Obrada prirodnog jezika

Pribisalić, Ena

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Ena Pribisalić

Obrada prirodnog jezika

Diplomski rad

Osijek, 2019

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Ena Pribisalić

Obrada prirodnog jezika

Diplomski rad

Mentorica: izv.prof.dr.sc. Darija Marković

Osijek, 2019

Sadržaj

1 Uvod **1**

2 Riječi **2**
2.1 Tekstualni korpus . . . 2
2.2 Regularni izrazi . . . 2
2.3 Što su riječi? . . . 5
2.4 Normalizacija teksta . . . 6
2.5 Mjerenje sličnosti riječi . . . 9
2.6 *N*-grami . . . 12
2.6.1 Procjena jezičnih modela . . . 15
2.6.2 Zaglađivanje . . . 16
2.6.3 *Backoff* i interpolacija . . . 17
2.7 Gramatičko tagiranje . . . 18

3 Govor **19**
3.1 Fonetika . . . 20
3.1.1 Fonetska transkripcija . . . 20
3.1.2 Artikularna fonetika . . . 21
3.1.3 Akustična fonetika i signali . . . 21
3.2 Sinteza govora . . . 27
3.2.1 Lančana sinteza . . . 27

4 Sintaksa **29**
4.1 Formalne gramatike . . . 29
4.1.1 Kontekstno-slobodne gramatike . . . 29
4.1.2 Osnovna gramatička pravila engleskog jezika . . . 31
4.1.3 Treebanks . . . 31
4.2 Sintaktičko parsiranje . . . 32
4.2.1 CKY algoritam . . . 32
4.2.2 Parcijalno parsiranje . . . 34
4.3 Statističko parsiranje . . . 34
4.3.1 Vjerojatnosne kontekstno-slobodne gramatike . . . 34
4.3.2 Vjerojatnosni CKY algoritam . . . 35

5 Primjena: Izvlačenje informacija **37**
5.1 Prepoznavanje imenskih entiteta . . . 38
5.1.1 Označavanje niza . . . 38
5.1.2 Procjena . . . 39

Literatura **40**

Sažetak **41**

Summary **42**

Životopis **43**

1 Uvod

Obrada prirodnog jezika (*natural language processing*) je grana umjetne inteligencije koja se bavi sposobnošću strojeva da razumiju i interpretiraju ljudski jezik kako je pisan ili izgovoren. Cilj obrade prirodnog jezika je učiniti računala ili strojeve inteligentnima poput ljudi kada je riječ o razumijevanju jezika.

U današnje vrijeme postoje glasovni asistenti kao što su Siri (Apple), Alexa (Amazon), Cortana (Microsoft), Bixby (Samsung) te Google Assistant (Google). Uz pravilo formuliranu naredbu, ovi asistenti puštaju glazbu, postavljaju alarme za buđenje, zovu kontakte koje želimo, daju nam pregled tjednih ili mjesečnih planova i mnogo više. Za to im je potrebno prepoznavanje govora, odnosno moraju snimiti zvuk pomoću mikrofona, obraditi audio podatke, interpretirati podatke te na kraju izvršiti naredbu koja im je zadana. Ovi asistenti postali su dio naše svakodnevice, no početci umjetne inteligencije i komunikacije ljudi i strojeva bili su malo drugačiji.

Programer Joseph Weizenbaum stvorio je ELIZA-u 1966. godine i ona nije bila vrsta asistenta kakvima se danas služimo. Naime, ELIZA, jedan od prvih programa umjetne inteligencije, je bila terapeut. Ideja je bila stvoriti računalo koje može komunicirati s ljudima i natjerati ih da misle kako ne razgovaraju sa strojem, nego s drugom osobom. ELIZA je bila terapeut iz nekoliko razloga. Terapeuti postavljaju otvorena pitanja te pokušavaju natjerati sugovornika da govori više o tome što ga muči. Stoga je ELIZA trebala samo parsirati rečenice koje je dobila na obradu te ih preformulirati u pitanje koje je povezano s ulaznom rečenicom. Mnogi ljudi su zaista vjerovali da razgovaraju s terapeutom, a ne sa strojem.

Obrada prirodnog jezika od tada je uvelike napredovala. No, ova grana umjetne inteligencije se još uvijek razvija i nailazi na velike poteškoće zbog prirode ljudskog jezika. Pravila jezika koja ljudi koriste pri govoru i pisanju mogu biti jako apstraktna, primjerice pri korištenju sarkazma. Upravo zato pri obradi prirodnog jezika veliku ulogu ima lingvistika.

Koraci koje poduzimamo pri obradi prirodnog jezika redom uključuju:

- leksičku analizu - identificiranje i analizu strukture riječi
- sintaktičku analizu - analizu strukture riječi u rečenici (gramatika)
- semantičku analizu - identificiranje značenja riječi i teksta
- integraciju diskursa - analizu značenja rečenice u odnosu na prethodnu rečenicu
- pragmatičku analizu - ponovnu interpretaciju onoga što se reklo kako bi se identificiralo ono što se mislilo reći.

U ovome radu krenut ćemo od osnovnih jedinica jezika, riječi. Vidjet ćemo kako možemo pretraživati tekstove, računati sličnost ili različitost riječi, te kako svakoj riječi pridružiti pripadnu vrstu. Nakon toga, pozabavit ćemo se najučinkovitijim načinom primanja i prenošenja informacija, govorom. U tom dijelu vidjet ćemo kako strojevi mogu digitalizirati audio datoteke te kako se govor može stvoriti iz teksta. Kod sintaktičke analize upoznat ćemo se s kontekstno-slobodnim gramatikama i različitim oblicima parsiranja. Na kraju rada, pojasnit ćemo proces izvlačenja informacija te nešto detaljnije objasniti njegov prvi korak - prepoznavanje imenskih entiteta.

2 Riječi

2.1 Tekstualni korpus

U jezikoslovlju, korpus (*corpus*) je kolekcija (skup) pisanog ili govornog materijala pohranjenog na računalu koji se koristi za otkrivanje načina korištenja jezika (definicija iz: Cambridge English Dictionary). Dakle, glavna svrha korpusa je potvrditi (ili opovrgnuti) hipotezu o nekom jeziku. Bitno je naglasiti da je korpus reprezentativni uzorak u odnosu na istraživačku hipotezu koji ima definiranu veličinu i sadržaj. Korpusi su dostupni svima kako bi se rezultati istraživanja mogli usporediti ili ponoviti te opisuju kako ljudi zaista koriste jezik, u njemu ne postoje izmišljeni primjeri.

Brownov korpus, jedan od najpoznatijih korpusa, je zbirka od milijun riječi sastavljena od 500 pisanih tekstova na engleskom jeziku iz 15 različitih žanrova. Sastavljen je na Brown Sveučilištu 1960.-ih godina, a više o njemu može se pročitati u [3].

2.2 Regularni izrazi

Regularni izrazi predstavljaju standardnu (algebarsku) notaciju za karakteriziranje tekstualnih nizova (stringova). Dakle, za regularne izraze potreban nam je uzorak teksta koji tražimo te korpus tekstova na kojima vršimo pretragu. Pretraga može vratiti čitave dokumente ili Web stranice te individualne riječi ili linije dokumenta, ovisno što se nalazi u korpusu.
Regularni izrazi, koje koristi i ELIZA, koriste se u računalnim jezicima, alatima za obradu riječi te teksta. No, kako ELIZA koristi regularne izraze? Ona prepoznaje određene izraze te ih modificira u prikladne odgovore. Slijedi primjer razgovora koji smo vodili s ELIZA-om (vidi [5]).

```
ELIZA: How do you do.  Please tell me your problem.
YOU: I can't sleep.
ELIZA: Do you really want to be able to sleep?
YOU: I do.
ELIZA: You say you do?
YOU: I need sleep.  We all do.
ELIZA: What does wanting sleep have to do with this discussion?
```

Moderni razgovorni agenti mogu i puno više od ovoga, kao na primjer davati odgovore na pitanja, pronaći restoran ili hotel, rezervirati karte za putovanja i slično.

Riječ u svijetu regularnih izraza označavat će bilo kakav niz znamenki, donjih crta ili slova. Operatori koji se koriste nad regularnim izrazima prikazani su u hijerarhijskom poretku u *Tablici 1.*

Ime operatora	**Oznaka (primjer) operatora**
Zagrade	()
Brojači	* + ? {}
Nizovi i „sidra" (*anchors*)	the ^my end$
Disjunkcija	|

Tablica 1: Hijerarhijski prikaz operatora nad regularnim izrazima

Operatori brojanja pomnije su objašnjeni u *Tablici 2*.

Regularni izraz	Podudaranje
*	0 ili više pojavljivanja prethodnog znaka ili izraza
+	1 ili više pojavljivanja prethodnog znaka ili izraza
?	točno 0 ili 1 pojavljivanje prethodnog znaka ili izraza
{n}	n pojavljivanja prethodnog znaka ili izraza
{n,m}	između n i m pojavljivanja prethodnog znaka ili izraza
{n,}	barem n pojavljivanja prethodnog znaka ili izraza
{,m}	najviše m pojavljivanja prethodnog znaka ili izraza

Tablica 2: Operatori brojanja za regularne izraze

Operatori usidravanja (`^ i $`) koriste se za usidravanje regularnih izraza na određena mjesta u stringu. Operator `^` odnosi se na početak linije, a operator `$` na kraj linije. Postoje još dva operatora usidravanja, `\b`, koji odgovara granici riječi, i `\B`, koji odgovara ne-granici. Tako će regularni izraz `/\bthe\b/` odgovarati riječi *the*, ali ne i riječi *other*.

Na regularne izraze možemo primijeniti i disjunkciju, grupiranje te određenim izrazima dati prioritet nad drugima. Za disjunkciju koristimo operator |, pa će izraz `/cat|dog/` odgovarati ili prvom (`cat`) ili drugom (`dog`) stringu. No, ako želimo disjunkciju primijeniti samo na određeni uzorak, koristit ćemo zagrade (i). Stoga će se disjunkcija u izrazu `/pupp(y|ies)/` odnositi samo na dio unutar zagrada, te ćemo time moći pronaći i jedninu i množinu tražene riječi (*puppy/puppies*).

U *Tablici 3* možemo vidjeti neke regularne izraze, kako su dani u [4].

Regularni izraz	Podudaranje
`/s/`	slovo „s”
`/pozdrav/`	riječ „pozdrav”
`/?/`	upitnik
`/[Pp]as/`	riječ „Pas” ili „pas”
`/[A-Z]/`	veliko slovo
`/[a-z]/`	malo slovo
`/[0-9]/`	znamenka
`/[^A-Z]/`	ne veliko slovo
`/[^ \.]/`	ne točka
`/[e^]/`	slovo „e” ili znak „^”
`/[a^b]/`	uzorak „a^b”
`/Ivana?/`	„Ivan” ili „Ivana”
`/a*/`	nula ili više slova „a”
`/[ab]*/`	nula ili više slova „a” ili „b”
`/[0-9][0-9]*/`	jedna ili više znamenki
`/[0-9]+/`	jedna ili više znamenki
`/don.t/`	riječi oblika „don”+znak+„t”, kao „donut”, „don’t” itd.

Tablica 3: Regularni izrazi

Važno je napomenuti da su regularni izrazi pohlepni, što znači da će uvijek vratiti najveći mogući string koji odgovara danoj formuli. Primjerice, primijenimo li izraz `/[a-z]*/` na tekst *once upon a time*, on će se poklapati s 0 ili više slova, pa su mogućnosti prazan string, prvo slovo, prva dva slova, itd. Stoga, rješenje je u ovom slučaju riječ *once*. Možemo provesti i ne-pohlepno pretraživanje tako da nakon operatora * dodamo još i operator `?`, pa će operator `*?` vratiti što je moguće manje teksta.

Radi uštede vremena pri tipkanju, postoje skraćenice nekih najčešće korištenih regularnih izraza prikazane u *Tablici 4*. Za neke posebne znakove u regularnim izrazima koristimo obrnutu kosu crtu \, a najčešće korišteni su oni za novi red i tabulator (vidi *Tablicu 5*).

Regularni izraz	**Proširen RI**	**Podudaranje**
`\d`	`[0-9]`	znamenka
`\D`	`[^0-9]`	ne-znamenka
`\w`	`[a-zA-Z0-9_]`	alfanumerički znak ili donja crta
`\W`	`[^\w]`	ne-alfanumerički znak
`\s`	`␣\r\t\n\f`	razmak
`\S`	`^\s`	ne-razmak

Tablica 4: Skraćenice nekih najkorištenijih regularnih izraza

Regularni izraz	**Podudaranje**
`\*`	zvjezdica „*”
`\.`	točka „.”
`\?`	upitnik „?”
`\n`	novi red
`\t`	tabulator

Tablica 5: Znakovi uz koje je potrebna obrnuta kosa crta

Jedna od vrlo važnih primjena regularnih izraza je supstitucija, odnosno zamjena. U tu svrhu koristimo `s/reg/uzorak`, čime stringove koji odgovaraju regularnom izrazu `reg` zamjenjujemo stringom `uzorak`. Kod ovoga procesa od pomoći je i referenciranje na dobivene rezultate pretrage, uz operator `\1`, kao u izrazu `/the (.*)er they were, the \1er they will be/`. Dakle, `\1` će se zamijeniti s onim stringom koji odgovara prvom izrazu u zagradama. Ovakvo korištenje zagrada za pohranjivanje uzorka u memoriji zove se obuhvaćena grupa (*capture group*). Pri korištenju obuhvaćenih grupa, rezultirajuće poklapanje pohranjuje se u registar. Grupe brojimo krećući od 1 na način da prebrojavamo broj otvorenih zagrada, s lijeva na desno. Tako će `\2` odgovarati drugoj, a `\3` trećoj obuhvaćenoj grupi. Na ovaj način grupiramo regularne izraze.

Vidimo da zagrade imaju dvostruku funkciju, koristimo ih za specificiranje poretka u kojem želimo da se operacije izvršavaju i za pohranjivanje nečega u registar. No, što ako ne želimo pohraniti uzorak u registar? Tada možemo koristiti ne-obuhvaćene grupe, na način da koristimo naredbe `?:` nakon otvorene zagrade: `(?: uzorak)`. Tako će se izraz `/(?:some|a few) (people|cats) like some \1/` poklapati sa stringom *some cats like some cats*, ali ne i sa stringom *some cats like some a few*.

ELIZA također koristi regularne izraze na ovaj način. Kada dobije tekst koji treba obraditi, zamjenjuje riječ *my* riječju *your*, riječ *I'm* s *you are*, itd. Na ovaj način „čita" kako se korisnik osjeća te odgovara u skladu s time (primjerice `s/.* I'm (depressed | sad) .*/I am sorry to hear you are \1./`)

2.3 Što su riječi?

Kada smo se prvi puta susreli s formalnom definicijom riječi, naučili smo da je riječ glas ili skup glasova koji ima svoje značenje. Riječ predstavlja autonomnu jedinicu jezika te se sastoji od korijena te prefiksa ili sufiksa. Promotrimo li rečenicu *The festival is very popular; people from all over the world visit each year.* i prebrojimo li riječi u njoj, većina nas reći će da ih ima 14. No, i interpunkcijski znakovi (; i .) mogu se promatrati kao riječi, pa bi tada prethodna rečenica imala 16 riječi. Stoga je potrebno odlučiti što se računa kao riječ pri obradi prirodnog jezika.

Nadalje, pri razgovoru, često zastajemo, preusmjeravamo tok misli, pa računalo može dobiti na obradu rečenicu poput *I do uh main- mainly business data processing.* Fragment *main-* i pomoćni glas *uh* se izbacuju iz rečenica (teksta) primjerice pri prevođenju, no ponekad bismo željeli zadržati ove disfluentnosti. Tako riječi poput *uh* ili *um* mogu biti korisne pri prepoznavanju govora u predviđanju iduće riječi jer mogu označavati resetiranje ideje, pa ih stoga u toj situaciji tretiramo kao regularne riječi. Također, različiti ljudi koriste drugačije disfluentnosti. U tom smislu, one se mogu koristiti pri identificiranju govornika.
Kapitalizacija slova također može igrati veliku ulogu (*they, They*) u označavanju imenskih entiteta (*named-entity tagging*).

Stoga ćemo uvesti nekoliko pojmova koji će nam pomoći pri obradi prirodnog jezika.

- Leksem je skup leksičkih formi koje imaju isti korijen (*stem*), istu vrstu riječi (*part-of-speech*) te isto značenje, a lema je kanonski oblik leksema.
- Oblik riječi (*wordform*) je deklinirani/konjugirani ili izvedeni oblik riječi.
- Tipovi predstavljaju broj različitih riječi u korpusu. Ako je skup riječi u rječniku označen s V, tada je broj tipova upravo $|V|$.
- Tokeni (pojavnice) su ukupan broj N riječi teksta.

Ignorirajući interpunkciju, sljedeća rečenica ima 16 tokena i 14 tipova (*the* se pojavljuje 3 puta):

They picnicked by the pool, then lay back on the grass and looked at the stars.

Veza između broja tipova i broja tokena naziva se Herdanov ili Heapsov zakon:

$$|V| = kN^{\beta}, \quad \text{gdje su } k,\ 0 \leq \beta \leq 1 \text{ pozitivne konstante.}$$

Općenito, kada govorimo o broju riječi u nekom jeziku, mislimo na tipove riječi. No, broj riječi nekog jezika možemo promatrati i kao broj leksema ili lema. U tome nam mogu pomoći rječnici (podebljani unosi u rječnicima su gruba gornja granica na broj lema).

2.4 Normalizacija teksta

Prije nego započnemo obradu teksta, potrebno ga je normalizirati. Svaki proces normalizacije obuhvaća sljedeće zadatke:

1. Segmentacija/tokenizacija riječi u tekstu
2. Normalizacija formata riječi
3. Segmentacija rečenica u tekstu.

Detaljnije ćemo objasniti ove zadatke te na kraju opisati kako možemo mjeriti sličnost ili različitost dva stringa (riječi).

Tokenizacija i normalizacija riječi

Kod tokenizacije radimo segmentaciju (dijeljenje) teksta u riječi, dok kod normalizacije stavljamo dobivene riječi u standardni format.

Kao što smo ranije spomenuli, često ćemo interpunkcijske znakove promatrati kao posebne tokene, zato što točke razdvajaju rečenice, a zarezi mogu biti korisna informacija parserima. S druge strane, točku ne želimo promatrati kao poseban token u situacijama poput *Ph.D.*, *m.p.h.* i slično. Posebne znakove koji označavaju valutu ili su dio formata datuma također nećemo htjeti odvajati u poseban token. Potrebno je pripaziti i na slučajeve kada su točka i zarez dio broja (123,456.99) u engleskom jeziku, dok neki drugi jezici koriste razmake i zareze.
Tokenizacijom također možemo proširiti neke enklitičke skraćenice koje koriste apostrof (*what're = what are, you're = you are*, itd.). Tokenizacija također može promatrati izraze kao *New York* ili *rock 'n' roll* kao jedan token. Ovdje vidimo poveznicu s detekcijom imenskih entiteta (*named entity detection*), odnosno detekcijom imena, datuma i organizacija.

Jedan od često korištenih tokenizacijskih standarda je Penn Treebank tokenizacijski standard, korišten za parsirane zbirke tekstova (*treebanks*) izdane od strane LDC-a (Linguistic Data Consortium). Dat ćemo jedan primjer rada navedenog standarda:

Ulaz: „The San Francisco-based restaurant," they said, „doesn't charge $10".

Izlaz:

„	The	San	Francisco-based	restaurant	,	"	they	said	,	„	
does	n't	charge	$	10	"	.					

Tokeni mogu biti normalizirani, pri čemu se odabire jedna normalizirana forma za riječi koje imaju više od jedne forme (*USA* i *US*).
Za povrat informacija (*information retrieval*), htjet ćemo da se upit za *US* poklapa s dokumentom koji sadrži riječ *USA* (primjerice, kada koristimo internetsku tražilicu).

Još jedna vrsta normalizacije je promjena veličine slova (*case folding*), a predstavlja mapiranje stringova koje želimo uspoređivati (odnosno obrađivati) u *uppercase* (svi znakovi stringa su velika slova) ili *lowercase* (svi znakovi stringa su mala slova) prije same obrade.
Kod zadataka poput prepoznavanja govora i povrata informacija, sve se mapira u *lowercase*.

U nekim drugim slučajevima, kao primjerice analizi sentimenata (osjećaja), izvlačenja (ekstrakcije) informacija (*information extraction*) te strojnog prevođenja ovakva obrada teksta se ne provodi jer su velika i mala slova od velike pomoći (primjerice kod razlike zamjenice *us* i imenice *US* koja predstavlja zemlju).

Treba spomenuti kako je vrlo bitno da proces tokenizacije bude odrađen vrlo brzo jer se on provodi prije bilo kakve druge obrade prirodnog jezika. Standardna metoda za tokenizaciju/normalizaciju je korištenje deterministčkih algoritama koji se temelje na regularnim izrazima kompiliranim u vrlo efikasne konačne automate (vidi [10], poglavlje 1.3).

Još ćemo spomenuti kako treba obratiti veliku pozornost na jezik koji se obrađuje. Obrađujemo li kineski ili japanski jezik, ne možemo koristiti razmak kao znak koji odvaja dvije riječi, nego trebamo posegnuti za nekom alternativnom segmentacijskom metodom. Jednostavan algoritam koji je učinkovit kod segmentacije kineskog jezika je MaxMatch algoritam naveden dolje o kojemu se može više pronaći u [4].

Algoritam 1 MaxMatch

Ulaz: rečenica *rec* i rječnik *dict*
Izlaz: lista riječi
  if rečenica *rec* prazna **then**
    return prazna lista
  end if
  for $i \leftarrow length(rec)$ **downto** 1 **do**
    $prva\_rijec \leftarrow$ prvih i znakova rečenice *rec*
    $ostatak \leftarrow$ ostatak rečenice
    if uRjecniku($prva\_rijec$, *dict*) **then**
      return list($prva\_rijec$, MaxMatch($ostatak$, *dict*))
    end if
  end for
  $prva\_rijec \leftarrow$ prvi znak rečenice *rec*
  $ostatak \leftarrow$ ostatak rečenice
  return list($prva\_rijec$, MaxMatch($ostatak$, *dict*))

Jedan od razloga zašto algoritam radi bolje s kineskim riječima nego engleskim je zato što su kineske riječi dosta kraće od engleskih. Najtočniji algoritmi za segmentaciju u kineskom jeziku obično koriste statističke sekvencijalne modele trenirane uz nadzirano strojno učenje na ručno segmentiranim skupovima za treniranje.

Lematizacija i korjenovanje

Ranije smo spomenuli kako riječi dolaze u različitim oblicima u tekstu. Tako se neki glagoli poput *be* mogu pojaviti u tekstu u oblicima *am, are, is*. Ne želimo posebno spremati svaki oblik, nego ih želimo promatrati kao jedan glagol. U tu svrhu se koriste lematizacija (još se naziva i lematiziranje) i korjenovanje.

Lematizacija je određivanje kanonskog oblika riječi (kao u navedenom primjeru *be*), odnosno leme. S druge strane, korjenovanje je postupak uklanjanja afiksa s kraja riječi.
Lematizacija koristi potpuno morfološko parsiranje riječi, u smislu da se morfemi odvoje (ako ih je više od jednog) te se pronalazi lema riječi, vrsta riječi, te ostale informacije poput lica i vremena (za glagole) i slično. Algoritmi za lematizaciju mogu biti kompleksni, pa se stoga ponekad ipak koristi korjenovanje, koje je jednostavnije, ali i sirovije.
Jedan od najkorištenijih algoritama za korjenovanje je Porterov koji se sastoji od 5 faza reduciranja riječi koje se primjenjuju sekvencijalno (u nizu). Neka od pravila koja se koriste u tim fazama su:

Pravilo	**Primjer**
SSES → SS	caresses → caress
IES → I	ponies → poni
S →	dogs → dog
EED → EE	agreed → agree
FULNESS → FUL	hopefulness → hopeful

Detaljan opis svake faze može se pronaći u [9].
Naravno, postoje i drugi algoritmi za korjenovanje, poput Lovinsovog (vidi [6]) te Paiceovog (vidi [8]).

Korjenovanje i lematizacija imaju još jednu korisnu posljedicu, a to je da pomažu pri rješavanju problema nepoznatih riječi, onih koje sustav do sada nije vidio. Nepoznate riječi su od velike važnosti za sustave strojnog učenja. BPE algoritam (*byte-pair encoding*) može se koristiti za tokenizaciju te tako spojiti korijen riječi s određenim čestim dijelovima riječi. Algoritam je dostupan u [1], kao i opširniji opis rada algoritma.

Segmentacija rečenica

Segmentacija rečenica je još jedan važan korak u obradi teksta. Najčešće rečenice odvajamo interpunkcijskim znakovima poput točke, upitnika te uskličnika. No, točka može imati i drugo značenje u rečenici. Ona se također koristi za skraćenice poput *Mr.* i *Inc.* Također, kao u prošloj rečenici, točka može imati i ulogu dijela skraćenice i kraja rečenice. Općenito, metode za ovaj postupak rade tako da se napravi binarni klasifikator (temeljen na nizu pravila ili strojnom učenju) koji će odlučiti je li točka dio riječi ili označava kraj rečenice. Korisno je imati i rječnik skraćenica.

2.5 Mjerenje sličnosti riječi

Pri korištenju alata za pisanje dokumenata (kao što je Microsoft Word), često možemo pokrenuti ispravljanje pravopisnih grešaka kako bismo vidjeli jesmo li u dokumentu pogrešno napisali jednu ili više riječi. Spomenuti postupak pronalazi nepoznatu riječ (primjerice *graffe*) te koristi pretpostavku da je korisnik imao na umu riječ koja joj je slična (primjerice *giraffe*). No, kako znati koje riječi su više, a koje manje slične?

Udaljenost uređivanja (*edit distance*) predstavlja način određivanja sličnosti, odnosno različitosti dva stringa. Točnije, minimalna udaljenost uređivanja (*minimum edit distance*) dva stringa definira se kao najmanji broj operacija uređivanja (ubacivanje, brisanje, zamjena) potrebnih da se jedan string transformira u drugi.

Radi lakšeg uočavanja operacija koje se trebaju izvršiti, stringove poravnavamo tako da se što veći broj znakova podudara. Nakon toga zapišemo koja operacija (ili više njih) se treba izvesti kako bismo od gornjeg stringa dobili donji (d-brisanje, s-zamjena, i-ubacivanje). Možemo smatrati da u slučaju istog znaka u izvornom i ciljnom stringu izvršavamo operaciju zamjene znaka samim sobom, čija je cijena nula. Stoga taj slučaj nećemo zapisivati.

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Slika 1: Minimalna udaljenost uređivanja stringova *intention* i *execution*

Na *Slici 1* možemo vidjeti da minimalna udaljenost uređivanja stringova *intention* i *execution* iznosi 5. Imamo jednu operaciju brisanja (*delete*), tri operacije zamjene (*substitute*) te jednu operaciju ubacivanja (*insert*).

Svakoj od ovih operacija moguće je dodati težinu. Levenshtein udaljenost dva niza je udaljenost uređivanja koja svakoj operaciji dodjeljuje težinu 1. Zamjena slova samim sobom ima težinu 0.

U primjeru na *Slici 1* Levenshtein udaljenost između stringova *intention* i *execution* je 5.

Postoji i alternativna verzija ove metrike u kojoj zamjene nisu dopuštene (odnosno težina zamjene iznosi 2 jer se svaka zamjena može prikazati kao jedno brisanje i jedno ubacivanje).

Algoritam

Problem traženja minimalne udaljenosti uređivanja možemo predstaviti kao problem traženja najkraćeg puta (niza uređivanja) od jednog stringa do drugoga. Prostor svih mogućih uređivanja je iznimno velik, stoga ne možemo pretraživati naivno. No, mnogo različitih putova završavat će u istom stringu, pa možemo pamtiti najkraći put do svakoga stanja. Dinamičko programiranje može pomoći u rješavanju ovoga problema.

Pretpostavimo da su dana dva stringa, izvorni string X duljine n i ciljni string Y duljine m. Udaljenost uređivanja $D(i, j)$ definirat ćemo kao udaljenost između $X[1..i]$, prvih i znakova od X, i $Y[1..j]$, prvih j znakova od Y. Tada je udaljenost uređivanja između X i Y jednaka $D(n, m)$.

U baznim slučajevima, kada imamo izvorni podstring duljine i i prazan ciljni string, odnosno prazan izvorni string i ciljni podstring duljine j, potrebno nam je i operacija brisanja, odnosno j operacija ubacivanja. Stoga $D(i,j)$ računamo po sljedećoj rekurzivnoj formuli:

$$D(i,j) = \begin{cases} \max(i,j), & \min(i,j) = 0, \\ \min \begin{cases} D(i-1,j) + del(X[i]) \\ D(i,j-1) + ins(Y[j]) \\ D(i-1,j-1) + sub(X[i], Y[j]) \end{cases} & , \quad \text{inače} \end{cases} \tag{2.1}$$

gdje funkcija *del* računa cijenu brisanja i-tog znaka stringa X, funkcija *ins* računa cijenu ubacivanja j-tog znaka stringa Y, a funkcija *sub* računa cijenu zamjene i-tog znaka stringa X s j-tim znakom stringa Y.

Implementacija algoritma koji udaljenost računa na ovaj način dana je u Algoritmu 2. Koristimo li drugu navedenu verziju Levenshteinove udaljenosti, dobivamo formulu (2.2).

$$D(i,j) = \begin{cases} \max(i,j), & \min(i,j) = 0, \\ \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 0, & X[i] = Y[j] \\ 2, & X[i] \neq Y[j] \end{cases} \end{cases} & , \quad \text{inače} \end{cases} \tag{2.2}$$

Algoritam 2 MIN-EDIT-DISTANCE

Ulaz: stringovi X i Y
Izlaz: minimalna udaljenost uređivanja između stringova X i Y

```
n ← LENGTH(X)
m ← LENGTH(Y)
Stvori matricu udaljenosti D veličine (n + 1) × (m + 1)
D[0, 0] ← 0
for i ← 1 to n do
    D[i, 0] ← D[i − 1, 0] + del(X[i])
end for
for j ← 1 to m do
    D[0, j] ← D[0, j − 1] + ins(Y[j])
end for
for i ← 1 to n do
    for j ← 1 to m do
        D[i, j] ← MIN(D[i − 1, j] + del(X[i]), D[i, j − 1] + ins(Y[j]),
                      D[i − 1, j − 1] + sub(X[i], Y[j]))
    end for
end for
return D[n, m]
```

Primjer

U *Tablici 6* prikazan je rad Algoritma 2 gdje koristimo formulu (2.2). Dani su stringovi *intention* i *execution*.

	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	↑ 1	2	3	4	5	6	7	6	7	8
n	2	↖←↑ 3	4	5	6	7	8	7	8	7
t	3	4	↖←↑ 5	6	7	8	7	8	9	8
e	4	3	4	↖← 5	← 6	7	8	9	10	9
n	5	4	5	6	7	↖←↑ 8	9	10	11	10
t	6	5	6	7	8	9	↖ 8	9	10	11
i	7	6	7	8	9	10	9	↖ 8	9	10
o	8	7	8	9	10	11	10	9	↖ 8	9
n	9	8	9	10	11	12	11	10	9	↖ 8

Tablica 6: Izračun minimalne udaljenosti uređivanja stringova *intention* i *execution* pomoću Algoritma 2. Cijena ubacivanja i brisanja iznosi 1, a zamjene 2. Vraćajući se unatrag i prateći odakle je došla trenutna vrijednost, osjenčali smo jedno moguće poravnavanje najmanje cijene.

Najmanja udaljenost uređivanja ovih dvaju stringova nalazi se u tablici dolje desno i iznosi 8. Radi jednostavnosti, nismo ucrtavali sve strelice koje pokazuju odakle smo došli u trenutnu ćeliju, nego smo prikazali one strelice koje pripadaju jednom mogućem rješenju (osjenčane ćelije).

Dakle, krećemo od donje desne ćelije i sjenčamo onu ćeliju koja je dio izraza pomoću kojeg smo dobili rezultat trenutne ćelije (minimalna od 3 opcije). Ponekad je moguće odabrati više od jednog smjera, pa proizvoljno odaberemo jedan od mogućih.

Poravnavanje minimalne cijene zatim iščitavamo iz osjenčanih ćelija tako da krenemo od one s vrijednosti 0. Ako su dvije uzastopne promatrane ćelije u istom stupcu, riječ je o operaciji brisanja, a ako su u istom retku, o operaciji ubacivanja. Ćelije koje se dijagonalno nastavljaju jedna na drugu mogu imati iste ili različite vrijednosti (u našem slučaju se razlikuju za 0 ili 2). Ćelije s istim vrijednostima označavaju da se slova podudaraju (zamjena slova samim sobom je cijene 0 i ne zapisujemo ju), a one s različitim vrijednostima upućuju na operaciju zamjene.

Stoga možemo zaključiti kako se operacije dobivene iz tablice podudaraju s operacijama zapisanima na *Slici 1.*

2.6 N-grami

Pri druženju s ljudima koji su nam bliski (a ponekad i onima s kojima nismo toliko bliski), možemo doživjeti da završavamo jedni drugima rečenice ili „znamo" što će druga osoba reći. Ljudi su jako dobri u predviđanju, ne samo konteksta, nego točnih riječi i fraza koje druga osoba planira izreći.

U ovome poglavlju bavit ćemo se modelima koji će davati vjerojatnosti riječima koje slijede trenutni niz riječi. Time mogu dati vjerojatnost i cijeloj rečenici. Primjerice, veću vjerojatnost dodijelit će rečenici

I think there's a good chance that people will live on the moon in the future

nego rečenici koja se sastoji od istih riječi, ali u drugačijem poretku, kao

live in chance moon good will that on the I a people future there's the think.

Modeli koji dodjeljuju vjerojatnosti nizovima riječi zovu se jezični modeli (*language models*). Predstavit ćemo najjednostavniji takav model koji dodjeljuje vjerojatnosti rečenicama i nizovima riječi, odnosno N-gramima.
Niz od N riječi nazivamo N-gram; 2-gram (bigram) je niz od 2 riječi kao *my bike*, 3-gram (trigram) je niz od 3 riječi poput *me and you* itd. Posljednju riječ N-grama predviđat ćemo pomoću prethodnih $N-1$ riječi. Ponekad se izraz N-gram poistovjećuje s prediktivnim modelom koji mu dodjeljuje vjerojatnost.

Stoga se pitamo kako ćemo izračunati vjerojatnost riječi w ako znamo neku povijest h, odnosno niz prethodnih riječi. Ovu uvjetnu vjerojatnost označit ćemo s $P(w \mid h)$, a radi lakšeg razumijevanja, analizu možemo provesti na primjeru.

Neka je dana povijest *its water is so transparent that* i želimo znati kolika je vjerojatnost da je iduća riječ *the*, odnosno zanima nas $P(the \mid its\ water\ is\ so\ transparent\ that)$. Jedan način računanja ove vjerojatnosti je koristeći relativne frekvencije: neka je dan dovoljno velik korpus, te funkcija C koja broji koliko puta se dani string pojavio u tom korpusu (odnosno računa frekvenciju stringa). Tada možemo pisati:

$$P(the \mid its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)},$$

odnosno prebrojimo slučajeve u kojima riječ *the* slijedi string *its water is so transparent that* i podijelimo dobivenu vrijednost s ukupnim brojem svih pojavljivanja stringa *its water is so transparent that*.

Ova metoda funkcionira dobro u većini slučajeva, no nove rečenice se stvaraju svakodnevno, što znači da čak ni internet nije dovoljno velik za dobre procjene. To možemo vidjeti napravimo li jednostavno proširenje rečenice u primjeru, *Walden Pond's water is so transparent that the*, koje može imati frekvenciju 0.

Stoga nam je potreban pametniji način (ili više njih) za procjenu vjerojatnosti $P(w \mid h)$, ili za procjenu cijelog niza riječi W.

Neka je X_i slučajna varijabla. Pojednostavit ćemo zapise na sljedeći način:

- niz od n riječi označavat ćemo s $w_1...w_n$ ili w_1^n,
- $P(the) \coloneqq P(X_i = the)$,
- $P(w_1, w_2, w_3, ...) \coloneqq P(X_1 = w_1, X_2 = w_2, X_3 = w_3, ...)$.

Pri računanju ćemo koristiti lančano pravilo:

$$P(X_1, ..., X_n) = P(X_1)\, P(X_2 \mid X_1)\, P\left(X_3 \mid X_1^2\right) \dots P\left(X_n \mid X_1^{n-1}\right) = \prod_{i=1}^{n} P\left(X_i \mid X_1^{i-1}\right),$$

odnosno

$$P(w_1^n) = P(w_1)\, P(w_2 \mid w_1)\, P\left(w_3 \mid w_1^2\right) \dots P\left(w_n \mid w_1^{n-1}\right) = P(w_1) \prod_{i=2}^{n} P\left(w_i \mid w_1^{i-1}\right), \quad (2.3)$$

što pokazuje vezu između računanja zajedničke vjerojatnosti niza i računanja uvjetne vjerojatnosti riječi ako su dane prethodne riječi. No, ne znamo izračunati točnu vjerojatnost riječi uz dani niz prethodnih riječi, $P\left(w_n \mid w_1^{n-1}\right)$. Intuicija modela N-grama je aproksimirati povijest preko nekoliko posljednjih riječi umjesto računanja vjerojatnosti riječi uz danu cijelu povijest.

Model bigrama aproksimira vjerojatnost riječi uz dane sve prethodne riječi $P\left(w_n \mid w_1^{n-1}\right)$ koristeći samo uvjetnu vjerojatnost prethodne riječi, $P(w_n \mid w_{n-1})$, odnosno

$$P\left(w_n \mid w_1^{n-1}\right) \approx P(w_n \mid w_{n-1}). \quad (2.4)$$

Ova pretpostavka da vjerojatnost riječi ovisi samo o prethodnoj riječi zove se Markovljeva pretpostavka (umjesto svih prethodnih riječi, promatramo ih manje). Markovljevi modeli stoga pretpostavljaju da možemo predvidjeti vjerojatnost neke buduće jedinice bez da gledamo predaleko u prošlost.
Generalizacijom modela bigrama možemo dobiti model N-grama koji će gledati $N-1$ riječi u prošlost. Za model N-grama i niz riječi w_1^n tada dobivamo

$$P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-(N-1)}^{n-1}\right),$$

gdje $w_{n-(N-1)}^{n-1}$ predstavlja posljednjih $N-1$ riječi niza w_1^{n-1}.

Sada, koristeći pretpostavku modela bigrama, možemo pretpostavku (2.4) ubaciti u jednadžbu (2.3) kako bismo dobili

$$P(w_1^n) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-1}).$$

No kako možemo procijeniti vjerojatnosti bigrama (ili N-grama)?

Najjednostavniji i ujedno najintuitivniji način je koristiti procjenitelj metodom maksimalne vjerodostojnosti (*Maximum Likelihood Estimation* - MLE). MLE procjenu za parametre modela N-grama dobijemo normalizacijom frekvencije iz korpusa.

Primjerice, kako bismo izračunali vjerojatnost riječi w_n modelom bigrama ako je dana prethodna riječ w_{n-1}, izračunat ćemo frekvenciju bigrama koji se sastoji od tih dviju riječi i normalizirati dobivenu vrijednost sumom frekvencija svih bigrama koji sadrže istu prvu riječ w_{n-1}:

$$P\left(w_n \mid w_{n-1}\right) = \frac{C(w_{n-1}w_n)}{\sum_w C\left(w_{n-1}w\right)}.$$

Ovu jednadžbu možemo pojednostaviti tako da, umjesto sume frekvencija svih bigrama koji počinju s w_{n-1}, jednostavno prebrojimo pojavljivanja riječi w_{n-1}:

$$P\left(w_n \mid w_{n-1}\right) = \frac{C\left(w_{n-1}w_n\right)}{C\left(w_{n-1}\right)}.$$

Promotrimo kako ovo funkcionira na primjeru.

Svaku rečenicu korpusa najprije modificiramo tako da na početak dodamo simbol `<s>`, a na kraj simbol `</s>`. Time smo označili početak i kraj svake rečenice (te znamo i kontekst bigrama za prvu riječ).

Pretpostavimo da nam je dan korpus od tri rečenice te ih modificiramo kao što je opisano.

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

Zatim, preko gornjih formula izračunamo neke vjerojatnosti bigrama.

$$\begin{array}{lll} P(\texttt{I}|\texttt{<s>}) = \frac{2}{3} = 0.67 & P(\texttt{Sam}|\texttt{<s>}) = \frac{1}{3} = 0.33 & P(\texttt{am}|\texttt{I}) = \frac{2}{3} = 0.67 \\ P(\texttt{</s>}|\texttt{Sam}) = \frac{1}{2} = 0.5 & P(\texttt{Sam}|\texttt{am}) = \frac{1}{2} = 0.5 & P(\texttt{do}|\texttt{I}) = \frac{1}{3} = 0.33 \end{array}$$

Za općeniti slučaj MLE procjene za parametre modela N-grama imat ćemo

$$P\left(w_n|w_{n-N+1}^{n-1}\right) = \frac{C\left(w_{n-N+1}^{n-1}w_n\right)}{C\left(w_{n-N+1}^{n-1}\right)}. \tag{2.5}$$

Primijetimo, ako bismo htjeli koristiti modele trigrama, rečenice bismo modificirali s dva simbola `<s>` na početku rečenice (zbog vjerojatnosti prve riječi). Analogno radimo za bilo koji model N-grama.

2.6.1 Procjena jezičnih modela

Model N-grama je dobar primjer statističkih modela koji se koriste u obradi govora i prirodnog jezika. Općenito, parametre modela treniramo na skupu podataka te ih primijenimo na novi skup podataka i analiziramo rezultate.

Stoga je dani korpus potrebno podijeliti u dva skupa, skup za treniranje i skup za testiranje (često se naziva i *hold-out set*). Ako želimo usporediti dva modela različite arhitekture, trenirat ćemo ih na istom skupu podataka za treniranje i tada ih primijeniti na isti skup za testiranje te usporediti rezultate. Zaključit ćemo da je bolji onaj model koji ima veći postotak, odnosno točnije predviđa testni skup podataka.

Korpus možemo podijeliti i na 3 dijela, tako da uvedemo skup za validaciju/provjeru (*development, validation, hold-on set*) pomoću kojega podešavamo hiperparametre modela. U praksi se korpus podijeli na skup za treniranje (80%), skup za validaciju (10%) te skup za testiranje (10%). Ovi omjeri mogu biti i 60-20-20.

Postoje dvije mogućnosti koje se mogu dogoditi vezano za riječi koje se pojavljuju u skupu za testiranje.
U sustavu zatvorenog rječnika (leksikona) skup za testiranje može sadržavati samo one riječi koje su dio zadanog rječnika. Stoga nećemo imati nepoznatih riječi.
S druge strane, u skupu za testiranje mogu se pojaviti riječi koje do sada nismo vidjeli. U sustavu otvorenog rječnika skup za testiranje modificiramo tako da nepoznatim riječima dodamo pseudo-riječ `<UNK>`. Drugi naziv za ove riječi je „riječi izvan rječnika" (*out of vocabulary words* - OOV), a njihov postotak u skupu za testiranje zove se OOV stopa.

Svaki jezični model želimo procijeniti, odnosno ocijeniti njegov rad. No, kako određene procjene mogu dugo trajati i biti skupe, htjeli bismo imati metriku kojom možemo brzo procijeniti model i zaključiti trebamo li uvesti poboljšanja.

U praksi koristimo zbunjenost (*perplexity*) kao metriku za procjenjivanje jezičnih modela. Označavamo ju s PP i ona predstavlja inverznu vjerojatnost skupa za testiranje normaliziranu brojem riječi.
Neka je dan testni skup $W = w_1 w_2 ... w_N$. Tada je zbunjenost skupa W

$$PP(W) = P\left(w_1 w_2 ... w_N\right)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P\left(w_1 w_2 ... w_N\right)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P\left(w_i | w_1 w_2 ... w_{i-1}\right)}}. \tag{2.6}$$

Koristimo li model bigrama, zbunjenost od W je

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P\left(w_i | w_{i-1}\right)}}.$$

Veća uvjetna vjerojatnost niza riječi znači da će zbunjenost biti manja. Stoga, umjesto maksimiziranja vjerojatnosti skupa za testiranje možemo minimizirati zbunjenost. Kako ćemo u skupu za testiranje imati cijele rečenice koje ćemo ubacivati u gornje dvije jednadžbe, moramo modificirati rečenice sa simbolima za početak i kraj (ranije spomenutima). Ne smijemo zaboraviti uključiti sve simbole za kraj rečenice u ukupan broj tokena.

Zbunjenost možemo shvatiti i kao prosječni težinski faktor grananja jezika. Faktor grananja jezika je broj mogućih sljedećih riječi koje mogu doći iza bilo koje riječi.
Pretpostavimo da trebamo prepoznati znamenke u engleskom jeziku (*zero, one, two, ..., nine*). Svaka od znamenki ima jednaku vjerojatnost pojavljivanja $\left(\frac{1}{10}\right)$. Izračunajmo zbunjenost ovog mini-jezika.
Neka je W string znamenki duljine N. Koristeći jednadžbu (2.6), imamo:

$$PP(W) = P\left(w_1 w_2 ... w_N\right)^{-\frac{1}{N}} = \left(\left(\frac{1}{10}\right)^N\right)^{-\frac{1}{N}} = \left(\frac{1}{10}\right)^{-1} = 10.$$

Sada pretpostavimo kako je jedna znamenka 10 puta frekventnija od ostalih i BSO neka je to nula (*zero*). To znači da ćemo u dugačkom stringu duljine N u prosjeku, odaberemo li 19 znamenki, imati 10 nula i od ostalih po jednu znamenku. Stoga podijelimo string na blokove duljine 19 i pretpostavimo da ih ima M. Dobivamo manju zbunjenost uz isti faktor grananja:

$$P\left(w_1 w_2 ... w_N\right)^{-\frac{1}{N}} = \left(\left(\frac{10}{19}\right)^{10M}\left(\frac{1}{19}\right)^{9M}\right)^{-\frac{1}{19M}} = \left(\left(\frac{10}{19}\right)^{10}\left(\frac{1}{19}\right)^{9}\right)^{-\frac{1}{19}} = 5.655.$$

2.6.2 Zaglađivanje

Ranije smo spomenuli pojam nepoznatih riječi. No, što je s onim riječima koje se nalaze u rječniku, ali se pojavljuju u skupu za testiranje u nekom do sada neviđenom kontekstu? Jezični model bi takvim događajima mogao dodijeliti vjerojatnost jednaku nuli, a to ne želimo. Stoga, trebamo „prebaciti" malo vjerojatnosti s frekventnijih događaja na događaje koje još nismo vidjeli. Takva modifikacija zove se zaglađivanje (*smoothing*), odnosno popuštanje (*discounting*). Ukratko ćemo opisati najjednostavniji način na koji možemo provesti ovaj postupak, a druge metode mogu se pronaći u [4].

Laplaceovo zaglađivanje

Najjednostavniji način zaglađivanja je da dodamo jedinicu svim frekvencijama bigrama, prije normalizacije u vjerojatnosti. Dakle, sve frekvencije koje su bile nula, sada će biti 1 itd. Takav postupak zaglađivanja naziva se Laplaceovo zaglađivanje.
Primijenimo li Laplaceovo zaglađivanje na vjerojatnosti unigrama (niz od jedne riječi), imat ćemo:

$$P(w_i) = \frac{c_i}{N} \quad \Longrightarrow \quad P_L(w_i) = \frac{c_i + 1}{N + V},$$

gdje je c_i broj pojavljivanja riječi w_i, N je broj tokena, a broj riječi u rječniku je V.

Ovaj postupak možemo provesti i za brojeve veće od jedan (dodajemo više od jednog događaja), kao i za brojeve između nule i jedinice. Tada u brojniku zbrojimo c_i s k umjesto s 1, a u nazivniku veličinu rječnika V pomnožimo s k, gdje je k proizvoljan pozitivan broj.

2.6.3 *Backoff* i interpolacija

Vidjeli smo kako možemo riješiti problem s bigramima koji imaju frekvenciju jednaku nula, no što ako želimo izračunati $P(w_n \mid w_{n-2}w_{n-1})$, a nemamo primjer trigrama $w_{n-2}w_{n-1}w_n$?

Tada možemo iskoristiti $P(w_n \mid w_{n-1})$, odnosno $P(w_n)$. Ovakav način „hijerarhije" naziva se *backoff*. Koristimo trigram ako imamo dovoljno dokaza, u suprotnome koristimo bigrame, odnosno unigrame. Pri tome je potrebno koristiti zaglađivanje kako vjerojatnosti u sumi ne bi bile veće od jedan. Jedan od modela *backoff*-a koji funkcionira na ovaj način (koji zahtijeva minimalno k pojavljivanja, a ne samo jedno), zove se Katz *backoff* i često se kombinira s Good-Turing zaglađivanjem.

S druge strane, kod interpolacije uvijek kombiniramo broj trigrama, bigrama i unigrama, i to uz težine.
Primjer jednostavne linearne interpolacije je:

$$\hat{P}(w_n \mid w_{n-2}w_{n-1}) = \lambda_1 P(w_n \mid w_{n-2}w_{n-1}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n), \quad \sum_i \lambda_i = 1.$$

Težine λ_i možemo računati i iz konteksta, no kako ćemo ih postaviti?
Spomenuli smo ranije kako korpus možemo podijeliti na 3 dijela, tako da imamo i skup za validaciju. Upravo će taj skup pomoći u treniranju težina. Tada će linearna interpolacija izgledati ovako:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1})P(w_n).$$

I interpolacija i *backoff* zahtijevaju zaglađivanje/popuštanje pomoću algoritama kao što su Kneser-Ney, Witten-Bell ili Good-Turing.

2.7 Gramatičko tagiranje

Vrste riječi su korisne zato što otkrivaju mnogo o samoj riječi i riječima koje su joj susjedne. Vidjet ćemo kako možemo riječima dodijeliti vrstu riječi, a ime tog postupka je gramatičko tagiranje (*part-of-speech tagging*).

Vrste riječi možemo podijeliti u dvije velike nadkategorije, zatvorene i otvorene.
Zatvorene vrste riječi čine one jedinice (vrste riječi) koje imaju konačan broj članova i kod kojih je dodavanje novih članova vrlo malo vjerojatno, kao primjerice prijedlozi i zamjenice. Otvorene vrste riječi nemaju određen broj članova, naprotiv, one su podložne stvaranju (ili posuđivanju) novih riječi, kao imenice i glagoli.

Engleski jezik ima 4 velike otvorene vrste riječi: imenice, glagole, pridjeve i priloge. Imenice možemo dalje podijeliti na vlastite i opće. Priloge možemo podijeliti na vremenske, mjesne, načinske i stupanjske (*degree*).
Neke važne zatvorene vrste riječi u engleskom jeziku su prijedlozi, čestice, determineri, veznici, zamjenice, pomoćni glagoli i brojevi. Za definicije navedenih pojmova (vrsta riječi) u engleskom jeziku vidi [15].

U engleskom jeziku postoje i riječi s drugačijim funkcijama, kao uzvici, negacije, markeri uljudnosti (*please, thank you*), pozdravi te egzistencijalni *there*. Ponekad ove vrste promatramo posebno, a ponekad ih pridružimo uzvicima ili prilozima, ovisno o svrsi pridruživanja.

Kao što smo spomenuli, želimo riječima ulaznog teksta pridružiti vrste riječi. Dakle, algoritam prima niz riječi i skup oznaka (*tagset*), a izlaz algoritma je niz oznaka, jedna po tokenu.

No, riječi mogu biti dvosmislene, a cilj algoritma je pridruživanje točne oznake svakoj riječi. Neke od najčešćih dvosmislenih riječi engleskog jezika su *that, back, down, put, set*. Najjednostavniji način pridruživanja oznake dvosmislenoj riječi zove se Osnova najfrekventije klase (*Most Frequent Class Baseline*), a radi se o sljedećem: svaki token pridruži onoj vrsti riječi (oznaci) u kojoj se pojavio najveći broj puta u skupu za treniranje. Standardan način ocjenjivanja izvršenja ovakvih algoritama je točnost, odnosno postotak točno označenih riječi.

Sekvencijalni model ili sekvencijalni klasifikator radi tako da dodjeljuje oznaku (vrstu) svakoj jedinici niza (sekvence), tako mapirajući sekvencu riječi u sekvencu oznaka. Postoje dva uobičajena pristupa modeliranju sekvenci: generativan pristup, skriven Markovljev model (*Hidden Markov Model* - HMM), i diskriminativan (onaj koji pravi razliku) pristup, Markovljev model maksimalne entropije (*Maximum Entropy Markov Model* - MEMM).

HMM je vjerojatnosni sekvencijalni model, on za dani niz jedinica (riječi, slova, morfemi, rečenice, itd.) računa vjerojatnosnu distribuciju mogućih nizova oznaka i odabire najbolji. Algoritam za dekodiranje za HMM je Viterbi algoritam koji koristi dinamičko programiranje. Postoji i varijanta Viterbi algoritma koja se zove *beam search* koja zadržava dio stanja, a ne sva stanja kako bi se smanjilo vrijeme izvršavanja.
S druge strane, MEMM trenira modele logističke regresije kako bi odabrao najbolju oznaku uz danu riječ i kontekst uz dosadašnje oznake, a zatim koristi Viterbi algoritam za odabir najboljeg niza oznaka.

Moderni označivači uobičajeno se pokreću u oba smjera, a ne samo u jednom.

3 Govor

Kada je riječ o govoru (ostvarenju jezika), grana jezikoslovlja koja ovdje igra veliku ulogu je fonetika (grčki *fon* znači glas). No, koja je razlika između fonologije i fonetike?

Fonologija proučava funkcionalna svojstva glasova, odnosno foneme. Bavi se ponašanjem govornih jedinica i jezičnom funkcijom. Fonem je najmanja jezična jedinica bez značenja, ali s razlikovnom funkcijom (izrazom).
Fonetika proučava glasove, odnosno fone. Bavi se akustičnim i artikulacijskim svojstvima glasova, bez obzira na njihovu funkciju. Fon je najmanja govorna jedinica, konkretan glas.
Alofoni su varijante fonema, drugačiji izgovori riječi koji ne mijenjaju značenje riječi. Kada su fonovi realizacija istog fonema, tada ih zovemo alofonima tog fonema.
Stoga, iako fonetika i fonologija proučavaju glasove, fonetika se bavi njihovim fizičkim, a fonologija kognitivnim aspektima.

ARPABET simbol	IPA simbol	Primjer riječi	ARPAbet transkripcija
[p]	[p]	parsley	[p aa r s l iy]
[t]	[t]	tea	[t iy]
[k]	[k]	cook	[k uh k]
[b]	[b]	bay	[b ey]
[d]	[d]	dill	[d ih l]
[g]	[g]	garlic	[g aa r l ix k]
[m]	[m]	mint	[m ih n t]
[n]	[n]	nutmeg	[n ah t m eh g]
[ng]	[ŋ]	baking	[b ey k ix ng]
[f]	[f]	flour	[f l aw axr]
[v]	[v]	clove	[k l ow v]
[th]	[θ]	thick	[th ih k]
[dh]	[ð]	those	[dh ow z]
[s]	[s]	soup	[s uw p]
[z]	[z]	eggs	[eh g z]
[sh]	[ʃ]	squash	[s k w aa sh]
[zh]	[ʒ]	ambrosia	[ae m b r ow zh ax]
[ch]	[tʃ]	cherry	[sh eh r iy]
[jh]	[dʒ]	jar	[jh aa r]
[l]	[l]	licorice	[l ih k axr ix sh]
[w]	[w]	kiwi	[k iy w iy]
[r]	[r]	rice	[r ay s]
[y]	[j]	yellow	[y eh l ow]
[h]	[h]	honey	[h ah n iy]
[q]	[ʔ]	uh-oh	[q ah q ow]
[dx]	[ɾ]	butter	[b ah dx axr]
[nx]	[ɾ̃]	winner	[w ih nx axr]
[el]	[l̩]	table	[t ey b el]

Tablica 7: Suglasnici: češće i rjeđe korišteni fonovi i alofoni

3.1 Fonetika

Fonetici ćemo pristupiti iz računarske perspektive. To znači da, osim navedenog, nas zanima i kako se glasovi (fonovi) mogu digitalizirati i obraditi.

Sustavi za prepoznavanje govora i pretvorbu teksta u govor moraju imati na raspolaganju izgovor svake riječi koju mogu prepoznati, odnosno izgovoriti. Izgovor riječi modelira se kao string simbola koji reprezentiraju fone ili segmente. Svaki fon reprezentiran je fonetskim simbolom koji sliči slovu engleske abecede (različiti jezici imaju različite skupove fonova).

3.1.1 Fonetska transkripcija

Postoje razne abecede kojima se opisuju fonovi. Jedna od njih je IPA (*International Phonetic Alphabet*). Ovaj standard razvijen je 1888. godine s ciljem obuhvaćanja zvukova iz svih ljudskih jezika. IPA je više od abecede, ona također sadrži skup načela transkripcije, koja variraju ovisno o potrebama transkripcije.

ARPAbet je još jedna fonetska abeceda nastala 1980. godine. No, ova je abeceda posebno dizajnirana za američki engleski jezik i koristi ASCII simbole. ARPAbet je vrlo česta u računarskoj reprezentaciji izgovora, pa ćemo se usredotočiti na nju.

U *Tablici 8* prikazani su samoglasnici, a u *Tablici 7* suglasnici preko ARPAbet i IPA simbola.

ARPABET simbol	IPA simbol	Primjer riječi	ARPAbet transkripcija
[iy]	[i]	lily	[l ih l iy]
[ih]	[ɪ]	lily	[l ih l iy]
[ey]	[eɪ]	daisy	[d ey z iy]
[eh]	[ɛ]	pen	[p eh n]
[ae]	[æ]	aster	[ae s t axr]
[aa]	[ɑ]	poppy	[p aa p iy]
[ao]	[ɔ]	orchid	[ao r k ix d]
[uh]	[ʊ]	wood	[w uh d]
[ow]	[oʊ]	lotus	[l ow dx ax s]
[uw]	[u]	tulip	[t uw l ix p]
[ah]	[ʌ]	buttercup	[b ah dx axr k ah p]
[er]	[ɝ]	bird	[b er d]
[ay]	[aɪ]	iris	[ay r ix s]
[aw]	[aʊ]	sunfloower	[s ah n f l aw axr]
[oy]	[oɪ]	soil	[s oy l]
[ax]	[ə]	lotus	[l ow dx ax s]
[axr]	[ɚ]	heather	[h eh dh axr]
[ix]	[ɨ]	tulip	[t uw l ix p]
[ux]	[ʉ]	dude	[d ux d]

Tablica 8: Samoglasnici: češće i rjeđe korišteni fonovi i alofoni

Možemo vidjeti da su u obje abecede simboli ekvivalentni slovima latinske (rimske) abecede. Međutim, jedno slovo može predstavljati različite zvukove u različitim kontekstima.

Tako, primjerice, englesko slovo *c* odgovara fonu [k] u riječi *cougar* [k uw g axr], ali fonu [s] u riječi *cell* [s eh l]. Osim što se pojavljuje kao *c* i *k*, fon [k] može se pojaviti i kao dio slova *x*, primjerice u riječi *fox* [f aa k s], zatim kao *ck*, kao u riječi *jackal* [jh ae k el], te kao *cc* u riječima poput *raccoon* [r ae k uw n].

3.1.2 Artikularna fonetika

Kada je riječ o nastajanju pojedinih fonova, okrećemo se artikularnoj fonetici. Ona proučava kako organi u ustima, grlu i nosu oblikuju protok zraka iz pluća i time stvaraju fonove.

Ukratko, zrak iz pluća putuje kroz dušnik i tada može ići ili u usta (većinom) ili u nos. Nazalni glasovi u engleskom jeziku su *m*, *n* te *ng*. Vibracija glasnica označava glas kao zvučni, inače je bezvučan. Suglasnike karakterizira prepreka na putu toka zraka. Suglasnici mogu biti zvučni ili bezvučni, a samoglasnici su obično zvučni te glasniji i dugotrajniji od suglasnika. Postoje i polu-suglasnici (kao [y] i [w]) koji imaju osobine i samoglasnika i suglasnika; zvučni su kao samoglasnici, ali su kratki i manje slogovni kao suglasnici. Detaljan opis nastanka glasova te njihova daljnja podjela s pripadnim ilustracijama dani su u [15].

Kombinacijom samoglasnika i suglasnika dobivamo slogove, za koje ne postoji egzaktna definicija. Riječ možemo stoga rastaviti na slogove, a struktura sloga povezana je i s fonotaktikom, proučavanjem pravila (načina) na koje fonemi mogu slijediti jedni druge u jeziku.

Što se tiče samog izgovora, on može varirati od osobe do osobe, te ovisiti o području na kojem osoba živi. Također, neki ljudi govore brže od drugih, imaju viši ili dublji glas, te govore različitim narječjima. Izgovor može ovisiti i o situaciji u kojoj se osoba nalazi ili čak o sugovorniku (ili više njih). Sve navedeno treba uzeti u obzir pri analizi govora.

3.1.3 Akustična fonetika i signali

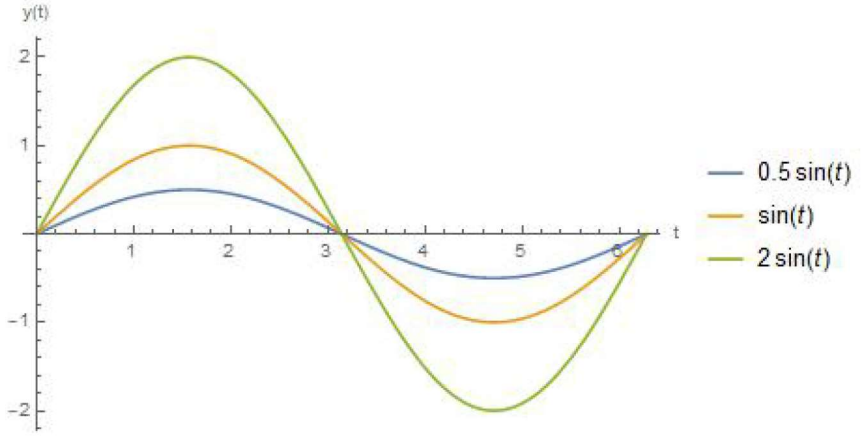
Još je zanimljivo spomenuti neke osnove o akustičnoj fonetici. Vidjet ćemo akustični valni oblik te kako se on digitalizira.

Akustična analiza temelji se na funkcijama sinus i kosinus, čiji grafovi sliče valu. Dvije osnovne karakteristike ovih funkcija su frekvencija (broj ciklusa po sekundi) i amplituda (maksimalna vrijednost koju funkcija postiže). Frekvencija se izražava u hercima (Hz). Period je vrijeme potrebno da se izvrši jedan ciklus, stoga je obrnuto proporcionalan frekvenciji. Neka je dana funkcija

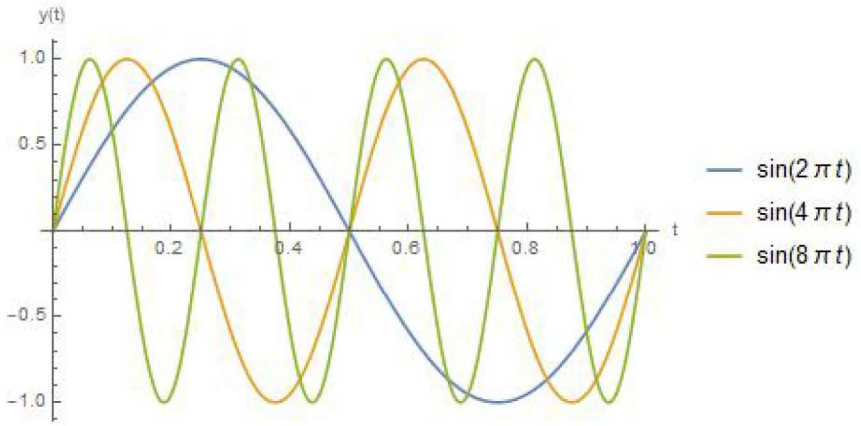
$$y(t) = A\sin(2\pi f t), \tag{3.1}$$

gdje je A amplituda, a f frekvencija. Na *Slici 2* prikazana je funkcija (3.1) za koju je frekvencija $f{=}\frac{1}{2\pi}$, a amplituda A poprima vrijednosti 0.5, 1 i 2. S druge strane, na *Slici 3* prikazana je ista funkcija s amplitudom $A{=}1$, a frekvencija f poprima vrijednosti 1, 2 i 4.

Sustav za prepoznavanje govora ili pretvorbu govora u tekst za ulazni podatak dobiva niz promjena u tlaku zraka. Ove promjene dolaze od govornika i uzrokovane su načinom na koji glas nastaje (o čemu smo ranije govorili).

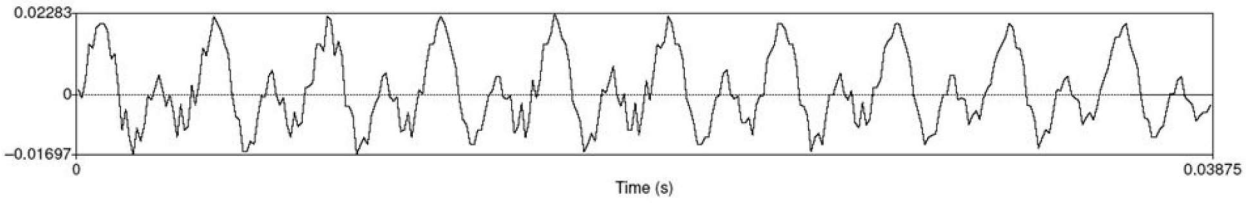


Slika 2: Funkcija sinus s fiksnom frekvencijom i različitim amplitudama



Slika 3: Funkcija sinus s fiksnom amplitudom i različitim frekvencijama

Zvučni valovi prikazuju se tako da grafički prikažemo promjenu u tlaku (pritisku) tijekom vremena. Graf će prikazivati količinu kompresije ili razrjeđenosti (dekompresije). Na *Slici 4* prikazan je kratki segment valnog oblika uzetog iz Switchboard korpusa telefonskih govora samoglasnika [iy] pri izgovaranju *she just had a baby*.



Slika 4: Valni oblik samoglasnika [iy]. Na y-osi prikazan je pritisak zraka iznad i ispod normalnog atmosferskog tlaka zraka. Na x-osi prikazano je vrijeme u sekundama.

Kako bismo konstruirali digitalnu reprezentaciju zvučnog vala na *Slici 4*?
Prvi korak u obradi govora je pretvaranje analognih prikaza (najprije tlak zraka, a zatim analogni električni signali u mikrofonu) u digitalni signal. Ovaj proces pretvaranja analognog u digitalno (*analog-to-digital conversion*) sastoji se od 2 koraka: uzorkovanja (*sampling*) i kvantizacije (*quantization*).

Signal se uzorkuje mjerenjem amplitude u određenom vremenu. Stopa uzorkovanja (*sampling rate*) označava broj uzoraka po sekundi. Kako bismo točno izmjerili val, potrebna su nam barem dva uzorka po ciklusu, jedan koji mjeri pozitivan dio vala i jedan koji mjeri negativan dio. Dakle, val maksimalne frekvencije koji se može izmjeriti je onaj čija frekvencija iznosi pola stope uzorkovanja (pošto svaki ciklus treba dva uzorka). Ova maksimalna frekvencija za danu stopu uzorkovanja zove se Nyquistova frekvencija.

Kako je većina informacija u ljudskom govoru ispod 10 000 Hz, potrebna je stopa uzorkovanja od 20 000 Hz za potpunu točnost. No, telefonski govor je filtriran, pa se telefonom mogu prenositi frekvencije samo ispod 4 000 Hz. Stoga je u tom slučaju dovoljna stopa uzorkovanja od 8 000 Hz. Za mikrofonski govor koristi se stopa uzorkovanja od 16 000 Hz.

Vidimo da čak i najmanja stopa uzorkovanja koju smo naveli (8 000 Hz) zahtijeva isto toliko mjerenja amplituda po sekundi govora, što je uistinu mnogo. Stoga je bitno efikasno pohraniti ta mjerenja, najčešće kao *integere*, ili 8-bitne ili 16-bitne. Ovaj proces prikaza realnih brojeva kao cijelih (*integers*) zove se kvantizacija. Postoji minimalna zrnatost (kvantna veličina) i sve vrijednosti koje su zbijenije (bliže jedna drugoj) od te vrijednosti prikazane su identično.

Nakon kvantizacije, podaci se pohranjuju u raznim formatima. Jedan parametar formata čine stopa uzorkovanja i veličina uzorka (telefonski govor ima stopu 8 kHz i pohranjuje se u 8-bitne uzorke, a mikrofonski ima stopu 16 kHz i pohranjuje se u 16-bitne uzorke). Još jedan parametar je broj kanala. Ako imamo dva sudionika, možemo podatke pohraniti ili u jedan dokument ili u dva različita dokumenta, jedan za svakog sugovornika. Zadnji parametar je način pohrane: linearno ili sažeto (komprimirano).

Jedan od uobičajenih sažetih formata koji se koristi za telefonske govore je μ-zakon. Intuicija logaritamskih kompresijskih algoritama poput ovoga zakona je da je ljudski sluh osjetljiviji na manje intenzitete nego na veće. Linearne vrijednosti (ne-logaritmirane) često se zovu linearne PCM vrijednosti (*Pulse Code Modulation*). Navest ćemo jednadžbu za kompresiju vrijednosti linearnog PCM uzorka x u 8-bitni μ-zakon, gdje je μ=255 za 8 bita:

$$F(x) = sign(x)\frac{\log(1+\mu|x|)}{\log(1+\mu)}, \quad -1 \leq x \leq 1.$$

Neki od standardnih formata za pohranu rezultirajućeg digitaliziranog vala su Microsoft-ov WAV, Apple-ov AIFF i Sun-ov AU.

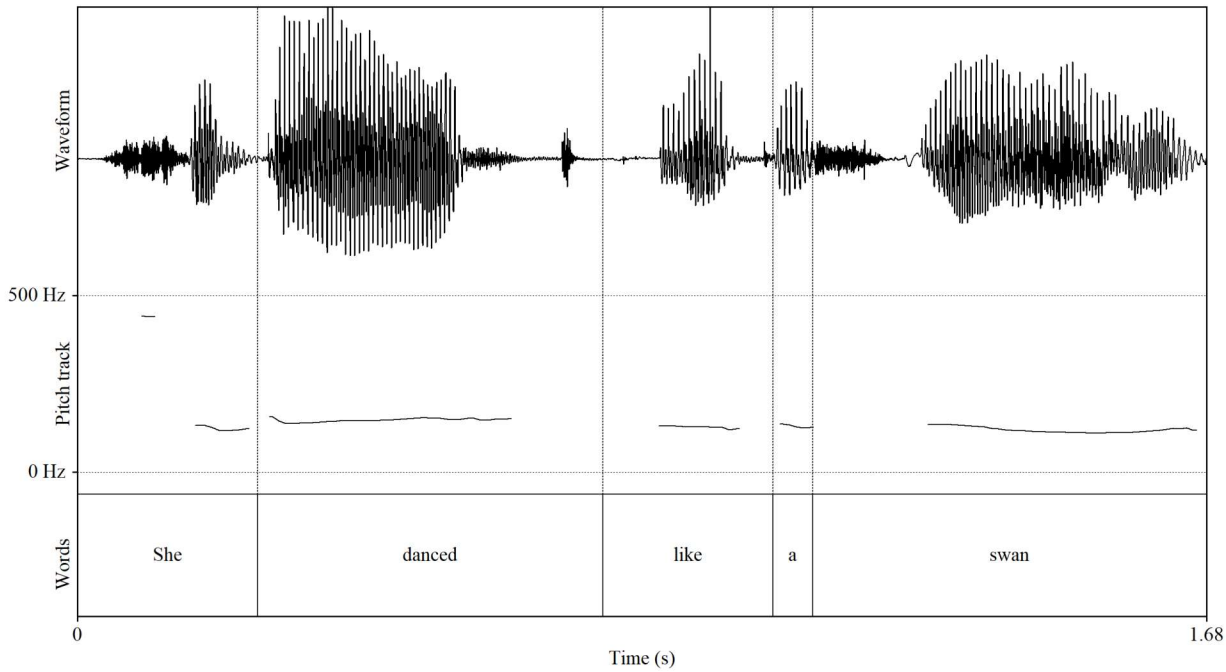
U *Tablici 9* prikazan je .wav format, podskup Microsoft-ovog RIFF formata za multimedijske datotetke. RIFF (*Resource Interchange File Format*) je općeniti format koji prikazuje nizove ugnježđenih komada (*nested chunks*) podataka i kontrolira informacije.
Stringovi se zapisuju u ASCII formatu, a endian govori o poretku bajtova od najvišeg (najsignifikantijeg) prema najnižem (najmanje signifikantnom). Big endian je način zapisa podatka u memoriji tako da je na nižoj adresi viši bajt, a little endian tako da je na nižoj adresi niži bajt. Primjerice, big endian zapis podatka 3A71 je 3A71, a little endian zapis istog podatka je 713A.

Nadalje, zvučni valovi, kao i svi drugi valovi, mogu se opisati u terminima frekvencije, amplitude i nekih drugih karakteristika koje smo ranije spomenuli. No, kod zvučnih valova

Endian	Ofset	Ime polja	Veličina polja	Opis/vrijednost
big	0	Chunk ID	4	„RIFF”
little	4	Chunk Size	4	36 + SubChunk2Size
big	8	Format	4	„WAVE”
big	12	SubChunk1 ID	4	„fmt ”
little	16	SubChunk1 Size	4	16
little	20	Audio Format	2	PCM=1 (nema kompresije), inače ima kompresije
little	22	Channel Number	2	broj kanala
little	24	Sample Rate	4	stopa uzorkovanja
little	28	Byte Rate	4	Sample Rate * Channel Number * Bits Per Sample/8
little	32	Block Align	2	Channel Number * Bits Per Sample/8
little	34	Bits Per Sample	2	broj bitova po uzorku
big	36	SubChunk2 ID	4	„data”
little	40	SubChunk2 Size	4	veličina podataka
little	44	Data	SubChunk2 Size	audio podaci

Tablica 9: Microsoft-ov RIFF format za valne datoteke

mjerenje ovih karakteristika je malo teže nego kod sinusnog vala. Kada glasnice vibriraju, očekujemo regularne šiljke u amplitudi kao na *Slici 4*. Frekvencija vibracije glasnice, ili frekvencija kompleksnog vala, zove se osnovna frekvencija (*fundamental frequency*) vala i označava se s F_0. Možemo grafički prikazati F_0 tijekom vremena u tragu visine tona (*pitch track*).



Slika 5: Trag visine tona (ispod valnog oblika) rečenice „She danced like a swan”

Na *Slici 5* prikazan je val i trag visine tona rečenice *She danced like a swan* koje smo konstruirali pomoću alata Praat (dostupan u [2]) i audio datoteke preuzete iz [14].

Osim što možemo vidjeti promjenu amplitude u valnom obliku tijekom vremena, zanimat će nas i prosječna amplituda tijekom nekog vremenskog intervala. Uzmemo li aritmetičku sredinu amplituda koje se pojavljuju, pozitivne i negativne vrijednosti će se (većinom) poništiti. Time bismo dobili broj blizu nuli.

Umjesto toga se koristi RMS (*root-mean-square*) amplituda. Neka je n broj uzoraka signala, a $x = \{x_1, ..., x_n\}$ amplitude tih uzoraka. Tada se RMS amplituda x_{rms} računa kao:

$$x_{\text{rms}} = \sqrt{\sum_{i=1}^{n} \frac{x_i^2}{n}},$$

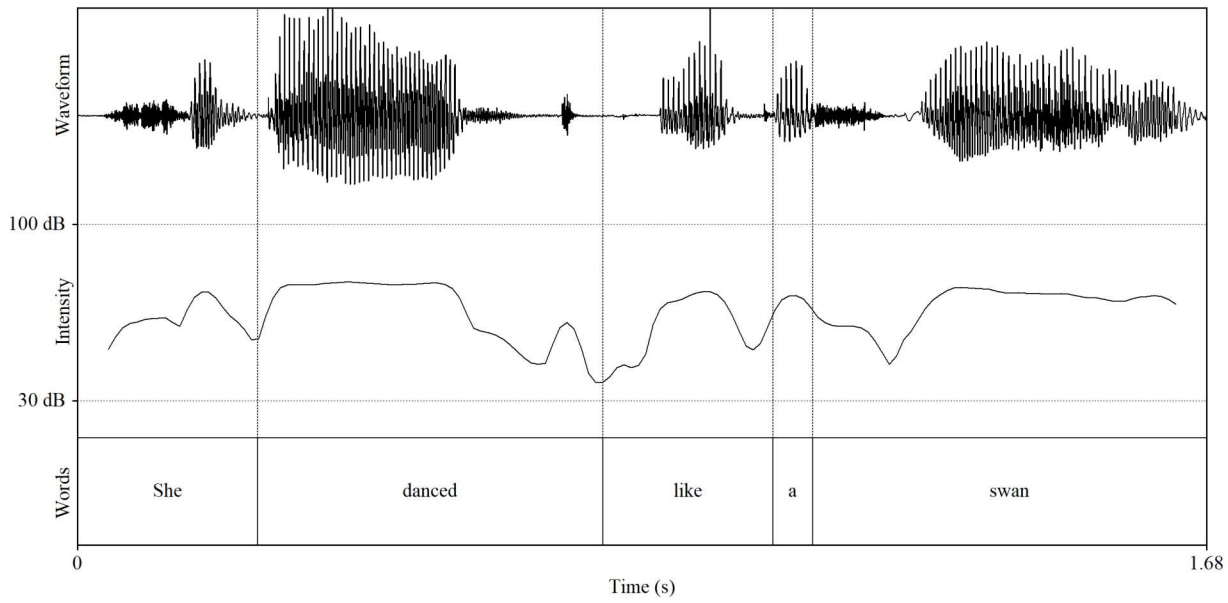
a jačina (*power*) P signala kao:

$$P(x) = \frac{1}{n} \sum_{i=1}^{n} x_i^2 = x_{\text{rms}}^2.$$

No, često nas više zanima intenzitet (*intensity*) nego jačina zvuka. Intenzitet I mjerimo u decibelima (dB), a računamo ga na sljedeći način:

$$I(x) = 10 \log \left(\frac{1}{nP_0} \sum_{i=1}^{n} x_i^2 \right) = 10 \log \left(\frac{P(x)}{P_0} \right), \quad P_0 = 2 \times 10^{-5} \text{Pa}.$$

Na *Slici 6* prikazan je intenzitet rečenice „She danced like a swan" uz pripadni valni oblik.

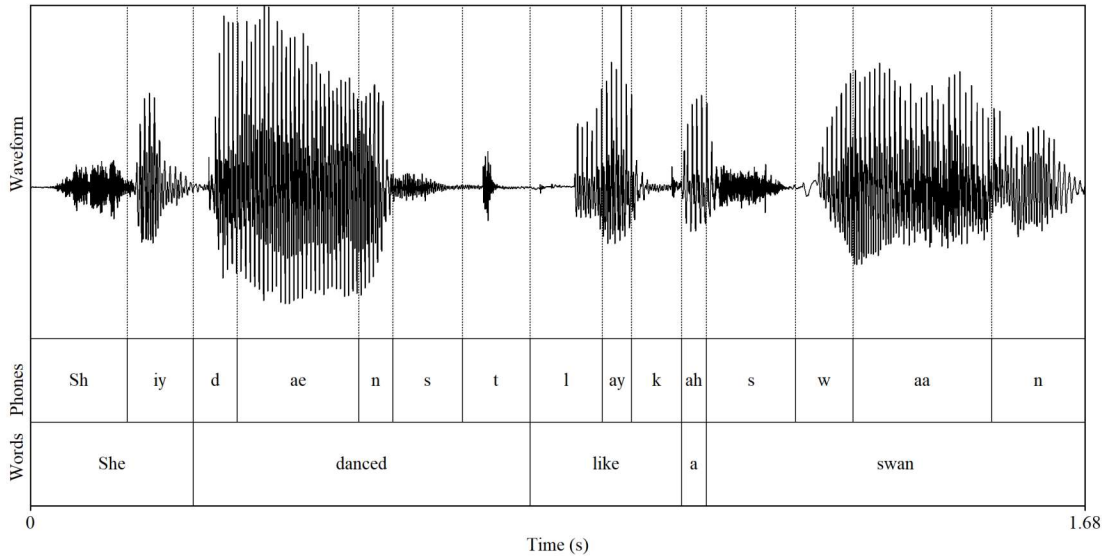


Slika 6: Intenzitet zvuka (ispod valnog oblika) rečenice „She danced like a swan"

Visina tona i glasnoća su svojstva zvučnog vala koja su povezana s frekvencijom i intenzitetom. Ljudski osjećaj frekvencije predstavlja visinu tona, a glasnoća zvuka odgovara jačini zvuka. Što je osnovna frekvencija veća, reći ćemo da je visina tona zvuka koji čujemo veća. Oni zvučni valovi koji imaju veću amplitudu su glasniji. No, veza između visine tona i frekvencije, kao i veza između glasnoće zvuka i jačine zvuka nije linearna.

Što se tiče traženja osnovne frekvencije F_0, u tu svrhu se koriste tzv. *pitch extraction* algoritmi (algoritmi za ekstrakciju visine tona).

Za kraj poglavlja spomenut ćemo kako možemo iz valnog oblika iščitati fonove. Samoglasnici su zvučni, obično dugi i relativno glasni. Duljinu gledamo na x-osi, a glasnoću vežemo za y-os prikaza valnog oblika.
Na *Slici 7* prikazani su fonovi rečenice „She danced like a swan" gdje možemo vidjeti upravo ova svojstva samoglasnika i prepoznatljiv oblik riječi „she".
Za provjeru ARPAbet transkripcije rečenice možemo koristiti [13].



Slika 7: Valni oblik rečenice „She danced like a swan" s pripadnim fonovima

3.2 Sinteza govora

Sinteza govora (*speech synthesis*), odnosno *text-to-speech* (TTS) je proces stvaranja govora (zvučnih valnih oblika) iz teksta. Taj proces podijeljen je na dva koraka:

- tekstualnu analizu (*text analysis*) - pretvaranje teksta u unutarnji fonemički prikaz te
- sintezu valnog oblika (*waveform synthesis*) - pretvaranje fonemičkog prikaza u valni oblik,

no postoje i drugi nazivi za ova dva koraka.

Postoje tri uvelike različite paradigme za sintezu valnog oblika: lančana sinteza (*concatenative synthesis*), formantska sinteza (*formant synthesis*) i artikularna sinteza (*articulatory synthesis*).
Većina TTS sustava temelji se na lančanoj sintezi, u kojoj su uzorci govora isjeckani i pohranjeni u bazu podataka, te kombinirani kako bi stvorili nove rečenice. S druge strane, artikularna sinteza pokušava direktno modelirati fiziku glasnica i artikularnog procesa, dok formantska sinteza pokušava stvoriti pravila koja će kombiniranjem formanata stvoriti riječi (čak i one koje ne postoje).

Ukratko ćemo opisati lančanu sintezu te prijeći na sintaksu.

3.2.1 Lančana sinteza

Kako bismo dobili unutarnji fonemički prikaz teksta, taj tekst najprije trebamo segmentirati i normalizirati. Tekst se stoga podijeli na rečenice te se obrade kratice, brojevi i slično. Dakle, prvi zadatak je tokenizacija (segmentacija) rečenica.

Nakon toga, trebamo se pozabaviti nestandardnim riječima kao što su brojevi, akronimi, skraćenice itd. Primjerice, *$1 billion* treba se izgovoriti kao „one billion dollars".
Ovaj dio zahtijeva barem 3 koraka: tokenizaciju (identificiranje potencijalnih nestandardnih riječi), klasifikaciju (označavanje tipa riječi), te ekspanziju (pretvaranje tipa u string standardnih riječi).

Kada imamo normalizirani tekst, trebamo stvoriti izgovor za svaku pojedinu riječ u tom tekstu. Stoga je potreban veliki rječnik izgovora. No, rječnici su često nedovoljni, kako tekst obično sadrži riječi koje se ne pojavljuju u rječniku. Najčešći problem koji se javlja je izgovor osobnih imena, geografskih imena (gradovi, ulice i slično) te komercijalnih imena (tvrtke i imena proizvoda).
Najraširenij rječnik koji se koristi pri TTS procesu je CMU Pronouncing Dictionary, koji sadrži izgovore za oko 120 tisuća riječi. Rječnik je dizajniran za prepoznavanje govora, a ne za sintezu govora. Bitno je koristiti i veliki rječnik imena. CMU Pronouncing Dictionary sadrži izgovore čak 50 tisuća najčešćih prezimena.

Nakon što pronađemo standardne riječi u rječniku, trebamo nabaviti i izgovore za nepoznate riječi. Za taj dio obrade koristimo konverziju grafema u fonem (*grapheme-to-phoneme conversion*). Algoritam treba niz slova pretvoriti u pripadni fonemski string. Moguće je koristiti poravnavanje ova dva stringa te pronaći sve moguće foneme za svako slovo. Nakon toga mogu se pronaći vjerojatnosti poravnavanja (koliko je vjerojatno da određeno slovo odgovara određenom fonemu) te iskoristiti Viterbi algoritam kako bismo pronašli poravnavanje

najveće vjerojatnosti.

Zadnji korak analize je prozodijska analiza (*prosodic analysis*). Prozodija se odnosi na proučavanje intonacije i ritma jezika. Sastoji se od 3 aspekta: prozodijske istaknutosti (*prosodic prominence*), prozodijske strukture (*prosodic structure*) i melodije (*tune*).

Prozodijska struktura rečenice znači da se neke riječi prirodno grupiraju, dok se kod drugih javlja primjetan prekid.
Prodijska istaknutost znači da neke riječi zvuče istaknutije od drugih. Takve riječi se izgovaraju glasnije, sporije, ili osnovna frekvencija varira kroz riječ. Istaknute riječi označavaju se naglaskom (*pitch accent*). Naglašeni slog označava dio riječi gdje se naglasak realizira. Mogu se koristiti čak 4 stupnja istaknutosti: naglašeni naglasak (*emphatic accent*), naglasak (*pitch accent*), nenaglašenost (*unaccented*) i skraćenost (*reduced*).
No, dvije izjave s istom istaknutosti i formulacijom mogu se prozodijski razlikovati po melodiji. Melodija izjave opisuje rast i pad osnovne frekvencije tijekom vremena. Rečenica može imati povišenu osnovnu frekvenciju na kraju kako bi se označilo pitanje, ili konačan pad frekvencije kako bi se označila izjavna intonacija.

Konačan izlaz tekstualne analize je unutarnja reprezentacija teksta. Slijedi sinteza valnog oblika koja se može napraviti na dva načina: difonskom sintezom (*diphone synthesis*) i sintezom izbora jedinice (*unit selection synthesis*).

Za difonsku sintezu, unutarnja reprezentacija sastoji se od liste fonova, gdje je svakom fonu pridruženo trajanje i skup osnovnih frekvencija. Ovaj model generira valni oblik iz niza fonova odabirom i konkatenacijom jedinica iz baze podataka difonova. Difon je jedinica slična fonu koja se proteže od sredine jednog fona do sredine idućeg fona. Difonovi su bolji izbor zbog toga što fon može biti različit obzirom na fonove koji ga okružuju.
Kako ova metoda ima nekoliko nedostataka, moderni komercijalni sintesajzeri temelje se na generalizaciji ove metode, sintezi izbora jedinice. Ovdje baza podataka ne sadrži samo jednu kopiju difona, nego mnogo kopija te se ne primjenjuje obrada signala (ili se primjenjuje u jako maloj mjeri), za razliku od difonske sinteze gdje je potrebna takva obrada.

Svaki sustav za sintezu govora procjenjuje se od strane ljudskih slušatelja. Metrika koja se koristi je razumljivost: mogućnost slušatelja da točno interpretira riječi i značenje sintetizirane izjave. Dodatna metrika je kvaliteta: priroda, tečnost, ili jasnoća govora.

4 Sintaksa

Riječ sintaksa dolazi iz grčkog jezika, od riječi *sýntaksis*, što znači slaganje u red. Sintaksa se odnosi na način poretka riječi u rečenici. Znamo da ne možemo riječi slagati proizvoljno, nego po određenim pravilima, ovisno o kojem jeziku je riječ. Stoga ćemo se upoznati s načinom stvaranja rečenica prema spomenutim pravilima.

4.1 Formalne gramatike

Neke grupe riječi ponašaju se kao jedna jedinica ili fraza, kao što su, primjerice, imenske fraze. U rečenici također postoje odnosi, kao što su subjekt i objekt. Također, postoje pravila (ovisnosti) među riječima, na način da glagol *want* može slijediti glagol u infinitivu (*to go*) ili imenska fraza (*a car*). No, glagol *find* ne može slijediti glagol u infinitivu. Ovo se zovu činjenice o podkategorizaciji glagola.

U ovome poglavlju pozabavit ćemo se kontekstno-slobodnim gramatikama na kojima se temelje mnogi formalni modeli sintakse prirodnog jezika.

4.1.1 Kontekstno-slobodne gramatike

Kontekstno-slobodne gramatike (*context-free grammars* - CFG) poznate su i pod imenom kontekstno-neovisne gramatike, a sastoje se od:

- skupa varijabli,
- skupa terminala,
- liste produkcija (pravila).

Pravila se pišu tako da je početna varijabla (najčešće S) na prvom mjestu (i to lijevo), varijable se obično pišu velikim tiskanim slovima, a terminali su konkretni elementi rječnika.

Primjerice, ako želimo prikazati činjenicu da se imenska fraza (NP - *noun phrase*) može sastojati od vlastite imenice (*ProperNoun*) ili determinera (*Det*) kojeg slijedi nominal (*Nom*) koji se dalje može sastojati od jedne ili više imenica (*Noun*), tada pišemo sljedeće:

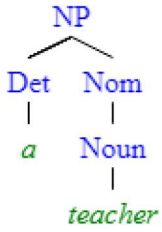
$$\begin{aligned} NP &\rightarrow Det\ Nom \\ NP &\rightarrow ProperNoun \\ Nom &\rightarrow Noun \mid Nominal\ Noun, \end{aligned}$$

gdje su sve navedene riječi u pravilima varijable.
S druge strane, terminali gramatike će biti riječi, kao primjerice:

$$\begin{aligned} Det &\rightarrow a \\ Det &\rightarrow the \\ Noun &\rightarrow teacher. \end{aligned}$$

Strelice u pravilima označavaju zamjene, točnije varijablu s lijeve strane zamjenjujemo izrazom s desne strane (ili ako ih ima više, jednim od izraza). Pogledajmo kako to funkcionira na našem primjeru.

Krećemo od varijable *NP* koju zamjenjujemo s *Det Nom*. Zatim varijablu *Det* zamijenimo s *a* pa dobivamo *a Nom*. Umjesto varijable *Nom* zapišemo *Noun* i dobivamo *a Noun*. Konačno, zamjenom *Noun* terminalom *teacher* dobivamo *a teacher*.
Ovakav niz proširenja pravila nazivamo izvod (*derivation*) i često ga prikazujemo pomoću stabla raščlanjivanja/parsiranja (*parse tree*). Stablo parsiranja za ovaj primjer dano je na *Slici 8*, a konstruirano je pomoću alata dostupnog u [11].



Slika 8: Stablo parsiranja za „a teacher"

Dakle, pomoću kontekstno-slobodnih gramatika (KSG) možemo generirati rečenice te im dati strukturu. Pogledajmo stoga još nekoliko pravila:

$$\begin{aligned} S &\to NP\ VP \\ VP &\to Verb\ NP \\ VP &\to Verb\ NP\ PP \\ VP &\to Verb\ PP \\ PP &\to Preposition\ NP, \end{aligned}$$

gdje je S početna varijabla, *VP* glagolska fraza, *Verb* glagol, *PP* prijedložna fraza, a *Preposition* prijedlog.

Nakon što smo se upoznali s osnovama kontekstno-slobodnih gramatika, uvedimo i njihovu formalnu definiciju (kao u [10]).

Kontekstno-slobodna gramatika je 4-torka (V, Σ, R, S) gdje je:

- V konačan skup varijabli,
- Σ konačan skup terminala, te vrijedi $V \cap \Sigma = \emptyset$,
- R konačan skup pravila oblika $A \to B$, gdje je A varijabla, a B string sastavljen od varijabli i terminala ($A \in V$, $B \in (\Sigma \cup V)^*$),
- $S \in V$ početna varijabla.

Varijable ćemo označavati velikim tiskanim slovima, a početnu varijablu sa S. Mala slova latinice označavat će terminale, a mala grčka slova elemente skupa $(\Sigma \cup V)^*$, gdje je $(\Sigma \cup V)^*$ skup svih mogućih kombinacija elemenata iz skupa $(\Sigma \cup V)$.

Ako su $\alpha, \beta, \gamma \in (\Sigma \cup V)^*$ i $A \in V$ te $A \to \beta \in R$, kažemo da $\alpha A \gamma$ daje (*yields*) $\alpha\beta\gamma$ i pišemo $\alpha A \gamma \Rightarrow \alpha\beta\gamma$.
Kažemo da α izvodi γ i pišemo $\alpha \overset{*}{\Rightarrow} \gamma$ ako je $\alpha = \gamma$ ili postoji niz $\alpha_1, ..., \alpha_k \in (\Sigma \cup V)^*$ za $k \geq 0$ takav da $\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow ... \Rightarrow \alpha_k \Rightarrow \gamma$.

Jezik gramatike je tada definiran kao skup $\left\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w,\ S \text{ početna varijabla}\right\}$.

Reći ćemo da su dvije gramatike slabo ekvivalentne ako generiraju isti skup riječi, a jako ekvivalentne ako su slabo ekvivalentne i dodjeljuju istu fraznu strukturu svakoj rečenici.

Ponekad je poželjno imati gramatiku u pojednostavljenoj formi. Jedna od najjednostavnijih i najkorisnijih je Chomskyjeva normalna forma. KSG je u Chomskyjevoj normalnoj formi ako je svako pravilo oblika

$$A \to BC$$
$$A \to a,$$

gdje su A, B i C varijable, pri čemu B i C nisu početne varijable, a a je terminal. Gramatike u Chomskyjevoj normalnoj formi imaju binarna stabla.

4.1.2 Osnovna gramatička pravila engleskog jezika

Postoje mnogi načini konstrukcija rečenica u engleskom jeziku, a četiri od njih su najvažnije i ujedno najuobičajenije: deklarativna struktura, imperativna struktura, struktura da-ne-pitanja i struktura wh-pitanja. Detaljan opis i izvod gramatičkih pravila za svaku od struktura nalazi se u [4]. Ukratko ćemo dati primjer jedne analize imenske fraze kako bismo opisali stvaranje gramatičkih pravila.

Imenska fraza može početi determinerom koji se može zamijeniti i složenijim izrazom. Ulogu determinera može imati i posvojni izraz koji se sastoji od imenske fraze koju slijedi *'s*, što zapisujemo kao *Det* $\to$ *NP 's*. Ponekad se determineri mogu ispustiti, kao u slučaju kada stoje uz imenicu koja je u množini ili kod zbirnih imenica. Nominal slijedi determiner i može sadržavati modifikatore ispred i iza imenice, ili jednostavno sadržavati samu imenicu (*Nom* $\to$ *Noun*). Modifikatori koji se mogu pojaviti ispred imenice uključuju kardinalne (glavne) brojeve, redne brojeve ili količinske priloge (*quantifiers*). Pridjevi dolaze između količinskih priloga i imenica te mogu biti grupirani u pridjevnu frazu (AP) koje mogu imati i prilog ispred pridjeva. Uzmemo li u obzir sve navedene opcije, možemo zapisati pravilo

$$NP \to (Det)\ (Card)\ (Ord)\ (Quant)\ (AP)\ Nominal,$$

gdje zagrade govore da je sadržaj koji se nalazi unutar njih opcionalan.

Osim što moramo paziti na redoslijed, bitno je voditi računa i o licima koja se pojavljuju u rečenici. Stoga, pravila trebamo pisati tako da naznačimo poklapanje u licu. Svako pravilo treba zamijeniti pravilima koja obuhvaćaju slaganje u jednini i slaganje u množini. No, ovaj način rješavanja problema slaganja u licu udvostručuje veličinu gramatike. U nekim drugim jezicima moramo voditi računa i o slaganju u spolu, što dodatno povećava veličinu gramatike.

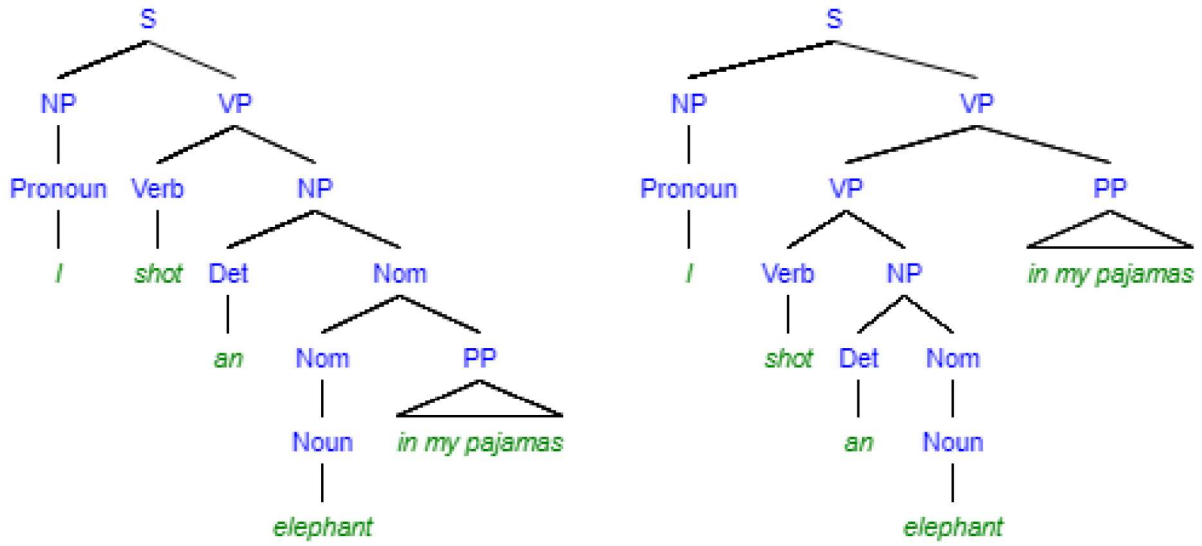
4.1.3 Treebanks

Banke stabala (*treebanks*) su korpusi gdje je svakoj rečenici korpusa pridruženo pripadno stablo parsiranja. Iz rečenica koje se nalaze u banci stabala možemo izvesti gramatiku, odnosno pravila. Stoga možemo i pretraživati ove korpuse kako bismo pronašli određena gramatička pravila. No, regularni izrazi s kojima smo se ranije susreli nisu od pomoći. Postoje posebni jezici u različitim alatima za pretraživanja ovakvih stabala (kao Tgrep i TGrep2). Nećemo ulaziti u daljnju analizu ovih alata. Više o njima može se pronaći u [12].

4.2 Sintaktičko parsiranje

Problem mapiranja stringa riječi u pripadno stablo parsiranja zove se sintaktičko parsiranje (*syntactic parsing*). Stabla parsiranja korisna su pri gramatičkim provjerama te predstavljaju jednu vrstu reprezentacije u sintaktičkom parsiranju (stoga imaju veliku ulogu pri odgovaranju na pitanja i izvlačenju informacija).

No, dvosmislenost može (ponovno) biti problem. Naime, gramatika može jednoj rečenici dodijeliti više od jednog parsiranja, kao što je prikazano na *Slici 9.*



(a) Slon je u pidžami (šaljiva rečenica)

(b) Subjekt je u pidžami dok slika slona

Slika 9: Stabla parsiranja za dvosmislenu rečenicu „I shot an elephant in my pajamas"

Dvije česte vrste dvosmislenosti koje se pojavljuju su dvosmislenost priljepljivanja (*attachment ambiguity*) i koordinacijska dvosmislenost (*coordination ambiguity*).
Dvosmislenost priljepljivanja znači da dio rečenice može „prilijepiti" na više od jednog mjesta u stablu parsiranja, dok koordinacijska dvosmislenost govori da veznik (poput veznika *and*) može spajati različite grupe fraza (*[old [men and women]]* ili *[old men] and [women]*).

Sintaktičko uklanjanje dvosmislenosti (*syntactic disambiguation*) je proces kojim sustavi za obradu prirodnog jezika odabiru jedno točno parsiranje od nekoliko mogućih. CKY (Cocke–Kasami-Younger) algoritam je jedan od algoritama dizajniranih za efikasno baratanje strukturalnim dvosmislenostima.

4.2.1 CKY algoritam

CKY algoritam koristi princip dinamičkog programiranja, a zahtijeva da gramatike budu u Chomskyjevoj normalnoj formi. Stoga ćemo ukratko objasniti kako možemo KSG pretvoriti u pripadnu CNF (prema [10]).

Najprije pretpostavimo da nema ϵ-pravila, odnosno pravila koja daju prazan string. Tada slijedimo sljedeća pravila:

- dodamo novu početnu varijablu S_0 i pravilo $S_0 \to S$, gdje je S originalna početna varijabla,
- svako pravilo oblika $A \to B$ brišemo i umjesto svakog pravila $B \to \gamma$ pišemo $A \to \gamma$ (osim ako to pravilo nije već uklonjeno),
- svako pravilo $A \to u_1 u_2 ... u_k$, gdje su $u_1, u_2, ..., u_k$ terminali ili varijable, zamjenjujemo pravilima $A \to u_1 A_1$, $A_1 \to u_2 A_2$,...,$A_{k-2} \to u_{k-1} u_k$, pri čemu su A_i nove varijable, $i = 1, ..., k-2$, a u novim pravilima svaki terminal u_j zamjenjujemo novom varijablom U_j i dodajemo pravilo $U_j \to u_j$.

Kako smo ranije spomenuli, u stablu parsiranja za gramatiku u CNF-u svaki čvor ima najviše dva djeteta (jedno dijete imaju samo oni čvorovi koji generiraju terminal). Dvodimenzionalna matrica može sadržavati strukturu takvog stabla.

Za ulaznu rečenicu od n riječi, imat ćemo gornjetrokutastu matricu reda $(n+1)$. Na poziciji (i, j) nalazi se skup ne-terminala (varijabli) koji predstavljaju sve sastavne jedinice koje se nalaze između i-te i j-te riječi. Skup varijabli na poziciji $(0, n)$ stoga predstavlja cijelu rečenicu. Kako svaka varijabla ima dva djeteta u parsiranju, za svaki sastavni dio na poziciji (i, j) postoji k takav da $i < k < j$ koji označava poziciju podjele (prvi dio je na (i, k), a drugi na (k, j)). Stoga se matrica popunjava od dolje prema gore i s lijeva na desno.

Algoritam 3 opisuje CKY prepoznavanje (ne parsiranje!). Matrica ima indekse od 0 do n, a lista riječi od 1 do n.

Algoritam 3 CKY-Parse

Ulaz: niz riječi *words* i gramatika *grammar*
Izlaz: matrica M

```
n ← Length(words)
Stvori matricu M reda (n + 1)
for j ← 1 to n do
    for all {A | A → words[j] ∈ grammar } do
        M[j − 1, j] ← M[j − 1, j] ∪ A
    end for
    for i ← j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
            for all {A | A → BC ∈ grammar and B ∈ M[i, k] and C ∈ M[k, j]} do
                M[i, j] ← M[i, j] ∪ A
            end for
        end for
    end for
end for
return M
```

Algoritam treba jednostavno pronaći varijablu S na poziciji $(0, n)$ kako bi radio. Kako bi algoritam mogao parsirati rečenicu i vratiti sva moguća parsiranja, uvedu se dvije modifikacije.

Prvo se na svakoj poziciji matrice svakoj varijabli koja je zapisana doda pokazivač na onu poziciju iz koje je izvedena. Zatim se dopuste višestruke verzije iste varijable (pravila mogu biti spojena operatorom disjunkcije). Tada možemo odabrati varijablu S na poziciji $(0, n)$

te rekurzivno odabirati pokazivače dok ne dobijemo sve terminale. Primijetimo da možemo vraćati eksponencijalan broj parsiranja u odnosu na ulaznu rečenicu. Ovaj problem može se riješiti statističkim parsiranjem koristeći izmijenjen Viterbi algoritam.

Bitno je napomenuti da bi pravilno bilo sačuvati podatke o pretvaranju KSG u CNF zato što je potrebno (radi preciznosti) stabla parsiranja (iz CKY algoritma) pretvoriti u ona stabla parsiranja koja odgovaraju originalnoj KSG.

4.2.2 Parcijalno parsiranje

Kada želimo napraviti ekstrakciju informacija, ne izvlačimo sve moguće informacije iz teksta, nego identificiramo i klasificiramo segmente teksta koji vrlo vjerojatno sadrže korisne informacije. U tu svrhu koristimo parcijalno parsiranje (ili plitko parsiranje) kojemu možemo pristupiti na mnogo načina.

Komadanje (*chunking*) je oblik parcijalnog parsiranja u kojem se identificiraju i klasificiraju plosnati segmenti rečenice koji se ne preklapaju i koji tvore osnovne ne-rekurzivne fraze koje odgovaraju imenskoj, glagolskoj, pridjevnoj ili prijedložnoj frazi. Stoga je dovoljno uglatim zagradama odvojiti spomenute dijelove rečenice (*[$_{NP}$ The morning flight] [$_{PP}$ from Denver] [$_{VP}$ has arrived]*). Dakle, napravi se segmentacija rečenice i svakom komadu se dodijeli oznaka. Moguće je da neke riječi neće biti niti u jednom komadu.

4.3 Statističko parsiranje

Statističko parsiranje (*statistycal parsing*) je vrlo korisno pri rješavanju problema dvosmislenosti. Naime, možemo izračunati vjerojatnosti svakog dobivenog stabla parsiranja i odabrati ono koje ima najveću vjerojatnost. Vjerojatnosni parseri koji rade na opisani način se danas koriste pri zadacima razumijevanja prirodnog jezika kao što su strojno prevođenje (*machine translation*), sažimanje (*summarization*), odgovaranje na pitanja i drugi.

Kada je riječ o jezičnom modeliranju (*language modeling*), vjerojatnosne gramatike i parseri se koriste za prepoznavanje govora. Najkorištenija takva gramatika je upravo vjerojatnosna kontekstno-slobodna gramatika (*probabilistic context-free grammar* - PCFG).

4.3.1 Vjerojatnosne kontekstno-slobodne gramatike

Vjerojatnosna kontekstno-slobodna gramatika je 4-torka (V, Σ, R, S) gdje je:

- V konačan skup varijabli,
- Σ konačan skup terminala, te vrijedi $V \cap \Sigma = \emptyset$,
- R konačan skup pravila oblika $A \to \beta\ [p]$, gdje je A varijabla, β string sastavljen od varijabli i terminala, a p vjerojatnost $P(\beta \mid A)$, $(A \in V,\ \beta \in (\Sigma \cup V)^*, p \in [0, 1])$,
- $S \in V$ početna varijabla.

Vjerojatnost $P(\beta \mid A)$ označava vjerojatnost da će se varijabla A proširiti u niz (string) β. Stoga možemo pisati i $P(A \to \beta)$. Stoga, gledamo li sva moguća proširenja varijable $A \in V$, mora vrijediti:

$$\sum_{\beta} P(A \to \beta) = 1.$$

Kažemo da je vjerojatnosna kontekstno-slobodna gramatika konzistentna ako je suma vjerojatnosti svih rečenica u jeziku jednaka 1. Rekurzivna pravila u gramatici mogu učiniti pripadnu gramatiku nekonzistentnom.

Spomenuli smo kako ove gramatike možemo koristiti pri uklanjanju dvosmislenosti. One će svakom stablu parsiranja T rečenice S dodijeliti vjerojatnost. Pretpostavimo da je i-to pravilo oblika $LHS_i \to RHS_i$ (LHS - *left-hand-side*, RHS - *right-hand-side*). Tada je vjerojatnost stabla parsiranja T rečenice S definirana kao produkt vjerojatnosti svih n pravila koja su korištena pri proširenju n varijabli u T:

$$P(T, S) = \prod_{i=1}^{n} P(RHS_i \,|\, LHS_i).$$

Kako stablo parsiranja T sadrži sve riječi rečenice S, onda je $P(S, T) = 1$ i vrijedi:

$$P(T, S) = P(T)P(S, T) = P(T). \tag{4.1}$$

Rečenicu S zovemo „plod" (*yield*) stabla parsiranja od S. Vjerojatnosni algoritmi za uklanjanje dvosmislenosti od svih stabla parsiranja s plodom S odabiru ono stablo $\hat{T}$ koje je najvjerojatnije uz dani S:

$$\hat{T}(S) = \underset{T\,:\,S=\text{yield}(T)}{\text{argmax}} P(T \,|\, S) = \underset{T\,:\,S=\text{yield}(T)}{\text{argmax}} \frac{P(T, S)}{P(S)} = \underset{T\,:\,S=\text{yield}(T)}{\text{argmax}} P(T, S).$$

Iskoristili smo definiciju uvjetne vjerojatnosti i činjenicu da promatramo stabla parsiranja za istu rečenicu S, što znači da je S konstanta. Zbog jednakosti (4.1) možemo pisati

$$\hat{T}(S) = \underset{T\,:\,S=\text{yield}(T)}{\text{argmax}} P(T).$$

Ako bismo željeli vjerojatnost za dvosmislenu rečenicu S, tada bismo zbrojili vjerojatnosti svih stabala parsiranja čiji je plod upravo S:

$$P(S) = \sum_{T\,:\,S=\text{yield}(T)} P(T, S) = \sum_{T\,:\,S=\text{yield}(T)} P(T).$$

4.3.2 Vjerojatnosni CKY algoritam

Kako bismo odabrali najvjerojatnije stablo parsiranja $\hat{T}$ za rečenicu S, potrebno je modificirati postupak transformacije KSG u CNF tako da vjerojatnosti pravila gramatike ostanu sačuvane. Matricu modificiramo tako da joj dodamo još jednu dimenziju kako bismo mogli spremiti vjerojatnost svakog dijela rečenice. Dakle, matrica je veličine $(n+1) \times (n+1) \times |V|$, gdje je $|V|$ veličina skupa varijabli gramatike. Svako polje (i, j, A) trodimenzionalne matrice sadrži vjerojatnost dijela A rečenice S koji se proteže od i-te do j-te riječi.

Vjerojatnosni CKY algoritam dan je u Algoritmu 4. Funkcija BUILD_TREE gradi stablo parsiranja pomoću polja pokazivača *back*. Vjerojatnostna kontekstno-slobodna gramatika može naučiti vjerojatnosti prebrojavanjem u parsiranom korpusu ili parsiranjem u korpusu.

Dva su glavna problema vjerojatnosnih kontekstno-slobodnih gramatika:

- loše pretpostavke nezavisnosti - pravila KSG-a nameću pretpostavku nezavisnosti vjerojatnostima, a u stablu parsiranja postoje strukturalne zavisnosti
- nedostatak uvjetovanja leksikona - pravila KSG-a ne modeliraju sintaktičke činjenice o određenim riječima, što dovodi do problema s podkategorizacijskim dvosmislenostima, dodavanjem prijedloga i slično.

Jedan način rješavanja problema je dijeljenje i spajanje varijabli. Postupak se može raditi automatski ili ručno.

Algoritam 4 PROBABILISTIC-CKY

Ulaz: niz riječi *words* i gramatika *grammar*
Izlaz: najvjerojatnije stablo parsiranja $\hat{T}$ i pripadna vjerojatnost

$n \leftarrow$ LENGTH(*words*)
Stvori matricu M veličine $(n+1) \times (n+1) \times |V|$
for $j \leftarrow 1$ **to** n **do**
  for all $\{A \mid A \rightarrow words[j] \in grammar\}$ **do**
    $M[j-1, j, A] \leftarrow P(A \rightarrow words[j])$
  end for
  for $i \leftarrow j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      for all $\{A \mid A \rightarrow BC \in grammar$ **and** $M[i,k,B] > 0$ **and** $M[k,j,C] > 0\}$ **do**
        if $M[i,j,A] < P(A \rightarrow BC) \times M[i,k,B] \times M[k,j,C]$ **then**
          $M[i,j,A] \leftarrow P(A \rightarrow BC) \times M[i,k,B] \times M[k,j,C]$
          $back[i,j,A] \leftarrow \{k, B, C\}$
        end if
      end for
    end for
  end for
end for
return BUILD_TREE($back[1,n,S]$), $M[1,n,S]$

5 Primjena: Izvlačenje informacija

Kroz rad smo nekoliko puta spomenuli proces ekstrakcije informacija. Izvlačenje informacija (*information extraction*) je proces automatskog izvlačenja nestrukturiranih informacija iz teksta i pretvaranja tih informacija u strukturirane. Možemo reći i da je ovaj proces jedan od učinkovitih načina popunjavanja relacijske baze podataka. Prođimo kroz najveće probleme koji se javljaju pri ovome procesu.

Prvi korak u ekstrakciji informacija je prepoznavanje imenskih entiteta (*named entity recognition* - NER), detekcija i klasifikacija svih vlastitih imena u tekstu. Instance imenskih entiteta koje se pojavljuju u tekstu zovu se spominjanja imenskih entiteta (*named entity mentions*), a svako od njih možemo klasificirati kao organizacije, ljude, mjesta, vremena ili količine.

Nakon toga, potrebno je grupirati (povezati) sva spominjanja imenskih entiteta u skupove koji odgovaraju pripadnim entitetima. Taj zadatak zove se *reference resolution.*

Izvlačenje odnosa (*relation extractionn*) pronalazi i klasificira semantičke odnose između entiteta koje smo pronašli u tekstu. Odnosi koji se mogu pojaviti su obitelj, zaposlenost, dio-cjelina, članstvo i geoprostorni odnosi. Ovaj proces najbliže odgovara problemu popunjavanja relacijske baze podataka.

Izvlačenje događaja (*event extraction*) je proces pronalaženja svih događaja u kojima pronađeni entiteti sudjeluju. Nakon toga potrebno je odrediti koja spominjanja pripadaju istom događaju.

Detekcija vremenskih izraza (*temporal expression detection*) identificira sve moguće vremenske izraze u tekstu, vrijeme na satu, dane u tjednu, mjesece, blagdane i slično.
Vremenska analiza (*temporal analysis*) mapira pronađene vremenske izraze u točne datume u kalendaru ili vremena u danu (primijetimo da se u tekstu mogu pojaviti i izrazi poput *day before yesterday*). Ta vremena se tada koriste za smještanje događaja u vrijeme.

Konačno, u mnogim tekstovima pojavljuju se situacije ili događaji koji se ponavljaju. Zadatak popunjavanja predloška (*template filling*) je pronalaženje takvih situacija u dokumentima i popunjavanje predložaka za te situacije.

Detaljnije ćemo proći kroz prvi korak izlačenja informacija, prepoznavanje imenskih entiteta. Ostali zadaci mogu se detaljnije pronaći u [4] te [7].

5.1 Prepoznavanje imenskih entiteta

Kao što smo ranije spomenuli, imenski entitet je ono što ima vlastito ime, a generički NER sustavi koji su orijentirani na vijesti najviše se usredotočuju na entitete poput osoba, mjesta i organizacija. Popis općih imenskih entiteta zajedno s pripadnom oznakom i tipom entiteta na koji se odnose dan je u *Tablici 10.*

Tip	Oznaka	Kategorije
Ljudi	PER	Pojedinci, izmišljeni likovi, male grupe
Organizacija	ORG	Kompanije, agencije, političke stranke, religiozne grupe, sportski timovi
Lokacija	LOC	Fizičke ekstenzije, planine, jezera, mora
Geopolitički entitet	GPE	Zemlje/države, provincije, okruzi
Instalacije	FAC	Mostovi, zgrade, zračne luke
Vozila	VEH	Zrakoplovi, vlakovi, automobili

Tablica 10: Popis općih imenskih entiteta

Popis entiteta od interesa ovisi o zadatku, području, vrsti teksta koji analiziramo i dr. Tako možemo imati i entitete poput oružja, proteina, umjetnina i slično.

I u ovom slučaju mogu se pojaviti dvosmislenosti, točnije dvije vrste dvosmislenosti. Prva vrsta odnosi se na situaciju kada se jedno ime može odnositi na različite entitete istoga tipa (primjerice, otac i sin mogu imati isto ime), a druga na situaciju kada se isto ime može odnositi na entitete potpuno različitih tipova (primjerice, *Washington* može biti osoba, lokacija, instalacija itd.).

5.1.1 Označavanje niza

Uobičajen pristup prepoznavanju imenskih entiteta je označavanje riječ po riječ, prolazeći kroz niz riječi. Oznake obuhvaćaju granicu i tip detektiranog imenskog entiteta. Klasifikatori su trenirani da označavaju tokene u tekstu s oznakama koje govore o prisutnosti određenih vrsta imenskih entiteta. IOB kodiranje (I - *inside*, O - *outside* , B - *beginning*) koristi se i u sintaktičkom komadanju (*syntactic chunking*).

Nakon IOB kodiranja, odabiremo skup svojstava koja želimo povezivati sa svakim tokenom. Neka uobičajena svojstva dana su u *Tablici 11.* Oblik se odnosi na *uppercase* oblike, *lowercase* oblike te oblike pisane velikim početnim slovom. Tu pripadaju i uzorci koji obuhvaćaju brojeve, interpunkciju, te neuobičajene promjene velikih i malih slova. Oblik je jedno od najvažnijih svojstava prepoznavanja imenskih entiteta, pogotovo kod sustava koji se usredotočuju na vijesti. Veliku ulogu imaju i svojstva pripadnosti u rječnik geografskih imena, prediktivnih riječi te vreće riječi.

Naravno, skup promatranih svojstava ovisi o primjeni, jeziku kojim je pisan tekst, tipu kodiranja teksta, žanru i slično. U *Tablici 12* prikazana su neka svojstva (vrsta riječi, sintaktičke oznake komada, oblik) te IOB kodiranje rečenice

> *[$_{ORG}$ American Airlines], a unit of [$_{ORG}$ AMR Corp.], immediately matched the move, spokesman [$_{PERS}$ Tim Wagner] said.*

Svojstvo	Opis
Element leksikona	Token koji označavamo
Korjenovani element leksikona	Korjenovana verzija tokena
Oblik	Ortografijski uzorak riječi
Slovni afiksi	Slova afiksa promatrane i okružujućih riječi
Vrsta riječi	Vrsta riječi
Sintatičke oznake komada	Osnovne oznake komada
Rječnik geografskih imena	Prisutnost riječi u jednom ili više popisa imenskih entiteta
Prediktivni tokeni	Prisutnost prediktivnih riječi u tekstu koji okružuje promatranu riječ
Vreća riječi/N-grama	Riječi i/ili N-grami koji se pojavljuju u okruženju riječi

Tablica 11: Često korištena svojstva u treniranju sustava za prepoznavanje imenskih entiteta

Svojstva				Oznaka
American	NNP	B_{NP}	cap	B_{ORG}
Airlines	NNPS	I_{NP}	cap	I_{ORG}
,	PUNC	O	punc	O
a	DT	B_{NP}	lower	O
unit	NN	I_{NP}	lower	O
of	IN	B_{PP}	lower	O
AMR	NNP	B_{NP}	upper	B_{ORG}
Corp.	NNP	I_{NP}	cap_punc	I_{ORG}
,	PUNC	O	punc	O
immediately	RB	B_{ADVP}	lower	O
matched	VBD	B_{VP}	lower	O
the	DT	B_{NP}	lower	O
move	NN	I_{NP}	lower	O
,	PUNC	O	punc	O
spokesman	NN	B_{NP}	lower	O
Tim	NNP	I_{NP}	cap	B_{PERS}
Wagner	NNP	I_{NP}	cap	I_{PERS}
said	VBD	B_{VP}	lower	O
.	PUNC	O	punc	O

Tablica 12: Jednostavno riječ po riječ kodiranje svojstava

5.1.2 Procjena

Česte metrike koje se koriste pri procjeni NER sustava su odziv (*recall*), preciznost (*precision*) i F_1 mjera. Odziv (R) je omjer broja točno označenih riječi i ukupnog broja riječi koje se trebaju označiti, a preciznost (P) je omjer točno označenih riječi i ukupnog broja označenih riječi. F_β mjera je težinska harmonijska sredina preciznosti i odziva:

$$F_\beta = \frac{(\beta^2 + 1)\, PR}{\beta^2 P + R},$$

gdje parametar β određuje težinu odziva. $\beta > 1$ ide u prilog odzivu, a $\beta < 1$ ide u prilog preciznosti. F_1 mjeru dobijemo tako da uvrstimo $\beta = 1$.

Literatura

[1] A. Birch, B. Haddow, R. Sennrich, *Neural Machine Translation of Rare Words with Subword Units*, https://arxiv.org/pdf/1508.07909.pdf, zadnje posjećeno 8.8.2019.

[2] P. Boersma, D. Weenink, *Praat: doing phonetics by computer*, http://www.fon.hum.uva.nl/praat/, zadnje posjećeno 13.8.2019.

[3] W.N. Francis, H. Kučera, *Brown Corpus Manual*, http://clu.uni.no/icame/manuals/BROWN/INDEX.HTM, zadnje posjećeno 3.8.2019.

[4] D. Jurafsky, J.H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Pearson Prentice Hall, 2009.

[5] N. Landsteiner, *Eliza*, https://www.masswerk.at/elizabot/, zadnje posjećeno 3.8.2019.

[6] J.B. Lovins, *Development of a stemming algorithm*, Mech. Translat. & Comp. Linguistics 11.1-2 (1968), str. 22–31.

[7] R. Mitkov, *The Oxford Handbook of Computational Linguistics*, Oxford Handbooks Series, OUP Oxford, 2004.

[8] C.D. Paice, *Another Stemmer*, SIGIR Forum 24.3 (1990), str. 56–61.

[9] M. Porter, *An algorithm for suffix stripping*, https://tartarus.org/martin/PorterStemmer/def.txt, zadnje posjećeno 8.8.2019.

[10] M. Sipser, *Introduction to the Theory of Computation*, Cengage Learning, 2005.

[11] *Syntax Tree Generator*, http://mshang.ca/syntree/, zadnje posjećeno 20.8.2019.

[12] *Tgrep & TGrep2*, https://web.stanford.edu/dept/linguistics/corpora/cas-tut-tgrep.html, zadnje posjećeno 20.8.2019.

[13] *The CMU Pronouncing Dictionary*, http://www.speech.cs.cmu.edu/cgi-bin/cmudict, zadnje posjećeno 13.8.2019.

[14] *The Open Speech Repository*, http://www.voiptroubleshooter.com/open_speech/american.html, zadnje posjećeno 13.8.2019.

[15] G. Yule, *The Study of Language*, Cambridge University Press, 2010.

Sažetak

Cilj ovoga rada je upoznavanje s osnovama obrade prirodnog jezika, točnije s analizom riječi i gramatičkih pravila koja koristimo u engleskom jeziku.

Pri pretraživanju ili filtriranju teksta služimo se regularnim izrazima dok sličnost riječi mjerimo pomoću udaljenosti uređivanja. Ulazni tekst je potrebno normalizirati, a pri predviđanju iduće riječi u nizu koristimo modele N-grama koji se procjenjuju pomoću zbunjenosti. Kod modeliranja koriste se zaglađivanje te *backoff* i interpolacija.

Drugi način komunikacije je govor, koji je potrebno digitalizirati i nakon toga obraditi. Danas postoje mnogi alati koji nam mogu pomoći u analizi zvučnih valova koji nastaju govorom. Sinteza govora može se provesti na 3 načina: lančanom sintezom, formantskom sintezom te artikularnom sintezom.

Računalni pristup sintaksi predstavljaju kontekstno-slobodne gramatikame koje implementiraju gramatička pravila. Izvod pravila prikazujemo stablima parsiranja, a problem mapiranja niza riječi u pripadno stablo parsiranja zove se sintaktičko parsiranje pri čemu koristimo CKY (Cocke-Kasami-Younger) algoritam.

Konačno, prvi korak pri procesu izvlačenju informacija je prepoznavanje imenskih entiteta. Taj dio izvlačenja informacija koristi označavanje riječi u nizu redom, uz IOB kodiranje, a pri procjeni ovakvih sustava koriste se opoziv, preciznost i F-mjera kao metrike.

Ključne riječi obrada prirodnog jezika, regularni izrazi, N-grami, govor, kontekstno-slobodna gramatika, parsiranje

Natural language processing

Summary

The aim of this paper is getting to know the basics of natural language processing, more precisely word and grammar rules analysis which we use in english language.

We use regular expressions in tasks like searching and filtering a text and we measure word similarity using edit distance. We need to normalize the input text, and during next word prediction we use N-gram models which we evaluate with perplexity. We use smoothing, backoff and interpolation during modeling.

Another way people can communicate is using speech, which needs to be digitized and processed. Today there is a plethora of tools which we can use to analyse speech sound waves. As for speech synthesis, there exist 3 different paradigms: concatenative synthesis, formant synthesis and articulatory synthesis.

Approaching syntax from computational point of view, we use context-free grammars. Syntax parsing is a problem of mapping a word sequence into a parse tree, where a parse tree can be obtained by applying grammar rules. Here we use CKY (Cocke-Kasami-Younger) algorithm.

Finally, the first step in information extraction is named entity recognition. This part of information extraction uses sequential word tagging, along with IOB encoding, and for system evaluation, we use recall, precision and F-measure.

Keywords natural language processing, regular expressions, N-grams, speech, context-free grammar, parsing

Životopis

Zovem se Ena Pribisalić i rođena sam 10. veljače 1996. godine u Osijeku.
Pohađala sam Osnovnu školu Vladimira Becića i III. gimnaziju u Osijeku. 2014. godine upisala sam sveučilišni nastavnički studij Matematike i informatike na Odjelu za matematiku u Osijeku, a godinu nakon toga sam se prebacila na preddiplomski studij Matematike koji sam završila 2017. godine s temom završnog rada „Jednodimenzionalno vođenje topline" pod mentorstvom izv. prof. dr. sc. Snježane Majstorović. Svoje obrazovanje nastavila sam na Odjelu za matematiku u Osijeku te sam 2017. godine upisala diplomski studij, smjer: Matematika i računarstvo.
Tijekom obrazovanja sam sudjelovala na brojnim natjecanjima, većinom iz matematike (osnovna škola), fizike (srednja škola) te programiranja (fakultet). 2016. godine sudjelovala sam na CERC i IEEE grupnim natjecanjima u programiranju, a na trećoj godini preddiplomskog studija bila sam demonstrator iz kolegija Strukture podataka i algoritmi. 2018. godine sudjelovala sam na GREEN konferenciji s radom „Zagrijavanje Zemljine površine", a 2019. godine sam dobila Rektorovu nagradu za seminarski rad „Model detekcije dijabetesa".