UNIVERZITET U BEOGRADU

ELEKTROTEHNIČKI FAKULTET

SOFTVERSKO INŽENJERSTVO

VELIKIH BAZA PODATAKA

Seminarski rad

Profesor:

prof. dr Miroslav Bojović

Student:

Komnen Knežević 2022/3093

Beograd, Februar 2023.

SADRŽAJ

1. UVOD

U ovom seminarskom radu je rešavan problem *Natural Language Processing with Disaster Tweets* [1]. U drugom poglavlju je detaljnije predstavljen problem, dok je u trećem poglavlju odrađena analiza ulaznih podataka – *tweet*-ova. U četvrtom poglavlju su predstavljeni načini pomoću kojih je vršena transformacija teksta, kako bi ulazni tekst prilagodili kao ulazni podatak za modele mašinskog učenja

U petom poglavlju su predstavljeni modeli mašinskog učenja pomoću kojih se rešavao navedeni problem, kao i analiza rezultata za svaki model, dok su u šestom poglavlju predstavljeni modeli neuralnih mreža i izvršena analiza rezultata za svaki model.

2. ANALIZA PROBLEMA

Natural Language Processing with Disaster Tweets predstavlja problem gde je potrebno kreirati *machine learning* model koji će određivati da li posmatrani *Tweet* daje informacije o stvarnim katastrofama, ili ne [1].

Postoje dva fajla koja su dostupna, a to su: *train.csv* i *test.csv*. Oba navedena fajla sadrže sledeće informacije:

- *id* – jedinstveni identifikator *tweet*-a
- *text* – tekst *tweet*-a
- *keyword* – ključna reč u *tweet*-u (može biti prazno)
- *location* – lokacija sa koje je poslat *tweet* (može biti prazno)
- *target* – postoji samo u *train.csv* fajlu, predstavlja da li je *tweet* o stvarnoj katastrofi (ima vrednost 1), ili nije (ima vrednost 0).

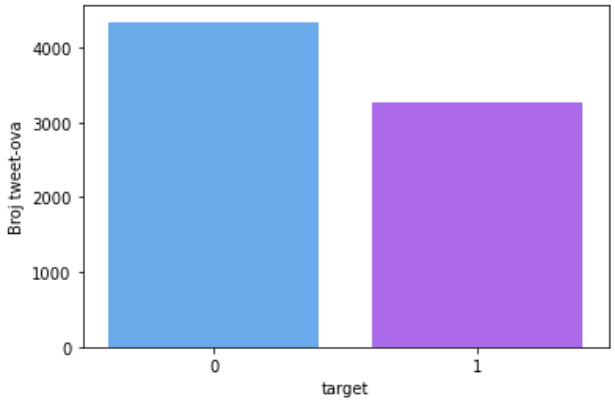
Ukoliko želimo da uradimo *submit* ovog zadatka, onda treba da priložimo *csv* fajl koji ima sledeće informacije:

- *id* – jedinstveni identifikator *tweet*-a
- *target* – predstavlja da li je tweet o stvarnoj katastrofi (ima vrednost 1), ili nije (ima vrednost 0)

3. ANALIZA PODATAKA

3.1. Target

Na slici 3.1.1 možemo da vidimo kako izgleda distribucija *tweet*-ova na osnovu *target* kolone. Možemo da primetimo da postoji više tweet-ova koji ne govore o stvarnim katastrofama



Slika 3.1.1 Distribucija *tweet*-ova na osnovu *target* kolone

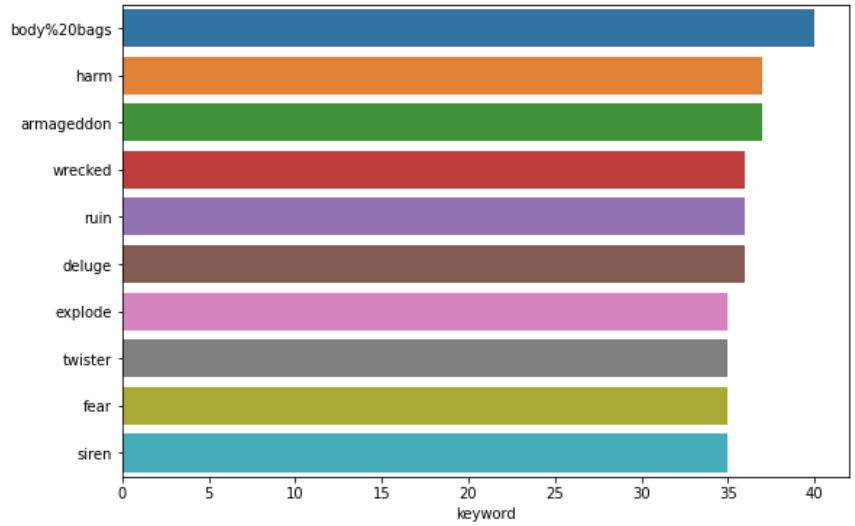
3.2. Keyword

Pošto kolona *keyword* može biti *null,* uradićemo analizu *tweet*-ova koji imaju i koji nemaju popunjenu kolonu *keyword*. U tabeli 3.2.1 su izloženi rezultati nad *train.csv* fajlom.

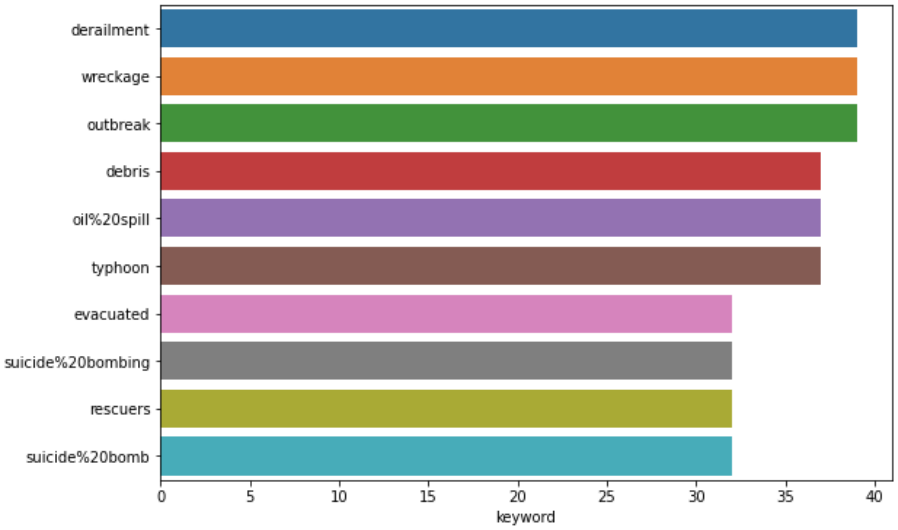
Tabela 3.2.1 Analiza tweet-ova na osnovu *keyword* kolone

Procenat tweet-ova u kojima nedostaje vrednost za keyword, posmatrajući sve tweet-ove	0.8 %
Procenat tweet-ova u kojima nedostaje vrednost za keyword, posmatrajući tweet-ove gde je target=0	0.44%
Procenat tweet-ova u kojima nedostaje vrednost za keyword, posmatrajući tweet-ove gde je target=1	1.28%

Na slici 3.2.1 možemo da vidimo 10 ključnih reči koje se najviše koriste u *tweet*-ovima koji imaju vrednost *target*=1, dok na slici 3.2.2 možemo da vidimo 10 ključnih reči koje se najviše koriste u *tweet*-ovima koji imaju vrednost *target*=0.



Slika 3.2.1 Distribucija tweet-ova(target = 0) pomoću *keyword* kolone



Slika 3.2.2 Distribucija tweet-ova (target = 1) pomoću *keyword* kolone

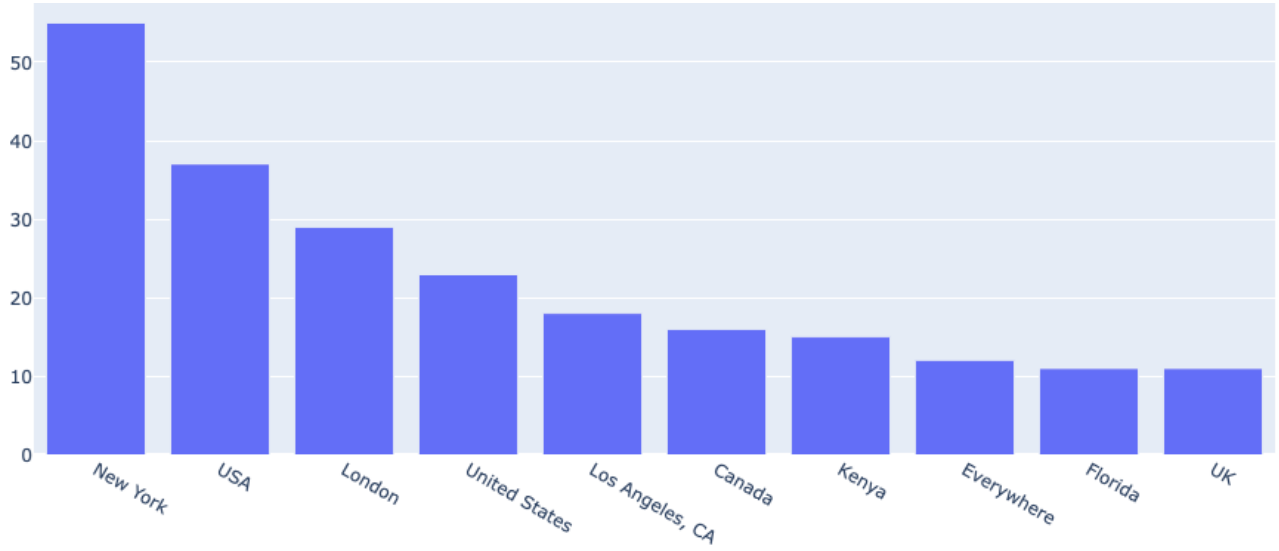
3.3. Location

Na isti način kao što je rađena analiza za *keyword* kolonu, urađena je analiza *tweet*-ova za kolonu *location* koja može biti *null.* U tabeli 3.3.1 su izloženi rezultati nad *train.csv* fajlom.

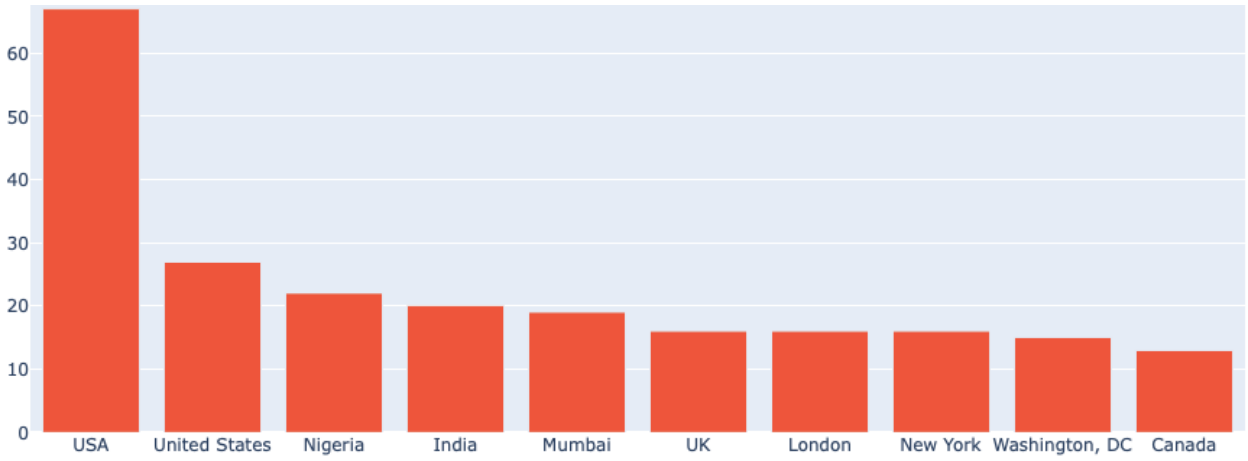
Tabela 3.3.1 Analiza tweet-ova na osnovu *location* kolone

Procenat tweet-ova u kojima nedostaje vrednost za location, posmatrajući sve tweet-ove	33.27 %
Procenat tweet-ova u kojima nedostaje vrednost za location, posmatrajući tweet-ove gde je target=0	33.58%
Procenat tweet-ova u kojima nedostaje vrednost za location, posmatrajući tweet-ove gde je target=1	32.86%

Na slici 3.3.1 možemo da vidimo 10 lokacija koje se najviše koriste u *tweet*-ovima koji imaju vrednost *target*=1, dok na slici 3.3.2 možemo da vidimo 10 lokacija koje se najviše koriste u *tweet*-ovima koji imaju vrednost *target*=0.



Slika 3.3.1 Distribucija tweet-ova (target=0) na osnovu *location* kolone



Slika 3.3.2 Distribucija tweet-ova (target=1) na osnovu *location* kolone

3.4. Text

Analiza *text* kolone se svodi na određivanje dužine teksta. Za svaki *tweet* ćemo odrediti dužinu teksta po broju reči, kao i dužinu teksta po broju karaktera koji se koriste. Kreiraćemo tabelu koja ima iste kolone kao i *train.csv* fajl, samo sa još dve dodatne kolone:

- *text_word_len* – broj reči unutar *text* kolone
- *text_char_len* – broj karaktera unutar *text* kolone

Na slici 3.4.1 je prikazan kod za određivanje broja reči i broja karaktera u tekstu, kao i prikaz prvih pet redova modifikovane tabele, gde su dodate dve nove kolone.

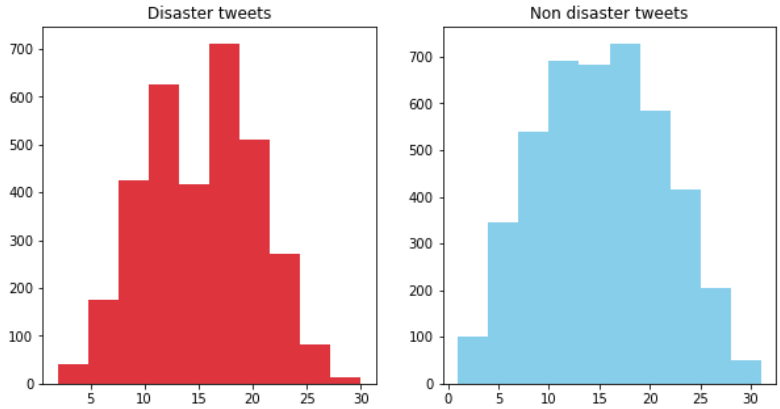
```
def char_count(text):
    cnt_char=0
    for word in text.split():
        cnt_char +=len(word)
    return cnt_char


train_df['text_word_len'] = train_df['text'].apply(lambda x: len(x.split()))
train_df['text_char_len'] = train_df['text'].apply(char_count)

train_df.head()
```

[7]:

	id	keyword	location	text	target	text_word_len	text_char_len
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	13	57
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	7	32
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	22	112
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	8	57
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	16	72

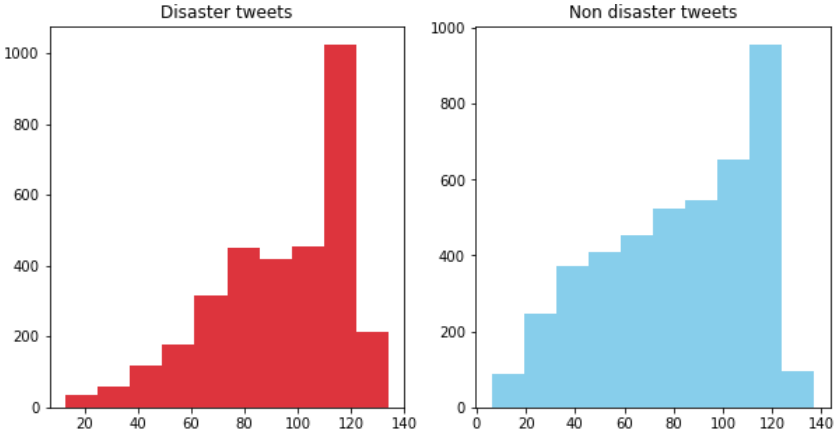
Slika 3.4.1 Dodavanje kolone *text_word_len* i *text_char_len*

Na slici 3.4.2 možemo da vidimo distribuciju *tweet*-ova po opsegu broja reči u koloni *text* koji pišu o stvarnim katastrofama (*Disaster tweets*) i onima koji ne pišu o stvarnim katastrofama (*Non disaster tweets*)



Slika 3.4.2 Analiza broja reči kod *Disaster* i *Non disaster tweet*-ova

Na slici 3.4.3 možemo da vidimo distribuciju *tweet*-ova po posegu broja karaktera u koloni *text* koji pišu o stvarnim katastrofama (*Disaster tweets*) i onima koji ne pišu o stvarnim katastrofama (*Non disaster tweets*)



Slika 3.4.2 Analiza broja karaktera kod *Disaster* i *Non disaster* tweet-ova

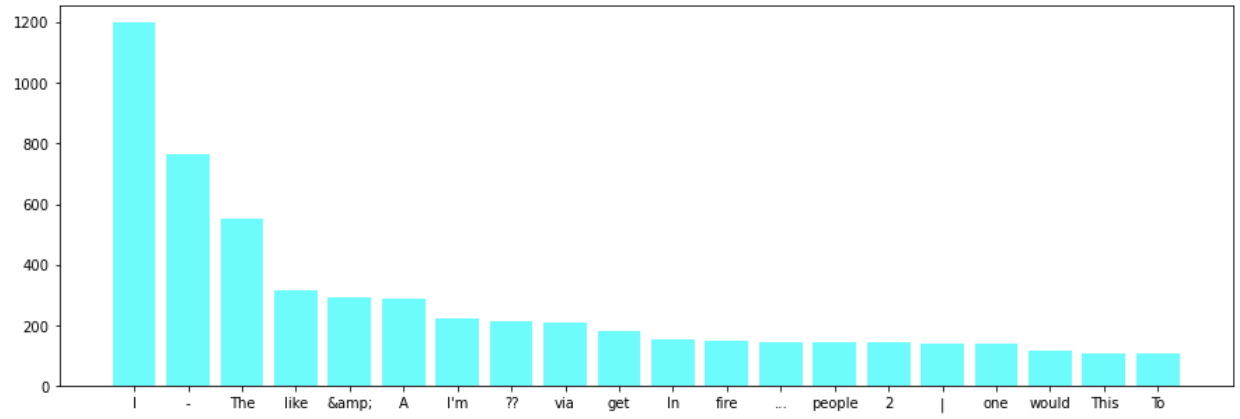
Sada želimo da vidimo rezultate najčešće korišćenih reči koje ne spadaju u skup ***stopwords***. *Stopwords* su reči koje ne sadrže mnogo informacija, pa se zanemarivanjem tih reči ne gubi kontekst samog teksta(‘the’, ‘an’…). Skup *Stopwords* reči možemo da koristimo iz *nltk* biblioteke, specifično za engleski jezik, pošto nam je skup podataka na engleskom jeziku. Prvo ćemo da odredimo

korpus(listu svih reči), a zatim ćemo da prikažemo prvih 20 reči koje se najviše koriste a da nisu u skupu *stopwords*. Na slici 3.4.3 možemo da vidimo način na koji je određen korpus reči.

```
corpus = []
for x in train_df['text'].str.split():
    for i in x:
        corpus.append(i)
```

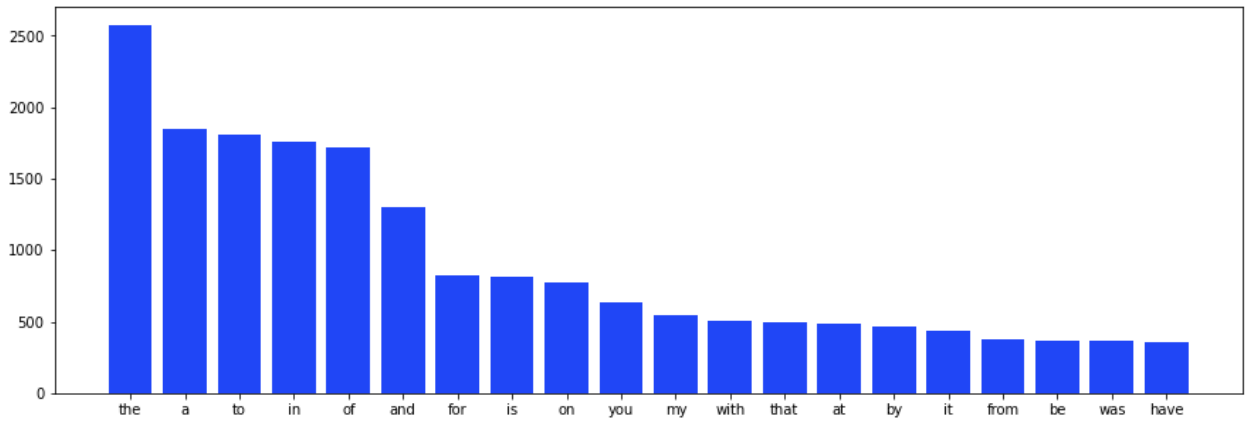
Slika 3.4.3 Kod za određivanje korpusa reči

Na slici 3.4.4 su prikazane 20 najčešće korišćenih reči koje nisu *stopwords,* i na osnovu rezultata možemo da primetimo da su neke od najčešće korišćeni reči specijalni karakteri, koje je potrebno izbaciti iz teksta.

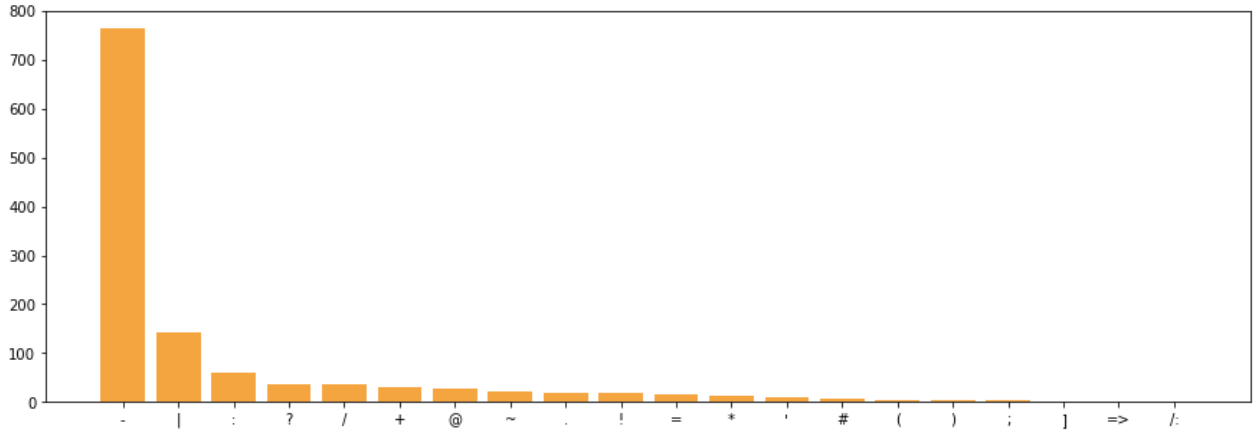


Slika 3.4.4 Najčešće korišćene reči koje nisu *stopwords*

Na slici 3.4.5 i 3.4.6 su prikazane redom 20 najčešće korišćenih reči koje pripadaju skupu *stopwords* kao i 20 najčešće korišćenih znakova interpunkcije.



Slika 3.4.5 Najčešće korišćene reči koje su *stopwords*



Slika 3.4.6 Najčešće korišćeni znakovi interpunkcije

4. TRANSFORMACIJA TEKSTA

Kada hoćemo kompjuterski da analiziramo tekst, teško je razumeti semantičko značenje teksta. Tako je potrebno prilagoditi ulazni tekst, na način koji je pogodan za analizu.

Ono što je potrebno da se uradi jeste sledeće:

1) Pretvaranje teksta u mala slova
2) Uklanjanje specijalnih karaktera (brojevi, url, simboli(*emojis*))
3) Uklanjanje znakova interpunkcije
4) Uklanjanje reči koji pripadaju skupu *stopwords*
5) Proces lematizacije

Na slici 4.1 se nalazi kod koji pretvara tekst u mala slova, uklanja specijalne karaktere kao i znakove interpunkcije, dok se na slici 4.2 nalazi kod koji uklanja sve reči koje su u skupu *stopwords*

```
def clean_text(text):
    text=str(text).lower() #Sve u mala slova
    text=re.sub('\d+', '', text) #uklanjanje brojeva
    text=re.sub('\[.*?\]', '', text) #uklanjanje html tagova
    text=re.sub('https?://\S+|www\.\S+', '', text) #uklanjanje url
    text=re.sub(r"["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", "", text) #removes emojis
    text=re.sub('[%s]' % re.escape(string.punctuation),'',text) #removes punctuations
    return text
```

Slika 4.1 prve tri faze transformacije teksta

```
train_df['clean_text'] = train_df['clean_text'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop]))
train_df.head()
```

Slika 4.2 uklanjanje reči iz skupa *stopwords*

Proces lematizacije jeste način da se reč predstavi preko korena sinonima posmatrane reči. Pri tom procesu se vodi računa da koren sinonima posmatrane reči postoji kao reč u rečniku jezika na kome se tekst piše (u ovom slučaju je to engleski jezik).

Proces lematizacije nad tekstom jeste rađen koristeći *WordNetLemmatizer* klasu iz *nltk* biblioteke. Na slici 4.3 jeste prikazan kod za process lematizacije.

```
lemma = WordNetLemmatizer()

def lematization_process(text):
    text = ' '.join(lemma.lemmatize(word) for word in text.split(' '))
    return text

train_df['final_text'] = train_df['clean_text'].apply(lematization_process)
train_df.head()
```

Slika 4.3 proces lematizacije

Na slikama 4.4, 4.5, 4.6 jesu prikazani prvih 5 *tweet*-ova na kojima možemo da vidimo kako je vršena transformacija teksta redom nakon prve tri faze, četvrte faze i poslednje pete faze (kolona *clean_text* jeste kolona transformacije teksta).

[9]:

	id	keyword	location	text	target	clean_text	text_word_len	text_char_len
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	our deeds are the reason of this earthquake ma...	13	57
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	forest fire near la ronge sask canada	7	32
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	all residents asked to shelter in place are be...	22	112
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	people receive wildfires evacuation orders in...	8	57
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	just got sent this photo from ruby alaska as s...	16	72

Slika 4.4 *tweet*-ovi nakon prve tri faze transformacije teksta

[10]:

	id	keyword	location	text	target	clean_text	text_word_len	text_char_len
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	deeds reason earthquake may allah forgive us	13	57
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	forest fire near la ronge sask canada	7	32
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	residents asked shelter place notified officer...	22	112
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	people receive wildfires evacuation orders cal...	8	57
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	got sent photo ruby alaska smoke wildfires pou...	16	72

Slika 4.5 *tweet*-ovi nakon četvrte faze transformacije teksta

[14]:

	id	keyword	location	text	target	clean_text	text_word_len	text_char_len	final_text
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	deeds reason earthquake may allah forgive us	13	57	deed reason earthquake may allah forgive u
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	forest fire near la ronge sask canada	7	32	forest fire near la ronge sask canada
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	residents asked shelter place notified officer...	22	112	resident asked shelter place notified officer ...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	people receive wildfires evacuation orders cal...	8	57	people receive wildfire evacuation order calif...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	got sent photo ruby alaska smoke wildfires pou...	16	72	got sent photo ruby alaska smoke wildfire pour...

Slika 4.6 *tweet*-ovi nakon pete faze transformacije teksta

5. MODELI MAŠINSKOG UČENJA

Nakon transformacija teksta, urađena je podela podataka na dva dela: odlike(*features*) i mete(*targets*). Odlike podataka predstavlja kolona *final_text* – završetak transformacije ulaznog teksta, dok mete podataka predstavlja kolona *target.* Nakon te podele, takve podatke smo podelili u podatke koje će se koristiti za treniranje(80%) i podatke koji će se koristiti za testiranje(20%), pomoću *train_test_split* metode iz *sckit-learn* biblioteke.

Nakon toga su podaci za treniranje i testiranje koji predstavljaju odlike(*features*) koji su zapravo u tekstualnom obliku, pretvoreni u numerički oblik kako bi bilo lakše da ih koristimo kao ulazne podatke za modele mašinskog učenja. To je urađeno pomoću *TfidfVectorizer* klase iz *scikit-learn* biblioteke. Navedena klasa ima metodu *fit_transform* koja rečima iz ulaznog teksta dodeljuje odgovarajuće težine. TF-IDF (*Term Frequency – Inverse Document Frequency*) ima za cilj da prikaže koliku važnost neka reč ima u nekom tekstualnom dokumentu. Na slici 5.1 možemo da vidimo način na koji je to urađeno.

```
tfidf=TfidfVectorizer()
X_train=tfidf.fit_transform(X_train).toarray()
X_test=tfidf.transform(X_test).toarray()
```

Slika 5.1 Pretvaranje teksta u numerički oblik

Modeli mašinskog učenja su trenirani nad skupom podataka za treniranje, a kasnije su testirani nad skupom podataka za testiranje. Procenu rada modela jeste predstavljena pomoću matrice konfuzije koja ima 4 moguća ishoda(predikcije) (*TruePositive, FalsePositive, TrueNegative, FalseNegative)* i pomoću *ROC* krive.

Takođe za svaki model je računato:

- tačnost(*accuracy*) – predstavlja procenat tačno predviđenih rezultata.
- Preciznost(precision) – predstavlja odnos *TruePositive* predikcija sa sumom *TruePositive* i *FalsePositive* predikcija.
- *Recall* - predstavlja odnos *TruePositive* predikcija sa sumom *TruePositive* i *FalseNegative* predikcija.
- F1 vrednost – harmonijska sredina od *precision* i *recall* vrednosti

Na slici 5.2 možemo da vidimo funkciju koja trenira zadati model i računa njegove rezultate.

```python
def train_model(model):
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    y_prob=model.predict_proba(X_test)
    accuracy=round(accuracy_score(y_test,y_pred),3)
    precision=round(precision_score(y_test,y_pred,average='weighted'),3)
    recall=round(recall_score(y_test,y_pred,average='weighted'),3)
    fscore = f1_score(y_test, y_pred)

    print(f'Accuracy of the model: {np.round(accuracy*100,2)}%')
    print(f'Precision Score of the model: {np.round(precision*100,2)}%')
    print(f'Recall Score of the model: {np.round(recall*100,2)}%')
    print(f'F1_Score of the model: {np.round(fscore*100,2)}%')
```

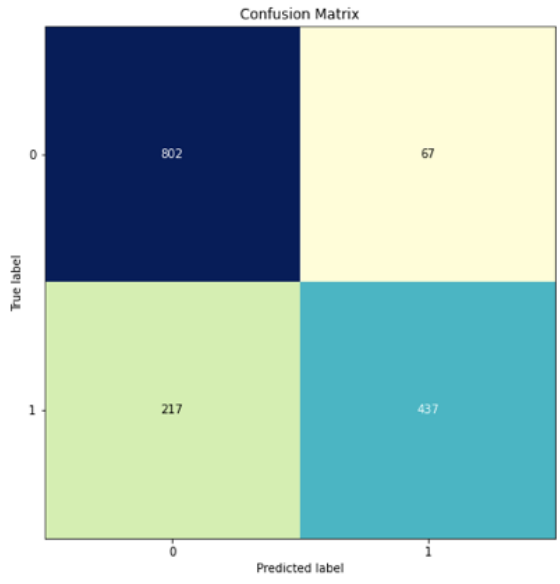
Slika 5.2 Treniranje odgovarajućeg modela

5.1. Bernoulli Naive Bayes

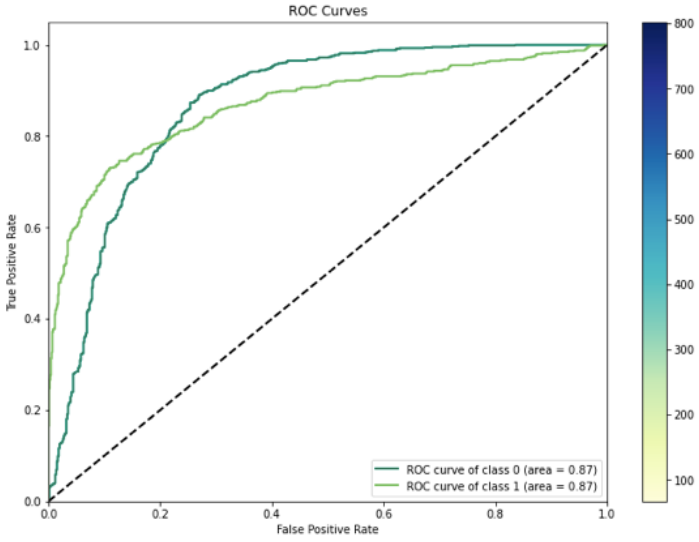
Korišćena je klasa *BernoulliNB* iz *sklearn* biblioteke. U tabeli 5.1.1 možemo da vidimo rezultate modela (accuracy, precision, recall, F1), dok na slici 5.1.1 i 5.1.2 možemo da vidimo matricu konfuzije kao i ROC krivu modela.

Tabela 5.1.1 Rezultati modela BNB

Accuracy	81.4 %
Precision	82.1%
Recall	81.4%
F1	75.47%



Slika 5.1.1 Matrica konfuzije za model BNB



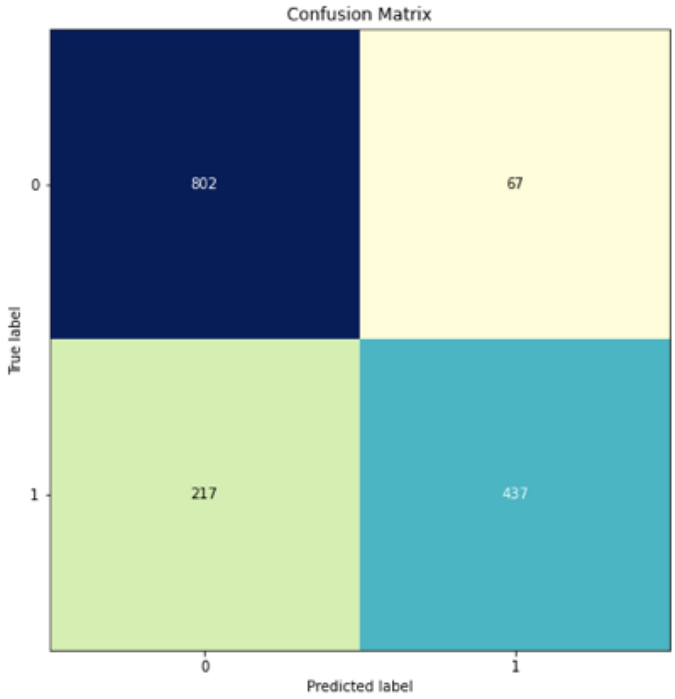
Slika 5.1.2 ROC kriva za model BNB

5.2. Logisticka regresija

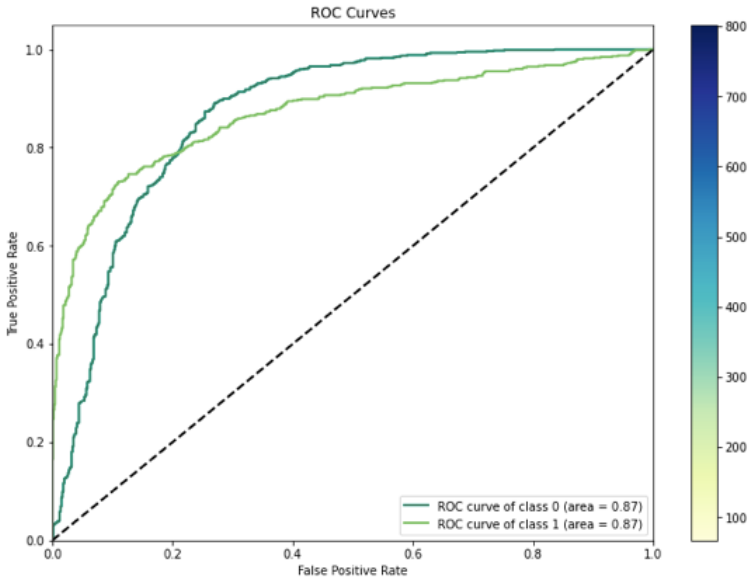
Korišćena je klasa *LogisticRegression* iz *sklearn* biblioteke. U tabeli 5.2.1 možemo da vidimo rezultate modela (accuracy, precision, recall, F1), dok na slici 5.2.1 i 5.2.2 možemo da vidimo matricu konfuzije kao i ROC krivu modela.

Tabela 5.2.1 Rezultati za model logisticke regresije

Accuracy	81.7 %
Precision	82.0%
Recall	81.7%
F1	76.61%



Slika 5.2.1 Matrica konfuzije za model logističke regresije



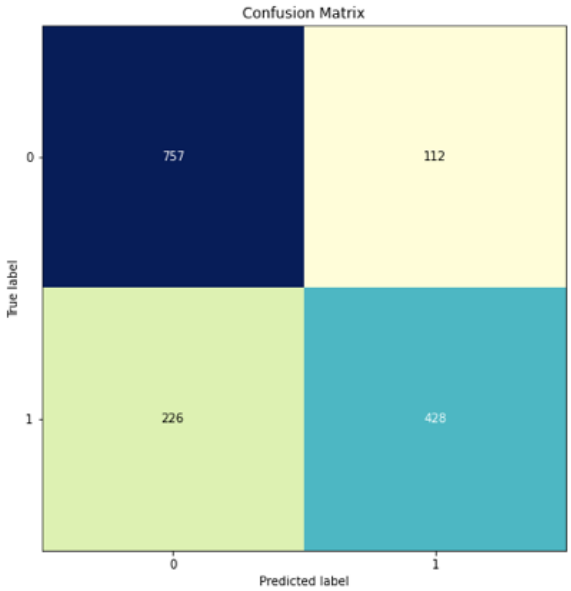
Slika 5.2.2 ROC kriva za model logističke regresije

5.3. K najbližih suseda

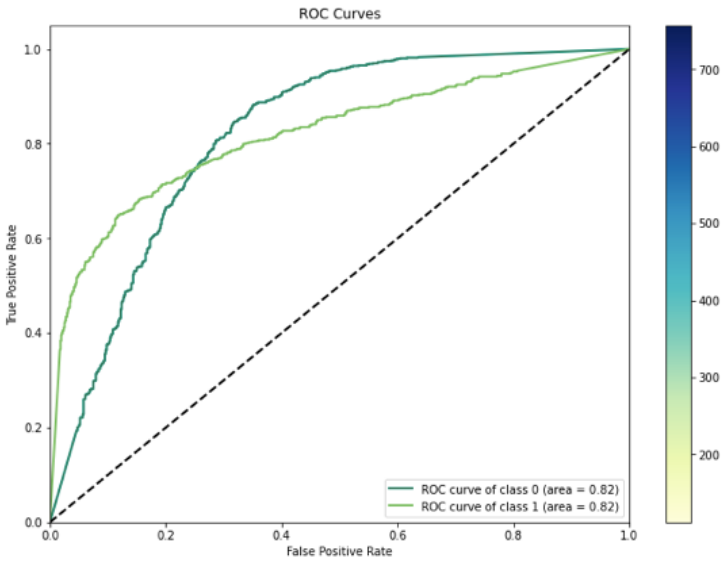
Korišćena je klasa *KNeighborsClassifier* iz *sklearn* biblioteke. Za parametar *k* je uzeta vrednost broja 7. U tabeli 5.3.1 možemo da vidimo rezultate modela (accuracy, precision, recall, F1), dok na slici 5.3.1 i 5.3.2 možemo da vidimo matricu konfuzije kao i ROC krivu modela.

Tabela 5.3.1 Rezultati modela *k najbližih suseda(k = 7)*

Accuracy	77.8 %
Precision	78.0%
Recall	77.8%
F1	71.69%



Slika 5.3.1 matrica konfuzije za model *k najbližih suseda(k=7)*



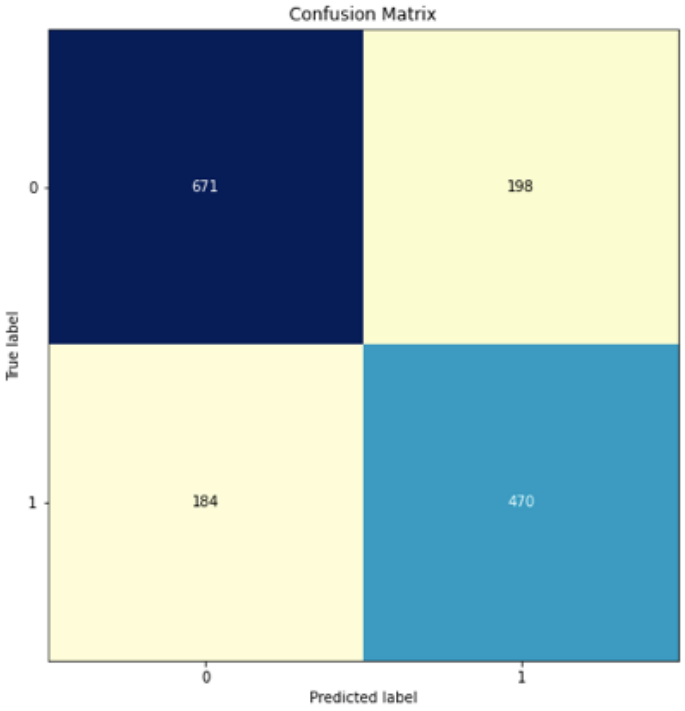
Slika 5.3.2 ROC kriva za model *k najbližih suseda(k=7)*

5.4. Stablo odluke

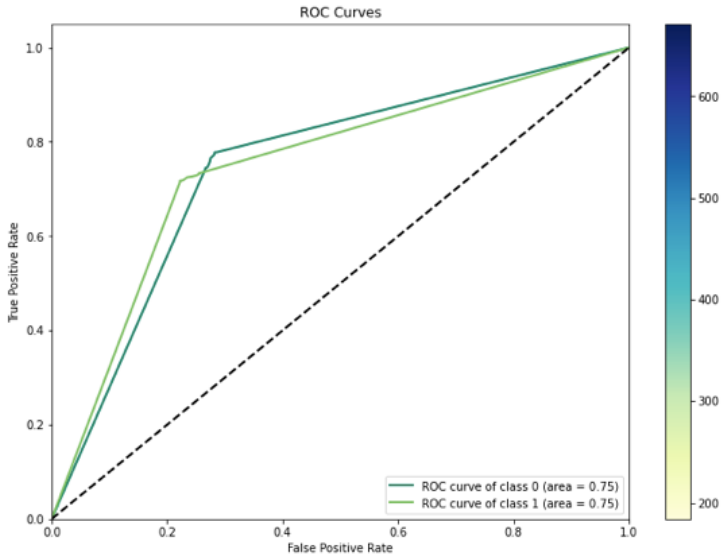
Korišćena je klasa *DecisionTreeClassifier* iz *sklearn* biblioteke. U tabeli 5.4.1 možemo da vidimo rezultate modela (accuracy, precision, recall, F1), dok na slici 5.4.1 i 5.4.2 možemo da vidimo matricu konfuzije kao i ROC krivu modela.

Tabela 5.4.1 Rezultati modela *stablo odluke*

Accuracy	74.9 %
Precision	75.0%
Recall	74.9%
F1	71.1%



Slika 5.4.1 matrica konfuzije za model *stablo odluke*



Slika 5.4.2 ROC kriva za model *stablo odluke*

5.5. Rezultati

Na osnovu rezultata, možemo da primetimo da najbolje vrednosti *accuracy* rezultata imaju model logističke regresije (81.7%) i Bernoulli Naive Bayes model (81.4%). Ukoliko bismo poredili po F1 rezultatima, najbolje rezultate takođe imaju model logističke regresije (76.61%) i Bernoulli Naive Bayes model (75.47%). Tako da na osnovu rezultata, možemo da zaključimo da je od navedena 4 modela, najbolje koristiti model logističke regresije.

6. MODELI NEURALNIH MREŽA

U ovom poglavlju su predstavljeni i analizirani određeni modeli neuralnih mreža pomoću koji je rešavan problem.

6.1. Priprema teksta

Pre samog korišćenja modela neuralnih mreža, kao kod modela za mašinsko učenje, urađena je transformacija *text* kolone u svim *tweet*-ovima. Transformacija se sastojala: pretvaranje teksta u mala slova, uklanjanje specijalnih karaktera, znakova interpunkcije kao i reči koje pripadaju skupu *stopwords*, i na samom kraju proces lematizacije.

Na slici 6.1.1 može da se vidi način na koji je urađena tokenizacija transformisanog teksta *tweet*-ova. Korišćena je *Tokenizer* klasa iz *keras* biblioteke, i na ovaj način su određene sve jedinstvene reči koje postoje u *tweet*-ovima, i tim jedinstvenim rečima je dodeljen odgovarajući indeks.

```
In [10]:
train_tweets = train_df['final_text'].values
train_target = train_df['target'].values
test_tweets = test_df['final_text'].values
```

```
In [11]:
# Tokenizacija!!!
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(train_tweets)

vocab_length = len(word_tokenizer.word_index) + 1
```

Slika 6.1.1 Tokenizacija transformisanog teksta

Nakon tokenizacije ulaznog teksta, na slici 6.1.2 je prikazan način pretvaranja rečenica (*tweet*-ova) u sekvencu brojeva, koji su dobijeni iz tokenizacije. Kako nisu svi *tweet*-ovi iste dužine, odrađen je *padding*, odnosno dodavanje nula na kraj sekvence onim *tweet*-ovima koji su kraće dužine. Ovakve sekvence su pogodne kao ulazni podaci kod modela neuralnih mreža kao i njihovu dalju analizu.

```
def embed(corpus):
    return word_tokenizer.texts_to_sequences(corpus)
```

```
In [13]:
#Padding!
longest_train = max(train_tweets, key=lambda sentence: len(word_tokenize(sentence)))
length_long_sentence = len(word_tokenize(longest_train))

train_padded_sentences = pad_sequences(
    embed(train_tweets),
    length_long_sentence,
    padding='post'
)
test_padded_sentences = pad_sequences(
    embed(test_tweets),
    length_long_sentence,
    padding='post'
)
```

Slika 6.1.2 Pretvaranje u sekvencu brojeva

Jedinstvene reči koje smo odredili u posmatranim *tweet*-ovima, ćemo predstaviti pomoću vektora koji se sastoji od 100 dimenzija. To je moguće uraditi koristeći već kreiran *GloVe* (*Global Vectors for Word Representation*) fajl koji ima 100 dimenzija. *GloVe* fajl jeste testiran nad velikom količinom teksta i pokazuje semantičku i sintaksnu vezu izmedju reči upravo preko 100 dimenzija. Ukoliko se naša reč nalazi u *GloVe* fajlu, onda ćemo da je predstavimo preko vektora iz *GloVe* fajla, dok će u drugom slučaju biti predstavljen vektorom dužine 100 koji je popunjen nulama. Na slici 6.1.3 može da se vidi način kako je korišćen *GloVe* fajl sa 100 dimenzija.

```
embeddings_dictionary = dict()
embedding_dim = 100

# Load GloVe 100D embeddings
with open('/kaggle/input/glove6b100dtxt/glove.6B.100d.txt') as fp:
    for line in fp.readlines():
        records = line.split()
        word = records[0]
        vector_dimensions = np.asarray(records[1:], dtype='float32')
        embeddings_dictionary [word] = vector_dimensions

# embeddings_dictionary
```

```
5]:
# Now we will load embedding vectors of those words that appear in the
# Glove dictionary. Others will be initialized to 0.

embedding_matrix = np.zeros((vocab_length, embedding_dim))

for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

Slika 6.1.3 GloVe fajl

6.2. Podela skupa podataka

Na slici 6.2.1 možemo da vidimo kako smo podelili skup podataka na skup za treniranje i na skup za testiranje (validaciju). Za testiranje je korišćeno 25% celokupnog skupa podataka, dok je za treniranje korišćeno 75%.

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    train_padded_sentences,
    train_target,
    test_size=0.25
)
```

Slika 6.2.1 Podela skupa podataka

6.3. Kreiranje slojeva kod modela neuralnih mreža

Kreiranje modela neuralnih mreža jeste izvršeno pomoću *Sequential* klase iz *keras* biblioteke, i pomoću koje možemo da dodajemo slojeve (*layers*) neurona koristeći *add* metodu.

Analizirali smo tri modela neuralnih mreža, a to su: *RNN* (*Recurrent Neural Network*), *LSTM* (Long Short-Term Memory), *BiLSTM* (*Bidirectional LSTM*). Za sva tri modela su kreirani određeni slojevi (*layers*) neurona.

Prvi *layer* kod svakog modela jeste *Embedding layer* koji se koristi iz pomoću biblioteke *keras,* on kao ulazne parametre ima veličinu rečnika, odnosno broj jedinstvenih reči, zatim veličinu vektora pomoću koje se predstavlja svaka reč, težine – koje smo odredili pomoću *GloVe* fajla i time imamo informacije semantičkoj i sintatskoj relaciji između reči, i na kraju imamo veličinu najvećeg *tweet*-a po broju reči. Na slici 6.3.1 možemo da vidimo dodavanje *Embedding layer*-a.

```
model.add(Embedding(
    input_dim=embedding_matrix.shape[0],
    output_dim=embedding_matrix.shape[1],
    weights = [embedding_matrix],
    input_length=length_long_sentence
))
```

Slika 6.3.1 Dodavanje Embedding layer-a

Drugi *layer* predstavlja layer izabranog modela (RNN, LSTM, BiLSTM). Služi da se procesiraju reči iz prethodnog *layer*-a. Za kreiranje ovog *layer*-a, korišćene su klase kao što su *SimpleRNN*, *LSTM, Bidirectional* iz *keras* biblioteke. Na slikama 6.3.2, 6.3.3, 6.3.4 možemo da vidimo način dodavanja drugog *layer*-a u zavisnosti od modela.

```
model.add(LSTM(
    length_long_sentence,
    return_sequences = True,
    recurrent_dropout=0.2
))
```

Slika 6.3.2 Drugi *layer* LSTM modela

```
model.add(SimpleRNN(
    length_long_sentence
))
```

Slika 6.3.3 Drugi *layer* RNN modela

```
model.add(Bidirectional(LSTM(
    length_long_sentence,
    return_sequences = True,
    recurrent_dropout=0.2
)))
```

Slika 6.3.4 Drugi layer BiLSTM modela

Ostali *layeri* koji su korišćeni u svakom modelu jesu *GlobalMaxPool1D*, *BatchNormalization, DropOut, Dense* koji su takođe iz *keras* biblioteke. *GlobalMaxPool1D layer* služi da se smanji dimenzionalnost, tako što će se za svaku odliku (*feature*) uzeti maksimalnu vrednost, i na taj način pomoći modelu da očuva najvažnije informacije za svaku odliku (*feature*). *BatchNormalization layer* služi da se ubrza proces treniranja, pomaže da izlaz iz *layera* ne bude previše mali ili previše veliki što može da uspori proces treniranja.

DropOut layer služi da se odbaci određeni procent podataka koji su došli iz prethodnog *layera* i na taj način pomogne da ne dođe do previše treniranog modela (*overfitting*). *Dense layer* potpuno povezani *layer* koji povezuje svaki neuron iz prethodnog, sa svakim neuronom u trenutnom *layer*-u. Na izlaz iz prethodnog layera se primenjuje linearna transformacija i aktivacijska funkcija (kao na primer *Rectified Linear Unit* koja je zapravo *f(x)=max(0, x)*. *Sigmoid* funkcija je još jedan primer, koja mapira ulazne vrednosti u vrednosti između 0 i 1.) Na slici 6.3.5 možemo da vidimo dodavanje ostalih *layera.*

```
model.add(GlobalMaxPool1D())
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(length_long_sentence, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(length_long_sentence, activation = "relu"))
model.add(Dropout(0.5))

model.add(Dense(1, activation = 'sigmoid'))
```

Slika 6.3.5 Završni *layer*-i za modele neuralnih mreža

6.4. RNN

Prvi model neuralnih mreža koji je korišćen jeste *RNN* (*Recurrent Neural Network*). Na slici 6.4.1 možemo da vidimo kod kojim treniramo naš model, gde se pozivanje funkcije simple_rnn() kreiraju *layer*-i na način koji je opisan u poglavlju 6.3.

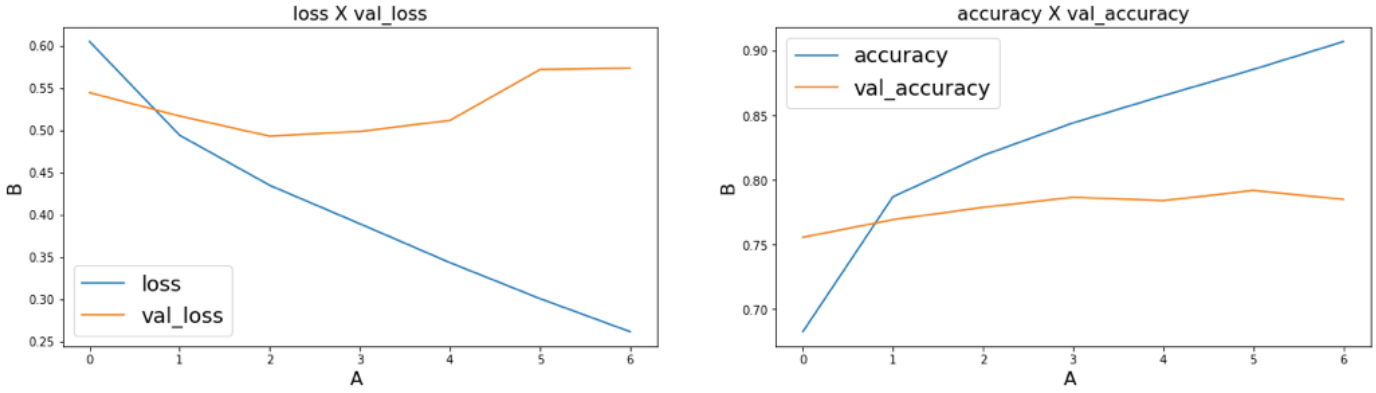
```
model = simple_rnn()

history = model.fit(
    X_train,
    y_train,
    epochs = 7,
    batch_size = 32,
    validation_data = (X_test, y_test),
    verbose = 1
)
```

Slika 6.4.1 RNN model

U tabeli 6.4.1 možemo da vidimo rezultate RNN modela, dok na slici 6.4.2 možemo da vidimo grafički prikaz za *Loss* i *Val_Loss* kao i *Accuracy* i *Val_Accuracy*. *Loss* predstavlja grešku između predviđenog rezultata i stvarnog rezultata nad skupom podataka za treniranje. *Val_Loss* predstavlja grešku između predviđenog rezultata i stvarnog rezultata ali nad skupom podataka za validaciju. *Accuracy* predstavlja procenat dobro procenjenih primera.

Tablea 6.4.1 Rezultati za RNN model

Accuracy	78.5 %
Precision	63.0%
Recall	83.3%
F1	71.7%



Slika 6.4.2 RNN – Loss, Val_Loss, Accuracy, Val_Accuracy

Možemo da primetimo da *loss* tokom vremena opada, dok *val_loss* tokom vremena raste, što može da bude indikacija za *overfitting*.

6.5. LSTM

Drugi model neuralnih mreža koji je korišćen jeste *LSTM* (*Long Short-Term Memory*). Na slici 6.5.1 možemo da vidimo kod kojim treniramo naš model, gde se pozivanje funkcije glove_lstm() kreiraju *layer*-i na način koji je opisan u poglavlju 6.3.

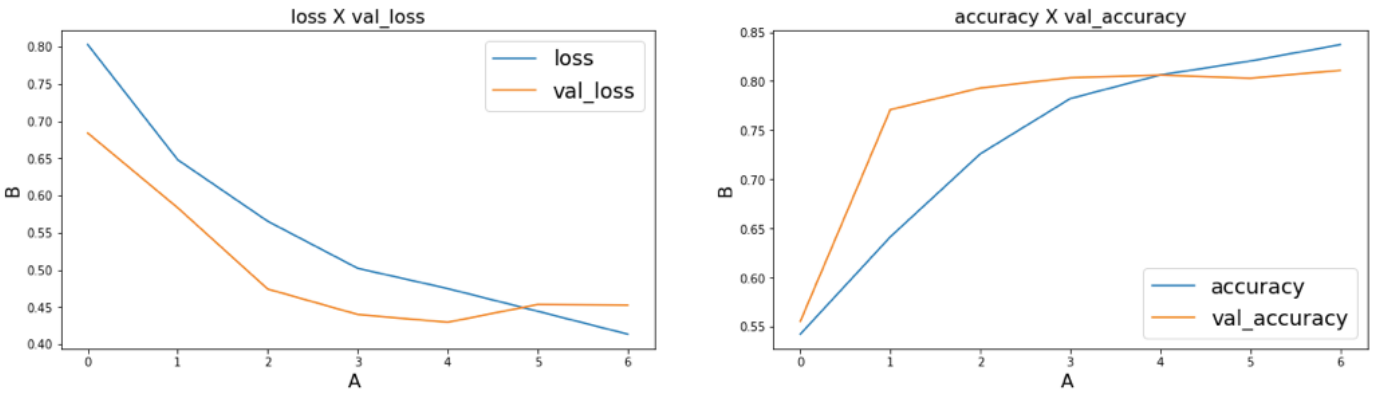
```
model = glove_lstm()

history = model.fit(
    X_train,
    y_train,
    epochs = 7,
    batch_size = 32,
    validation_data = (X_test, y_test),
    verbose = 1
)
```

Slika 6.5.1 LSTM model

U tabeli 6.5.1 možemo da vidimo rezultate LSTM modela, dok na slici 6.5.2 možemo da vidimo grafički prikaz za *Loss* i *Val_Loss* kao i *Accuracy* i *Val_Accuracy*. *Loss* predstavlja grešku između predviđenog rezultata i stvarnog rezultata nad skupom podataka za treniranje. *Val_Loss* predstavlja grešku između predviđenog rezultata i stvarnog rezultata ali nad skupom podataka za validaciju. *Accuracy* predstavlja procenat dobro procenjenih primera.

Tabela 6.5.1 Rezultati za LSTM model

Accuracy	81.1 %
Precision	76.8%
Recall	78.9%
F1	77.8%



Slika 6.5.2 LSTM – Loss, Val_Loss, Accuracy, Val_Accuracy

Kod LSTM modela, *loss* vremenom opada, kao i *val_loss*, što može biti indikacija da ne postoji *overfitting*.

6.6. BiLSTM

Treći model neuralnih mreža koji je korišćen jeste *BiLSTM* (Bidirectional *Long Short-Term Memory*). Na slici 6.6.1 možemo da vidimo kod kojim treniramo naš model, gde se pozivanje funkcije bidirectional_lstm() kreiraju *layer*-i na način koji je opisan u poglavlju 6.3.

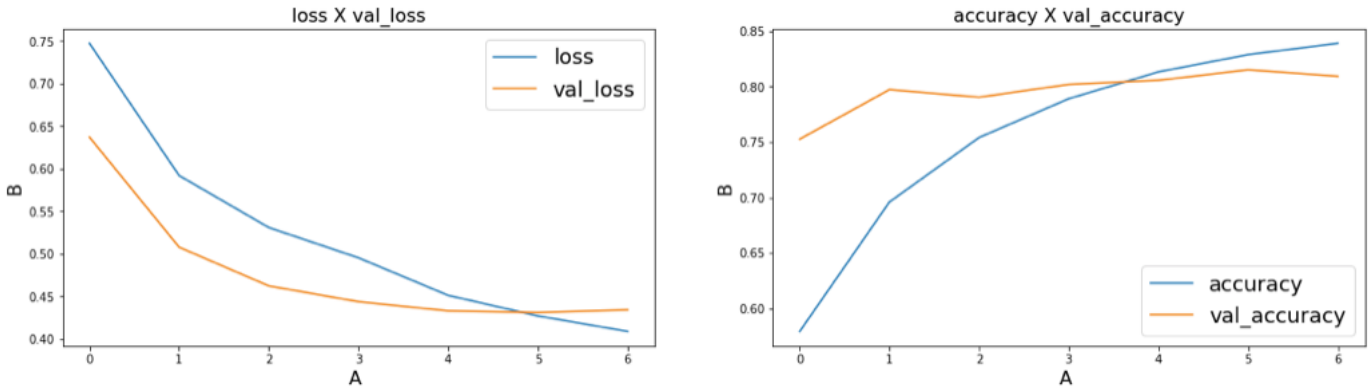
```
model = bidirectional_lstm()

history = model.fit(
    X_train,
    y_train,
    epochs = 7,
    batch_size = 32,
    validation_data = (X_test, y_test),
    verbose = 1
)
```

Slika 6.6.1 BiLSTM model

U tabeli 6.6.1 možemo da vidimo rezultate BiLSTM modela, dok na slici 6.6.2 možemo da vidimo grafički prikaz za *Loss* i *Val_Loss* kao i *Accuracy* i *Val_Accuracy*. *Loss* predstavlja grešku između predviđenog rezultata i stvarnog rezultata nad skupom podataka za treniranje. *Val_Loss* predstavlja grešku između predviđenog rezultata i stvarnog rezultata ali nad skupom podataka za validaciju. *Accuracy* predstavlja procenat dobro procenjenih primera.

Tabela 6.6.1 Rezultati za BiLSTM model

Accuracy	80.9 %
Precision	67.6%
Recall	85.3%
F1	75.4%



Slika 6.6.2 BiLSTM – Loss, Val_Loss, Accuracy, Val_Accuracy

Kod BiLSTM modela možemo da primetimo da tokom vremen *loss* i *val_loss* opadaju što je indikacija može biti indikacija da ne postoji *overfitting*.

6.7. Rezultati

Na osnovu analize rezultata prethodna tri modela, možemo da primetimo da najslabije performanse ima RNN model za koji smo uočili da ima indikacije za *overfitting* jer se *loss* tokom vremena smanjuje, dok se *val_loss* tokom vremena povećava. Takodje Accuracy iznosi 78.5%. LSTM i BiLSTM modeli imaju slične performanse, nemaju indikacije za *overfitting*, međutim LSTM model ima Accuracy = 81.1% dok BiLSTM model ima Accuracy = 80.9%. Takodje LSTM ima F1_score = 77.8% dok BiLSTM ima F1_score = 75.4%. Na osnovu ovoga, možemo da zaključimo da od navedena 3 modela za neuralne mreže, LSTM model ima najbolje rezultate.

LITERATURA

[1] Natural Language Processing with Disaster Tweets [Online] Available: https://www.kaggle.com/competitions/nlp-getting-started (30.01.2023.)

SPISAK SLIKA

SPISAK TABELA