TakeLab

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

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3794

Otkrivanje teksta neprimjerenog sadržaja postupcima strojnog učenja

Sven Vidak

Zagreb, lipanj 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 13. ožujka 2014.

ZAVRŠNI ZADATAK br. 3794

Pristupnik: **Sven Vidak**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Otkrivanje teksta neprimjerenog sadržaja postupcima strojnog učenja**

Opis zadatka:

Razvoj interneta doprinio je bržoj i učinkovitijoj komunikaciji, ali je, zbog anonimnosti koju nudi korisnicima, također povećao broj nepoželjnih aktivnosti. Objavljivanje poruka neprimjerenog odnosno uvredljivog sadržaja najčešći je oblik nepoželjne aktivnosti, koji je moguće spriječiti postupcima za automatsko otkrivanje takvih poruka. S obzirom na izražajnost jezika, ovaj zadatak nije trivijalno rješiv, no zadovoljavajući se rezultati mogu postići primjenom modela strojnog učenja za klasifikaciju teksta.
U okviru završnoga rada potrebno je proučiti postupke za klasifikaciju teksta temeljene na strojnom učenju te postojeće postupke za otkrivanje teksta neprimjerenog odnosno uvredljivog sadržaja. Razraditi model za otkrivanje tekstova uvredljivog sadržaja na hrvatskome jeziku. Izgraditi i ručno označiti prikladan skup podataka na hrvatskome jeziku. Razviti sustav za otkrivanje teksta uvredljivoga sadržaja u proizvoljnome programskom jeziku te ga primijeniti na korisnički generirane poruke na hrvatskome jeziku. Isprobati nekoliko modela strojnog učenja, provesti iscrpno eksperimentalno vrednovanje sustava na ispitnim skupovima podataka te analizu pogrešaka. Radu priložiti izvorni i izvršni kod razvijenog sustava, skupove podataka i programsku dokumentaciju te citirati korištenu literaturu.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 13. lipnja 2014.

Mentor:

Doc. dr.sc. Jan Šnajder

Djelovođa:

Doc. dr.sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:

Prof. dr.sc. Siniša Srbljić

Sadržaj

1. Uvod

Ubrzanim razvojem Interneta posljednjih godina, porastao je i broj ljudi koji svakodnevno na raznim portalima čitaju vijesti, raspravljaju o aktualnim temama ili gledaju razne video materijale o tematici koja ih zanima. Kako gotovo svaka internetska stranica takvog tipa korisnicima nudi mogućnost rasprave (u obliku foruma ili običnog komentiranja), jasno je kako svako može u bilo kojem trenutku s drugima podijeliti svoja razmišljanja o nekoj temi.

Takav način komunikacije između korisnika uvelike je pridonio povećanju komentara neprimjerenog sadržaja na gotovo svakoj stranici na kojoj je komunikacija između korisnika moguća. Kako takvih komentara ne bi bilo previše, na nekim forumima se tome problemu doskače brisanjem takvih komentara te isključivanjem osoba iz daljnjih rasprava. Međutim, takva praksa nije ustaljena na svim stranicama koje omogućuju neku vrstu komentiranja, pa na njima svatko može bez ikakvih posljedica napisati što god želi. To dovodi do jednog velikog problema današnjice – zlostavljanja na internetu (engl. *cyberbullying*) koje nije nimalo bezopasno. Kako je danas internet dostupan gotovo svakome, često se događaju samoubojstva mladih ljudi koji su na neki način zlostavljani na društvenim mrežama ili forumima. Manje radikalan primjer je kada se djeca povuku u sebe i ne žele imati kontakt s drugim ljudima te zbog toga imaju probleme u školi. Sve u svemu, problem tekstova neprimjerenog sadržaja može imati ozbiljne posljedice te bi bilo dobro kak bi postojale učinkovite metode koje bi taj problem riješile.

Općenito govoreći, neprimjereni sadržaj je bilo kakav tekst, slika ili video koji direktno vrijeđa osobu ili skupinu osoba. U tom kontekstu možemo govoriti o općenitim uvredama kao što su one koje se tiču rase, nacionalnosti, religijske pripadnosti ili pak seksualne orijentacije ili o uvredama kojima se pokušava narušiti ugled pojedinca. U ovu drugu skupinu možemo svrstati uvrede koje vrijeđaju čovjeka na osobnoj razini (intelekt, izgled, invaliditet i sl.) te one koje su usmjerene prema njemu bliskim osobama, ali i sadržaj koji nije usmjeren prema nikome,

nego je sam sadržaj pisan pogrdnim jezikom. Naravno, ovo je samo manji dio onoga što se danas može naći na Internetu, ali je dovoljno kako bi dalo okvirnu sliku što se točno smatra neprimjerenim sadržajem.

Ovaj, za čovjeka naizgled, jednostavan problem, predstavlja problem računalu jer računalo nije sposobno razumjeti ono što piše u nekom tekstu na način na koji to čovjek razumije. Problem prepoznavanja tekstova neprimjerenog sadržaja pripada skupini problema unutar područja prirodne obrade jezika (engl. *natural language processing, NLP*) koji se jedinim imenom zovu klasifikacija teksta (engl. *text classification*).

Taj je problem ljudima poznat od 80-ih godina 20. stoljeća kada su nastale prve metode koje su bile korištene za klasifikaciju. Jedan od najočitijih primjera klasifikacije teksta je raspoznavanje neželjene elektroničke pošte koji je sličan kao i problem raspoznavanja teksta neprimjerenog sadržaja – u oba slučaja je na temelju teksta potrebno odrediti radi li se o neprimjerenom (neželjenom) sadržaju. Ono što razlikuje ta dva problema jest to što je kod otkrivanja neprimjerenog sadržaja raspon tema puno veći te ima puno više faktora koji se mogu uzeti u obzir prilikom klasifikacije.

Cilj ovog rada je isprobati postojeće metode nadziranog strojnog učenja na problemu raspoznavanja teksta neprimjerenog sadržaja u komentarima na hrvatskome jeziku te analizirati koje od tih metoda će dati najbolje rezultate. Također, bitno će biti shvatiti na koji način obraditi tekstove iz komentara, a da iz njih bude što jednostavnije izvući što više informacija o pojedinom komentaru.

U nastavku ovog rada dan je pregled značajnijih radova koji se bave sličnom tematikom. Nakon toga je opisan općenit pristup rješenju te njegova implementacija. Na kraju su obrazloženi rezultati te su u zaključku predložene neke druge metode koje bi se mogle isprobati u budućnosti.

2. Srodni radovi

Područje klasifikacije teksta neprimjerenog sadržaja nije u potpunosti nova grana te postoji dosta radova koji se bave tim problemom. Međutim, tehnike koje se danas većinom koriste oslanjaju se na početno definiran rječnik koji se kasnije može, ali i ne mora, proširivati. Prema [2], *Youtube* i *Facebook* koriste rječnike koji sakrivaju sadržaj u komentarima i statusima. Razlika između njih je da *Youtube* koristi strogo definiran rječnik dok *Facebook* korisnicima omogućava da u rječnik dodaju riječi ili fraze koje smatraju neprimjerenima. Međutim, jasno je kako ovakav pristup nije najbolji jer je izrazito krut i nedjelotvoran. Zbog tog problema su u [2] predložene naprednije metode, ali i dodatne analize "okoline" u kojoj su postovi nastali. Ono što autori predlažu je analiza sintakse (koje riječi su karakteristične u rečenicama u kojima se obraćamo specifičnom korisniku, npr. *ti, tvoj* i sl.) i analiza korisnika (na temelju prošlih korisnikovih aktivnosti, vidjeti je li sklon svađanju i vrijeđanju), a posebno su analizirane teme u kojima je vrijeđanje češće nego inače (religija, seks, rase i sl.). Takav pristup dovodi do puno boljih rezultata u usporedbi s ostalim radovima čiji autori ne ulaze toliko duboku u problematiku.

U [9] autori analizu podataka rade na tri razine: lokalne značajke (koristeći *TFIDF*), semantičke značajke (koriste se posebne oznake za specifične riječi) te kontekstne značajke (ako je jedna poruka uvredljiva, veća je mogućnost da su uvredljive i one oko njega). Rezultati koji su dobiveni su zadovoljavajući, ali i dalje slabiji od [2].

U [8] i [4] autori prilikom klasifikacije teksta koriste popularne metode strojnog učenja: stroj s potpornim vektorima (engl. *support vector machines, SVM*), stabla odluke (engl. *decision trees*) i logističku regresiju (engl. *logistic regression*). Međutim, ono što je kod [8] bitno spomenuti je to da su autori puno pažnje posvetili obradi teksta s kojim su radili prije nego su ga predali algoritmima. Naime, takav pristup je dobar iz razloga što se minimiziraju učinci riječi koje nemaju

nikakvog značaja kao što su: strane riječi, URL-ovi, označavanje korisnika ili sadržaja (znakovi @ ili #) i druge.

Još jedna popularna metoda za klasifikaciju teksta korištena je u [5] i [6]. Naivan Bayesov klasifikator je metoda koja se, kako ćemo u nastavku vidjeti, temelji na vjerojatnosti pojavljivanja riječi u tekstu i na temelju toga predviđa kojoj klasi pripada dani tekst.

2.1. Korišteni algoritmi

Prema [2], najjednostavnija metoda koja se može koristiti za detekciju teksta neprimjerenog sadržaja su rječnici, no oni nisu previše korisni, pa dalje neće biti razmatrani.

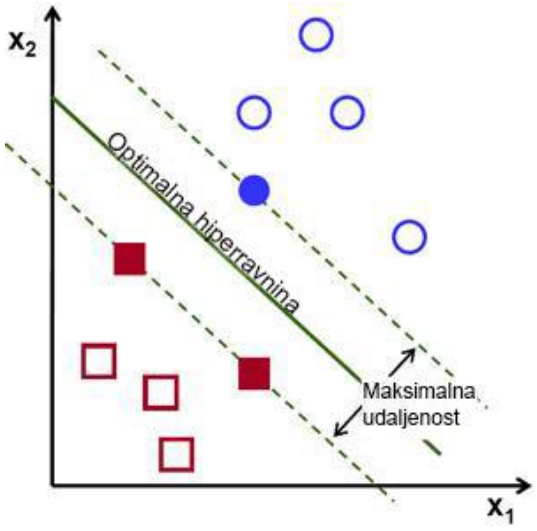
U novijim radovima, većina autora koristi gore nabrojane metode te se one pokazuju kao dobar alat za uspješnu klasifikaciju i biti će detaljnije pojašnjene u sljedećim poglavljima. Uz te metode, često korištena metoda je i metoda N-grama koja je korištena u [[2]], ali ona ovdje neće biti razmatrana.

2.1.1. Stroj s potpornim vektorima

Stroj s potpornim vektorima često je korištena metoda klasifikacije teksta. Način na koji on radi je razdvajanje skupova podataka po nekoj ravnini (hiperravnini) tako da su ti skupovi što udaljeniji od obje strane ravnine. Naravno, ako postoji više ravnina po kojima se skupovi mogu razdijeliti, tada se uzima ona kod koje je udaljenost najveća.

Ono što je ključ za dobar rad ove metode je tzv. jezgrena funkcija (engl. *kernel function*) kojom se na jednostavan način (skalarnim umnoškom vektora) može jednostavno baratati podacima u visoko dimenzionalnom prostoru. Naime, iako se podaci najčešće mogu linearno razdvojiti, postoje slučajevi u kojima se podaci moraju prebaciti u neku puno višu dimenziju od trenutne da bi se mogli razdvojiti i tada se jezgrene funkcije izrazito korisne.

Također, bitna stvar kod ovog algoritma je parametar C (standardna oznaka). Taj parametar algoritmu govori na koji način odabire hiperravninu. Ako je parametar C veliki, algoritam će pokušati što više primjera ispravno klasificirati, no zbog



Slika 2.1: Hiperravnina koja razdvaja skupove

toga će udaljenost s obje strane odabrane hiprravnine biti nešto manja. S druge strane, ako parametar C postavimo na malu vrijednost, algoritam će pokušati naći što veću udaljenost obiju strana s obzirom na hiperravninu, ali će samim time više primjera biti krivo klasificirano. Iz ovoga je vidljivo kako je parametar C izuzetno bitan kod primjene ovog algoritma.

2.1.2. Naivan Bayesov klasifikator

Kao što je prije spomenuto, naivan Bayesov klasifikator radi na temelju vjerojatnosti koristeći Bayesovu formulu:

$$P(c|\boldsymbol{d}) = \frac{P(\boldsymbol{d}|c) \cdot P(c)}{P(\boldsymbol{d})} \tag{2.1}$$

gdje je $\vec{d}$ vektor dokumenta (posta), a c klasa kojoj dokument pripada. Odluka o tome koja klasa je najvjerojatnija klasa danog dokumenta donosi se na temelju formule:

$$c_{ML} = \arg\max_{c}(P(\boldsymbol{d}|c) \cdot P(c)) \tag{2.2}$$

koja kaže da je klasa kojoj dokument pripada ona koja maksimizira izraz s desne strane jednakosti. Treba primijetiti kako je to ista formula kao i početna, samo što je iz nje maknuti nazivnik *P(d)* koji je konstantan za svaki dokument te time nema utjecaj na krajnji rezultat. Kako bi ovu formulu dodatno pojednostavili, potrebne su nam dvije pretpostavke. Prva je ona koja kaže da poredak riječi nije bitan. Ta pretpostavka se veže sa pojmom "vreće riječi" (engl. *bag of words*)

gdje sve riječi iz tekstova možemo staviti u strukturu podataka kojoj poredak nije bitan te time ubrzati algoritam. Druga pretpostavka je vezana za vjerojatnosti. Ako vektor $\vec{d}$ prikažemo kao skup riječi $x_1, x_2, \ldots, x_n$ tada možemo reći da je vjerojatnost $P(x_i|c)$ ne ovisi o poziciji riječi u tekstu. Iz toga slijedi da možemo pisati: $P(x_1, \cdots, x_n|c) = \prod_{i=1}^{n} P(x_i|c)$. Koristeći te pretpostavke, možemo u potpunosti pojednostavniti formulu te onda sada izgleda ovako:

$$c_{NB} = \arg\max_{c}(P(c) * \prod_{x} P(x|c)) \tag{2.3}$$

Sada nam još preostaje razjasniti na koji način izračunati *P(c)* i *P(x|c)*. Vjerojatnost pojave određene klase je jednostavno odrediti. Prebrojimo koliko se dokumenata u našem skupu postova nalazi u toj klasi i podijelimo taj broj za ukupnim brojem dokumenata. Problem se javlja kod izračuna vjerojatnosti pojave riječi ako znamo klasu. Početna formula koja bi se mogla iskoristiti je:

$$P(w_i|c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)} \tag{2.4}$$

gdje je V rječnik svih riječi. Međutim, u slučaju da neka riječ ne postoji u našem skupu, a kasnije se pojavi u nekom drugom skupu, ova će formula za nju dati vrijednost 0 što stvara problem jer to znači da se ta riječ nikad ne bi smjela pojaviti. Zbog toga se radi postupak koji se zove Laplaceovo zaglađivanje (engl. *Laplace smoothing*) koji broj pojavljivanja svake riječi povećava za 1. Konačna formula sada glasi:

$$P(w_i|c_j) = \frac{count(w_i, c_j) + 1}{\sum_{w \in V} count(w, c_j) + |V|} \tag{2.5}$$

gdje je $|V|$ ukupan broj riječi u rječniku.

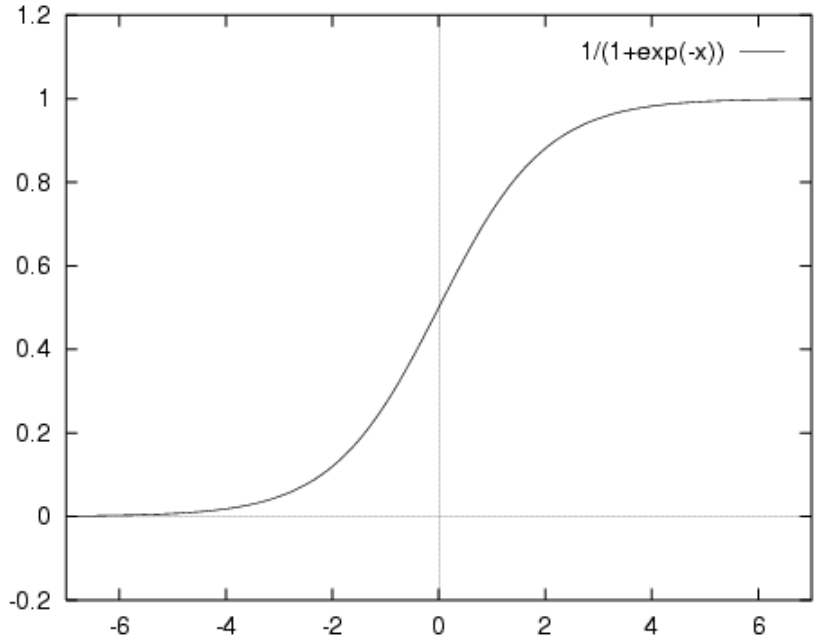
2.1.3. Logistička regresija

Ova metoda klasifikacije radi na vrlo jednostavnom principu: za dani skup vektora treba naći parametre koji što bolje opisuju taj skup i na temelju tih parametara raditi klasifikaciju novih vektora. Kako je ovo dosta općenit opis, detalji će biti razmotreni u nastavku.

Funkcija koju ćemo koristiti je tzv. sigmoidna funkcija. Naime, parametri koje moramo naći će pomoću te funkcije raditi klasifikaciju, pa je oblik te funkcije dosta bitan.

$$h_\theta(x) = h(\theta^T \vec{x}) = \frac{1}{1 + e^{-\theta^T \vec{x}}} \tag{2.6}$$

Na slici 2.2 vidimo kako za vrlo mali pomak oko 0 po x-osi, ova funkcija naglo



Slika 2.2: Sigmoidna funkcija

mijenja svoj iznos te ju to čini dobrom klasifikacijskom funkcijom. Ovdje je bitno spomenuti kako npr. funkcija skoka koja ima još brži porast (skok) nije bolja opcija. Razlog leži u činjenici da ova metoda radi gradijentni spust (engl. *gradient descent*) koji koristi derivaciju funkcije. Kako *step* funkcija ima skok u x=0, njena derivacija nije definirana u toj točci te je zbog toga nemoguće do kraja provesti izračun. Za razliku od nje, derivacija sigmoidne funkcije daje zanimljiv rezultat – ako sa $f(x)$ označimo sigmoidnu funkciju, njena derivacija je $f(x)(1 - f(x))$. Parametar θ je parametar koji je potrebno naći za kasniju klasifikaciju, a vektor $\vec{x}$ je vektor dokumenta. Sada imamo sve potrebno za konačni oblik formula koje se koriste kod ove metode.

Formula 2.7 je tzv. funkcija cijene koja nam daje na temelju parametara daje jedan broj kojim možemo procijeniti koliko su ti parametri dobri.

$$J(\theta) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} \cdot log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot log(1 - h_\theta(x^{(i)}))\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{2.7}$$

Na prvi pogled formula izgleda vrlo kompliciranо, ali ustvari nije. m označava broj dokumenata u skupu, y su oznake (klase) dokumenata, λ je parametar koji moramo na neki način izabrati, a $h_\theta(x^{(i)})$ je vektor i-tog dokumenta u skupu. To bi značilo da, ovisno o oznaci (koja je 1 ili 0), za svaki dokument jedan od

pribrojnika u zagradi nestaje, a drugi "preživi". Desni član koji se nalazi izvan zagrade je neka vrsta tzv. funkcije kazne (regularizacija) koja, ovisno o parametru λ određuje koliko ćemo paziti na to da θ bude što preciznije određena. Ako je λ postavljena na veliku vrijednost, tada će se pogreška kažnjavati strože nego u slučaju da je postavljena na manju vrijednost. To znači da u slučaju velike vrijednosti parametra λ može doći do nedovoljne naučenosti modela, a u slučaju male vrijednosti do prenaučenosti modela (model će na skupu na kojem je učio davati jako dobre rezultate, a na skupu za testiranje rezultati će biti loši). Ono što je također bitno primijetiti kako se regularizacija ne primjenjuje na prvi član vektora θ.

Kako je već prije spomenuto, ovaj algoritam napreduje metodom gradijentnog spusta. Formule koje se pritom koriste dobiju se derivacijom gornje formule pazeći pritom da se za θ_0 regularizacijski dio formule ne uzme u obzir. Zbog jednostavnosti, te formule neće biti ovdje prikazane.

2.2. Vektor dokumenta

Do sada se već nekoliko puta u tekstu spominjao pojam vektora dokumenta. Vektor dokumenta je način na koji se dokumenti prikazuju u numeričkom obliku kako bi algoritmi s njima znali raditi. U ovom su radu korištena su dva pristupa izračuna vektora: broj pojavljivanja riječi i *TFIDF (Term frequency – inverse document frequency)* (pobliže opisan u [7]).

Ako koristimo pristup koji broji pojavljivanje riječi u postu, algoritam je izuzetno jednostavan – za svaku riječ svakog posta gledamo koliko se puta ona pojavila u postu.

Ako koristimo *TFIDF* postoje, uvjetno rečeno, dva koraka: izračun TF-a te potom izračun IDF-a. TF član je identičan kao i kod brojanja pojavljivanja riječi u pojedinom postu, dok se IDF član odnosi na to u koliko postova se pojedina riječ pojavljuje.

$$tfidf(t_k, d_j) = \underbrace{\#(t_k, d_j)}_{\text{TF}} * \underbrace{ln(\frac{T_r}{\#T_r(t_k)})}_{\text{IDF}} \tag{2.8}$$

Iz ovoga je vidljivo da je značenje IDF člana malo kompliciranije nego što je to

opisano gore. Naime, nazivnik argumenta logaritma je broj postova u kojima se riječ pojavljuje dok je njegov brojnik ukupan broj postova. IDF član je logaritam omjera tih dviju vrijednosti. To znači da ako se riječ pojavljuje u puno postova, onda nije previše bitna. Treba primijetiti kako u slučaju da se riječ pojavljuje u svim postovima, cijeli izraz poprima vrijednost 0.

3. Rješenje

U ovom poglavlju će biti opisani početni pristup problemu, način ekstrakcije postova (o kojem će više riječi biti u slijedećem poglavlju) te način na koji su komentari označavani. Također, bit će dano i nekoliko primjera komentara po raznim kategorijama.

3.1. Izvori komentara

Kako se u ovome radu klasifikacija teksta radi na hrvatskome jeziku, prva stvar na koju je bilo bitno pripaziti je to da komentari koji će se koristiti za učenje algoritama sadrže što manje riječi koje pripadaju nekom drugom jeziku (idealno, da riječi koje pripadaju nekom drugom jeziku uopće ne postoje).

Zbog toga je izbor pao na portale *Index* i *Dnevnik*. Naravno, na ovim, kao i svim ostalim portalima, postoje komentari koji sadrže riječi iz drugih jezika (što zbog česte upotrebe engleskog jezika u svakodnevnoj komunikaciji, što zbog korisnika sa srpskog i bosansko-hercegovačkog govornog područja). Međutim, količina takvih komentara je zanemariva ako pogledamo komentare s nekih drugih stranica koje su bile razmatrane (npr. *Youtube*). Oba odabrana portala korisnicima nude mogućnost rasprave vezano za svaki objavljeni članak u obliku foruma. Svaki objavljeni članak na forumu ima zasebnu temu u kojoj korisnici mogu komentirati članak. Jedna od stvari koja je također pomogla prilikom odabira izvora postova je da su ta dva foruma puno manje moderirana (nema brisanja i uređivanja postova) od bilo kojih drugih, što znači da većina postova koji su napisani ostaje nedirnuta.

3.2. Ekstrakcija postova

Za problem ekstrakcije postova korištena je tehnika sustavnog pretraživanja internetskih stranica (engl. *web crawling*), kojom je prikupljeno više od 1000 postova iz raznih tema na oba foruma (Dnevnik i Index forumi). Svi postovi su zapisani u zasebne datoteke iz kojih su se lagano mogli dohvatiti. Nakon prikupljanja dovoljne količine postova, iz svih datoteka je izvučeno slučajnih 1000 postova koji su služili kao skup postova s kojima su radili algoritmi.

3.3. Označavnje postova

Na početku, postovi su idejno trebali biti označeni samo jednom oznakom (engl. *label*) – 0 ako je post u redu, a 1 ako je post sadrži uvredljiv tekst. Međutim, takvo označavanje se pokazalo prejednostavnim te je prošireno na više kategorija pri čemu je prošireno i značanje oznaka – ako se post može svrstati u neku od kategorija, dobiva oznaku *1*, ako ne, onda je oznaka *0*, a ako nije jasno kojoj kategoriji pripada, onda je oznaka *x*. Kategorije su:

1. Uvredljivo
   - tekst koji direktno ili indirektno vrijeđa neku osobu ili skupinu osoba (prije u radu opisano detaljnije). Bitno je primjetiti da je ovdje napravljena razlika između uvredljivog i nepristojnog sadržaja, dok je prije bilo riječi samo o neprimjerenom sadržaju koji je, možemo to tako reći, zajednički naziv za ta dva tipa.
2. Nepristojno
   - tekst koji ne mora nužno biti uvredljiv, štoviše, može biti i izrazito pozitivan, ali je pisan neprimjerenim jezikom (kasnije će biti dani primjeri takvih postova)
3. Sarkastično
   - zloban tekst koji direktno vrijeđa osobu kojoj je namijenjen. Često vrijeđanje nije eksplicitno, nego prikriveno.
4. "Meta" vrijeđanja
   - ova kategorija je korištena kako bi se dobio uvid u to vrijeđa li se češće ljude na forumu (korisnike foruma) ili izvan foruma (političari,

nogometaši, pjevači i sl.). Za "metu" je oznaka *1* korištena ako je "meta" korisnik, a *0* ako je osoba izvan foruma. Značenje oznake x nije se promijenilo.

Prije označavanja cijelog skupa podataka provedeno je testno označavanje u kojemu je sudjelovalo troje ljudi. Na temelju toga izračunati je Cohenov kapa (κ) koeficijent koji je mjera podudaranja označenih postova. Donja granica podudaranja koja mora biti zadovoljena varira ovisno o izvoru koji se koristi, ali je najčešće između 0,65 i 0,7. U našem slučaju, za dvije najbitnije kategorije, koeficijent je u prosjeku bio nešto iznad 0,8 što se može vidjeti u tablici 3.1.

Tablica 3.1: Cohenovi kapa (κ) koeficijenti

Par označivača	Uvredljivo	Nepristojno
1	0,86	0,95
2	0,75	0,90
3	0,72	0.70

Prilikom samog označavanja postova, najproblematičnije kategorije su bile: uvredljivo i sarkazam. Naime, postovi u kojima je korišten nepristojan jezik je lagano uočiti, međutim, korisnik može u svome postu jako dobro prikriti sarkazam koji je zapravo i uvreda te je zbog toga ponekad problem odlučiti u koju kategoriju svrstati neki post.

Nakon što je označavanje cijelog skupa postova završeno, napravljene su neke analize te izdvojeni postovi koji su ili rubni slučajevi ili jasni primjeri postova koji (ne) pripadaju određenoj kategoriji.

3.4. Analiza podataka

Prva analiza koja se može napraviti je ona u kojoj ćemo pogledati duljinu postova. Duljina postova na forumima može poprilično varirati – od kratkih postova koji se sastoje od jedne ili dvije riječi pa sve do postova koji se protežu kroz nekoliko redaka.

Tablica 3.2: Analiza duljine postova – znakovi

Maksimalno znakova	Minimalno znakova	Prosječno znakova
2353	10	145,16

Tablica 3.3: Analiza duljine postova – riječi

Maksimalno riječi	Minimalno riječi	Prosječno riječi
418	1	28,23

Kako bi ova analiza bila potpuna, bitno je spomenuti kako su ovi podaci dobiveni na postovima koji su obrađeni tako da se sve riječi i brojevi odvojene od interpunkcija kako bi se iz postova mogle izvući riječi. Uzevši to u obzir, broj riječi i znakova je nešto manji, ali ne značajno.

Druga ograničenje je to da ovdje nisu razmatrani postovi kraći od 10 znakova jer takvi su odmah u početku označeni kao nebitni.

Druga analiza je ona gdje će se vidjeti koliko postova neprimjerenog sadržaja uopće ima u odabarnim postovima. Očekivano, broj postova uvredljivog i nepristojnog (neprimjerenog) sadržaja manji je od ostatka postova. U danim tablicama prikazani su omjeri:

Tablica 3.4: Omjer broja pozitivnih i negativnih uvredljivih postova

Pozitivni(1)	Negativni(0)
182	818

U tablicama koje slijede nalaze se primjeri nekih postova koji su posebno zanimljivi. Ono što je izdvojeno je: postovi koji se na granici dviju kategorija (npr. uvredljivo/neuvredljivo) i postovi koji pripadaju ili ne pripradaju nekoj kategoriji.

U tablici 3.6 možemo pogledati prvi i treći post. Prvi post očigledno pripada kategoriji nepristojnih postova, ali je zato kategorija uvredljivosti diskutabilna. S druge strane, treći post možda na prvu ruku ne izgleda uvredljivo, ali ako se pogleda njegov sadržaj (pod pretpostavkom da znamo što se događalo u navedenim državama) jasno je kako je ovaj post uvredljiv.

Tablica 3.5: Omjer broja pozitivnih i negativnih nepristojnih postova

Pozitivni(1)	Negativni(0)
187	813

Tablica 3.6: Primjer graničnih postova

Post	Uvredljivo (0/1)	Nepristojno (0/1)
`Ko (bezveze) *ebe taj i na*ebe....(bezveze)`	0	1
`Ako i ne moze svjetsko dobro ce doci kao izgovor nasima zasto nema gospodarskog rasta.`	0	0
`Koreju , Vijetnam , Srbiju ( nije da mi je žao ).....`	1	0

Tablice 3.7 i 3.8 prikazuju neke tipične primjere postova koji pripadaju njihovim kategorijama. Također, u tablici 3.8 namjerno su stavljeni postovi koji je pripadaju kategoriji uvredljivih postova kako bi se pokazalo da ima i takvih postova. Naime, ako pogledamo oba posta, jasno je kako postovi nisu uvredljivi (ili možda jesu?), ali sadrže riječi koje se mogu klasificirati kao nepristojan jezik.

Iz ovih nekoliko primjera, jasno je kako sam zadatak označavanja teksta nije jednostavan i često puta dosta zamršen i nejasan. U nastavku ćemo pogledati implementacijske detalje te pobliže razmotriti metode obrade postova kojih smo se dotakli u ovom poglavlju.

Tablica 3.7: Primjer uvredljivih postova

`Ma što ćeš ti jado dopustiti ???  Niti išta znaš niti itko tebe išta pita !`
`Nabijem te BRE na moj pravoslavni BRE koji Hrvatice najviše vole BRE !!!`
`komedija?  da je komedija od države onda bi boravio bez dozvole.  kremenko`

Tablica 3.8: Primjer nepristojnih postova

Svjetski vodje okupili su se u Rimu, gdje razmatraju mogucnost obuzdavanja nasilja u toj zemlji. Ooooooo svemoguci, Ti koji sve vidis I znas, odapni neku raketu medju "svjetske vodje" matere ti . !
Mi muskarci mozemo se sada po*rati na nas alat i sluzit ce nam samo za pisanje.

4. Implementacija

U prethodna dva poglavlja su spomenute neke metode i algoritmi koji su korišteni u programskoj implementaciji rješenja, ali na vrlo osnovnoj razini. U ovom poglavlju će te metode biti pobliže opisane, ali će se opisati i ostale bitne stvari kao što su način prikupljanja i obrade podataka, način kreiranja skupova podataka te postupak korjenovanja. Zbog složenosti same programske implementacije, ona je podijeljena u nekoliko dijelova:

1. Prikupljanje i obrada postova
2. Kreiranje skupova postova
3. Učenje
4. Evaluacija

Svaki se od ovih dijelova može podijeliti na više koraka, ali prije toga ćemo obrazložiti izbor programskog jezika koji je korišten u ovom radu.

4.1. Programski jezik

Programski jezik korišten prilikom implementacije ovog rada većim je dijelom Java. Iako je Javin sustav za rad s datotekama relativno spor, što ponekad predstavlja problem prilikom intenzivnog pisanja i čitanja datoteka, ono što Javu izdvaja je velika rasprostranjenost i količina gotovih alata koji su javno dostupni te se time lako uočavaju i rješavaju greške kod njihove implementacije.

Naravno, gotovo je nemoguće u obrazloženju *zašto Java?* ne spomenuti Javinu neovisnost o korištenoj platformi i mogućnost jednostavne nadogradnje aplikacije u budućnosti.

Uz Javu, za obradu tekstnih datoteka, na nekoliko mjesta je korišten i Python.

4.2. Prikupljaje i obrada postova

4.2.1. Pretraživanje foruma

Za sistematično pretraživanje internetskih stranica danas postoje mnogi gotovi alati. Ti alati rade na način da krenu od početne internetske stranice te na njoj skupe sve poveznice (engl. *link*) i njih pretražuju kasnije. Sličan je pristup korišten i u ovom radu. Naime, na samom početku je uočeno kako gotovi alati pretražuju preveliki skup stranica koje ne moraju biti pretražene jer ne sadrže relevantne podatke. Zbog toga je napravljena jednostavna implementacija algoritma koji u obzir uzima samo stranice određenog formata URL-a (svaka stranica teme na forumu ima određeni format URL-a) te je pomoću njega obavljeno pretraživanje stranica.

4.2.2. Parsiranje dohvaćenog sadržaja

Kada se s neke internetske stranice dohvati njen sadržaj, on je u formatu HTML (*HyperText Markup Language*). Takav format je posebno pogodan za obradu zbog oznaka (engl. *tag*) koje se koriste kako bi se odredio način prikaza teksta na stranici. Naravno, kao i za pretraživanje stranica, i za postupak parsiranja dokumenata u HTML-u postoje gotovi alati te su oni korišteni. Preciznije, korišteni alat je *jsoup*. Ovaj parser HTML-a nudi jednostavno programsko sučelje za dohvaćanje teksta određenog oznakama u obliku teksta koji se potom lagano može spremiti u datoteke. Pomoću tog alata, s navedenih foruma, prikupljeno je nekoliko tisuća postova s nekoliko stotina tema koje su pokrivale razna društvena događanja (politika, sport, estrada, Hollywood i sl.). Od tih svih postova, za algoritme je izdvojeno 1000 slučajno odabranih postova pri čemu su, kao što je prije spomenuto, odbacivani postovi kraći od 10 znakova (kako bi se izbjegli prazni i prekratki postovi koji nisu relevantni).

4.2.3. Obrada prikupljenih postova

Postovi dohvaćeni u prethodnom koraku su u izvornom obliku te nisu na nikakav način obrađivani tokom dohvaćanja. Takav oblik teksta je izrazito nepogodan za

trenutno korištenje jer je prepun izraza koji su jedinstveni, a zapravo to ne bi trebali biti. Tablica 4.1 jasno ilustrira na što se točno misli.

Tablica 4.1: Primjer jedinstvenih izraza koji to nisu

Jedinstveni izraz	"Pravilan" oblik
slučaju.....	slučaju
(sport)	sport
nakon??	nakon
izgleda..bravo	izgleda bravo
sve:D	sve

Ovakav problem je bio očekivan i prije same implementacije algoritma jer je sasvim normalno da ljudi previše ne obraćaju pozornost na to kako formatiraju tekst prilikom pisanja postova. Ovome problemu doskočilo se jednostavnim skriptama napisanima u programskom jeziku Python gdje su sve (ili barem velika većina) interpunkcije od riječi odvoje razmakom. To znači da je, na primjer, izraz *"nakon??"* pretvoren u *"nakon ??"*, izraz *"sve:D"* u *"sve :D"* (primjetite da *D* nije odvojeno od *:*) itd. Time je završen prvi korak obrade.

Drugi dio obrade napravljen je direktno u Javi. Drugi korak se sastojao u tome da se iz postova izvuku sve riječi te se iz njih stvori popis svih riječi (u kojem se svaka riječ pojavljuje točno jednom) kako bi se pomoću njega izračunali vektori koji će se koristiti u algoritmima. Ovaj postupak će biti detaljnije opisan u poglavlju gdje ćemo se osvrnuti na izvedbu algoritama učenja.

4.3. Kreiranje skupova postova

Općenito govoreći, kod algoritama učenja dobra je praksa podijeliti početni skup podataka na više dijelova. U slučaju da se skup podataka ne podijeli na više dijelova i da algoritam učenja koristi taj nepodijeljeni skup, parametri koji algoritam izračuna mogu biti prenaučeni (opisano prije u radu) (engl. *overfit*). Takav pristup bi mogao raditi u slučaju kad bi se naš početni skup podataka sastojao od svih mogućih primjera koje možemo zamisliti. Naravno, taj zahtjev je nemoguće ispuniti te se zbog toga problemu treba pristupiti na neki pametniji način.

Taj pametniji način je podjela početnog skupa na dva ili tri dijela. Ako skup dijelimo na dva dijela, tada je uobičajeni omjer podjele 80:20 – 80% podataka stavimo u skup za učenje (engl. *train set*), a 20% podataka u skup za testiranje (engl. *test set*). Ako skup dijelimo na tri dijela, tada je uobičajeni omjer 60:20:20 gdje 60% podataka ide u skup za učenje, 20% u skup za testiranje, a preostali podaci idu u skup za unakrsnu provjeru (engl. *cross validation*). Taj skup nam služi za određivanje parametara algoritma ako oni postoje i to na način da oponaša skup za testiranje. Uz to, postoji još jedna metoda unakrsne provjere koja se zove k-struka unakrsna provjera kod koje se skup za učenje dijeli na k dijelova, trenira na $k-1$ dijelova te testira na preostalom dijelu. Taj postupak se ponavlja k puta te se na kraju uzima prosječna pogreška svake iteracije. Ova metoda će detaljnije biti opisana u dijelu rada koji se bavim samom implementacijom algoritama.

4.4. Korjenovanje riječi

Korjenovanje riječi (engl. *stemming*) je proces skraćivanje riječi do njenog korijena. Tako, na primjer, riječ *papiri* postane *papir*, ali i riječ *papiru* postane *papir*. U ovom radu korišten je algoritam razvijen na Filozofskom fakultetu u Zagrebu. Najpoznatiji algoritam za korjenovanje riječi (koji radi s engleskim jezikom) je Porterov algoritam za korjenovanje (engl. *Porter Stemming Algorithm*) te je u njemu definirano niz pravila na temelju kojih se riječi svode na svoj korijen (na tom principu rade i svi ostali algoritmi korjenovanja).

Ono što se primjernom ovog postupka dobiva u algoritmu je manji broj značajki (engl. *feature*) s kojima se radi, ali i isto tako prepoznavanje jednakih riječi koje u izvedenom obliku nisu jednake. Time možemo dobiti bolje parametre tijekom procesa učenja, ali i bolje performanse kasnije tijekom klasifikacije novih primjera.

4.5. Algoritmi i učenje

U poglavlju 2.1 algoritmi koji su korišteni opisani su s teorijskog gledišta. U ovom poglavlju pozabavit ćemo se načinom na koji su oni izvedeni u implementacijskom dijelu ovog rada. Međutim, kako bi algoritmi dobro radili, tekst se njima mora prezentirati na njima razumljiv način – numerički. Zato ćemo se prvo osvrnuti

na sam postupak pretvorbe običnog teksta u numerički tip podataka razumljiv algoritmima učenja.

4.5.1. Stvaranje mape riječi i vektora postova

Kako bi različiti algoritmi "znali" učiti na temelju dohvaćenog teksta, taj tekst mora ne neki njima smisleni način biti predstavljen. Danas je uobičajeno da se, kod problema klasifikacije teksta, koristi vektorski prikaz tekstova. Ideja je slijedeća. Iz svih postova koja su dohvaćeni stvoriti mapu (rječnik, engl. *dictionary*) koja će svakoj riječi dodijeliti određen broj. Taj broj će kasnije predstavljati indeks polja u vektoru posta.

Algorithm 1 WordMapCreator

```
Ulaz: A – popis svih riječi
Izlaz: map – mapa riječ → indeks
id = 0
for all word in A do
  if word not in map then
    map{word} = id
    id++
  end if
end for
```

Nakon što je mapa stvorena, imamo podatke koji nam trebaju kako bismo mogli stvoriti vektore postova. U jednom od prethodnih poglavlja opisano je kako ti vektori izgledaju i što sadrže, pa je sada jednostavno shvatiti kako ih dobiti. Naime, za svaki post stvorimo novi vektor te potom prolazimo kroz sve njegove riječi. Ovisno o tome čime prikazujemo vektor (broj ponavljanja riječi ili *TFIDF*) računamo iznos u vektoru za tu riječ te pomoću prethodno stvorene mape indeksiramo poziciju u vektoru i na nju stavljamo izračunati broj. Broj pojavljivanje riječi se svodi na jednostavno povećanje tog broja za 1 kada se dođe to te riječi, a izračun *TFIDF* vrijednosti se radi po formuli 2.8.

Nakon što su vektori stvoreni prethodno opisanom metodom, može ih se po potrebi normalizirati. Normalizacija se provodi na način da se u svim vektorima na određenoj poziciji (indeksu) nađe najveća i najmanja vrijednost, vrijednosti

svakog vektora na toj poziciji oduzme se minimalna vrijednost te se tako dobiveni broj podijeli s razlikom maksimalne i minimalne vrijednosti. Ako vektore prikažemo u matrici gdje jedan redak matrice odgovara jednom vektoru i potom normaliziramo, dobit ćemo nešto slično ovome:

$$start = \begin{bmatrix} 4 & 5 & 7 & 9 & 3 \\ 2 & 1 & 1 & 0 & 4 \\ 7 & 3 & 2 & 5 & 1 \end{bmatrix} \Longrightarrow norm = \begin{bmatrix} \frac{4}{7} & \frac{5}{5} & \frac{7}{7} & \frac{9}{9} & \frac{3}{4} \\ \frac{2}{7} & \frac{1}{5} & \frac{1}{7} & \frac{0}{9} & \frac{4}{4} \\ \frac{7}{7} & \frac{3}{5} & \frac{2}{7} & \frac{2}{9} & \frac{1}{4} \end{bmatrix}$$

Gornji primjer nije reprezentativan (u smislu da se radu nisu pojavljivali slični vektori), ali lijepo prikazuje na koji način točno radi normalizacija. Na primjer, u prvom stupcu je najveća vrijednost 7, te su vrijednosti u prvom stupcu svih vektora podijeljene sa 7. Slično vrijedi i za ostale primjere.

4.5.2. Stroj s potpornim vektorima

Stroj s potpornim vektorima je izrazito učinkovit algoritam ako je njegova implementacija dobro izvedena. Međutim, rasprava o tome što točno pojam dobro izvedene implementacije znači bi uzela previše vremena tako da ovdje neće biti razmatrana, a čitatelja se upućuje na [3].

Zbog prethodno opisanog zahtjeva, u ovom je radi korištena gotova biblioteka *libsvm*. *libsvm* su razvili C.–C. Chang i C.–J. Lin sa sveučilišta u Tajvanu. Više detalja o samoj implementaciji može se naći na [1]. Ono što je za ovaj rad bitno je da ta biblioteka nudi jednostavan Java API (engl. *application programming interface*) koji nudi mogućnosti treniranja, testiranja i unakrsne provjere na jednostavan način. Postupak treniranja modela ovim algoritmom odvijao se na sljedeći način: nad skupom za treniranje pokrenuti 5-struku unakrsnu provjeru kako bi se dobio optimalan parametar C, s tim parametrom pokrenuti treniranje modela nad cijelim skupom za treniranje te pomoću dobivenog modela i skupa za testiranje pogledati kako algoritam radi za dosad neviđene primjere.

Kako bi *libsvm* biblioteka "znala" raditi s našim numeričkim vektorima, bilo je potrebno izgraditi određene strukture podataka. Svi vektori svih skupova pretvoreni su u oblik $pozicija : vrijednost$, pri čemu varijabla $pozicija$ kreće od 1. Iz ovoga je jasno kako je naš skup podataka za učenje predstavljen kao matrica koja ima dimenzije $veličina\_skupa\_za\_učenje \times broj\_značajki$. Ta matrica

je matrica čvorova (engl. *node*) koji se u korištenoj biblioteci zovu *svm_node*. Labele postova zapisane su u polje *double* vrijednosti (brojevi s pomičnim zarezom u dvostrukoj preciznosti). Matrica čvorova, polje labela i veličina skupa predani su razredu *svm_problem* koji se zajedno s parametrima algoritma šalje u metodu *svm_train* koja uči model i na kraju vrati naučeni model. Ako radimo unakrsnu provjeru, radimo s metodom *svm_cross_validation* u koju uz problem i parametre šaljemo i broj preklopa k (engl. *fold*) te prazno polje *double* vrijednosti u koje algoritam spremi procijenjene vrijednosti labela koje se kasnije mogu usporediti sa stvarnim vrijednostima. Nakon što je model naučen, koristimo metodu *svm_predict* u koju šaljemo dobiveni model i polje objekata razreda *svm_node* koji predstavlja jedan testni vektor, a iz nje dobijemo procjenu vrijednost labele za taj vektor.

Iz gore opisanog postupka jasno je vidljivo kako je vrlo jednostavno napraviti procjenu kvalitete rada algoritma koristeći neku od postojećih metrika o kojima će više riječi biti u sljedećem poglavlju.

4.5.3. Logistička regresija

Algoritam logističke regresije, poput algoritma stroja s potpornim vektorima, ima jedan parametar koji je potrebno optimirati. Zbog toga je i kod implementacije ovog algoritma korištena k-struka unakrsna provjera. Taj parametar određuje utjecaj regularizacije te je opisan u poglavlju 2.1.

U prvoj iteraciji rada ovaj algoritam je implementiran na temelju formula koje su dane u poglavlju 2.1. Kako se te formule mogu vrlo jednostavno implementirati matričnim množenjem, iskorištena je biblioteka *Apache Commons Math*[1] koja, uz sve ostalo, nudi osnovne operacija s matricama koje su bile potrebne za implementaciju samog algoritma. Uz to, implementiran je i API za unakrsnu provjeru, koji je ponudio metodu za podjelu skupa podataka na preklope te metodu za dohvat preklopa koji su se koristili za učenje modela prilikom unakrsne provjere.

U drugoj iteraciji, algoritmi su zamijeni algoritmima iz gotove biblioteke *liblinear* koja radi na vrlo sličan način kao i biblioteka *libsvm*. Razred *Problem* čuva podatke bitne za rad algoritma: veličinu skupa, oznake postova, vektore postova i broj značajki. Razred *Parameters* sadrži parametre algoritma: tip regularizacije,

[1] `http://commons.apache.org/proper/commons-math/`

parametar C i grešku. Parametar C je isti onaj kao i kod stroja s potpornim vektorima, a i zanimljivo je spomenuti kako je on u izravnoj vezi za parametrom λ. Naime, parametar C proporcionalan je recipročnoj vrijednosti parametra λ. Nakon što je sve postavljeno, koriste se metode *train* za učenje modela, *predict* za procjenu i *crossValidation* za unakrsnu provjeru ovisno o zadatku koji moramo napraviti. Sve metode pripadaju razredu *Linear*. Princip na kojem rade te metode identičan je onome kod stroja s potpornim vektora te ovdje neće biti ponovno razmatran. Jedina razlika između ove biblioteke i one korištene kod stroja s potpornim vektorima je u razlici imena razreda koji služe za spremanje podataka o vektorima ($pozicija : vrijednost$).

4.5.4. Naivni Bayesov klasifikator

Za razliku od prethodna dva, ovaj algoritam nema dodatan parametar čija vrijednost bi se morala određivati unakrsnom provjerom. Također, implementacijski gledano, puno je jednostavniji od prethodno opisanih algoritama, te je sve što nam treba sadržano u formuli 2.5. Prvi korak bilo je cijelokupni skup postova za učenje podijeliti na pozitivni (postovi označeni oznakom 1) i negativni (postovi označeni oznakom 0) skup te izračunati ukupan broj različitih riječi. Potom je iz tako dobivenih skupova izračunato koliko riječi (ne nužno različitih) se nalazi u pojedinoj klasi postova (desni pribrojnik nazivnika u formuli 2.5).

Nakon što su sve gore navedene metode implementirane, može se izgraditi mapa vjerojatnosti koja se kasnije koristi za klasifikaciju novih postova. Proces izgradnje mape je ustvari proces učenja naivnog Bayesovog klasifikatora. Za svaku riječ koja se nalazi u mapi svih riječi koje se pojavljuju u postovima skupa za učenje izračuna se koliko se puta ona pojavljuje u negativnim, a koliko puta u pozitivnim postovima te se te vjerojatnosti zapisuju u mapu koja pamti sve vjerojatnosti.

Tako naučen klasifikator koristimo za klasifikaciju novih primjera. Jedini problem na koji ovdje valja obratiti pozornost je problem malih vjerojatnosti. Naime, kako ima puno značajki (riječi), tipične vjerojatnosti s kojima radimo kreću se oko 10^{-4}. Kako se prilikom izračuna vjerojatnosti za nove primjere te vjerojatnosti množe (formula 2.3), vrlo lako može doći do situacije gdje će konačne vrijednosti biti toliko male da će računalu izgledati kao da su one jednake (točnije, jednake nuli). Taj problem jednostavno možemo riješiti koristeći logaritme, točnije, pravilo da je logaritam umnoška jednak zbroju logaritama. Služeći se tim trikom, umjesto

da se sve vjerojatnosti međusobno množe, njihove logaritamske vrijednosti se zbrajaju.

5. Evaluacija

Evaluacija je provedena nad 200 postova iz početnog skupa postova koristeći model naučen na preostalih 800 postova. Kako usporedbom stvarnih oznaka postova i oznaka dobivenih iz algoritma nije moguće najbolje procijeniti (ne)uspješnost rada klasifikatora, za to se koriste razne mjere koje su danas standardne kada se radi klasifikacija teksta.

Točnost je mjera koja nam govori koliki je postotak primjera iz skupa podataka točno klasificiran te je definirana kao:

$$Accuracy = \frac{n_A}{N} \cdot 100\% \tag{5.1}$$

gdje je n_A broj točno klasificiranih postova, a N ukupan broj dokumenata.

Preciznost definiramo kao postotak ispravno klasificiranih primjera u skupu onih primjera koji su prepoznati kao primjeri kategorije postova neprimjerenog sadržaja.

$$Precision = \frac{n_{TP}}{n_P} \cdot 100\% \tag{5.2}$$

n_{TP} je broj ispravno klasificiranih postova iz kategorije postova neprimjerenog sadržaja, a n_P je broj svih postova za koje je algoritam procijenio da su postovi neprimjerenog sadržaja.

Odziv je postotak ispravno klasificiranih primjera u skupu onih primjera koji su označeni kao postovi neprimjerenog sadržaja.

$$Recall = \frac{n_{TP}}{n_T} \cdot 100\% \tag{5.3}$$

n_{TP} je, kao i prije, broj ispravno klasificiranih postova iz kategorije postova neprimjerenog sadržaja, a n_T je ukupan broj postova označenih kao postovi neprimjerenog sadržaja.

Kao konačna mjera uspješnosti uzima se tzv. F1 rezultat (engl. *F1 score*).

$$F1\ score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \cdot 100\% \tag{5.4}$$

Motivacija za ovakav sustav procjene rada klasifikatora opravdan je zbog slučajeva u kojima imamo klase kod kojih je broj predstavnika jedne klase puno manji od broja predstavnika druge klase (što je slučaj i u ovom radu). Ako bismo koristimo samo točnost nad takvim klasama, ono što se može dogoditi je slučaj u kojem je krajnji rezultat vrlo visok, a algoritam cijelo vrijeme predviđa samo onu klasu koja ima više primjera. Time zanemarujemo činjenicu da za takve rezultate nismo niti morali napraviti složeni sustav, nego za svaki primjer na izlaz poslati oznaku brojnije klase. Zbog toga koristimo F1 koji nam date potpuniju sliku o tome koliko dobro klasifikator radi.

U nastavku su prikazane dvije tablice s rezultatima. Referentni model predstavlja one vrijednosti koje bi se dobile ako bi algoritam svaki primjer označio kao negativan (oznaka 0).

Tablica 5.1: Rezultati - uvredljivo

Model	Acc	P	R	F1
Referentni model	0,82	0	0	NaN
SVM	0,82	0	0	NaN
Logistička regresija	0,81	0	0	NaN
Naivni Bayes	0,81	0	0	NaN

Tablica 5.2: Rezultati - nepristojno

Model	Acc	P	R	F1
Referentni model	0,813	0	0	NaN
SVM	0,795	0	0	NaN
Logistička regresija	0,785	0	0	NaN
Naivni Bayes	0,785	0	0	NaN

Kao što je vidljivo iz tablica, rezultati nisu dobri i gotovo ih je nemoguće analizirati. Ipak, vidimo kako je točnost za razne algoritme različita te tu možemo

razmotriti njihovu efikasnost.

Ako pogledamo klasifikaciju uvredljivih postova, vidimo kako jedino metoda stroja s potpornim vektorima postiže granicu referentnog modela. Općenito govoreći, ovo nije dobar rezultat, nego minimum koji bi se morao zadovoljiti, ali uzevši u obzir ostale rezultate, u ovom slučaju može poslužiti kao referentna točka. Preostale dvije metoda, logistička regresija i naivan Bayesov klasifikator daju nešto slabije rezultate (točnije, jedan primjer više krivo klasificiraju).

S druge strane, kod klasifikacije nepristojnih postova niti jedan od algoritama ne dostiže referentni model. Ako malo razmislimo o tome, shvatit ćemo da je to zapravo vrlo čudno. Naime, klasifikacija nepristojnih postova bi trebala biti jednostavnija od klasifikacije uvredljivih postova. Prije u radu su opisane obje kategorije i najveći problem kategorije uvredljivih postova je sarkazam gdje post može biti uvredljiv, a da to nije odmah jasno. S druge strane, kategorija nepristojnih postova jasno je definirana eksplicitnim riječima koje je lagano prepoznati. Prema toma, kako se u ovome radu koristi metoda "vreće riječi", algoritmima bi trebalo biti puno lakše prepoznati nepristojne postove.

Ako se zanemare svi problemi koji postoje s ovim rezultatima, ono što se može vidjeti je ono što pokazuju rezultati drugih radova koji su se bavili sličnom tematikom – metoda stroja s potpornim vektorima daje bolje rezultate od svih ostalih.

5.1. Rasprava rezultata

Kako su u radu isprobane danas često korištene tehnike koje su se pokazale djelotvornima, dobiveni rezultati ukazuju na to da problem klasifikacije uvredljivog sadržaja na hrvatskome jeziku nije jednostavan kako se to u početku činilo.

Glavni problem bi mogao biti korištenje modela "vreće riječi". Naime, taj model pretpostavlja da su sve riječi međusobno neovisne te nemaju utjecaj jedna na drugu. Naravno, ta pretpostavka nije ispravna. Međutim, ovaj problem nije novi problem koji se nedavno javio. Postoje radovi koji su taj problem za tekstove na engleskome jeziku vrlo uspješno riješili.

U tim radovima (radovi spomenuti u Poglavlju 2) korišteni su ovakvi jednostavni modeli koji su dali rezultate koji su bili iznad granice referentnog modela. Upravo

zbog svih tih činjenica, dobiveni rezultati su poprilično iznenađujući.

5.2. Umjetni skup podataka

Ono što je također moguće je kriva (ili loša) implementacija algoritama. Kod toga se specifično misli na izradu numeričkih vektora postova iz teksta gdje postoji mogućnost krivog brojanja riječi ili korištenja krivog rječnika koji sadržava sve riječi. Sami algoritmi učenja (osim naivnog Bayesovog klasifikatora), ne mogu biti krivo implementirani jer su korištene gotove biblioteke koje danas koriste ljudi širom svijeta i za koje se pouzdano zna da daju dobre rezultate.

Kako bi se testirala mogućnost krive izrade vektora, napravljen je umjetni skup podataka za rječnikom od 50 riječi. Od tog rječnika stvorene su dvije skupine postova: prva skupina u kojoj su se nalazili uvredljivi postovi sastavljeni samo od uvredljivih riječi te druga skupina u kojoj su uvredljivi postovi imali i nekoliko (slučajan broj) riječi koje nisu uvredljive.

Za prvu skupinu, svaki korišteni algoritam dao je najbolje moguće rezultate, dok su za drugu skupinu rezultati bilo malo lošiji, ali je točnost bila oko 92%, a F1 oko 82% (stoj s potpornim vektorima).

5.3. Klasifikacija nad skupom oznaka

Još jedna od ideja kako bi se algoritam mogao popraviti je ideja klasifikacije na skupom oznaka. Pri tome se misli na to da se dvije klase ujedine u jednu i na te dvije klase se gleda kao na jednu. U ovom slučaju to bi konkretno značilo da postovi koji pripadaju i kategoriji uvredljivih i kategoriji nepristojnih gledaju kao pozitivni primjeri, a svi ostali kao negativni.

Ono što se ovakvim modelom postiže je veća odvojenost ta dva skupa te bi samim time algoritmima trebalo biti lakše naučiti takav model.

Ovaj pristup je implementiran, ali rezultati koji su dobiveni i dalje nisu bili iznad granice referentnog modela.

Iz ovog i prethodnog razmatranja umjetnih postova dolazimo do zaključka da je implementacija korektna te da je problem klasifikacije teksta uvredljivog sadržaja

puno teži nego što je na početku izgleda.

6. Zaključak

Klasifikacija teksta kao jedno veliko područje strojnog učenja danas je izrazito bitno u svakodnevnom životu (npr. neželjena elektronička pošta). Klasifikacija specifično teksta neprimjerenog sadržaja također je široko primjenjiva na raznim forumima, društvenim mrežama ili bilo kojoj drugoj stranici gdje postoji mogućnost nekog komentiranja.

U poglavlju 5.1 opisani su neki od problema zbog kojih bi rezultati mogli biti loši. Ono što možemo razmotriti su promjene u obradi postova ili općenito, u tome što smatramo postom. Prije je u radu spomenuto kako su kod samog odabira postova zanemareni svi postovi kraći od 10 znakova. Međutim, to možda i nije najbolji način za procjenu kvalitete posta. Općenito, broj značajki u ovom radu se kretao između 7000 i 8000. Ako se uzme u obzir da među postovima koji imaju više od 10 znakova postoje postovi koji sadrže svega jednu ili dvije riječi, jasno je kako bi bilo bolje da se u obzir uzmu postovi koji imaju npr. više od 5 riječi.

Takav pristup bi možda mogao pomoći, ali i kod njega postoji još jedna sitnica zbog koje bi ponovo mogli imati postove koji efektivno imaju jednu ili dvije riječi. Problem koji se javlja je taj da ljudi jako malo paze na stil pisanja pri čemu se događa da post ima dvije konkretne riječi, a ostatak čine interpunkcije, riječ od jednog slova koje je krivo napisano i sl. Takav problem se javlja jer se u koraku obrade postova u postovima sve interpunkcije odvajaju od riječi te se kao riječi koje su bitne uzimaju samo one koje počinju slovom. Kao što je spomenuto, to mogu biti i slučajno pritisnuta slova koja se broje kao riječ i više se nikad ne pojavljuju.

Sljedeći veliki problem koji je povezan s nepažnjom na stil pisanja su krivo napisane riječi ili riječi koje su namjerno cenzurirane od strane samo korisnika. Takve riječi su specifične i gotovo nikad se neće pojaviti u ostalim postovima (osim u slučaju da je korisnik često aktivan i drži se tog stila, što najčešće nije slučaj) i

algoritam od njih nema prevelike koristi.

Naravno, uz ove prijedloge, postoji još gomila drugih metoda kojima bi se tekst mogao dodatno označiti ili obraditi, a neke od metoda spomenute su u radovima iz poglavlja 2.

Iz svega ovoga možemo zaključiti kako problem klasifikacije teksta uvredljivog sadržaja na hrvatskome jeziku nije jednostavan koliko se to na početku činilo. Postoji puno raznih metoda kojima bi se mogli dobiti bolji rezultati, ali ih je u okviru ovog rada bilo vremenski nemoguće napraviti. Naravno, neke od predloženih metoda će u budućnosti biti implementirane kao nadogradanja kako bi se pokušali dobiti neki inicijalni rezultati koji neće biti ispod granice referentnog modela kao trenutni.

Literatura

[1] Chih-Chung Chang i Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[2] Ying Chen, Yilu Zhou, Sencun Zhu, i Heng Xu. Detecting offensive language in social media to protect adolescent online safety. *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012*, 2012.

[3] Corinna Cortes i Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[4] Karthik Dinakar, Roi Reichart, i Henry Lieberman. Modeling the detection of textual cyberbullying. *The Social Mobile Web*, 2011.

[5] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. *DTIC Document*, 1996.

[6] A.H. Razavi, D. Inkpen, S. Uritsky, i S. Matwin. Offensive language detection using multi-level classification. *Advances in Artificial Intelligence*, 2010.

[7] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 2002.

[8] Guang Xiang, Bin Fan, Ling Wang, Jason Hong, i Carolyn Rose. Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012.

[9] Dawei Yin, Zhenzhen Xue, Liangjie Hong, Brian D Davison, April Kontostathis, i Lynne Edwards. Detection of harassment on web 2.0. *Proceedings of the Content Analysis in the WEB*, 2009.

Otkrivanje teksta neprimjerenog sadržaja postupcima strojnog učenja

Sažetak

Razvoj interneta doprinio je bržoj i učikovitijoj komunikaciji, ali je zbog anonimnosti koju nudi korisnicima također povećao broj nepoželjnih aktivnosti. Cilj je takvu aktivnost u potpunosti spriječiti ili barem minimizirati kako bi korištenje interneta bilo što ugodnije. U okviru ovog rada proučeni su postojeći postupci za klasifikaciju teksta koji se danas standardno primjenjuju u radovima vezanim za to područje. Prikupljena je velika količina podataka koja je tada obrađena te su implementirani neki od najčešće korištenih algoritama strojnog učenja koji se koriste za klasifikaciju teksta. Rezultati koji su dobiveni na temelju skupljenih podataka nisu niti blizu očekivanih te je stoga razmotreno nekoliko načina kojima bi se rezultati mogli poboljšati.

Ključne riječi: Obrada prirodnog jezika, nadzirano strojno učenje, algoritmi učenja, klasifikacija teksta

Offensive text detection

Abstract

With an increasing number of users who use internet on a daily basis, offensive content on many pages became a big issue. Offensive text detection is a useful technique that aims to prevent offensive content from being posted on pages we visit every day. This thesis gave an overview on some of the existing algorithms in the field of a text classification, collect reasonably large amount of raw data, preprocessed it and implemented some of the most commonly used algorithms that are used in modern text classification. Results obtained using collected and preprocessed data were not even close to those we expected so we considered many other possibilities and methods that could perhaps improve the results.

Keywords: natural language processing, supervised machine learning, learning algorithms, text classification