Strojno učenje u analizi korisničkih anketa

Vurnek, Marko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

STROJNO UČENJE U ANALIZI KORISNIČKIH ANKETA

Diplomski rad

Marko Vurnek

Osijek, 2020.

Sadržaj

1. UVOD

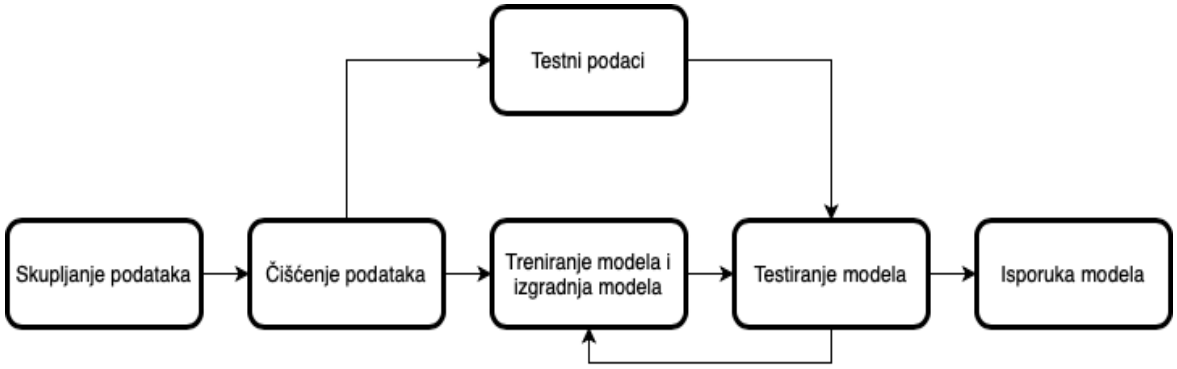
Strojno učenje je jedna od najpopularnijih tema i tehnologija današnjice. Primjenjuje se u svim industrijama za riješavanje problema iz različitih domena. Zbog rapidnog razvoja programskih sustava i povećanja broja korisnika koji generiraju podatke na tim sustavima razvijeni su različiti algoritmi i metodologije strojnog učenja kako bi se omogućilo kreiranje riješenja baziranih na tim podatcima bez eksplicitnog programiranja. Za potrebe ovoga rada dizajniran je sustav u kojemu se korisnici mogu pridružiti različitim grupama odnosno predmetima i komunicirati u stvarnom vremenu unutar podgrupa kako bi se izvršila korisnička anketa za određenu podgrupu. Razmijenjene poruke zajedno sa pripadajućom grupom su proslijeđene na Spark platformu na kojoj se vrši klasifikacija teksta kako bi se odredilo dali je rečenica pozitivna ili negativna. Svrha ovog sustava je da se izvrši analiza na temelju poruka od korisnika kako bi se dobio dodatan uvid u kvaliteti predmeta. Za izradu klasifikatora koji će odrediti semantičku vrijednost poruke koristiti će se nadzirani algoritmi strojnog učenja. Rad je strukturiran na slijedeći način. U prvom dijelu rada se objašnjava strojno učenje kao i različiti pristupi strojnog učenja poput nadziranog, nenadziranog, polu-nadziranog te pojačanog učenja. U drugom dijelu rada se objašnjava podatkovna pipa kao i sve metode i faze obrade teksta te se objašnjavaju klasifikatori koji su implementirani u ovom radu. U trećem dijelu je objašnjena sama arhitektura sustava koja je implementirana u svrhu demonstracije i izrade rada kao i sve tehnologije koje su korištene. Naposljetku u četvrtom dijelu ovoga rada objašnjena je implementacija sustava i dobiveni rezultati.

1.1. Zadatak diplomskog rada

Potrebno je napraviti pregled trenutno aktualnih metoda strojnog učenja za analizu tekstualnih dokumenata. Izgraditi skup podataka dohvaćanjem korisničkih upita iz različitih izvora. Tehnologije koje bi se koristile u pri analizi korisničkih anketa su Spark, Kafka, Cassandra te programski jezik Python.

2. STROJNO UČENJE

Strojno učenje je grana računalnih znanosti koja se oslanja na kolekcije primjera nekog fenomena kako bi bila uspješna u izgradnji algoritama, predstavlja metodu koja automatizira analitičku izgradnju modela za analizu podataka. Primjeri koji se koriste za treniranje modela mogu biti izgrađeni od strane ljudi, generirani od strane nekog drugog algoritma ili mogu doći iz prirode. Proces strojnog učenja se sastoji od algoritama koji iterativno uče od podataka koji služe kao primjeri nekog fenomena ili događaja te omogućuje računalima da bez eksplicitnog programiranja koji su bazirani na pravilima dobiju dodatne sakrivene uvide iz podataka. Razlikujemo nekoliko tipova strojnog učenja kao što su nadzirano (eng. *supervised*), polu nadzirano (eng. *semi-supervised*), nenadzirano (eng. *unsupervised*) i pojačano učenje (eng. *reinforcment learning*) koji će se objasniti u ostatku rada.

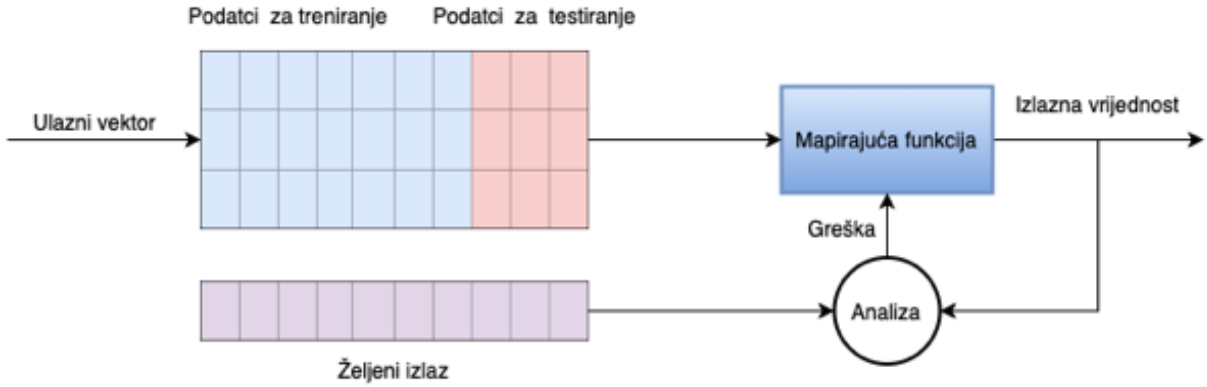


Slika 2.1. *Dijagram strojnog učenja*

Prema slici 2.1. se vidi da prvi korak pri izradi modela strojnog učenja je dohvaćanje podataka. Podatci se mogu dohvatiti iz različitih izvora te ih je potrebno strukturirati i očisti. Prema [1] nakon toga taj skup podataka se dijeli na dva dijela. Prvi dio uključuje podatke za treniranje i u pravilu se uzima 70% podataka iz čitavog skupa podataka. Sa tim podatcima se vrši treniranje i izgradnja samog modela dok sa preostalih 30% podaka koji se koriste kao testni podatci se vrši evaluacija odnosno testiranje samog modela. Treniranje modela se vrši kroz nekoliko iteracija sve dok se ne zadovolje određene postavljene metrike za performansu.

2.1. Nadzirano učenje

Algoritmi izgrađeni metodom nadziranog učenja su trenirani korištenjem prethodno označenim primjerima. Ulazne vrijednosti sadrže podatke i klase koje ih označavaju i željena izlazna vrijednost je poznata. Odnosno algoritam prima skup vrijednosti zajedno sa odgovarajućim izlaznim vrijednostima te algoritam uči uspoređujući stvarne izlazne vrijednosti sa datim izlaznim vrijednostima koje je primio na ulazu i pokušava naći odstupanja ukoliko ima odstupanja model se korigira. Kroz metode kao klasifikacija (eng. *classification*), regresija (eng. *regression*), predikcija (eng. *prediction*) i pojačavanje gradijenta (eng. g*radient boosting*) model nadziranog strojnog učenja koristi uzorke iz podataka koji su označeni kako bi predvidio oznaku odnosno klasu za podatak koji nije označen**.**

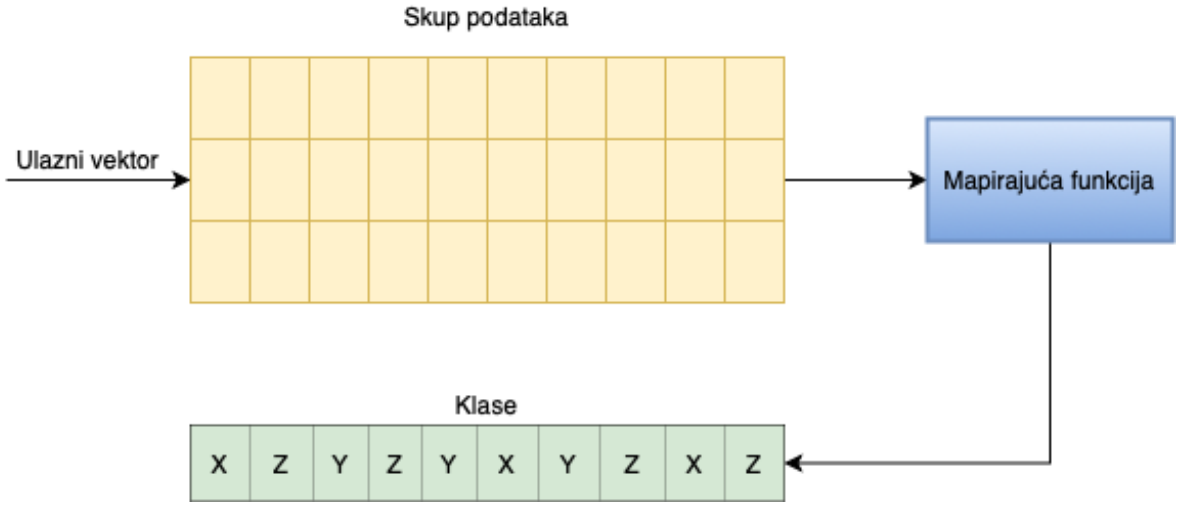


Slika 2.2. *Dijagram nadziranog učenja*

Prema slici 2.2. mapirajuća funkcija sa nadziranim učenjem se izgrađuje i testira u dvije faze. Prva faza uključuje segmentiranje podataka u dva skupa, za testiranje i trening najčešće po principu 70/30 di se skup podataka segmentira na 70% za trening i 30% za testiranje. Podatci sadrže testni vektor koji predstavlja ulazne podatke zajedno sa jednom ili više izlaznih vrijednosti koji pripadaju odgovarajućem podatku. Mapirajuća funkcija se trenira podatcima za trening dok se ne dobije željena perfromansa. Metrika koja prikazuje preciznost odnosno točnost mapirajuće funkcije u mapiranju podataka za trening sa odgovarajućim željenim izlazom se pojavljuje iterativno za svaki uzorak podataka za trening koji ulazi u mapirajuću funkciju. Nakon svake iteracije se vrši analiza prilikom koje se analizira pojavljivanje greške odnosno odstupanja koji prikazuje odstupanje stvarne vrijednosti od željenje vrijednosti te se mapirajuća funkcija korigira. Naposljetku se testnim podatcima evaluira sama mapirajuća funkcija.

2.2. Nenadzirano učenje

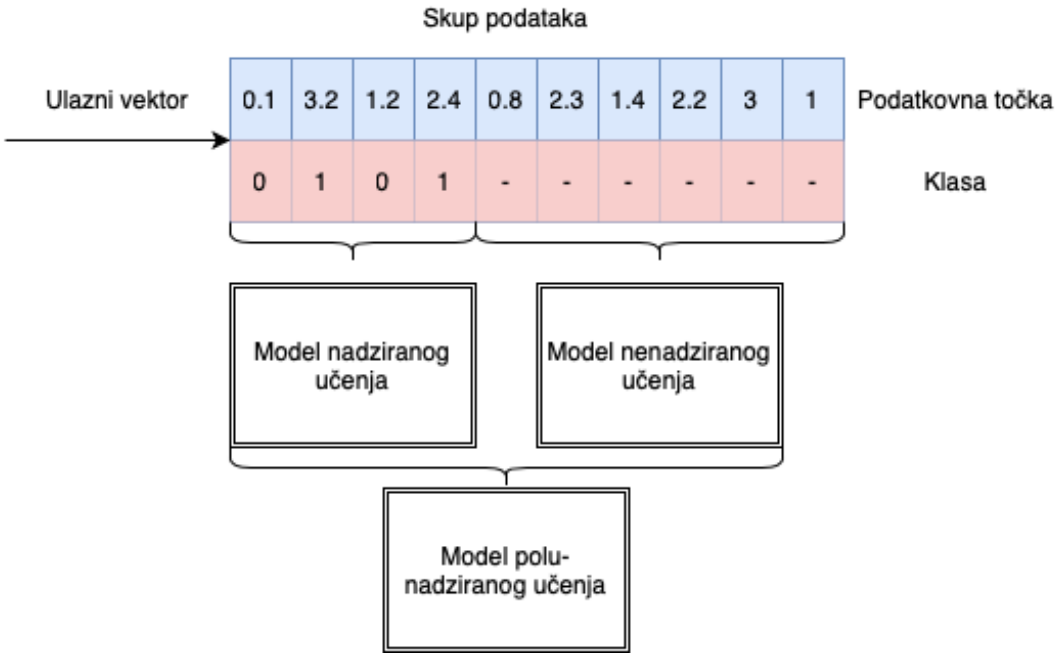
Nenadzirano učenje se koristi kod riješavanje problema gdje podatci nisu označeni. Cilj ovih algoritama je istraživanje uzoraka u podatcima i pronalaženje struktura unutar podataka. Popularne metode ove vrste strojnog učenja su samoorganizirajuće mape (eng. *self-organising maps*), mapiranje najbližeg susjeda (eng. *nearest-neighbor mapping*), *k-means* grupiranje (eng. *k-means custering*) te dekompozicija pojedinačne vrijednosti (eng. *singular value decomposition*). Glavni problem za razliku od nadziranog učenja je taj što ne postoji metrika pomoću koje se mjeri performansa samog modela. Cilj kod ove metode je izgradnja mapirajuće funkcije koja će na temelju skrivenih svojstava unutar podataka kategorizirati podatke u klase. Mapirajuća funkcija segmentira skup podataka u klase pri ćemu svaki ulazni vektor postane član određene klase na temelju svojstva podataka koji je analizom određen unutar mapirajuće funkcije (Slika 2.3.).



Slika 2.3 *Dijagram nenadziranog učenja*

2.3. Polu-nadzirano učenje

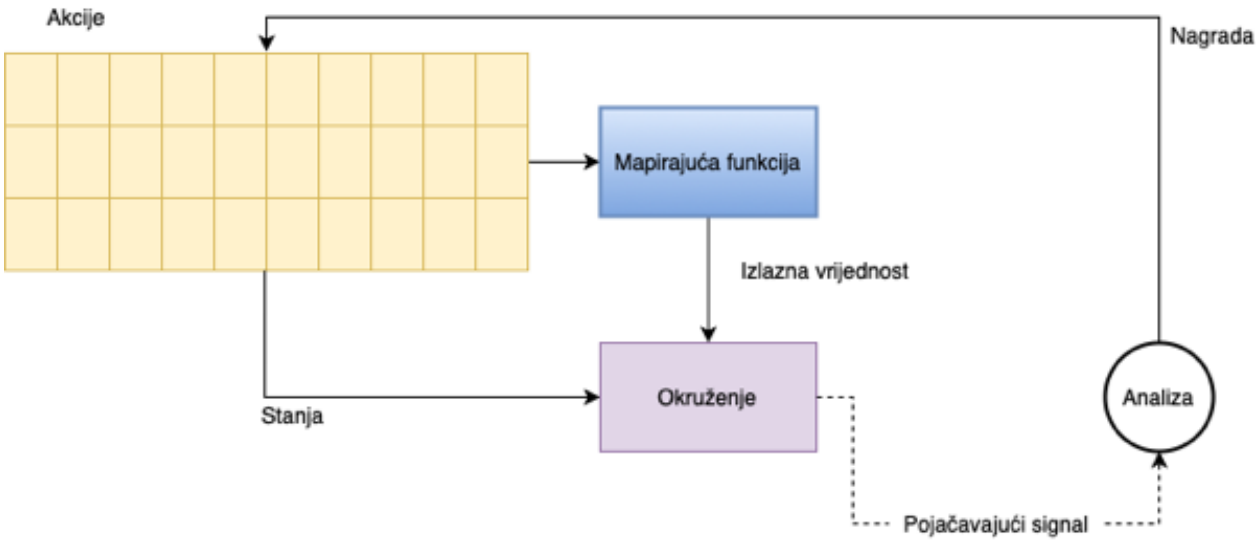
Polu-nadzirano učenje je kombinacija nadziranog i nenadziranog učenja (Sl.2.4.). Za izvršavanje određenog zadataka odnosno riješavanje problema koriste se podatci koji su označeni i ne označeni prilikom čega takav skup podataka čini u velikoj većini podatci koji su ne označeni. Ovim algoritmom se pokušava poboljšati performansa nadziranog i nenadziranog učenja iskorištavanjem odnosno utiliziranjem informacije koja je asocirana sa jednom od prethodno navedenih modela strojnog učenja. Kod problema klasificiranja mogu se utilizirati podatci koji nisu označeni kako bi se poboljšala performansa samog algoritma kao i kod riješavanja problema grupiranja gdje postoji veliki skup ne označenih podataka mogu se koristiti označeni podatci čija svojstva bi pomogla u klasificiranju ne označenih podataka.



Slika 2.4. *Skup podataka za polu-nadzirano učenje*

2.4. Pojačano učenje

Glavni cilj ove vrste strojnog učenja je riješavanje problema kako naučiti inteligentnog agenta da odabire pravilne odnosno dobre sekvence odluka. Inteligentni agent se nalazi u okruženju i sposoban je percipirati stanje toga okruženja u kojemu se nalazi. Agent može izvršavati akcije u svakom stanju pri ćemu različite akcije donose različite nagrade. Cilj je da algoritam nauči politiku sustava u kojemu se nalazi pri ćemu za ulaznu vrijednost koje predstavlja različito stanje da određenu akciju na izlazu, te se kroz iterativno davanje nagrada korigira dok se ne zadovolji postavljena metrika.



Slika 2.5. *Dijagram pojačanog učenja*

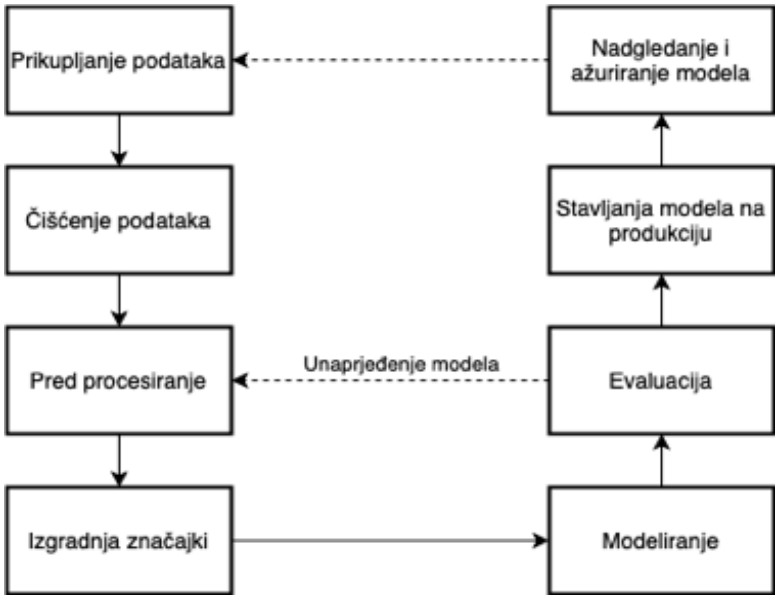
3. PRIRODNO PROCESIRANJE JEZIKA

Pod prirodnim jezicima se smatraju svi jezici koji se koriste ili su se koristili tijekom ljudske evolucije. Odnosno to su oni jezici koji su se koristili ili se koriste u svakodnevnoj komunikaciji kao hrvatski, njemači i engleski za razliku od umjetnih jezika kao što su različite matematičke notacije i programski jezici. Kako bi se omogućilo manipuliranjem nad prirodnim jezicima i dobivanje uvida iz tih jezika nastala je nova grana računalnih znasnosti pod imenom prirodno procesiranje jezika (eng. *natural language processing*). Prirodno procesiranje jezika uključuje bilo koju tehniku računalne manipulacije prirodnoga jezika. Glavni problem kod kreiranja modela strojnog učenja na temelju tekstualnih podataka je to što tekstualni podatci su ne strukturirani te ih je poretbno pretvoriti u numeričku vrijednost za daljnje procesiranje. Postoje različiti primjeri riješenja koji su dobiveni korištenjem prirodnog procesiranja jezika kao što je preporuka filmova na temelju sadržaja knjige, grupiranje pravnih dokumenata, detekriranje pronevjere unutar orgranizacije, detektiranje neželjene pošte, analiza korisnikovih objava na društvenim mrežama u svrhu ciljanog marketinga, nakupljanje artikala .Tehnike za obradu teksta kao i za kreiranje modela strojnog učenja prikazat će se u nastavku ovoga rada u kojemu će se analizirati korisnička anketa u svrhu određivanja semantičke vrijednosti poruke.

3.1. Podatkovni kanal za priodno procesiranje jezika

Prema [2] korak kod izrade podatkovnog kanala za izradu modela za prirodno procesiranje jezika je dohvaćanje tekstualnih podataka. Dohvaćeni podatci su nestrukturirani i neuređeni pa je zbog toga potrebno nakon dohvaćanja podataka očistiti ih. Pod čišćenjem podataka se smatra preimenovanje stupaca koja sadrže svojstva podataka za lakše procesiranje, uklanjanje redaka sa nepotpunim podacima ili strategijsko dopunjavanja istih, uklanjanje praznog prostora ili nepotrebnih simbola odnosno bilo koji proces koji bi formatirao podatke u željenu strukturu. Nakon što su podatci strukturirani i uređeni slijedi proces pred procesiranja koji uključuje tokeniziranje, reprezentacija teksta u numeričke vrijednosti, ukljanjanje zaustaving riječi (eng. stop words) kao što su članovi, vezinici i prijedlozi. Nakon što je izvršeno predprocesiranje slijedi izgradnja značajki pri ćemu se vrši analiza trenutnih značajki svojstva teksta kako bi se ili uklonila postojeća značajka ili dodala nova koja bi nam dala bolji uvid ili svojstvo u same podatke. Također se podatci odvajaju po principu 70/30 gdje se 70% podataka korisiti za treniranje modela dok preostalih 30% za testne podatke. Modeliranje slijedi nakon pred procesiranja prilikom kojega se izabire model strojnog učenja koji će vršiti estmiranje i

transformiranje podataka. Nakon treniranja modela strojnog učenja vrši se evaluacija istoga te ako je preciznost niska koraci pred procesiranja, izgradnji značajki i modeliranja se izvršavaju ponovno i iterativno dok se ne postigne željena preciznost. Ukoliko evaluacija pokazuje zadovoljavajuću preciznost model se stavlja na produkciju te se nadgleda i ažurira ukoliko je potrebno.



Slika 3.1. *Dijagram podatkovnog kanala za prirodno procesiranje jezika*

3.2. Čišćenje podataka

Čišćenje podataka predstavlja najvažniji korak pri kreiranju modela strojnog učenja. Dohvaćeni teksutalni podatci su najčešće nestrukturirani te ih potrebno kroz različite tehnike strukturirati, normalizirati i očistiti od nepotrebnih znakova. Prema [3]Dohvaćeni podatci se prvo strukturiraju u korpus formu koja predstavlja skup podataka koji se sastoji od različitih tekstualnih dokumenata. Standardna procedura prilikom čišćenja podatak uključuje uklanjanje simbola i interpunkcijskih znakova, promjena svih velikih slova u mala, uklanjanje svih nepotrebnih i suvišnih riječi te uklanjanje svih brojeva. Nakon što je prvi korak čišćenja podataka izvršen slijedi drugi korak u kojemu se riječi segmentiraju, tehnika koja omogućuje segmentiranje riječi naziva se tokeniziranje. Tokeniziranjem se riječi, parovi riječi ili rečenice segmentiraju u tablicu koja predstavlja vokabular svih riječi iz dohvaćenog teksta. Nakon što je stvoren vokabular svih riječi iz njega je potrebno maknuti sve zaustavne riječi jer ne predstavljaju nikakvu dodatnu vrijednost prilikom kreiranja modela strojnog učenja. Nad svim riječima se potom može izvesti morfološka normalizacija koja se sastoji od dva postupka, korjenovanje i lematizacija.

Korjenovanje (eng. s*temming*) je proces koji uključuje reduciranje fleksije riječi odnosno uklanjanje sufika i prefiksa riječi kako bi se dobila korijen riječi, predstavlja grubu proceduru prilikom koje se može dobiti riječ koja ne postoji. Lematizacija (eng. *lemmatization*) predstavlja složeniji postupak koji je sličan korijenovanju no za razliku od korijenovanja vraća kanonski oblik riječi. Naposljetku potrebno je svakoj riječi pridodati numeričku vrijednost kako bi ulazni podaci bili podobni za model strojnog učenja. Kada je svakoj riječi pridodana numerička vrijednost odnosno svaka riječ je enkodirana unutar dokumenta onda čitav dokument možemo predstaviti kao numeričku sekvencu, matricu ili tenzor.

3.3. Pred procesiranje

Riječi se enkodiraju korištenjem tehnika umetanja riječi (eng. *word embeddings*). Razlikujemo tri vrste umetanja riječi odnosno enkodiranja riječi u numeričku vrijednost a to su jedno vruće enkodiranje (eng. *one-hot encoding*), enkodiranje bazirano na frekvenciji riječi (eng. *frequency based encoding*) te enkodiranje bazirano na predikcijama (eng. *prediction based encoding*).

3.3.1. Jedno-vruće enkodiranje

Kako je prethodno navedeno u radu, korpus je izgrađen od dokumenata. Dokumenti predstavljaju segmentirane tekstove ili rečenice.

Tablica 3.1. Primjer korpusa riječi sa dokumentima

Poruke
Thank you!
I love this.
I hate this.

Prvi korak je ekstraktiranje svake riječi iz dokumenata koji čine korpus. Prilikom ekstraktiranja riječi potrebno je izostaviti riječi koje se ponavljaju taj pristup se naziva tokeniziranje riječi. Sami skup riječi koji se dobije ekstrakcijom naziva se vokabular koji čine sve riječi iz korpusa.

Tablica 3.2. Primjer skupa rječi koji predstavljaju vokabular

Skup riječi
Thank
you
I
love
this
hate

Nakon izgradnje vokabulara riječi iz korpusa, možemo odrediti dali su riječi iz vokabulara prisutne ili ne unutar rečenice. Označavanje se vrši binarno, ukoliko je riječ prisutna pridodjeljuje se numerička vrijednost 1 ukoliko riječ nije prisutna dodjeljuje se 0.

Tablica 3.3. Primjer pridodjeljivanje numeričke vrijednosti riječima korištenjem tehnike jednog-vrućeg enkodiranja

Vokabular	Thank you!	I love this.	I hate this.
Thank	1	0	0
you	1	0	0
I	0	1	1
love	0	1	0
this	0	1	1
hate	0	0	1

Jedno vruće enkodiranje konvertira svaki dokument unutar korpusa u vektor značajke (eng. *feature vector*) gdje je sama duljina vektora prethodno određena duljinom odnosno količinom riječi unutar vokabulara. Prednost ove tehnike je jednostavnost implementacije. Mana ovog pristupa je veliki set podataka odnosno veličina vokabulara, jer povećanjem vokabulara povećava se i sami vektor značajki. Ukoliko dokument unutar korpusa sadrži samo jednu riječ iz vokabulara, vekor značajki će svejendo morati sadržavati sve enkodirane vrijednosti od toga vokabulara. Također jedna od velikih mana ovog pristupa je to što su podaci neuređeni, gubi se značanje riječi i sama informacija konteksta rečenice u kojoj se riječ nalazi, također nije moguće odrediti semantičku informaciju kao ni relacije između riječi unutar korpusa. Zbog binarnog

sustava koji se koristi u jednom-vrućem enkodiranje gubi se i informacija o frekvenciji odnosno ponavljanju riječi unutar korpusa.

3.3.2 Vektorizacija bazirana na frekvenciji riječi

Postoje tri kategorije vektorizacije riječi bazirano na frekvenciji odnosno ponavljanju riječi u kojima razolikujemo vektor brojanja (eng. *count vector embeddings*), *TF/IDF* (eng. *term frequency/inverse document frequency*) i matrica ponavljanja događaja (eng. *co-occurance matrices*).

Vektor brojanja

Ova tehnika je najednostavnija jer je bazirana na dohvaćanje koliko puta se riječ pojavila unutar dokumenta odnosno temelji se na frekvenciji ponavljanja riječi unutar dokumenta. Uzmimo u obzir korpus koji se sastoji od dva dokumenta:

- d1 = I really like this thesis
- d2 = Thesis is awful and the presenter is awful

Nakon što riječi tokenizirani unutar dokumenta dobijemo listu riječi koje se ne ponavljaju koje predstavljaju vokabular. (Tab.3.4.)

Tablica 3.4. Primjer skupa rječi koji predstavljaju vokabular

Vokabular
I
really
like
this
thesis
is
awful
and
the
presenter

Tablica 3.5. Primjer pridodjeljivanje numeričke vrijednosti riječima korištenjem tehnike vektora brojanja koji je baziran na frekvenciji riječi

Vokabular	D1: I really like this thesis	D2: Thesis is awful and the presenter is awful
I	1	0
really	1	0
like	1	0
this	1	0
thesis	1	1
is	0	2
awful	0	2
and	0	1
the	0	1
presenter	0	1

Prilikom korištenja ove tehnike nakon izgradnje vokabulara iz korpusa kao i u jednom-vrućem enkodiranju sve riječi su označene prilikom čega se vidi dali se riječ nalazi ili ne unutar vokabulara no također je označena frekvencija ukoliko se riječ više puta ponavlja unutar istog dokumenta kao što se može vidjeti u tablici 3.5.. U odnosu na jedno-vruće enkodiranje prednost je informacija ponavljanja riječi no mane ostaju iste kao jako veliki vektori značajki, gubi se kontekst kao i semantičko značenje te relacije između riječi.

TF/IDF

Ova tehnika kao i jedno vruće-enkodiranje omogućuje dokvaćanje frekvenicije ponavljanja riječi unutar dokumento no također daje uvid koliko se riječ puta ponovila unutar čitavog korpusa.

$$x_i = TF(w_i) \times IDF(w_i) \tag{3-1}$$

Specifična tehnika za numeričko predstavljanje riječi koja se koristi u algoritmima za pretraživanje. x_i predstavlja numeričku reprezentaciju za datu riječi, TF mjeri koliko neka riječ se ponavlja u dokumentu dok IDF mjeri koliko se rijeko riječ ponavlja u čitavom korpusu. *TF/IDF* daje nam ocijenu važnosti riječi unutar čitavog korpusa. Prema [3] ukoliko se riječ ponovi unutar istog dokumenta *TF* joj povećava važnost no IDF smanjuje ukoliko se riječ često ponavlja unutar čitavog korpusa. Ovaj pritup ima velike prednosti jer uz frekvenciju ponavljanja riječi unutar dokumenta dohvaća i važnost same riječi unutar čitavog korpusa no mana je što kontekst u kojoj se određena riječ pojavi izgubi.

Matrica ponavljanja događaja

Matrice ponavljanja događaja bazirane su na ideji odnosno principu da slične riječi će se pojaviti zajedno i da će imati sličan kontekst. Ovom tehnikom pokušava se dohvatiti kontekts u kojemo se riječ pojavila. Generiranje tih matrica zahtjeva korštenje kontekstnog prozora (eng. *context widow*) odnosno okvira. Predstavlja prozor koji je centriran oko riječi, koji uključuje određeni broj susjednih riječi oko centrirane riječi. Za primjer će se koristiti samo jedan dokument u korpusu zbog jednostavnosti i kontekstni prozor će biti 2 što znači da će uzimat dvije susjedne riječi oko centrirane riječi. Za primjer će se korisiti slijedeća rečenica:

- d1 = He is great. He is always ready to support us.

Tablica 3.7. Primjer matrice ponavljanja događaja kada je kontekstni prozor postavljen na prvu riječ *He*

	He	is	great	always	ready	to	support	us
He		1	1					
is	2		1					
great	2	2						
always								
ready								
to								
support								
us								

Ako postavimo kontekst prozora na riječ *He* susjedne riječi će biti *is* i g*reat* te one će poprimiti vrijednos 1 u prvom retku. Nakon što se kontekstni okvir pomakne na riječ *is* susjedne riječi će biti *He*, *great* i ponovit će se riječ *he* te u drugom retku će riječ *He* poprimiti vrijednost 2 jer se riječ unutar prozornog konketsa dva puta ponovila. Pomakom na riječ *great* susjedne riječi će biti zbog kontekstnog prozora *He*, *is* te će se te dvije riječi ponoviti te zbog toga te riječi poprimaju vrijednost 2. (Tab.3.7.)

Pomakom kontekstnog prozora na iduću riječ *He* koja se nalazi u drugoj rečenici susjedne riječi su *is, great*, *always* gdje se riječ *is* dva puta ponavlja te zbog toga će se vrijednost unutar prvog retka povećati. (Tab.3.8.)

Tablica 3.8. Primjer matrice ponavljanja događaja kada je kontekstni prozor postavljen na treću riječ *great*

	He	is	great	always	ready	to	support	us
He		3	2	1				
is	2		1					
great	2	2						
always								
ready								
to								
support								
us								

Finalne vrijednosti se mogu vidjeti u tablici nakon što se ovaj postupak ponovi kroz nekoliko iteracija i prozorni kontekst se postavi na zadnju riječ unutar dokumenta. (Tab.3.9.)

Tablica 3.9. Primjer matrice ponavljanja događaja kada je kontekstni prozor postavljen na zadnju riječ *us*

	He	is	great	always	ready	to	support	us
He		3	2	1				
is	3		2	1	1			
great	2	2						
always	1	1			1	1		
ready		1		1		1	1	
to				1	1		1	1
support					1	1		1
us						1	1	

3.3.3. Enkodiranje bazirano na predikcijama

Enkodiranje bazirano na predikcijama je numerička reprezentacija teksta koja dohvaća značenje i semantičke odnose riječi. Generiranjem numeričke reprezentacije predikcijskim baziranim umetanjem odnosno ugradnjom riječi zahtjeva tehnike strojnog učenja. Modelom strojnog učenja mogu se izvršiti dvije stvari ovom tehnikom, za datim riječima iz konteksta pokušati predvidjeti riječ, te datom riječi pokušati predvidjeti riječi u njenom kontekstu. Nakon treniranja modela strojnog učenja velikim korpusom svakoj riječi na ulazu u model biti će pridodana numerička vrijednost na izlazu, numeričke vrijednosti će biti slične ukoliko riječi imaju sličnu kontekstnu vrijednost. Modeli koji se koriste su sustavi bez nadgledanja je pokušavaju dobiti značajku riječi bez odgovarajuće klase za tu riječ, također je potrebno ovakve modele trenirati sa velikom korpusom tekstualnih podataka. Ovom tehnikom se pokušava enkodirati svaka riječi kao vektor drugih riječi. Ova tehnika ima veliku prednost u odnosu na ostale tehnike jer pomoću nje se dohvaća i semantička informacija riječi kao i relacije između riječi unutar teksta. Postoji predtrenirani modeli kao što su GloVe i Word2Vec koji sadrže generirane ugradnje riječ.

3.4. Algoritmi strojnog učenja

Za riješavanje problema analize korisničkih anketa koristit će se algoritam nadziranog učenja koji riješava probleme u domeni klasifikacije. Klasifikacija dokumenata je klasičan problem nadziranog strojnog učenja. Ako postoji skup kategoriziranih odnosno označenih dokumenata, zadatak je automatski kategorizirati novi dokument u jednu od postojećih kategorija. U daljnjem tekstu objasnit će se algoritmi koji se koriste za klasifikaciju teksta u ovom radu.

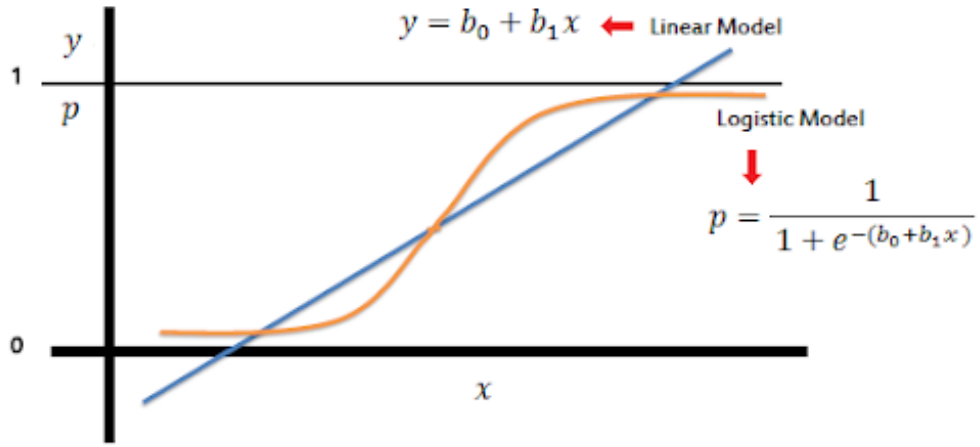
3.4.1. Naive Bayes

Naive Bayes klasifikator je baziran na Bayesovom teoremu te je pogodan za korištenje u slučajevima kada je dimenzionalnost ulaznih veličina velika. Iako je vrlo jednostavan, može postići dobre performanse koje se mogu mjeriti sa ostalim sofisticiranim klasifikatorima. Koristi se u raznolikim klasifikacijskim zadatcima zbog jednostavnosti implementacije te zbog brze predikcije i efikasnosti kada se primjeni na jako velikim skupovima podataka. Bayesov teorem je formula koja opisuke kako dodati najnovije informacije odnosno kako ažurirati već postojeće vjerojatnost hipoteze kad su joj dati novi dokazi. Za datu hipotezu X i dokaz Y, Bayesov teorem govori da odnos između vjerojatnosti hipozete prije dobivanja dokaza P(X) i vjerojatnost hipoteze nakon dobivanja dokaza P(X/Y) može se izraziti formulom (3-2). Ime naivni je ovaj algoritam dobio zato što pretpostavlja da značajke koje idu u novi model su nezavisne jedna od druge. Odnosno da mjenjanjem vrijednosti jedne značajke ne utječe direktno na druge značajke algoritma.

$$P(X|Y)=\frac{P(Y|X)\cdot P(X)}{P(Y)} \tag{3-2}$$

3.4.2. Logstička regresija

Logisitčka regresija (eng. *Logistic regression*) je jedan od osnovnih modela strojnog učenja ra riješavanja problema u domeni klasifikacije. Koristi se kod binarne klasifikacije za predikciju distribuiranih kategorija.



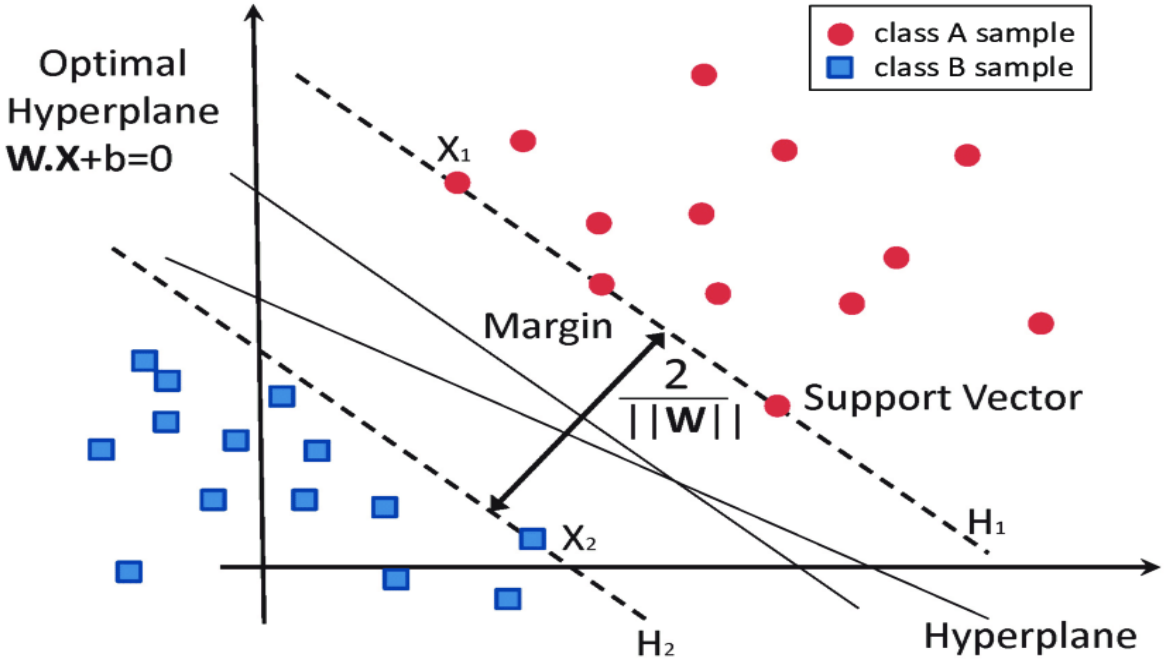
Slika 3.2. *Logistička regresija*

Prema [4] kao što je prikazano na slici 3.2. prema [5] sigmoidna odnosno logistička funkcija prima bilo koju vrijednost našeg linearnog modela na ulazu te na izlazu daje vrijednost između 0 ili 1 (3-3). Pri ćemu se može postaviti srednja točka 0.5, te ukoliko vrijednost na izlazu je manja od 0.5 vrijednost se zaokružuje na 0 no ukoliko je vrijednost 0.5 ili veća, vrijednost se zaokružuje na 1.

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{3-3}$$

3.4.2. Strojevi sa potpornim vektorima

Strojevi sa potpornim vektorima (eng. *Support vector machines*) su algoritmi strojnog učenja koji se koriste najčešće u domeni klasifikacije ali i regresije. Prema [4] predstavljaju klasifikator velikih margina, smatra se metodom strojnog učenja koja je temeljena na vektorskom prostoru gdje je cilj pronaći granicu odluke između dvije klase.

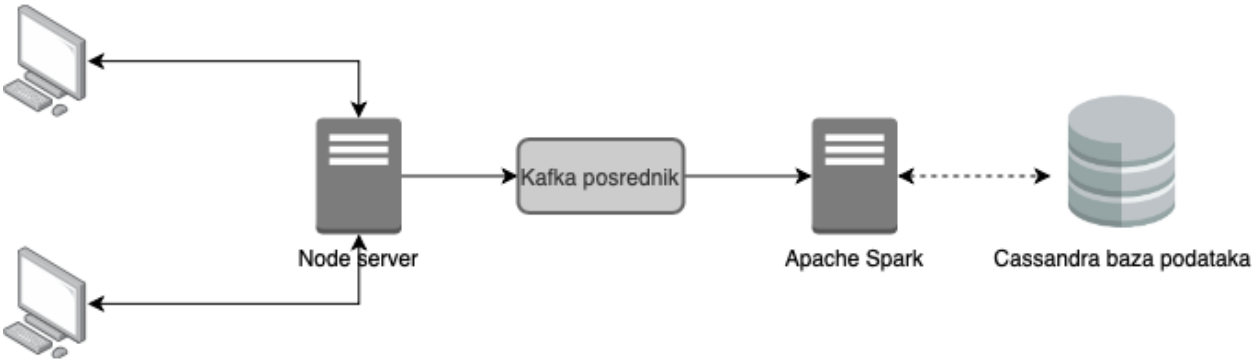


Slika 3.2. *Strojevi sa potpornim vektorima*

Koristeći algoritam strojeva sa potpornim vektorima mapiraju se podatkovne točke u n-dimenzionalnom prostoru gdje svaka podatkovna točka predstavlja koordinatu. Prema slici 3.2. prema [6] klasifikacija se vrši pronalaženjem hiperravnine koja odvaja dvije klase koje kategoriziraju podatkovnu točku u određenu klasu.

4. ARHITEKTURA SUSTAVA

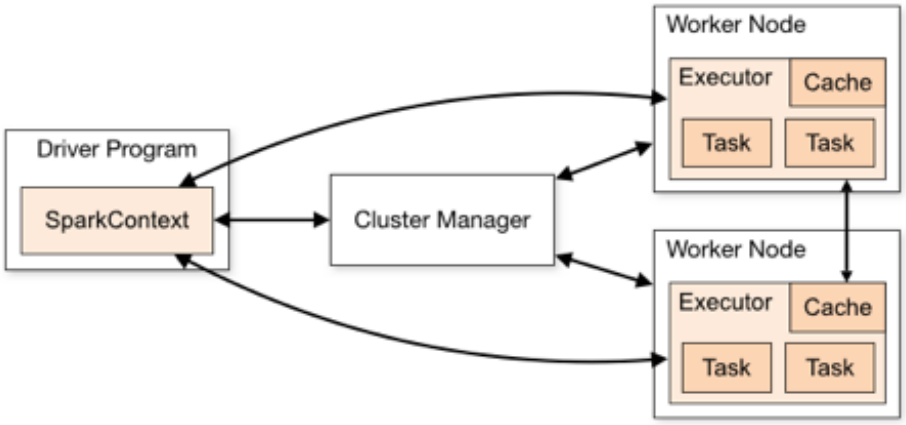
Krajnji uređaji komuniciraju u stvarnom vremenu preko *web socketa,* komunikacija se odvija bidirekcionalno odnosno dvosmjerno. Poruke se razmjenjuju preko Node servera prilikom čega Kafka proizviđač uzima poruku i prosljeđuje ju na određenu temu na Kafka posredniku koja je prethodno definirana. Kafka potrošač koji se nalazi na Spark serveru dohvaća poruku sa teme na koju se prethodno pretlatio te slijede svi koraci podatkovne pipe koji su prethodno navedeni u radu kako bi se poruka strukturirala, normalizirala i obradila. Naposljetku rezultati obrade se spremaju u Cassandru bazu podataka. (Sl.4.1.)



Slika 4.1. *Arhitektura sustava*

4.1. Apache Spark

Apache Spark je programski okvir (eng. *framework*) za procesiranje podataka te velikom brzinom izvršava zadatke procesiranja nad velikim skupovima podataka. Također omogućuje raspodijelu odnosno distribuciju zadataka za procesiranje na veliki broj računala. Nastao je 2009. godine na U.C. Brekley-u te vrlo brzo postao jedan od ključnih programskih okvira za distibuirano procesiranje velikih skupova podataka. Podržani programski jezici su Java, Scala, Python i R jezik za statističku analizu. Platforma također uključuje podršku za tok podataka (eng. *streaming data*), *SQL*, strojno učenje (eng. *machine learning*) te obradu grafova (eng. *graph processing*). Apache Spark aplikacija se sastoji od svije glavne komponente, upravljačkog programa (eng. *driver*) i izvršitelja (eng. *executor*). Upravljački program konvertira kod u višestruke zadatke koji su distribuirani preko različitih radnih čvorova (eng. *worker nodes*). Izvršitelji se nalaze na tim radnim čvorovima i izvršavaju distribuirane zadatke. (Sl.4.2.)



Slika 4.2. *Apache Spark arhitektura*

Apache Spark ima hijerahijsku arhitekturu koja se sastoji od nadređenog (eng. *master*) i podređenog (eng. *slave*) sustava. Spark upravljački program predstavlja nadređeni čvor koji upravlja klaster upravljačem (eng. *cluster manager*). Upravljački klaster upravlja podređenim čvorovima i dostavlja rezultate aplikacijskom klijentu. Spark upravljački program generira Spark kontekst koji preko klaster upravitelja raspodijeljuje zadatke i nadgleda izvršavanje tih zadataka na podređenim čvorovima.

Otporni distribuirani skup podataka

RDD (eng. *Resilient Distributed Dataset*) odnosno otporni distribuirani skup podataka predstavlja kolekciju elemenata koji mogu biti raspodijeljeni na više čvorova u klasteru na kojima se može raditi paralelno. Ima četiri glavna svojstva kao što je tolerancija na kvarove, predstavlja distribuiranu kolekciju podataka, omogućuje paralelne operacije te korištenje višestrukih izvora podataka. Spark dohvaća podatke paralelizacijom postojeće kolekcije ili referenciranjem podatkovnog izvorišta te sprema u *RDD* za daljnje procesiranje. Nakon što su podatci spremljeni u *RDD*, Spark može izvršavati različite transformacije i akcije nad *RDD* memorijom. Svaki podatkovni skupu unutar *RDD*-a je podijeljen u logične particije čije se procesiranje može izvoditi na različitim čvorovima unutar klastera. Korisniku je omogućeno izvršavanje dva tipa operacija, transformacije i akcije. Transformacije su operacije koje se primjenjuju u izradi novog *RDD*-a, dok akcije služe kao instrukcije Apache Sparku za primjenu procesiranja i vraćanja rezultata upravljačkom programu. Za raspored zadataka kao i za menadžment radnih čvorova unutar klastera Spark kreira direktni aciklični graf (eng. *Direct Acyclic Grap*) koji poboljšava efikacnost sustava.

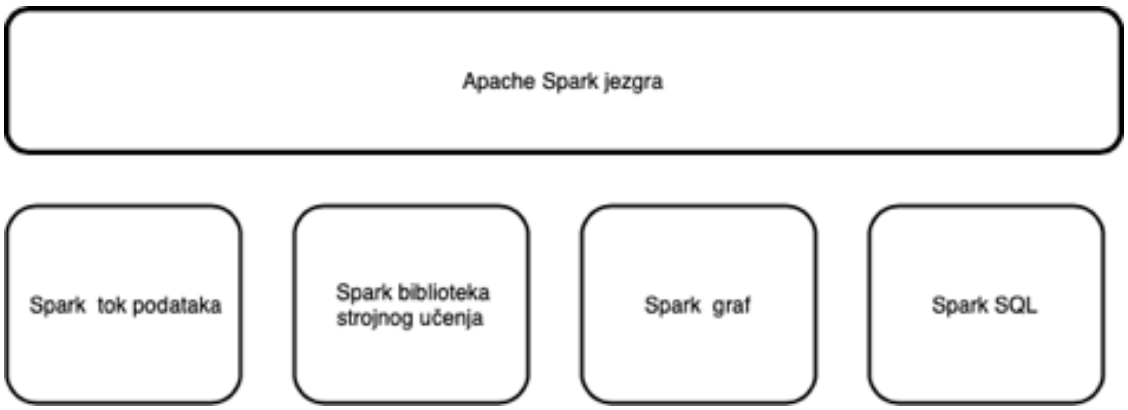
Spark okvir podataka

Spark okvir podataka (eng. *dataframe*) je proizašao kao unaprijeđenje otpornog distribuiranog skupa podataka te omogućuje čišći i jednostavniji rad pri ćemu izgleda uniformno preko više *API*-ja (eng. *application programmable interface*). Okvir podataka drži podatke u formatu stupac i redak. Svaki stupac predstavlja neko svojstvo podataka ili varijablu dok svaki redak predstavlja individualnu točku podataka. Različiti podatkovni formati se mogu koristiti kao ulaz i izlaz okvira podataka.

Spark komponente

Apache Spark sastoji se od četiri pet glavnih komponenti:

- Apache Spark jezgra je opći mehanizam Spark platforme na kojoj su izgrađene ostale funkcionalnosti. Omogućuje procesiranje u memoriji te referenciranje podatkovnih skupava koji se nalaze na vanjskim sustavima za pohranu.
- Spark *SQL* komponenta je izgrađena na Spark jezgri te pruža podatkovnu abstrakciju zvanu *RDD*, koja daje podršku za obradu strukturiranih i polu strukturiranih podataka.
- Spark tok podataka komponenta koja je također izgrađena na Spark jezgri iskorištava njene sposobnosti za brzo raspoređivanje kako bi se izvršila analitika nad tokom podataka. Grupira ulazne podatke u manje blokove podataka nad kojima se izvršavaju *RDD* transformacije.
- *MLlib* je raspodijeljeni programski okvir za strojno učenje.
- *GraphX* je raspodijeljeni programski okvir za procesiranje grafova.



Slika 4.3. *Apache Spark komponente*

4.2. Apache Kafka

Apache Kafka je raspodijeljena platforma koja omogućuje protočnost događaja odnosno predstavlja centralizirani sustav koji omogućuje sustavima pretplaćivanje na određene događaje kao i objavljivanje određenih događaja. Jedna od najvećih prednosti je to što ovaj sustav omogućuje vertikalno i horizontalno skaliranje sa robustnom pohranom podataka. Pruža programerima abstrakciju kako bi se mogli usredotočiti na riješavanju poslovnog problema umjesto na samu platformu za razmjenu poruka. Vrlo je jednostavno za skalirati jer sustavi koji čitaju ili rade promjene nad podatcima se mogu dinamično dodavati na server dok ne dođe do prevelike saturacije na serveru no i taj problem se vrlo lako riješava dodavanjem novog servera u sustav. Također pruža abstrakciju vremena jer sustavima je omogućeno da čitaju podatke bilo kojom preddefiniranom brzinom jer Kafka radi kao međuspremnik (eng. *buffer*) između generiranja podataka i konzumacije istih tih podataka. Omogućuje rukovanjem podatkovnim blokovima (eng. *batch data*) kao i podatcima koji stižu u stvarnom vremenu odnosno tokovima podataka (eng. *data streaming*)

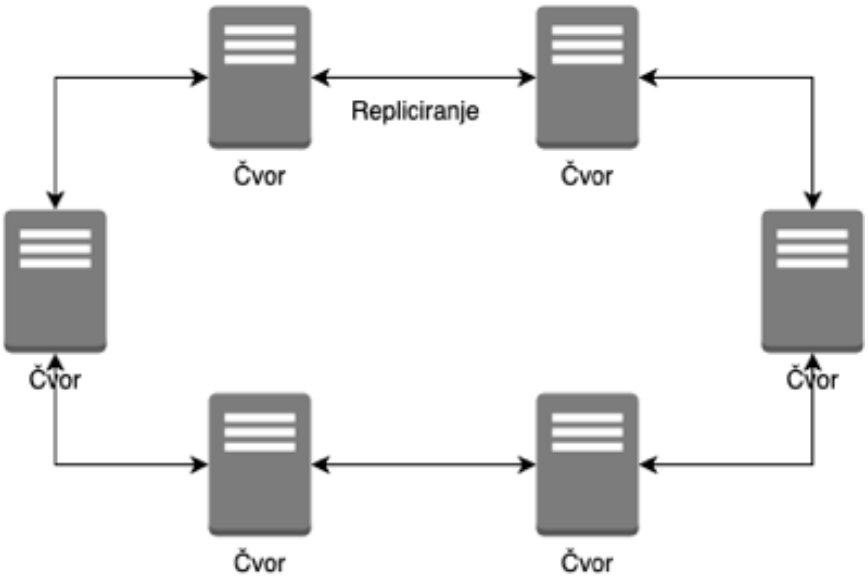
Apache Kafka arhitektura

Proizvođač šalje poruke na Kafka broker koji se sastoji od jednog servera ili klastera servera. Poslani podatci su poruke koje stižu u obliku nizova bajtova (eng. *byte arrays*) bez specifičnog formata koji mogu biti upareni sa jedinstvenim ključem. Također je omogućeno prikačiti podatkovnu shemu u formatu JSON, XML, AVRON ili na shemu koju progameri sami definiraju od koju se podatkovna struktura može verificirati. Svakoj poruci je dodijeljena određena tema ili događaj koja se nalazi na Kafka međuspremniku. Potrošaći konzumiraju poruke iz Kafka brokera na temelju teme ili događaja na koji su se prethodno pretplatili.(Sl.4.4.)

Slika 4.4. *Kafka arhitektura*

4.3. Cassandra

Cassandra je menadžment sustav za distribuirane baze podataka dizajniran za rukovanjem velike količine strukturiranih podataka koji su raspodijeljeni preko većeg broja servera. Zbog svoje raspodijeljene arhitekture Cassandra lako upravlja velikom količinom podataka. Podatci su raspodijeljeni i replicirani na više servera te samim time je postignuta transparentnost kvarova i transparentost repliciranja. Glavno svojstvo Cassandre je da spremi podatke na višestrukom broju čvorova bez ijedne točke neuspjeha (eng. *no single point of failure*). Razlog ovoga dizajna je da ukoliko dođe do sklopovske pogreške na jednom od čvorova podatci ostaju sačuvani. Svi čvorovi razmjenjuju informacije ogovaračkim protokolom (eng. *gossip protocol*). Ogovarački protokol je vrsta kopirajućeg protokola koji se koristi za širenje informacija preko decentraliziranih mreža bez centraliziranog koordinatora što čini ovaj protokol skalabilnim i otpornim na kvarove. Cassandra je *NoSQL* baza podataka odnosno ne relacijska baza podataka. Glavne značajke *NoSQL* baze podataka je jednostavnost dizajna, horizontalno skaliranje i visoka dostupnost.



Slika 4.5. *Cassandra arhitektura*

4.3. Node.js

Node.js je platforma koja je izgrađena na Google Chrome Javascript modulu (eng. *engine*) V8. Razvijo ju je Ryand Dahl 2009. godine. Node.js je platforma otvorenog koda koja omogućuje razvijanje skalabilnih mrežnih aplikacija. Pogonjena je događajima, ima blokirajući ulazno/izlazni model koji ga čini lakim i efikasnim. Najveću primjenu ima kod sustava stvarnog vremena koji moraju upravljati velikom količinom podataka preko raspodijeljenih uređaja. Izabran je u ovom radu zbog svih svojstava koji su prethodno navedeni kako bi se kreirala korisnička anketa u kojoj se poruke mogu razmjenjivati u stvarnom vremenu. Kako bi se omogućila ta vrsta komunikacije također se integrirala Socket.io biblioteka. Socket.io je Javascrip biblioteka za mrežne aplikacije koje zahtjevaju dvosmjernu komunikaciju u stvarnom vremenu te je bazirana na događajima. Sastoji se od dva glavna dijela, prvi dio se koristi na klijentskoj dok drugi dio se koristi na serverskoj strani kako bi se uspostavila komunikacija u stvarnom vremenu.

5. IMPLEMENTACIJA KLASIFIKATORA I REZULTATI

5.1. Dohvaćanje podataka

Prema [8] podatci su dohvaćeni sa istraživačkog rada u kojima se vršila semnatička analiza korisničkih objava sa Twitter platforme. Podatci sadrže različite podatke kao što je klasa koja označava dali je rečenica pozitivna ili negativna, datum objave, korisničko ime.

```
In [3]: data = spark.read.csv('data/Sentiment140.csv')

In [4]: data.show()
```

```
+---+----------+--------------------+--------+---------------+--------------------+
|_c0|       _c1|                 _c2|     _c3|            _c4|                 _c5|
+---+----------+--------------------+--------+---------------+--------------------+
|  0|1467810369|Mon Apr 06 22:19:...|NO_QUERY|_TheSpecialOne_|@switchfoot http:...|
|  0|1467810672|Mon Apr 06 22:19:...|NO_QUERY|  scotthamilton|is upset that he ...|
|  0|1467810917|Mon Apr 06 22:19:...|NO_QUERY|       mattycus|@Kenichan I dived...|
|  0|1467811184|Mon Apr 06 22:19:...|NO_QUERY|        ElleCTF|my whole body fee...|
|  0|1467811193|Mon Apr 06 22:19:...|NO_QUERY|         Karoli|@nationwideclass ...|
|  0|1467811372|Mon Apr 06 22:20:...|NO_QUERY|       joy_wolf|@Kwesidei not the...|
|  0|1467811592|Mon Apr 06 22:20:...|NO_QUERY|        mybirch|         Need a hug |
|  0|1467811594|Mon Apr 06 22:20:...|NO_QUERY|           coZZ|@LOLTrish hey  lo...|
|  0|1467811795|Mon Apr 06 22:20:...|NO_QUERY|2Hood4Hollywood|@Tatiana_K nope t...|
|  0|1467812025|Mon Apr 06 22:20:...|NO_QUERY|        mimismo|@twittera que me ...|
|  0|1467812416|Mon Apr 06 22:20:...|NO_QUERY| erinx3leannexo|spring break in p...|
|  0|1467812579|Mon Apr 06 22:20:...|NO_QUERY|   pardonlauren|I just re-pierced...|
|  0|1467812723|Mon Apr 06 22:20:...|NO_QUERY|           TLeC|@caregiving I cou...|
|  0|1467812771|Mon Apr 06 22:20:...|NO_QUERY|robrobbierobert|@octolinz16 It it...|
|  0|1467812784|Mon Apr 06 22:20:...|NO_QUERY|    bayofwolves|@smarrison i woul...|
|  0|1467812799|Mon Apr 06 22:20:...|NO_QUERY|     HairByJess|@iamjazzyfizzle I...|
|  0|1467812964|Mon Apr 06 22:20:...|NO_QUERY| lovesongwriter|Hollis' death sce...|
|  0|1467813137|Mon Apr 06 22:20:...|NO_QUERY|       armotley|about to file taxes |
|  0|1467813579|Mon Apr 06 22:20:...|NO_QUERY|     starkissed|@LettyA ahh ive a...|
|  0|1467813782|Mon Apr 06 22:20:...|NO_QUERY|      gi_gi_bee|@FakerPattyPattz ...|
+---+----------+--------------------+--------+---------------+--------------------+
only showing top 20 rows
```

Slika 5.1. *Dohvaćeni skup podataka*

Dodatnom analizom može se vidjeti da sadrži sveukupno 1600000 podataka svrstanih u dvije klase. (Sl.5.2.)

```
In [10]: from pyspark.sql.functions import countDistinct

In [11]: rename_columns.agg(countDistinct('class')).show()
```

```
+----------------------+
|count(DISTINCT class)|
+----------------------+
|                     2|
+----------------------+
```

```
In [12]: rename_columns.groupBy('class').count().show()
```

```
+-----+------+
|class| count|
+-----+------+
|    0|800000|
|    4|800000|
+-----+------+
```

Slika 5.2. *Broj podataka i klasa*

5.1. Čišćenje podataka

Podatci su nestrukturirani te ih je potrebno normalizirati i očistiti. Prema slici 5.2. korištena je standardna pocedura čišćenja teksta u kojoj se uklanjaju poveznice, simboli i interpunkcijski znakovi, prazni prostor te svi ostali suvišni tekstualni podatci koji nisu potrebni.

```
In [13]: from pyspark.sql.functions import regexp_replace
         import string, re

         #Uklanjanje poveznica
         remove_links = rename_columns.withColumn('text', regexp_replace('text', '(?:https?|ftp):\/\/[\n\S]+' , ''))

         #Uklanjanje # znaka
         remove_hashtag = remove_links.withColumn('text', regexp_replace('text', '#', ''))

         #Uklanjanje interpunkcijskih znakova
         remove_interpunctions = remove_hashtag.withColumn('text', regexp_replace('text', '[%s]' % re.escape(string.punctuati

         #Uklanjanje brojeva
         remove_numbers = remove_interpunctions.withColumn('text', regexp_replace('text', '[\d-]', ''))

         #Uklanjanje praznog prostora
         remove_character = remove_numbers.withColumn('text', regexp_replace('text', '(\s\s\s*)', ''))

         #Uklanjanje korisničkih imena
         remove_mentions = remove_character.withColumn('text', regexp_replace('text', '\@\w\w+\s?', ''))

         change_class_negative = remove_mentions.withColumn('class', regexp_replace('class', '0', 'negative'))
         change_class_positive = change_class_negative.withColumn('class', regexp_replace('class', '4', 'positive'))
```

Slika 5.3. *Proces čišćenja podataka*

Za čišćenje podataka upotrebljeni su regularni izrazi te pripadajućim klasama su promjenjene vrijednosti kako bi olakšao daljnji rad.

```
change_class_positive.show(truncate=False)
```

```
+--------+--------------------------------------------------------------------------------------------------------------------+
|class   |text                                                                                                                |
+--------+--------------------------------------------------------------------------------------------------------------------+
|negative|switchfootAwww thats a bummerYou shoulda got David Carr of Third Day to do it D                                     |
|negative|is upset that he cant update his Facebook by texting it and might cry as a resultSchool today also Blah             |
|negative|Kenichan I dived many times for the ball Managed to saveThe rest go out of bounds                                   |
|negative|my whole body feels itchy and like its on fire                                                                      |
|negative|nationwideclass no its not behaving at all im mad why am i here because I cant see you all over there               |
|negative|Kwesidei not the whole crew                                                                                         |
|negative|Need a hug                                                                                                          |
|negative|LOLTrish heylong time no see Yes Rains a bit only a bitLOLIm fine thankshows you                                    |
|negative|TatianaK nope they didnt have it                                                                                    |
|negative|twittera que me muera                                                                                               |
|negative|spring break in plain city its snowing                                                                              |
```

Slika 5.4. *Očišćeni i strukturiani podatci*

5.2. Predprocesiranje podataka

Prvi korak predprocesiranja podataka je segmentiranja riječi unutar teksta u tokene.

```
from pyspark.ml.feature import (Tokenizer, StopWordsRemover, IDF, CountVectorizer, StringIndexer)
```

```
tokenizer = Tokenizer(inputCol='text', outputCol='token_text')
tokenized = tokenizer.transform(change_class_positive)
tokenized.select('token_text').show(truncate=False)
```

```
+----------------------------------------------------------------------------------------------------------------------------------+
----------+
|token_text
|
+----------------------------------------------------------------------------------------------------------------------------------+
----------+
|[switchfootawww, thats, a, bummeryou, shoulda, got, david, carr, of, third, day, to, do, it, d]
|
|[is, upset, that, he, cant, update, his, facebook, by, texting, it, and, might, cry, as, a, resultschool, today, a
lso, blah] |
|[kenichan, i, dived, many, times, for, the, ball, managed, to, savethe, rest, go, out, of, bounds]
|
|[my, whole, body, feels, itchy, and, like, its, on, fire]
|
|[nationwideclass, no, its, not, behaving, at, all, im, mad, why, am, i, here, because, i, cant, see, you, all, ove
r, there]  |
|[kwesidei, not, the, whole, crew]
|
|[need, a, hug]
|
```

Slika 5.5. *Tokenizacija*

Prema slici 5.5 vidi se da su riječi rastavljene u listu no također je vidljivo da liste sadrže zaustavne riječi koje ne daju nikakvo dodatno značenje prilikom klasificiranja teksta pa ih je potrebno ukloniti. Na slici 5.6. mogu se viditi sve zaustavne suvišne riječi.

```
stop_remover = StopWordsRemover(inputCol='token_text', outputCol='stop_token', locale='en_US')
stop_remover.getStopWords()
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 'her',
```

Slika 5.6. *Zaustavne riječi*

Slijedeći korak je implementacija vektora brojanja koji je prethodno objašnjen u ovom radu koji je baziran na frekvenciji ponavljanja riječi. (Sl.5.7.)

```
count_vect = CountVectorizer(inputCol='stop_token', outputCol='c_vect')

idf = IDF(inputCol='c_vect', outputCol='tf_idf')

to_numeric = StringIndexer(inputCol='class', outputCol='label')

from pyspark.ml.feature import VectorAssembler

data_cleanup = VectorAssembler(inputCols=['tf_idf'], outputCol='features')
```

Slika 5.7. *Pretvorba riječi u numeričke vrijednosti*

Nakon vektorizacije metodom vektora brojanja implementiran je *IDF* (*eng inverse document frequency*) kako bi dobili *TF/IDF* koji predstavlja vektorizaciju baziranu na frekveniciju riječi no za razliku od samoga vektora brojanja daje uvid koliko se riječ rijetko ponavlja u čitavom korpusu kako bi joj se povećala značajnost. Također sve riječi unutar klasa su indeksirane kako bi se pripremile za kreiranje modela strojnog učenja te naposljetku se kreira dodatan stupac u *dataframeu* koji prikazuje sve informacije iz ostalih stupaca. Nakon toga se 70% podataka uzima za podatke za treniranje modela dok preopstalih 30% za testiranje. Sve tehnike predprocesiranja su pripremljene kako bi se podatci mogli prosljediti kroz podatkovni kanal koji će sekvencijalno izvršavati sve u fazama kako je prethodno navedeno. (Sl.5.8.)

Slika 5.8. *Podatkovna pipa i podijela podataka za treniranje i testiranje*

```
training_data, test_data = change_class_positive.randomSplit([0.7, 0.3], seed=12345)
```

```
training_data.groupBy('class').count().show()
```

```
+--------+------+
|   class| count|
+--------+------+
|positive|560171|
|negative|559932|
+--------+------+
```

```
from pyspark.ml import Pipeline
data_pipeline = Pipeline(stages=[to_numeric, tokenizer, stop_remover, count_vect, idf, data_cleanup])

pipelineFit = data_pipeline.fit(training_data)
train_data_df = pipelineFit.transform(training_data)
test_data_df = pipelineFit.transform(test_data)
```

5.3. Implementacija klasifikatora i evaluacija

Za izradu rada koristila su se tri različita klasifikatora koji su prethodno obrađeni u ovome radu. Prvi klasifikator koji je treniran podatcima za trening koji su obrađeni kroz podatkovnu pipu je Naive Bayes klasifikator pri ćemu se za evaluaciju koristio višeklasni klasifikacijski evaluator (eng. multiclass classification evaluator) . U svrhe testiranja koristili su se testni podatci koji su također obrađeni kroz podatkovnu pipu tehnikama koje su objašnjene prethodno u ovome radu. Iz slike 5.8. vidi se da precisnost ovoga klasifikatora iznosi 74%.

```
from pyspark.ml.classification import NaiveBayes
naive_bayes = NaiveBayes()
nb_model = naive_bayes.fit(train_data_df)
prediction_NB = nb_model.transform(test_data_df)
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
accuracy_evaluation_NB = MulticlassClassificationEvaluator(metricName="accuracy")
```

```
accuracy_NB = accuracy_evaluation_NB.evaluate(prediction_NB)
print(accuracy_NB)
```

```
0.7409152380615007
```

Slika 5.9. *Treniranje Naive Bayes modela i evaluacija*

Drugi klasifikator koji se koristio u ovom radu je logistička regresija te za evaluaciju precisnosti se koristio binarni klasifikacijski evaluator (eng. *binary classification evaluator*). Kao što se može vidjeti na slici 5.10. rezultat modela logističke regresije je lošiji u odnosu na model Naive Bayesa te mu je preciznost evaluirana na 50%.

```
from pyspark.ml.classification import LogisticRegression
logistic_regression = LogisticRegression().setRegParam(0.01).setThreshold(0.5)
logistic_regression_model = logistic_regression.fit(train_data_df)
prediction_LR = logistic_regression_model.transform(test_data_df)
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
accuracy_evaluation = BinaryClassificationEvaluator().setMetricName("areaUnderROC")
```

```
accuracy_LR = accuracy_evaluation.evaluate(prediction_LR)
print(accuracy_LR)
```

```
0.5086730369934793
```

Slika 5.10. *Treniranje modela logističke regresije i evaluacija*

Zadnji klasifikator koji se implementirao u ovom radu je stroj sa potpornim vektorima te prema slici 5.11. se vidi da je imao najbolji rezultat kako je bilo i očekivano. Za evaluaciju se koristio binarni klasifikacijski evaluator.

```
from pyspark.ml.classification import LinearSVC
linear_svc = LinearSVC().setRegParam(0.01).setThreshold(0.5)
linear_svc_model = linear_svc.fit(train_data_df)
prediction_SVC = linear_svc_model.transform(test_data_df)
```

```
accuracy_SVC = accuracy_evaluation.evaluate(prediction_SVC)
print(accuracy_SVC)
```

```
0.8313044645620018
```

Slika 5.10. *Treniranje modela stroja sa potpornim vektorima i evaluacija*

```
prediction_NB
```

```
DataFrame[class: string, text: string, label: double, token_text: array<string>, stop_token: array<string>, c_vect:
vector, tf_idf: vector, features: vector, rawPrediction: vector, probability: vector, prediction: double]
```

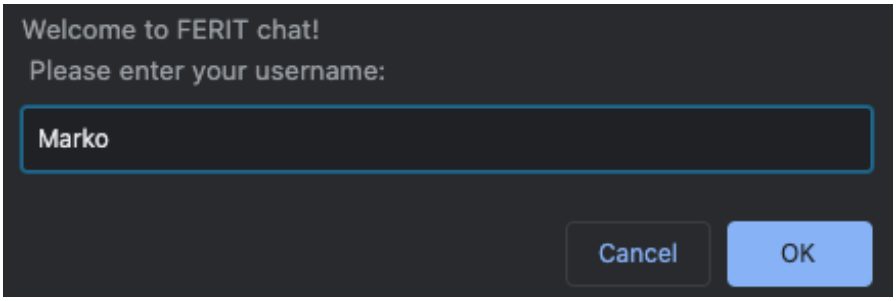
```
prediction_NB.select(['class','text','features','prediction']).show()
```

```
+--------+--------------------+--------------------+----------+
|   class|                text|            features|prediction|
+--------+--------------------+--------------------+----------+
|negative| A doctor who per...|(262144,[67,106,5...|       1.0|
|negative| A just got the s...|(262144,[9,67,189...|       1.0|
|negative|  A sunny day wasted|(262144,[3,67,403...|       1.0|
|negative| AAAARRRRRGGGGGHH...|(262144,[10,67,73...|       1.0|
|negative| AH THIS IS IT FO...|(262144,[67,391],...|       0.0|
|negative| ALEXANDRAbwi my ...|(262144,[67,77,18...|       1.0|
|negative| ALL GOOD THINGS ...|(262144,[1,55,67,...|       1.0|
|negative| AM I got locked ...|(262144,[9,67,506...|       1.0|
|negative| AM Trying to go ...|(262144,[5,13,50,...|       1.0|
|negative| AM its raining o...|(262144,[5,23,67,...|       1.0|
|negative| AMsoon to be mas...|(262144,[67,346,1...|       1.0|
|negative| AND I woke up w ...|(262144,[32,44,67...|       1.0|
|negative| Abit claim it su...|(262144,[67,103,1...|       1.0|
|negative| About to head ba...|(262144,[13,67,21...|       1.0|
|negative| Ack Something at...|(262144,[67,103,5...|       0.0|
|negative| Airport Express ...|(262144,[67,113,2...|       1.0|
|negative| AlcioneG I love ...|(262144,[8,35,67,...|       0.0|
|negative| Alistairs friend...|(262144,[0,21,67,...|       1.0|
|negative| All i can say is...|(262144,[67,84,30...|       1.0|
|negative| All those NonRob...|(262144,[19,48,67...|       1.0|
+--------+--------------------+--------------------+----------+
only showing top 20 rows
```

Slika 5.11. *Prikaz svih stupaca i selektiranih*

6. IMPLEMENTACIJA SUSTAVA

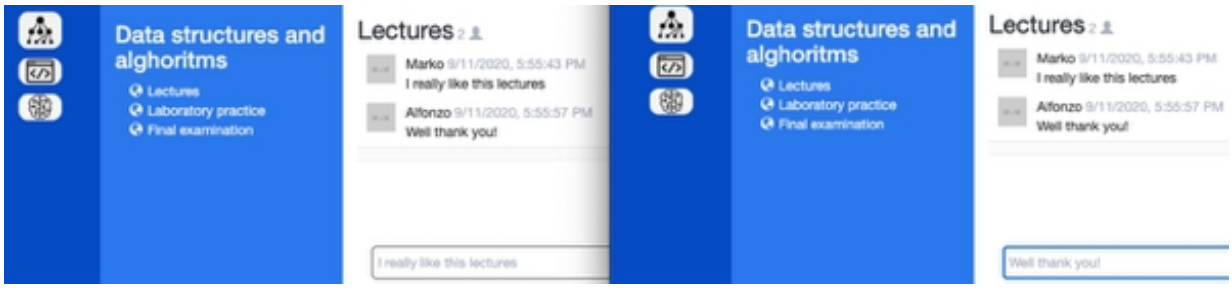
Za dohvaćanje podataka implementiran je sustav koji omogućuje dvosmjernu komunikaciju u stvarnom vremenu koristeći prethodno navedene tehnologije. Pri dolasku na stranicu zahtjeva korisnika da unese korisničko ime. (Sl.5.1.)



Slika 5.1. *Unos korisničkog imena*

Nakon što korisnik unese željeno korisničko ime, korisnik na stranici vidi tri različite grupe koje predstavljaju predmete. Svaka grupa ima istu podgrupu u kojoj korisnici mogu komunicirati u stvarnom vremenu . (Sl.5.2.) (Sl.5.3.)

Slika 5.2. *Korisničko sučelje*



Slika 5.3. *Primjer komunikacije*

Prilikom slanja poruke, poruka prvo pristuže na Node server koji ju proslijeđuje Kafka proizvođaču. (Sl.5.4.)

```
nsSocket.on('newMessage', (msg)=>{
    const fullMsg = {
        text: msg.text,
        time: Date.now(),
        username: username,
        avatar: 'https://via.placeholder.com/30'
    }
    console.log(namespace.nsTitle)
    const roomTitle = Object.keys(nsSocket.rooms)[1];
    console.log(roomTitle)
    const nsRoom = namespace.rooms.find((room)=>{
        return room.roomTitle == roomTitle;
    });
    const client = new kafka.KafkaClient({kafkaHost: 'localhost:9092'});
    const Producer = kafka.Producer,
    producer = new Producer(client),
    payloads = [
        { topic: 'textData', messages:msg.text+'#$%$'+roomTitle+'#$%$'+namespace.nsTitle, partition: 0 },

    ];
    producer.on('ready', function () {
     producer.send(payloads, function (err, data) {

     });
```

Slika 5.4. *Kafka proizvođač*

Prema slici 5.4. također se vidi da se uz samu poruku prosljeđuje odgovarajuća grupa i podgrupa sa određenim znakovima kako bi se čitava poruka mogla segmentirati na odgovarajuće dijelove. Kafka proizviđač šalje čitavu poruku na određenu temu kako bi Kafka potrošač koji se nalazi na Spark serveru mogao konzumirati poruku sa te iste teme na koju se prethodno pretplatio. (Sl.5.5)

```
from cassandra.cluster import Cluster
cluster = Cluster(['127.0.01'])
session = cluster.connect(keyspace='chat')
import pandas as pd
import uuid
from kafka import KafkaConsumer

consumer = KafkaConsumer('textData', bootstrap_servers=['localhost:9092'])
```

```
for message in consumer:
    data = {'text':[message.value.decode("utf-8")]}
    df = pd.DataFrame(data)
    input_value = spark.createDataFrame(df)
    test_data_NEW = pipelineFit.transform(input_value)
    prediction_new = nb_model.transform(test_data_NEW)
    prediction_new.select(['text', 'probability', 'prediction'])
    get_value = prediction_new.select(['prediction']).collect()[0][0]
    result = ''
    if get_value == 1.0:
        result = 'negative'
    else:
        result = 'positive'
    final = f'{result}: {message.value.decode("utf-8")}'
    node_message = message.value.decode("utf-8").split('#$%$')
    session.execute("INSERT INTO survey (id, subject, group, message, polarity) VALUES (uuid(), %s, %s, %s, %s)"
```

Slika 5.5. *Kafka potrošač i spremanje podataka u bazu*

Prema slici 5.5. vrši se transformacija pristiglih podataka koji su prethodno dekodirani i strukutirani u *dataframe.* Naposljetku se vrši određiavnja polariteta pristigle poruke te se svi podatci spremaju u Cassandra bazu podataka.

id	group	message	polarity	subject
b515d81e-7247-4db5-b145-8a3c9bfee9f4	Lectures	this is amazing	positive	DSA
92e66e0e-d7ed-4a5c-9109-f465c28cb7fc	Laboratory practice	I need help	negative	DSA
493065d6-cc32-43c6-9577-acd8bf27dd97	Final examination	I really hate this	negative	DSA
ddc928db-58e8-4502-a7c8-b3ce88304a17	Lectures	I love this	positive	ML
d7c05ebe-1b2c-4c02-9255-eb2ca0c6c7f6	Lectures	This is boring	positive	DSA

Slika 5.6. *Cassandra baza podataka*

Spremljeni podatci mogu poslužiti za daljnje procesiranje, analiziranje ili za različite vrste vizualizacije.

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
value = result_positive.select('message').collect()

value_array = [row.message for row in value]

text = " ".join(value_array)
```

```
wordcloud = WordCloud().generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Slika 5.7. *Vizualizacija pozitivnih riječi*

```
value_negative = result_negative.select('message').collect()
value_array = [row.message for row in value_negative]
text = " ".join(value_array)
wordcloud = WordCloud().generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Slika 5.8. *Vizualizacija negativnih riječi*

7. ZAKLJUČAK

Kroz izradu rada opisane su različite metode prirodnog procesiranja teksta te stečena su različita znanja iz područja strojnog učenja i raspodijeljenih sustava. Cilj ovoga rada je da se prikažu sve aktualne metode strojnog učenja za obradu teksta. Za svrhu demonstracije implementiran je sustav koji dohvaća poruke od korisnika u stvarnom vremenu kako bi se izvršila korisnička anketa. Dohvaćeni podatci su proslijeđeni kroz Kafka posrednik na Spark platformu na kojoj se tekst obrađuje i transformira u oblik pogodan za računalno procesiranje te naposljetku su implementirana tri različita klasifikatora Naive Bayes, logistička regresija te stroj s potpornim vektorima. Rezultati su zadovoljavajući te klasifikatori uspješno klasificiraju korisničke poruke pri ćemu se određuje polaritet samih poruka odnosno poruke se uspješno kategoriziraju u pozitivne ili negativne poruke. Dobiveni podatci su analizirani i spremljeni u Cassandra bazu podataka kako bi se daljnje mogli procesirati i vizualizirati. Korištene metode pri obradi teksta se mogu unaprijediti te također se mogu implementirati napredniji klasifikatori. Najvažnijom fazom pri prirodnom procesiranju teksta se pokazala faza čišćenja i predprocesiranja teksta ali i izbor skupa podataka koji služi za treniranje i testiranje samog klasifikatora.

LITERATURA

[1] A.C., Muller, S.Guido, *Introduction to Machine Learning with Python*, O'Reilly Media, USA, 2017.

[2] D., Medghurst, *Natural Language Processing Pipeline: NLP Pipeline*, Morioh, USA, 2020, dostupno na: https://morioh.com/p/a7b8982e5a5a [10.08.2020]

[3] S., Bird, E., Klein, E., Loper, *Natural Language Processing with Python*, O'Reilly Media, Inc., Sebastopol USA, 2009.

[4] G., James, D., Witten, T., Hastie, R., Tibshirani, *An introduction to Statistical Learning with Applications in R*, Springer, USA, 2013.

[5] J.G., Gomila, *A Simple but Efective Logistic Regression Derivation*, dostupno na: http://juangabrielgomila.com/en/logistic-regression-derivation/

[6] E. Garcia-Gonzalo, Z., Fernandez-Muniz, P.J.G., Nieto, A., A.B., Sanchez, M., Menendez, *Hard-Rock Stability Analysis for Span Design in Entry-Type Excavations with Learning Classifiers,* 2016, dostupno na: https://www.researchgate.net/figure/Classification-of-data-by-support-vector-machine-SVM_fig8_304611323

[7] H., Lane, C., Howard, H.M., Hapke, *Natural Language Processing in action*, Manning Publications Co, USA, 2019.

[8] A., Go, R., Bhayani, L., Huang, *Twitter sentiment classification using distant supervision*, CS224N Project Report, Stanford, 2009., dostupno na:http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip [15.07.2020.]

[9]*Node.js v12.18.4 Documentation,* dostupno na: https://nodejs.org/docs/latest-v12.x/api/ [17.08.2020]

[10] *Express documentation,* Express, dostupno na: https://expressjs.com [17.08.2020]

[11] *Socket.io documentation*, Socket.io, dostupno na: https://socket.io/docs/ [17.08.2020]

[12] *Apache Spark documentation*, Apache Spark, dostupno na https://spark.apache.org/docs/2.4.6/ [02.09.2020]

[13] *Apache Cassandra Documentation v4.0-beta3*, Apache Cassandra, dostupno na: https://cassandra.apache.org/doc/latest/ [05.09.2020]

[14] *Kafka 2.6 Documentation, Apache Kafka*, dostupno na: https://kafka.apache.org/documentation/ [07.09.2020]

[15] *Kafka-node*, Github, dostupno na: https://www.npmjs.com/package/kafka-node [07.09.2020]

SAŽETAK

Naslov: Strojno učenje u analizi korisničkih anketa

Sažetak: U ovom radu dan je pregled trenutno aktualnih metoda strojnog učenja za analizu tekstualnih dokumenata. Prikazane su sve faze prirodnog procesiranja jezika kao i različiti algoritmi za klasifikaciju kao što je logistička regresija, Naive Bayes te strojevi s potpornim vektorima. U svrhu demonstracije implementiran je sustav u kojemu korisnici razmjenjuju poruke u stvarnom vremenu pri ćemu se poruke klasificiraju kao pozitivne ili negativne. Pri implementaciji sustava koristio se Apache Spark za izvršavanja svih faza podatkovne pipe za obradu teksta, Node.js i Socket.io za razmjenu poruka u stvarnom vremenu, Apache Kafka za prosljeđivanje poruka sa Node servera na Spark platformu te Apache Cassandra za spremanje rezultata. Također je izvršena evaluacija preciznosti prethodno navedenih algoritama za klasifikaciju pri kojoj su se koristili testni podatci. Poruke se uspješno klasificiraju te prikazana je jednostavna vizualizacija klasificiranih poruka.

Ključne riječi: strojno učenje, prirodno procesiranje teksta, klasifikacija

ABSTRACT

Title: Machine learning in the analysis of user surveys

Abstract: This paper provides an overview of current machine learning methods for textual document analysis. All phases of natural language processing are presented as well as various classification algorithms such as logistic regression, Naive Bayes and support vector machines. For the purpose of demonstration, a system has been implemented in which users exchanges messages in real time, whereby messages are classified as positive or negative. The system was implemented using Apache Spark to execute all phases of natural languge processing pipeline, Node.js and Socket.io for real-time messaging, Apache Kafka for forwarding messages from the Node server to the Spark platform and Apache Cassandra for saving results. An evaluation of the accuracy of the aforementioned classification algorithms using test data was also performed. Messages are successfully classified and a simple visualization of classified messages is shown.

Keywords: machine learning, natural language processing, classification

ŽIVOTOPIS

Marko Vurnek rođen je 06.veljače 1992. godine u Osijeku, Hrvatska. Osnovnu školu Jagode Truhelke završio je 2006. godine u Osijeku. Iste godine upisuje III. Gimanziju u Osijeku te 2009 upisuje Poljoprivrednu i veterinarsku školu u Osijeku. Srednju školu završava 2011. godine i iste godine upisuje stručni studij Elektrotehničkog Fakulteta u Osijeku, smjer Informatika. Trenutno je na zadnjoj godini diplomskog studija, smjer Programsko inženjerstvo. Prethodno je bio zaposlen u državnoj upravi kao programski inženjer te nakon toga se zapošljava kao vanjski suradnik u razvojnoj tvrtci Glooko sa središtem u Silicijskoj dolini. Nakon toga se zapošljava u istraživačko-razvojnom odjelu tvrtke Enea. Trenutno je zaposlen kao programski arhitekt u tvrtci Farshore Partners.

PRILOZI

Prilozi se nalaze na CD-u:

- Diplomski rad u .pdf fromatu
- Diplomski rad u .docx formatu
- Izvorni kod koji se nalazi na repozitoriju : https://github.com/MarkoVurnek-hub