)$. Kategoriju 1 biramo ako je $L_X(1) \leq L_X(2)$, a upotrebom Bajesovog pravila dobija se:

$$\lambda(1|2)p(X|2)p(2) \leq \lambda(2|1)p(X|1)p(1)$$

Specijalno, ako je $\lambda(1|2) = \lambda(2|1)$ i $p(1) = p(2)$ onda je taj uslov: $p(X|2) \leq p(X|1)$ (tj. izbor najverovatnije kategorije u X modelom najverovatnijeg ishoda - MLE, Maximum Likelihood Estimation), gde se koristi i opštije $k(i|j) = \lambda(i|j)p(j)$. Naravno, za proizvoljan skup kategorija C se može zadatak definisati uopšteno, gde se bira kategorija $j \in C$ za koju važi:

$$j = argmin_{i \in C} L_X(i)$$

Konkretan kriterijum se dobija izborom konkretne pretpostavljene raspodele. Jedan pristup i statistički koncept učenja koji može u nekim slučajevima biti dualan ovakvom je metod modela maksimalne entropije (Maximum Entropy Principle). Ovakav statistički model koristi program za etiketiranje MXPOST (Adwait Ratnaparkhi, 1996) - ako je $\mathbb{A}$ skup klasa koje treba prepoznati (npr. skup etiketa), $\mathbb{B}$ skup konteksta (ili istorija) u kojima se mogu naći klase (skup instanci, npr. delovi rečenice u odnosu na reč za koju se traži etiketa), $c : \mathbb{B} \to \{\top, \bot\}$ karakteristična funkcija u skupu konteksta (ovakvih kontekstnih predikata ima prebrojivo mnogo: $c_1, ..., c_m$, $m \leq |\mathbb{B}|$), onda se *osobina* $f_{c,a'}$ definiše kao preslikavanje:

$$f_{c,a'}(a, b) = \begin{cases} 1 & a = a',\ c(b) = \top \\ 0 & \text{inače} \end{cases}$$

i onda se može definisati uslovna verovatnoća:

$$p(a|b) = \frac{1}{Z(b)} \prod_{j}^{k} {\alpha_j}^{f_j(a,b)}$$

gde je $Z(b) = \sum_{a \in \mathbb{A}} \prod_{j}^{k} {\alpha_j}^{f_j(a,b)}$ činilac normalizacije, $f_j(a, b)$ ima vrednost 0 ili 1 ako dati par (a, b) ima datu osobinu j, i $\alpha_j > 0$ predstavljaju težinske koeficijente. Ovi koeficijenti se mogu odrediti kao MLE problem kojim se traži raspodela p^* tako da odgovara skupu obučavanja ($\widehat{p}(a, b)$ je verovatnoća da se par (a, b) nalazi u skupu obučavanja, Q je skup modela koji odgovaraju datom skupu obučavanja tj. odgovarajućih raspodela) i važi:

$$Q = \{p|\ p(a|b) = \frac{1}{Z(b)} \prod_{j}^{k} {\alpha_j}^{f_j(a,b}\}$$

$$L(p) = \sum_{a,b} \widehat{p}(a, b) \log p(a|b)$$

$$p^* = argmax_{q \in Q} L(q)$$

Model maksimalne entropije (Haynes, 1957, Good, 1963) je onda dat sledećim uslovima ($H(p)$ je uslovna entropija u srednjem nad skupom obučavanja):

$$P = \{p|\ E_p f_j = E_{\widehat{p}} f_j,\ j = \overline{1,k}\}$$

$$H(p) = -\sum_{a,b} \widehat{p}(b) p(a|b) \log p(a|b)$$

$$E_{\widehat{p}} f_j = \sum_{a,b} \widehat{p}(a,b) f_j(a,b)$$

$$E_p f_j = \sum_{a,b} \widehat{p}(b) p(a|b) f_j(a,b)$$

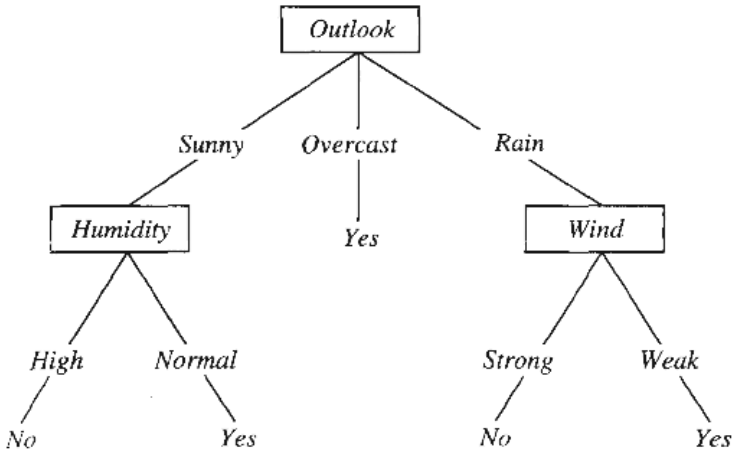
$$p^* = argmax_{p \in P} H(p)$$

2.5 Učenje drvetom odluke

Metod sličan prethodno opisanom učenju koncepta koji koristi sistem i algoritam ID3 (kao i ASSISTANT i C4.5) koji generiše pravilo, odnosno drvo klasifikacije za dati koncept sa svojim atributima i njihovim vrednostima. Njegov induktivni bias je prednost malih drveća nad većim, a u stanju je da klasifikuje i disjunktivne koncepte (bias je implicitno ,,znanje" ugrađeno u sam algoritam i metod učenja dato njegovom prirodom s jedne strane, i prirodom strukture prostora hipoteza koji se pretražuje s druge strane - logički i konceptualni bias). Ovakav metod može da bude daleko efikasniji od drugih sistema za induktivno učenje, ali i neprimenjiv u nekim složenim domenima. Postoje varijante (bagging, random forest) koje uporedo razvijaju više takvih drveta.

2.5.1 Reprezentacija

Učenje drvetom odluke predstavlja vid aproksimacije funkcije (diskretne vrednosti) atributa i njihovih diskretnih vrednosti. Drvo odluke klasifikuje instancu prihvatajući atribute od korena do lista jedne grane, a može se posmatrati i kao spisak ako-onda pravila (svaka grana predstavlja konjunkciju uslova nad atributima, a celo drvo disjunkciju). Primer (Quinlan, 1986, za ID3) drveta za *PlayTennis* koncept:



Slika 4: primer takođe preuzet iz [6]

koji predstavlja izraz:

$$(Outlook = Sunny) \wedge (Humidity = Normal)$$
$$\vee\ (Outlook = Overcast)$$
$$\vee\ (Outlook = Rain \wedge Wind = Weak)$$

Kao što se vidi, instance su predstavljene listama parova atribut-vrednost, ciljna funkcija u primeru je Bulova ali se može proširiti na diskretnu ili čak realnu funkciju. Instance primera mogu sadržati greške (bilo u vrednosti atributa ili klasifikacije) ili nedefinisane vrednosti atributa. Glavno pitanje je od kojeg atributa krenuti s klasifikacijom - cilj je izabrati najkorisniji atribut (slično biranju instance koja polovi prostor verzija u problemu formacije koncepta), i to se čini ocenjivanjem statističke osobine *informacione dobiti (snage*, information gain) koja se definiše entropijom skupa S:

$$E(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

gde je $p_{\oplus}$ proporcionalan odnos pozitivnih instanci u S, a $p_{\ominus}$ odnos negativnih u S (ima vrednost nula ako svi primeri pripadaju istoj klasi). Ako ciljni atribut u opštem slučaju ima c diskretnih vrednosti onda je
$E(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$. Ako je $Values(A)$ skup vrednosti atributa A i $S_v = \{s \in S :\ A(s) = v\}$ (skup instanci gde A ima vrednost v) onda je:

$$Gain(S, A) \quad \equiv \quad E(S) \ - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} E(S_v)$$

Sličan pristup koristi i Tree Tagger program za etiketiranje, s tim da se kod njega u listovima drveta odluke ne nalaze samo obeležja etiketa koje

se klasifikuju (u trigramima etiketa), već čitavi vektori verovatnoća etiketa (dobijeni kao odnos frekvencije trigrama koji ispunjavaju kriterijum i ukupnog broj trigrama korpusa).

Verzija ID3 algoritma za Bulove funkcije (CLS algoritam, Hunt, 1966):

ID3(S, c, *atributi*)

1. kreiraj čvor $Koren$ stabla
2. ako su svi primeri u S pozitivni,
   vrati $Koren$ sa oznakom=+
3. ako su svi primeri u S negativni,
   vrati $Koren$ sa oznakom=−
4. ako je *atributi* lista atributa koje treba testirati prazna,
   vrati $Koren$ sa oznakom=najčešća vrednost u primerima
5. inače:
   (a) $A \leftarrow$ iz *atributi* liste atribut koji najbolje klasifikuje prema $Gain(S, A)$,
   $Koren \leftarrow A$
   (b) za svaku vrednost v_i atributa A:
      i. dodaj novu granu ispod $Koren$ za uslov $A = v_i$
      ii. ako je S_{v_i} prazan
         - onda: ispod dodaj list sa oznakom=najčešća vrednost u primerima
         - inače: dodaj ispod poddrvo ID3(S_{v_i}, c, $atributi - A$)
6. vrati $Koren$

Ovakav ID3 algoritam se lako može uopštiti za funkciju diskretnih vrednosti (oznaka), i koristi se obično prozor koji čini podskup skupa primera (zadate veličine) nad kojim se primeni ovakav algoritam, a onda se u prozor unesu svi izuzeci iz skupa primera koji ne odgovaraju klasifikaciji i ako ih ima ponovi se postupak. Time se prevazilazi ograničenje veličine skupa primera. Induktivni bias ID3 algoritma koji proističe iz redosleda i prostora pretrage (bias preferencije ili pretrage) je zapravo taj da preferira kraće drveće sa čvorovima veće informacione snage bliže korenu, dok konceptualnog biasa nema (sve konačne diskretne funkcije dolaze u obzir jer se mogu predstaviti nekim drvetom odluke). Zato je ID3 nema problem konceptualnog biasa, i

pošto koristi statističke ocene svih primera daleko je manje osetljiv na greške nego formacija koncepta, a lako se može omogućiti i da prihvata hipoteze koje ne odgovaraju primerima sasvim tačno. Neki sistemi mogu kombinovati biase restrikcije i pretrage, kao kod primera učenja funkcije evaluacije igre i LMS (Least Mean Square) učenja.

2.5.2 Okamova oštrica i problem prezasićenja (overfitting) kod učenja

Da li je ID3 bias opravdan ? Okam (William of Occam, 1320) je to formulisao otprilike ovako: bolja je uvek jednostavnija hipoteza koja odgovara podacima (eksperimentima, primerima). Fizičari vole ovu hipotezu - čisto kombinatornim argumentima može se pokazati da kraćih hipoteza ima daleko manje nego dužih - ali postoje i kritike: postoje i druge malobrojne klase hipoteza (veštačke), a rezultat zavisi i od interne reprezentacije hipoteze (obično se onda koristi evolucioni argument da biva izabrana interpretacija koja ovaj kriterijum čini uspešnijim).

Za hipotezu h se kaže da overfituje (overfitting) primere ako ima manju grešku nego neka druga hipoteza h' nad primerima, ali veću nad ukupnom raspodelom instanci (van skupa primera, poznato i kao problem gubitka sposobnosti generalizacije kod učenja). Problem nastaje kada postoje nasumične greške u podacima kada nakon određenog broj primera počinje da opada ukupna tačnost klasifikacije iako raste na skupu primera. Jedan način je da se zaustavi rast drveta pre idealne klasifikacije primera, drugi je da se nakon klasifikacije potkreše drvo (post-pruning). Kriterijumi u oba slučaja mogu biti:

- obuka i validacija - izdvajanje iz skupa primera jedan podskup na kome se testira tačnost izvan skupa primera
- χ-test ili neki drugi statistički test procene performanse nad celom raspodelom instanci (Quinlan, 1986)
- princip minimalne dužine opisa (Minimum Description Length - MDL, Quinlan i Rivest 1989, Mehta 1995) - kodiranjem drveta i primera na neki način dobija se mera složenosti tj. veličina kodiranja - kada je veličina minimizovana prekida se rast drveta odluke

Kresanje se vrši zamenom poddrveta listom sa najčešćom (prosečnom) klasifikacijom sve dok se time ne dobija manje performantno drvo odluke. ID3 drži samo jednu hipotezu tokom pretrage za razliku od formacije koncepta, u osnovnom obliku nema backtracking i zato je moguće da nađe rešenje koje

nije globalno najbolje iako je obično performantniji od prethodnog. Ovo se takođe rešava potkresivanjem.

Postoji tehnika *potkresivanja pravila* (rule post-prunning, Quinlan 1993) u sistemu C4.5 skicirana u sledećim koracima:

1. generiši drvo uz moguće overfitovanje
2. pretoči drvo odluke u niz pravila (za svaku granu, redom)
3. izbaci iz pravila (generalizuj) preduslove ako se time ne narušavaju performanse
4. sortiraj dobijena pravila prema ocenjenoj tačnosti i razmatraj ih tim redosledom tokom klasifikacije instanci

Slično ovome, moguće je drvo odlučivanja pretočiti u optimizovani graf (odlučivanja) kojim se pre svega eliminiše redundantno poddrveće i time smanjuje nepotreban broj primera.

Ograničenje da atributi moraju biti diskretne vrednosti se može prevazići dodelom karakterističnih intervala kontinualnim vrednostima, gde se za svaku diskretnu vrednost ciljnog atributa nalazi prag ocenom najveće informacione snage primera sortiranih po atributu koji se ocenjuje (Fayyad 1991).

Postoji problem kod atributa kao što je datum - iako nosi veliku informacionu snagu (informacioni dobitak, information gain), obično razdvaja primere u male grupe bez nekog velikog uticaja na vrednost ciljnog atributa. Jedno rešenje je da se koristi alternativna ocena atributa - npr. odnos snage (gain ratio, Quinlan 1986) koji je osetljiv na uniformno deljenje primera:

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i}{|S|}$$

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

Svaki atribut može prema nekim kriterijumima imati dodatno cenu $Cost(A)$ gde je cilj da se atributi sa većom cenom koriste prilikom pouzdane klasifikacije (da bi se povećala zahtevana tačnost). Tada se koristi umesto informacione snage $\frac{Gain^2(S,A)}{Cost(A)}$ ili $\frac{2^{Gain(S,A)}-1}{(Cost(A)+1)^w}$ gde je $w \in [0, 1]$ parametar koji ocenjuje značaj cene u odnosu na informacionu snagu (Tan, Schlimmer 1990/93, Nunez 1991).

Ako ne postoji vrednost nekog atributa A u čvoru u kome treba oceniti informacionu snagu može se koristiti:

- najčešća vrednost primera u tom čvoru
- najčešća vrednost primera sa vrednošću ciljnog atributa datog primera
- umesto najčešće vrednosti može se koristiti vrednost s najvećom procenjenom verovatnoćom primera u datom čvoru (Quinlan 1993)

Kod statističkih metoda se prezasićenje često rešava dodatnim uslovima (regularizacijom) objektivnih funkcija (čijom se optimizacijom uče parametri modela) - ,,peglanjem" (smoothing) vrednosti koje u suštini ne doprinose informacionom dobitku.

2.6 Učenje instancama, metode klasifikacije

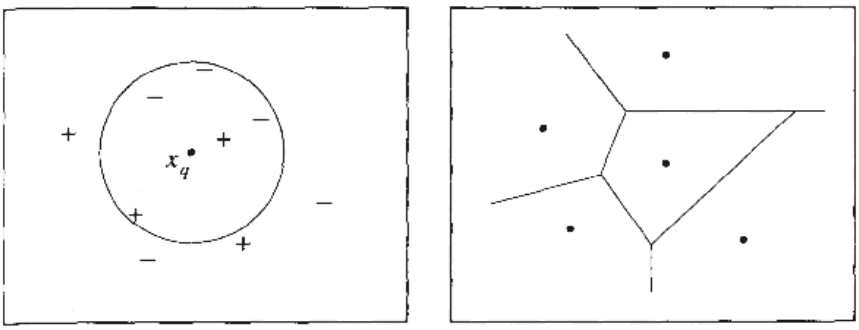
Naspram ranije pomenutih metoda, učenje instancama pamti primere i generalizuje ih tek unosom nove instance - do tada je ono *odloženo*, i zato se ove metode nazivaju i ,,lenjim" (lazy) - nasuprot njima su ,,radoznale" (eager) metode. Osnovna prednost metoda učenja s odloženim uopštavanjem je njihova prilagodljivost novim primerima.

2.6.1 Metod najbližih suseda

Jedan metod učenja instancama (koji se smatra takođe i statističkim) je metod *najbližih suseda* ili metod memorije (Dasarathy 1991, Moore 1994). Data je funkcija $f : \mathbb{R}^n \to V$ gde je $V = \{v_i\}$ skup kategorija diskretnih vrednosti i funkcije $a_j : X \to \mathbb{R}$ koje dodeljuju vrednost j-tog atributa instance td. je skup primera $D = \{(x_i, f_i)\}$ (kompozicija $f_i = f(a_1(x_i), ..., a_n(x_i))$ se skraćeno obeležava sa $f(x_i)$), i onda se za instancu (upit) x_q ocenjuje njena kategorija $f(x_q)$. Odabiranjem najbližih k suseda vektoru koji se testira (kNN, k Nearest Neighbours) u odnosu na skup primera po nekoj metrici, na primer euklidskoj ($d(x_1, x_2) = \sqrt{\sum_{j=1}^{n} (a_j(x_1) - a_j(x_2))^2}$, ovde npr. a_j vrše korekciju koeficijentima koji se biraju tako da po svakoj dimenziji daju približno ujednačen uticaj svakog atributa - ako su vektori normirani bitan je samo ugao i onda se može posmatrati njihov skalarni proizvod), bira se kategorija koja je najbrojnija kao ocena kategorije ($\delta(a, b) = 1$ ako $a = b$, inače je $\delta(a, b) = 0$):

$$\widehat{f}(x_q) \leftarrow argmax_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

Što je gustina tačaka oko instance upita veća, poželjnija je veća vrednost k. Ova metoda traži dosta memorije za primere, postoje efikasne primene (kd-tree method, Friedman, 1977), a još su Kaver i Hart (Cover, Hart 1967) pokazali da je metoda 1-najbližeg suseda po performansama približna i vezana za metodu minimalne verovatnoće greške (veće k podrazumeva i nejasniju granicu između kategorija klasifikacije). Ako se razdaljina definiše nekom funkcijom jezgra ($d(x_1, x_2) = K(x_1, x_2) - 2K(x_1, x_1) + K(x_2, x_2)$), onda je to KNN (Kernel Nearest Neighbours) metoda. Ispod je prikazana granica među kategorijama za metodu 1-najbližeg suseda što predstavlja reprezentaciju implicitno naučene hipoteze (ovakvi se dijagrami nazivaju *Voronoj dijagramima*):



Slika 5

Varijanta aproksimacije neprekidne funkcije ovakvim algoritmom se dobija ocenom:

$$\widehat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Očigledno poboljšanje algoritma je da se daje prednost bližim primerima prilikom glasanja, što se može postići sa:

$$\widehat{f}(x_q) = argmax_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i)),\ w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

tako da ako se x_q poklopi sa nekom instancom x_i iz skupa primera onda se dodeljuje $\widehat{f}(x_q) \leftarrow f(x_i)$ po definiciji (ako ima više vrednosti u skupu primera za istu instancu onda se dodeljuje najbrojnija vrednost). U neprekidnom slučaju to je:

$$\widehat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

Pomenuti težinski koeficijenti za vrednosti a_j mogu biti korisni ako neki atributi nisu relevantni (dve instance su veoma udaljene zbog atributa koji ne

utiču toliko na rezultat, a imaju istu vrednost - ,,prokletstvo dimenzionalnosti"). Odgovarajuća statistička terminologija:

- *regresija* predstavlja aproksimaciju neprekidne funkcije realne vrednosti
- realna funkcija $K(x, y)$ je *jezgro* ako je nenegativna, parna i integrabilna (kao raspodela, poželjno je i da bude neprekidna, simetrična i pozitivna semi-definitna kvadratna forma kojom se onda može definisati skalarni proizvod po Mercerovoj teoremi), kako kod KNN tako i kod drugih sličnih metoda
- *rezidualna greška* aproksimacije je $\widehat{f}(x) - f(x)$

2.6.2 Lokalno-težinska regresija

Jedno uopštenje prethodnog metoda je aproksimacija f u okolini x_q (u smislu odabranih primera) nekom funckijom: linearnom (linearna regresija), kvadratnom funkcijom, višeslojnom neuronskom mrežom, itd. Lokalno-težinska linearna regresija tako koristi $\widehat{f}(x) = w_0 + w_1 a_1(x) + ... + w_n a_n(x)$. Ako se greška definiše tako da se dobije lokalna aproksimacija, mogući su pristupi:

- $E_1(x_q) \equiv \frac{1}{2} \sum_{x \in D_k(x_q)} (f(x) - \widehat{f}(x))^2$ gde je $D_k(x_q)$ skup k primera iz D najbližih x_q
- $E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \widehat{f}(x))^2 K(d(x_q, x))$
- kombinacija prethodna dva: $E_3(x_q) \equiv \frac{1}{2} \sum_{x \in D_k(x_q)} (f(x) - \widehat{f}(x))^2 K(d(x_q, x))$

i tada je npr. za E_3 gradijent pravilo:

$$\Delta w_j = \eta \sum_{x \in D_k(x_q)} (f(x) - \widehat{f}(x)) a_j(x)$$

Obično se ne koriste složenij forme aproksimacije od linearnih i kvadratnih jer su ostale računski daleko skuplje, a navedene na dovoljno maloj okolini tačke upita imaju u većini slučajeva zadovoljavajuće performanse.

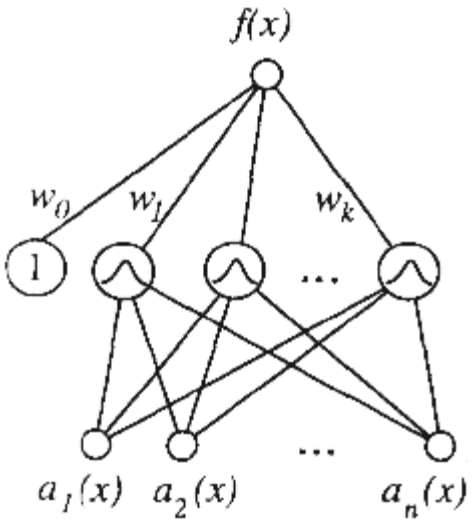
2.6.3 Funkcije radijalne baze

Ovaj pristup je blizak prethodnom Ima ,,ugrađeno" leksičko pravilo za reči koje nisu u leksikonu a koje počinju velikim slovom - smatraju se vlastitim imenima (inače obična imenica), nakon čega se primenjuje pot-program za nepoznate reči (ukratko, sufiksi i prefiksi dužine do 4 karaktera se uzimaju u obzir i traže podreči u leksikonu i uče se nova pravila ako

su nađene).i učenju NM - funkcija hipoteza, funkcije radijalne baze - FRB (Radial Basis Function - RBF), je oblika:

$$\widehat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

gde je x_u instanca u X i K_u jezgro takvo da mu vrednost opada sa rastom rastojanja $d(x_u, x)$. Uobičajen izbor jezgra je Gausovo $K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$ (gustina normalne raspodele sa sredinom u x_u i disperzijom σ_u^2, mada to može biti kao kod Furijeove transformacije - trigonometrijski polinom ili funkcija kompleksnog argumenta). Funkcije radijalne baze se mogu posmatrati kao kao dvoslojna (neuronska) mreža gde se u prvom sloju nalaze jezgra, a u drugom sloju njihova linearna kombinacija. FRB predstavljaju glatku linearnu kombinaciju lokalnih aproksimacija ciljne funckije. Obučavanje se obično vrši u dve faze: najpre se određuje broj k skrivenih jedinica i njihovih parametara x_u i σ_u^2, a onda se se u drugoj fazi određuju koeficijenti w_u tako da se minimizuje globalna greška E. Jedan pristup od mnogih je da se svakom primeru dodeli jedno jezgro (jednake disperzije σ^2) tako da je FRB tačna za svaki primer ($\widehat{f}(x_i) = f(x_i)$ za svako $x_i \in D$). Drugi pristup je da je broj jezgara manji od broja primera (pogotovu ako je broj primera veliki) - tada se, zavisno od domena, mogu birati jezgra uniformno raspoređena u nekoj oblasti prostora instanci ili oko klastera (grozd - cluster) ili njihovih hijerarhija (vezano za metodu nenadgledanog učenja). Prednost FRB je jednostavnija obuka nego kod NM sa povratnim propagiranjem.



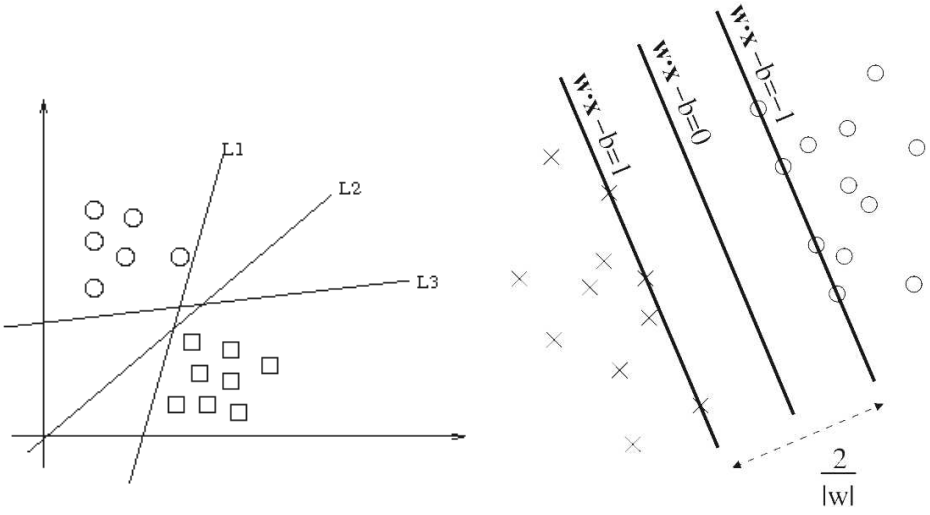
Slika 6: ilustracija preuzeta iz [6]

2.6.4 Zaključivanje izborom slučaja

Zaključivanje izborom slučaja (Case-Based Reasoning) polazi od odloženog zaključivanja i instanci primera bliskih instanci upita (one udaljene zanemaruje) slično prethodnim metodama, ali ne koristi instance koje su uređene n-torke u $\mathbb{R}^n$ već simboličke zapise u smislu deklarativnog znanja. Primer takvog sistema je CADET (Sycara, 1992).

2.6.5 SVM, mašine potpornih vektora

Support Vector Machine (SVM, Vladimir Vapnik, 1963, [8]) poznata je i kao metoda maksimalno granične hiperravni jer se traži hiperravan koja razdvaja hiperprostor primera u dve particije (poluprostora) koje odgovaraju dvema kategorijama (može se uopštiti onda i za veći broj kategorija) tako da je udaljenost te hiperravni od najbližih tačaka podataka maksimalna. Hiperravan je određena vektorom w i jednačinom $w \cdot x - b = 0$. Podaci su *linearno separabilni* ako postoje dve hiperravni paralelne datoj (u različitim podprostorima) između kojih nema tačaka iz skupa primera, tj. $|w \cdot x_i - b| \geq 1$, što se može opisati i sa $c_i(w \cdot x_i - b) \geq 1$. Primeri koji leže na ovim paralelnim hiperravnima se nazivaju vektorima nosačima (support vectors) :



Slika 7a, 7b

Ako se pođe od skupa primera oblika $\{(x_1, c_1), ..., (x_n, c_n)\}$ gde su $c_i \in \{-1, 1\}$ (vrednosti x_i je poželjno normalizovati, recimo $x_i \in [-1, 1]$, da ne bi uticale na klasifikaciju), zadatak se onda svodi na minimizovanje $|w|$, tj. na kvadratnu optimizaciju $\min \frac{1}{2}|w|^2$ uz pomenute uslove sa koeficijentima $c_i(w \cdot x_i - b) \geq 1$ $(i = \overline{1, n})$. Dualni oblik zadatka pokazuje da problem zavisi samo od vektora

nosača - ako je α_i zadato sistemom $w = \sum_i \alpha_i c_i x_i$, dualni oblik je:

$$\max \sum_{i=1}^{n} \alpha_i - \sum_{i,j} \alpha_i \alpha_j c_i c_j x_i^T x_j, \quad \alpha_i \geq 0$$

Ako se dozvoli određena greška klasifikacije ξ onda se dobija *meka granica* (soft margin):

$$c_i(w \cdot x_i - b) \geq 1 - \xi_i, \quad i = \overline{1, n}$$

pa ako je ocena greške linearna, cilj postaje minimizovanje objektivne funkcije $|w|^2 + C \sum_i \xi_i$ uz prethodne uslove i C kao parametar (stepen ,,kažnjavanja" grešaka u klasifikovanju). To se može rešiti upotrebom metode Lagranžovih koeficijenata, što se može pokazati korisnim kod nelinearnih jezgara, i tada se dobija efikasan iterativan algoritam LSVM.

Jedna od veoma efikasnih i jednostavnih implementacija, SMO (Sequential Minimal Optimization), koristi razbijanje na najmanji podproblem gde se onda lako određuje vrednost jednog po jednog preostalog koeficijenta ([9]) umesto skupog numeričkog rešavanja problema kvadratnog programiranja. Pored nje, poznata je i SVMLight implementacija koju koristi program za etiketiranje SVMTool ([www]). SVR (Support Vector Regression) koristi model takve funkcije koji koristi samo deo skupa primera a ostale ignoriše. Ako se skalarni proizvod u jednačinama komponuje sa nekom funkcijom jezgra (polinomijalno homogeno $k(x_1, x_2) = (x_1 \cdot x_2)^k$, polinomijalno nehomogeno $k(x_1, x_2) = (x_1 \cdot x_2 + 1)^k$, Gausovo, sigmoid, itd.) dobijaju se nelinearne varijante metode (Aizerman). SMV je izuzetno efikasna metoda u eksploataciji uz date pretpostavke, i nije metoda učenja instancama u osnovnom obliku - međutim, postoje implementacije (SVMHeavy, [18]) koje podržavaju i ,,lenjo" učenje. Lenjo ili inkrementalno učenje nasuprot radoznalim metodama (eager, ili batch learning) je poželjna osobina učenja ako postoji zahtev za stalnim menjanjem baze znanja (korpusa u slučaju programa za etiketiranje), gde svaka takva promena ne povlači ponavljanje celog postupka učenja već samo efikasno inkrementalno dodavanje znanja.

Obuka kod SVM može biti zahtevnija za veliki broj primera i kategorija, ali je metoda i u učenju u suštini linearno kompleksna $O(nm)$ (m je dimenzija prostora) za razliku od ostalih sličnih poznatih metoda mašinskog učenja koje mahom eksponencijalno zavise od m.

3 Odabrana rešenja i programi

Ovaj tekst predstavlja evaluaciju nekoliko različih vrsta programa za etiketiranje na primeru korpusa i teksta na srpskom jeziku, i zato se u ovom poglavlju opisuje detaljnije i praktična upotreba konkretnih programa koji se testiraju i evaluiraju. Pored pomenutog kriterijuma izbora ovih rešenja kao referentnih i/ili nekomercijalnih, jedan od razloga za izbor je i njihova praktična dostupnost (programa za etiketiranje, besplatnih i upotrebljivih na internetu ima na desetine pored mnogih drugih NLP alata i razvojnih okruženja ([14]), [www] - ali nisu svi referentni predstavnici).

3.1 Tree Tagger

TreeTagger je program koji koristi drvo odlučivanja kao metod učenja, pored uobičajenih statističkih metoda učenja (kako je već pomenuto). Leksikon se sastoji iz 3 dela: punog (fullform) leksikona, leksikona sufiksa i pretpostavljene vrednosti. Ako reč nije nađena u punom leksikonu ni nakon promene u mala slova, pretražuje se leksikon sufiksa drvetom odlučivanja sufiksa (svaka reč). Pored pomenutog drveta odlučivanja za etikete, postoji i posebno drvo sufiksa koje se kreira tokom učenja kao deo leksikona: za svaku reč koja je označena etiketom iz klase otvorenih etiketa, karakter po karakter (do dužine 5 ako nije drugačije zadato) sufiksa. Listovi su označeni vektorima verovatnoće etiketa sa datim sufiksom.

Tree Tagger ([14]) podržava i lematizaciju (za razliku od ostalih programa koji su ovde navedeni).

3.1.1 Etiketiranje

```
tree-tagger {-options-} <parameter file> \
  {<input file> {<output file>}}
```

`<parameter file>` je datoteka koja se dobija u fazi učenja (treniranja) nad zadatim korpusom. Među opcijama koje se tiču načina izdavanja izlazne datoteke izdvajaju se opcije koje utiču i na rad samog programa - opcija `-threshold` sa zadatim pragom iza kazuje da treba dati listu etiketa date reči sa verovatnoćom većom od proizvoda praga i najvećeg praga. Parametar epsilon se može zadati opcijom `-eps` gde je pretpostavljena vrednost 0.1 (za parove reč - etiketa koji su u leksikonu ali ne i u korpusu obučavanja). Leksikon datoteka čija svaka linija predstavlja listu čiji je prvi element reč ili forma, a zatim sledi jedan ili više parova etiketa i odgovarajućih leksema. Na primer:

```
aback       RB aback
abacuses    NNS abacus
abandon     VB abandon      VBP abandon
abandoned   JJ abandoned    VBD abandon     VBN abandon
abandoning  VBG abandon
```

Lista otvorenih klasa je lista mogućih etiketa za reči koje su nepoznate (tu, recimo, treba navesti sve moguće etikete pored onih koje su potpuno definisane leksikonom). Etiketiranje teksta se može vršiti i bez leksema na primer tako što se u leksikonu zadaje ista vrednost lekseme svim rečima kao što je ,,-".

3.1.2 Učenje

```
train-tree-tagger <leksikon> <lista otvorenih klasa> \
  <ulazna datoteka> <izlazna datoteka> [opcije]
```

Ulazna datoteka je niz linija, gde svaku liniju čini par reč i etiketa razdvojenih belim razmakom.

Opcije koje se mogu koristiti tokom učenja su:

- `-st` opcijom se zadaje etiketa kraja rečenice (tačka kao znak interpunkcije) - pretpostavljena vrednost je `SENT`
- `-cl` broj reči koji prethodi formiranju statističkog konteksta - pretpostavljena vrednost 2 odgovara formiranju 3-grama
- `-dtg` prag informacionog dobitka (information gain) ispod koga se briše list drveta odlučivanja (pretpostavljena vrednost je 0.7)
- `-ecw` vrednost težine ocene verovatnoće klasom ekvivalencije (pretpostavljena vrednost 0.15)
- `-atg` prag brisanja lista ,,affix" drveta (pretpostavljena vrednost 1.2)

U nekim distribucijama je na raspoloaganju i dodatni alata tok-english.pl (Perl) za tokenizaciju ulaznog teksta (token po liniji, razdvajanje od znakova interpunkcije i slično).

3.2 TnT - Trigrams'n'Tags

Zaista je primer programa koji je jednostavan za upotrebu, brzo uči i izuzetno brzo etiketira (cca 60K token/sec).

TnT ([15]) je primer statistički baziranog programa za etiketiranje (koristi HMM i Viterbi algoritam). Ne podržava lekseme.

Ovaj sistem koristi nekoliko vrsta datoteka. Neetiketiran oblik datoteke (korpus, obično sa ekstenzijom „.t") je zadat spiskom rečenica gde je svaka reč (token) u jednoj liniji (razdvojena sa LF karakterom - `\\0x0a`, bilo kakav beli razmak i tekst iza njega u liniji se ignoriše), a rečenica se završava tačkom („."). Etiketirani oblik datoteke (etiketirani korpus, obično sa ekstenzijom „.tt")) pored toga u svakoj liniji ima etiketu razdvojenu od reči belim razmakom (TAB `\\0x09` ili SPC `\\0x20`), kao drugu kolonu. Leksikon je datoteka (ekstenzije „.lex") koja se sastoji od minimalno 4 kolone (ili više): prva je reč (token), druga je frekvencija reči u korpusu učenja (broj ponavljanja), treću kolonu čini etiketa, a četvrtu njena frekvencija. Etiketa može biti više za zadatu reč, i iza svake sledi njena frekvencija - zbir frekvencija etiketa jednak je frekvenciji reči. Svaka linija ovih datoteka koja počinje sa `%%` jeste komentar. Postoje posebne klase reči koje se mogu koristiti u leksikonu:

- `@CARD` - niske brojeva
- `@CARDPUNCT` - niske brojeva sa tačkom na kraju
- `@CARDSUFFIX` - niske brojeva sa bilo kojim sufiksom
- `@CARDSEPS` - niske brojeva razdvojene separatorima (tačka, crta, itd)
- `@UNKNOWN` - za nepoznate reči (ako se koristi takav režim obuke)
- `@USECASE` - da li se koriste i velika i mala slova (0 ili 1)
- `@CAPCODE` - ne treba menjati ručno

Pored leksikona, koriste se i n-gram datoteke (ekstenzija „.123") gde za svaki n-gram etiketa (n=1,2,3 kolona za unigrame, bigrame i trigrame) u liniji poslednja kolona predstavlja njegovu frekvenciju. Na kraju, posebne datoteke preslikavanjafs etiketa se koriste da bi se zadali nazivi etiketa u izlaznoj datoteci (ako je potrebno) drugačije nego u samom jezičkom modelu, kao i sam spisak raspoloživih etiketa (skup etiketa, tagset). Format datoteke (ekstenzija „.map"): po dve kolone u liniji, prva je etiketa u jezičkom modelu, druga je naziv koji ide u izlaznu datoteku. Tako se može i smanjiti broj etiketa bez promene samog jezičko modela.

3.2.1 TnT učenje

Obukom se generišu parametri modela na osnovu zadatog etiketiranog modela. To se vrši komandom:

```
tnt-para [opcije] korpus.tt
```

Opcijama se može uticati na proces obučavanja koji po pretpostavljenim vrednostima bez opcija kao rezultat kreira .lex i .123 datoteke istog imena kao i korpus (koji može biti i datoteka spakovana gzip/bzip2) sa n-gramima do n=3.

3.2.2 TnT etiketiranje

Eksploatacija se vrši komandom:

```
tnt [opcije] model korpus.tt > korpus.tts
```

Model je zadat `model.lex` i `model.123` datotekama obavezno, a opciono i model.map datotekom. Za rad programa je neophodno postaviti putanju do datoteka modela u promenljivu `TNT_MODELS` (kao i putanju do izvršnih datoteka u PATH promenljivu). Opcije koje su na raspolaganju su:

- `-a<length>` - maksimalna dužina sufiksa kod nepoznatih reči (pretpostavljena vrednost je 10)
- `-b<file>` - pomoćni leksikon koji se koristi ako u glavnom ne bude pronađena reč pre prelaska na pretrživač nepoznatih reči
- `-d<mode>` - režim retkih podataka (pretpostavljena vrednost `<mode>`=4), gde `<mode>` može biti:
  - `1/c` - zameniti nula frekvencije konstantom `c` - ako se ne navede `/c`, podrazumeva se 0.5
  - `2/c` - dodati svim frekvencijama konstantu `c` - ako se ne navede `/c`, podrazumeva se 0.5
  - `4/`λ_1`/`λ_2 - linearnom interpolacijom se određuje vrednost - λ_1 za unigrame, λ_2 za bigrame, i $1-\lambda_1-\lambda_2$ za trigrame; ako se ne navedu vrednosti, podrazumevaju se automatski dobijene vrednosti interpolacijom brisanja
- `-h` - spisak opcija
- `-H` - HTML etikete u ulaznoj datoteci se ne etiketiraju (već se prepisuju direktno u izlaznu datoteku)
- `-m` - nepoznate reči dobijaju „*” kao etiketu
- `-n<N>` - koriste se n-grami do N=1,2,3 za etiketiranje (3 je pretpostavljena vrednost)

- `-P` - ne koriste se verovatnoće u izlazu - koristi se u `-z` opciju:
- `-z<v>` - ispisuje se alternativna etiketa ako je njena verovatnoća p veća od M/v, gde je M verovatnoća najbolje etikete (inače određuje i vrednost i za `-Z` kao $1000/v$)
- `-Z<p>` - za razliku od prethodne opcije, ovom se ograničava stablo pretrage za najboljom etiketom (pretpostavljena vrednost je 1000)
- `-u<U>` - način rešavanja nepoznatih reči - pretpostavljeno je `U=3`, kombinovanje statistika svih singlotna (reči frekvencije 1)
- `-v<V>` - opširnost (verbosity) izlaza - pretpostavljeno je `V=3`, maksimalna opširnost

3.2.3 Pomoćni alati

Alat kojim se mogu upoređivati ispravni korpusi sa statistički generisanim (prethodnom komandom):

```
tnt-diff [opcije] korpus.tt korpus.tts
```

Pored ovog alata, na raspologanju je i `tnt-wc` za prebrojavanje reči (tokena), tipova i etiketa datih korpusa.

3.3 Brill - Rule Based Tagger

Eric Brill je jedan od pionira učenja transformacijama (Transformation-based Learning) i programa za etiketiranje RBT (Rule Based Tagger, [16]) zasnovanim na tome koji je svojevremeno postavio mnoge standarde. Jednim delom je realizovan Perl skriptovima (Bril to smatra idealnim programskim jezikom za ovakva lingvistička istraživanja zbog bogatstva rada sa niskama i regularnim izrazima, ali to je i jedan od glavnih uzroka sporosti učenja). Ovaj sistem predstavlja jedan od osnovnih primera programa za etiketiranje koji nisu zasnovani na verovatnosnim modelima već pre svega na inkrementalnom učenju pravila kontekstnih transformacija (pravila prezapisivanja kojima se daje rezultat sa najvećom ocenom tačnosti). Algoritam: u suštini, etiketiranje se vrši u dve faze: najpre se inicira početno stanje korpusa koji se etiketira prema leksikonu i leksičkim pravilima modela (da bi se premostila ograničenja HMM gde se koriste samo n-grami nad leksemama, uvode se pravila koja se referišu i na etikete), a onda se u drugoj fazi povećava tačnost etiketiranja primenom kontekstnih pravila modela (slična leksičkim, ali sadrže i drugu etiketu u koju se menja prva - sva pravila u suštini menjaju etiketu ako su

ispunjeni neki uslovi). Učenje: faza primene leksikona, ugrađenih i leksičkih pravila je faza incijalizacije etiketa, nakon čega sledi faza učenja vođenog ocenom greške (iterativno se ocenjuje greška za svako pravilo kandidata pre i posle transformacije, i onda se najbolje ocenjeno dodaje u skup pravila sve dok ne bude ni jednog primenjivog pravila sa ocenom greške iznad zadatog praga). Ima ,,ugrađeno" leksičko pravilo za reči koje nisu u leksikonu a koje počinju velikim slovom - smatraju se vlastitim imenima (inače obična imenica), nakon čega se primenjuje potprogram za nepoznate reči (ukratko, sufiksi i prefiksi dužine do 4 karaktera se uzimaju u obzir i traže podreči u leksikonu i uče se nova pravila ako su nađene).

3.3.1 Etiketiranje

Komanda kojom se vrši etiketiranje je:

```
tagger LEXICON CORPUS BIGRAMS LEXICALRULEFULE CONTEXTUALRULEFILE
```

- `LEXICON` je datoteka koja sadrži leksikon (po instalaciji to je simbolička veza ka LEXICON.BROWN.AND.WSJ, leksikonu izvedenom od standardnog WSJ Treebank korpusa)
- `CORPUS` - ulazni tekst
- `BIGRAMS` - lista uzastopnih parova nađenih u korpusu učenja
- `LEXICALRULEFILE` - lista transformacija za početno etiketiranje reči koje ne nalaze u leksikonu
- `CONTEXTUALRULEFILE` - lista kontekstno osetljivih transformacija

Jedno od ograničenja ovog programa je da zahteva da se izvršava u direktorijumu `Bin_and_Data` gde se nalaze izvršni i pomoćni programi, ali i datoteke modela.

3.3.2 Učenje

Tokenizacijom (prema Penn Tree Bank standardu) neetiketirani korpus sa rečenicama oblika:

```
John (I think) said: "Who are you?"  He then gave me $10.
```

postaje:

```
John ( I think ) said : " Who are you ? "
He then gave me $ 10 .
```

Etiketirani deo korpusa (`TAGGED-CORPUS-ENTIRE` u daljem tekstu, i u `README` datoteci uz ove programe) tokom učenja se deli na dve particije:

- prvu (`TAGGED-CORPUS`, slično prethodno pomenutoj datoteci), za učenje leksikona i leksičkih pravila za reči izvan leksikona, i
- drugu particiju, za učenje kontekstnih pravila (objašnjeno dalje u tekstu)

Dakle, preduslov za učenje je da postoji manji deo etiketiranog korpusa koji je ,,horizontalnog" oblika:

```
The/det cat/noun sat/verb on/prep the/det mat/noun ./.
```

... od koga se sledećom komandom može kreirati neetiketirani korpus `UNTAGGED-CORPUS` (naziv takve datoteke dat je velikim slovima - na odgovarajući način će i ostalim vrstama datoteka biti dato ime dalje u tekstu):

```
cat TAGGED-CORPUS | Utilities/tagged-to-untagged.prl \
  > UNTAGGED-CORPUS
```

... onda je moguće vršiti proces delimično nenadgledanog učenja sledećim postupkom inkrementalno, uvećavajući tako obim etiketiranog korpusa (iako je u suštini ovo učenje ipak nadgledano početnim etiketiranim korpusom), čiji su koraci:

(1) najpre se kreira spisak svih reči korpusa:

```
cat UNTAGGED-CORPUS | Utilities/wordlist-make.prl | \
  sort +1 -rn | awk '{ print $1}' > BIGWORDLIST
```

(2) spisak linija tj. lista reč-etiketa-broj:

```
cat TAGGED-CORPUS | Utilities/word-tag-count.prl \
      | sort +2 -rn > SMALLWORDTAGLIST
```

(3) svi bigrami UNTAGGED-CORPUS-a:

```
cat UNTAGGED-CORPUS | Utilities/bigram-generate.prl | \
      awk '{ print $1,$2}' > BIGBIGRAMLIST
```

opciono, mogu se izbaciti bigrami koji se pojavljuju malom frekvencijom, jednom ili dvaput u ovom primeru:

```
awk '{ if ($3>2) print $1,$2}'
```

(4) leksičko učenje:

```
unknown-lexical-learn.prl BIGWORDLIST SMALLWORDTAGLIST \
    BIGBIGRAMLIST 300 LEXRULEOUTFILE
```

gde je `LEXRULEOUTFILE` rezultat koji predstavlja datoteku sa spiskom pravila etiketiranja nepoznatih reči (izvan leksikona) sa najvećom verovatnoćom. Na primer:

```
NN s fhassuf 1 NNS 3022.2549507603848724
```

... znači: promeniti etiketu iz NN u NNS ako reč ima sufiks ,,s".

Broj 300 govori da se koriste samo bigrami koji se javljaju među 300 najfrekventnijih reči u korpusu (do ranga 300). Moguće je u samom Perl kôdu promeniti prag `$THRESHOLD = 3;` ocene naučenog pravila (ako učenje traje predugo treba povećati vrednost; recimo do 50K reči može koristiti pretpostavljen prag 3, a za preko 400K reči se može koristiti i 8). Postoje neka pravila koja su ,,ugrađena" u proces učenja i etiketiranja nepoznatih reči (etiketa NNP za reči koje počinju velikim slovom, NN inače), što se može promeniti u unknown-lexical-learn.prl ako uopšte predstavlja problem. Učenje pravila se može vršiti inkrementalno, konkateniranjem postojećeg `OLDLEXRULEFILE` i novonaučenog `LEXRULEOUTFILE` dobijenog komandom:

```
unknown-lexical-learn-continue.prl BIGWORDLIST SMALLWORDTAGLIST \
      BIGBIGRAMLIST 300 LEXRULEOUTFILE OLDLEXRULEFILE
```

3.3.3 Učenje kontekstualnih pravila

Sledećom etapom učenja se uče pravila kojima se povećava tačnost na osnovu kontekstualnih redova. Neka je TAGGED-CORPUS ono što je dosad naučeno, TAGGED-CORPUS-2 ostatak (još neiskorišćen), a TAGGED-CORPUS-ENTIRE kompletno ručno etiketiran korpus, leksikon je onda oblika (horizontalnog):

```
 REČ ETIKETA-1 ETIKETA-2 ... ETIKETA-N
```

(`ETIKETA-1` je najverovatnije, ostale su opcije bez posebnog redosleda).

(5) Sledećom komandom za naučeni ili ručno etiketirani korpus kreira se leksikon (bez `TAGGED-CORPUS-2` koji će se koristiti za dalje učenje kontekstualnih pravila, gde se obično koristi odnos 1:2 po broju reči u odnosu na `TAGGED-CORPUS` umesto 1:1, što se može postići i menjanjem praga u pomoćnom programu `Utilities/divide-in-two-rand.prl`, linija "`...if ($x > .5) {...` "):

```
cat TAGGED-CORPUS | Utilities/make-restricted-lexicon.prl \
  > TRAINING.LEXICON
```

(6) Nakon toga se pokrene:

```
cat TAGGED-CORPUS-ENTIRE | Utilities/make-restricted-lexicon.prl \
  > FINAL.LEXICON
```

(7) i potrebno je još i:

```
cat TAGGED-CORPUS-2 | Utilities/tagged-to-untagged.prl \
  > UNTAGGED-CORPUS-2
```

(8) i može se napraviti probna evaluacija:

```
tagger TRAINING.LEXICON UNTAGGED-CORPUS-2 BIGBIGRAMLIST \
 LEXRULEOUTFILE /dev/null -w BIGWORDLIST  \
 -i DUMMY-TAGGED-CORPUS > /dev/null
```

(9) Učenje kontekstualnih pravila u datoteci `CONTEXT-RULEFILE` se onda radi komandom:

```
contextual-rule-learn TAGGED-CORPUS-2 DUMMY-TAGGED-CORPUS \
      CONTEXT-RULEFILE TRAINING.LEXICON
```

3.3.4 Ostali korisni alati

Na primer, pored pomenutih skriptova za razdvajanje korpusa, za potrebe inkrementalnog učenja se koristi i:

```
incorporate-new-bigrams.prl LEXICALRULEFILE BIGRAMS \
  NEWCORPUS > NEWBIGRAMS
```

3.4 MXPOST

Ovaj program ([20]) je bliži Brilovom i SVM po ,,brzini" učenja (koje u njihovom slučaju ipak za red veličine traje duže). Program radi 100 iteracija bez obzira na performanse tokom učenja. Datoteka primera za učenje ima sličnu horizontalnu rečeničnu strukturu kao i kod Brilovog programa za etiketiranje:

```
R-1_TAG-1 R-2_TAG-2 ... R-N_TAG-N
```

Program je realizovan u javi tako da je neophodno podesiti `CLASSPATH=mxpost.jar`. Ulazna datoteka se sastoji od jedne reči po liniji, a komanda za eksploataciju je:

```
mxpost projectdir < wordfile
```

a za učenje se koristi:

```
trainmxpost projectdir traindata
```

3.5 SVMTool

SVMTool ([17]) jedan je od programa kojima je dostignut ,,state-of-the-art" rezultat na WSJ korpusu, i svakako program sa najbogatijim opsegom opcija za učenje i uticanje na rad samog programa (podržava, pored ostalog, i dvosmerno učenje i etiketiranje, i unakrsnu validaciju rezultata, posebne jezičke izuzetke i popravke rečnika, različite modele učenja). Ovo navodi i ujedno i na glavni nedostatak - prilično neefikasno učenje (pitanje je da li se zbog nekog procenta isplati za najmanje jedan red veličine duže raditi obučavanje modela). Naprednija opcija podrazumeva otkrivanje SVM parametara validacijom nad skupom obučavanja, ali sa tom i drugim opcijama treba pažljivo i strpljivo raditi da bi se došlo do značajnih rezultata.

Ukratko, ideja je da se sekvencijalno kodiraju različite osobine teksta u prozoru date širine (merene leksemama, centrirano oko lekseme koja se posmatra - pretpostavljene širine 7), kako osobine reči (da li sadrže veliko slovo, specijalne znake, brojeve i slično), tako i n-grame, etikete i drugo. Vektor takvih osobina se klasifikuje SVM algoritmom, gde svakoj klasi odgovara jedna etiketa.

3.5.1 Učenje

Nakon instalacije programa (neophodno je instalirati i SMVlight) prema uputstvima (nije bilo posebnih poteškoća u odnosu na ostale, jedino je TnT uporediv u smislu jednostavnosti korišćenja), potrebno je podesiti konfiguracione parametre u datoteci `bin/ config.short.svmt` (u dodatku), pošto je ,,duža" varijanta `config.svmt` davala ubedljivo najduže vreme učenja (više od dva sata za 2.5K korpus nije bilo dovoljno da konvergira, a inače učenje sa kraćom konfiguracijom traje tek nešto kraće nego kod Brilovog programa koji je šampion sporosti uvčenja). U toj konfiguracionoj datoteci se pored ostalog zadaje obavezno korpus za učenje, ime modela i putanja do SVMlight.

Nakon toga je dovoljno pokrenuti:

```
SVMTlearn [opcije] bin/config.short.svmt
```

Program obiluje mnoštvom opcija i mogućnosti kako učenja, tako i etiketiranja kasnije.

3.5.2 Etiketiranje

Da bi se etiketiranje izvršilo, treba pokrenuti:

```
SVMTagger [opcije] model
```

Program nudi i pomoćni alata za evaluaciju rezultata - još tokom učenja se može primetiti da se posebno prate reči koje su prepoznate (u leksikonu), koje su nejasne (ambiguous), koje su nepoznate i ukupne statistike posebno.

4 Evaluacija odabranih programa

Sam postupak evaluacije odabranih rešenja je rađen automatizovano bash skriptom `data.sh` (videti dodatak). Struktura izvršavanja ovog skripta je:

1. transformacija ulaznih datoteka korpusa u vertikalni format (jedna leksema i opciono druge kolone - jedna linija) - rezultat je datoteka `out1.txt` (ceo korpus)

2. inicijalizacija stat_TAG.txt datoteka sa statističkim brojačima

3. glavna petlja (`n=`$\overline{0,9}$):

   - pripreme se pomoćnim skriptom `cross-tab10.sh` korpus za učenje (`learn.txt`) i korpus za testiranje (`test.txt`) kao `n`-te ,,9/10+1/10" particije celog korpusa
   - za svaki program TAG za etiketiranje se odgovarajućim AWK skriptovima pripreme potrebni transformisani korpusi, pokrene učenje i zatim etiketiranje neetiketiranog test korpusa
   - rezultat etiketiranja TAG programa se u vertikalnom formatu smešta u datoteku `test_`TAG`.txt`, a ranije pripremljen etiketirani test korpus u `test_`TAG`.txt` - vertikalni format omogućava lako prebrojavanje različito etiketiranih leksema (linija)
   - za svaki program TAG se pomoćnim skriptovima `unknow.sh` i `unknown.awk` dodaju brojači o nepoznatim rečima (one koje su loše etiketirane se traže u leksikonu učenja - ako nisu tamo, nepoznate su)

4. pomoćnim skriptovima `report1.awk` i `report2.awk` se prave agregirane statistike

5. brisanje (clean-up) datoteka (nastalih učenjem, kao i pomoćnih datoteka)

Ovakvo automatizovanje postupka testiranja i evaluacije programa za etiketiranje daje mogućnost lakog proširivanja drugim programima za etiketiranje koje treba evaluirati i upoređivanje sa ranijim, smanjuje mogućnost grešaka u računanju statističkih osobina, i pre svega predstavlja okruženje za isprobavanje ovih programa. U dodatku su date i neke napomene u vezi podešavanja ovakvog okruženja i instalacije ovih programa.

4.1 Korpusi

Osnovna osobina korpusa je njegova dužina, odnosno broj etiketiranih reči koje čine tekst korpusa. Međutim, skup reči koje čine leksikon sačinjen nad korpusom je manji ali bitniji. Takođe, veoma važna osobina korpusa je i broj elemenata klase etiketa nad kojom se korpus proučava. Klasa etiketa može biti složena i velike kardinalnosti, što veoma utiče na performanse programa za etiketiranje - a to zavisi kako od koncepta etiketiranja, tako i od samog jezika. Primera radi, srpski jezik je svakako zahtevniji u tom smislu u odnosu na engleski jezik ako se etiketiraju i padeži. Pored toga, različiti programi za etiketiranje su (kao i kod svakog zadatka nadgledanog mašinskog učenja) različito osetljivi na kvalitet informacija u samom korpusu. Manji ali izvestan broj grešaka može bitno uticati na performanse programa (na primer, znak interpunkcije nadostaje pa program dodeli etiketu kraja rečenice poslednjoj rečenici, više različitih etiketa dodeljeno istoj leksemi, ili jednostavno pogrešno uneta etiketa). Neke manje greške moguće je uočiti i ručno ispraviti ako tokom provere sam skript javi grešku, neke je moguće predvideti i ispraviti jednostavnim leksičkim obradama - ali mnoge ostaju ,,skrivene". U tom pogledu je poslednji korpus kvalitetniji, ali iz nekih razloga daje slabije rezultate - verovatna posledica većeg broja etiketa, kao i nedovoljne podešenosti programa specifičnostima srpskog jezika. Svi programi su trenirani sa pretpostavljenim (default) vrednostima parametara učenja, bez ikakvih izmena (i u tom smislu ravnopravni). Naravno, kod nekih programa je moguće postići i bolje rezultate, ali uz dodatni trud i specifična prilagođavanja.

U okviru ove evaluacije korišćena su dve vrste datoteka za izgradnju polaznih korpusa. Jedna vrsta je struktuirana u etiketiranom XML formatu, gde je, ukratko:

- svaka reč data XML etiketom `mw` sa atributom
  - `id` koji je jedinstveno identifikuje,
  - atributom `lex` koji daje leksemu date reči,
  - atributom `lemma` koji daje lemu date reči,
  - i atributom `tag` koji daje etiketu date reči.
- svaka rečenica je data XML etiketom `seg` i atributom `id` koji je jedinstveno identifikuje
- svaka strana je data XML etiketom `p`, a odeljak etiketom `div`

Primer ovakve datoteke i njene strukture dat je ispod:

```
<Annotation type="morpho">
  <body>
    <div>
      <head>
        <mw id="mw__1 " lex="ZAKLJUCAK" lemma="ZAKLJUCAK" tag="?"/>
      </head>
      <p>
      <seg id="n1">
        <mw id="mw_1_1 " lex="Na" lemma="na" tag="PREP+p4"/>
        <mw id="mw_1_2 " lex="meunarodnom" lemma="meunarodni" tag="A"/>
        <mw id="mw_1_3 " lex="planu" lemma="plan" tag="N"/>
        <mw id="mw_1_4 " lex="poslednjih" lemma="poslednji" tag="A"/>
        <mw id="mw_1_5 " lex="decenija" lemma="decenija" tag="N"/>
        <mw id="mw_1_6 " lex="preduzeti" lemma="preduzeti" tag="V+Perf+Tr"/>
        <mw id="mw_1_7 " lex="su" lemma="jesam" tag="V+Imperf+It+Iref"/>
        <mw id="mw_1_8 " lex="znacajni" lemma="znacajan" tag="A"/>
...
      </seg>
      <seg id="n2">
```

Ovakvu datoteku sam XSLT transformacijama zadatim ispod pretvarao u tekstuelni oblik potreban za dalji rad (`data.xsl`):

```
<?xml version="1.0" encoding="UTF8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" omit-xml-declaration="yes" indent="no"/>

  <xsl:template match="//seg">
    <xsl:for-each select="mw">
     <xsl:value-of select="@lex"/>~
     <xsl:value-of select="@tag"/>~
     <xsl:value-of select="@lemma"/>~
    </xsl:for-each>*SENT*
  </xsl:template>

</xsl:stylesheet>
```

Drugi tip datoteka koričćenih za evaluaciju dat je po TEI normi ([23]), dosta slično prethodnim datotekama, ali sa strukturom koja je donekle drugačija (XML etikete i atributi nisu isti, a npr. lekseme nisu date kao atributi već vrednosti XML etikete `w`) i mnogo bogatija. TEI standard ([23]) podrazumeva i posebno zaglavlje sa bibliografskim podacima, podacima o kodnom rasporedu,

strukturi datoteke i mnogim drugim meta-podacima, ali i mnoge druge pomoćne strukture (npr. za opis semantičke strukture etiketa). Primer ovakve datoteke dat je ispod:

```
<TEI.2 id="Osr" lang="sr">
  <teiHeader creator="CK" status="update" ... id="Osr.teiHeader">
    <fileDesc>
        <titleStmt>
...
    </fileDesc>
    <encodingDesc>
        <projectDesc>
...
    </encodingDesc>
    <revisionDesc>
...
    </revisionDesc>
</teiHeader>
<text lang="sr" id="Osr.">
  <body>
    <div id="Osr.1" type="part" n="1">
    <div id="Osr.1.2" type="chapter" n="1">
      <p id="Osr.1.2.2">
        <s id="Osr.1.2.2.1">
          <w lemma="biti" ana="Vmps-sman-n---p">Bio</w>
          <w lemma="jesam" ana="Va-p3s-an-y---p">je</w>
          <w lemma="vedar" ana="Afpmsnn">vedar</w>
          <w lemma="i" ana="C-s">i</w>
          <w lemma="hladan" ana="Afpmsnn">hladan</w>
          <w lemma="aprilski" ana="Aopmpn">aprilski</w>
          <w lemma="dan" ana="Ncmsn--n">dan</w>
          <c>;</c>
          <w lemma="na" ana="Spsa">na</w>
          <w lemma="&#x10D;asovnik" ana="Ncmsa--n">&#x10D;asovnicima</w>
          <w lemma="jesam" ana="Va-p3s-an-y---p">je</w>
          <w lemma="izbijati" ana="Vmps-snan-n---e">izbijalo</w>
          <w lemma="trinaest" ana="Mc---l">trinaest</w>
          <c>.</c>
        </s>
        <s id="Osr.1.2.2.2">
          <w lemma="Vinston" ana="Npmsn--y">Vinston</w>
          <w lemma="Smit" ana="Npmsn--y">Smit</w>
          <c>,</c>
...
```

```
<!-- pb n=283 -->
</p>
</div>
</body>
</text>
</TEI.2>
```

Odgovarajuća XSLT transformacija (sa izlazom istim kao i za prethodnu vrstu datoteka) je data ispod (`data2.xsl`):

```
<?xml version="1.0" encoding="UTF8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" omit-xml-declaration="yes" indent="no"/>

 <xsl:template match="//s">
   <xsl:for-each select="w">
     <xsl:value-of select="."/>~
     <xsl:value-of select="@ana"/>~
     <xsl:value-of select="@lemma"/>~
   </xsl:for-each>*SENT*
 </xsl:template>

</xsl:stylesheet>
```

... i pored toga, da bi obezbedio isti izlazni format, koristio sam i AWK kôd (jer se znakovi interpunkcije zapisuju u posebnim XML etiketama `c` u okviru rečenice) - `punct.awk`:

```
BEGIN { FOUND=0; OFS=""; }
# vi\v{s}e linija: /<c>/,/<\/c>/
# { gsub(/<c>/,""); gsub(/<\/c>/,""); print $0; }
/<c>(.)*<\/c>/ {
   line = gensub(/<c(.)*>((.)*)<\/c>/,"\\2","g");
   line = gensub(/^([[:space:]]|\n|\r|\t)*/,"","g",line);
   line = gensub(/([[:space:]]|\n|\r|\t)*$/,"","g",line);
   print "<w lemma=\"" line "\" ana=\"PUNCT\">" line "</w>";
   FOUND = 1;
}
{ if (!FOUND) { print $0;} FOUND=0; }
```

Nakon ovih transformacija, sledi dalja obrada korpusa u cilju dobijanja vertikalnog formata (tab-delimited datoteka sa kolonama: LEKSEMA, ETIKETA, LEMA

- uz specijalnu etiketu `SENT` koja označava kraj rečenice, što inače koristi i program Tree Tagger) - `data.awk`:

```
BEGIN {
  FS = "~\t";
  OFS = "\t";
  PREV1 = PREV2 = PREV3 = "";
}
{
 if (($0 ~ "~")||($0 ~ "*SENT*")) {
   sub(/^(\x20|\t)+/,"",$0);
   gsub(/(\x20)+/,"",$0);
   if (($1!="")&&(PREV1!="")) {
     if ($1=="*SENT*") {
       print PREV1, "SENT", PREV3;
     }
     else if (PREV1!="*SENT*") {
       if (PREV2=="") PREV2="?";
       if (PREV3=="") PREV3="?";
       print PREV1,PREV2,PREV3;
     }
   }
   PREV1=$1; PREV2=$2; PREV3=$3;
 }
}
```

Za evaluaciju su korišćena tri korpusa koje ću u tekstu dalje pominjati kao korpus 1, korpus 2 i korpus 3. Semantička struktura klasa etiketa (često se koristi i termin MSD, morphosyntactic descriptions) koje se koriste u korpusima 1 i 2 nisu iste kao i korpusu 3, pored same strukture i formata datoteka kojima su zapisani, kako je već pomenuto. Korpusi 1 i 2 se razlikuju samo po obimu, gde je korpus 2 nastao kao spoj datoteke koje čini korpus 1 i još četiri datoteke (videti glavni skript `data.sh` u dodatku u kojem su navedena imena datoteka). Korpus 3 je nastao od teksta Orvelovog romana „1984.”([13]) u okviru evropskog MULTEXT-East projekta (koji je ujedno i primer paralelnog višejezičnog korpusa). Po TEI standardu ([23]) MSD su zadate bibliotekama struktura osobina (features). Primera radi, u okviru pomenutog MULTEXT-East projekta ([1]) imenica je opisana sa 11 pozicija (N, Noun):

```
**** **** **** **** **** ---- ---- ---- ---- ---- ----
PoS  Type Gend Numb Case Def  Cltc Anim OwnN OwnP OwdN
**** **** **** **** **** ---- ---- ---- ---- ---- ----

= ============== ============== =  EN  RO  SL  CS  BG  ET  HU  HR
P ATT            VAL            C  x   x   x   x   x   x   x   x
= ============== ============== =
```

1 Type		common	c	x	x	x	x	x	x	x	x
		proper	p	x	x	x	x	x	x	x	x
-	---	---	-								
2 Gender		masculine	m	x	x	x	x	x			x
		feminine	f	x	x	x	x	x			x
		neuter	n	x	x	x	x	x			x
-	---	---	-								
3 Number		singular	s	x	x	x	x	x	x	x	x
		plural	p	x	x	x	x	x	x	x	x
		dual	d			x	x				
	l.s.	count	t					x			
-	---	---	-								
4 Case		nominative	n			x	x	x	x	x	x
		genitive	g			x	x		x	x	x
		dative	d			x	x			x	x
		accusative	a			x	x			x	x
		vocative	v		x		x	x			x
		locative	l			x	x				x
		instrumental	i			x	x			x	x
	l.s.	direct	r		x						
	l.s.	oblique	o		x						
	l.s.	partitive	1						x		
		illative	x						x	x	

...

U okviru tog projekta razvijen je i poseban program za etiketiranje ToTaLe, a koristi se uporedo i TNT i MBT. Postoji i alat (izvorni C program na [1]) kojim se mogu dobiti detaljni opisi svake etikete čiji izlaz na primer izgleda ovako:

```
Ncfsg: Noun common feminine singular genitive
```

4.2 Rezultati

Rezultati dobijeni (prethodno opisanim postupkom i korpusima) za sva tri korpusa predstavljeni su sledećom tabelom:

Korpus:	Korpus 1	Korpus 2	Korpus 3
Dužina:	7.5K	75K	105K
Broj leksema:	2.5K	11K	18K
Broj lema:	1.6K	5K	7.6K
Broj etiketa:	79	129	908
trajanje testa:	22min.	9č : 50min.	5 dana, 1č, 29min.
min/max/avg *	2290/2335/2378	9766/10368/10952	16550/16919/17372
min/max/avg **	73 / 77 / 79	120 / 126 / 129	840/884/897
TT - min/nep.:	77.64% / 58.57%	92.13% / 0.00%	76.43% / 31.58%
TT - avg/nep.:	85.44% / 64.93%	**94.39**% / **33.30**%	79.65% / 35.05%
TT - max/nep.:	89.54% / 71.43%	97.39% / 52.47%	82.68% / 37.51%
TT - dev/nep.:	3.90% / 3.87%	1.86% / 20.25%	1.92% / 1.85%
SVM - min/nep.:	75.08% / 56.20%	92.41% / 0.00%	82.35% / 31.55%
SVM - avg/nep.:	84.93% / 64.70%	94.27% / 38.02%	85.24% / 34.67%
SVM - max/nep.:	89.48% / 75.00%	97.41% / 61.40%	88.77% / 37.87%
SVM - dev/nep.:	3.60% / 5.51%	1.72% / 22.61%	1.87% / 2.27%
TNT - min/nep.:	77.96% / 60.14%	91.82% / 0.00%	82.50% / 28.86%
TNT - avg/nep.:	**86.18**% / 67.65%	94.11% / 37.42%	**85.47**% / 32.26%
TNT - max/nep.:	90.64% / 74.07%	96.89% / 59.71%	88.07% / 35.49%
TNT - dev/nep.:	3.60% / 4.33%	1.65% / 21.85%	1.75% / 2.19%
MX - min/nep.:	73.00% / 49.61%	90.21% / 0.00%	79.50% / 24.18%
MX - avg/nep.:	82.69% / **54.01**%	92.78% / **29.43**%	82.07% / **28.62**%
MX - max/nep.:	86.05% / 57.97%	95.78% / 45.69%	84.78% / 31.34%
MX - dev/nep.:	3.84% / 2.49%	1.79% / 16.93%	1.69% / 2.25%
RBT - min/nep.:	73.16% / 72.02%	89.65% / 0.00%	82.15% / 34.65%
RBT - avg/nep.:	84.96% / 82.15%	93.14% / 47.24%	85.20% / 37.96%
RBT - max/nep.:	88.75% / 89.71%	98.15% / 70.11%	88.55% / 40.73%
RBT - dev/nep.:	4.34% / 4.32%	3.21% / 26.29%	1.95% / 1.97%

(Tabela 2)

* - broj leksema u korpusu za obučavanje (ostatak u korpusu za testiranje)
** - broj etiketa u korpusu za obučavanje (-- || --)

Korpusi 1 i 2 se neznatno razlikuju po broju leksema, ali zato se značajno razlikuju po broju etiketa. Udeo nepoznatih reči među onima koje nisu ispravno etiketirane koji je ovde prikazan („nep." je skraćeno od „nepoznatih", u prvoj koloni tabele) dat je kao ilustracija odnosa među programima i korpusima. Za nepoznate reči je važniji udeo loše etiketiranih nepoznatih reči u odnosu na sve nepoznate reči u test korpusu (kao i odnos prema udelu ispravno etiketiranih reči u odnou na ostatak korpusa). Prosečan broj nepoznatih reči za korpuse 1, 2 i 3 redom je 164, 584, 1196 (razlika broja leksema celog korpusa i prosečnog korpusa obučavanja), i uz prosečan broj ispravno i pogrešno etiktetiranih reči po metodi dobija se tabela:

Program	Korpus 1: poz./nep.	Korpus 2: poz./nep.	Korpus 3: poz./nep.
TT	98.37% / 56.71%	97.53% / **71.49**%	91.78% / 36.79%
SVM	98.29% / 55.18%	97.69% / 67.17%	93.98% / 54.60%
TNT	**98.54**% / **57.50**%	97.57% / 67.17%	93.86% / **58.36**%
MX	97.43% / 57.01%	96.48% / 69.09%	92.06% / 54.26%
RBT	99.10% / 43.96%	**97.97**% / 48.17%	**94.24**% / 50.33%

(Tabela 3)

Ovakva tabela malo drugačije govori o rezultatima etiketiranja nad nepoznatim rečima u odnosu na prethodnu, ali zato ima manje smisla u opštoj proceni performansi programa. Pošto razdvajanje celog korpusa na particije za učenje i testiranje ide redom i prema broju rečenica (moglo bi se to poboljšati nasumičnim biranjem rečenica kao što to radi `divide-in-two-rand.prl`, ili još bolje - korpusima veće dužine, jer uniformnija nasumična raspodela na particije predstavlja ujedno i manje realan test), neke od particija za testiranje su u drugom korpusu praktično ostale bez nepoznatih reči (odatle velika standardna devijacija za nepoznate reči i minimum 0). Torsten na [15] pokazuje rezultate sa standardnom devijacijom 0.13 za Penn Treebank (0.76 za Susanne Corpus na engleskom, 0.29 za NEGRA korpus na nemačkom), što pokazuje da i standardna devijacija može da se uporedi i opravda sa ovde dobijenim rezultatima. Naravno, ovakvo poređenje sa statističkim rezultatima drugih nije pravi način dokazivanja ispravnosti zaključaka u ovom tekstu, ali među navedenim radovima u ovoj oblasti postoje detaljniji dokazi o osobinama programa za etiketiranje i njihovim performansama (a neke osobine proističu i iz same prirode programa za etiketiranje kao programa mašinskog učenja).

Ovde nisu posebno razmatrane brzine etiketiranja, osim procene trajanja učenja i cele evaluacije koja je najvećim delom trajala koliko i učenje Brilovog i SVMTool programa, a tek manjim delom MXPOST, i ostalih čije učenje traje zanemarivo manje. TnT je svakako šampion i brzine etiketiranja i učenja, a i njegove performanse u smislu tačnosti jesu za pohvalu, kao i jednostavnost upotrebe.

5 Zaključak

Prema [1] dobijeni su sledeći rezultati za dva programa za etiketiranje:

	TnT	MBT
Poznate	93.55%	93.58%
Nepoznate	60.77%	44.45%

(Tabela 4)

Ovi rezultati su uporedivi sa ovde dobijenim rezultatima za TnT program kako za nepoznate reči (dobijen je možda i nešto bolji rezultat), tako i za poznate reči. Slični rezultati upravo na korpusu 3, Orvelovoj "1984.", imaju još više smisla kao prilog poredjenju programa na ovaj način. Iako je veći broj etiketa narušio performanse korpusa 3 u odnosu na korpus 2 gde su dostignuti maksimalni (praktično vrunski rezultati), ovaj test je sigurno realniji i upotrebljiviji. Sudeći prema tabeli 2, program TnT se pokazao kao najbolji za korpuse 1 i 3, dok je program Tree Tagger ,,pobedio" u slučaju korpusa 2. Takođe, stiče se utisak da se MX generalno bolje snalazi sa nepoznatim rečima od drugih - ali, ako pogledamo tabelu 3, tamo se vidi opet da nad nepoznatim rečima bolje prolazi TnT nad korpusima 1 i 3, i Tree Tagger nad korpusom 2. Iz te tabele bi se moglo zaključiti da Bril ima bolje rezultate nego drugi u etiketiranju poznatih reči, ali to treba uzeti s rezervom kao ocenu performansi. Možda bi neki opšti zaključak bio da se Tree Tagger veoma dobro snalazi sa manjim brojem etiketa, ali da TnT u suprotnom odnosi laku ,,pobedu". Na kraju, reč je o jako malim razlikama (SVM je tu negde blizu, isto).

Svaki od ovih programa se može do izvesne granice dodatno podešavati da bi se dobili bolji rezultati (u čemu SVM prednjači po mogućnostima, mada tada učenje postaje sporo preko svake mere), ali performanse svakako zavise i od korpusa. Tako je BNC oko 1000 veći korpus, primera radi, ima skoro tri puta manje etiketa od korpusa 3 i tek nešto manji broj leksema - dakle, presudno utiču: dužina i broj etiketa. Korpus, kakav je korpus 2, sa manjim brojem etiketa (ili čak manje dužine) je idealan za istraživanja na pojedinim programima i istraživanju njihovih krajnjih dometa performansi, ali ne i za krajnju eksploataciju programa za etiketiranje. Neistraženi izazov u ovom radu je otkrivanje upravo tih krajnjih rezultata i naknadno poređenje ovih programa.

Optimalna veličina skupa obučavanja (korpusa) jedno je od najbitnijih dostignuća statističke teorije mašinskog učenja - pokazuje se da zavisi samo od željene greške i verovatnoće učenja, i od veličine i strukture prostora prostora hipoteza (detalji o tome se mogu naći u [8] čiji je autor Vladimir Vapnik, jedan od pionira u ovoj oblasti i otac SVM metode klasifikacije). Veći korpus (tipa korpusa 3) bi svakako bio potreban za preciznija ispitivanja i bolje performanse, uz vođenje računa o prezasićenju učenja (što svi ovi programi implicitno imaju ugrađeno).

6 Dodatak

Svi programi su instalirani bez većih problema prema uz njih priloženim uputstvima. Jedino zbog jednostavnosti skripta i Brilovog ograničenja pokretanja programa u `Bin_ and_Data` direktorijumu sam izabrao da svig programi u tom direktorijumu (`/usr/brill/RULE_BASED_TAGGER_V1.14/Bin_and_Data`) drže i svoje modele (korpusi su izdvojeni u `/usr/data`). Zbog starog Unix stila Brilovih skriptova trebalo je promeniti sort argumente (ili odgovarajuće promenljive okruženja) i kreirati simbolički link za perl (ln -s /usr/bin/perl /usr/local/bin/perl, zbog stare sheebang linije), kao i za /bin/ksh u kod MX skriptova (ln -s /bin/zsh /bin/ksh). Najveći problem sam imao sa Tree Tagger-om koji ne podržava učenje leksikona, ali leksikon jeste preduslov učenja - mogao sam možda da iskoristim neki od leksikona iz drugih programa, ali oni ne podržavaju leme. Zato sam napisao pomoćni AWK skript koji na osnovu datog etiketiranog i lematiziranog korpusa generiš leksikon za Tree Tagger (`lex.awk`). Ukupno vreme trajanja skripta sa učenjem i evaluacijom za 2.5K korpus je ispod sat vremena, dok je za 11K korpus trebalo više od 9 sati (na Intel(R) Core(TM)2 Duo CPU / T7700 @ 2.40GHz). Izabrao sam Linux Fedora 8 kao platformu za testiranje kako zbog toga što mnogi programi (Tree Tagger, Brill RBT) ne podržavaju učenje na ne-x platformama (Windows, mada postoje i neki programi za etiketiranje i pod Windows-om) ili imaju i druga ograničenja - tako i zbog udobnosti rada sa niskama (regularnim izrazima, itd) koja je prisutna na ovoj platformi (pre svega). Skriptove sam ostavio na raspolaganju za preuzimanje sa [download].

Konfiguraciona datoteka `config.short.svmt`:

```
# ------------------------------------------------------------
# SVMT configuration file
# ------------ location of the training set------------------
TRAINSET = /usr/brill/RULE_BASED_TAGGER_V1.14/Bin_and_Data/learn.svm
# -------------- location of the validation set ---------------
VALSET = /usr/brill/RULE_BASED_TAGGER_V1.14/Bin_and_Data/test.txt
# -------------- location of the test set --------------------
TESTSET = /usr/brill/RULE_BASED_TAGGER_V1.14/Bin_and_Data/test.txt
# -------------- location of the Joachims svmlight software ---
SVMDIR = /usr/svm_light
# -------------- name of the model to create -----------------
NAME = test
# --------action items ----------------------------------------
do M0 LRL
```

```
# ------------------------------------------------------------
```

Skriptovi za razdvajanje celog korpusa na korpus za učenje i korpus za testiranje, `cross-tab10.sh`:

```
#!/bin/sh
cat $1 | ../Utilities/cross-tab10.prl $(cat $1 | grep "SENT" \
 | wc -l | grep -Eo "^[0-9]+") $2 $3 $4
```

... realizovan kao Perl skript, `cross-tab10.prl`:

```
#!/usr/bin/perl
#
# Za potrebe 10-fold cross validacije Tree Tagger korpusa
# - upotreba:
# cat CORPUS | cross10 <broj linija u CORPUS-u> \
#                 <CHUNK> CORPUS-1 CORPUS-2
#
$cnt = 1;
$sentences = $ARGV[0];
$offset = $ARGV[1]*$sentences/10;
$offset2 = $offset+($sentences/10);
open(XX,">$ARGV[2]");
open(YY,">$ARGV[3]");
while(<STDIN>) {
    if (($cnt <= $offset) || ($cnt>$offset2)) {
      print XX $_;
    }
    else {
      print YY $_;
    }
    if ($_ =~ /\tSENT\t/) { $cnt = $cnt + 1; }
}
close(XX);
close(YY);
```

AWK kôd za generisanje leksikona za Tree Tagger, `lex.awk`:

```
BEGIN { FS = "\t|[[:space:]]"; }
{
  if (!($1 in lex_cnt)) {
     lex_cnt[$1]=1;
     lex_tag[1,$1]=$2;
```

```
     lex_lem[1,$1]=$3;
  }
  else {
     idx=0;
     for(n=1;n<=lex_cnt[$1];n++) {
          if (lex_tag[n,$1]==$2) { idx=n; break; }
     }
     if (idx>0) {
          #if (lex_lem[idx,$1]!=$3) {
          #     lex_cnt[$1]++;
          #     lex_tag[lex_cnt[$1],$1]=$2;
          #     lex_lem[lex_cnt[$1],$1]=$3;
          #}
     }
     else {
          lex_cnt[$1]++;
          lex_tag[lex_cnt[$1],$1]=$2;
          lex_lem[lex_cnt[$1],$1]=$3;
     }
  }
}
END {
  for (x in lex_cnt) {
     line = x "\t";
     for(n=1;n<lex_cnt[x];n++) {
          if (lex_lem[n,x]=="") lex_lem[n,x]="?";
          line = line lex_tag[n,x] "\t" lex_lem[n,x] "\t";
     }
     if (lex_lem[n,x]=="") lex_lem[n,x]="?";
     line = line lex_tag[n=lex_cnt[x],x] "\t" lex_lem[n,x];
     print line;
  }
}
```

Za pretvaranje vertikalnog ulaznog korpusa u horizontalni Brilov koristi se `tt.awk` (slično je rešen i `mx.awk` za MXPOST):

```
BEGIN { FS="\t"; ORS=""; }
{
   if ($2!="SENT") print $1 "/" $2 " ";
   else print $1 "/PUNCT\n";
}
```

Za vraćanje Brilovog horizontalnog formata u vertikalni koristi se `b2t.awk` (i slično njemu, `m2t.awk` za MXPOST):

```
BEGIN { RS=" "; }
{
  for (n=1; n<=NF; n++) {
     split($n,polja,"/");
     print polja[1] "\t" polja[2];
  }
}
```

Pomoćni skript za učenje Brilovog programa, `script.sh`:

```
#!/bin/sh
cat TAGGED-CORPUS-ENTIRE | divide-in-two-rand.prl TAGGED-CORPUS \
  TAGGED-CORPUS-2
cat TAGGED-CORPUS | tagged-to-untagged.prl > UNTAGGED-CORPUS
cat UNTAGGED-CORPUS | wordlist-make.prl | sort -k 1n -rn | \
  awk '{ print $1}' > BIGWORDLIST
cat TAGGED-CORPUS | word-tag-count.prl | sort -k 2n -rn \
  > SMALLWORDTAGLIST
cat UNTAGGED-CORPUS | bigram-generate.prl | awk '{ print $1,$2}' \
  > BIGBIGRAMLIST
unknown-lexical-learn.prl BIGWORDLIST SMALLWORDTAGLIST \
  BIGBIGRAMLIST 300 LEXRULEOUTFILE
cat TAGGED-CORPUS | make-restricted-lexicon.prl \
  > TRAINING.LEXICON
cat TAGGED-CORPUS-ENTIRE | make-restricted-lexicon.prl \
  > FINAL.LEXICON
cat TAGGED-CORPUS-2 | tagged-to-untagged.prl \
  > UNTAGGED-CORPUS-2
tagger TRAINING.LEXICON UNTAGGED-CORPUS-2 BIGBIGRAMLIST \
  LEXRULEOUTFILE /dev/null -w BIGWORDLIST \
  -i DUMMY-TAGGED-CORPUS > /dev/null
contextual-rule-learn TAGGED-CORPUS-2 DUMMY-TAGGED-CORPUS \
  CONTEXT-RULEFILE TRAINING.LEXICON
```

Glavni skript, `data.sh`:

```
#!/bin/sh
PATH=$PATH:/usr/brill/RULE_BASED_TAGGER_V1.14/Bin_and_Data:
/usr/brill/RULE_BASED_TAGGER_V1.14/Learner_Code:
/usr/brill/RULE_BASED_TAGGER_V1.14/Utilities:
/usr/svm_light:/usr/SVMTool-1.3/bin
CLASSPATH=/usr/mx/mxpost.jar
PERL5LIB=/usr/SVMTool-1.3/lib:$PERL5LIB
MODELS=.
export PERL5LIB
export PATH
export CLASSPATH
xsltproc -v -o out01.txt data.xsl /usr/data/01HP-SR-Lemma.xml
xsltproc -v -o out02.txt data.xsl /usr/data/01HP-SR-Lemma.xml
xsltproc -v -o out03.txt data.xsl /usr/data/01HP-SR-Lemma.xml
xsltproc -v -o out04.txt data.xsl /usr/data/Radionica-SR-lemma.xml
xsltproc -v -o out05.txt data.xsl /usr/data/Radiodif-SR-lemma.xml
#cat /usr/data/oana-sr.xml | awk -f punct.awk | xsltproc \
# -v data2.xsl - | awk -f data.awk > out1.txt
cat out01.txt | awk -f data.awk > out1.txt
cat out02.txt | awk -f data.awk >> out1.txt
cat out03.txt | awk -f data.awk >> out1.txt
cat out04.txt | awk -f data.awk >> out1.txt
cat out05.txt | awk -f data.awk >> out1.txt
for t in tt svm tnt mx rbt
do
  echo statistike za $t: > stat_${t}.txt
done
date > stat_all.txt
cat out1.txt | awk -f lex.awk > lex2.txt
echo broj leksema, etiketa i lema: \
  $(cat out1.txt | awk -f lex.awk | wc -l), $(cat out1.txt \
    | cut -f 2 | \
   sort -u | wc -l), $((cat lex2.txt | cut -f 3; cat lex2.txt \
    | cut -f 5; \
   cat lex2.txt | cut -f 7) | sort -u | wc -l) >> stat_all.txt
for n in 0 1 2 3 4 5 6 7 8 9
do
  #. cross10.sh brill.txt $n TAGGED-CORPUS-ENTIRE TEST-CORPUS
  . cross-tab10.sh out1.txt $n learn.txt test.txt
```

```
cat learn.txt | awk -f tt.awk > TAGGED-CORPUS-ENTIRE
cat test.txt | awk -f tt.awk > TEST-CORPUS
cat learn.txt | awk -f lex.awk > lexicon.ltt
# Tree Tagger:
cat learn.txt | cut -f 1,2 > input.txt
cat learn.txt | cut -f 2 | sort -u > openclass.txt
echo $(cat lexicon.ltt | wc -l), $(cat openclass.txt | wc -l) \
  >> stat_all.txt
/usr/tt/bin/train-tree-tagger lexicon.ltt openclass.txt \
  input.txt tt.par
cat test.txt | cut -f 1 > test.tt
/usr/tt/bin/tree-tagger -token -lemma tt.par test.tt \
 | cut -f 1,2 > test_tt.txt
yes | cat test.txt | cut -f 1,2 > test_tt0.txt
echo $(cat test_tt.txt | wc -l),$(diff -Bd test_tt.txt \
 test_tt0.txt | grep ">" | wc -l) >> stat_tt.txt
# SVM:
cat learn.txt | sed -r 's/\t/\x20/g' > learn.svm
SVMTlearn /usr/SVMTool-1.3/bin/config.short.svmt
cat test.tt | SVMTagger -V 2 -T 0 test | sed 's/\x20/\t/g' \
  > test_svm.txt
yes | cp test_tt0.txt test_svm0.txt 2> /dev/null
echo $(cat test_svm.txt | wc -l),$(diff -Bd test_svm.txt \
  test_svm0.txt | grep ">" | wc -l) >> stat_svm.txt
# TNT:
yes | cp input.txt input.tt 2> /dev/nul
/usr/tnt/tnt-para input.tt
/usr/tnt/tnt input test.tt | grep -E "^[^%]" | sed -r \
  's/(\x20|\t)+/\t/g' > test_tnt.txt
cat test.txt | cut -f 1,2 > test_tnt0.txt
echo $(cat test_tnt.txt | wc -l),$(diff -Bd test_tnt.txt \
  test_tnt0.txt | grep ">" | wc -l) >> stat_tnt.txt
# MX:
cat learn.txt | awk -f mx.awk > mx.txt
cat TEST-CORPUS | tagged-to-untagged.prl > UNTAGGED-TEST-CORPUS
/usr/mx/trainmxpost . mx.txt
/usr/mx/mxpost . < UNTAGGED-TEST-CORPUS | awk -f m2t.awk \
  > test_mx.txt
cat TEST-CORPUS | awk '{gsub(/[/]/,"_",$0); print $0;}' \
| awk -f m2t.awk > test_mx0.txt
echo $(cat test_mx.txt | wc -l),$(diff -Bd test_mx.txt \
```

```
   test_mx0.txt | grep ">" | wc -l) >> stat_mx.txt
  # Brill Tagger:
  . script.sh
  tagger FINAL.LEXICON UNTAGGED-TEST-CORPUS BIGRAMS \
    LEXRULEOUTFILE CONTEXT-RULEFILE \
   | awk -f b2t.awk > test_rbt.txt
  cat TEST-CORPUS | awk -f b2t.awk > test_rbt0.txt
  echo $(cat test_rbt.txt | wc -l),$(diff -Bd test_rbt.txt \
   test_rbt0.txt | grep ">" | wc -l) >> stat_rbt.txt
  #
  # svaki od testova za sobom ostavi test_{tagger}.txt etiketirani
  # test korpus i test_{tagger}0.txt "gold standard"
  # na osnovu kojh se prate performanse (koristi se token po liniji
  # i komanda za brojanje razli\v{c}itih linija)
  #
  for i in tt svm tnt mx rbt
  do
    . unknown.sh ${i} >> stat_${i}.txt
  done
done
cat stat_all.txt | awk -f report2.awk >> stat_all.txt
for t in tt svm tnt mx rbt
do
  cat stat_${t}.txt | awk -f report1.awk >> stat_all.txt
done
date >> stat_all.txt
# cleanup:
yes | rm input.* 2> /dev/null
yes | rm model.* 2> /dev/null
yes | rm *.par 2> /dev/null
yes | rm *.voc 2> /dev/null
yes | rm test*.txt 2> /dev/null
yes | rm out*.txt 2> /dev/null
yes | rm learn.txt 2> /dev/null
yes | rm mx.txt 2> /dev/null
yes | rm *.DIC* 2> /dev/null
yes | rm test.* 2> /dev/null
```

Skript za prebrojavanje nepoznatih reči, `unkown.awk`:

```
BEGIN { cnt=0; prev=""; }
{
 if ($0 ~ /\_$/) {
  if ((prev!="")&&($0 != prev "_")) if ($0 != prev) {
     cnt++;
     # print prev "/" $0;
  }
 }
 prev = gensub(/^(([[:alnum:]]|-|\/|')*)(.)*/,"\\1","g");
}
END { print "cnt=" cnt;}
```

... i njegov pomoćni skript, `unknown.sh`:

```
#!/bin/sh
diff -Bd test_$1.txt test_$10.txt | grep ">" | cut -d " " \
  -f 2 | cut -f 1 > t1.txt
#cat lexicon.ltt | cut -f 1 > t2.txt
cat learn.txt | cut -f 1 | sort -u > t2.txt
cat t1.txt | awk '{print $0 "_"; }' > t.txt
cat t2.txt >> t.txt
cat t.txt | sort | awk -f unknown.awk
```

Skriptovi za agregiranje statističkih podataka, `report1.awk`:

```
BEGIN { HEADER=0; avg=avg2=sum=max=min=umax=umin=cnt=usum=0; }
{
 if (!HEADER) { hdr = $0; HEADER=1; }
 else {
   if ($0 !~ /^cnt=/) {
     split($0,cols,/,([[:space:]])*/);
     ratio = 100*(1-(cols[2]/cols[1]));
     if (cnt==0) min=max=ratio;
     cnt++; sum += ratio; r[cnt]=ratio;
     if (min>ratio) min = ratio;
     if (max<ratio) max = ratio;
   }
   else {
     unk[cnt] = gensub(/cnt=((.)*)/,"\\1","g")/cols[2]*100;
     avg += unk[cnt]*cols[2]/100; avg2 += cols[2];
     if (cnt==1) umin=umax=unk[cnt];
```

```
     if (umin>unk[cnt]) umin = unk[cnt];
     if (umax<unk[cnt]) umax = unk[cnt];
     usum += unk[cnt];
   }
 }
}
END {
  print "\t" hdr; var=uvar=0;
  for (i=1; i<=cnt; i++)
    { var += ((sum/cnt)-r[i])^2; uvar += ((usum/cnt)-unk[i])^2; }
  printf "MIN/UNKNOWN:\t\t%3.2f%%/%3.2f%%\n", min, umin;
  printf "AVG/UNKNOWN:\t\t%3.2f%%/%3.2f%%\n", sum/cnt, usum/cnt;
  printf "MAX/UNKNOWN:\t\t%3.2f%%/%3.2f%%\n", max, umax;
  printf "DEV/UNKNOWN:\t\t%3.2f%%/%3.2f%%\n", sqrt(var/cnt), sqrt(uvar/cnt);
  print  "AVG. KN./UNK.: " (avg2-avg)/cnt "/" avg/cnt;
  # ((avg/cnt)*(1-(sum/cnt)/100)*(usum/cnt)/100) ", "
}
```

... i `report2.awk`:

```
BEGIN { HEADER=0; sumsum2=max1=max2=min1=min2=cnt=0; }
/broj leksema/ { next; }
{
 if (!HEADER) { hdr = $0; HEADER=1; }
 else {
   split($0,cols,/,([[:space:]])*/);
   if (cnt==0) { min1=max1=cols[1]; min2=max2=cols[2]; }
   cnt++; sum1 += cols[1]; sum2 += cols[2];
   if (min1>cols[1]) min1 = cols[1];
   if (max1<cols[1]) max1 = cols[1];
   if (min2>cols[2]) min2 = cols[2];
   if (max2<cols[2]) max2 = cols[2];
 }
}
END {
  print "MIN:\t\t", min1 ", " min2;
  printf "AVG:\t\t %d, %d\n", sum1/cnt, sum2/cnt;
  print "MAX:\t\t", max1 ", " max2;
}
```

Dokumenti, knjige i izvori korišćeni tokom pisanja ovog rada -

Literatura

[1] Massive multi lingual corpus compilation: Acquis Communautaire and totale, Tomaž Erjavec, Camelia Ignat, Bruno Pouliquen, Ralf Steinberger

http://nl.ijs.si/ME/V3/msd/ http://nl.ijs.si/et/talks/SFB441/tue-slides/
http://langtech.jrc.it/Documents/LTC-2005_Multilingual-corpus-compilation_Erjavec-et-al.pdf

[2] A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Lawrence R. Rabiner

[3] HMM and the Baum-Welch Algorithm, IEEE IT Society Newsletter, Lloyd R. Welch - http://www.itsoc.org/publications/nltr/it_dec_03final.pdf

[4] Pattern Recognition in Speech and Language Processing, ISBN 0-8493-1232-9, Georgia Institute of Technology, 2003.

[5] The CLAWS Word-tagging System,
R. Garside, G. Leech and G. Sampson (1987)
http://www.comp.lancs.ac.uk/computing/research/ucrel/claws/

[6] Machine Learning, Tom M. Mitchell, 1997.

[7] Introduction To Machine Learning, Nils J. Nillson

[8] The Nature of Statistical Learning Theory, Vapnik Vladimir N.

[9] Fast Training of Support Vector Machines using Sequential Minimal Optimization, John C. Platt

[10] http://aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art)

[11] Stanford Tagger http://nlp.stanford.edu/software/tagger.shtml

[12] LTAG POS Tagger http://www.cis.upenn.edu/~xtag/spinal/

[13] Orwell http://nl.ijs.si/ME/bib/mte-nlprs01.pdf

[14] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html

[15] http://www.coli.uni-saarland.de/~thorsten/tnt/ http://coli.uni-sb.de/~thorsten/tnt/

[16] http://research.microsoft.com/users/brill/ http://www.cs.jhu.edu/~brill
http://www.cst.dk/download/tagger/
http://www.tech.plym.ac.uk/soc/staff/guidbugm/software/RULE_BASED_TAGGER_V.1.14.tar.Z
http://www.ling.gu.se/~lager/Home/brilltagger_ui.html
http://www.calcaria.net/brilltagger_win32/
http://www.umiacs.umd.edu/~jimmylin/downloads/index.html

[17] http://www.lsi.upc.edu/~nlp/SVMTool/ http://svmlight.joachims.org/

[18] http://www.ee.unimelb.edu.au/staff/apsh/svm/

[19] TiMBL: http://ilk.kub.nl http://ilk.uvt.nl/mbt/

[20] http://www.cis.upenn.edu/adwait/jmx/ ftp://ftp.cis.upenn.edu/pub/adwait/jmx/
http://www.inf.ed.ac.uk/resources/nlp/local_doc/MXPOST.html

[21] Penn TB http://www.cis.upenn.edu/~treebank/
ICE http://www.comp.leeds.ac.uk/amalgam/tagsets/ice.html
viewer http://dingo.sbs.arizona.edu/~sandiway/treebankviewer/index.html

[22] http://wortschatz.uni-leipzig.de/~cbiemann/pub/2007/BiemannGiulianoGliozzoRANLP07.pdf

[23] http://www.tei-c.org

[24] http://tactweb.humanities.mcmaster.ca/tactweb/doc/tactweb.htm

[download] http://users.hemo.net/shoom/taggers.tar.gz

[www] http://xmlsoft.org/XSLT/ http://xmlsoft.org/XSLT/xsltproc.html http://www.w3schools.com/xsl/
http://svmlight.joachims.org/
http://research.microsoft.com/~jplatt/smo.html
http://www.support-vector-machines.org/SVM_soft.html
http://www.cs.waikato.ac.nz/ml/weka/
http://world.std.com/~rjs/pubs.html
http://linkage.rockefeller.edu/wli/zipf/cmpsci546_spring2002_notes.pdf
http://math.ucsd.edu/~crypto/java/ENTROPY/
http://en.literateprograms.org/POS_tagger_(Java)
http://en.wikipedia.org/wiki/Colorless_green_ideas_sleep_furiously
http://en.wikipedia.org/wiki/Word_sense_disambiguation
http://en.wikipedia.org/wiki/Natural_language_processing
http://en.wikipedia.org/wiki/Part-of-speech_tagging
http://en.wikipedia.org/wiki/Parts_of_speech
http://en.wikipedia.org/wiki/Information_retrieval
http://en.wikipedia.org/wiki/Lemmatisation
http://en.wikipedia.org/wiki/Chomsky_hierarchy
http://en.wikipedia.org/wiki/Algorithmic_probability
http://en.wikipedia.org/wiki/Kolmogorov_complexity
http://en.wikipedia.org/wiki/Zipf's_law
http://www-nlp.stanford.edu/links/statnlp.html#Taggers
http://www-a2k.is.tokushima-u.ac.jp/member/kita/NLP/nlp_tools.html
http://morphix-nlp.berlios.de/
http://www.pdg.cnb.uam.es/martink/LINKS/nlp_tools_links.htm
http://personal.cityu.edu.hk/~davidlee/devotedtocorpora/software.htm
http://crl.nmsu.edu/cgi-bin/Tools/CLR/clrcat
http://people.csail.mit.edu/mcollins/ ftp://ftp.cis.upenn.edu/pub/mcollins/PARSER.tar.gz
http://www.cis.upenn.edu/~dbikel/software.html
http://gate.ac.uk/ http://en.wikipedia.org/wiki/General_Architecture_for_Text_Engineering