UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

Diplomski rad

Primena mašinskog prevođenja u analizi sentimenta tekstova na srpskom

Nenad Popović, 15/2010

septembar 2019.

Sadržaj

Slike

Tabele

Skraćenice

ARFF	**A**ttribute-**R**elation **F**ile **F**ormat
SVM	**S**upport-**V**ector **M**achine
API	**A**pplication **P**rogramming **I**nterface
IMDb	**I**nternet **M**ovie **D**ata**b**ase
MNB	**M**ultinomial **N**aive **B**ayes
IDF	**I**nverse **D**ocument **F**requency
TF	**T**erm **F**requency
MAP	**M**aximum **a** **p**osteriori

Poglavlje 1

Uvod

Analiza sentimenta, kao deo šire oblasti obrade prirodnih jezika, nalazi veliku primenu u modernom dobu, gde postoji izražena kultura razmene mišljenja na različitim platformama poput foruma i društvenih mreža poput Fejsbuka i Tvitera. Veliki biznisi, kompanije, politički analitičari nalaze veliku korist u tumačenju mišljenja i stavova velikog broja pojedinaca (*opipavanje pulsa javnosti*), a od interesa može biti sve od recenzija hotela do stavova o potezima inostranih državnika. Kako je manuelna analiza izuzetno spora i resursno ograničena odn. broj analiziranih tekstova je direktno proporcionalan broju uloženih radnih sati, i gotovo je nemoguće vršiti takvu analizu u realnom vremenu, automatizovana rešenja se prirodno nameću, a samim tim i oblast analize sentimenta.

Nažalost, uprkos raznolikosti jezika i formi teksta koje postoje na globalnom nivou, ova oblast ostaje gotovo ekskluzivno fokusirana na engleski jezik, koji je moderni *lingua franca*. Samim tim su i resursi fokusirani na manje jezike, specifično na srpski i njemu srodne jezike, izuzetno ograničeni. Stoga, polaznu tačku ovog rada predstavlja pokušaj da se količina dostupnih resursa na engleskom iskoristi i za analizu sentimenta tekstova na srpskom jeziku, a kroz primenu mašinskog prevođenja.

Kao što je često slučaj u ovoj oblasti, kao skupovi podataka su korišćene filmske recenzije, veći korpus na engleskom jeziku i značajno manji na srpskom jeziku, koji su zatim prevedeni korišćenjem *Microsoft*-ovih alata za mašinsko prevođenje. Za klasifikaciju je zatim korišćen alat *Weka*, uz upotrebu multinomijalnog naivnog bajesovskog klasifikatora.

Ostatak ovog rada biće organizovan na sledeći način. U poglavlju 2 osvrnućemo se na srodne radove koji pokrivaju slične postupke na drugim jezicima, kao i na radove koji su poslužili kao osnova za razvoj ovog. U poglavlju 3 će ukratko biti objašnjena metodologija iza postupka analize sentimenta, uz prikaz različitih podešavanja parametara koji mogu značajno uticati na poboljšanje klasifikacije. U poglavlju 4 ćemo detaljno prikazati korake u pripremi skupova podataka na oba jezika. Poglavlje 5 predstavlja rezultate klasifikacije uz inkrementalni proces dodavanja novih izmena u podešavanjima *Weke*. Konačno, poglavlje 6 predstavlja zaključak i predloge za budući rad.

Poglavlje 2
Srodni radovi

Ovaj rad kao osnovu koristi skup podataka koji su kreirali Batanović, Nikolić i Milosavljević (2016) [1]. U pitanju je korpus od 2523 filmske recenzije na srpskom jeziku pod nazivom *SerbMR*, koji je izbalansiran tako da su „eliminisane nebitne sistematske razlike između klasa sentimenata" poput razlika u broju tokena/reči u recenziji, razlika u broju recenzija u svakoj od klasa (pozitivne, neutralne, negativne) itd. Takođe, upotrebićemo i sličan princip inkrementalnih izmena podešavanja tokom klasifikacije kao u spomenutom radu. Od značaja je i dobijeni rezultat od ∼84% tačnosti prilikom uprosečene unakrsne validacije, uz korišćenje tri različita klasifikatora. Mi ćemo iskoristiti binarnu varijantu njihovog skupa podataka (*SerbMR-2C*), koja sadrži 841 pozitivnu i 841 negativnu recenziju.

Korpus filmskih recenzija na engleskom (ukupno 50000 njih) pod nazivom *Large Movie Review Dataset* je delo Maas et al. (2011) [2]. Recenzije su pribavljene sa sajta *IMDb*, te ih stoga odlikuje raznolikost po brojnim faktorima, uključujući i upotrebu neformalnog jezika. Ceo skup podataka uključuje dvostruko više recenzija, međutim, nama su od interesa isključivo prethodno klasifikovane.

Radovi koji kombinuju mašinsko prevođenje i analizu sentimenta su brojni:

- Balahur i Turchi (2012) su iskoristili tri različita sistema za prevođenje (*Bing, Google, Moses*) i preveli originalni engleski korpus na tri različita jezika (španski, nemački, francuski), da bi zatim pokušali binarnu klasifikaciju. Dobijeni su rezultati uporedivi sa „zlatnim standardom" klasifikacije na engleskom [3]. Autori su pokušali i da iskombinuju sva tri prevedena korpusa u jedan, a korišćenje takvog kombinovanog korpusa je dovelo do pada performansi zbog povećanja šuma. Bitno je spomenuti da su i skup za obučavanje i skup za testiranje bili prevođeni, i to sa istog jezika, pa ovo nije „pravi primer" međujezične analize sentimenta kao u našem slučaju.
- Shalunts et al. (2016) su koristili pristup zasnovan na leksikonima (rečnicima sentimenata) korišćenjem alata *SentiSAIL*, kao i alat za mašinsko prevođenje *SDL Language Weaver*, koji su u tom trenutku predstavljali *state-of-the-art* u ovoj oblasti. Prevođenjem korpusa na engleski, dobijeni su rezultati uporedivi sa uprosečenim slaganjem između više anotatora. Autori sugerišu da je prevođenje korpusa na engleski jezik i korišćenje analize sentimenta na

engleskom umesto međujezične varijante poželjna strategija, posebno „kada je potrebno uzeti u obzir odnos između troškova kreiranja resursa specifičnih za dati jezik i troškova mašinskog prevođenja" [4]. Vredi primetiti da su jezici korišćeni u ovom radu dosta zastupljeniji globalno u odnosu na srpski (španski, ruski, nemački jezik), te iako nemaju resurse uporedive sa engleskim jezikom, i dalje imaju značajnu prednost u poređenju sa manjim jezicima.

- Balamurali et al. (2013) su pokušali da pokažu kako je korišćenje resursa na originalnom jeziku za obučavanje uvek efikasnije od korišćenja prevođenih resursa, suprotno mnogim sličnim radovima [5]. Pokazali su da je dovoljan broj dokumenata za obučavanje oko 400-500, i da se tu postiže zasićenje. Koristili su četiri različita algoritma - direktan mašinski prevod, bidirekcioni rečnik, kao i kombinacije korpusa prevedenih sa dva ili tri jezika, i u svim slučajevima se pokazalo da su rezultati dobijeni korišćenjem originalnog korpusa superiorni. Primećujemo da je korišćen samo jedan sistem za mašinsko prevođenje (*Microsoft/Bing*), kao i samo jedan klasifikator (SVM - *support-vector machine*), te ne možemo generalizovati njihov zaključak, uprkos opsežnim eksperimentima. Autori se takođe osvrću i na jezike koje su koristili, slično gorenavedenom radu - francuski, nemački, ruski, engleski - koji su kulturno i komercijalno zastupljeniji, te za druge, manje zastupljene jezike, očekuju još lošije performanse. Rezultati ovog rada će nam biti od interesa, s obzirom na vrlo sličnu metodologiju.

- Araujo et al. (2016) su ispitivali analizu sentimenta na nivou rečenica za 21 različit metod i 9 različitih jezika. Njihov zaključak je da mašinsko prevođenje dovodi do pada performansi u poređenju sa klasifikacijom nad engleskim skupovima podataka, ali i da mašinsko prevođenje može biti dobra strategija ukoliko se napravi dobar izbor metode za analizu sentimenta. U dva slučaja su čak pokazali da je rezultat dobijen korišćenjem mašinskih prevoda uporediv sa dva metoda koja su specifična za date jezike [6].

- Barhoumi et al. (2018) su na primeru arapskog jezika pokazali da je analiza sentimenta engleskog prevoda bolja u poređenju sa neobrađenim arapskim tekstom, i uporediva sa arapskim tekstom koji je lako stemovan (*light stemming*), što podrazumeva svođenje arapskih reči na redukovanu verziju bez prefiksa i sufiksa. Autori predlažu kombinaciju lakog stemovanja i mašinskog prevoda, kao i buduću upotrebu *deep learning* klasifikatora [7].

Poglavlje 3

Metodologija rada

U ovom poglavlju ćemo detaljno opisati najvažnije pojmove koje ćemo nadalje koristiti, koncepte specifično povezane sa načinom na koji ćemo izvršiti analizu sentimenta, kao i alate koji su korišćeni tom prilikom.

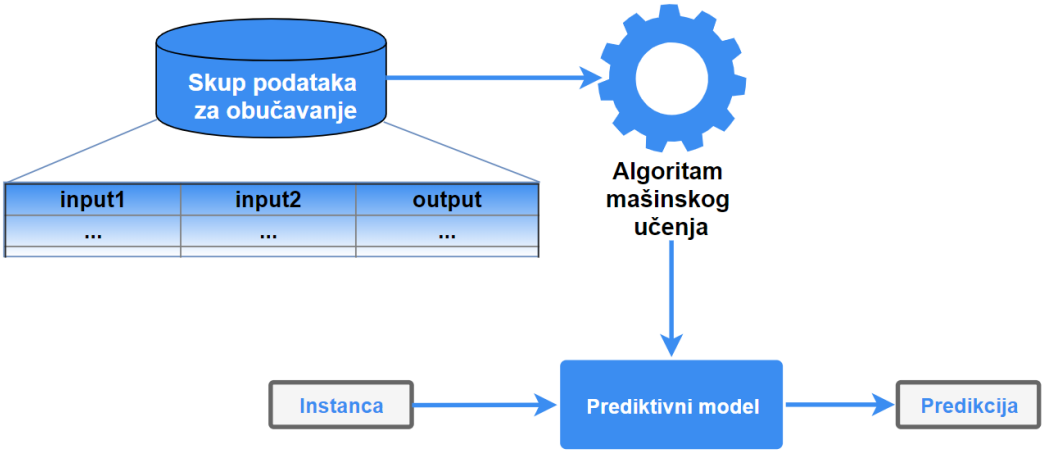
3.1 Binarna klasifikacija i analiza sentimenta

U oblasti mašinskog učenja, razlikujemo nekoliko tipova učenja, gde će nam od interesa biti nadgledano učenje (*supervised learning*). Ono podrazumeva generalizaciju preslikavanja iz ulaznog u izlazni podatak kada je priložen veliki broj konkretnih instanci (primera) takvog preslikavanja. Mogu postojati različite kombinacije ulaznih i izlaznih podataka, poput preslikavanja više zdravstvenih parametara u dužinu života pacijenta, slika različitih životinja u vrstu kojoj pripadaju, originalnih tekstova u njihov mašinski prevod... Ukoliko je izlazni podatak neprekidna vrednost, tada govorimo o problemu regresije, a kada je u pitanju diskretan skup vrednosti, tada govorimo o klasifikaciji. Binarna klasifikacija je specifičan slučaj klasifikacije gde postoje samo dve klase [8].

Rezultat obučavanja je prediktivni model ili funkcija koju dalje možemo koristiti za predikcije klasa za svaku datu instancu u budućnosti. Na slici 3.1 je dat jednostavan prikaz opisanog procesa[1].

Analiza sentimenta, prema prethodnim definicijama, predstavlja samo varijantu klasifikacije (binarne ili višeklasne) gde ulazni podatak predstavlja tekst određene dužine (rečenica ili dokument), dok izlaz predstavlja klasa ili kategorija kojoj pripada dati tekst [10]. U ovom radu ćemo vršiti jednostavnu binarnu klasifikaciju filmskih recenzija na „pozitivne" ili „negativne", a česta je i trinarna klasifikacija, gde dodajemo klasu „neutralnih" recenzija.

[1] Slika prilagođena iz [9]



Slika 3.1: Formiranje i upotreba prediktivnog modela

3.2 Odlike i *bag-of-words* reprezentacija teksta

Svaka ulazna instanca se može okarakterisati svojim vrednostima na predefinisanom konačnom skupu atributa ili odlika [11]. U ranije spomenutom primeru zdravstvenih parametara nekog pacijenta, odlike mogu biti npr. vrednosti sistolnog i dijastolnog pritiska, da li je pacijent pušač ili ne, spisak bolesti... U tabelarnom prikazu, svaka odlika mogla bi biti predstavljena jednom kolonom, a svaka instanca jednom vrstom koja sadrži vrednosti za zadatu odliku[2]. Praktično je svaka instanca jedna torka vrednosti odlika, $(v_1, v_2, ..., v_n)$ na skupu odlika $(f_1, f_2, ..., f_n)$.

U našem slučaju, ulazne instance date su u formi teksta proizvoljne dužine. Vredi diskutovati na koji način se takav ulaz može dekomponovati na skup odlika. Jedan pokušaj bi uzimanje samog teksta kao jedinstvene odlike. Mana takvog pristupa je očigledna - ne postoji mogućnost učenja jer je svaka recenzija jedinstvena, te je nemoguće generalizovati. S druge strane, uzimanje konačnog broja odlika tipa „vrednost karaktera na i-toj poziciji" bi dovelo do kombinatorne eksplozije zbog potencijalno velikog broja dimenzija torke, uz veliki broj mogućnosti za vrednost svakog pojedinačnog karaktera. Dakle, naš idealan izbor se krije „negde između", na nivou reči ili rečenice.

[2]Baza podataka se prirodno nameće kao način čuvanja takvih skupova vrednosti odlika

Međutim, i na tom nivou postoji mogućnost za veliku složenost. Uzimanje n odlika u formi „reč na poziciji i", gde je n maksimalni broj reči u nekom tekstu iz datog skupa, bi ponovo bilo primer dekompozicije koja bi se loše generalizovala, jer bi npr. dodavanje samo jedne reči u recenziji napravilo značajnu razliku u reprezentaciji između dve gotovo identične recenzije. [3]

U literaturi je poznat jednostavan način dekompozicije teksta na odlike na nivou reči gde zanemarujemo njihovu poziciju, već samo uzimamo njihovo prisustvo u tekstu kao bitnu stavku. Ova reprezentacija se naziva *bag-of-words* (torba reči), po analogiji gde smo reči ubacili u torbu i promućkali, te je njihov redosled izgubljen [10]. Prema ovoj reprezentaciji, torka bi mogla biti formirana na sledeći način: prolaskom kroz sve dokumente u datom skupu, formiramo rečnik svih reči, i zatim za svaku reč kreiramo po jednu odliku. Vrednost odlike može biti binarna, da li reč postoji u datom dokumentu ili ne, ili broj pojavljivanja date reči u datom dokumentu, i oba tipa vrednosti mogu biti korisna u različitim kontekstima. Uprkos jednostavnosti ovakve postavke, ona je veoma praktična i nailazi na masovnu primenu, a biće korišćena i u ovom radu.

3.3 Multinomijalni naivni bajesovski klasifikator

Problem klasifikacije dokumenta može se prevesti u sledeću matematičku formu - ukoliko događaj „dokument je D" označimo sa d, a događaj „klasa je C" sa c, klasifikaciju vršimo prema događaju „ako je dokument D, klasa je C" koji je najverovatniji. Ovo je *maximum a posteriori*, ili MAP odlučivanje [9]. Odgovarajuću uslovnu verovatnoću možemo označiti sa $P(c \mid d)$.

Prema Bajesovoj teoremi, ovu verovatnoću je moguće izračunati kao

$$P(c \mid d) = P(d \mid c) \cdot \frac{P(c)}{P(d)}$$

Kada želimo da izračunamo i uporedimo sve uslovne verovatnoće klasa za neki dokument, vrednost imenioca $P(d)$ je ista, a u slučaju izbalansiranog skupa podataka i vrednost $P(c)$, te se poređenje tih verovatnoća se svodi na poređenje između vrednosti $P(d \mid c)$.

[3] Sve reči nakon pozicije umetanja u originalnoj i novodobijenoj recenziji bi se međusobno razlikovale, te ne bi bilo moguće uočiti njihovu sličnost posmatranjem pojedinačnih odlika.

Međutim, šta predstavlja događaj opisan datim izrazom? Ukoliko je dati dokument prisutan u skupu dokumenata sa datom klasom, ta verovatnoća će biti $1/n$, gde je n broj dokumenata u datoj klasi, u suprotnom 0. Ovo nije naročito informativan podatak, i ovakav klasifikator će umeti da klasifikuje isključivo dokumente koje je imao u skupu za obučavanje, a svi drugi dokumenti će imati nulte verovatnoće. Očigledno, potrebno je predstaviti dokument na neki drugi način, i kao prirodna reprezentacija se nameću njegove odabrane odlike. Umesto verovatnoće $P(d \mid c)$ ćemo pokušati da izračunamo $P(v_1, v_2, ..., v_n \mid c)$ na skupu odlika $(f_1, f_2, ..., f_n)$, gde v_i označava događaj „odlika f_i ima vrednost v_i".

U podpoglavlju 3.2 smo predstavili bag-of-words reprezentaciju dokumenta koju možemo iskoristiti kao skup odlika u ovom slučaju. U ovom trenutku je potrebno uvesti *naivnu pretpostavku o nezavisnosti odlika*, odn. pretpostavku da je pojavljivanje jedne reči u dokumentu potpuno nezavisno od pojavljivanja neke druge reči.[4] U ovom slučaju, po definiciji važi

$$P(v_1, v_2, ..., v_n \mid c) \sim P(v_1 \mid c) \cdot P(v_2 \mid c) ... \cdot P(v_n \mid c)$$

Konačno, sveli smo klasifikaciju na događaje oblika „kada je vrednost klase C, reč w_i se pojavljuje u dokumentu". Dosadašnja postavka predstavlja opis *naivnog bajesovskog klasifikatora*. U zavisnosti od predviđene raspodele vrednosti odlika odn. načina izračunavanja faktora $P(v_i \mid c)$ zavisi o kakvom klasifikatoru govorimo. U našem radu će dokument biti tretiran kao skup reči određene dužine kreiran multinomijalnim izborom nad fiksiranim rečnikom [12]. Stoga možemo govoriti o *multinomijalnom naivnom bajesovskom klasifikatoru* (MNB). Uvedimo sledeće oznake: N_i će predstavljati broj pojavljivanja i-te reči u dokumentima sa određenom klasom, n će predstavljati veličinu rečnika, a N ukupan broj reči u dokumentima određene klase. U skladu sa multinomijalnom pretpostavkom, važi sledeća formula

$$P(d \mid c) \sim N! \cdot \prod_{i=1}^{n} \frac{P(v_i \mid c)^{N_i}}{N_i!}$$

Faktor $P(v_i \mid c)$ može se izračunati kao broj pojavljivanja date reči u dokumentima klase C podeljen sa ukupnim brojem reči u tim dokumentima.

[4] Ova pretpostavka očito ne odgovara realističnim situacijama kao npr. većoj učestalosti zajedničkog pojavljivanja reči „voda" i „brašno" u receptima (odatle i naziv „naivna"), ali nam njena jednostavnost omogućava značajne optimizacije

Nažalost, ovaj način izračunavanja je problematičan kada je vrednost nekog od pojedinačnih činilaca nula - npr. ukoliko je „užasno" jedna od reči u rečniku prediktivnog modela i ona se uopšte ne pojavljuje u pozitivnim recenzijama, verovatnoća $P(užasno \mid pozitivna)$ će biti nula, te će gornji proizvod biti jednak nuli za sve recenzije koje ne sadrže tu reč![5] Jednostavno rešenje je Laplasovo poravnanje, gde broju pojavljivanja za svaku reč dodajemo 1. Konačan izraz za faktore glasi

$$P(v_i \mid c) = \frac{N_i + 1}{\sum_{i=1}^{n}(N_i + 1)}$$

Na ovaj način smo obezbedili nenulte verovatnoće koje će biti dodeljene svakoj reči u prediktivnom modelu. Zahvaljujući ovome, moguće je izvesti još jednu optimizaciju - naime, kako su verovatnoće u opsegu $[0, 1]$, množenje velikog broja takvih činilaca može dovesti do vrednosti suviše malih za predstavljanje u računarima (*underflow*). S druge strane, faktorijeli mogu imati ogromne vrednosti koje mogu dovesti do prekoračenja za cele brojeve (*integer overflow*). Zbog ovih problema, pogodno je koristiti logaritamski oblik gornjeg proizvoda[6], te je, uz korišćenje Stirlingove aproksimacije za faktorijele, konačan oblik naše funkcije procene

$$f(d, c) \sim \sum_{i=1}^{n} N_i \cdot \log P(v_i \mid c) + N \log N - \sum_{i=1}^{n} N_i \log N_i$$

Kako logaritamska funkcija raste na domenu verovatnoća $[0, 1]$, prikazana suma će rasti na isti način kao i originalni proizvod, te će poređenje suma prilikom odlučivanja o klasi dokumenta dati isti rezultat kao da smo poredili proizvode.

Umesto pojedinačnih reči, postoji mogućnost da uzmemo nekoliko susednih reči u pokušaju da proširimo prediktivni model. Na primer, „nije dobar" očito indikuje negativan sentiment više nego reči „nije" i „dobar" zasebno. U tom slučaju, možemo govoriti o n-gramskim odlikama. Dakle, jedna reč bi predstavljala unigram, dve bigram itd. Nema značajne izmena u pretpostavkama za MNB klasifikator kada koristimo ove odlike, a kao što je ilustrovano na primeru, njihova upotreba nam može biti od pomoći.

[5] Primer prilagođen iz [10]

[6] Vrednost logaritama za jako male brojeve u domenu $[0, 1]$ ima veliku apsolutnu vrednost.

3.4 Stemovanje

U oblasti obrade prirodnih jezika, korisna je primena postupka lematizacije koji podrazumeva svođenje sličnih reči poput „sedeti", „sedi", „sedeo" na zajednički oblik, kako bi se na taj način grupisale odn. tretirale kao ista reč prilikom obrade. Lematizacija kao postupak zahteva složene transformacije i poznavanje originalnog jezika kako bi se sve gorenavedene reči projektovale u originalni oblik „sedeti", odn. u oblik u kojem bi se reč pojavila u rečniku. Iz tog razloga, koristi se pojednostavljena forma lematizacije pod nazivom stemovanje, gde samo uklanjamo sufiks bez poznavanja originalne strukture reči, i svodimo ih na zajednički koren ili neki sličan oblik [9]. Nama su od interesa stemeri na srpskom i engleskom jeziku, te sledi opis stemera koji su korišćeni u ovom radu.

Stemer koji će najviše biti od koristi za engleski jezik je Porterov stemer, koji je delo Martina Portera, dostupan još od 1979 [13]. Korišćeni algoritam se svodi na jednostavna pravila oblika $(Uslov)\ S_1 \longrightarrow S_2$, gde je S_1 neki sufiks, a S_2 transformisani oblik reči.[7] Ovakva pravila se primenjuju u nekoliko koraka, pa jedna reč može biti transformisana više puta. I sam autor predlaže korišćenje unapređene verzije ovog stemera koja je poznata pod nazivom *Snowball* stemer ili *English (Porter2)* stemer.[8] Mi smo koristili obe verzije ovog stemera.

Što se tiče srpskog jezika, korišćeni su identični stemeri kao u [1], a to su dva stemera koje koje su kreirali Kešelj i Šipka (2008) [14] (*greedy* i optimalni), stemer koji je kreirao Milošević (2012) [15], kao i stemer za hrvatski jezik koji su kreirali Ljubešić i Pandžić, baziran na radu Ljubešića et al. (2007) [16]. Korišćene su prilagođene Java verzije koje je pripremio Vuk Batanović.[9]

[7] Koristili smo Java verziju dostupnu na sledećem linku: https://tartarus.org/martin/PorterStemmer/ (posećen 2019-09-19)

[8] Koristili smo Java verziju prilagođenu iz kôda dostupnog na sledećem linku: http://snowball.tartarus.org/download.html (posećen 2019-09-19)

[9] Dostupne na sledećem linku: https://github.com/vukbatanovic/SCStemmers (posećen 2019-09-19)

3.5 *Weka* i njena podešavanja

Najznačajniji alat koji ćemo koristiti u ovom radu je *Weka*, koji predstavlja *open source* kolekciju algoritama za mašinsko učenje koji se primenjuju u *data mining*-u.[10] Konkretnije, sadrži sve alate potrebne za binarnu klasifikaciju koja nam je od interesa, kao i podršku za sve prethodno opisane postupke. Verzija *Weka* alata koju smo koristili je 3.6.15.

Weka omogućuje proširivanje svojih mogućnosti dodavanjem odgovarajućih *Java* klasa, što nam omogućuje uporebu stemera koje smo spomenuli u podpoglavlju 3.4. Da bi to bilo moguće, odgovarajuće *Java* klase su dodate u izvorni kod *weka-src.jar*, i zatim je osnovna datoteka *weka.jar* ponovo napravljena. Modifikovana instanca *weka.jar* koja sadrži stemere, kao i njen modifikovani izvorni kod, date su u prilogu „Weka.zip" uz ovaj rad.[11]. Potrebno je samo zameniti lokalnu prisutnu *weka.jar* datoteku modifikovanom verzijom.

Kako bi skupovi podataka bilu upotrebljivi u alatu *Weka*, potrebno je konvertovati ih u fajl specifičnog formata pod nazivom ARFF (*Attribute-Relation File Format*), koji uparuje atribut i njegovu klasu, u našem konkretnom slučaju pojedinačnu recenziju i njoj dodeljeni sentiment (pozitivna/negativna). Struktura takvog fajla opisana je u zaglavlju koje predstavlja tipove instanci i klasa, koje su zatim praćene nizom primera ulaznih i izlaznih podataka, u našem slučaju stringova recenzija i celobrojnog tipa klase. Primer jednog konkretnog zaglavlja dat je na slici 3.2.

```
@relation 'EngMR_large'

@attribute Text string
@attribute class-att {'POSITIVE','NEGATIVE'}

@data

'Example review text', POSITIVE
...
```

Slika 3.2: Primer ARFF zaglavlja

[10] https://www.cs.waikato.ac.nz/ ml/weka/index.html

[11] U kasnijim verzijama alata *Weka* ($\geq$ 3.7) moguće je koristiti *package manager* za ovakve modifikacije.

Weka nam omogućava upotrebu MNB klasifikatora u kombinaciji sa *bag-of-words* reprezentacijom na sledeći način:

1. Nakon učitavanja nekog skupa za obučavanje, možemo otići pod karticu *Classify*, gde možemo izvršiti izbor klasifikatora klikom na dugme *Choose*.
2. Biramo opciju *meta* > *FilteredClassifier*. Ovo predstavlja složeni klasifikator koji nam dozvoljava da primenimo filter koji transformiše tekst recenzije u *bag-of-words* reprezentaciju.
3. U konfiguraciji dobijenog metaklasifikatora dobijamo mogućnost izbora klasifikatora i filtera. Izborom klasifikatora *bayes* > *NaiveBayesMultinomial* dobijamo MNB klasifikator. Izborom filtera *unsupervised* > *attribute* > *StringToWordVector* dobijamo *bag-of-words*.
4. Postavljanjem željenog skupa za testiranje pod *Test options* > *Supplied test set* završavamo eksperimentalnu postavku.

Osvrnimo se i na najbitnija podešavanja koja su dostupna za *StringToWordVector* filter.

wordsToKeep - verovatno jedna od najvažnijih postavki, podešavanje ovog celobrojnog parametra određuje veličinu rečnika našeg modela. Može se odnositi na broj reči po klasi ili ukupan broj reči, u zavisnosti od drugih podešavanja.

stemmer - omogućava izbor stemera koji će biti primenjen na pojedinačne reči

lowerCaseTokens - pretvara velika u mala slova u rečima, što je korisno jer se reči sa velikim slovom tretiraju drugačije od onih gde su sva mala, što često nije poželjno

tokenizer - omogućava izbor načina na koji se mogu dobiti pojedinačne odlike, najčešće podelom na reči ili n-grame, uz izbor karaktera koji ih odvajaju.

minTermFreq - minimalni broj pojavljivanja neke odlike; praktično, ovo znači da reči koje se pojavljuju broj puta koji je manji od ove vrednosti neće biti uzimane u obzir

outputWordCounts - ova vrednost odlučuje da li ćemo tretirati rečnik kao skup (vrednost odlika će biti 0 ili 1), ili kao multiskup gde će se kao vrednost odlike uzimati broj pojavljivanja reči

doNotOperateOnPerClassBasis - ova vrednost odlučuje da li se *minTermFreq* i *wordsToKeep* odnose na klase ili na ceo dokument

normalizeDocLength - ova vrednost nam govori da li će frekvencije reči biti normalizovane u skladu sa prosečnom dužinom dokumenta u skupu za obučavanje, i to da li ćemo to učiniti za oba skupa ili samo za skup za testiranje

IDFTransform - IDF u ovom slučaju znači *inverse document frequency*, što predstavlja primenu heuristike prema kojoj su reči koje su ređe u svim dokumentima korisnije pri klasifikaciji od onih koje su „generičke" i češće se ponavljaju. Primer je član *a* u engleskom jeziku koji ne nosi u sebi nikakav sentiment. *Weka* transformiše frekvenciju (broj pojavljivanja) reči u dokumentu na sledeći način

$$idf(i,j) = f_{ij} \cdot \log \frac{N}{N_i}$$

gde je f_{ij} originalna frekvencija reči i u dokumentu j, N ukupan broj dokumenata, a Ni broj dokumenata gde se pojavljuje reč i,

TFTransform - TF predstavlja *term frequency*, gde ponovo koristimo heuristiku koja umanjuje značaj rečima koje se izuzetno često pojavljuju, ali na nivou dokumenta. Weka transformiše frekvenciju reči u dokumentu na sledeći način

$$tf(i,j) = \log(1 + f_{ij})$$

gde je f_{ij} ponovo frekvencija reči i u dokumentu j.

Poslednje dve stavke se često koriste u kombinaciji pod nazivom *tf-idf*.

3.6 Ostali alati

- *Visual Studio 2015*, za *build* i kodiranje programa iz priloga B.1.
- *Python 2.7* i njegovo okruženje *IDLE*, za izvršavanje svih ostalih programa u prilogu B, kao i izvršavanje programa spomentih u podpoglavlju 4.1.2
- *Eclipse* okruženje za *build* modifikovanih *Weka* klasa, kao i obradu negacije
- *Notepad++*, za jednostavno editovanje velikih ARFF fajlova i programa
- *LaTeX* i *Overleaf*[12] za izradu samog rada

[12]https://www.overleaf.com/

3.7 Konačna postavka eksperimenata

Naši eksperimenti biće izvršavani na oba jezika, engleskom i srpskom. Kao skup za obučavanje biće korišćen veći korpus recenzija, originalno na engleskom jeziku, koji sadrži 50000 recenzija jednako podeljenih na pozitivnu i negativnu klasu. Kao skup za testiranje biće korišćen manji korpus recenzija, originalno na srpskom jeziku, od 1682 recenzije jednako podeljene na pozitivnu i negativnu klasu.

Eksperimente ćemo izvršavati tako što ćemo uparivati prevedeni engleski korpus sa originalnim srpskim korpusom, i obrnuto, uz potpunu nezavisnost između dva skupa eksperimenata.

Kako je skup za testiranje balansiran, ima smisla kao metriku uspešnosti uzeti *tačnost*, koja predstavlja odnos između ispravno klasifikovanih instanci pri testiranju i ukupnog broja instanci (1682), izraženu u procentima.

Započinjaćemo učitavanjem većeg korpusa u alatu *Weka*, da bismo zatim prešli na izbor klasifikatora koji je opisan u podpoglavlju 3.5. Najbitnije korake će predstavljati izbor i kombinovanje već opisanih opcija u *StringToWordVector* filteru koji mogu uticati na konačnu tačnost, u cilju poboljšanja. Koristićemo iterativni proces sličan kao u [1], uz tu razliku što nemamo više klasifikatora, pa ćemo kombinovati neke od opcija kako bismo mogli da vidimo njihov uticaj u više primera, te birati ona podešavanja koja daju pozitivan/maksimalan učinak.

Kao referentne vrednosti za tačnost ćemo uzimati vrednosti dobijene pri 10-slojnoj unakrsnoj validaciji na oba korpusa (koji pokazuju kako bi se prediktivni model ponašao na sličnim podacima kao skup za obučavanje), kao i najbolje dobijene rezultate iz [1], koji iznose 84.44% i značajni su jer su dobijeni isključivo korišćenjem originalnog korpusa.

Poglavlje 4

Priprema skupova podataka

U ovom poglavlju ćemo se osvrnuti na pripremu skupova podataka na engleskom i srpskom jeziku, koja je podrazumevala mašinsko prevođenje oba originalna skupa, kao i dodatna prilagođavanja usled grešaka, dupliranja, itd. Osvrnućemo se na specifičnosti na koje smo nailazili za svaki jezik pojedinačno. Vredi spomenuti da su prevedene sve recenzije koje su bile dostupne u oba korpusa, uključujući i one koje nisu korišćene u ovom radu (neutralne ili naprosto nelabelirane), kako bi bile šire dostupne za dalja istraživanja.

Za prevođenje je korišćeno *Microsoft*-ovo rešenje dostupno na *Azure* platformi[1]. Pod sekcijom *Cognitive Services* moguće je pronaći *Translator Text API*[2] koji nam omogućava da prevedemo proizvoljan tekst sa engleskog na srpski i obrnuto. Od koristi je bio i alat *Microsoft Document Translator*[3], koji značajno pojednostavljuje korišćenje zadatog API-ja kroz grafički interfejs. Ukupna prevedena količina teksta bila je približno 220 miliona karaktera na engleskom, i 23 miliona karaktera na srpskom jeziku.

Značajno je spomenuti da je dole opisani proces prevođenja izveden pre dve godine, te je moguće diskutovati o napretku alata za mašinsko prevođenje u međuvremenu. Nažalost, kao što se može videti u prilogu A, koji sadrži jednu od recenzija na engleskom i dva različita mašinska prevoda na srpski jezik u razmaku od par godina, progres postoji, ali je prevod i dalje daleko od ispravne strukture i leksike srpskog jezika. Naravno, predlažemo korišćenje novijih prevoda u budućem istraživanju.

[1] https://azure.microsoft.com
[2] https://azure.microsoft.com/en-us/services/cognitive-services/translator-text-api/
[3] https://github.com/MicrosoftTranslator/DocumentTranslator/

4.1 Priprema engleskog korpusa

Kao što smo spomenuli u poglavlju 2, veći skup podataka na engleskom jeziku je predstavljao 50000 klasifikovanih filmskih recenzija sa sajta *IMDb* [2], sa podjednakim brojem pozitivnih i negativnih recenzija. Skup podataka uključivao je i dodatnih 50000 nelabeliranih recenzija, koje su takođe prevedene. Recenzije su bile priložene u pojedinačnim tekstualnim fajlovima, te je strukturirani pristup gde su dokumenti prevođeni u grupama od 2500 kroz *Microsoft Document Translator* i održavanje iste strukture i imena fajlova pomoglo lakšem praćenju kroz svaki korak. Problem su predstavljale i linije teksta duže od 5000 karaktera[4], gde bi se alat „zaglavljivao" bez ikakve notifikacije, te su takvi slučajevi usporavali prevođenje tako velikih grupa fajlova i zahtevali posebnu pažnju i česta ponovna pokretanja procesa. Nakon inicijalnog prevođenja, dobijen je skup prevoda priložen uz ovaj rad (videti „ENG-SER 01 - Inicijalni prevod"). Taj prevod sadrži više problema, uključujući izvesnu količinu neispravnih karaktera (najčešće ćiriličnih u latiničnom tekstu) i veliki broj reči bez ispravnih dijakritika.

4.1.1 Uklanjanje neispravnih/ćiriličnih karaktera

Prvi problem kojim smo se pozabavili bilo je uklanjanje ćiriličnih karaktera koji su bili ubačeni umesto ispravnih latiničnih karaktera sa dijakriticima poput š, ć itd. U tabeli 4.1 se mogu videti sve zamene karaktera koje su ispravljane, koje imaju pandan i za velika slova. Ovaj problem je rešen rudimentarnim programom pod imenom „Manual Editor", datim u prilogu B.1. Glavna funkcija u tom programu, *getNextWord*, se bavi pojedinačnim rečima u kojima primećujemo „čudne" *Unicode* karaktere (sa celobrojnom vrednošću preko 1000) koji suštinski predstavljaju već spomenute ćirilične karaktere. Ukoliko se takav karakter pojavi, program bi korisniku ponudio da ručno popravi datu reč. Nakon nekoliko pokretanja programa, izdvojili smo često primećene karaktere i reči kao slučajeve koji se automatski razrešavaju, što se može jasno videti u samoj funkciji. Naravno, teško je proveriti da su baš sve greške ovog tipa ispravljene, međutim, proverom velikog broja nasumičnih fajlova nisu više uočena ovakva odstupanja. Dobijeni tekstovi su priloženi u „ENG-SER 02 - Izbačeni ćirilični karakteri".

[4]Autori alata su nedavno ipak rešili ovaj bag.

Pogrešan	**Ispravan**
љ	š
ћ	ž
ж	ć
и	č
р	đ

Tabela 4.1: Karakteri dobijeni u prevodu sa engleskog i njihove ispravne verzije

4.1.2 Vraćanje dijakritika

Problem vraćanja nedostajućih dijakritika u pojedinačnim rečima uspešno je rešen korišćenjem alata koji su razvili Ljubešić, Erjavec, Fišer (2016) - prvog koji tokenizuje srpski/hrvatski tekst [17], i drugog koji vrši restauraciju dijakritika sa preko 99% tačnosti [18]. Rezultati nakon primene oba alata dati su priloženi uz ovaj rad kao „ENG-SER 03 - Tokenizovani skup podataka" i „ENG-SER 04 - Restaurirani dijakritici". Kako je rezultat nakon korišćenja ovih alata tokenizovana verzija dokumenta, bilo je potrebno izvršiti rekonstrukciju natrag u običan tekst. To je postignuto korišćenjem *Python* programa *reconstruct.py* (videti prilog B.2). Rezultat nakon rekonstrukcije priložen je u „ENG-SER 05 - Rekonstruisani prevodi".

4.1.3 Uklanjanje tagova

Nakon prethodnog koraka, primećeno je da veliki broj recenzija sadrži HTML tagove za novu liniju (
). Kako je postupak prevođenja već bio pri kraju, bilo je potrebno ukloniti ih i u originalnom i u prevedenom korpusu, iako su mogli biti izbačeni već u prvom koraku. To je učinjeno korišćenjem kratkog programa *remove_ br.py* (videti prilog B.3), uz manje modifikacije za oba korpusa. Rezultat za prevedeni korpus je priložen u „ENG-SER 06 - Konačni prevedeni skup".

4.1.4 Kreiranje ARFF fajla

U prilogu B.4 je dat program koji smo pokrenuli kako bismo spojili sve recenzije u jedan fajl. Kako ARFF zahteva da stringovi budu pod jednostrukim (apostrof) ili dvostrukim navodnicima, bilo je potrebno svim apostrofima dodati *escape* karakter kako bi bili validni unutar stringa okruženog apostrofima. Nažalost, primećen je i izvestan broj dupliranih recenzija u originalnom engleskom korpusu kojima smo morali da se pozabavimo. Kreirana su dva fajla - jedan sačinjen isključivo od recenzija koje su originalno služile za obučavanje, njih 25000, za koji su autori rekli da su veoma polarizovane [2], i drugi fajl sačinjen od recenzija i za obučavanje i za testiranje, ukupno 50000. Ovo je urađeno za oba jezika, i recenzije su sortirane kako bi se lakše poredile pri traženju duplikata. Kako bi se dobio ispravan ARFF format, bilo je potrebno dodati i odgovarajuće opisno zaglavlje o tipu atributa i klasama (videti 3.2).

Iz originalnog i prevedenog engleskog korpusa je uspešno uklonjeno oko 400 recenzija koje su bile duplikati. Kako su, kao i u prethodnom slučaju, duplikati primećeni daleko u procesu, i kako broj pronađenih duplikata nije bio identičan za oba korpusa, bilo je potrebno obaviti još jedan prolaz kroz njih. Koristili smo naivan pristup gde smo poredili nekoliko prvih desetina karaktera kao indikator sličnosti, računajući da se izuzetno slične recenzije poklapaju barem na početku. Označene slične recenzije su zatim ručno proveravane i izbačene iz korpusa. U prilogu B.5 se može naći program koji smo koristili u ovom slučaju.

Konačno, dobijena su četiri fajla: *EngMR_large* i *EngMR_large_SR* (49548 recenzija), *EngMR_small* i *EngMR_small_SR* (24887 recenzija). Manji korpus sadrži polarizovanije recenzije, ali smo se na kraju odlučili za veći korpus kako bismo dobili kvantitativnu prednost. Fajlovi su priloženi uz ovaj rad.

4.2 Priprema srpskog korpusa

Prevođenje srpskog korpusa na engleski jezik nije predstavljalo ozbiljan problem - originalni korpus je već bio veoma strukturiran, kao i sam ARFF fajl SerbMR-3C koji smo primarno koristili. Problem za *Microsoft Document Translator* su ponovo predstavljale duge recenzije, ali i njihovim filtriranjem je bilo moguće uspešno završiti proces. Najčešće rešenje bilo je prelamanje izuzetno dugih linija u nekoliko manjih. Radi kompletnosti, preveden je i originalni korpus recenzija [1] (koji je označen prema sajtu porekla i oceni filma) za buduću upotrebu. Prevedeni originalni srpski korpus je priložen kao „SER-ENG - Originalni prevod".

Kako su u ARFF fajlu takođe bile prisutne dugačke recenzije zapisane u jednoj liniji, postojala je mogućnost da se prevođenje celog fajla zaustavi na takvim linijama. Stoga je veliki ARFF fajl podeljen na manje fajlove koji su sadržali 300 linija. Duže recenzije su podeljene na nekoliko linija, ali njih je bilo tek 10. Zbog male veličine korpusa, podelu je bilo moguće uraditi i ručno, a za izuzetno dugačke recenzije je bilo dovoljno samo zabeležiti pozicije, razdvojiti ih na više linija i kasnije u procesu ponovo spojiti. Fajlovi su označeni kao SerbMR3.0x.txt, x = 1..9 (priloženi kao „SER-ENG 01 - podeljene recenzije"). Inicijalni pokušaj prevoda zajedno sa klasama nije održavao strukturu prigodnu za ARFF - stringovi su gubili apostrofe na krajevima, ili su apostrofi neispravno umetani u sredinu, te smo stoga u svim fajlovima uklonili klase i apostrofe (u srpskim recenzijama nije bilo drugih apostrofa osim na početku i kraju). Koristili smo kratak program dat u prilogu B.6, a rezultati su priloženi u „SER-ENG 02 - Recenzije bez klasa".

Nakon ove pripreme, bilo je moguće prevesti sve fajlove bez problema. Fajlovi su zatim spojeni uz postavljanje *escape* karaktera pre svakog apostrofa korišćenjem programa datog u prilogu B.7. Ovakav skup recenzija uključen je kao fajl SerbMR3.en zajedno sa prevodima u prilogu „SER-ENG 03 - Prevod recenzija"

Konačno, pošto je originalni ARFF fajl bio strukturiran tako da su recenzije bile grupisane po klasama, (841 pozitivna, 841 neutralna, 841 negativna), bilo je veoma lako vratiti im klase. Za to je korišćen program dat u prilogu B.8. Nakon postavljanja uobičajenog ARFF zaglavlja (videti raniji primer), dobijena je prevedena verzija SerbMR-3C fajla (*SerbMR-3C.en.arff*, 2523 recenzije). Od njega je moguće dobiti SerbMR-2C fajl (*SerbMR-2C.en.arff*, 1682 recenzije) uklanjanjem neutralnih recenzija. Oba fajla sa prevodima su priložena uz ovaj rad.

4.3 Obrada negacije

Jedan od čestih problema prilikom analize sentimenta jesu složene negacije dobijene dodavanjem negativnih reči ispred pozitivnih prideva, poput „nije dobar" ili „ne valja". Jedno od očitih rešenja, osim korišćenja n-grama, je i dodavanje negativnih prefiksa rečima koje slede nakon negativne reči. Kako *Weka* nema ovu opciju, ona mora biti implementirana van tog alata. Korišćen je kôd identičan onom u [1], priložen od strane autora.

Primenom koda imamo opciju da označimo određen broj reči nakon negacije određenim prefiksom (NE_u srpskom jeziku, NOT_u engleskom jeziku). Izabrali smo varijante gde je označeno od 1 do 5 reči nakon negacije, i svaka od ovih varijanti za svaki jezik je sačuvana kao zaseban korpus. Za engleski smo iskoristili dva skupa negativnih reči, širi i uži, te stoga dobili dva različita označena korpusa, priložena uz ovaj rad („Obrada negacije – engleski jezik, verzija 1", odn. „verzija 2"). Ta lista reči je prikazana na slici 4.1, s tim što su reči korišćene isključivo u drugoj varijanti podebljane.

not, no, don't, doesn't, didn't, isn't, aren't, can't, won't, weren't, wasn't, ain't, mustn't, none, never, nowhere, nothing, nobody, neither, nor

Slika 4.1: Spisak negativnih reči za engleski jezik

Sličan postupak primenjen je i za srpski jezik, s tim što smo koristili samo jednu varijantu negacija. Lista negativnih reči za srpski jezik je data na slici 4.2. Dobijeni korpus negacija priložen je kao „Obrada negacije – srpski jezik".

ne, nemoj, nemojmo, nemojte, nisam, nije, nismo, niste, nisu, neću, nećeš, neće, nećemo, nećete, nemam, nemaš, nema, nemamo, nemate, nemaju

Slika 4.2: Spisak negativnih reči za srpski jezik

Poglavlje 5

Rezultati

U ovom poglavlju ćemo prikazati rezultate dobijene iterativnim procesom postepenih izmena podešavanja u *StringToWordVector* filteru alata *Weka*, primenjenim na dva dostupna jezika. Rezultati će biti izraženi kroz tačnost, odn. procenat ispravno klasifikovanih instanci u skupu za testiranje. U slučaju kada je potrebna negacija, potrebno je učitati odgovarajuće korpuse označene sa _nx, gde $x = 1..5$ predstavlja broj negiranih reči. U tabeli 5.1 su prikazana inicijalna *Weka* podešavanja koja ćemo nadalje referencirati. Kako je preduslov za korišćenje stemera da tokeni budu u malim slovima, *lowerCaseTokens* = True će takođe biti uziman za inicijalnu vrednost, posebno jer se u prošlosti pokazivalo da gotovo uvek dovodi do poboljšanja tačnosti.

Naziv opcije	**Inicijalna vrednost**
wordsToKeep	1000
lowerCaseTokens	false
minTermFreq	1
stemmer	NullStemmer
tokenizer	WordTokenizer
doNotOperateOnPerClassBasis	False
outputWordCounts	False
normalizeDocLength	No normalization
IDFTransform	False
TFTransform	False

Tabela 5.1: Inicijalna podešavanja *StringToWordVector* filtera u alatu *Weka*

Prvo ćemo za svaki jezik izračunati tačnost dobijenu 10-slojnom unakrsnom validacijom na oba relevantna skupa (uz podrazumevana *Weka* podešavanja), kao i tačnost uz korišćenje skupova za obučavanje i testiranje pri inicijalnim podešavanjima. Zatim ćemo izmenama postepeno povećavati tačnost ukoliko je to moguće, uzimanjem vrednosti podešavanja kojima se to postiže kao ulaz u sledeći korak. Kako bismo stekli bolju sliku o uticaju pojedinačne izmene, najčešće ćemo kombinovati dve opcije i uz tumačenje rezultata birati koju vrednost zadržavamo u daljoj konfiguraciji. Opcija koju izdvojimo kao uspešnu biće u tabeli obojena u sivo, osim ukoliko je u pitanju prva vrsta ili kolona - one predstavljaju podrazumevane vrednosti.

5.1 Engleski jezik

Inicijalno se kao skup za obučavanje uzima fajl *EngMR_large.arff*, a kao skup za testiranje *SerbMR-2C_EN.arff*, i tako će ostati ukoliko ne naglasimo drugačije.

Vrsta eksperimenta	Dobijena tačnost
10-slojna unakrsna validacija na originalnom engleskom korpusu	83.5796
10-slojna unakrsna validacija na prevedenom srpskom korpusu	76.9917
Inicijalna *Weka* podešavanja	70.2140
Inicijalna *Weka* podešavanja + lowercase	**70.9869**

Tabela 5.2: Inicijalne vrednosti tačnosti za engleski jezik

Podebljanu vrednost iz prethodne tabele ćemo uzeti kao inicijalnu tačnost, a konfiguraciju kao početnu koju ćemo modifikovati. Prve dve opcije koje ćemo razmatrati su veličina modela u broju tokena (opcija *wordsToKeep*) i uticaj dostupnih stemera za engleski jezik (opcija *stemmer*). Dobijeni rezultati za različite kombinacije su date u tabeli 5.3.

	1 000 r.	**10 000 r.**	**100 000 r.**	**1 000 000 r.**	**2 000 000 r.**
Bez stemera	70.9869	70.0357	70.9869	70.9869	70.9869
Porter	68.4304	69.2628	69.3817	69.3817	69.3817
Porter2	68.3710	69.0250	69.3817	69.3817	69.3817

Tabela 5.3: Kombinovanje različitih stemera i veličina modela na engleskom jeziku

Kao što vidimo, upotreba stemera dovela je do pogoršanja tačnosti, ali se primećuje da je veličina modela od 100 000 reči predstavljala maksimum u svakoj vrsti, a veći modeli nisu dovodili do poboljšanja, te zbog brzine izvršavanja (koja je proporcionalna veličini modela) u konfiguraciju dodajemo *wordsToKeep*=100000.

Sledeća dve opcije koje ćemo kombinovati su minimalna frekvencija reči/tokena i dodavanje negativnog prefiksa nakon negativne reči, kao što je opisano u podpoglavlju 4.3. Za svaku vrstu u narednim tabelama, potrebno je učitati odgovarajuće korpuse sa negativnim prefiksima, *EngMR_large_nx.arff* i *SerbMR-2C_EN_nx.arff*, $x = 1..5$, u zavisnosti od toga na koliko reči nakon negacije je potrebno dodati prefiks. Kao što je već rečeno, postoje dve verzije za engleski jezik. Rezultati za obe verzije prikazane su u tabelama 5.4 i 5.5 (nn. je skraćenica za „nakon negacije", dok *mtf* označava opciju *minTermFreq*).

	***mtf*=1**	***mtf*=2**	***mtf*=3**	***mtf*=4**	***mtf*=5**
bez prefiksa	70.9869	71.1653	70.5113	70.5113	70.5113
1 reč nn.	71.4031	71.2842	71.0464	70.6897	71.1653
2 reči nn.	70.8680	70.8086	70.214	70.5113	70.7491
3 reči nn.	70.9275	70.868	70.2735	70.4518	70.2735
4 reči nn.	70.7491	70.5707	70.0951	70.3924	70.2735
5 reči nn.	70.4518	70.2735	69.9168	70.0357	69.9762

Tabela 5.4: Kombinovanje obrade negacije za engleski jezik (verzija 1) i frekvencije pojavljivanja reči

	***mtf*=1**	***mtf*=2**	***mtf*=3**	***mtf*=4**	***mtf*=5**
bez prefiksa	70.9869	71.1653	70.5113	70.5113	70.5113
1 reč nn.	71.1653	71.522	71.2842	71.2842	71.7004
2 reči nn.	71.5220	71.4625	71.3436	71.0464	70.9869
3 reči nn.	71.5815	71.4031	70.9869	70.5113	70.6897
4 reči nn.	71.1653	70.868	70.7491	70.5113	70.6897
5 reči nn.	70.4518	70.214	70.0357	69.7384	69.7979

Tabela 5.5: Kombinovanje obrade negacije za engleski jezik (verzija 2) i frekvencije pojavljivanja reči

Primećujemo da jedino prilikom dodavanja prefiksa za 1 ili 2 reči nakon negacije u verziji 2 dobijamo barem bolje rezultate, dok minimalna frekvencija reči ne utiče konzistentno na tačnost, te je nećemo uzimati u obzir. Zbog blago veće tačnosti, u konfiguraciju dodajemo korišćenje fajlova *EngMR_large_n2.arff* i *SerbMR-2C_EN_n2.arff* iz verzije 2 kao osnovne korpuse.

Naredne opcije koje ćemo kombinovati su *IDFTransform* i *TFTransform*, na sada izmenjenim korpusima. Rezultati su prikazani u tabeli 5.6.

	IDFTransform=False	**IDFTransform=True**
TFTransform=False	71.5220	71.5220
TFTransform=True	71.4031	71.7598

Tabela 5.6: Kombinovanje IDF i TF normalizacije za engleski jezik

Kako je samo kombinacija ova dva parametra dovela do marginalnog poboljšanja tačnosti, nećemo menjati inicijalne vrednosti ovih opcija u konfiguraciji.

Naredne dve opcije za kombinovanje su upotreba nebinarnih odlika (*outputWordCounts*) i normalizacija dužine dokumenta (*normalizeDocLength*). Rezultati su prikazani u tabeli 5.7.

	No normalization	**Norm. all data**	**Norm. test data only**
binarne odl.	71.5220	71.1058	71.5220
nebinarne odl.	69.4411	69.3817	69.4411

Tabela 5.7: Kombinovanje nebinarnih odlika i normalizacije dužine dokumenta za engleski jezik

Nisu uočena značajna poboljšanja, pa stoga ni ove dve opcije neće biti uključene u dalju konfiguraciju.

Konačno, primenimo izmene opcije *doNotOperateOnPerClassBasis* i različitih varijanti n-grama. Korišćenje n-grama možemo dobiti izmenom *tokenizer* opcije i izborom najmanjeg i najvećeg n. U našem slučaju, uz podrazumevanu opciju *WordTokeniser* dobijamo unigrame, a upotrebom *NGramTokeniser*-a dobijamo n-grame u rasponu od najmanjeg do najvećeg postavljenog n. Nama će od interesa biti $n = 1..2$ i $n = 1..3$. Rezultati su prikazani u tabeli 5.8. Korišćena je skraćenica *npcb* umesto punog naziva opcije *doNotOperateOnPerClassBasis*.

	unigrami	**unigrami + bigrami**	**unigrami + bigrami + trigrami**
npcb=False	71.5220	71.0464	71.4625
npcb=True	71.5220	71.0464	70.8086

Tabela 5.8: Kombinovanje *doNotOperateOnPerClassBasis* i korišćenja n-grama za engleski jezik

Kako ni ovde nema poboljšanja, kao optimalnu konfiguraciju za engleski jezik uzimamo inicijalna Weka podešavanja uz *lowerCaseTokens*=True, *wordsToKeep*=100000, i korišćenje korpusa gde su prefiksi dodavani na dve reči nakon negacije. Optimalna tačnost je **71.522%**.

5.2 Srpski jezik

Inicijalno se kao skup za obučavanje uzima fajl *EngMR_large_SR.arff*, a kao skup za testiranje *SerbMR-2C.arff*, i tako će ostati ukoliko ne naglasimo drugačije.

Vrsta eksperimenta	Dobijena tačnost
10-slojna unakrsna validacija na prevedenom engleskom korpusu	82.7117
10-slojna unakrsna validacija na originalnom srpskom korpusu	75.5054
Inicijalna *Weka* podešavanja	69.6195
Inicijalna *Weka* podešavanja + lowercase	**71.1058**

Tabela 5.9: Inicijalne vrednosti tačnosti za srpski jezik

Analogno prethodnom podpoglavlju, podebljanu vrednost ćemo uzeti kao inicijalnu tačnost, i kao prve dve opcije razmatrati veličinu modela u broju tokena (opcija *wordsToKeep*) i uticaj dostupnih stemera za srpski jezik (opcija *stemmer*).

	1 000 r.	**10 000 r.**	**100 000 r.**	**1 000 000 r.**	**2 000 000 r.**
Bez stemera	71.1058	70.9275	71.1058	71.6409	71.6409
Kešelj i Šipka - greedy	70.7491	72.0571	72.0571	72.0571	72.0571
Kešelj i Šipka - optimal	69.5006	72.4138	72.1165	72.1165	72.1165
Milošević	71.1653	72.5922	72.6516	72.6516	72.6516
Ljubešić i Pandžić	69.4411	72.2354	72.9489	72.9489	72.9489

Tabela 5.10: Kombinovanje različitih stemera i veličina modela na srpskom jeziku

Optimalna kombinacija u ovom slučaju je 100 000 reči uz korišćenje stemera koji su razvili Ljubešić i Pandžić, što je identično zaključku iz [1]. Vredi napomenuti da je izabrani stemer ujedno i najsporiji od ponuđenih, ali ćemo nastaviti sa njegovom upotrebom zbog najbolje postignute tačnosti.

Sledeće dve opcije koje ćemo kombinovati su minimalna frekvencija reči/tokena i dodavanje negativnog prefiksa nakon negativne reči, kao što je opisano u podpoglavlju 4.3. Slično kao u prethodnom podpoglavlju, potrebno je učitati odgovarajuće korpuse sa negativnim prefiksima, *EngMR_large_SR_nx.arff* i *SerbMR-2C_nx.arff*, $x = 1..5$, u zavisnosti od toga na koliko reči nakon negacije je potrebno dodati prefiks. Rezultati su prikazani u tabeli 5.11 (nn. je skraćenica za „nakon negacije", dok *mtf* označava opciju *minTermFreq*).

	***mtf*=1**	***mtf*=2**	***mtf*=3**	***mtf*=4**	***mtf*=5**
bez prefiksa	72.9489	73.4839	73.7218	74.1379	73.5434
1 reč nn.	73.7812	73.8407	73.4839	74.0785	73.8407
2 reči nn.	73.1867	73.365	73.4245	73.7218	73.1867
3 reči nn.	73.1867	73.4245	73.0678	73.7812	73.2461
4 reči nn.	73.8407	73.7218	73.6029	73.8407	73.4245
5 reči nn.	72.5922	72.7705	73.1272	73.7812	73.1867

Tabela 5.11: Kombinovanje obrade negacije za srpski jezik i frekvencije pojavljivanja reči

Kako u ovom slučaju imamo poboljšanja u gotovo svim vrstama i kolonama, optimalne vrednosti izabrane su uprosečavanjem po vrstama i kolonama. Od ovog trenutka, u konfiguraciji ćemo koristiti korpuse *EngMR_large_SR_n1.arff* i *SerbMR-2C_n1.arff*, kao i minimalnu frekvenciju reči/tokena jednaku 4.

Naredne opcije koje ćemo kombinovati su IDFTransform i TFTransform, na sada izmenjenim korpusima. Rezultati su prikazani u tabeli 5.12.

	IDFTransform=False	**IDFTransform=True**
TFTransform=False	74.0785	74.0785
TFTransform=True	73.9001	74.0190

Tabela 5.12: Kombinovanje IDF i TF normalizacije za srpski jezik

Kako ne primećujemo nikakva poboljšanja prilikom korišćenja ove dve opcije, nećemo ih uključivati u našu optimalnu konfiguraciju.

Naredne dve opcije za kombinovanje su upotreba nebinarnih odlika (*outputWordCounts*) i normalizacija dužine dokumenta (*normalizeDocLength*). Rezultati su prikazani u tabeli 5.13.

	No normalization	**Norm. all data**	**Norm. test data only**
binarne odl.	74.0785	74.7325	74.0785
nebinarne odl.	72.7705	73.2461	72.7705

Tabela 5.13: Kombinovanje nebinarnih odlika i normalizacije dužine dokumenta za srpski jezik

Kako je pri normalizaciji dužine svih tekstova primećeno značajno poboljšanje za obe vrste, usvojićemo ovo u našu trenutnu konfiguraciju.

Konačno, iskombinujmo upotrebu opcije *doNotOperateOnPerClassBasis* i različitih varijanti n-grama. Rezultati su prikazani u tabeli 5.14. Korišćena je skraćenica *npcb* za pun naziv opcije *doNotOperateOnPerClassBasis*.

	unigrami	**unigrami + bigrami**	**unigrami + bigrami + trigrami**
npcb=False	74.7325	76.0999	76.5161
npcb=True	75.0892	76.5161	76.6944

Tabela 5.14: Kombinovanje *doNotOperateOnPerClassBasis* i korišćenja n-grama za srpski jezik

Možemo primetiti da su obe opcije dovele do poboljšanja tačnosti, te stoga dodajemo dve označene vrednosti za gornje opcije u optimalnu konfiguraciju našeg rada, koja konačno glasi: *wordsToKeep*=100000, stemer Ljubešića i Pandžića, *minTermFreq*=4, korpus koji označava jednu reč nakon negacije, normalizacije dužine svih dokumenata, *doNotOperateOnPerClassBasis*=True i korišćenje unigrama, bigrama i trigrama. Optimalna tačnost za srpski jezik je **76.6944%**.

5.3 Osvrt na rezultate

Nažalost, ni za jedan jezik nije dobijena tačnost koja može može da parira našem cilju od 84.44%, a iznenađujuće je i da za engleski jezik nismo uspeli da nadmašimo ni tačnost dobijenu unakrsnom validacijom skupa za testiranje, koji je imao dosta lošiju tačnost od skupa za obučavanje, a u slučaju srpskog jezika, poboljšali smo istu tačnost za manje od 2%.

Možemo se osvrnuti na simptomatičnu situaciju prisutnu u svim eksperimentima - tabela 5.15 predstavlja matricu zabune u jednom od njih. Može se jasno primetiti da se instance pozitivne klase uspešno klasifikuju u značajno većem procentu nego instance negativne klase, koje su praktično prepolovljene između dve klase. Ovo sugeriše da u budućnosti treba posebnu pažnju posvetiti ispravnoj klasifikaciji negativnih instanci, kao i da naš metod dodavanja prefiksa nije urodio plodom.

	POSITIVE_{pred}	NEGATIVE_{pred}
POSITIVE_{stv}	781	60
NEGATIVE_{stv}	332	509

Tabela 5.15: Tipična matrica zabune pri našim eksperimentima

Činjenica da je dobijena tačnost lošija čak i od one dobijene unakrsnom validacijom sugeriše da proces mašinskog prevođenja uvodi značajnu količinu šuma, i da skupovi za obučavanje i testiranje nemaju sličnu strukturu, iako su formalno na istom jeziku. Upotreba novijih prevoda bi potencijalno razrešila ovaj problem.

Osvrnimo se i na srpski korpus, gde njegova skromna veličina predstavlja problem u smislu da jako mali broj instanci može značajno uticati na tačnost. Za promenu od 1% bi bilo dovoljno samo 17 instanci! Stoga, jedan od prioriteta mora biti i proširivanje srpskog korpusa.

Konačno, kako je u pitanju bio samo jedan sistem za mašinsko prevođenje i samo jedan klasifikator, korišćenje više različitih opcija bi moglo biti od pomoći, a prema sugestijama brojnih autora iz poglavlja 2, možda bi trebalo pokušati i postupke bazirane na leksikonima, što ponovo zahteva proširivanje resursa na srpskom jeziku.

Poglavlje 6

Zaključak

U ovom radu, pokušali smo da poboljšamo uspešnost analize sentimenta dokumenata (filmskih recenzija) na srpskom jeziku proširivanjem postojećih resursa, što je postignuto korišćenjem mašinskog prevoda značajno većih resursa na engleskom jeziku. Kao klasifikator korišćen je multinomijalni naivni bajesovski klasifikator u kombinaciji sa *bag-of-words* reprezentacijom dokumenata, uz korišćenje alata *Weka* za izvršavanje eksperimenata.

Mašinski prevod dobijen je korišćenjem *Microsoft*-ovih alata, s tim što smo prevodili oba dostupna korpusa, sa srpskog na engleski i obrnuto. Priprema korpusa uključivala je izbacivanje dupliranih recenzija, popravljanje grešaka koje je uveo proces prevođenja, a od interesa je bila i obrada negacije gde smo modifikovali reči koje su pratile negaciju kako bismo ispravnije odražavali njihov potencijalno negativni sentiment.

Eksperimenti su podrazumevali korišćenje većeg korpusa kao skup za obučavanje, i manjeg kao skup za testiranje, uz podrazumevanu kompatibilnost jezika. Koristili smo i nekoliko stemera dostupnih za oba jezika kako bismo tretirali reči istog korena kao jednu reč. Varijacije u eksperimentima predstavljala su podešavanja modela u skladu sa *bag-of-words* reprezentacijom koje nam je nudio alat *Weka*. Kako je broj tih podešavanja bio dovoljan za kombinatornu eksploziju, korišćen je iterativni proces izmena gde smo se trudili da zadržavamo isključivo izmene koje su dovodile do pozitivnih efekata.

Dobijeni rezultati, iskazani kao tačnost u procentima, nisu nadmašili već postignute rezultate dobijene korišćenjem isključivo lokalnog korpusa. Postignuta tačnost za engleski jezik bila je 71.522%, a za srpski jezik 76.6944%.

Kao glavni problem nametnule su se poteškoće u ispravnoj klasifikaciji negativnih recenzija, kao i kvalitet mašinskog prevoda za koji se mogu očekivati poboljšanja u bliskoj budućnosti. Uz ovaj rad su priloženi svi prevedeni resursi, te se istraživanje može nastaviti u ovom pravcu, korišćenjem drugih klasifikatora i metoda.

Literatura

[1] Vuk Batanović, Boško Nikolić, Milan Milosavljević. Reliable Baselines for Sentiment Analysis in Resource-Limited Languages: The Serbian Movie Review Dataset. *Proceedings of the 10th International Conference on Language Resources and Evaluation*, pages 2688–2696, 2016.

[2] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, Christopher Potts. Learning Word Vectors for Sentiment Analysis. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, jun 2011.

[3] A. Balahur, M. Turchi. Multilingual Sentiment Analysis Using Machine Translation? *Proceedings of the 3rd Workshop in Computational Approaches to Subjectivity and Sentiment Analysis*, pages 52–60, jul 2012.

[4] G. Shalunts, G. Backfried, N. Commeignes. The Impact of Machine Translation on Sentiment Analysis. *DATA ANALYTICS 2016 : The Fifth International Conference on Data Analytics*, pages 51–56, 2016.

[5] A.R. B., Khapra M.M., Bhattacharyya P. Lost in Translation: Viability of Machine Translation for Cross Language Sentiment Analysis. *Computational Linguistics and Intelligent Text Processing. CICLing 2013. Lecture Notes in Computer Science*, pages 38–49, mart 2013.

[6] M. Araújo, A. C. M. Pereira, J. C. S. Reis, F. Benevenuto. An evaluation of machine translation for multilingual sentence-level sentiment analysis. *SAC '16 Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1140–1145, april 2016.

[7] Amira Barhoumi, Chafik Aloulou, Nathalie Camelin, Yannick Estève, Lamia Belguith. Arabic sentiment analysis: an empirical study of machine translation's impact. *LANGUAGE PROCESSING AND KNOWLEDGE MANAGEMENT INTERNATIONAL CONFERENCE (LPKM2018)*, oktobar 2016.

[8] Shai Shalev-Shwartz, Shai Ben-David. *UNDERSTANDING MACHINE LEARNING: From Theory to Algorithms*. Cambridge University Press, 2014.

[9] John D. Kelleher, Brian Mac Namee, Aoife D'Arcy. *FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS: Algorithms, Worked Examples, and Case Studies*. The MIT Press, 2015.

[10] Daniel Jurafsky, James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Third Edition Draft*. 2018.

[11] Ian H. Witten, Eibe Frank, Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition*. Morgan Kaufmann Publishers, 2011.

[12] Andrew Mccallum, Kamal Nigam. A Comparison of Event Models for Naive Bayes Text Classification. *AAAI-98 Workshop on 'Learning for Text Categorization'*, 1998.

[13] M.F. Porter. An algorithm for suffix stripping. *Program, Vol 14*, pages 130–137, 1980.

[14] Vlado Kešelj, Danko Šipka. A Suffix Subsumption-Based Approach to Building Stemmers and Lemmatizers for Highly Inflectional Languages with Sparse Resources. *Infotheca 9(1-2)*, pages 23a–33a, 2008.

[15] Nikola Milošević. Stemmer for Serbian language. *preprint arXiv:1209.4471*, 2012.

[16] Nikola Ljubešić, Damir Boras, Ozren Kubelka. Retrieving information in Croatian: Building a Simple and Efficient Rule-Based Stemmer. *Digital Information and Heritage*, pages 313–320, 2007.

[17] Nikola Ljubešić, Tomaž Erjavec. Croatian and Serbian tokeniser, posećen 2019-09-19. URL `https://reldi.spur.uzh.ch/blog/tokeniser/`.

[18] Nikola Ljubešić, Tomaž Erjavec, Darja Fišer. Corpus-Based Diacritic Restoration for South Slavic Languages. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 3612—-3616, maj 2016.

Prilog A

O novijim prevodima

U ovom prilogu smo izdvojili jednu originalnu recenziju na engleskom jeziku i njena dva prevoda na srpski jezik iz 2017 i 2019 godine, uz označavanje grešaka u njima.

The Invisible Ray is an excellent display of both the acting talents of Boris Karloff and Bela Lugosi. Karloff pulls off a flawless performance as a sullen and conflicted scientist who appears to put his scientific achievements ahead of his relationships with others, even his wife. His already loner personality becomes unbearable as he becomes paranoid. Lugosi plays the consummate professional, who is passionate about his work but still finds time to maintain on good terms with everyone, but still seems to have no real close friends. This was one of his few roles as a good guy and he plays it very well. It is hard, however to hear his accent and believe he is French. The biggest problem with the movie was that it was all based on "junk science"but, in a way, even the junk science makes it work well. Since the ideas and theories are completely idiotic, they are as "relevant"today as they were when the movie was made. And they are also as forward reaching- and always will be. This is a perfectly delightful movie to watch again and again. I saw it maybe 5 times this weekend and I could easily sit through it five more times. The acting is marvelous and the science is amusing. I highly recommend it.

Slika A.1: Originalna recenzija na engleskom jeziku

"Nevidljivi **zrači**" je jedan izvanredan prikaz oba **na glumu** Boris Karloff i Bela Lugoši. Karloff **izvede bez mane performanse** kao naučnik **zlovoljan i u konfliktu** koji se **pojavljuje da stavi njegovu** naučnih dostignuća ispred njegovog odnosa sa drugima, čak i **njegova supruga**. Njegova već **usamljenik** ličnost postaje nepodnošljiva dok postaje paranoik. Lugoši **glumi profesionalac**, koji je strastveni o njegovom poslu, ipak nalazi vremena **za održavanje u dobrim odnosima sa svima**, ali ipak izgleda da **ima** vrlo bliske prijatelje. Ovo je bila jedna od **njegove nekoliko** uloga kao dobar momak, i to vrlo dobro igra. Teško je, međutim da čujete njegov akcenat i **verujem** da je Francuz. najveći problem sa filmom je bio da je zasnovana na **sve „sa neželjene nauka"**, ali na neki način, čak i **neželjene nauke olakšava rad pa**. S obzirom **na** ideje i teorije potpuno **idiotski**, **su kao „relevantna" danas kakve su bile kada je napravio film**. I **oni su takođe kao napred dostigavši** - i uvek će biti. **ovo je predivno film** gledati iznova i iznova. Video sam ga možda 5 puta ovog vikenda i **sam mogao lako dosaditi** pet puta **više**. Gluma je **predivno**, a nauka je zabavna. **To se ne preporučuje**.

Slika A.2: Prevod recenzije na srpski jezik iz 2017. god.

Nevidljivi **Ray** je odličan prikaz **i glumice vršioca dužnosti** Borisa Karloff i **bele Lugiije**. Karlof **izvlači besprekoran učinak** kao mrštiti i neusaglašeni naučnik koji izgleda da stavlja svoje naučne uspehe ispred svojih odnosa sa drugima, čak i sa svojom suprugom. Njegova već **lonerna** ličnost postaje nepodnošljiva kada postane paranoičan. Lugosi **svira potrošni stručnjak**, koji je strastven prema svom radu, ali i dalje pronalazi vreme da bi održao dobre *uslove* sa svima, ali ipak izgleda da nema baš bliske prijatelje. To je bila jedna od njegovih nekoliko uloga *kao dobar momak* i veoma dobro **svira**. Teško je, međutim, čuti njegov naglasak i verovati mu da je Francuz. Najveći problem sa filmom bio je da je sve zasnovano na **„Neželjena nauka"** Ali, na neki način, čak i **neželjena nauka** čini da dobro funkcioniše. Pošto su ideje i teorije potpuno **idibiotici**, oni su **kao „relevantni"** danas kao i kada je film snimljen. I oni su takođe **u pogledu postizanja**-i uvek će biti. Ovo je savršeno divan film za gledanje **[iznova i iznova]**. Video sam ga možda 5 puta ovog vikenda i lako bih mogao da **sedim** još pet puta. Gluma je čudesna i nauka **se zabavlja**. Preporučuje se.

Slika A.3: Prevod recenzije na srpski jezik iz 2019. god.

Primetan je napredak u prevodu, posebno na nivou deklinacije reči i smisla rečenica. Broj označenih grešaka je gotovo duplo manji! Međutim, te greške su i dalje značajne, te je teško proceniti koliko bismo veću tačnost dobili korišćenjem ovih prevoda.

Prilog B

Programi

B.1 ManualEditor.cpp

```
#include <locale>
#include <codecvt>

using namespace std;

/*
Ova funkcija uzima liniju teksta i poziciju unutar nje, i kao rezultat vraća ispravnu
reč koja sledi nakon te pozicije. Reč može biti ispravljena jednostavnom zamenom
ćiriličnih karaktera, ili ručnom izmenom reči sa „čudnim" karakterima.
*/
wstring getNextWord(wstring line, int *pidx) {
    int index = *pidx;
    wstring word = L"";
    bool weirdChar = false;
    wchar_t ch;
    while (index != line.length() && line[index] != ' ') {
        ch = line[index];
        switch (ch) {
            case 1113: ch = 353; break; // љ -> š
            case 1115: ch = 382; break; // њ -> ž
            case 1078: ch = 263; break; // ж -> ć
            case 1080: ch = 269; break; // и -> č
            case 1088: ch = 273; break; // р -> đ
            case 1033: ch = 352; break; // Љ -> Š
            case 1046: ch = 262; break; // Ж -> Ć
            case 1048: ch = 268; break; // И -> Č
            case 1035: ch = 381; break; // Њ -> Ž
            case 1056: ch = 272; break; // Р -> Đ
            case 8211: ch = 45; break;  // EN DASH -> -
        }
        word += ch;
        if (ch > 1000 && ch != 8222 && ch != 8221) weirdChar = true;
        // ne tretiramo „ i " kao čudne karaktere
        index++;
    }
```

```
    // ove reči su se pojavljivale veliki broj puta,
    // i originalno su bile potpuno ćirilične,
    // latinični karakteri su dobijeni primenom gornjih zamena
             if (word == L"опčс") { word = L"opis"; }
    else if (word == L"Скувај") { word = L"Skuvaj"; }
    else if (word == L"покваđеност") { word = L"pokvarenost";  }
    else if (word == L"čзвоđне") { word = L"izvorne";  }
    else if (word == L"нађатčв") { word = L"narativ"; }
    else if (word == L"čсечцčма") { word = L"isečcima"; }
    else if (word == L"кčч") { word = L"kič"; }
    else if (word == L"здđавог") { word = L"zdravog"; }
    else if (word == L"нађацčју") { word = L"naraciju"; }
    else if (weirdChar) {
        // ukoliko imamo reč koja nije među ponuđenima,
        // korisnik je ručno ispravlja
        wcout << word << endl;
        wstring fixed; wcin >> fixed;
        word = fixed;
    }
    while (index != line.length() && line[index] == ' ') index++;
    *pidx = (index == line.length())? -1 : index;

    return word;
}

// Ova funkcija fajl zadat folderom i imenom deli na pojedinačne
// linije teksta koje zatim prethodna funkcija obrađuje reč po reč
void handleFile(string folder, string name) {
    wfstream file;
    wofstream outfile;

    file.open(folder + "\\" + name);
    file.imbue(std::locale(std::locale::empty(),
            new std::codecvt_utf8<wchar_t, 0x10ffff, std::consume_header>));

    outfile.open(folder + "\\" + name.substr(0, name.length()-4) + "_.txt");
    outfile.imbue(std::locale(std::locale::empty(),
            new std::codecvt_utf8<wchar_t, 0x10ffff, std::consume_header>));

    wcout.imbue(std::locale(std::locale::empty(),
            new std::codecvt_utf8<wchar_t, 0x10ffff, std::consume_header>));
    wcin.imbue(std::locale(std::locale::empty(),
            new std::codecvt_utf8<wchar_t, 0x10ffff, std::consume_header>));
```

```
    if (file.is_open()) {
        wstring line;
        while (getline(file, line))
        {
        int index = 0;
            while (index != -1) {
                wstring word = getNextWord(line, &index);
                outfile << word << " ";

            }
        }
        file.close();
        outfile.close();
    }
}

// Pokretanjem programa u folderu sa tekstualnim fajlovima
// prolazimo kroz svaki pozivom funkcije handleFile,
// da bismo popravili pojedinačne reči sa neispravnim karakterima
int main(int argc, char* argv[]) {
    UINT oldcodepage = GetConsoleOutputCP();
    SetConsoleOutputCP(65001);
    string exename(argv[0]);
    string base = exename.substr(0, exename.find_last_of("\\"));

    DIR *currDir = opendir(base.c_str());
    struct dirent *ent;
    int i = 0;

    while ((ent = readdir(currDir)) != NULL) {
        string fileName(ent->d_name);
        if (ent->d_type == DT_REG && fileName.substr(fileName.length() - 3, 3) == "txt")
        {
            cout << fileName << endl; i++;
            handleFile(base, ent->d_name);
        }

    }

    cout << "\n\n" << i << "\n\n\n" << endl;

    closedir(currDir);
    SetConsoleOutputCP(oldcodepage);

    system("PAUSE");
}
```

B.2 reconstruct.py

```
import os

# folder sa fajlovima nakon vraćanja dijakritika
dir = "folder_with_dcr_files"

# Linije u .dcr fajlovima su strukturisane na sledeći način
#         <l>.<r>.<bt>.<pt>-<kt> \t <ot> \t <dt>
#         l - redni broj linije, r - recenica fajla u kojoj se nalazi token
#         bt - redni broj tokena u recenici
#         pt - početna pozicija tokena, kt - krajnja pozicija tokena
#         ot - originalni token, dt - token sa vraćenim dijakriticima
# Dodatno, tokeni iz jedne recenice orig. fajla su odvojeni praznim redom.

for file in os.listdir(dir):
    if file.endswith(".dcr"):
        inf = open(dir + "/" + file)
        outf = open(dir + "_/" + file[:-17] + ".txt", 'w')
        entry_list = []
        prev_idx = 1;
        prev_line = '0'
        # Svaka linija .dcr fajla sadrži tačno jedan token, ili je prazna.
        # Ukoliko linija nije prazna, dodajemo popravljeni token u bafer.
        # Ukoliko je linija prazna, praznimo bafer i rekonstruišemo
        # recenicu originalnog fajla sa popravljenim tokenima iz bafera.
        for line in inf:
            if line.strip()=='':
                for entry in entry_list:
                    nums = entry[0].decode('utf8').split('.')
                    line_num = nums[0]
                    if (line_num != prev_line):
                        prev_line = line_num
                        prev_idx = 1
                    idcs = nums[3].split('-')
                    if (int(idcs[0])>int(prev_idx)):
                        for i in range(int(prev_idx), int(idcs[0])):
                            outf.write(' ')
                    outf.write(entry[2].encode('utf8'))
                    prev_idx = int(idcs[1])+1

                entry_list=[]
            else:
                entry_list.append(line[:-1].decode('utf8').split('\t'))
```

B.3 remove_br.py

```
import os

dir = "folder_path"

for file in os.listdir(dir):
    if file.endswith(".txt"):
        inf = open(dir + "/" + file)
        outf = open(dir + "_/" + file, 'w')
        print file
        for line in inf:
            new_line = line.decode('utf8').replace("< br / > < br / >","")
            # replace("<br /><br />"," ")
            # za englesku verziju
            outf.write(new_line.encode('utf8'))
        inf.close()
        outf.close()
```

B.4 combine_files.py

```
import os

dir = "folder_to_eng_corpus"
file_tuples = []
option_tuples = [("train", "small"), ("test", "large")]

# Kreiraćemo torke koje sadrže recenziju, njenu klasu sentimenta i naziv fajla.
# Sortiraćemo ih po tekstu recenzije, i zatim izbaciti duplikate prilikom upisivanja
# u konačni fajl. Prvo ćemo to učiniti sa prvih 25000 recenzija iz foldera "train",
# da bi zatim dodali i drugih 25000 recenzija iz foldera "test".
for option in option_tuples:
    for file in os.listdir(dir + "/"+ option[0] +"/neg"):
        if file.endswith(".txt"):
            print file
            inf = open(dir + "/"+ option[0] +"/neg/" + file)
            for line in inf:
                file_tuples.append((line.replace("'", "\\'"), "NEGATIVE", file))
            inf.close()

    for file in os.listdir(dir + "/"+ option[0] +"/pos"):
        if file.endswith(".txt"):
            print file
            inf = open(dir + "/"+ option[0] +"/pos/" + file)
            for line in inf:
                file_tuples.append((line.replace("'", "\\'"), "POSITIVE", file))
            inf.close()

    file_tuples = sorted(file_tuples, key=lambda tuple: tuple[0])

    outf = open(dir + "/EngMR_" + option[1] +".txt", 'w')
    outf.write("'" + file_tuples[0][0] + "', " + file_tuples[0][1] + "\n")

    dup = 0
    for i in range(1, len(file_tuples)):
        if (file_tuples[i][0] != file_tuples[i-1][0]):
            outf.write("'" + file_tuples[i][0] + "', " + file_tuples[i][1] + "\n")
        else:
            dup +=1

    print dup
    outf.close()
```

B.5 remove_similar.py

```
import os
from operator import itemgetter

inf = open("arff_file")
file_tuples = []

i = 0
for line in inf:
    i+=1
    if (i > 7):
        review_snippet = line[:100]
        file_tuples.append((review_snippet, i))

dupes = 0
for x in range(1,len(file_tuples)):
    if file_tuples[x][0] == file_tuples[x-1][0]:
        # Ukoliko je isečci recenzija identični,
        # ispisaćemo njihove redne brojeve radi provere
        print file_tuples[x-1][1]
        print file_tuples[x][1]
        print ""
        dupes +=1
print dupes

inf.close()
```

B.6 remove_classes.py

```
import os

dir = "path_to_SerbMR_files"

# Za svaku liniju svakog fajla ćemo proveriti da li počinje apostrofom
# ili se završava klasom nakon apostrofa, i te oznake ćemo ukloniti, jer
# one ne predstavljaju tekst recenzija. Ovde uzimamo pretpostavku da ne važi
# pravilo "jedna recenzija po liniji", jer smo neke morali da podelimo na
# više linija.
for file in os.listdir(dir):
    if file.endswith(".txt"):
        inf = open(dir + "/" + file)
        print file
        outf = open(dir + "_bez_klasa/" + file, 'w')
        for line in inf:
            review = line
            if (len(review) > 1 and review[0] == '\''):
                review = review[1:]
            if (len(review) > 11 and
            (review[-11:-1] == "',POSITIVE" or review[-11:-1] == "',NEGATIVE")):
                review = review[:-11]
            outf.write(review + "\n")
        inf.close()
        outf.close()
```

B.7 merge_and_escape_apostrophes.py

```
import os

# location of SER-ENG 03 files
dir = "path_to_translated_files"

# Ovaj program spaja sve prevedene fajlove u jedan, i ujedno
# stavlja escape karakter pre svakog apostrofa.
outf = open(dir + "/SerbMR3.en", 'w')
for file in os.listdir(dir):
    if file.endswith(".txt"):
        inf = open(dir + "/" + file)
        for line in inf:
            new_line = line.decode('utf8').replace("'","\\'")
            outf.write(new_line.encode('utf8'))
        inf.close()
outf.close()
```

B.8 arff_format.py

```
import os

dir = "path_to_SerbMR3.en"

inf = open(dir + "/SerbMR3.en")
outf = open(dir + "/SerbMR-3C.en.arff", 'w')

# Koristeći činjenicu da su recenzije u SerbMR-3C fajlu bile
# grupisane po klasama, vratićemo klase prevodima u istom redosledu.
idx = 0
flag = "POSITIVE"
for line in inf:
    new_line = "'"+line[:-1].decode('utf8')+"'," + flag + "\n"
    idx+=1
    outf.write(new_line.encode('utf8'))
    if (idx==841):
        flag = "NEUTRAL"
    if (idx==1682):
        flag = "NEGATIVE"

inf.close()
outf.close()
```