Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Kristjan Pičulin

Identifikacija in analiza oglasnih člankov

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleksander Sadikov

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Kandidat naj zasnuje in implementira program, ki za določen spletni članek prepozna, ali je oglasni plačljiv članek ali pa je prava novica. Cilj je čimbolj natančno napovedovanje ali je članek oglas, zato mora imeti čimmanj napačno klasificiranih oglasov. Podpira naj slovenski in angleški jezik. Program naj kot vhod dobi spletni naslov članka in vrne svojo napoved ter kakšne so najpomembnejše besede v članku. V primeru, da prepozna kot oglas naj izpiše kaj ta članek predvidoma oglašuje. Kandidat naj analizira kako dobro se je program odrezal glede na zastavljene cilje.

Iskreno se zahvaljujem mojemu mentorju za nasvete, vsem mojim profesorjem, ki so me v štirih letih na fakulteti učili ter moji družini in prijateljem za podporo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
kratica	**angleško**	**slovensko**
CA	classification accuracy	klasifikacijska točnost
SGD	Stochastic gradient descent	Metoda stohastičnega gradienta
SVM	support vector machine	metoda podpornih vektorjev
TF	Term frequency	pojavitev besede
IDF	inverse document frequency	inverzna pojavitvena frekvenca
RBF	radial basis function	funkcija na radialni osnovi
NLTK	natural language toolkit	orodje za obdelavo in analizo naravnega jezika

Povzetek

Naslov: Identifikacija in analiza oglasnih člankov

Avtor: Kristjan Pičulin

Za diplomsko nalogo sem ustvaril program, ki prepoznava ali je spletni članek oglas ali pravi članek in tudi analiziral rezultate, ki jih je program dal. Analiziral sem, zakaj program določene članke označi kot prave članke ali oglase, zakaj nekatere članke narobe označi in kaj najbolj vpliva na njegovo delovanje. Zanimalo me je predvsem, kako ločiti navadne članke od oglasov. Program sem ustvaril v programskem jeziku python. Uporabil sem knjižnice selenium, pyqt, sklearn in podobne. Dosegel sem kar dobro in uspešno delovanje programa glede na moje cilje ter odkril mnogo zanimivega glede oglasov in novic.

Ključne besede: strojno učenje, oglasi, oglaševanje, SVM, Naivni Bayes, SGD, TF-IDF.

Abstract

Title: Discovery and analysis of advertisements from textual data

Author: Kristjan Pičulin

For my thesis i made a program, that recognizes if a web article is a pai advertisement or if it is a real news article and also analized the results that were made by the program. I analized why articles are classified the way they are, why are some articles misclassified and what things affect how program is recognizing articles. I was especially interested in a way to separate news articles and advertisements. The program was made in Python programming language. I used libraries such as: pyqt, sklearn and similar. I was quite successful in making the program work the way i wanted and i also found out many interesting things about articles and advertisements.

Keywords: machine learning, paid articles, advertisement, SVM, Naive Bayes, SGD, TF-IDF.

Poglavje 1

Uvod

V današnjem času smo vse bolj obremenjeni z informacijami. Z rastjo spleta so postale vse lažje dostopne in vse več jih je. S količino pa se je tudi povečalo število neuporabnih ali celo zavajajočih informacij. Mnogo teh informacij je spletnih člankov, ki jih dobimo na raznih spletnih straneh. Zanimajo nal lahko novice iz tehnike, gospodarstva, črne kronike, mode ali marsičesa drugega, povsod lahko najdemo nekaj zase. Ker je vsebina informacij zelo pomembna, sem se odločil narediti program, ki nam pomaga določiti ali je članek pristen ali oglaševanje ter narediti analizo, kako so oglasi drugačni od preostalih člankov. Predvsem je bil namen, da bi lahko odkril priktrito oglaševanje.

Program sem izdelal v programskem jeziku python s pomočjo orodja pycharm. Uporabil sem knjižnice sklearn, NumPy, SciPy, PyQt, matplotlib in nltk. Za uporabniški vmesnik sem uporabil ogrodje Qt. Program deluje po principu algoritma SVM (Support Vector Machine) z linearnim jedrom. Izberemo si pa lahko tudi druge metode, kot so SGD, Naivni Bayes in SVM z jedrom RBF.

Program deluje tako, da se najprej nauči na predpripravljeni učni množici, ki je razdeljena na 2 razreda; oglase in prave članke. Nato za vsak nov primer napove ali je oglas ali pravi članek. Izpiše pa tudi 5 najbolj podobnih člankov, 10 najpomembnejših besed v članku in v primeru oglasov izpiše,

kateri predmet ali storitev je oglaševan.

Članke za učno množico sem dobil z raznih spletnih strani. Članki so v slovenskem in angleškem jeziku ter so razdeljeni na različne rubrike (svet, Slovenija, tehnologija ipd.). Program torej podpira slovenski in angleški jezik.

Moj cilj je bil, da oglase zaznava čimbolje. Bolj pomembno mi je bilo, da oglas pravilno zazna kot oglas. Manj huda napaka je, če novico zazna kot oglas, kot, če oglas napačno prepozna kot novico oz. mi je bil bolj pomemben priklic za oglase kot novice.

Poglavje 2

Opis uporabljenih orodij in metod

Pri izdelavi naloge sem uporabil veliko različnih orodij in knjižnic, ki jih v tem poglavju nameravam opisati.

Uporabil sem programski jezik python, knjižnice sklearn, NumPy, SciPy, nltk in PyQt ter orodja Pycharm podjetja JetBrains in qtDesigner.

2.1 Uporabljene knjižnice in orodja

2.1.1 Python

Slika 2.1: Pythonov logotip.

Python je višjenivojski programski jezik. Njegova prednost je predvsem lahka berljivost in hitra implementacija. Ustvaril ga je nizozemski programer

Guido van Rossum. [3] Prva različica je izšla leta 1991 [12].

Ta programski jezik sem izbral, ker ima veliko uporabnih knjižnic kot na primer sklearn, ki so mi zelo olajšale delo. Implementacija v Pythonu je tudi zelo hitra in enostavna. Zaradi osredotočenosti na berljivost kode je odpravljanje napak zelo hitro.

Python namesto prevajalnika uporablja tolmača (interpreter), ki kodo sproti prevaja. Namesto zavitih oklepajev je koda razdeljena s pomočjo presledkov. Jezik je bil zgrajen tako, da namesto da bi imel cel kup funkcionalosti, je raje bolj razširljiv. Osnovna ideja Pythona je predstavljena s temi točkami [17]:

- Lepo je lepše od grdega
- Eksplicitno je boljše kot implicitno
- Preprosto je boljše kot kompleksno
- Kompleksno je boljše kot zapleteno
- Berljivost šteje

Kratka zgodovina

Prva verzja imenovana 0.9.0. je izšla februarja 1991. Na tej stopnji je jezik že podpiral razrede z dedovanjem, lovljenje in obravnavo napak ter osnovne podatkovne tipe.

Verzija 1.0 je izšla januarja 1994 in je dodala nekaj novih podatkovnih tipov. Verzija 2.0 je izšla oktobra leta 2000, zadnja podverzija 2.7, ki je široko uporabljena še danes, je izšla junija 2009.

Verzija 3.0 je izšla decembra 2008 in je prinesla veliko novih sprememb. Verzija 3.0 in podverzije niso združljive s prejšnjimi verzijami, saj je bila sintaksa ponekod spremenjena (na primer funkcija print, v kateri moramo zdaj stavek, ki ga izpišemo napisati v oklepaju funkcije podobno kot v programskem jeziku Java). Spremembe so bile narejene zato, da se je popravilo večje napake pri oblikovanju jezika [4].

2.1.2 Scikit-learn (sklearn)

Slika 2.2: Scikit-learn logotip.

Scikit-learn je knjižnica za programski jezik Python, ki vsebuje orodja za strojno učenje. Napisana je v programskem jeziku C++ in je zaradi tega delovanje hitrejše, kot če bi bila napisana v čistem Pythonu [15].

Za mene so bila predvsem zanimiva orodja za tekstovno analizo in učenje. To so predvsem orodja za ustvarjenje in preobdelavo korpusa (ustvarjenje učne množice), orodja za učenje modela in klasifikacijo, ter funkcije za ovrednotenje klasifikacije.

Od tukaj sem pobral funkcijo SVM z linearnim jedrom za učenje in predikcijo, saj se je izkazala za najbolj primerno. Preizkusil pa sem tudi druge metode, kot je Naivni bayes in SVM z jedrom rbf.

2.1.3 NumPy

Slika 2.3: NumPy logotip.

NumPy je knjižnica za programski jezik Python namenjena predvsem za znanstvene namene. Vsebuje številne podatkovne tipe in funkcije za hranjenje in obdelavo različnih podatkov [9].

Jaz sem predvsem uporabljal podatkovni tip za hranjenje korpusa.

2.1.4 SciPy

Slika 2.4: SciPy logotip.

SciPy je odprtokodna knjižnica za programski jezik Python, ki vsebuje orodja za matematiko in znanost. Vsebuje orodja in metode, kot so Fourierjeve Transformacije, Interpolacije in orodja za statistiko [16].

Zadnja različica je 1.15, ki je izšla julija 2018

Slika 2.5: Natural Language ToolKit.

2.1.5 Nltk

Nltk (Natural Language ToolKit) je zbirka orodij za Python za obdelavo naravnega jezika. Vsebuje orodja in metode za lematizacijo, tokenizacijo in prepoznavanje besedila. Nltk ima že vgrajenih veliko različnih virov besedil in korpusov. Podpira pa tudi veliko število jezikov, kot so ruščina, angleščina španščina ipd [8].

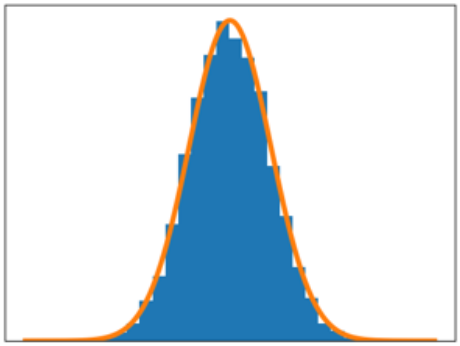
Jaz sem to knjižnico potreboval predvsem za odstranjevanje mašilnih besed in ločil ter lematizacijo. Na žalost pa večina orodij privzeto ne podpira slovenščine.

2.1.6 Matplotlib

Slika 2.6: Matplotlib logo.

Matplotlib je knjižnica za programski jezik Python, namenjena vizualizaciji in risanju grafov ter diagramov. Podpira stolpične grafe, barvne diagrame, razpršilne grafe ipd.

To knjižnico sem uporabil za vizualizacijo in analizo podatkov. Predvsem sem rabil modul pyplot.



Slika 2.7: Primer histograma ustvarjenega s knjižnico matplotlib.

2.1.7 Qt, PyQt in Qt designer

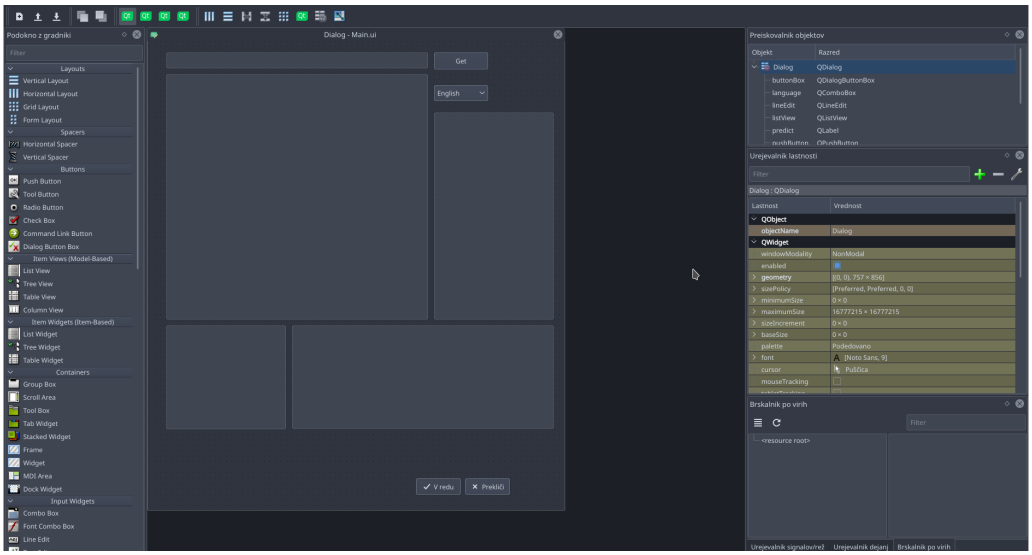
Slika 2.8: Qt logotip.

Za uporabniški vmesnik programa sem uporabil ogrodje Qt. Qt je ogrodje za ustvarjanje uporabniških vmesnikov, ki je na voljo na vseh večjih platformah in podpira večino popularnih programskih jezikov, kot so Python, C in C++. Qt uporablja tudi kar nekaj večjih produktov kot so na primer KDE Plasma 5, Panasonic Avionics in LG Electronic Smart TV [13]. Trenutno sta podprti dve različici: starejši Qt4 in novejši Qt5 .

PyQt je knjižnica, ki omogoča uporabo ogrodja Qt v programskem jeziku Python. Razvit je s strani angleškega podjetja Riverbank Computing.

Obstaja v dveh različicah PyQt4 za Qt4 in PyQt5 za Qt5. Deluje pa za Python 2 in Python 3. PyQt 4 sicer ni več podprt in je za novejše projekte priporočen PyQt5 [11].

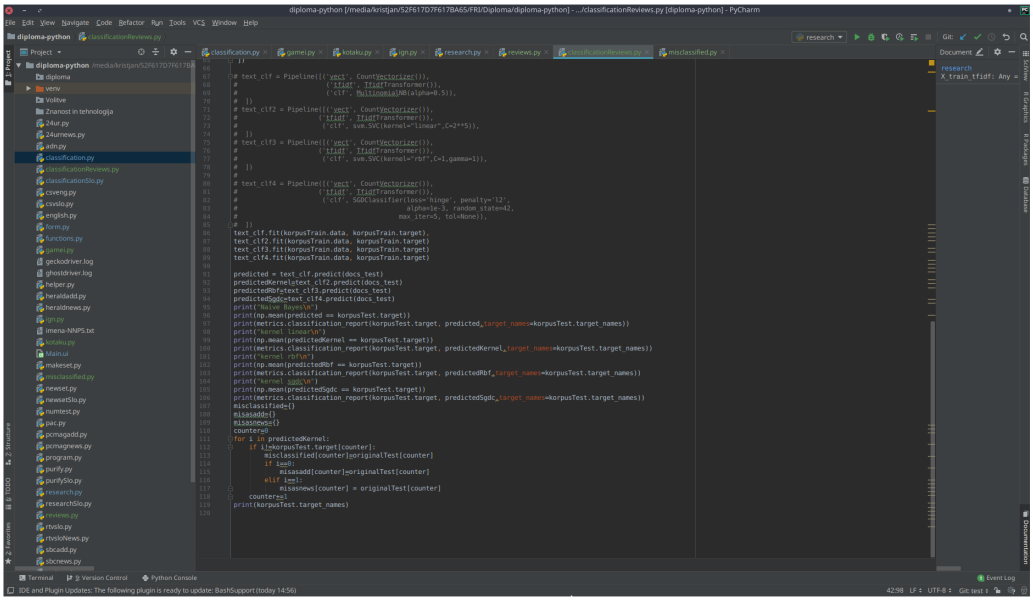
Qt Designer je orodje za izdelavo uporabniškega vmesnika z ogrodjem Qt. Okna se razvija s "Kar vidiš, to dobiš"(What you see is what you get) principom [13].



Slika 2.9: Qt Designer.

Qt designer je bil uporabljen za izdelavo vmesnika in knjižnico PyQt4 ter ogrodje Qt4. Starejšo verzijo sem uporabil zaradi tega, ker sem prej že večkrat delal z PyQt 4 ko sem moral razviti uporabniški vmesnik za aplikacijo v Pythonu. Kot alternative so mi bile na voljo tudi orodje GTK+ in knjižnica PyGTK.

2.1.8 PyCharm



Slika 2.10: Glavno okno v pycharmu.

Pycharm je integrirano razvojno okolje (Integrated Development Environment - IDE) za programski jezik Python. Razvit je s strani podjetja JetBrains, ki je sicer najbolj znano po razvojnem okolju za Javo, imenovanem Intellij Idea. Zraven tega pa so ustvarili tudi precej drugih razvojnih okolij za druge programske jezike npr. Ruby mine za Ruby in Clion za C ter C++ [10].

PyCharm ima vse osnovne gradnike Integriranega razvojnega okolja kot so urejevalnik besedila, razhroščevalnik in drugo. Prav tako je razširljiv s pomočjo vtičnikov.

PyCharm sem uporabil zaradi pozitivnih izkušenj z njim v preteklosti, saj sem ga uporabljal čez celotno obdobje študija, ko sem kaj delal v programskem jeziku Python.

2.1.9 Selenium

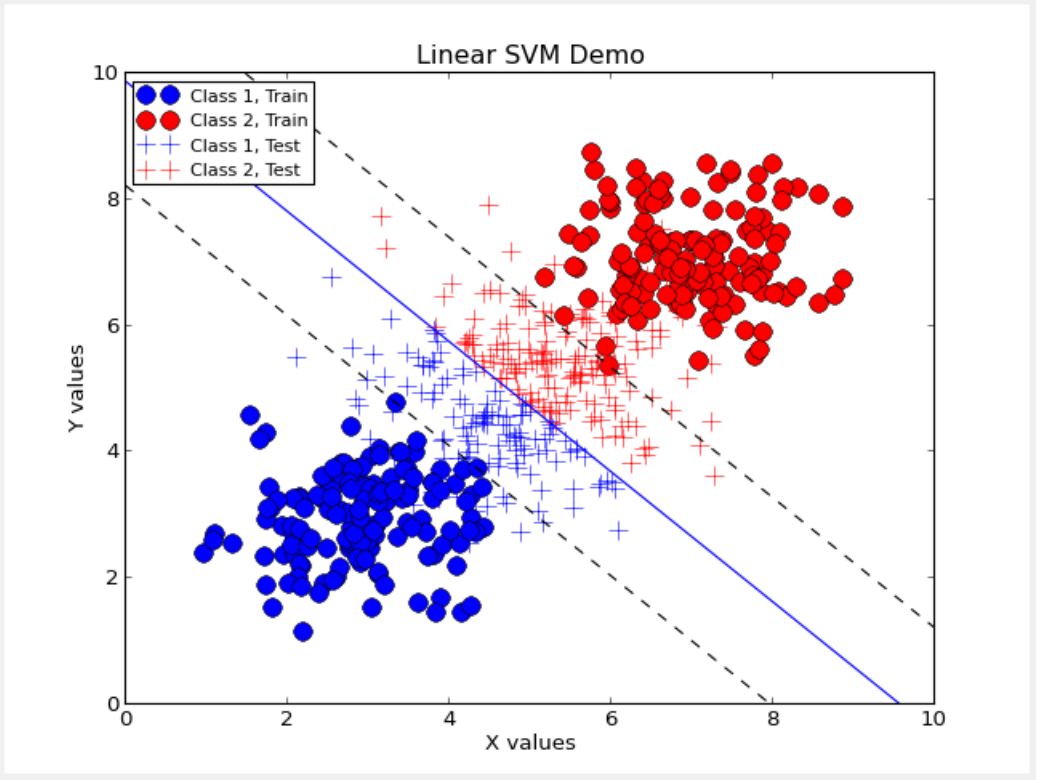
Selenium je orodje za avtomatizacijo brskalnikov. S pomočjo pripadajoče knjižnicce za Python sem pri zbiranju podatkov pridobil vsebine člankov s spletnih strani (Web scrapping).

Ugotovil sem, da je ta način najlažji za pridobivanje podatkov, saj deluje tudi na straneh, ki so prikazane s pomočjo JavaScripta. Način, ki je narejen v programu, deluje na večini spletnih strani, ampak ne na straneh, ki imajo nekoliko drugačno sestavo HTML značk.

2.2 Delovanje predikcijskih metod

V tem podpoglavju bom opisal delovanje štirih metod za predikcijo, ki sem jih uporabil v programu: SVM z linearnim jedrom, SVM z RBF jedrom, SGD in Naivni Bayes

2.2.1 SVM z linearnim jedrom



Slika 2.11: Linearni SVM.

Ta metoda je privzeta metoda v mojem programu, saj sem ugotovil, da za doseganje mojih ciljev deluje najbolje.

SVM deluje tako, da vse primere porazdeli v nek koordinatni sistem in nato skuša narediti neko črto ali ravnino med njimi, ki jih čimbolje ločuje. Tako ko dobimo nov primer, nato gleda na kakšen del polja pade [2].

Vzemimo nek preprost primer. Recimo, da imamo neko zbirko živali v kateri so sesalci in žuželke. Nato želimo narediti sistem, ki jih bo ločeval na naboljši možni način. Te živali nato predstavimo kot nek vektor z dvema parametroma. Kot vektor X vzamemo število nog, kot parameter Y vzamemo težo. Te vrednosti nato normaliziramo in jih nato vrišemo v koordinatni sistem.

Cilj je nato narisati črto, ki bo imela razdaljo med primeri iz obeh enako oddaljeno in je ta čimvečja. To črto imenujemo hiperravnina. Najbližjim primerom hiperravnine pravimo podporni vektorji, razdalji hiperravnine od teh vektorjev pa pravimo rob.

Tako naredimo neko hiperravnino, ki ločuje žuželke od sesalcev. Za vsek nov primer nato postavimo v koordinatni sistem in pogledamo, na kateri konec črte pade. Recimo postavimo domačo mačko v koordinatni sistem. Mačka ima 4 noge in okrog 6 kilogramov. To postavimo v koordinatni sistem in vidimo, da pade med sesalce.

V tem preprostem primeru smo uporabili le dve dimenziji. Sistem pa deluje tudi na večih dimenzijah. V mojem primeru v programu je dimenzij 160, saj so podatki predstavljeni kot vektor z TF-IDF vrednostmi 160-ih besed.

2.2.2 SVM z RBF jedrom

Pri nekaterih primerih tudi če damo vse primere v koordinatni sistem, ponavadi ne moremo narediti hiperravnine, ki bi dobro ločevala te primere. Za take primere lahko uporabimo jedrne funkcije [1]. Pri jedrni funkciji RBF (Radial Basis Function) dodamo še dodatne dimenzije v naš koordinatni sistem (npr. spremenimo dvodimenzionalni koordinatni sistem v tridimenzionalnega) in tako lažje vrišemo ravnino [14].

2.2.3 Naivni Bayes

Pri Naivnem Bayesovem klasifikatorju s pomočjo učne množice podatkov aproksimiramo apriorne in pogojne vrednosti razredov pri dani vrednosti atributa [6].

Enačba je:

$$P(r_k|V) = P(r_k) \prod_{i=1}^{a} \frac{P(r_k|v_i)}{P(r_k)} \tag{2.1}$$

Žival/Lastnost	`Živi v vodi`	`Težja od 5 kg`	`Večja od 10 cm`	Tip živali
`Zlata ribica`	Da	Ne	Ne	Riba
`Kit`	Da	Da	Da	Sesalec
`Netopir`	Ne	Ne	Ne	Sesalec
`Morski pes`	Da	Da	Da	Riba
`Losos`	Da	Ne	Da	Riba
`Kenguru`	Ne	Da	Da	Sesalec

Tabela 2.1: Primer živali.

Vzemimo za primer živali, ki so razdeljene na sesalce in ribe v spodnji tabeli:

Zdaj vzamimo za primer mačko in jo skušamo klasificirati z Naivnim Bayesom. Vzamimo m-oceno enako 2. Mačka ne živi v vodi, je težja od 5 kg in je večja od 10 cm. Zdaj želimo vedeti ali je na podlagi zgornjih učnih primerov mačka sesalec ali riba.

Prvo vzamemo splošno verjetnost, da je žival sesalec P(Sesalec). Po enačbi

$$P(r_k) = \frac{n_k + 1}{n + m_0} \tag{2.2}$$

je ta verjetnost enaka:

$$P(Sesalec) = \frac{3 + 1}{6 + 2} = 0,5 \tag{2.3}$$

Nato zračunamo vse pogojne verjetnosti za sesalca po enačbi

$$P(r_k) = \frac{n_{k,i} + mP(r_k)}{n_i + m} \tag{2.4}$$

Primer za pogojno verjetnost, da je sesalec in ne živi v vodi:

$$P(Sesalec|Ne\_zivi\_v\_vodi) = \frac{2 + 2 * 0,5}{3 + 2} = 0,75 \tag{2.5}$$

To je, ker imamo 2 sesalca od 3, ki ne živita v vodi. Izračunamo še preostale pogojne verjetnosti:

$$P(Sesalec|Teza\_vecja\_od\_5kg) = \frac{2 + 2 * 0,5}{3 + 2} = 0,6 \tag{2.6}$$

$$P(Sesalec|Velikost\_vecja\_od\_10cm) = \frac{2 + 2 * 0,5}{4 + 2} = 0,5 \qquad (2.7)$$

Na koncu izračunamo skupno verjetnost da je mačka sesalec po bayesovi enačbi:

$$P(Sesalec|Voda = Ne, Teza = Da, Velikost = Da) = \frac{0,75}{0,5} * \frac{0,6}{0,5} * \frac{0,5}{0,5} * 0,5 = 0,9 \qquad (2.8)$$

Verjetnost, da je mačka sesalec, je enaka 0,9. Če po istem postopku izračunamo verjetnost, da je mačka riba, dobimo rezultat 0.06. Se pravi da je glede na naš model mačka sesalec.

V programu se sicer rabi multinominalni Naivni Bayes in dela s pojavitvami besed v posameznem dokumentu.

2.2.4 SGD

SGD (Stochastic Gradient Descent) je iterativna metoda, ki optimizira neko diferencialno objektivno funkcijo. Nek problem optimiziramo, dokler se rešitev ne izboljšuje več. Razvila sta jo Herbert Robbins in Sutton Munro leta 1951 [7].

2.3 Metode za ovrednotenje učinkovitosti delovanja programa

Najbolj uporabljane metode za ugotavljanje, kako dober je algoritem za klasifikacijo, so klasifikacijska točnost, priklic (recall), preciznost (precision) in F-ocena (F-score). Te rezultate dobimo tako, da algoritem preverimo na neki testni množici in pogledamo, kako dobro je napovedal rezultate.

2.3.1 Term frequency- Inverse document frequency

TD-IDF nam pove, kako zelo je pomembna posamezna beseda v dokumentu. Dobimo jo tako, da število pojavitev besede pomnožimo z IDF vrednostjo

besede. IDF vrednost je vrednost, ki pove, kako redka je beseda v korpusu. Formula je:

$$idf_b = log\frac{N}{n_b} \tag{2.9}$$

kjer je N število vseh dokumentov v zbirki in nb število vseh dokumentov z besedo b. Potem lahko utež besede v dokumentu dobimo tako:

$$tfidf_{b,d} = f_{b,d} * idf_b \tag{2.10}$$

kjer je f(b,d) frekvenca besede v dokumentu in idf(b) idf vrednost besede.

2.3.2 Klasifikacijska točnost in večinski klasifikator

Klasifikacijska točnost nam pove, koliko primerov je v testni množici algoritem prav napovedal. Na primer če je klasifikacijska točnost 0,8 pomeni, da je algoritem v 80 odstotkih primerov prav napovedal rezultat. Klasifikacijska točnost je definirana z:

$$T = \frac{N^{(p)}}{N} \times 100 \tag{2.11}$$

Kjer je N število vseh danih primerov in $N^{(p)}$ število pravilnih rešitev primerov [5].

Večinski klasifikator je klasifikator, ki vse primere označi kot večinski razred npr. pomeni, da, če imamo v množici 100 živali, ki so razdeljene na ribe in sesalce, 60 rib in vse živali označimo kot ribe, je klasifikacijska točnost večinskega klasifikatorja enaka 0,6.

2.3.3 Priklic, preciznost in F-ocena

Priklic pove, kolikšen odstotek pomembnih dokumentov je bil pravilno klasificiran glede na vse pomembne dokumente. Npr. če ima v mojem programu nek algoritem priklic za oglase 0,6, pomeni, da je 60 odstotkov vseh oglasov v testni množici pravilno klasificiral kot oglase. Enačba za priklic je:

$$recall = \frac{TP}{TP + FN} \tag{2.12}$$

kjer TP pove število vseh pravilno klasificiranih pozitivnih primerov in FN pove število vseh nepravilno klasificiranih pozitivnih primerov [5].

Preciznost pove, kolikšen delež vseh primerov, ki so bili klasificirani kot pozitivni, je bilo pravilno klasificiranih. Če ima v mojem programu algoritem preciznost enako 0,8 pomeni, da je 80 odstotkov vseh dokumentov, ki jih je klasificiral kot oglase dejansko oglasov. Enačba za preciznost je:

$$precision = \frac{TP}{TP + FP} \tag{2.13}$$

kjer so TP vsi pravilno klasificirani pozitivni primeri in FP vsi primeri, ki so bili nepravilno klasificirani kot pozitivni.

F-ocena pove razmerje med priklicem in preciznostjo. Enačba za F-oceno je:

$$F = \frac{2 \times recall \times precision}{recall + precision} \tag{2.14}$$

Poglavje 3

Izdelava in delovanje programa

3.1 Zbiranje podatkov

3.1.1 Priprava

Najprej sem ustvaril osnoven izgled podatkov. Podatke sem razdelil v dva razreda: na oglase (Ad) in prave članke (News). Nato sem začel iskati primerne vire člankov.

Članki in oglasi so morali biti dovolj raznoliki, da sem lahko ustvaril dobre učne in testne množice. Oglasi in članki so torej morali biti iz različnih skupin. Oglasi na primer vsebujejo oglase za storitve, predmete, prireditve, turistične destinacije ipd. Morali so tudi pokrivati čim večje število storitev in predmetov.

Novičarski članki so porazdeljeni po rubrikah: črna kronika, Slovenija, svet, tehnologija, kultura ipd. Viri so morali biti iz čimveč različnih spletnih strani in čimveč različnih virov. Članke sem nabral v slovenščini in angleščini.

Med glavnimi viri v angleščini so strani Wtop, Pc magazine, BBC, The Strabane Chronicles, The signal in Fermanagh Herald. Med slovenskimi viri sta glavna 24ur.com in rtvslo.si.

3.1.2 Izdelava skript za zbiranje podatkov in zbiranje podatkov

Skripte za zbiranje podatkov sem napisal v programskem jeziku Python. Podatke sem se odločil, da bom uporabil v tekstovni obliki v .txt datotekah. V tekstovne datoteke sem jih dobil s pomočjo Web scrappinga.

Iz strani sem pobral naslov, ki je ponavadi v značkah h1 in besedilo, ki je ponavadi v značkah p. To deluje na večini strani. Na straneh, ki nimajo take sestave, pa skripte ne delujejo.

Sprva sem nameraval skupaj z besedilom iz strani pobrisati dele besedila, ki jih nisem potreboval, (na primer obvestila o piškotkih in razna pravna obvestila) vendar se je to izkazalo kot nepraktično in sem to raje naredil kasneje pri pripravi učne in testne množice.

Za pridobivanje besedila sem uporabil knjižnico Selenium z gonilnikom za Chrome. Sprva sem nameraval uporabiti knjižnico Beautiful Soup, vendar se je izkazalo, da ne deluje na straneh, ki so prikazane s pomočjo Javascripta in je na večini sodobnih strani neuporabna.

Skripte za zbiranje podatkov delujejo tako, da prvo podam kazalo ali seznam strani, potem se stran odpre v brskalniku, kjer nato pregledam vse HTML značke za uporabno vsebino, ki jo nato shranim v nek niz. Ta niz se nato shrani v tekstovno datoteko z naključno generiranim (hash) imenom. Ime mora biti naključno, zato, da pri sestavi učnih in testnih množic ne prepišemo datotek.

Seznam teh datotek in njihovih lastnosti sem dal v csv datoteko. Naredil sem 2 datoteki. Eno za angleščino in eno za slovenščino.

3.1.3 Razdelitev na učno in testno množico

Najprej sem iz vseh tekstovnih datotek za učno množico pobrisal vse stavke in besede, ki so bile nekoristne ali celo škodljive za moj program. Taki stavki so npr. obvestila o piškotkih, pogoji uporabe in pravna obvestila.

Nato sem naredil glavno učno in testno množico za oba jezika (slovenski

in angleški jezik). Učni množici sem sestavil tako, da sem iz vseh datotek iz naključno izbranih virov naključno zbral nekaj pravih člankov in jih dal v mapo news in nekaj oglasov in jih dal v mapo add. Pri angleški množici je oglasov 744 in pravih člankov 2192. Pri slovenski je oglasov 1446 in pravih člankov 1054.

Pri testnih množicah sem izbral nekaj virov, ki so bili neodvisni od virov za učno množico in sem oglase in novice iz posameznih strani kopiral v pripadajoče mape, kot pri učni množici. Za vsak vir sem kopiral nekje med 50 in 100 člankov. V testni množici je 375 oglasov in 308 novic. Pri testni množici so bili članki prečiščeni samo osnovnih besed ter znakov, kot so znaki za copyright in podobno, saj so obvestila za piškotke na straneh, ki jih vnašamo v program na vsaki strani drugačna in ni mogoče predvideti, kakšna so prav za vsako stran.

Zelo pomembno je, da imata testna in učna množica različne vire. To je zato, ker bi če bi imela iste vire, algoritem potencialno prepoznal članke že zaradi podobnih imen in sestave. Zaradi tega bi pri ovrednotenju dobili netočne rezultate. Pomembno je tudi, da je imel dosti različnih primerov iz večih različnih rubrik.

3.1.4 Izdelava programa

Prva stvar, ki sem jo moral narediti, je bila izbira najbolj učinkovitega algoritma. Naredil sem nekaj preizkusov in ugotovil sem, da je metoda SVM z linearnim jedrom še najbolj učinkovita. To metodo sem si izbral kot osnovo za nadaljnje delo.

Nato sem naredil funkcijo za predpripravo teksta. To je funkcija, ki mi iz teksta pobriše ločila in spremeni besedilo v male črke. Če je besedilo angleško, naredi tudi lematizacijo in korenjenje.

Za pridobivanje besedila iz spletnih strani sem uporabil kar predelano funkcijo, ki sem jo uporabil pri zbiranju podatkov.

Po tem sem naredil nek prototip programa, ki mi je za neko podano spletno stran povedal ali je članek ali pa oglas. Ko sem bil zadovoljen z

delovanjem, sem dodal nove funkcije kot na primer za pridobivanje najpomembnejših besed in pridobivanje podobnih člankov.

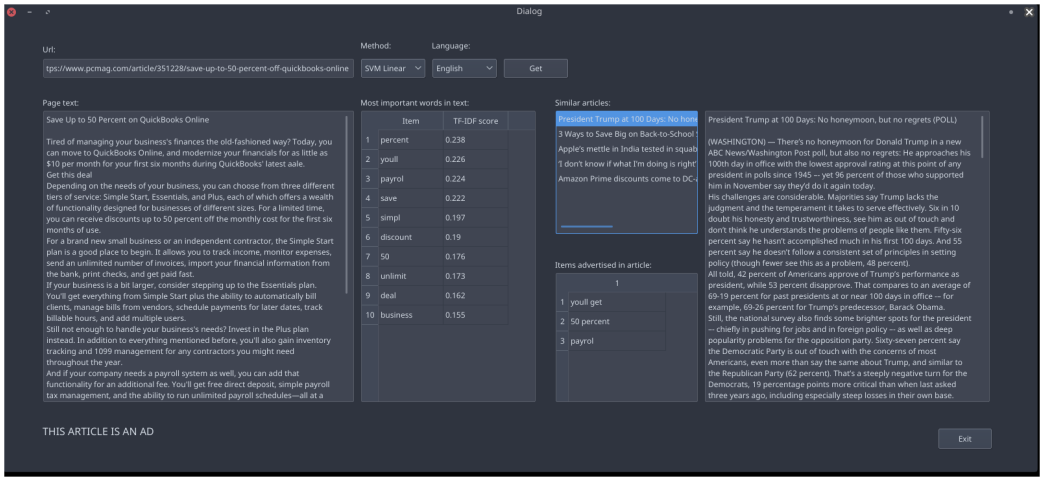
Ko so bile vse osnovne funkcije pripravljene, sem se lotil priprave grafičnega vmesnika. Vmesnik sem naredil v programu QtDesigner in sem ga nato z orodjem pyuic4 pretvoril v Pythonov razred. Vse funkcije sem nato ustrezno preoblikoval in uporabil v programu.

Program sem nato razširil tako, da sem dodal funkcijo, ki je za primer, če je program oglas, napovedala stvari, ki jih oglašuje. Napove do 3 različne (možne) predmete.

Zadnja funkcionalnost, ki sem jo dodal, je izbira med različnimi možnimi metodami za napovedovanje. Privzeto je ostal izbran SVM z linearnim jedrom. Dodal sem še SVM z RBF jedrom, SGCD in naivni Bayes.

Zadnja stvar je bila to, da sem še izpilil izgled uporabniškega vmesnika.

3.2 Uporaba programa



Slika 3.1: Glavno okno programa.

Najprej v testovno okno vpišemo url naslov strani, ki jo želimo ovrednotiti. Nato iz spustnega seznama izberemo metodo za napovedovanje. Izbiramo lahko med štrimi metodami: SVM z linearnim jedrom (SVM Linear - ta opcija je privzeto izbrana), SVM z RBF jedrom (SVM RBF), Naivni Bayes (Naive Bayes) in SGD (SGD).

Potem si iz spustnega seznama izberemo angleški ali slovenski jezik glede na to, v katerem jeziku je stran. To je pomembno, saj je predpriprava besedila za prepoznavanje v obeh jezikih drugačna.

Ko je to narejeno, pritisnemo gumb Get. Potem počakamo nekaj časa. Če je uspešno pridobilo podatke iz strani, se po krajšem čakanju izpišejo vsi pomembni podatki. V večjem oknu se izpiše besedilo strani. Pod tem oknom se izpiše, kako je algoritem prepoznal članek ali kot oglas ali kot pravi članek.

Desno od okna z besedilom je seznam desetih najpomembnejših besed v besedilu v vrstnem redu od najpomembnejše do najmanj pomembne. Poleg tega zraven izpiše TF-IDF oceno besede.

Potem imamo seznam z naslovi petih člankov iz učne množice, ki so najbolj podobni ovrednotenemu članku. S klikom na posamezen naslov si lahko ogledamo vsebino teh člankov v oknu na desni.

V primeru, da je članek oglas, se prikaže tudi okno z do tremi predvidevanimi elementi, ki jih članek oglašuje. To so lahko storitve, predmeti ali lokacije. Lahko pa so samo imena podjetij ali naročnikov članka.

Na zadnje imamo še gumb Exit, ki program zapre.

3.3 Funkcije v programu

V tem podpoglavju bom opisal pomembnejše funkcije, ki jih program uporablja.

3.3.1 Razred Model in njegov inicializator

Razred Model, ki se nahaja v datoteki functions.py, je razred, ki hrani vse pomembnejše informacije o klasifikaciji posameznega razreda. Vse glavne

funkcje od prepoznavanja do učenja na učni množici so v tem razredu.

Pri inicializaciji najprej nastavi poti do angleške in slovenske množice. Nato naloži datoteke iz korpusov in jih preprocesira (odstrani ločila, lematizacija, korenjenje...). Po tem inicializira cevovod za predikcijo z linearno SVM metodo (ta metoda je privzeta metoda za prepoznavanje v programu) in nauči model na učni množici.

Zadnje kar naredi je, da nekaj preostalih spremenljivk, ki jih rabi kasneje postavi na privzete vrednosti.

3.3.2 Predpriprava besedila

Predpriprava besedila se zgodi s klicem funkcij preprocess za angleško besedilo in preprocessSlo za slovensko besedilo.

Funkciji najprej naložita seznam vseh ločil, ki jih kasneje pobrišeta. Nato iz besedila odstranita vse znake za novo vrstico (newline) in spremenita besedilo v male črke in pobrišeta ločila. Nato odstrani odvečne presledke na krajih kjer so predolgi presledki.

Če je besedilo v angleščini naredi tudi korenjenje in lematizacijo. Za slovenska besedila orodja ne podpirajo teh funkcij.

3.3.3 Funkcije za nastavljanje predikcijskih metod

Ker v programu lahko izbiramo med različnimi metodami, sem za vsako naredil posebej funkcijo, ki jo nastavi, ko jo izberemo iz spustnega seznama.

Vse funkcije delujejo popodebnem principu. Najprej nastavijo cevovoda za oba jezika (slovenski in angleški jezik) in nastavijo parametre za vsako metodo posebej. Nato nauči oba modela na učnih podatkih.

Cevovodi so nastavljeni tako, da najprej besedilo spremenijo v vektor s številom besed v besedilu (Count Vectorizer). Ta funkcija ima nastavljene parametre ngram, ki je med 1 in 2, min df, ki je enak 0,1, kar pomeni da ne upošteva besed, ki so v manj kot 10 odstotkih besedil, max df na 0,7, kar pomeni da izpusti besede, ki se pojavljajo v več kot 70 odstotkih besedil in

max features enak 160, kar pomeni, da je vektor dolg 160 besed. Naslednji del cevovoda je TF-IDF transformer, ki vektor pojavitev besed spremeni v vektor TF-IDF vrednosti. Zadnji del cevovoda kliče metodo za predikcijo. Tukaj so tudi nastavljeni parametri za posamezne metode.

Parametri za posamezne metode so naslednji: Linearni SVM ima nastavljen C na 1000 (to pove kako strogo naj kaznuje primere pri učenju, ki jih narobe postavi), Naivni Bayes ima alfa nastavlen na 0.5, SVM RBF ima nastavljen C na 1 in gamma na 1.

3.3.4 Klasifikacija besedila iz spletne strani

Najprej funkcija imenovana Fit besedilo predprocesira z zgoraj opisanimi funkcijami, nato stran klasificira in rezultat shrani v spremenljivko prediction. Kateri model za klasifikacijo in funkcijo za predpripravo besedila funkcija uporabi, je odvisno od izbranega jezika. Nato program kliče funkcijo verdict, ki vrne, kakšna je vrednost spremenljivke prediction.

3.3.5 Vračanje najpomembnejših besed

Najprej se v funkciji getTfidArticle besedilo spremeni v matriko s številom besed in s funkcijo Tfidf Vectorizer v matriko s TF-IDF vrednostmi besed v besedilu. V tej funkciji se shranijo tudi besede, uporabljene v besedilu v spremenljivko featureNames. Ta funkcija nato vrne matriko z TF-IDF vrednostmi.

To matriko nato uporabi funkcija topTenWords. Ta funkcija nato vse besede da v slovar, ki ima za ključ besedo in kot vrednost TF-IDF vrednost besede. Funkcija slovar nato posortira in ga vrne.

3.3.6 Vračanje podobnih člankov

Funkcija getSimilar gre čez cel korpus in s pomočjo kosinusne podobnosti primerja besedilo spletne strani s posamezno datoteko iz korpusa. Nato v slovar shrani za vsako datoteko podobnost z besedilom.

Funkcija nato vrne slovar s petimi najbolj podobnimi datotekami, ki ima za ključ naslove člankov, in za vrednosti besedila člankov, ki se nato prikažejo v programu.

3.3.7 Vračanje oglaševanih storitev ali izdelkov v primeru oglasov

V primeru, če je članek klasificiran kot oglas, program vrne do 3 besede ali besedne zveze, ki naj bi imele nekaj opraviti s tem, kar članek oglašuje. Te besede naj bi bile npr. izdelki, storitve, turistične lokacije ipd. Te besede vrača funkcija getAdvertised.

Funkcija deluje tako, da najprej izlušči 3 najpomembnejše besede glede na TF-IDF vrednot v besedilu. Ugotovil sem, da se oglaševan izdelek ali storitev v oglasih v veliki večini primerov pojavi med prvimi tremi besedami.

Nato gre čez cel članek, da preveri, katere besede se dostikrat pojavljajo skupaj z izbranimi besedami. Če vidi da se neka beseda pojavlja skupaj s katero od teh 3 najpomembnejših besed, jo doda k tej besedi. Preveri pa tudi, ali se kateri 2 besedi izmed teh 3, ki so najpomembnejše, pojavlja skupaj v besedilu in če se ju v seznamu združi. Na koncu še preveri, ali so katere izmed besednih zvez v seznamu oglaševanih besed nadaljevanja prejšnje besedne zveze. Če imamo v seznamu besedni zvezi Nov prenosnik in prenosnik HP bo besedni zvezi združil v nov prenosnik HP. Na koncu ta funkcija ta seznam vrne.

Poglavje 4

Rezultati in analiza

V tem poglavju bom opisal, kako dobro se je program odrezal, in analiziral rezultate. Ovrednotenje in analizo sem naredil samo na angleških člankih, saj pri slovenskih nisem imel dovolj virov in nisem uspel narediti testne ter učne množice, ki bi bili neodvisni.

4.1 Ovrednotenje klasifikacije

4.1.1 Ocenjevanje učinkovitosti programa

Po tem, ko sem izbral algortime za program sem moral oceniti njihovo učinkovitost. Najprej sem naložil korpusa za testno in učno množico z ukazom load files. Testno množico sem shranil v spremenljivko korpusTest, učno množico pa v spremenljivko korpusTrain. Nato sem vsa besedila v obeh korpusih prečistil: pobrisal ločila, spremenil vse besede v majhne črke in naredil korenjenje ter lematizacijo.

Nato sem nastavil cevovode za vse izbrane algoritme in jim nastavil parametre. Nastavil sem tudi, da se v cevovodu pobrišejo vse mašilne besede in besede, ki se prevečkrat ponavljajo v korpusu. Nato sem te cevovode naučil na učni množici s funkcijo fit, ki sem ji za prvi parameter dal seznam besedil v učnem korpusu (korpusTrain.data) in drugi parameter razrede, ki jim ta besedila pripadajo (korpusTrain.target).

Nato sem preveril, kako učinkoviti so algoritmi tako, da sem najprej naredil predikcije na testni množici za vsak algoritem posebej in jih shranil v 4 različne spremenljivke. Nato sem za vsako spremenljivko izpisal matriko zmot in njihove priklice ter preciznosti tako, da sem primerjal z razredi, ki so jih cevovodi napovedali, z razredi v testnem korpusu. Dobil sem naslednje rezultate:

Naivni Bayes	Oglas	Novica	Priklic	Preciznost
Oglas	212	163	0,57	0,91
Novica	20	288	0,94	0,64
Klas. točnost	0,73			
Linearni SVM	Oglas	Novica	Priklic	Preciznost
Oglas	305	70	0,81	0,79
Novica	83	225	0,73	0,76
Klas. točnost	0,76			
SVM z RBF	Oglas	Novica	Priklic	Preciznost
Oglas	293	82	0,78	0,84
Novica	57	251	0,81	0,75
Klas. točnost	0,79			
SGD	Oglas	Novica	Priklic	Preciznost
Oglas	279	96	0,74	0,83
Novica	57	251	0,81	0,72
Klas. točnost	0,76			

Tabela 4.1: Matrika zmot skupaj s priklicem in preciznostjo. V stolpcih so pravi razredi v vrsticah napovedani razredi.

4.2 Analiza

Kot vidimo, ima največjo klasifikacijsko točnost SVM z RBF jedrom. Za njim sta Linearni SVM in SGD. Najslabše pa se je odrezal Naivni Bayes. Vendar sem jaz osebno za privzeti algoritem izbral linearni SVM, ker mi je bil cilj

čimboljše prepoznavanje oglasov, s čimmanj oglasi prepoznanimi kot novice in pri tem se je linearni SVM odrezal najbolje, saj ima najboljši priklic za oglase. Je sicer napačno klasificiral več novic kot prejšnji algoritmi, vendar mi je bilo to meni manj pomembno kot napačna klasifikacija oglasov zaradi tega, ker je cilj tudi prepoznavanje prikritega oglaševanja, kjer bi vseeno ročno pregledali vse novice, ki bi jih klasificiralo kot oglase, raje, kot da bi mi kakšen (prikrit) oglas ponesreči izpustilo.

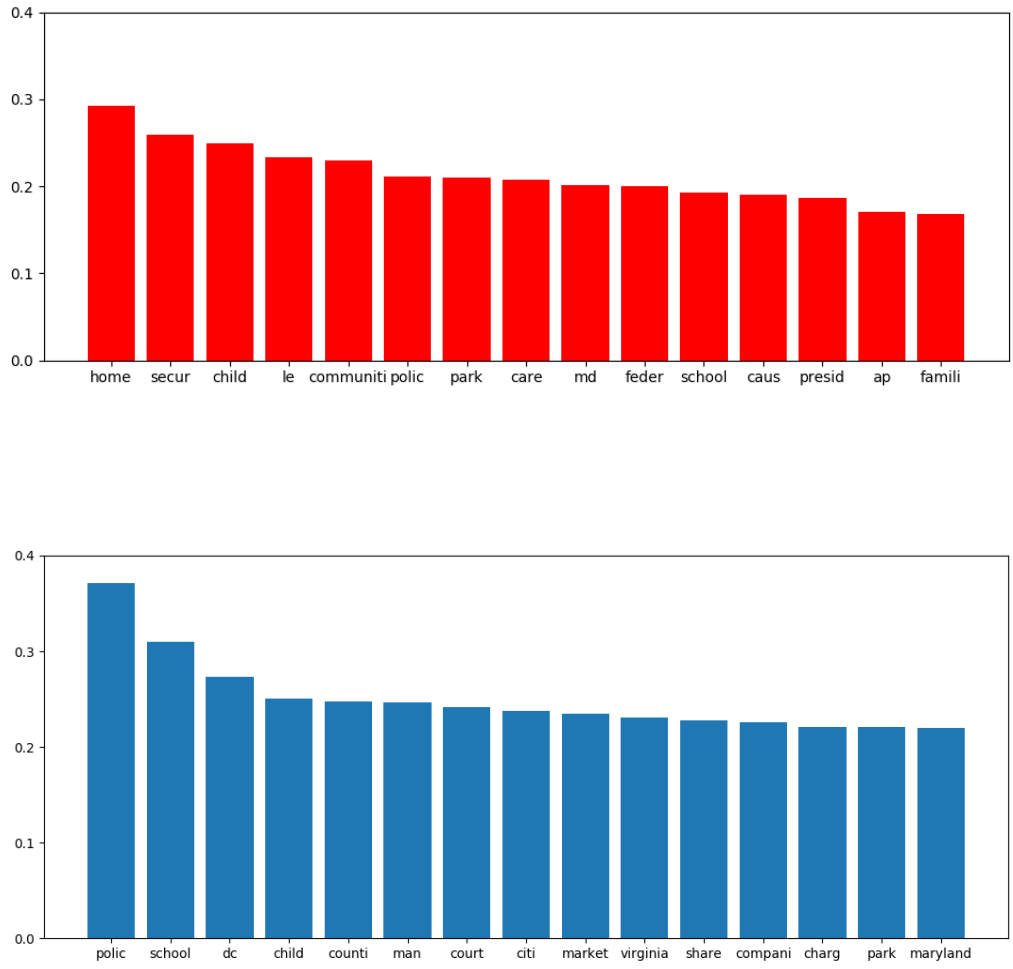
Če bi pa hotel na primer algoritem, ki bi mi avtomatsko brisal oglase, bi raje izbral Naivnega Bayesa, saj ima najboljši priklic za novice in bi si tam raje želel, da mi oglas napačno klasificira kot da mi izbriše novico. Naivni Bayes je samo 3 odstotke novic napačno klasificiral kot oglase.

4.2.1 Splošne razlike med oglasi in novicami

Razlike med oglasi in novicami najlažje dobimo tako, da pogledamo, katere besede imajo največjo povprečno TF-IDF oceno v oglasih in novicah posebej. Da sem to naredil, sem najprej vsa besedila iz korpusa prečistil (odstranil mašilne besede, velike začetnice, ločila...) in pretvoril v TF-IDF ocene z funkcijo tfidf vectorizer. Nato sem naredil dva slovarja, ki sta za vsako besedo naredila seznam vseh TF-IDF ocen za novice in oglase posebej.

Po tem sem naredil še 2 urejena slovarja (Ordered dictionary); enega za oglase in enega za novice, v katera sem dodal izračune povprečij vseh besed. Slovarja sta bila urejena po višini TF-IDF ocene, od najvišje do najnižje. 15 najvišjih besed za oba slovarja sem nato vstavil v grafe in tako izpisal rezultate.

Rezultati so sledeči.



Slika 4.1: 15 najpomembnejših besed v oglasih (zgoraj) in novicah (spodaj) glede na TF-IDF oceno.

Vidimo, da so najpomembnejše besede oz. koreni besed v oglasih home, secur, child, le in community. Iz tega lahko približno vidimo, kaj večina oglasov oglašuje. Veliko oglasov zgleda oglašuje domove, varnostne storitve (verjetno predvsem glede računalniške varnosti) in skupnosti.

V novicah pa vidimo da so najpomembnejše besede polic, school, dc, child in counti. Zgleda, da je dosti člankov govorilo o policiji in šolah. Dosti

člankov pa ima tudi koren dc, kar je logično saj je bil eden od virov lokalni časopis iz Washingtona d.c. Iz teh korenov tudi vidimo, da imajo novice v glavnem besede o dogodkih, medtem, ko imajo oglasi bolj besede v zvezi s predmeti ali storitvami.

Poglejmo še povprečno število besed v oglasih in novicah. Najprej sem s funkcijo count vectorizer naredil seznam s številom vseh besed na dokument. Nato sem šel čez cel seznam in naredil 2 slovarja za oglase in novice, ki sta imela indeks dokumenta za ključ in število besed v dokumentu za vrednost. Nato sem preprosto izpisal povprečja vrednosti za oba slovarja.

Povprečno število besed v oglasih je 291,78 in v novicah 673.91. Se pravi, so oglasi precej krajši. Oglasi se bolj trudijo nekoga prepričati, naj nekaj kupi ali obišče in so bolj osredotočeni na to, da so kratki in jedrnati, medtem ko novice skušajo čimveč in čimbolj podrobno povedati o nečem.

4.2.2 Narobe klasificirani primeri

Poglejmo si še primere, ki so jih algoritimi narobe klasificirali. Zbral sem vse članke, ki so jih posamezni algoritmi narobe klasificirali, in pogledal povprečne TF-IDF vrednosti za posamezne besede v teh primerih.

To sem naredil tako, da sem najprej izluščil, katere datoteke so posamezni algoritmi napačno klasificirali. To sem naredil tako, da sem primerjal tip, ki ga je algoritem predlagal za tisto besedilo, s tipom, ki je dejansko v testnem korpusu. Take primere sem nato dal v pripadajoč seznam (ali seznam s primeri, ki so bili napačno klasificirano kot oglasi, ali primeri, ki so bili narobe označeni kot novice). Nato sem jih pretvoril v TF-IDF vrednosti za misklasificirane primere za oglase miskvalificirane kot novice in za novice miskvalificirane kot oglase na podoben način, kot sem to naredil pri prejšnjem podpoglavju.

Poglejmo si najprej prvih 10 najpomembnejših besed v oglasih po primerih, ki so jih posamezni algoritmi narobe klasificirali:

Pomembnost	Linearni SVM	SVM z RBF	Naivni Bayes	SGD
1.	feder	hous	way	hous
2.	school	visit	team	visit
3.	hous	child	child	school
4.	presid	school	cost	presid
5.	visit	presid	visit	case
6.	world	case	thing	park
7.	better	park	better	better
8.	park	better	presid	child
9.	communiti	cost	communiti	depart
10.	use	market	hour	market

Tabela 4.2: 10 najpomembnejših besed v novicah, ki so bile napačno klasificirane kot oglasi.

Vidimo, da je večina algoritmov narobe klasificirala primere, ki so imeli podobne besede, verjetno celo večinoma iste članke. Večina je članke narobe klasfificirala zaradi korenov besed visit in hous, ki se velikokrat pojavljata v oglasnih člankih in so tam bolj pogosti, ker veliko oglasnih člankov oglašuje domove. Pri Naivnem Bayesu pa je slika nekoliko drugačna. Zaradi drugačnega načina delovanja je narobe klasificiral drugačne članke z drugačnimi besedami kot preostali algoritmi.

Poglejmo si še najpomembnejše besede v narobe klasificiranih novicah:

Pomembnost	Linearni SVM	SVM z RBF	Naivni Bayes	SGD
1.	school	school	school	school
2.	busi	friday	busi	friday
3.	share	share	investig	share
4.	friday	monday	friday	busi
5.	compani	compani	share	park
6.	offici	busi	monday	monday
7.	govern	best compani	compani	
8.	tuesday	open	open	thursday
9.	local	thursday	park	open
10.	thursday	tuesday	tuesday	local

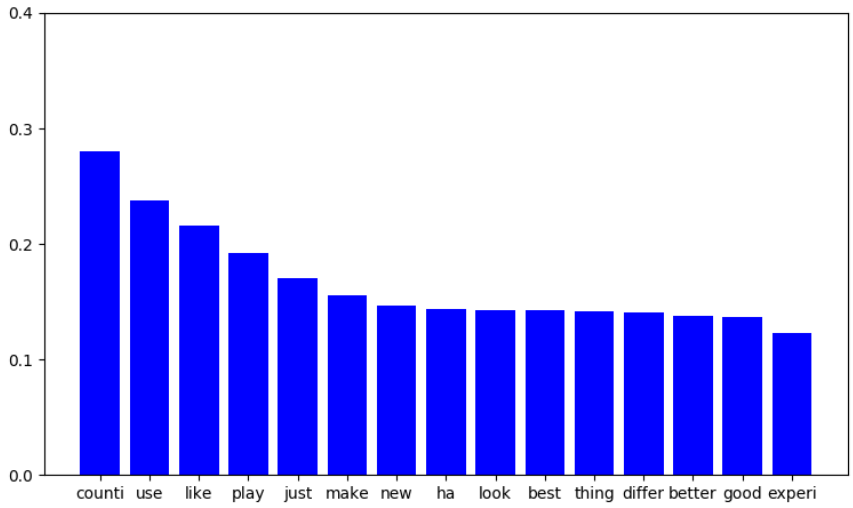
Tabela 4.3: 10 najpomembnejših besed v oglasih, ki so bile napačno klasificirane kot novice.

Tukaj pa ni nobenih očitnih razlik med algoritmi. Vsi imajo zelo podobne najbolj pomembne besede. Očitno je, da beseda school zelo vpliva na napačno klasifikacijo, saj ima precejšnjo težo pri pravih novicah in če je beseda v oglasu, to močno vpliva na klasifikacijo. Ena od slabosti programa so torej oglasi za šolske izdelke. Zanimivo je, da se dostikrat med temi besedami pojavljajo dnevi in podobne besede, kot so tuesday, monday, thursday in podobne. Te besede imajo zgleda večjo težo pri novicah in če jih oglas vsebuje, je večja verjetnost, da bo članek napačno klasificiran. Zgleda, da so ena večjih slabosti mojega programa tudi oglasi za dogodke, saj dostikrat vsebujejo te besede in dostikrat padejo nekje vmes.

4.2.3 Problemi z recenzijami

Med preizkušanjem programa sem opazil sledeči problem. Recenzije zmeraj označi kot oglase, čeprav to niso. Ko sem naredil zbirko sestavljeno samo iz recenzij, jih je 93 odstotkov označilo kot oglase pri metodi linearni SVM. Druge metode so pa kar 97 odstotkov recenzij označile kot oglase.

Moja domneva je, da zaradi tega, ker recenzije nek predmet ocenjujejo, uporabljajo podobne besede kot pri oglasih. Da sem to preveril sem podobno kot pri napačno klasificiranih primerih dobil ven najpomembnejše besede po TF-IDF oceni. Pa poglejmo najpomembnejše besede v recenzijah:



Slika 4.2: 15 najpomembnejših besed v recenzijah gelde na TF-IDF oceno.

Vidimo, da niti nimajo toliko podobnih besed oglasom. Sicer imajo besede kot play, ki imajo večjo težo pri oglasih kot pri novicah, in so po mojem mnenju glavni krivec, da jih klasificira kot oglase.

Ker vidimo, da imajo oglasi toliko besed, ki niso toliko pomembne ne v oglasih in ne v preostalih novicah, je ena mogoča izboljšava ta, da se naredi še en razred posebej imenovan reviews. To sem naredil tako, da sem v učno množico dodal mapo z mapo reviews v katero sem dodal 163 recenzij (predvsem iger in pametnih telefonov). V testno množico pa sem prav tako dodal mapo reviews in vanjo 134 recenzij s strani nepovezanih z recenzijami iz učne množice.

Naivni Bayes	Oglas	Novica	Recenzija	Priklic	Preciznost
Oglas	211	162	2	0,56	0,80
Novica	28	280	0	0,91	0,61
Recenzija	24	20	90	0,67	0,98
Klas. točnost	0,71				
Linearni SVM	Oglas	Novica	Recenzija	Priklic	Preciznost
Oglas	263	108	4	0,70	0,72
Novica	77	231	0	0,75	0,66
Recenzija	23	12	99	0,74	0,96
Klas. točnost	0,72				
SVM z RBF	Oglas	Novica	Recenzija	Priklic	Preciznost
Oglas	281	94	0	0,75	0,77
Novica	58	250	0	0,81	0,70
Recenzija	25	13	96	0,72	1,00
Klas. točnost	0,77				
SGD	Oglas	Novica	Recenzija	Priklic	Preciznost
Oglas	226	133	16	0,60	0,82
Novica	38	269	1	0,87	0,62
Recenzija	13	31	90	0,67	0,84
Klas. točnost	0,72				

Tabela 4.4: Ocenjevanje klasifikacije z dodanim razredom recenzije.

Vidimo, da se klasifikacijska točnost ni kaj dosti poslabšala. Je pa pri tem primeru za moje cilje boljši SVM z RBF jedrom, saj ima višji priklic oz. je narobe prepoznal manj oglasov kot drugi algoritmi. Malo me je skrbelo, ker za učno množico nisem imel časa, da bi priskrbel večji vzorec recenzij, vendar je recenzije še zmeraj prepoznalo zelo dobro, saj je po mojih merilih bolj malo recenzij narobe klasificiralo, čeprav se je prepoznavanje oglasov zaradi tega nekoliko poslabšalo.

Iz tega lahko zaključimo, da so recenzije precej drugačne od oglasov in

drugih novic.

Poglavje 5

Sklepne ugotovitve in zaključek

Ko sem pričel s to diplomsko nalogo, nisem vedel, kaj točno pričakovati in kako uspešen bom z rezultati. Zbiranje podatkov je bil precej dolg proces in je vzel veliko časa. Res sem se potrudil, da sem našel čimbolj raznolike podatke iz čimbolj raznolikih virov. Za novice to ni bil tak problem. Za oglase pa se je izkazalo, da so tipi oglasov na večini strani zelo podobni. Večinoma so oglasi za lokacije, tehniko, potovanja, nepremičnine in potovanja, kar je zgornja analiza tudi pokazala. Zgleda, da so te panoge dosti bolj nagnjene k naročanju časopisnih oglasov.

Glede slovenskih člankov in oglasov pa nisem imel veliko sreče, saj nima dosti slovenskih novičarskih strani oglaševalskih člankov. Na voljo sem tako imel le dva vira, kar se ni izkazalo prav ugodno za ocenjevanje klasifikacije. Analizo sem v glavnem zaradi tega naredil samo na angleških člankih.

Algoritmi, ki sem jih uporabil, so se izkazali kot uspešne, saj je natančnost precej presegla moja pričakovanja. Pričakoval sem, da bo dosti več oglasov narobe klasificiralo in da klasifikacijska točnost ne bo dosti nad večinskim klasifikatorjem v najboljšem primeru. Uspelo mi je narediti program, ki precej natančno ocenjuje ali je članek oglas ali prava novica in da ne oceni preveč oglasov napačno kot novice. Mislim, da mi je prvotni cilj, da naredim program, ki bi med drugim zaznaval tudi prikrito oglaševanje, uspel.

Sem pa pri ovrednotenju in analizi prišel do precej zanimivih zaključkov,

ki jih nisem pričakoval. Predvsem, ko sem opazil, da recenzije označuje kot oglase, sem pričakoval, da je to zaradi tega, ker so zelo podobne oglasom in zaradi tega ne morem narediti izboljšave za program, ki bi uspešno ločevala oglase od recenzij. Presenetljivo so se recenzije izkazale kot povsem drugačne od oglasov in novic. Zaradi tega mi je uspelo dobiti potencialno nadgradnjo programa.

Še ena možnost za nadgradnjo programa, ki je nisem imel časa preizkusiti, je, da bi združil napovedi vseh algoritmov in tako dobil potencialno boljšo klasifikacijo. To bi naredil tako, da bi pogledal, kako je večina algoritmov klasificirala nek članek in mogoče bi nekaterim algoritmom glede na njihovo natančnost dal večjo težo.

Mislim pa, da bi bilo tudi bolje, če bi imel učno množico z več različnimi vrstami oglasov, saj se je kot največja pomankljivost programa izkazala pri oglasih, ki so na primer oglaševali šolske potrebščine. Če bi imel več tipov oglasov bi imel program manj šibkih točk. Zato se je bogata učna množica izkazala kot zelo pomembna.

Bolj kot samo delovanje algoritmov me je zanimala sama sestava in analiza člankov. In res mislim, da sem se v tej diplomski nalogi veliko naučil. Moja največja napaka je bila, da sem napačno ocenil, koliko časa bo vzelo zbiranje podatkov. Iz teh in podobnih napak sem se precej naučil za naprej. Veliko sem se naučil tudi glede sestave samih oglasov in tudi glede same sestave novic. Upam, da bom vnaprej srečal še mnogo podobnih zanimivih izzivov.

Literatura

[1] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11:1471−1490, 2010.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(2):273–297, 1995.

[3] Guido van rossum. Dosegljivo: `https://en.wikipedia.org/wiki/Guido_van_Rossum`. [Dostopano: 27.8.2018].

[4] History of python. Dosegljivo: `https://en.wikipedia.org/wiki/History_of_Python`. [Dostopano: 27.8.2018].

[5] Igor Kononenko and Marko Robnik Šikonja. Osnove strojnega učenja. In Janez Demšar and Matjaž Kukar, editors, *Inteligentni sistemi*, pages 45–89. Založba FE in FRI, 2010.

[6] Igor Kononenko and Marko Robnik Šikonja. Predstavitev znanja in operatorji. In Janez Demšar and Matjaž Kukar, editors, *Inteligentni sistemi*, pages 89–109. Založba FE in FRI, 2010.

[7] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Machine Learning*, 115(33), 2018.

[8] Nltk. Dosegljivo: `https://www.nltk.org/`. [Dostopano: 27.8.2018].

[9] Numpy. Dosegljivo: http://www.numpy.org/. [Dostopano: 27.8.2018].

[10] Pycharm. Dosegljivo: https://www.jetbrains.com/pycharm/. [Dostopano: 27.8.2018].

[11] Pyqt. Dosegljivo: https://riverbankcomputing.com/software/pyqt/intro. [Dostopano: 27.8.2018].

[12] Python. Dosegljivo: https://en.wikipedia.org/wiki/Python_(programming_language). [Dostopano: 27.8.2018].

[13] Qt. Dosegljivo: http://doc.qt.io/qt-5/qtdesigner-manual.html. [Dostopano: 27.8.2018].

[14] Bernhard Schölkopf, Koji Tsuda, and Jean Philippe Vert. Kernel primer. In *Kernel Methods in Computational Biology*. MIT Press, 2004.

[15] Scikit-learn. Dosegljivo: http://scikit-learn.org/stable/. [Dostopano: 27.8.2018].

[16] Scipy. Dosegljivo: https://www.scipy.org/. [Dostopano: 27.8.2018].

[17] Zen of python. Dosegljivo: https://www.python.org/dev/peps/pep-0020/. [Dostopano: 27.8.2018].