Twitter Sentimental Analysis

--a Machine Learning project

Report

**ABSTRACT**

The objective of this task is to detect hate speech in tweets. For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets***.***

Formally, given a training sample of tweets and labels, where label ‗1‗ denotes the tweet isracist/sexist and label ‗0‗ denotes the tweet isnot racist/sexist, your objective isto predict the labels on the given test dataset.

We willdo so byfollowing a sequence ofsteps needed to solve a general sentiment analysis problem. We will start with preprocessing and cleaning of the raw text of the tweets. Then we willexplore the cleaned text and try to get some intuition about the context of the tweets. After that, we will extract numerical features from the data and finally use these feature sets to train models and identify the sentiments of the tweets

## INTRODUCTION

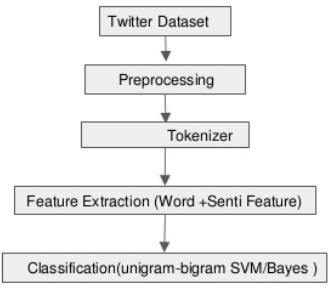
Natural Language Processing (NLP) is a hotbed of research in data science these days and one of the most common applications of NLP is sentiment analysis. From opinion polls to creating entire marketing strategies, this domain has completely reshaped the way businesses work, which is why this is an area every data scientist must be familiar with.

Thousands of text documents can be processed for sentiment (and other features including named entities, topics, themes, etc.) in seconds, compared to the hours it would take a team of people to manually complete the same task.

.

## Related Works:-

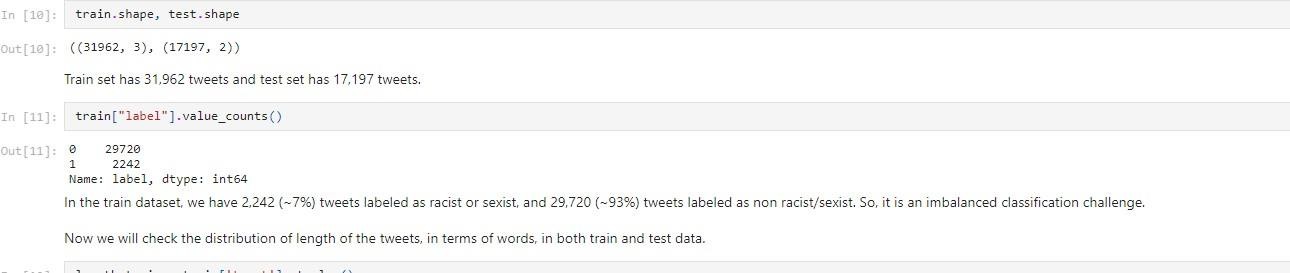
1. Understand the Problem Statement
2. Tweets Pre processing and Cleaning
3. Story Generation and Visualization from Tweets
4. Extracting Features from Cleaned Tweets
5. Model Building: Sentiment Analysis

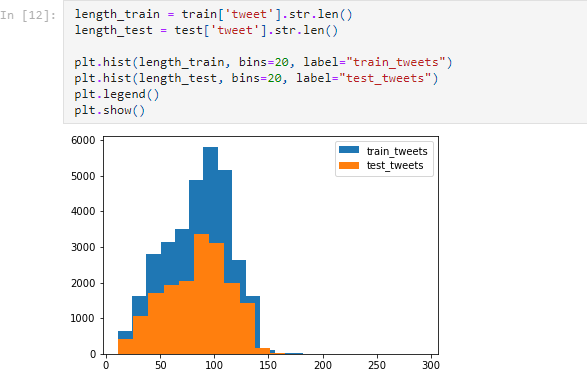


**Methodology**

Analysing data

Inthefirst step wehave analysed thedata settoknow how muchand whattypeofdata are we dealing here with. First of all we have found the total numbers of entries in the data set and thereafter proceeded to finding other valuable information like the length of each tweet. We have plotted this information using a graph.



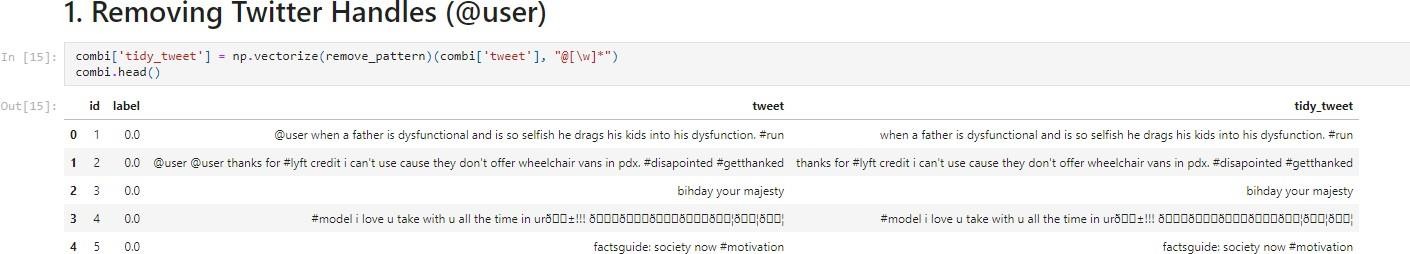


## Data Cleaning

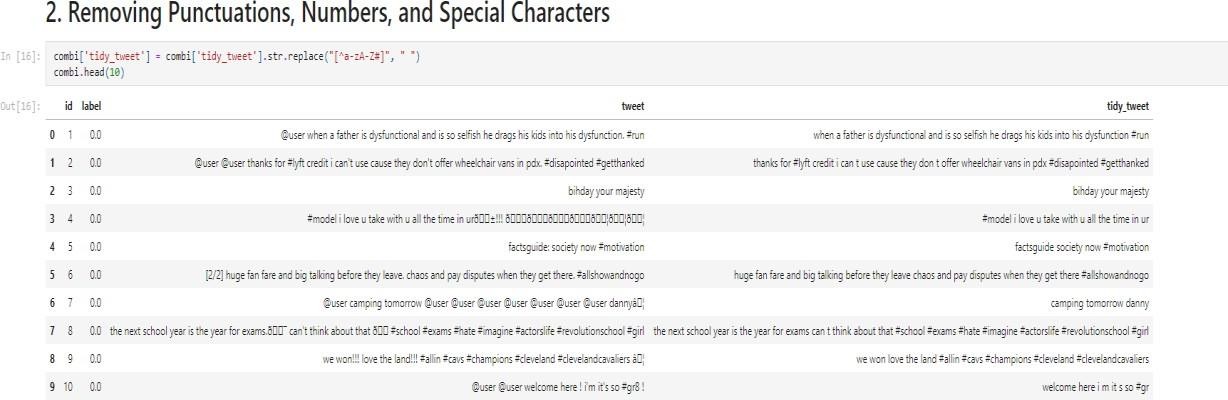
In this part of our project we have tried to pre process the data before applying any machine learning algorithm to it.

What this means is the data set that we have here consist of various unwanted elements which will bring down the accuracy of our model if not removed.

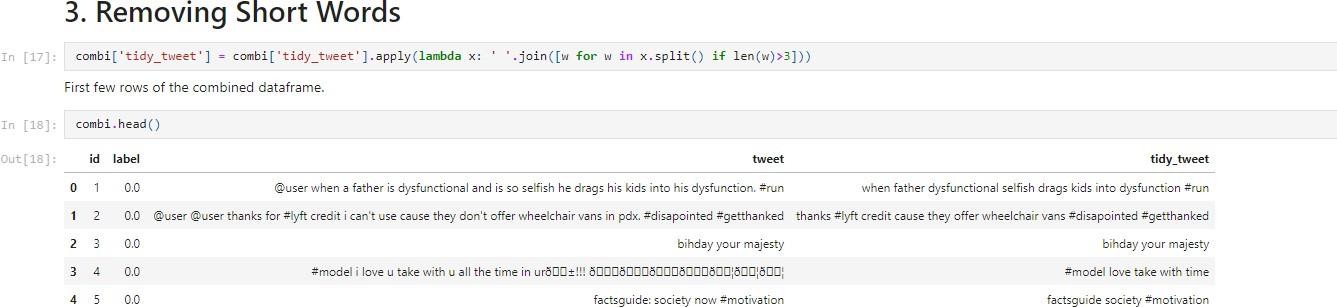
Hence we have removed the elements which include user handles like @user1. As mentioned above, the tweets contain lots of twitter handles (@user) that is how a Twitter user acknowledged on Twitter. We will remove all these twitter handles from the data as they don‗t convey much information.



Punctuations, numbers and special characters do not help much. It is better to remove them from the text just as we removed the twitter handles. Here we will replace everything except characters and hash tags with spaces.



Terms like ―hmm‖, ―oh‖ are of very little use. It is better to get rid of them. Hence we are removing all the words which have length less than 3.



Now this leaves us with filtered data to work with.

## Tokenisation andNormalisation

Now we will tokenize all the cleaned tweets in our dataset. Tokens are individual terms or words, and tokenization is the process of splitting a string of text into tokens.

Along with which stemming has been done. Stemming is a rule-based process of stripping the suffixes (―ing‖, ―ly‖, ―es‖, ―s‖ etc) from a word. For example, For example –

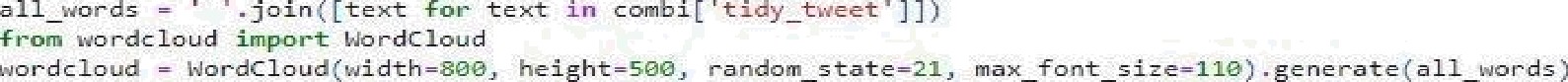
―play‖, ―player‖, ―played‖, ―plays‖ and ―playing‖ are the different variations of the word –



## StoryGenerationandVisualizationfromTweets

To see how well the given sentiments are distributed across the train dataset we have used the method of plotting word clouds. A word cloud is a visualization wherein the most frequent words appear in large size and the less frequent words appear in smaller sizes.

V is ufaiztioil of Eva rd s u sing worctctaud plot.



pit.figure(figsize=(i0, 7)) pit.imshow(wordcloud, intewpolation=”bilinear™)

pIt\*shou()



1ifei. .thi

peo

Lack

o ay

foil will

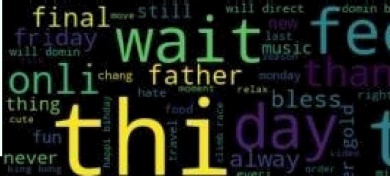
great r n o+

norma1\_words = ' ' . join ( [text -for text in comb1§ ' I idy\_tiveet ’ ] [ comb1 [ ' label ’ == 0) ] }

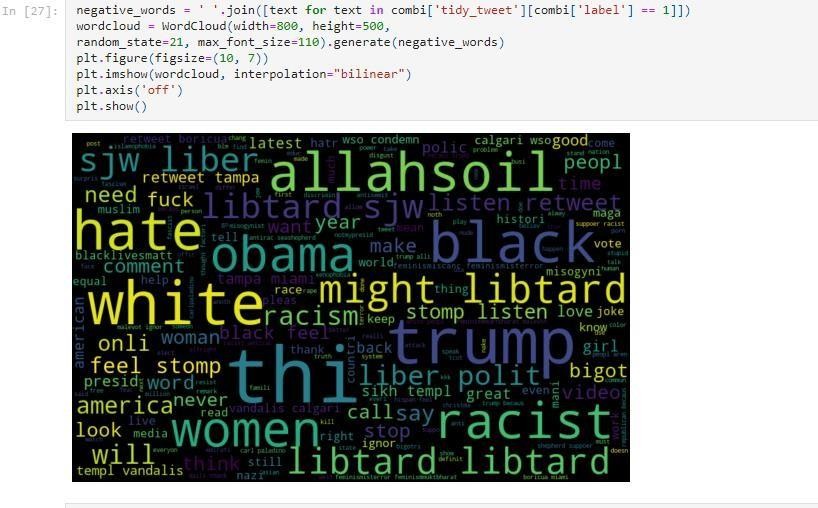
uordcloud = lVordCloud(uidth=a0e, height=500, random\_state=21, max\_{ont\_size=1I0).generate(normal\_words) pit.figure(figsize=(10, 7))

pit.imshow(wordcloud, interpolation="bilinear”)

plt . s h out ( )

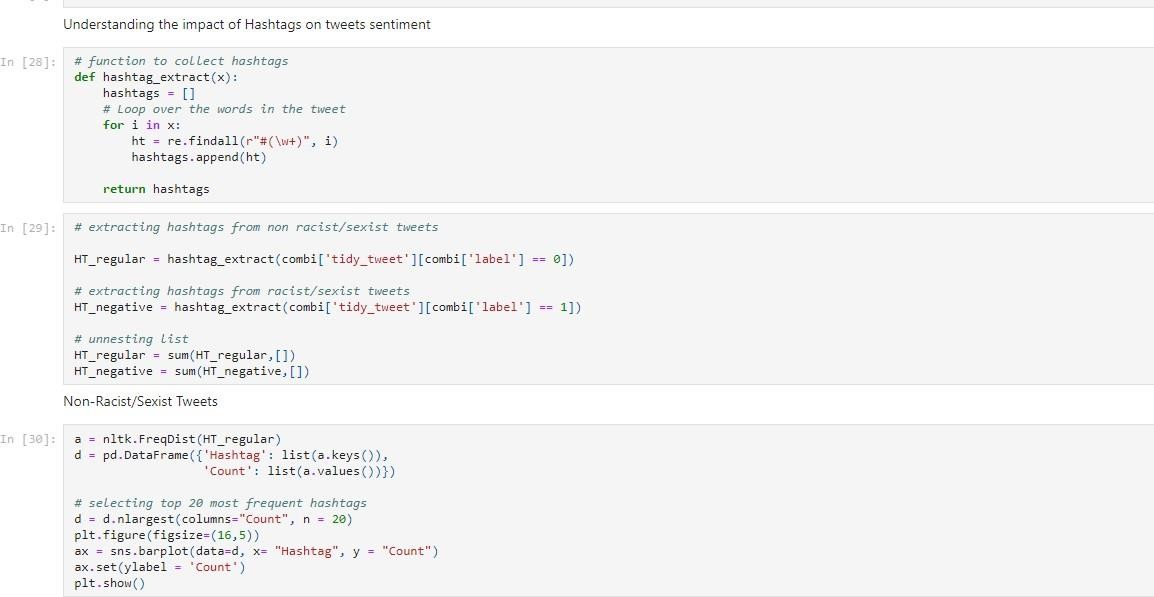


Similarly we made word cloud for racist and sexist tweets.

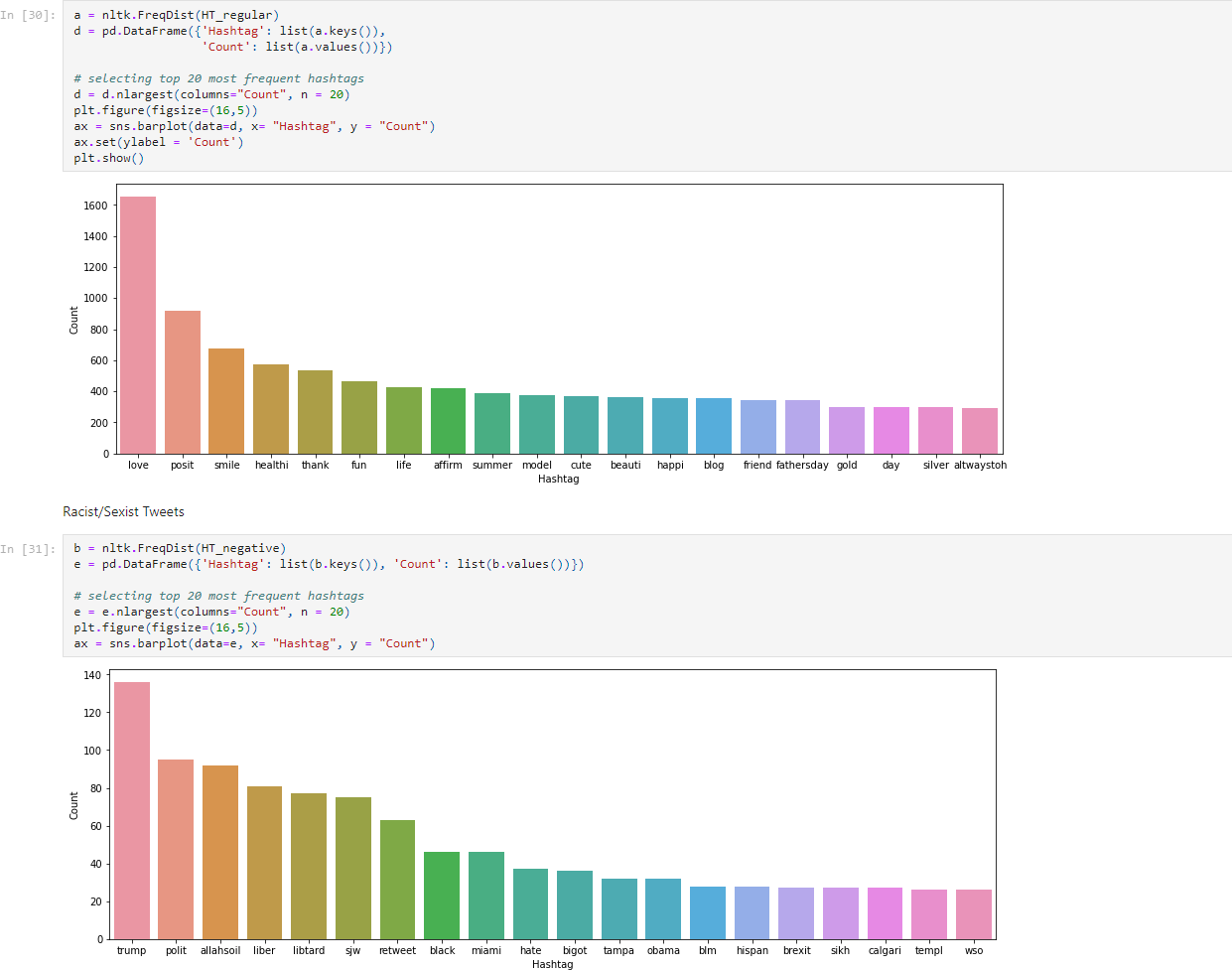


## Working with hashtags

Hash tags in twitter are synonymous with the ongoing trends on twitter at any particular point in time. We tried to check whether these hash tags add any value to our sentiment analysis task that isifthey help in distinguishing tweets intothe different sentiments.



Next we plotted a graph of the number of times these hash tags were found in the tweet. One graph depicts the hash tags in non racist and non sexist tweets while other is for racist and sexist tweets.

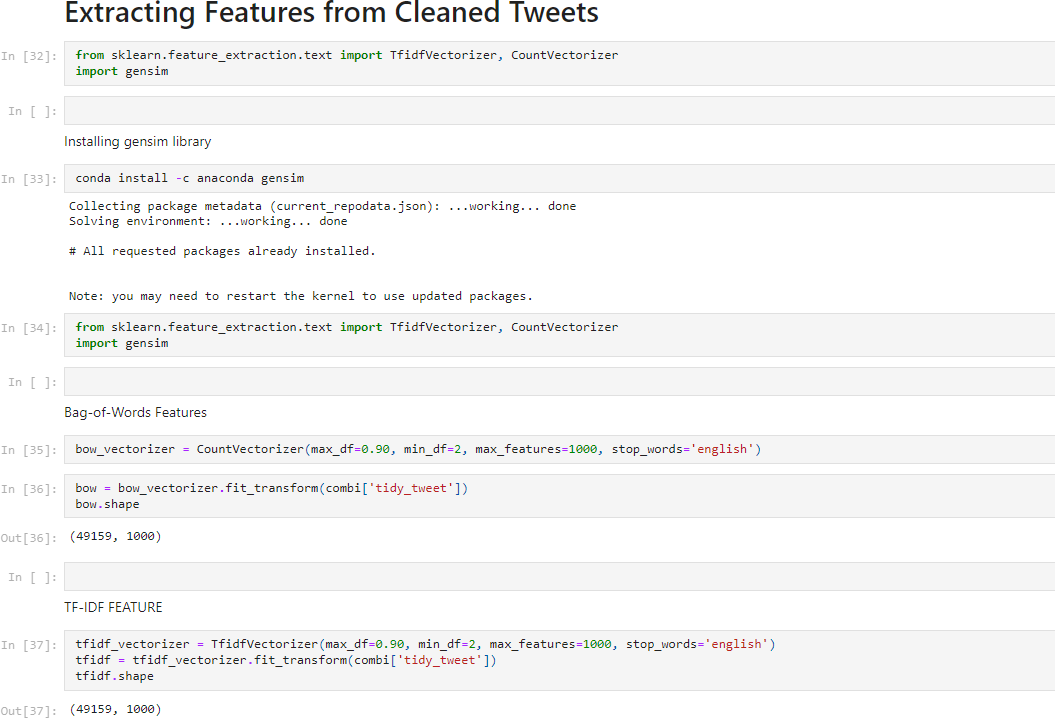


## Extracting features from cleaned tweets

To analyze a pre-processed data, it needs to be converted into features. We have used Bag-of-Words and TF-IDF in our project.

Bag-of-Words is a method to represent text into numerical features.

TF-IDF works by penalizing the common words by assigning them lower weights while giving importance to words which are rare in the entire corpus but appear in good numbers in few documents.



Model Building:-

We are now done with all the pre-modeling stages required to get the data in the proper form and shape. Now we will be building predictive models on the dataset using the two feature set — Bag-of-Words and TF-IDF.

We will use the following algorithms to build models:

-LOGISTIC REGRESSION

-SVM

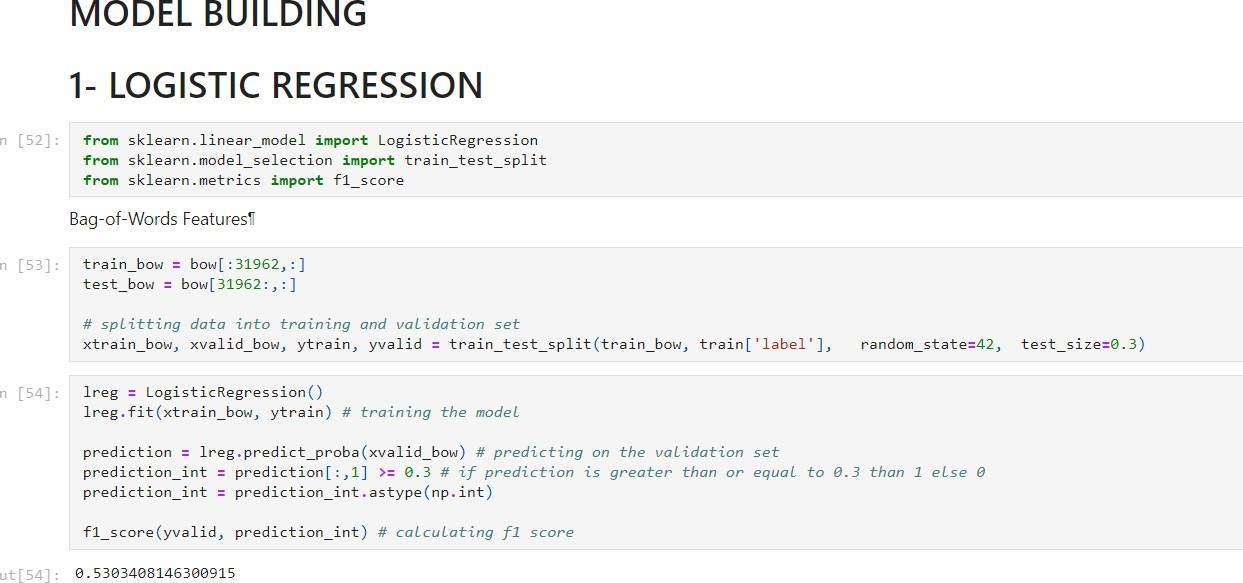
-RANDOMFOREST

-XGBOOST

Evaluation metrics:

F1 score is being used as evaluation metric. Its is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is suitable for uneven class distribution problems.

SS of model buildings and F1 scores for Logistic Regression:-



Predictions for test data

t est\_pred = lneg . predict\_pnoba (I es t\_bow ) I est\_pred\_int - I es t\_pred [ : , 1j › - 6. 3

I est\_pred\_int - I es t\_pred\_int . a stype (np . int)

I est[ ' l a ' z I est pred int

submi s s TOn - test [ [ ' d ' , ' la '

s ubmi s s ion .to\_c sv ( ' s ub\_1neg\_bo . c s v ' , indexzF aI se) # ‹vr'iti n@ *do la* to a CSA /T £ e

Leaderboard F1 Score: 0.567 TF-IDF Featuresl

I ra in\_tf1df - tfid’[: 31962, :

t est tfidf - tftdf [31962 : , :

xtrain\_tfidf : train\_tfidf[ytrain.index] xvaTid\_tfidf : train\_tfidf[yvalid.index]



prediction : lreg.predict\_proBa(xva1id\_tfidf)

predi cI 1on\_int - predi ction[ : , 1 ›- 6.3

predi cI 1on\_int - predi ct 1on\_int . a stype . int)



**8.5451327433628319**

Public Leaderboard F1 Score: 0.564

Word2Vec Features

train\_w]v : wordvec\_df.iloc[:)1962,:] test\_w2v : wordvec\_df.iloc[31962:,:]

xtrain\_w]v : train\_w2v.iloc[ytrain.index,:]

xva 1id\_w2v - tea ?n\_ w 2v . i 1oc [yvalid . i ndex, : ]

lreg.fit(xtrain w2v, ytrain)

prediction : lreg.predict\_proBa(xvalid\_wZv) prediction int: prediction :,l] ›: 0.3 prediction\_int: prediction\_int.astype(np.int) f1\_score(yvalid, prediction\_int)

0.6299183B56502249

Public Leaderboard F1 Score. 0.661 Doc2Vec Features

train\_d]v : docvec\_df.iloc[:31962,:]

I es t\_d2 v n docvec\_df . 11oc 31962 : , : ]

xtrain\_d]v : train\_d2v.iloc[ytrain.index,:] xvalid\_d]v : train\_d2v.iloc[yvalid.index,:]



prediction : lreg.predict\_proba(xvalid\_dZv) prediction\_int: prediction :,lj ›: 0.3 prediction\_int: prediction\_int.astype(np.int) f1\_score(yvalid, prediction\_int)

0.9698630186986301

Public Leaderboard F1 Score. 0.381

* 1. Support Vector Machine

In [61]:

I n [ 62 ] :

Out[62]:

In [63]:

from sklearn import svm

Bag-of-Words Features

svc: svm.SVC(kernel:'linear', C:1, probability:True).fit(xtrain bow, ytrain) prediction = svc.predict proba(xvalid bow)

prediction int = prediction[:,1] ›= 0.3 prediction int = prediction int.astype(np.int) fl score(yvalid, prediction int)

8. 58969529B8587258

Again let's ma e predictions for the test dataset and create another submission file.

test pred = svc.predict proba(test bow) test pred int = test pred[:,l] ›= 8.3

test\_pred\_int : test\_pred\_int.astype(np.int) test['label']: test pred int

submission : test[['id','label']]

submission.to csv('sub svc bow.csv', index=False)

Public Leaderboard F1 Score: 0.554

Here both validation score and leaderboard score are slightly lesser than the Logistic Regression scores for bag-of-words feature!

TF-IDF Features

svc = svrn. PVC ( <e *me i= '* 1i nea *r '* C =1, probabi11ty=True) .f1t(xtnain\_tfidf, yt*rain)*

Out[6z]:



pred i ct1on = svc . pnedlet proba(xvaIld tfid+) pred i ct1on 1nt = pred1ct1on[ : , 1] ›= 0.3

pred i ct1on 1nt = pred1ct1on 1nt . as type(np. Int)

f1 score(yvalid, prediction int)

0.51B0182149362478

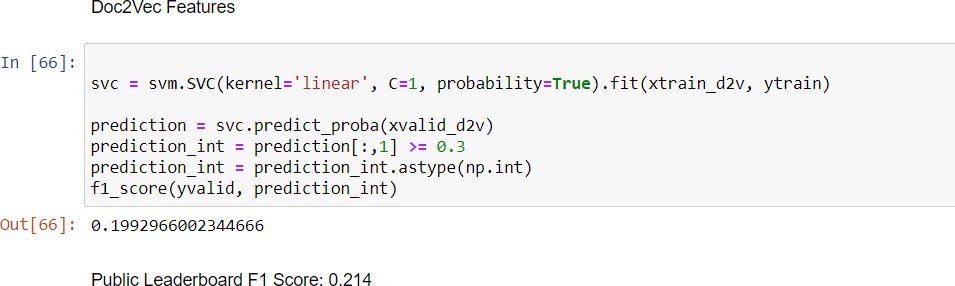
Public Leaderboard F1 Score: 0.546 Word2Vec Features

svc: svm.5VC( ernel:'linear', C:1, probability:True).fit(xtrain •2v, ytrain)

pred ict1on = svc . predict proba(xvat1d N2v) pred ict1on 1nt = pred1ct1on [ : , 1] ›= 0. 3

pred ict1on 1nt = pred1ct1on 1nt . astype(np. Int)

f1 score(yvalid, prediction int)



## What are Support Vector Machines?

Support Vector Machine (SVM) is a relatively simple **Supervised Machine Learning Algorithm** used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this

hyper-plane is nothing but a line.

In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems.

## Support Vector Machine for Multi-CLass Problems

To perform SVM on multi-class problems, we can create a binary classifier for each class of the data. The two results of each classifier will be :

* The data point belongs to that class OR
* The data point does not belong to that class.

For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the ‗mango‘ class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of the SVM.

* 1. RandomForest

In [67):





I n [ 69 ] :



from sklearn.ensemble import RandomForestClassifier

Bag-of-Words Features

rf = Random°orestClassifier(n\_estimato s=400, random state=l1).fit(xtrain\_bow, ytrain) prediction = r\*.peict(xvalid\_bou)

f1 scope(yvalid, prediction)

**8.552923B9B8373828**

Let's make predictions for the test dataset and create another submission file test\_p ed = n+. pred1ct(Iest\_bow)

test [ ' Gabe1 ' ] = test\_p ed

submi s sion = test [ [ ' id ' , ' label ’ ] ]

submi ssion. to\_csv ( ' r+ bo'.‹ . c sv ' , 1ndex=Fa1se)

Pu blic Leaderboard F1 Score: 0 598 TF-IDF Features

rf = Random°orestClassifier(n\_estimato s=400, random state=l1).fit(xtrain\_t\*idf, ytrain)

Public Leaderboard FI Score: 0 TF-IDF Features

In [70]:

0u- [ 70 ] :

In [71]:

Gu-[71:

In [72):

0u- [ 72 ] :

rf = RandomForestClassifier(n estimators=400, random state=11).fit(xtrain tfidf, ytrain)

prediction = rf.predict(xvalid tfidf)

\*1 sco°e (yva11d , pnedie I ion }

0.562152133580705

Public Leaderboard FI Score: 0 589 Word2Vec Features

rf = RandomForestClassifier(n estimators=400, random state=11).fit(xtrain w2v, ytrain)

prediction = rf.predict(xvalid w2v)

+1 scope(yva1Id , pnedict ion )

0.49134109134199125

Public Leaderboard FI Score: 0 549 Doc2Vec Features

Doc2Vec Features

rf = RandomForestClassifier(n estimators=400, random state=11).fit(xtrain d2v, ytrain)

pred1ction = rd . p nedlct (xvaLi d\_d2 v)

f1 sco e(yvalid, prediction)

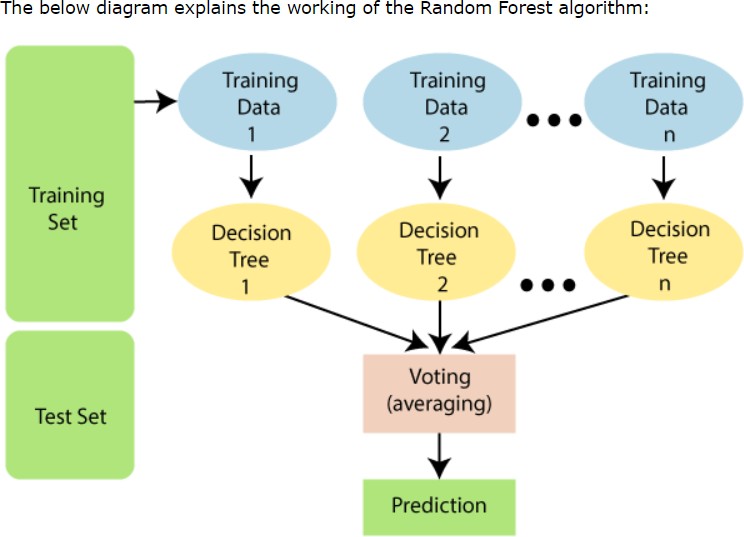
0.081232d92997l9887

Public Leaderboard F1 Score: 0 07

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

## The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



* 1. **XGBoost**

Iu [73]:

conda install -c conda-forge libxgboost

Collecting package metadata (current repodata.json): ...working... done Solving environment: ...working... done

## Package Plan ##

environment location: C:\Users\DELL\anaconda3

added / updated specs:

- libxgboost

The packages will be SUPERSEDED by a higher-priority channel:

conda pkgs/main::conda-4.10.1-py38haa9Z532 1 --› conda-{orge: :conda-4.10.1-py38haa244fe 0

Prepar1ng transaction: . . .working . . . done Ver1fy1ng transact1on: . . .working . . . done Executing transaction: ...working... done

Note: you iuay need to restart the kernel to use updated packages.

Iu [74a: from xgboost import XGBClassifier

Bag-of-Words Features Bag-of-Words Features

In [75 ] :

Out [TS ] :

In [76 ] :

In [77 j :

xgb model = XGBClassiTier(max depth=6, n estimators=1B00).fit(xtrain bow, ytrain) prediction = xgb model.predict(xvalid bow)

f1\_score(yvalid, prediction)

C:\Users\DELL\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning: The use of label encoder in recated and will be removed in a future release. To remve this warning, do the following: 1) Pass option se when constructing xsBclassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. lass - 1].

warnings.warn(label encoder deprecation\_msg, UserWarning)

[19:16:32] WARNING: ..\src\learner.cc:1061: Starting in xGBoost 1.3.0, the default evaluation metric used inary:logistic’ was changed from ’error’ to ’logloss’. Explicitly set eval metric if you’d like to restore

P. 52477B6422B18349

test pred = xgb mode1. predict (test bow)

test['label'] = test pred

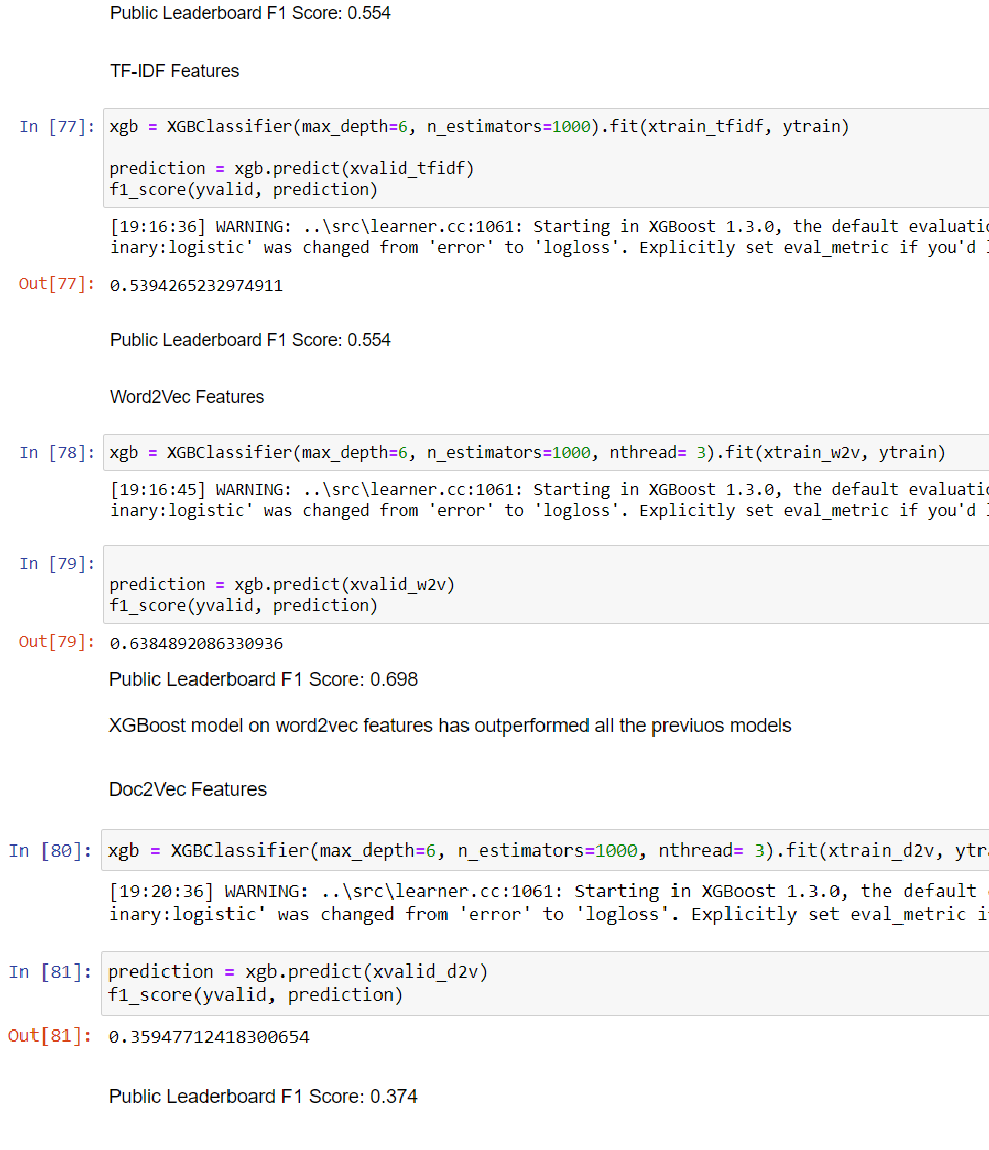
subm1s s1on = test [ [ ' 1d ' , ' 1abel ' j j

submiss1on.to csv( ’ sub xgb bow. csv' , 1ndex= False)

**Public Leaderboard F1 Score: 0.554**

TF-IDF Features

xgb = XGBClassifier(max\_depth=6, n estimators=1000).{it(xtrain tfidf, ytrain)



## Understanding XGBoost Algorithm

XGBoost stands for eXtreme Gradient Boosting. It became popular in the recent days and is dominating applied machine learning and Kaggle

competitions for structured data because of its scalability.

XGBoost is an extension to gradient boosted decision trees (GBM) and specially designed to improve speed and performance.

XGBoost is a faster algorithm when compared to other algorithms because of

its parallel and distributed computing. XGBoost is developed with both deep

considerations in terms of systems optimization and principles in machine

learning. The goal of this library is to push the extreme of the computation

limits of machines to provide a scalable, portable and accurate library.





In [ B6 ] :



def custom\_eval(preds, dtrain):

labels: dtraiu.get label().astype(up.iut) preds = (preds ›= 0.3).astype(np.int)

return [('f1 score', fl score(labels, preds))]

General Approach for Parameter Tuniug

Ne will follow the steps below to tune the parameters.

-Choose a relatively high learning rate. Usually a learning rate of 0.3 is used at this stage.

-Tune tree-specific parameters such as max depth, min child weight, subsample, colsample bytree keeping the learning fixed.

-Tune the learning rate.

-Finally tuue gamma to avoid overfitting.

Tuning max depth and min child weight

gridsearchgarams = [

(nax depth, m1n ch1ld weight)

for nax dept h in range (6, 18)

for n1n\_ch11d ue1ght 1n range (5, 8)



Tuning max depth and min child weight

In [ B6 ] :

gridsearchgarams = [

(max depth, min child weight)

for max depth in range(6,10)

for min\_child\_weight in range(5,8)

In [87] : nax f1 = 6. # *in iU a* t *izing* ivTth 8

best\_params = None

for max\_depth, min\_child\_weight in gridsearch\_params:

print(“CV with max depth={}, min child weight={}“.format(

nax dept h, In1n\_ch1\ d\_uie1ght ) )

*# tipdaze our paradeners*

params['max depth']: max depth params['min\_child\_weight’] = min\_child\_weight

# Cross - *vat i dat ion*

cv resu1ts = xgb . cv(

panaus,

dtraln,

feval= custom eval, nun boost round=266, max1r41ze=True, seed=16,

*# Finding best FI Score*

mean\_f1 = cv\_results['test-f1\_score-mean’].max()

boost\_nounds = c v\_result s [ ' test or-e nean ' ] . a *rgmax ()*

pr1nt( " \t F1 Sco°e (} fo° ( } rounds" . format(nean f1, boost rounds ) )

if meau f1 › max f1: max f1 = mean f1

best *params* = (max depth,min child weight)

print(”Best params: {}, {}, F1 Score: (}”.format(best\_params[0], best\_params[\*], max\_f1))

[19:24:38] WARNING: ..\src\learner.cc:1061: Starting in xGBoost 1.3.0, the default evaluation metric used with 'binary:logistic' was changed from 'error’ to ’logloss’. Explicitly set eval\_metric if you'd like to restore 1

F\* Score B.6414704 for 79 rounds CV with max depth=6, min child weight=6

[19:25:51] WARNING: ..\src\learner.cc:1061: Starting iu XGBoost 1.3.0, the default evaluation metric used with 'biuary:logistic' was changed from *'error’* to ’logloss’. Explicitly set eval metric if you'd like to restore 1

[19:Z5:51] WARwINs: ..\src\learner.cc:1061: Starting iu XGBoost 1.3.0, the default evaluation metric used

'binary:logistic' was changed from 'error’ to ’logloss’. Explicitly set if you’d like to restore 1

[19:Z5:51] WARwING: ..\src\learner.cc:1061: Starting in xGBoost 1.3.0, the default evaluation metric used with 'binary:logistic' was changed from 'error’ to ’logloss’. Explicitly set eva1\_metric if you’d like to restore 1

[z9:z5:51] WARNING: ..\src\learner.cc:1B6l: Starting in xGBoost 1.3.0, the default evaluation metric used with 'binary:logistic' was changed from 'error’ to ’logloss’. Explicitly set eval\_metric if you'd like to restore 1

[19:25:51] WARNING: ..\src\learner.cc:1061: Starting iu XGBoost 1.3.0, the default evaluation metric used with

Updatinq max depth and min child weiqht parameters.

Updating max depth and min child weight parameters.

In [88]:

params [ ’ max depth ' ] = 8

params [ ’ o1n ch1Id we1ght ' ] 6

Tuning subsample and colsample.

In [89] : gr1dsearch aranis = [

( subsample, cot sanple)

for subsample 1n [i/TO. for 1 1n range(5,10) ] for cot sample 1n [i/TO. for 1 1n range(5,10) ]

In [90]: max f1 = 0.

best params = None

for subsample, colsample in gridsearch params: print(”CV with subsample={}b col ample={}”.format(

lsample))

# Updote our Porometers params['colsample'] = colsample params['subsample'] = subsample

cv results = xgb.cv(

params, dtrain,

feval= custom eval,

Updabng subsample and colsample bytree.

In [91]:

params[’subsample'] = .9

params[’colsanple bytree'] .5

Now let's 1une the learning rate.

In [92] : nax II = e.

best\_params: None

for eta in [.3, .2, .1, .05, .01, .005]:

print(”CV with eta=(}”.format(eta))

*# uPda te ETA*

parans['eta’] = eta

# Run CV

cv res ults = xgb. cv ( pagans,

dtrain,

feval= custom\_eval, num boost round=l000, naximize=True, seed=16,

nfold=5,

Now lets tune gamma value using the parameters already tuned above. We’ll check for 5 values here.

In [94]: max f1 = 0.

best params = None

for gamma in range(0,15):

print(“CV with gamma=(}”.format(gamma/10.))

*# uPda te ETA*

params['gamma'] = gamma/10.

cv results = xgb.cv( params,

dtrain,

feval= custom eval, num boost round=200, maximize:True, seed=16,

nfold=5,

early stopping rounds=10

# *Finds ng best FT Score*

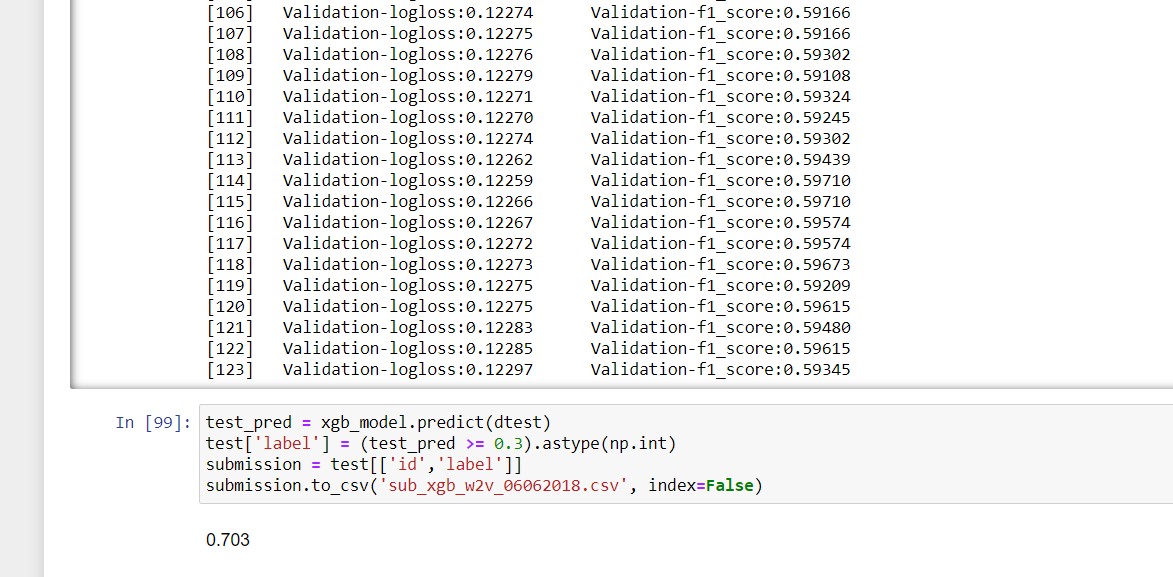
mean f1 = cv results['test-f1 score-mean’].max()

boost **rounds** = cv *result s (* ' test —f1 sc ore -mean ' ] . ar gniax ( ) print(“\tF1 Score (} for {} rounds”.format(mean f1, boost rounds)) if mean f1 › max {1:

max f1 = mean fl







# Results and Discussion:-

So we tried different models and got different F1 scores. Among all the models XGboost outperformed all previous models. Then after model finetuning we get F1 score of >0.700 which is our best score on public leaderboard.

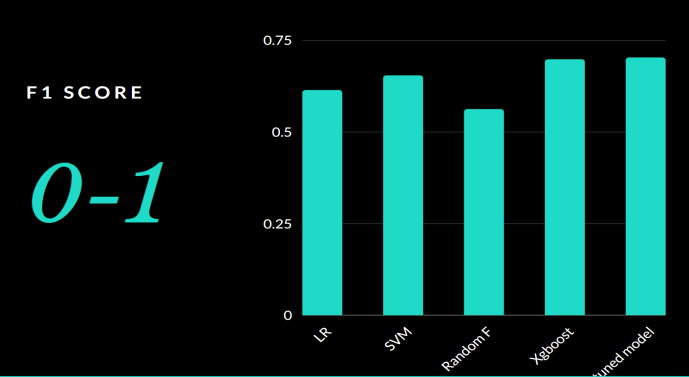
In Logistic Regression we get the best score with Word2Vec feature

of 0.614 score.

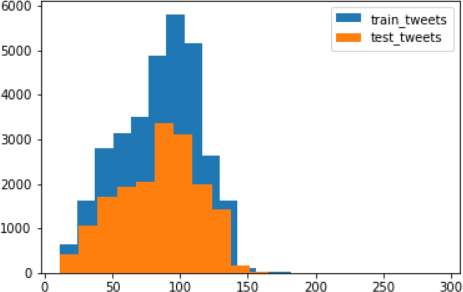
In SVM we get the best score with with Word2Vec feature of 0.654 score.

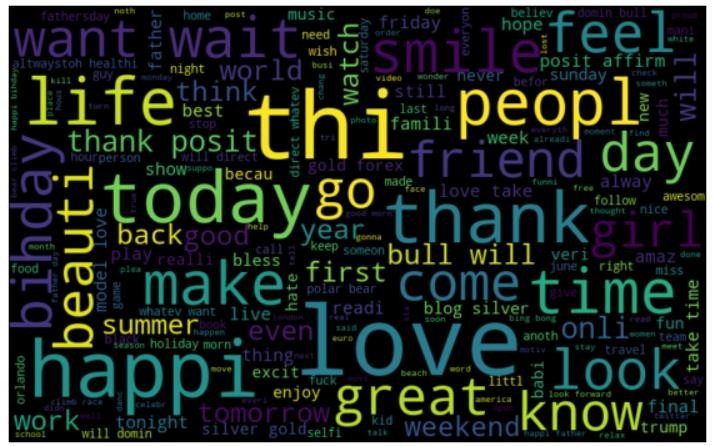
In RandomForest we got 0.562 with TD-IDF. In XGboost we got a score of 0.698.

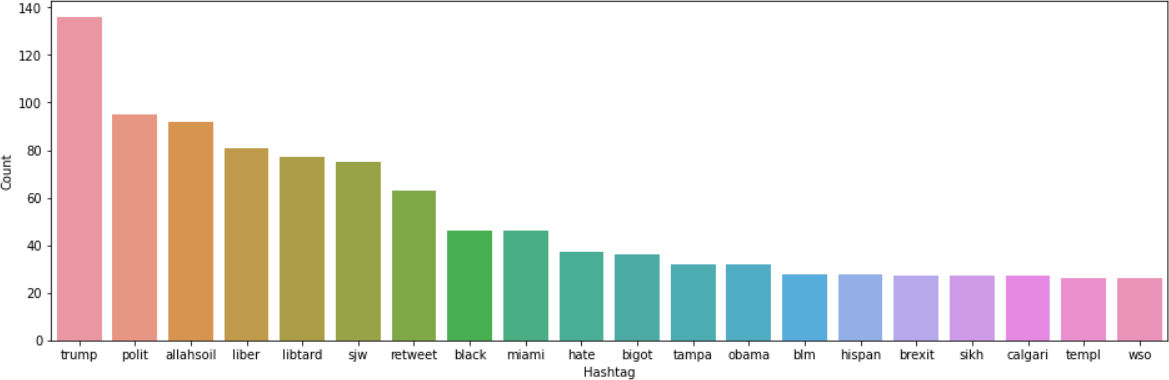
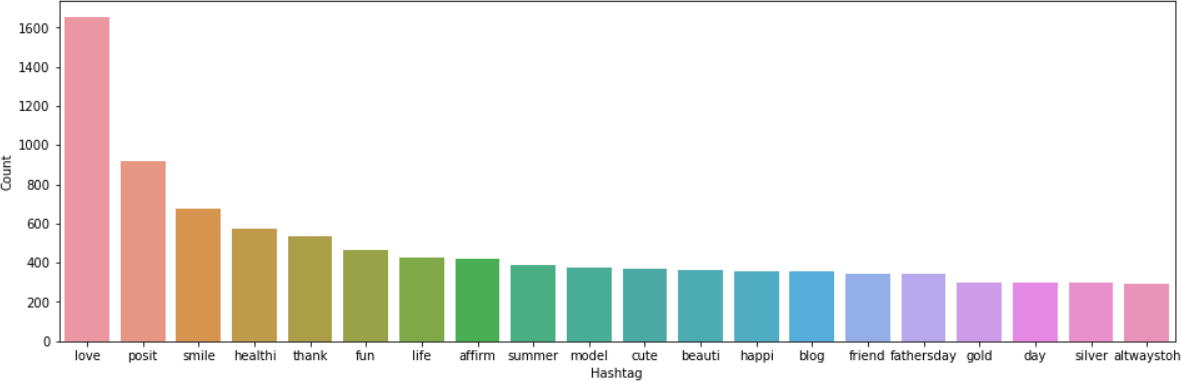
After Model Finetuning we got a score of 0.703 .



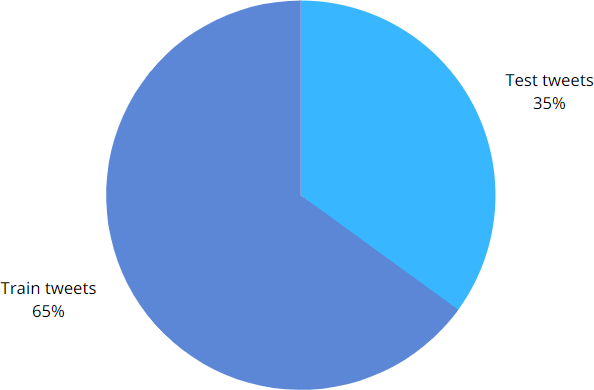
Graphs and wordclouds visualization:-

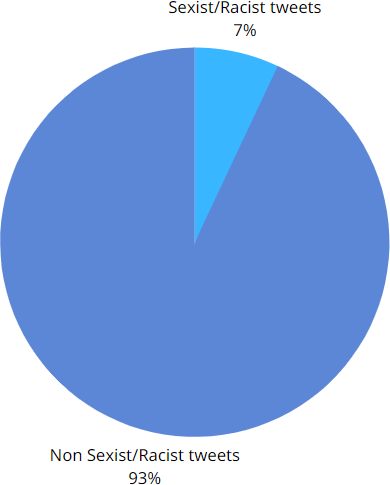






Dataset Division:





# Conclusion:-

So, now wrap up things quickly. We started with preprocessing and exploration of data. We cleaned the tweets and removed the unwanted characters and symbols making the tweets clean, short and tidy. Later we used hashtags for getting information about the

sentiments of tweets. Then we extracted features from the cleaned text using Bag-of-Words and TF-IDF. Finally, we were able to build a couple of models using both the feature sets to classify the tweets. Later we did finetuning of Xgboost to enhance the accuracy of the model which was around 0.7