

**BaseLine Modelling**

# 1.Importing Libraries

In [1]:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import os
6 from datetime import datetime
7 from dateutil.parser import parse
8 import pickle
9 %matplotlib inline
10 import warnings
11 warnings.filterwarnings("ignore")
12
13 from sklearn.feature_extraction.text import TfidfVectorizer
14 from sklearn.feature_extraction.text import CountVectorizer
15 from sklearn.model_selection import GridSearchCV
16 from scipy.stats import randint as sp_randint
17 from sklearn.model_selection import RandomizedSearchCV
18 from sklearn.naive_bayes import MultinomialNB
19 from sklearn.model_selection import train_test_split
20 import re
21 from nltk.corpus import stopwords
22 from nltk.stem.snowball import SnowballStemmer
23 from bs4 import BeautifulSoup
24 import pickle
25 from scipy.sparse import hstack
26 from tqdm import tqdm
27 import os
28 import pickle
29 from scipy.sparse import csr_matrix
30 from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, f1_score, recall_score
31 from sklearn.ensemble import RandomForestClassifier
32 from sklearn.preprocessing import Normalizer
33 from sklearn.preprocessing import MinMaxScaler
34 from sklearn.linear_model import LogisticRegression
35 from sklearn.svm import SVC
36 from sklearn.ensemble import RandomForestClassifier
37 from xgboost import XGBClassifier
38 from lightgbm import LGBMClassifier
39 import tensorflow as tf
40 from tensorflow.keras.models import Sequential
41 from tensorflow.keras.layers import Dense
42 from tensorflow.keras import Input, Model
43 from tensorflow.keras.optimizers import Adam
44 from tensorflow.keras.callbacks import EarlyStopping
45 from tensorflow.keras.layers import Dense, Concatenate, Activation, Flatten
46 from tensorflow.keras.layers import BatchNormalization, Dropout, LSTM
47 from tensorflow.keras.layers import Concatenate
48 from tensorflow.keras.layers import Embedding, Conv1D
49 from tensorflow.keras.preprocessing.text import Tokenizer
50 from tensorflow.keras.preprocessing.sequence import pad_sequences
51 from tensorflow.keras.utils import plot_model
52 from tensorflow.keras.callbacks import ModelCheckpoint
53 from sklearn.metrics import log_loss
54 %load_ext tensorboard
55 import datetime
```

## 2.Importing Preprocessed data

In [4]:

```
1 with open('/content/drive/MyDrive/LSTM/featured_data','rb') as f:
2     data=pickle.load(f)
```

In [ ]:

```
1 data.shape
```

Out[6]:

(118698, 17)

In [5]:

```
1 data.head(2)
```

Out[5]:

	ReviewTitle	CompleteReview	URL	Rating	ReviewDetails	score
0	Good Company for Every employee	Good company for every Engineers dream, Full M...	https://in.indeed.com/cmp/Reliance-Industries-...	5	(Former Employee) -- August 17, 2021	1
1	Productive	I am just pass out bsc in chemistry Typical da...	https://in.indeed.com/cmp/Reliance-Industries-...	5	(Current Employee) -- August 17, 2021	1

In [28]:

```
1 data[data['score']==0]['ReviewTitle'].iloc[1]
```

Out[28]:

'Not good'

The following are the features which we can drop at this stage from EDA.

URI, Rating, ReviewDetails, Company names, Employee Type, location, year, date.

In [ ]:

```
1 data.drop(['URL','Rating','ReviewDetails','Company_name','Employee_type','location','Ye
```

In [ ]:

1 data.head(2)

Out[5]:

	ReviewTitle	CompleteReview	score	compound	neg	neu	pos	polarity	subjectivity
0	Good Company for Every employee	Good company for every Engineers dream, Full M...	1	0.8625	0.0	0.654	0.346	0.483333	0.65
1	Productive	I am just pass out bsc in chemistry Typical da...	1	0.7264	0.0	0.804	0.196	0.406667	0.54

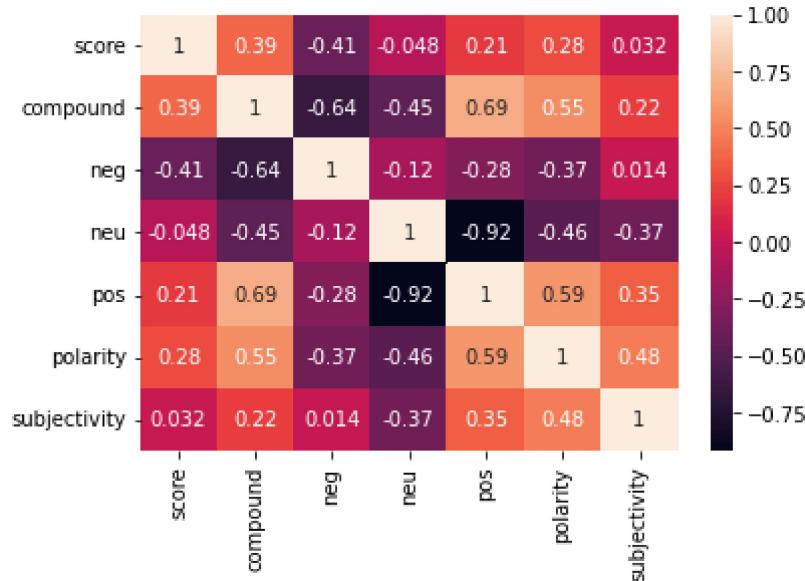
[ ]

In [ ]:

1 sns.heatmap(data.drop(['ReviewTitle', 'CompleteReview'], axis=1).corr(), annot=True)

Out[10]:

&lt;matplotlib.axes.\_subplots.AxesSubplot at 0x7fb9905ed5d0&gt;



From above we can see that neu and pos have a high correlation. We can drop one of the feature.

Dropping neu feature

In [ ]:

1 data.drop('neu', axis=1, inplace=True)  
2

## 3.Text Preprocessing

Concatinating both text columns into one

In [ ]:

```
1 data['Review']=data['ReviewTitle']+ " "+data['CompleteReview']
2
```

In [ ]:

```
1 data.drop(['ReviewTitle','CompleteReview'],axis=1,inplace=True)
```

In [ ]:

```

1 import re
2
3 def decontracted(phrase):
4     # specific
5     phrase = re.sub(r"won't", "will not", phrase)
6     phrase = re.sub(r"can't", "can not", phrase)
7
8     # general
9     phrase = re.sub(r"\n\t", " not", phrase)
10    phrase = re.sub(r"\re", " are", phrase)
11    phrase = re.sub(r"\s", " is", phrase)
12    phrase = re.sub(r"\d", " would", phrase)
13    phrase = re.sub(r"\ll", " will", phrase)
14    phrase = re.sub(r"\t", " not", phrase)
15    phrase = re.sub(r"\ve", " have", phrase)
16    phrase = re.sub(r"\m", " am", phrase)
17    return phrase
18
19 # we are removing the words from the stop words list: 'no', 'nor', 'not'
20 stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
21             "you'll", "you'd", "your", 'yours', 'yourself', 'yourselves', 'he', 'him',
22             'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'th',
23             'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'th',
24             'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha',
25             'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
26             'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
27             'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov',
28             'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
29             'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too',
30             's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no',
31             've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
32             "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh',
33             "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w',
34             'won', "won't", 'wouldn', "wouldn't"]
35
36 def preprocess_text(text_data):
37     #stemmer=SnowballStemmer(Language="english")
38     preprocessed_text = []
39     # tqdm is for printing the status bar
40     for sentance in tqdm(text_data):
41         sentance = re.sub(r"http\S+", "", sentance)
42         sentance = BeautifulSoup(sentance, 'lxml').get_text()
43         sentance = re.sub("\S*\d\S*", "", sentance).strip()
44         sent = decontracted(sentance)
45         sent = sent.replace('\r', ' ')
46         sent = sent.replace('\n', ' ')
47         sent = sent.replace('\'', ' ')
48         sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
49         # https://gist.github.com/sebleier/554280
50         sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
51         sent=sent.lower().strip()
52         #sent=".join(stemmer.stem(e) for e in sent.split())"
53         preprocessed_text.append(sent)
54     return preprocessed_text
55

```

In [ ]:

```
1 data["process_review"] = preprocess_text(data["Review"].values)
```

100%|██████████| 118698/118698 [00:51<00:00, 2322.71it/s]

In [ ]:

1 data

Out[10]:

	score	compound	neg	pos	polarity	subjectivity	Review	process_review
0	1	0.8625	0.000	0.346	0.483333	0.650000	Good Company for Every employee Good company f...	good company every employee good company every...
1	1	0.7264	0.000	0.196	0.406667	0.540000	Productive I am just pass out bsc in chemistry...	productive pass bsc chemistry typical day work...
2	0	-0.9008	0.369	0.000	0.033333	0.166667	Non productive Not so fun at work just blame g...	non productive not fun work blame games target...
3	1	0.4927	0.000	0.106	0.450000	0.550000	Work Place at kunool Full time work Management...	work place kunool full time work management fe...
4	0	0.8947	0.092	0.374	-0.175000	0.450000	Not good Work culture not good and no benefits...	not good work culture not good no benefits wou...
...	...	...	...	...	...	...	...	...
118693	1	0.9423	0.000	0.453	0.833333	0.550000	excellent employer to work with Was a pleasure...	excellent employer work pleasure working great...
118694	1	0.0258	0.124	0.105	0.141667	0.428333	Definitely very good place to work and can hav...	definitely good place work lots learning get l...
118695	1	0.0000	0.000	0.000	0.230159	0.476984	IT Services Company; Great scope for improveme...	services company great scope improvement lot s...
118696	1	0.9422	0.000	0.362	0.421212	0.632576	Productive, fun to work, great place to do cer...	productive fun work great place certification ...
118697	1	0.6705	0.000	0.314	0.700000	0.600000	Nice place to work Got good experience and kno...	nice place work got good experience knowledge ...

118698 rows × 8 columns



In [ ]:

```
1 data.drop('Review',axis=1,inplace=True)
```

In [ ]:

```
1 drop_features=['URL','Rating','ReviewDetails','Company_name','Employee_type','location']
2 with open('drop_features','wb') as f:
3     pickle.dump(drop_features,f)
```

## Train Test Split

In [ ]:

```
1 X=data.drop('score',axis=1)
2 y=data['score']
3 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,stratify=y,random_state=42)
```

In [ ]:

```
1 print("Shape of X_train is",X_train.shape)
2 print("Shape of X_test is ",X_test.shape)
3 print("Shape of y_train is",y_train.shape)
4 print("Shape of y_test is",y_test.shape)
```

Shape of X\_train is (94958, 6)

Shape of X\_test is (23740, 6)

Shape of y\_train is (94958,)

Shape of y\_test is (23740,)

In [ ]:

```
1 X_train.reset_index(drop=True,inplace=True)
2 y_train.reset_index(drop=True,inplace=True)
3 X_test.reset_index(drop=True,inplace=True)
4 y_test.reset_index(drop=True,inplace=True)
```

In [ ]:

```
1 from sklearn.preprocessing import OneHotEncoder
2 encoder=OneHotEncoder(sparse=False)
3 y_train_encoded=encoder.fit_transform(y_train.values.reshape(-1,1))
4 y_test_encoded=encoder.transform(y_test.values.reshape(-1,1))
```

## Normalizing Numerical Features

In [ ]:

```
1 normalizer = MinMaxScaler()
2 #normalizer.fit(X_train['compound'].values.reshape(1,-1))
3 X_train_compound_norm = normalizer.fit_transform(X_train['compound'].values.reshape(-1,1))
4 X_test_compound_norm = normalizer.transform(X_test['compound'].values.reshape(-1,1))
5
6 normalizer = MinMaxScaler()
7 #normalizer.fit(X_train['neg'].values.reshape(1,-1))
8 X_train_neg_norm = normalizer.fit_transform(X_train['neg'].values.reshape(-1,1))
9 X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(-1,1))
10
11 normalizer = MinMaxScaler()
12 #normalizer.fit(X_train['pos'].values.reshape(1,-1))
13 X_train_pos_norm = normalizer.fit_transform(X_train['pos'].values.reshape(-1,1))
14 X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(-1,1))
15
16 normalizer = MinMaxScaler()
17 #normalizer.fit(X_train['polarity'].values.reshape(1,-1))
18 X_train_polarity_norm = normalizer.fit_transform(X_train['polarity'].values.reshape(-1,1))
19 X_test_polarity_norm = normalizer.transform(X_test['polarity'].values.reshape(-1,1))
20
21
22 normalizer = MinMaxScaler()
23 #normalizer.fit(X_train['subjectivity'].values.reshape(1,-1))
24 X_train_subjectivity_norm = normalizer.fit_transform(X_train['subjectivity'].values.reshape(-1,1))
25 X_test_subjectivity_norm = normalizer.transform(X_test['subjectivity'].values.reshape(-1,1))
26
27
```

## Standard Scaler

In [ ]:

```

1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
3 X_train_compound_norm = sc.fit_transform(X_train['compound'].values.reshape(-1,1))
4 X_test_compound_norm = sc.transform(X_test['compound'].values.reshape(-1,1))
5
6 sc = StandardScaler()
7 X_train_neg_norm = sc.fit_transform(X_train['neg'].values.reshape(-1,1))
8 X_test_neg_norm = sc.transform(X_test['neg'].values.reshape(-1,1))
9
10 sc = StandardScaler()
11 X_train_pos_norm = sc.fit_transform(X_train['pos'].values.reshape(-1,1))
12 X_test_pos_norm = sc.transform(X_test['pos'].values.reshape(-1,1))
13
14 sc = StandardScaler()
15 X_train_polarity_norm = sc.fit_transform(X_train['polarity'].values.reshape(-1,1))
16 X_test_polarity_norm = sc.transform(X_test['polarity'].values.reshape(-1,1))
17
18
19 sc = StandardScaler()
20 X_train_subjectivity_norm = sc.fit_transform(X_train['subjectivity'].values.reshape(-1,1))
21 X_test_subjectivity_norm = sc.transform(X_test['subjectivity'].values.reshape(-1,1))
22
23 X_train_numerical=pd.DataFrame({'compound':X_train_compound_norm.ravel(),'neg':X_train_
24 'polarity':X_train_polarity_norm.ravel(),'subjectivity':X_train_
25 'subjectivity_norm':X_train_subjectivity_norm.ravel()})
26 X_test_numerical=pd.DataFrame({'compound':X_test_compound_norm.ravel(),'neg':X_test_
27 'polarity':X_test_polarity_norm.ravel(),'subjectivity':X_test_
28 'subjectivity_norm':X_test_subjectivity_norm.ravel()})
29 X_train_numeric_concatenate=X_train_numerical[['compound','neg','pos','polarity','subje
30 X_test_numeric_concatenate=X_test_numerical[['compound','neg','pos','polarity','subje

```

## 4.Text to Vector

### 4.1 BOW

In [ ]:

```

1 # We are considering only the words which appeared in at least 10 documents(rows or pro
2 vectorizer=CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
3 X_train_bow=vectorizer.fit_transform(X_train['process_review'].values)
4 X_test_bow=vectorizer.transform(X_test['process_review'].values)
5 print(X_train_bow.shape,y_train.shape)
6 print(X_test_bow.shape,y_test.shape)

```

```
(94958, 5000) (94958,)
(23740, 5000) (23740,)
```

In [ ]:

```

1 with open("vectorizer","wb") as f:
2     pickle.dump(vectorizer,f)

```

In [ ]:

```
1 with open("vectorizer","rb") as f:
2     vect1=pickle.load(f)
```

In [ ]:

```
1 X_testing=vect1.transform(X_test['process_review'].values)
```

In [ ]:

```
1 feature_bow=[]
2 feature_bow.extend(vectorizer.get_feature_names())
3 feature_bow.extend(list(X_train.drop('process_review',axis=1).columns))
```

In [ ]:

```
1 X_tr_bow = hstack((X_train_bow,X_train_compound_norm,X_train_neg_norm,X_train_pos_norm,
2 X_te_bow = hstack((X_test_bow,X_test_compound_norm,X_test_neg_norm,X_test_pos_norm,X_te
3
4 print("Final Data matrix")
5 print(X_tr_bow.shape, y_train.shape)
6 print(X_te_bow.shape, y_test.shape)
7 print("=*100)
```

Final Data matrix  
(94958, 5005) (94958,)  
(23740, 5005) (23740,)  
=====

In [ ]:

```
1 X_tr_bow1 = hstack((X_train_bow, X_train.drop('process_review',axis=1).values)).tocsr()
2 X_te_bow1= hstack((X_test_bow,X_test.drop('process_review',axis=1).values)).tocsr()
3
4 print("Final Data matrix")
5 print(X_tr_bow1.shape, y_train.shape)
6 print(X_te_bow1.shape, y_test.shape)
7 print("=*100)
```

Final Data matrix  
(94958, 5005) (94958,)  
(23740, 5005) (23740,)  
=====

## 4.2 TFIDF

In [ ]:

```

1 # We are considering only the words which appeared in at least 10 documents(rows or pro
2 vectorizer=TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
3 X_train_tfidf=vectorizer.fit_transform(X_train['process_review'].values)
4 X_test_tfidf=vectorizer.transform(X_test['process_review'].values)
5 print(X_train_tfidf.shape,y_train.shape)
6 print(X_test_tfidf.shape,y_test.shape)

```

(94958, 5000) (94958,)  
(23740, 5000) (23740,)

In [ ]:

```

1 feature_tfidf=[]
2 feature_tfidf.extend(vectorizer.get_feature_names())
3 feature_tfidf.extend(list(X_train.drop('process_review',axis=1).columns))
4
5 X_tr_tfidf = hstack((X_train_tfidf,X_train_compound_norm,X_train_neg_norm,X_train_pos_norm))
6 X_te_tfidf = hstack((X_test_tfidf,X_test_compound_norm,X_test_neg_norm,X_test_pos_norm))
7
8 print("Final Data matrix")
9 print(X_tr_tfidf.shape, y_train.shape)
10 print(X_te_tfidf.shape, y_test.shape)
11 print("=*100")

```

Final Data matrix  
(94958, 5005) (94958,)  
(23740, 5005) (23740,)

---



---

In [ ]:

```

1 feature_tfidf=[]
2 feature_tfidf.extend(vectorizer.get_feature_names())
3 feature_tfidf.extend(list(X_train.drop('process_review',axis=1).columns))
4
5 X_tr_tfidf1 = hstack((X_train_tfidf, X_train.drop('process_review',axis=1).values)).tocsr()
6 X_te_tfidf1 = hstack((X_test_tfidf,X_test.drop('process_review',axis=1).values)).tocsr()
7
8 print("Final Data matrix")
9 print(X_tr_tfidf1.shape, y_train.shape)
10 print(X_te_tfidf1.shape, y_test.shape)
11 print("=*100")

```

Final Data matrix  
(94958, 5005) (94958,)  
(23740, 5005) (23740,)

---



---

In [ ]:

```

1 with open("tfidf","wb") as f:
2     pickle.dump([X_tr_tfidf,X_te_tfidf,y_train,y_test,feature_tfidf],f)

```

## 4.3 Avg W2V

In [ ]:

```

1 with open('/content/drive/MyDrive/LSTM/glove_vectors', 'rb') as f:
2     model = pickle.load(f)
3     glove_words = set(model.keys())

```

In [ ]:

```

1 # average Word2Vec
2 # compute average word2vec for each review.
3 avg_w2v_vectors_train = [] # the avg-w2v for each sentence/review is stored in this list
4 for sentence in tqdm(X_train['process_review'].values): # for each review/sentence
5     vector = np.zeros(300) # as word vectors are of zero length
6     cnt_words = 0; # num of words with a valid vector in the sentence/review
7     for word in sentence.split(): # for each word in a review/sentence
8         if word in glove_words:
9             vector += model[word]
10        cnt_words += 1
11    if cnt_words != 0:
12        vector /= cnt_words
13    avg_w2v_vectors_train.append(vector)
14
15 print(len(avg_w2v_vectors_train))
16 print(len(avg_w2v_vectors_train[0]))

```

100% | 94958/94958 [00:06&lt;00:00, 14744.15it/s]

94958

300

In [ ]:

```

1 # average Word2Vec
2 # compute average word2vec for each review.
3 avg_w2v_vectors_test = [] # the avg-w2v for each sentence/review is stored in this list
4 for sentence in tqdm(X_test['process_review'].values): # for each review/sentence
5     vector = np.zeros(300) # as word vectors are of zero length
6     cnt_words = 0; # num of words with a valid vector in the sentence/review
7     for word in sentence.split(): # for each word in a review/sentence
8         if word in glove_words:
9             vector += model[word]
10        cnt_words += 1
11    if cnt_words != 0:
12        vector /= cnt_words
13    avg_w2v_vectors_test.append(vector)
14
15 print(len(avg_w2v_vectors_test))
16 print(len(avg_w2v_vectors_test[0]))

```

100% | 23740/23740 [00:01&lt;00:00, 16173.68it/s]

23740

300

In [ ]:

```
1 avg_w2v_vectors_train=np.array(avg_w2v_vectors_train)
2 avg_w2v_vectors_test=np.array(avg_w2v_vectors_test)
```

In [ ]:

```
1 avg_w2v_vectors_train
```

Out[49]:

```
<94958x5005 sparse matrix of type '<class 'numpy.float64'>'  
with 3076667 stored elements in Compressed Sparse Row format>
```

In [ ]:

```
1 X_tr_avgw2v = hstack((csr_matrix(avg_w2v_vectors_train),X_train_compound_norm,X_train_r  
2 X_te_avgw2v = hstack((csr_matrix(avg_w2v_vectors_test),X_test_compound_norm,X_test_neg  
3  
4 print("Final Data matrix")
5 print(X_tr_avgw2v.shape, y_train.shape)
6 print(X_te_avgw2v.shape, y_test.shape)
7 print("=*100)
```

```
Final Data matrix
(94958, 305) (94958,)
(23740, 305) (23740,)  
=====
```

## 4.4 TFIDF W2V

In [ ]:

```
1 tfidf_model = TfidfVectorizer()
2 tfidf_model.fit(X_train['process_review'].values)
3 # we are converting a dictionary with word as a key, and the idf as a value
4 dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
5 tfidf_words = set(tfidf_model.get_feature_names())
```

In [ ]:

```

1 # average Word2Vec
2 # compute average word2vec for each review.
3 tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this
4 for sentence in tqdm(X_train['process_review'].values): # for each review/sentence
5     vector = np.zeros(300) # as word vectors are of zero Length
6     tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
7     for word in sentence.split(): # for each word in a review/sentence
8         if (word in glove_words) and (word in tfidf_words):
9             vec = model[word] # getting the vector for each word
10            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence
11            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())))
12            vector += (vec * tf_idf) # calculating tfidf weighted w2v
13            tf_idf_weight += tf_idf
14        if tf_idf_weight != 0:
15            vector /= tf_idf_weight
16        tfidf_w2v_vectors_train.append(vector)
17
18 print(len(tfidf_w2v_vectors_train))
19 print(len(tfidf_w2v_vectors_train[0]))

```

100% |██████████| 94958/94958 [00:20&lt;00:00, 4533.28it/s]

94958

300

In [ ]:

```

1 # average Word2Vec
2 # compute average word2vec for each review.
3 tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this
4 for sentence in tqdm(X_test['process_review'].values): # for each review/sentence
5     vector = np.zeros(300) # as word vectors are of zero Length
6     tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
7     for word in sentence.split(): # for each word in a review/sentence
8         if (word in glove_words) and (word in tfidf_words):
9             vec = model[word] # getting the vector for each word
10            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence
11            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())))
12            vector += (vec * tf_idf) # calculating tfidf weighted w2v
13            tf_idf_weight += tf_idf
14        if tf_idf_weight != 0:
15            vector /= tf_idf_weight
16        tfidf_w2v_vectors_test.append(vector)
17
18 print(len(tfidf_w2v_vectors_test))
19 print(len(tfidf_w2v_vectors_test[0]))

```

100% |██████████| 23740/23740 [00:05&lt;00:00, 4147.91it/s]

23740

300

In [ ]:

```

1 tfidf_w2v_vectors_train=np.array(tfidf_w2v_vectors_train)
2 tfidf_w2v_vectors_test=np.array(tfidf_w2v_vectors_test)

```

In [ ]:

```

1 X_tr_tfidf2v = hstack((csr_matrix(tfidf_w2v_vectors_train), X_train_compound_norm,X_tr
2 X_te_tfidf2v = hstack((csr_matrix(tfidf_w2v_vectors_test),X_test_compound_norm,,X_te
3
4 print("Final Data matrix")
5 print(X_tr_avgw2v.shape, y_train.shape)
6 print(X_te_avgw2v.shape, y_test.shape)
7 print("=*100")

```

## 4.5 Tokenizer

In [ ]:

```

1 token=Tokenizer()
2 token.fit_on_texts(X_train['process_review'])
3 X_train_text_token=token.texts_to_sequences(X_train['process_review'])
4 X_test_text_token=token.texts_to_sequences(X_test['process_review'])

```

In [ ]:

```

1 # analyzing length of each vector
2 length=[]
3 for i in range(X_train.shape[0]):
4     length.append(len(X_train_text_token[i]))
5 length.sort()
6 print("Maximum sequence length is ",max(length))

```

Maximum sequence length is 183

In [ ]:

```

1 vocab_size_text=max(token.word_index.values())+1
2 X_train_text_padded=pad_sequences(X_train_text_token,maxlen=183,padding='post')
3 X_test_text_padded=pad_sequences(X_test_text_token,maxlen=183,padding='post')

```

## 5. BaseLine Modelling

### 5.1 Naive Bayes

#### 5.1.1 BOW

In [ ]:

```

1 #Finding best alpha
2 classifier=MultinomialNB(class_prior=[0.5,0.5])
3 parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,1
4 clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=True)
5 clf.fit(X_tr_bow, y_train)
6

```

Out[56]:

```
GridSearchCV(cv=3, estimator=MultinomialNB(class_prior=[0.5, 0.5]),
            param_grid={'alpha': [1e-05, 0.0005, 0.0001, 0.005, 0.001, 0.0
5,
                           0.01, 0.1, 0.5, 1, 5, 10, 50, 100]}, return_train_score=True, scoring='roc_auc')
```

In [ ]:

```

1 #Printing Best AUC score and best alpha
2 print("The best alpha value is ",clf.best_params_)
3 print("The best AUC score is ",clf.best_score_)
4
5

```

The best alpha value is {'alpha': 0.5}  
 The best AUC score is 0.8772243467108282

In [ ]:

```

1 classifier_bow= MultinomialNB(alpha=clf.best_params_['alpha'],class_prior=[0.5,0.5])
2 classifier_bow.fit(X_tr_bow, y_train)
3 y_pred=classifier_bow.predict(X_te_bow)

```

In [ ]:

```

1 print("Accuracy for BOW Data for Naive Bayes",accuracy_score(y_test,y_pred))
2 print("ROC AUC score for BOW Data for Naive Bayes",roc_auc_score(y_test,y_pred))
3 print("f1 score for BOW Data for Naive Bayes",f1_score(y_test,y_pred))
4 print("log loss for Bow data for Naive Bayes",log_loss(y_test,y_pred))

```

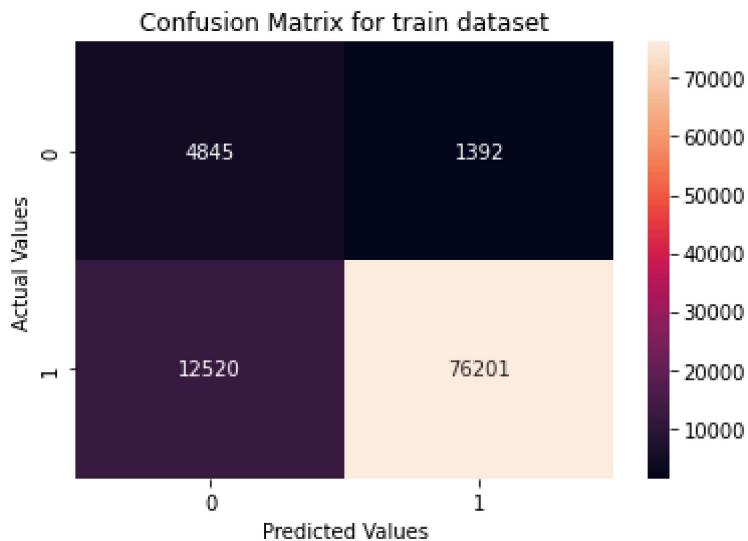
Accuracy for BOW Data for Naive Bayes 0.8557708508845829  
 ROC AUC score for BOW Data for Naive Bayes 0.8032480109486999  
 f1 score for BOW Data for Naive Bayes 0.9179683756588404  
 log loss for Bow data for Naive Bayes 4.981511837183812

In [ ]:

```
1 y_pred_train=classifier_bow.predict(X_tr_bow)
2 plt.title("Confusion Matrix for train dataset")
3 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt='.7g')
4 plt.xlabel("Predicted Values")
5 plt.ylabel("Actual Values")
```

Out[60]:

Text(33.0, 0.5, 'Actual Values')

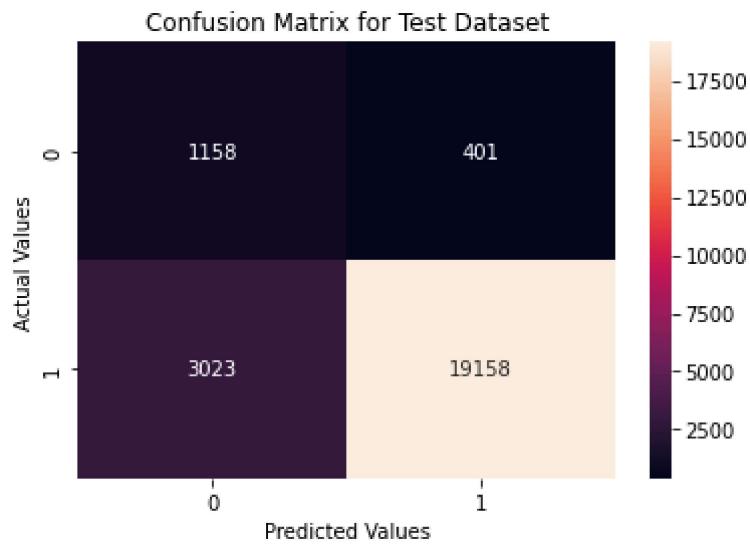


In [ ]:

```
1 plt.title("Confusion Matrix for Test Dataset")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[61]:

Text(33.0, 0.5, 'Actual Values')



Predicting 10 Sample points and comparing it with ground truth

In [ ]:

```

1 y_pred_sample=y_pred[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Thruth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[62]:

	Ground Thruth	Prediction
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
5	0	0
6	1	1
7	1	1
8	1	0
9	1	1

—

Misclassification can be seen while predicting class 1

## 5.1.2 TFIDF

In [ ]:

```

1 #Finding best alpha
2 classifier=MultinomialNB(class_prior=[0.5,0.5])
3 parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]}
4 clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=True)
5 clf.fit(X_tr_tfidf, y_train)
6 classifier_tfidf= MultinomialNB(alpha=clf.best_params_['alpha'],class_prior=[0.5,0.5])
7 classifier_tfidf.fit(X_tr_tfidf, y_train)
8
9 y_pred=classifier_tfidf.predict(X_te_tfidf)

```

In [ ]:

```

1 print("Accuracy for Tfifd Data for Naive Bayes",accuracy_score(y_test,y_pred))
2 print("ROC AUC score for Tfifd Data for Naive Bayes",roc_auc_score(y_test,y_pred))
3 print("f1 score for Tfifd Data for Naive Bayes",f1_score(y_test,y_pred))
4 print("log loss for Tfifd data for Naive Bayes",log_loss(y_test,y_pred))

```

Accuracy for Tfifd Data for Naive Bayes 0.8612468407750632

ROC AUC score for Tfifd Data for Naive Bayes 0.8100547426316098

f1 score for Tfifd Data for Naive Bayes 0.9212790364209922

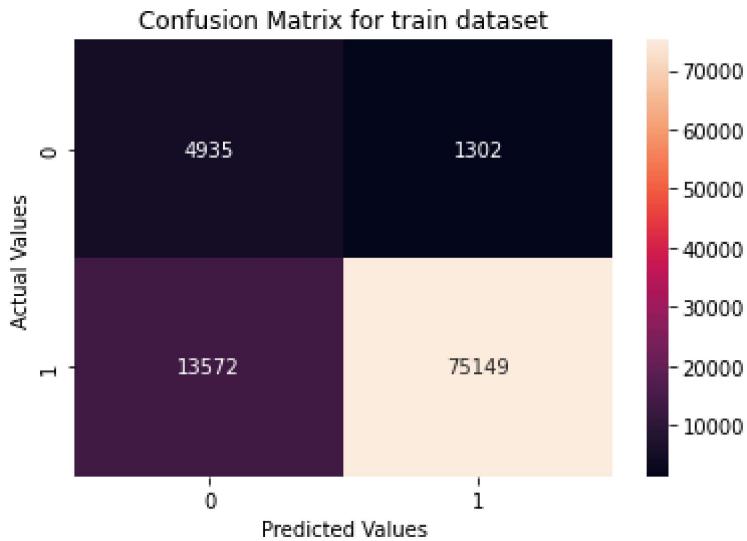
log loss for Tfifd data for Naive Bayes 4.79237740895698

In [ ]:

```
1 y_pred_train=classifier_tfidf.predict(X_tr_bow)
2 plt.title("Confusion Matrix for train dataset")
3 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt='.7g')
4 plt.xlabel("Predicted Values")
5 plt.ylabel("Actual Values")
```

Out[66]:

Text(33.0, 0.5, 'Actual Values')

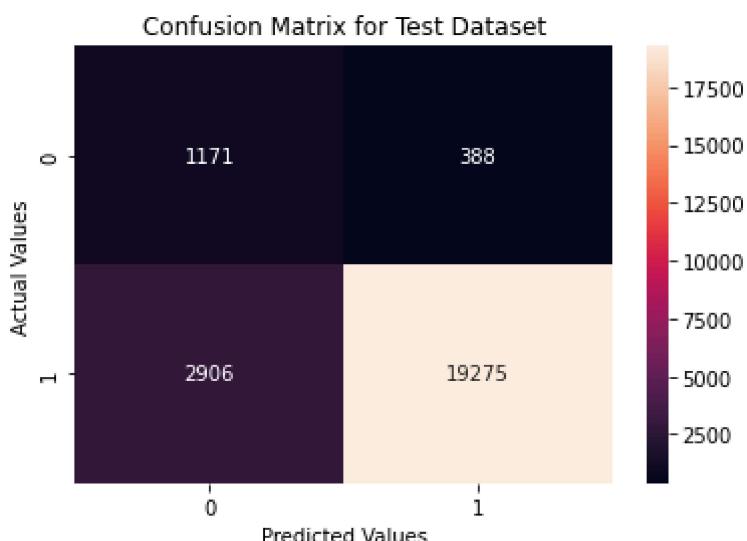


In [ ]:

```
1 plt.title("Confusion Matrix for Test Dataset")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred)),annot=True, fmt='.7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[65]:

Text(33.0, 0.5, 'Actual Values')



## Comparing Ground thruth and prediction

In [ ]:

```

1 y_pred_sample=y_pred[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Thruth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[67]:

	Ground Thruth	Prediction
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
5	0	0
6	1	1
7	1	1
8	1	0
9	1	1

—

### Note:

AvgW2V and Tfifdf W2V can't be used with Naive Bayes as it contains negative values

In [ ]:

```

1 from prettytable import PrettyTable
2
3 x = PrettyTable()
4 x.field_names = ["Vectorizer", "Model", "Accuracy", "AUC", "F1 score"]
5 x.add_row(["BOW", "Naive Bayes", 0.844, 0.80, 0.91])
6 x.add_row(["TFIDF", "Naive Bayes", 0.857, 0.80, 0.91])
7
8 print(x)

```

Vectorizer	Model	Accuracy	AUC	F1 score
BOW	Naive Bayes	0.844	0.8	0.91
TFIDF	Naive Bayes	0.857	0.8	0.91

## 5.2 Logistic Regression

### 5.2.1 BOW

In [ ]:

```

1 classifier=LogisticRegression()
2 parameters = {'C':[100, 10, 1.0, 0.1, 0.01]}
3 clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=True)
4 clf.fit(X_tr_bow1, y_train)
5 classifier_bow_lr= LogisticRegression(C=clf.best_params_[ 'C'])
6 classifier_bow_lr.fit(X_tr_bow1, y_train)
7 y_pred_train=classifier_bow_lr.predict(X_tr_bow1)
8 y_pred_test=classifier_bow_lr.predict(X_te_bow1)

```

In [ ]:

```

1 print("Accuracy for BOW Data for Logistic Regressions",accuracy_score(y_test,y_pred_test))
2 print("ROC AUC score for BOW Data for Logistic Regression",roc_auc_score(y_test,y_pred))
3 print("f1 score for BOW Data for Logistic Regression",f1_score(y_test,y_pred_test))
4 print("log loss for Bow data for Logistic Regression",log_loss(y_test,y_pred_test))

```

Accuracy for BOW Data for Logistic Regressions 0.9534540859309183  
 ROC AUC score for BOW Data for Logistic Regression 0.6998742979323502  
 f1 score for BOW Data for Logistic Regression 0.9755005210296434  
 log loss for Bow data for Logistic Regression 1.6076700060995952

In [ ]:

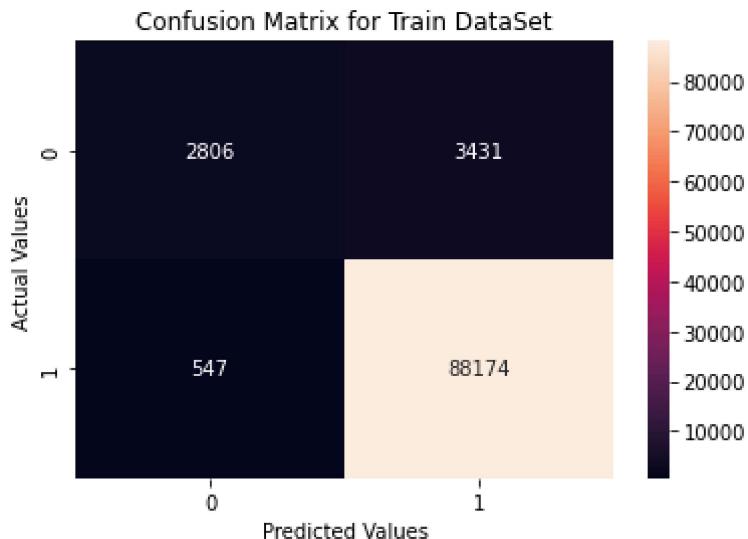
```

1 plt.title("Confusion Matrix for Train DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt='.7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[70]:

Text(33.0, 0.5, 'Actual Values')

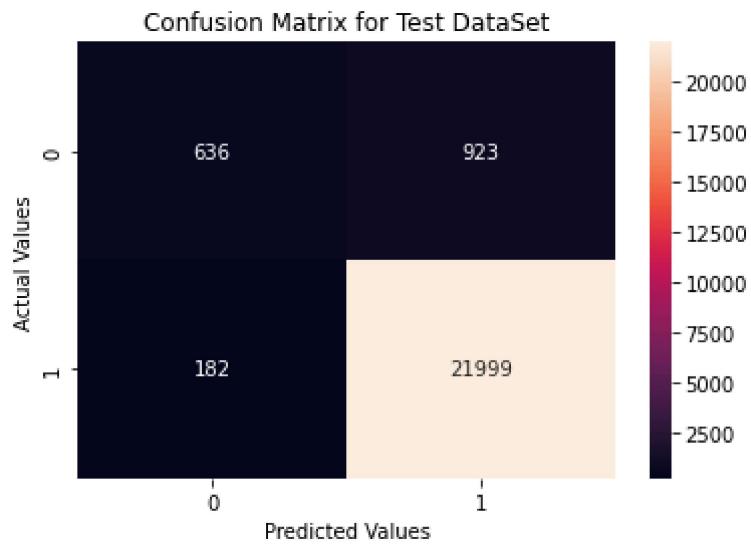


In [ ]:

```
1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g ')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[71]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

```

1 y_pred_sample=y_pred_test[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Thruth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[72]:

	Ground Thruth	Prediction
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
5	0	1
6	1	1
7	1	1
8	1	1
9	1	1

=

## 5.2.2 TFIDF

In [ ]:

```

1 classifier=LogisticRegression()
2 parameters = {'C':[100, 10, 1.0, 0.1, 0.01]}
3 clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=True)
4 clf.fit(X_tr_tfidf1, y_train)
5 classifier_tfidf_lr= LogisticRegression(C=clf.best_params_['C'])
6 classifier_tfidf_lr.fit(X_tr_tfidf1, y_train)
7 y_pred_train=classifier_tfidf_lr.predict(X_tr_tfidf1)
8 y_pred_test=classifier_tfidf_lr.predict(X_te_tfidf1)

```

In [ ]:

```

1 print("Accuracy for Tfifd Data for Logistic Regressions",accuracy_score(y_test,y_pred_tfidf))
2 print("ROC AUC score for Tfifd Data for Logistic Regression",roc_auc_score(y_test,y_pred_tfidf))
3 print("f1 score for Tfifd Data for Logistic Regression",f1_score(y_test,y_pred_test))
4 print("log loss for Tfifd data for Logistic Regression",log_loss(y_test,y_pred_test))

```

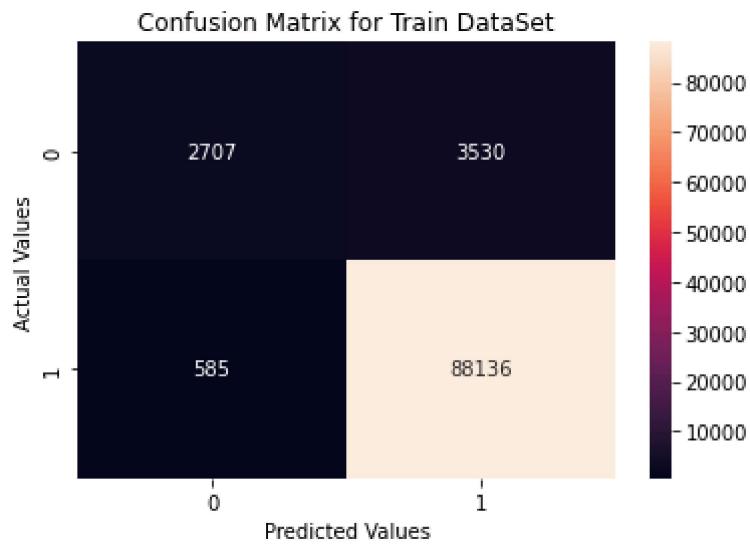
Accuracy for Tfifd Data for Logistic Regressions 0.9537489469250211  
 ROC AUC score for Tfifd Data for Logistic Regression 0.6958576183194425  
 f1 score for Tfifd Data for Logistic Regression 0.9756745978997741  
 log loss for Tfifd data for Logistic Regression 1.5974863396968844

In [ ]:

```
1 plt.title("Confusion Matrix for Train DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[75]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

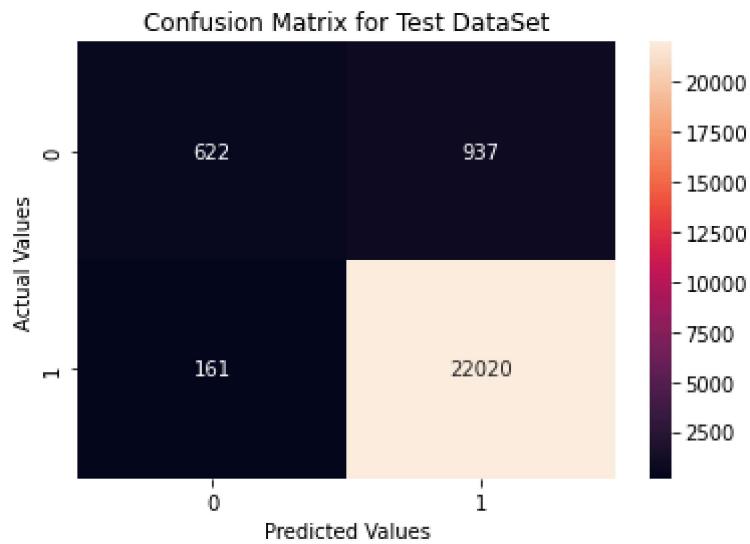
```

1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[76]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

```

1 y_pred_sample=y_pred_test[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[78]:

	Ground Truth	Prediction
0	0	1
1	1	1
2	1	1
3	1	1
4	1	1
5	0	1
6	1	1
7	1	1
8	1	1
9	1	1

## 5.3 SVC

### 5.3.1 BOW

In [ ]:

```

1 classifier_bow_svc= SVC(kernel='linear',C=1)
2 classifier_bow_svc.fit(X_tr_bow1[:50000,:], y_train[:50000])
3 y_pred_train=classifier_bow_svc.predict(X_tr_bow1)
4 y_pred_test=classifier_bow_svc.predict(X_te_bow1)

```

In [ ]:

```

1 with open("bow_svc_model","wb") as f:
2     pickle.dump(classifier_bow_svc,f)

```

In [ ]:

```

1 print("Accuracy for BOW Data for SVC",accuracy_score(y_test,y_pred_test))
2 print("ROC AUC score for BOW Data for SVC",roc_auc_score(y_test,y_pred_test))
3 print("f1 score for BOW Data for SVC",f1_score(y_test,y_pred_test))
4 print("log loss for Bow data for SVC",log_loss(y_test,y_pred_test))

```

Accuracy for BOW Data for SVC 0.9449452401010952  
 ROC AUC score for BOW Data for SVC 0.7018807363605608  
 f1 score for BOW Data for SVC 0.9708629645317342  
 log loss for Bow data for SVC 1.9015543886029016

In [ ]:

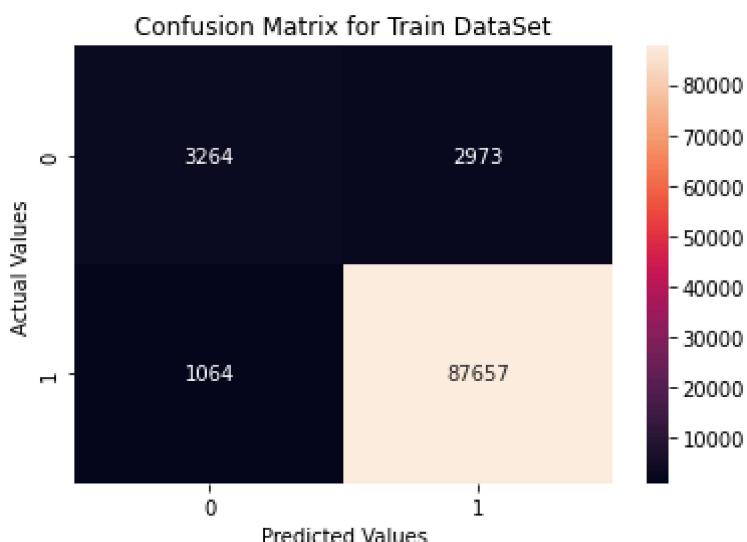
```

1 plt.title("Confusion Matrix for Train DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt='.7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[20]:

Text(33.0, 0.5, 'Actual Values')

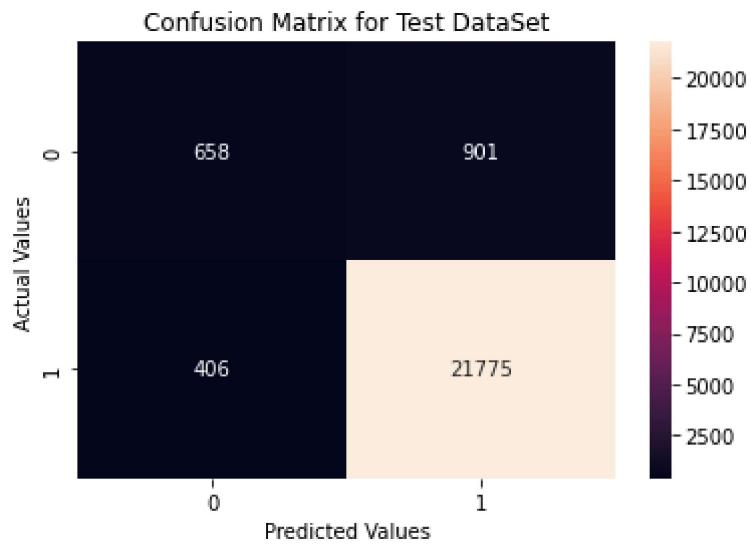


In [ ]:

```
1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g ')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[21]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

```

1 y_pred_sample=y_pred_test[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[22]:

	Ground Truth	Prediction
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
5	0	0
6	1	1
7	1	1
8	1	1
9	1	1

=

### 5.3.2 Tfifdf

In [ ]:

```

1 classifier_tfidf_svc= SVC(kernel='linear',C=1)
2 classifier_tfidf_svc.fit(X_tr_tfidf1[:50000,:], y_train[:50000])
3 y_pred_train=classifier_tfidf_svc.predict(X_tr_tfidf1)
4 y_pred_test=classifier_tfidf_svc.predict(X_te_tfidf1)

```

In [ ]:

```

1 print("Accuracy for Tfifdf Data for SVC",accuracy_score(y_test,y_pred_test))
2 print("ROC AUC score for tfidif Data for SVC",roc_auc_score(y_test,y_pred_test))
3 print("f1 score for tfidif Data for SVC",f1_score(y_test,y_pred_test))
4 print("log loss for tfidif Data for SVC",log_loss(y_test,y_pred_test))

```

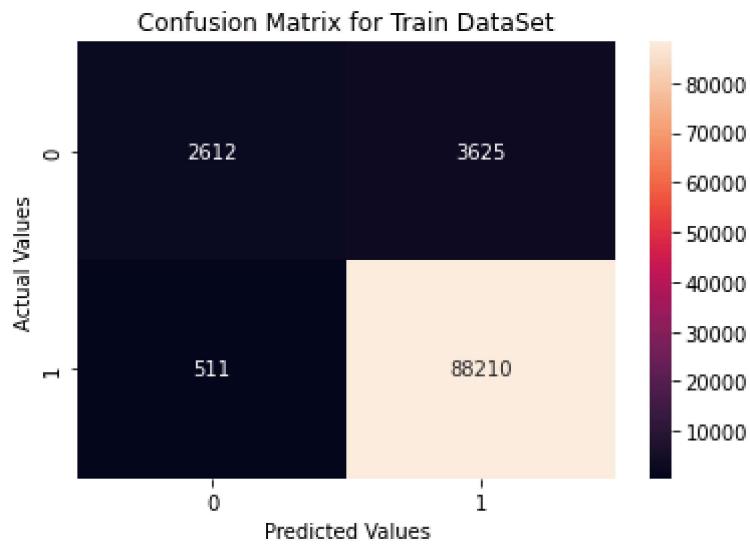
Accuracy for Tfifdf Data for SVC 0.9530328559393428  
 ROC AUC score for tfidif Data for SVC 0.6856345798557029  
 f1 score for tfidif Data for SVC 0.9753247615464625  
 log loss for tfidif Data for SVC 1.6222203576172165

In [ ]:

```
1 plt.title("Confusion Matrix for Train DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[25]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

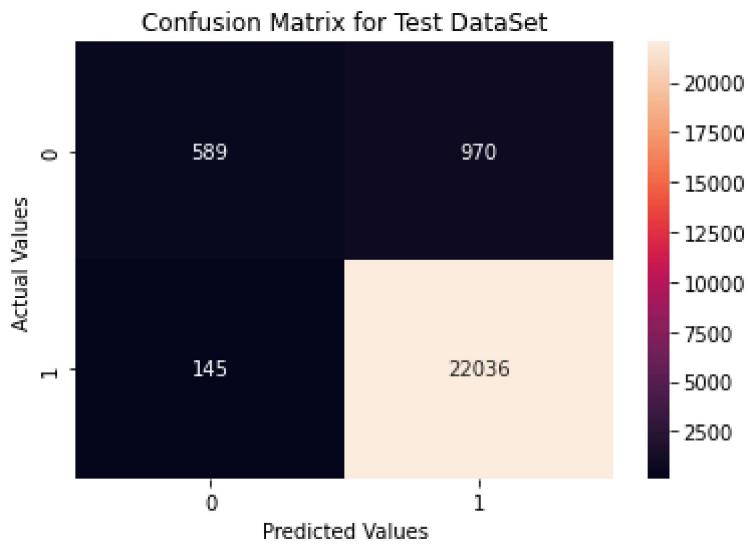
```

1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[26]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

```

1 y_pred_sample=y_pred_test[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[27]:

	Ground Truth	Prediction
0	0	1
1	1	1
2	1	1
3	1	1
4	1	1
5	0	1
6	1	1
7	1	1
8	1	1
9	1	1

## 5.4 SGD

### 5.4.1 BOW

In [ ]:

```

1 classifier=LGBMClassifier(boosting_type='gbdt')
2 parameters={'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3], 'n_estimators':[5,10,50,100,200,500,1000], 'max_depth':[3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99,101,103,105,107,109,111,113,115,117,119,121,123,125,127,129,131,133,135,137,139,141,143,145,147,149,151,153,155,157,159,161,163,165,167,169,171,173,175,177,179,181,183,185,187,189,191,193,195,197,199,201,203,205,207,209,211,213,215,217,219,221,223,225,227,229,231,233,235,237,239,241,243,245,247,249,251,253,255,257,259,261,263,265,267,269,271,273,275,277,279,281,283,285,287,289,291,293,295,297,299,301,303,305,307,309,311,313,315,317,319,321,323,325,327,329,331,333,335,337,339,341,343,345,347,349,351,353,355,357,359,361,363,365,367,369,371,373,375,377,379,381,383,385,387,389,391,393,395,397,399,401,403,405,407,409,411,413,415,417,419,421,423,425,427,429,431,433,435,437,439,441,443,445,447,449,451,453,455,457,459,461,463,465,467,469,471,473,475,477,479,481,483,485,487,489,491,493,495,497,499,501,503,505,507,509,511,513,515,517,519,521,523,525,527,529,531,533,535,537,539,541,543,545,547,549,551,553,555,557,559,561,563,565,567,569,571,573,575,577,579,581,583,585,587,589,591,593,595,597,599,601,603,605,607,609,611,613,615,617,619,621,623,625,627,629,631,633,635,637,639,641,643,645,647,649,651,653,655,657,659,661,663,665,667,669,671,673,675,677,679,681,683,685,687,689,691,693,695,697,699,701,703,705,707,709,711,713,715,717,719,721,723,725,727,729,731,733,735,737,739,741,743,745,747,749,751,753,755,757,759,761,763,765,767,769,771,773,775,777,779,781,783,785,787,789,791,793,795,797,799,801,803,805,807,809,811,813,815,817,819,821,823,825,827,829,831,833,835,837,839,841,843,845,847,849,851,853,855,857,859,861,863,865,867,869,871,873,875,877,879,881,883,885,887,889,891,893,895,897,899,901,903,905,907,909,911,913,915,917,919,921,923,925,927,929,931,933,935,937,939,941,943,945,947,949,951,953,955,957,959,961,963,965,967,969,971,973,975,977,979,981,983,985,987,989,991,993,995,997,999,1001,1003,1005,1007,1009,1011,1013,1015,1017,1019,1021,1023,1025,1027,1029,1031,1033,1035,1037,1039,1041,1043,1045,1047,1049,1051,1053,1055,1057,1059,1061,1063,1065,1067,1069,1071,1073,1075,1077,1079,1081,1083,1085,1087,1089,1091,1093,1095,1097,1099,1101,1103,1105,1107,1109,1111,1113,1115,1117,1119,1121,1123,1125,1127,1129,1131,1133,1135,1137,1139,1141,1143,1145,1147,1149,1151,1153,1155,1157,1159,1161,1163,1165,1167,1169,1171,1173,1175,1177,1179,1181,1183,1185,1187,1189,1191,1193,1195,1197,1199,1201,1203,1205,1207,1209,1211,1213,1215,1217,1219,1221,1223,1225,1227,1229,1231,1233,1235,1237,1239,1241,1243,1245,1247,1249,1251,1253,1255,1257,1259,1261,1263,1265,1267,1269,1271,1273,1275,1277,1279,1281,1283,1285,1287,1289,1291,1293,1295,1297,1299,1301,1303,1305,1307,1309,1311,1313,1315,1317,1319,1321,1323,1325,1327,1329,1331,1333,1335,1337,1339,1341,1343,1345,1347,1349,1351,1353,1355,1357,1359,1361,1363,1365,1367,1369,1371,1373,1375,1377,1379,1381,1383,1385,1387,1389,1391,1393,1395,1397,1399,1401,1403,1405,1407,1409,1411,1413,1415,1417,1419,1421,1423,1425,1427,1429,1431,1433,1435,1437,1439,1441,1443,1445,1447,1449,1451,1453,1455,1457,1459,1461,1463,1465,1467,1469,1471,1473,1475,1477,1479,1481,1483,1485,1487,1489,1491,1493,1495,1497,1499,1501,1503,1505,1507,1509,1511,1513,1515,1517,1519,1521,1523,1525,1527,1529,1531,1533,1535,1537,1539,1541,1543,1545,1547,1549,1551,1553,1555,1557,1559,1561,1563,1565,1567,1569,1571,1573,1575,1577,1579,1581,1583,1585,1587,1589,1591,1593,1595,1597,1599,1601,1603,1605,1607,1609,1611,1613,1615,1617,1619,1621,1623,1625,1627,1629,1631,1633,1635,1637,1639,1641,1643,1645,1647,1649,1651,1653,1655,1657,1659,1661,1663,1665,1667,1669,1671,1673,1675,1677,1679,1681,1683,1685,1687,1689,1691,1693,1695,1697,1699,1701,1703,1705,1707,1709,1711,1713,1715,1717,1719,1721,1723,1725,1727,1729,1731,1733,1735,1737,1739,1741,1743,1745,1747,1749,1751,1753,1755,1757,1759,1761,1763,1765,1767,1769,1771,1773,1775,1777,1779,1781,1783,1785,1787,1789,1791,1793,1795,1797,1799,1801,1803,1805,1807,1809,1811,1813,1815,1817,1819,1821,1823,1825,1827,1829,1831,1833,1835,1837,1839,1841,1843,1845,1847,1849,1851,1853,1855,1857,1859,1861,1863,1865,1867,1869,1871,1873,1875,1877,1879,1881,1883,1885,1887,1889,1891,1893,1895,1897,1899,1901,1903,1905,1907,1909,1911,1913,1915,1917,1919,1921,1923,1925,1927,1929,1931,1933,1935,1937,1939,1941,1943,1945,1947,1949,1951,1953,1955,1957,1959,1961,1963,1965,1967,1969,1971,1973,1975,1977,1979,1981,1983,1985,1987,1989,1991,1993,1995,1997,1999,2001,2003,2005,2007,2009,2011,2013,2015,2017,2019,2021,2023,2025,2027,2029,2031,2033,2035,2037,2039,2041,2043,2045,2047,2049,2051,2053,2055,2057,2059,2061,2063,2065,2067,2069,2071,2073,2075,2077,2079,2081,2083,2085,2087,2089,2091,2093,2095,2097,2099,2101,2103,2105,2107,2109,2111,2113,2115,2117,2119,2121,2123,2125,2127,2129,2131,2133,2135,2137,2139,2141,2143,2145,2147,2149,2151,2153,2155,2157,2159,2161,2163,2165,2167,2169,2171,2173,2175,2177,2179,2181,2183,2185,2187,2189,2191,2193,2195,2197,2199,2201,2203,2205,2207,2209,2211,2213,2215,2217,2219,2221,2223,2225,2227,2229,2231,2233,2235,2237,2239,2241,2243,2245,2247,2249,2251,2253,2255,2257,2259,2261,2263,2265,2267,2269,2271,2273,2275,2277,2279,2281,2283,2285,2287,2289,2291,2293,2295,2297,2299,2301,2303,2305,2307,2309,2311,2313,2315,2317,2319,2321,2323,2325,2327,2329,2331,2333,2335,2337,2339,2341,2343,2345,2347,2349,2351,2353,2355,2357,2359,2361,2363,2365,2367,2369,2371,2373,2375,2377,2379,2381,2383,2385,2387,2389,2391,2393,2395,2397,2399,2401,2403,2405,2407,2409,2411,2413,2415,2417,2419,2421,2423,2425,2427,2429,2431,2433,2435,2437,2439,2441,2443,2445,2447,2449,2451,2453,2455,2457,2459,2461,2463,2465,2467,2469,2471,2473,2475,2477,2479,2481,2483,2485,2487,2489,2491,2493,2495,2497,2499,2501,2503,2505,2507,2509,2511,2513,2515,2517,2519,2521,2523,2525,2527,2529,2531,2533,2535,2537,2539,2541,2543,2545,2547,2549,2551,2553,2555,2557,2559,2561,2563,2565,2567,2569,2571,2573,2575,2577,2579,2581,2583,2585,2587,2589,2591,2593,2595,2597,2599,2601,2603,2605,2607,2609,2611,2613,2615,2617,2619,2621,2623,2625,2627,2629,2631,2633,2635,2637,2639,2641,2643,2645,2647,2649,2651,2653,2655,2657,2659,2661,2663,2665,2667,2669,2671,2673,2675,2677,2679,2681,2683,2685,2687,2689,2691,2693,2695,2697,2699,2701,2703,2705,2707,2709,2711,2713,2715,2717,2719,2721,2723,2725,2727,2729,2731,2733,2735,2737,2739,2741,2743,2745,2747,2749,2751,2753,2755,2757,2759,2761,2763,2765,2767,2769,2771,2773,2775,2777,2779,2781,2783,2785,2787,2789,2791,2793,2795,2797,2799,2801,2803,2805,2807,2809,2811,2813,2815,2817,2819,2821,2823,2825,2827,2829,2831,2833,2835,2837,2839,2841,2843,2845,2847,2849,2851,2853,2855,2857,2859,2861,2863,2865,2867,2869,2871,2873,2875,2877,2879,2881,2883,2885,2887,2889,2891,2893,2895,2897,2899,2901,2903,2905,2907,2909,2911,2913,2915,2917,2919,2921,2923,2925,2927,2929,2931,2933,2935,2937,2939,2941,2943,2945,2947,2949,2951,2953,2955,2957,2959,2961,2963,2965,2967,2969,2971,2973,2975,2977,2979,2981,2983,2985,2987,2989,2991,2993,2995,2997,2999,3001,3003,3005,3007,3009,3011,3013,3015,3017,3019,3021,3023,3025,3027,3029,3031,3033,3035,3037,3039,3041,3043,3045,3047,3049,3051,3053,3055,3057,3059,3061,3063,3065,3067,3069,3071,3073,3075,3077,3079,3081,3083,3085,3087,3089,3091,3093,3095,3097,3099,3101,3103,3105,3107,3109,3111,3113,3115,3117,3119,3121,3123,3125,3127,3129,3131,3133,3135,3137,3139,3141,3143,3145,3147,3149,3151,3153,3155,3157,3159,3161,3163,3165,3167,3169,3171,3173,3175,3177,3179,3181,3183,3185,3187,3189,3191,3193,3195,3197,3199,3201,3203,3205,3207,3209,3211,3213,3215,3217,3219,3221,3223,3225,3227,3229,3231,3233,3235,3237,3239,3241,3243,3245,3247,3249,3251,3253,3255,3257,3259,3261,3263,3265,3267,3269,3271,3273,3275,3277,3279,3281,3283,3285,3287,3289,3291,3293,3295,3297,3299,3301,3303,3305,3307,3309,3311,3313,3315,3317,3319,3321,3323,3325,3327,3329,3331,3333,3335,3337,3339,3341,3343,3345,3347,3349,3351,3353,3355,3357,3359,3361,3363,3365,3367,3369,3371,3373,3375,3377,3379,3381,3383,3385,3387,3389,3391,3393,3395,3397,3399,3401,3403,3405,3407,3409,3411,3413,3415,3417,3419,3421,3423,3425,3427,3429,3431,3433,3435,3437,3439,3441,3443,3445,3447,3449,3451,3453,3455,3457,3459,3461,3463,3465,3467,3469,3471,3473,3475,3477,3479,3481,3483,3485,3487,3489,3491,3493,3495,3497,3499,3501,3503,3505,3507,3509,3511,3513,3515,3517,3519,3521,3523,3525,3527,3529,3531,3533,3535,3537,3539,3541,3543,3545,3547,3549,3551,3553,3555,3557,3559,3561,3563,3565,3567,3569,3571,3573,3575,3577,3579,3581,3583,3585,3587,3589,3591,3593,3595,3597,3599,3601,3603,3605,3607,3609,3611,3613,3615,3617,3619,3621,3623,3625,3627,3629,3631,3633,3635,3637,3639,3641,3643,3645,3647,3649,3651,3653,3655,3657,3659,3661,3663,3665,3667,3669,3671,3673,3675,3677,3679,3681,3683,3685,3687,3689,3691,3693,3695,3697,3699,3701,3703,3705,3707,3709,3711,3713,3715,3717,3719,3721,3723,3725,3727,3729,3731,3733,3735,3737,3739,3741,3743,3745,3747,3749,3751,3753,3755,3757,3759,3761,3763,3765,3767,3769,3771,3773,3775,3777,3779,3781,3783,3785,3787,3789,3791,3793,3795,3797,3799,3801,3803,3805,3807,3809,3811,3813,3815,3817,3819,3821,3823,3825,3827,3829,3831,3833,3835,3837,3839,3841,3843,3845,3847,3849,3851,3853,3855,3857,3859,3861,3863,3865,3867,3869,3871,3873,3875,3877,3879,3881,3883,3885,3887,3889,3891,3893,3895,3897,3899,3901,3903,3905,3907,3909,3911,3913,3915,3917,3919,3921,3923,3925,3927,3929,3931,3933,3935,3937,3939,3941,3943,3945,3947,3949,3951,3953,3955,3957,3959,3961,3963,3965,3967,3969,3971,3973,3975,3977,3979,3981,3983,3985,3987,3989,3991,3993,3995,3997,3999,4001,4003,4005,4007,4009,4011,4013,4015,4017,4019,4021,4023,4025,4027,4029,4031,4033,4035,4037,4039,4041,4043,4045,4047,4049,4051,4053,4055,4057,4059,4061,4063,4065,4067,4069,4071,4073,4075,4077,4079,4081,4083,4085,4087,4089,4091,4093,4095,4097,4099,4101,4103,4105,4107,4109,4111,4113,4115,4117,4119,4121,4123,4125,4127,4129,4131,4133,4135,4137,4139,4141,4143,4145,4147,4149,4151,4153,4155,4157,4159,4161,4163,4165,4167,4169,4171,4173,4175,4177,4179,4181,4183,4185,4187,4189,4191,4193,4195,4197,4199,4201,4203,4205,4207,4209,4211,4213,4215,4217,4219,4221,4223,4225,4227,4229,4231,4233,4235,4237,4239,4241,4243,4245,4247,4249,4251,4253,4255,4257,4259,4261,4263,4265,4267,4269,4271,4273,4275,4277,4279,4281,4283,4285,4287,4289,4291,4293,4295,4297,4299,4301,4303,4305,4307,4309,4311,4313,4315,4317,4319,4321,4323,4325,4327,4329,4331,4333,4335,4337,4339,4341,4343,4345,4347,4349,4351,4353,4355,4357,4359,4361,4363,4365,4367,4369,4371,4373,4375,4377,4379,4381,4383,4385,4387,4389,4391,4393,4395,4397,4399,4401,4403,4405,4407,4409,4411,4413,4415,4417,4419,4421,4423,4425,4427,4429,4431,4433,4435,4437,4439,4441,4443,4445,4447,4449,4451,4453,4455,4457,4459,4461,4463,4465,4467,4469,4471,4473,4475,4477,4479,4481,4483,4485,4487,4489,4491,4493,4495,4497,4499,4501,4503,4505,4507,4509,4511,4513,4515,4517,4519,4521,4523,4525,4527,4529,4531,4533,4535,4537,4539,4541,4543,4545,4547,4549,4551,4553,4555,4557,4559,4561,4563,4565,4567,4569,4571,4573,4575,4577,4579,4581,4583,4585,4587,4589,4591,4593,4595,4597,4599,4601,4603,460
```

In [ ]:

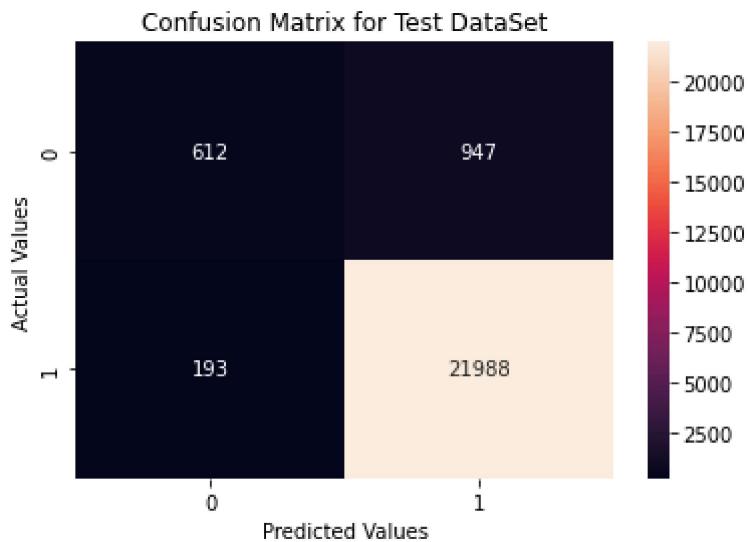
```

1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[31]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

```

1 y_pred_sample=y_pred_test[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[32]:

	Ground Truth	Prediction
0	0	1
1	1	1
2	1	1
3	1	1
4	1	1
5	0	1
6	1	1
7	1	1
8	1	1
9	1	1

## 5.4.2 Tfifdf

In [ ]:

```

1 classifier=LGBMClassifier(boosting_type='gbdt')
2 parameters={'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3], 'n_estimators':[5,10,50,100,200,500,1000,2000,5000,10000]}
3 clf=RandomizedSearchCV(classifier,param_distributions=parameters,scoring='roc_auc',cv=3)
4 clf.fit(X_tr_tfidf1, y_train)
5 classifier_tfidf_lgbm= LGBMClassifier(learning_rate=clf.best_params_['learning_rate'],n_estimators=clf.best_params_['n_estimators'])
6 classifier_tfidf_lgbm.fit(X_tr_tfidf1, y_train)
7 y_pred_train=classifier_tfidf_lgbm.predict(X_tr_tfidf1)
8 y_pred_test=classifier_tfidf_lgbm.predict(X_te_tfidf1)

```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

In [ ]:

```

1 print("Accuracy for tfidf Data for SGD",accuracy_score(y_test,y_pred_test))
2 print("ROC AUC score for tfidf Data for SGD",roc_auc_score(y_test,y_pred_test))
3 print("f1 score for tfidf Data for SGD",f1_score(y_test,y_pred_test))
4 print("logloss for tfidf Data for SGD",log_loss(y_test,y_pred_test))

```

Accuracy for tfidf Data for SGD 0.951642796967144  
 ROC AUC score for tfidf Data for SGD 0.6908542318418884  
 f1 score for tfidf Data for SGD 0.9745533537261162  
 logloss for tfidf Data for SGD 1.6702306200048933

In [ ]:

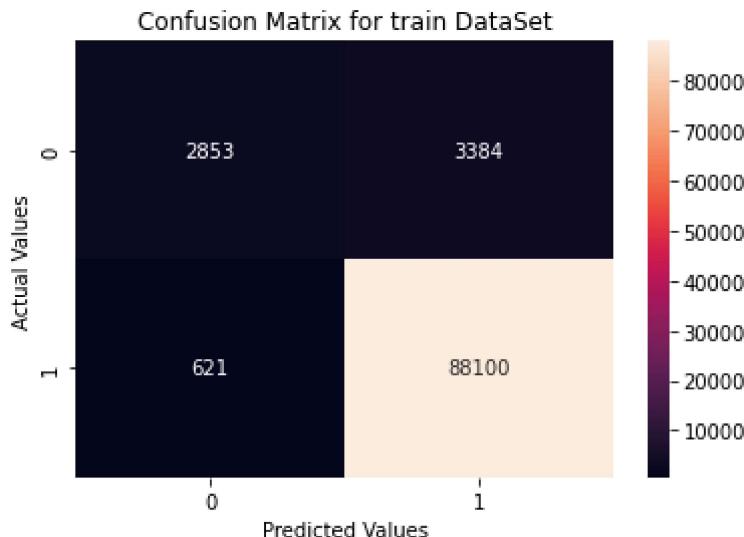
```

1 plt.title("Confusion Matrix for train DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt='.7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[35]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

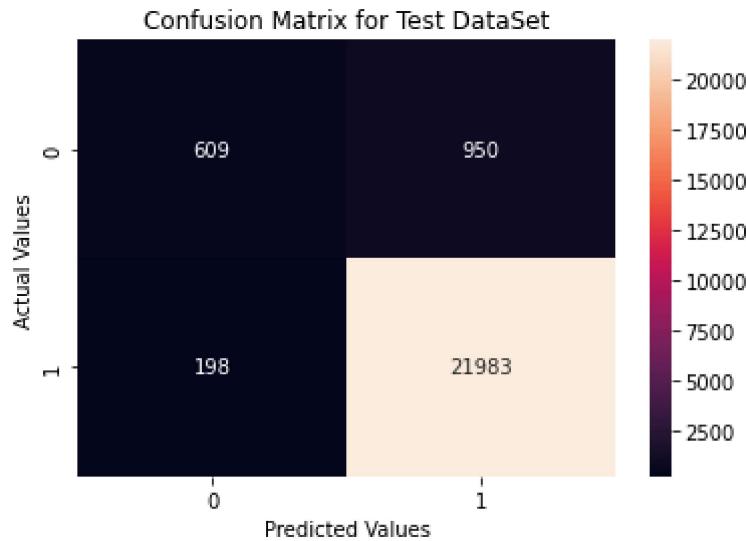
```

1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[36]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

```

1 y_pred_sample=y_pred_test[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[37]:

	Ground Truth	Prediction
0	0	1
1	1	1
2	1	1
3	1	1
4	1	1
5	0	1
6	1	1
7	1	1
8	1	1
9	1	1

## 5.5 Random Forest

### 5.5.1 BOW

In [ ]:

```

1 classifier_bow_rf = RandomForestClassifier(n_estimators=100,max_depth=250)
2 classifier_bow_rf.fit(X_tr_bow1, y_train)
3 y_pred_train=classifier_bow_rf.predict(X_tr_bow1)
4 y_pred_test=classifier_bow_rf.predict(X_te_bow1)
5

```

In [ ]:

```

1 print("Accuracy for bow Data for Random Forest",accuracy_score(y_test,y_pred_test))
2 print("ROC AUC score for bow Data for Random Forest",roc_auc_score(y_test,y_pred_test))
3 print("f1 score for bow Data for Random Forest",f1_score(y_test,y_pred_test))
4 print("log loss for bow data for Random Forest",log_loss(y_test,y_pred_test))

```

Accuracy for bow Data for Random Forest 0.9504633529907329  
 ROC AUC score for bow Data for Random Forest 0.6493728676187593  
 f1 score for bow Data for Random Forest 0.9740740740740741  
 log loss for bow data for Random Forest 1.7109717861340183

In [ ]:

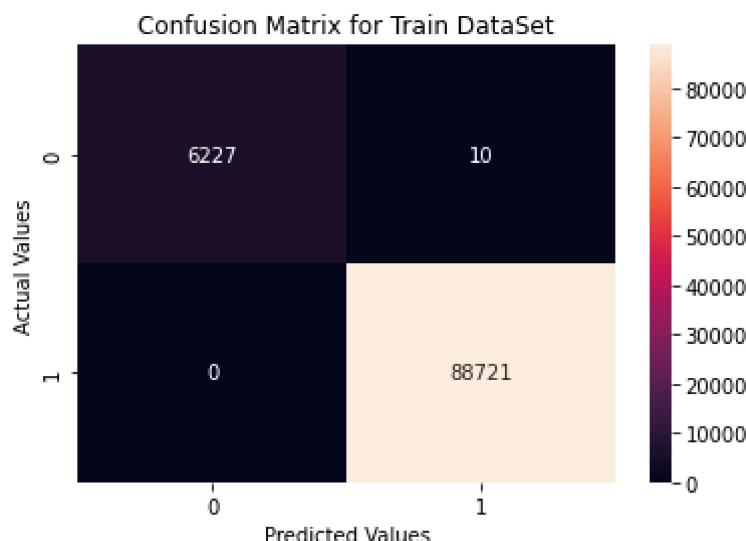
```

1 plt.title("Confusion Matrix for Train DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt=' .7g ')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[41]:

Text(33.0, 0.5, 'Actual Values')

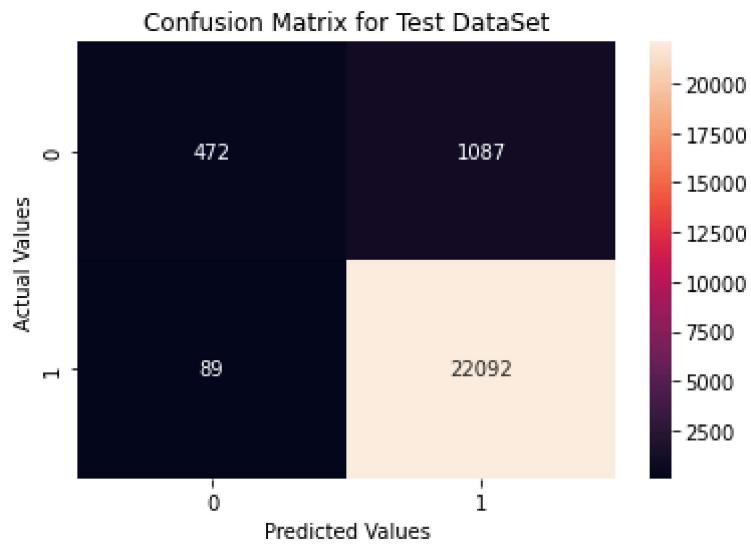


In [ ]:

```
1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g ')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[40]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

```

1 y_pred_sample=y_pred_test[:10]
2 y_test_sample=y_test[:10].values
3 compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
4 compare_df

```

Out[42]:

	Ground Truth	Prediction
0	0	1
1	1	1
2	1	1
3	1	1
4	1	1
5	0	1
6	1	1
7	1	1
8	1	1
9	1	1

	Ground Truth	Prediction
0	0	1
1	1	1
2	1	1
3	1	1
4	1	1
5	0	1
6	1	1
7	1	1
8	1	1
9	1	1

—

## 5.5.2 TFIDF

In [ ]:

```

1 classifier_tfidf_rf= RandomForestClassifier(n_estimators=100,max_depth=250)
2 classifier_tfidf_rf.fit(X_tr_tfidf1, y_train)
3 y_pred_train=classifier_tfidf_rf.predict(X_tr_tfidf1)
4 y_pred_test=classifier_tfidf_rf.predict(X_te_tfidf1)
5

```

In [ ]:

```

1 print("Accuracy for Tfifd Data for Random Forest",accuracy_score(y_test,y_pred_test))
2 print("ROC AUC score for Tfifd Data for Random Forest",roc_auc_score(y_test,y_pred_test))
3 print("f1 score for Tfifd Data for Random Forest",f1_score(y_test,y_pred_test))
4 print("log loss for Tfifd Data for Random Forest", log_loss(y_test,y_pred_test))

```

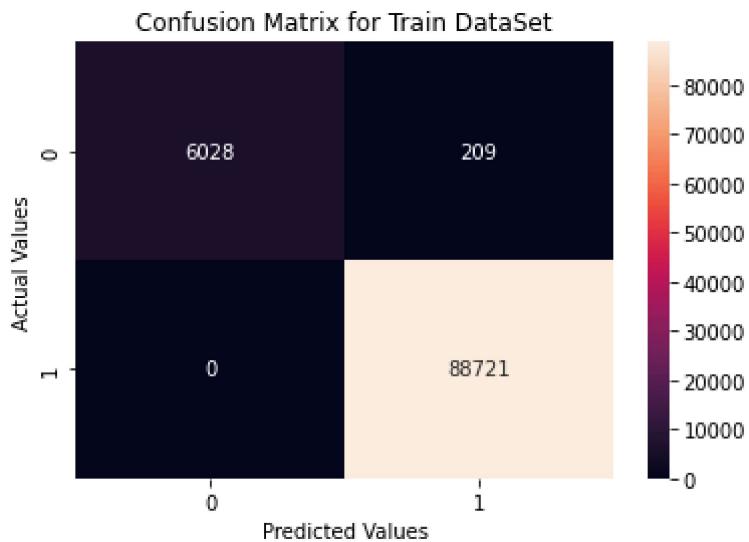
Accuracy for Tfifd Data for Random Forest 0.9503791069924179  
 ROC AUC score for Tfifd Data for Random Forest 0.6466441946410977  
 f1 score for Tfifd Data for Random Forest 0.9740391396332861  
 log loss for Tfifd Data for Random Forest 1.713881842964966

In [ ]:

```
1 plt.title("Confusion Matrix for Train DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_train,y_pred_train)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")
```

Out[45]:

Text(33.0, 0.5, 'Actual Values')



In [ ]:

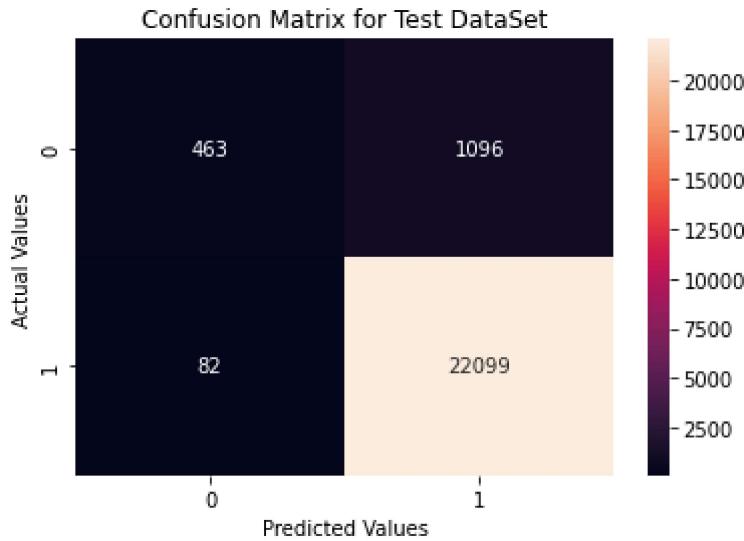
```

1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,y_pred_test)),annot=True, fmt=' .7g')
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[46]:

Text(33.0, 0.5, 'Actual Values')



## 5.6 MLP

In [ ]:

```

1 #https://medium.com/@thongonary/how-to-compute-f1-score-for-each-epoch-in-keras-a1acd17
2 class Metrics(tf.keras.callbacks.Callback):
3     def __init__(self):
4         self.validation_data = (X_te_bow1,y_test)
5     def on_train_begin(self, logs={}):
6         self.val_f1s = []
7         self.val_auc=[]
8
9     def on_epoch_end(self, epoch, logs={}):
10        val_predict = (np.asarray(self.model.predict(self.validation_data[0]))).round()
11        val_targ = self.validation_data[1]
12        _val_f1 = f1_score(val_targ, val_predict)
13        _val_auc=roc_auc_score(val_targ,val_predict)
14        self.val_auc.append(_val_auc)
15        self.val_f1s.append(_val_f1)
16        print(" - val_f1: %f - val_auc: %f" %(_val_f1,_val_auc))

```

### 5.6.1 BOW

In [ ]:

```

1 tf.keras.backend.clear_session()
2 !rm -rf ./logs

```

In [ ]:

```

1 model = Sequential()
2 model.add(Dense(32, input_dim=X_tr_bow1.shape[1], activation='relu'))
3 model.add(tf.keras.layers.Dropout(0.3))
4 model.add(Dense(16, activation='relu'))
5 model.add(tf.keras.layers.Dropout(0.3))
6 model.add(Dense(1, activation='sigmoid'))
7 optimizer=Adam(0.001)
8 model.compile(loss='binary_crossentropy',metrics=['accuracy'])

```

In [ ]:

```

1 metrics=Metrics()
2 filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
3 checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
4                               save_best_only=True, mode='auto')
5
6 logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
7 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,
8 callbacks_list=[metrics,checkpoint,tensorboard_callback]
9 model.fit(X_tr_bow1.toarray(),y_train,epochs=30,validation_data=(X_te_bow1.toarray()),y_

```

WARNING:tensorflow: `write\_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.

Epoch 1/30

2960/2968 [=====>.] - ETA: 0s - loss: 0.1900 - accuracy: 0.9477 - val\_f1: 0.974242 - val\_auc: 0.658819

Epoch 1: val\_accuracy improved from -inf to 0.95084, saving model to model\_save/weights-01-0.9508.hdf5

2968/2968 [=====] - 19s 6ms/step - loss: 0.1898 - accuracy: 0.9478 - val\_loss: 0.1656 - val\_accuracy: 0.9508

Epoch 2/30

2962/2968 [=====>.] - ETA: 0s - loss: 0.1762 - accuracy: 0.9515 - val\_f1: 0.975107 - val\_auc: 0.676739

Epoch 2: val\_accuracy improved from 0.95084 to 0.95257, saving model to model\_save/weights-02-0.9526.hdf5

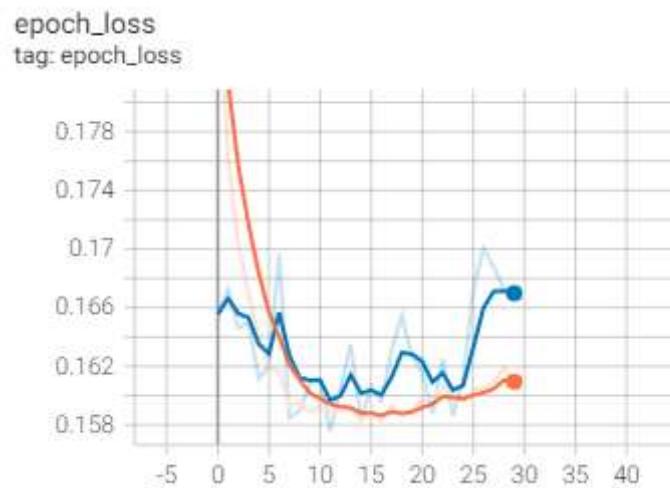
2968/2968 [=====] - 19s 7ms/step - loss: 0.1763 - accuracy: 0.9515 - val\_loss: 0.1673 - val\_accuracy: 0.9526

Epoch 3/30

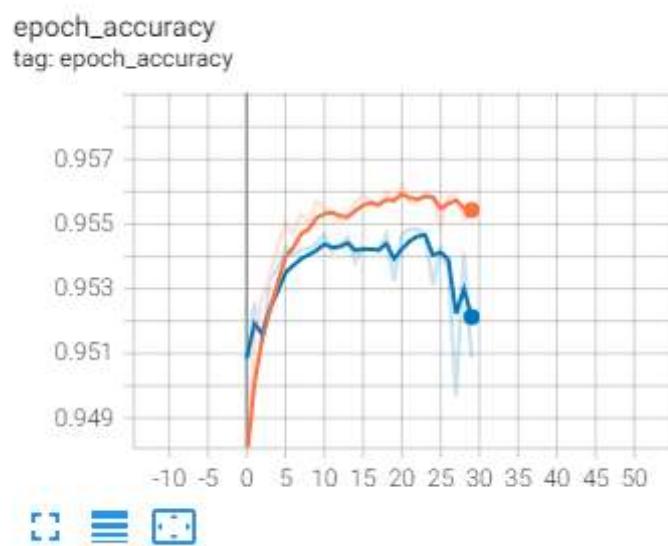
In [ ]:

```
1 %tensorboard --logdir logs
```

<IPython.core.display.Javascript object>



)



The following are the results achieved with MLP for BOW

**Best accuracy is 95.48%, best AUC is 0.714 and best f1 score is 0.976 and best validation loss is 0.16 at epoch 22**

## 5.6.2 TFIDF

In [ ]:

```
1 #https://medium.com/@thongonary/how-to-compute-f1-score-for-each-epoch-in-keras-a1acd17
2 class Metrics(tf.keras.callbacks.Callback):
3     def __init__(self):
4         self.validation_data = (X_te_tfidf1,y_test)
5     def on_train_begin(self, logs={}):
6         self.val_f1s = []
7         self.val_auc=[]
8
9     def on_epoch_end(self, epoch, logs={}):
10        val_predict = (np.asarray(self.model.predict(self.validation_data[0]))).round()
11        val_targ = self.validation_data[1]
12        _val_f1 = f1_score(val_targ, val_predict)
13        _val_auc=roc_auc_score(val_targ,val_predict)
14        self.val_auc.append(_val_auc)
15        self.val_f1s.append(_val_f1)
16        print(" - val_f1: %f - val_auc: %f" %(_val_f1,_val_auc))
```

In [ ]:

```
1 from tensorflow.keras import regularizers
2 model = Sequential()
3 model.add(Dense(32, input_dim=X_tr_tfidf1.shape[1], activation='relu'))
4 model.add(tf.keras.layers.Dropout(0.3))
5 model.add(Dense(16, activation='relu'))
6 model.add(tf.keras.layers.Dropout(0.3))
7 model.add(Dense(1, activation='sigmoid'))
8 optimizer=Adam(0.0001)
9 model.compile(loss='binary_crossentropy',metrics=['accuracy'])
```

In [ ]:

```
1 metrics=Metrics()
2 filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
3 checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
4                               save_best_only=True, mode='auto')
5
6 logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
7 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,
8 callbacks_list=[metrics,checkpoint,tensorboard_callback]
9 model.fit(X_tr_tfidf1.toarray(),y_train,epochs=30,validation_data=(X_te_tfidf1.toarray()
```

WARNING:tensorflow: `write\_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.

Epoch 1/30

1/5935 [...........................] - ETA: 2:03:29 - loss: 0.6887 - accuracy: 0.6250WARNING:tensorflow:Callback method `on\_train\_batch\_end` is slow compared to the batch time (batch time: 0.0045s vs `on\_train\_batch\_end` time: 0.0046s). Check your callbacks.

5930/5935 [=====>.] - ETA: 0s - loss: 0.1864 - accuracy: 0.9457 - val\_f1: 0.973576 - val\_auc: 0.660551

Epoch 1: val\_accuracy improved from -inf to 0.94962, saving model to model\_save/weights-01-0.9496.hdf5

5935/5935 [=====] - 33s 5ms/step - loss: 0.1864 - accuracy: 0.9457 - val\_loss: 0.1626 - val\_accuracy: 0.9496

Epoch 2/30

5931/5935 [=====>.] - ETA: 0s - loss: 0.1750 - accuracy: 0.9494 - val\_f1: 0.974724 - val\_auc: 0.675760

Epoch 2: val\_accuracy improved from 0.94962 to 0.95185, saving model to mo

In [ ]:

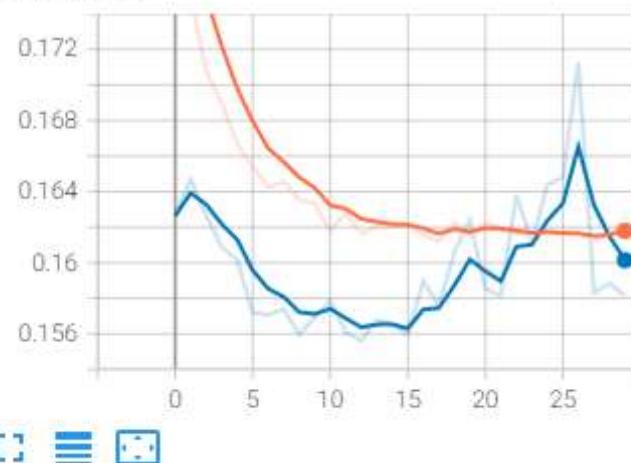
```
1 %tensorboard --logdir logs
```

Reusing TensorBoard on port 6006 (pid 2290), started 2:10:05 ago. (Use '!kill 2290' to kill it.)

<IPython.core.display.Javascript object>

epoch\_loss

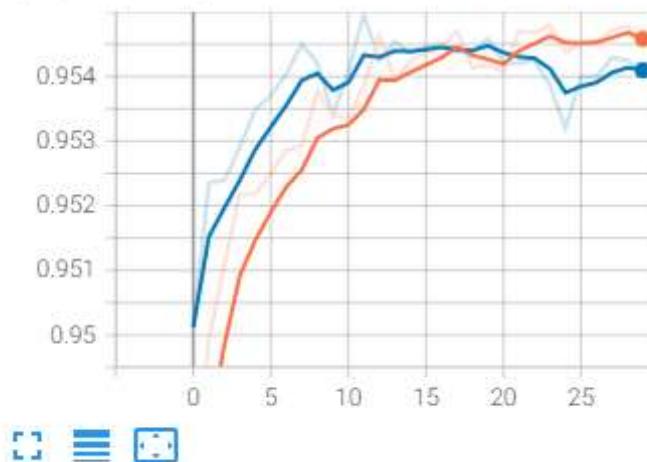
epoch\_loss  
tag: epoch\_loss



)

epoch\_accuracy

epoch\_accuracy  
tag: epoch\_accuracy



The following are the results achieved with MLP for BOW

**Best accuracy is 95.4%, best AUC is 0.70 and best f1 score is 0.976 and best log loss is 0.156 at epoch 12**

## 5.7 LSTM

### 5.7.1 Pretrained Glove Vectors

In [ ]:

```
1 # https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
2 pad_length=183
3 file=open('/content/drive/MyDrive/LSTM/glove_vectors','rb')
4 model=pickle.load(file)
5 words=set(model.keys())
6 file.close()
7 embedding_matrix=np.zeros((vocab_size_text,300))
8 for word, index in token.word_index.items():
9     embedding_vector=model.get(word)
10    if embedding_vector is not None:
11        embedding_matrix[index]+=embedding_vector
```

In [ ]:

```
1 tf.keras.backend.clear_session()
```

Building Model

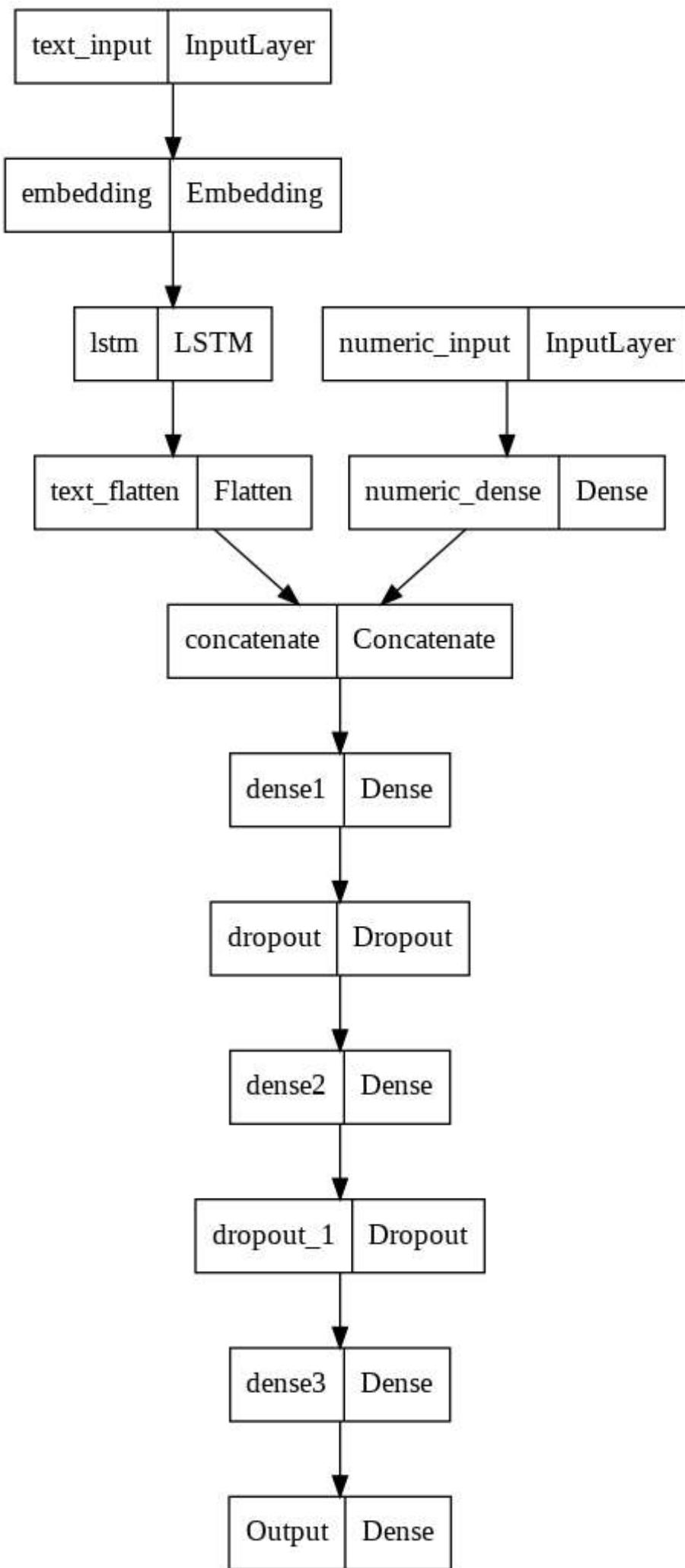
In [ ]:

```
1 #Model
2 #Text Model
3 pad_length=183
4 input_text=Input(shape=(pad_length,),name='text_input')
5 embedd_text=Embedding(input_dim=vocab_size_text,output_dim=300,
6                         input_length=pad_length,weights=[embedding_matrix],trainable=False)
7 lstm=LSTM(units=100,return_sequences=True)(embedd_text)
8 flatten_text=Flatten(name='text_flatten')(lstm)
9 text_model=Model(input_text,flatten_text)
10
11 #Numerical
12 input_numeric=Input(shape=(5,),name='numeric_input')
13 dense_numeric=Dense(units=16,activation='relu',kernel_initializer="glorot_uniform",name='dense1')(input_numeric)
14 numeric_model=Model(input_numeric,dense_numeric)
15
16 #Concatenating
17 concatenate_layer=Concatenate(axis=-1)([text_model.output,numeric_model.output])
18
19 #dense Layer1 after concatenate
20 dense1=Dense(units=256,activation='relu',kernel_initializer="glorot_uniform",
21               name='dense1')(concatenate_layer)
22
23 #Dropout Layer1
24 dropout1=Dropout(0.3)(dense1)
25
26 #Dense Layer2 after concatenate
27 dense2=Dense(units=128,activation='relu',kernel_initializer="glorot_uniform",
28               name='dense2')(dropout1)
29
30 #Dropout Layer2
31 dropout2=Dropout(0.3)(dense2)
32
33 #Dense Layer3
34 dense3=Dense(units=64,activation='relu',kernel_initializer="glorot_uniform",
35               name='dense3')(dropout2)
36
37 #Output Layer
38 output=Dense(units=2,activation='softmax',kernel_initializer="glorot_uniform",name="Output")
39
40 model1=Model([text_model.input,numeric_model.input],output)
41
42
43
```

In [ ]:

```
1 plot_model(model1)
```

Out[22]:



## Compiling the model

In [ ]:

```

1 from tensorflow.keras.callbacks import Callback
2 class RocCallback(Callback):
3     def __init__(self, training_data, validation_data):
4         self.x = training_data[0]
5         self.y = training_data[1]
6         self.x_val = validation_data[0]
7         self.y_val = validation_data[1]
8
9
10    def on_train_begin(self, logs={}):
11        return
12
13    def on_train_end(self, logs={}):
14        return
15
16    def on_epoch_begin(self, epoch, logs={}):
17        return
18
19    def on_epoch_end(self, epoch, logs={}):
20        y_pred_train = self.model.predict(self.x)
21        roc_train = roc_auc_score(self.y, y_pred_train)
22        y_pred_val = self.model.predict(self.x_val)
23        roc_val = roc_auc_score(self.y_val, y_pred_val)
24        print('roc-auc_train: %s - roc-auc_val: %s' % (str(round(roc_train,4)),str(round(roc_val,4))))
25        return
26
27    def on_batch_begin(self, batch, logs={}):
28        return
29
30    def on_batch_end(self, batch, logs={}):
31        return
32

```

In [ ]:

```

1 #https://stackoverflow.com/questions/53500047/stop-training-in-keras-when-accuracy-is-above-a-threshold
2 class MyThresholdCallback(tf.keras.callbacks.Callback):
3     def __init__(self, threshold):
4         super(MyThresholdCallback, self).__init__()
5         self.threshold = threshold
6
7     def on_epoch_end(self, epoch, logs=None):
8         accuracy = logs["val_accuracy"]
9         if accuracy >= self.threshold:
10             self.model.stop_training = True

```

In [ ]:

```

1 model1.compile(optimizer=Adam(learning_rate=1e-4),loss='categorical_crossentropy',
2                 metrics=['accuracy'])
3
4 X_train_complete=[np.array(X_train_text_padded),np.array(X_train_numeric_concatenate)]
5
6
7 X_test_complete=[np.array(X_test_text_padded),np.array(X_test_numeric_concatenate)]
8
9 filepath="model_save/weights_lstm-{epoch:02d}-{val_accuracy:.4f}.hdf5"
10 checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
11                               save_best_only=True, mode='auto')
12
13 logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
14 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,
15
16 roc = RocCallback(training_data=(X_train_complete, y_train_encoded),validation_data=(X_
17 early_stop=MyThresholdCallback(threshold=0.956)
18 callbacks_list=[roc,checkpoint,early_stop,tensorboard_callback]
19
20 model1.fit(X_train_complete,y_train_encoded,epochs=50,validation_data=(X_test_complete,
21               class_weight={0:14,1:1},batch_size=32,callbacks=callbacks_list)
22

```

WARNING:tensorflow: `write\_grads` will be ignored in TensorFlow 2.0 for the  
`TensorBoard` Callback.

Epoch 1/50

4/2968 [........................] - ETA: 1:02 - loss: 1.6093 - accuracy: 0.8516    WARNING:tensorflow:Callback method `on\_train\_batch\_end` is slow compared to the batch time (batch time: 0.0135s vs `on\_train\_batch\_end` time: 0.0147s). Check your callbacks.  
roc-auc\_train: 0.9 - roc-auc\_val: 0.8908

Epoch 1: val\_accuracy improved from -inf to 0.86959, saving model to model\_save/weights\_lstm-01-0.8696.hdf5

2968/2968 [=====] - 74s 23ms/step - loss: 0.8139 - accuracy: 0.8621 - val\_loss: 0.3734 - val\_accuracy: 0.8696

Epoch 2/50

roc-auc\_train: 0.9129 - roc-auc\_val: 0.8947

Epoch 2: val\_accuracy improved from 0.86959 to 0.91500, saving model to model\_save/weights\_lstm-02-0.9150.hdf5

2968/2968 [=====] - 58s 20ms/step - loss: 0.7385

In [ ]:

```
1 %tensorboard --logdir logs
```

<IPython.core.display.Javascript object>

**Best Validation accuracy achieved is 94.63% at epoch 28 where validation AUC is 0.85 and validation loss is 0.3759**

## 5.7.2 Training Embedded Vectors

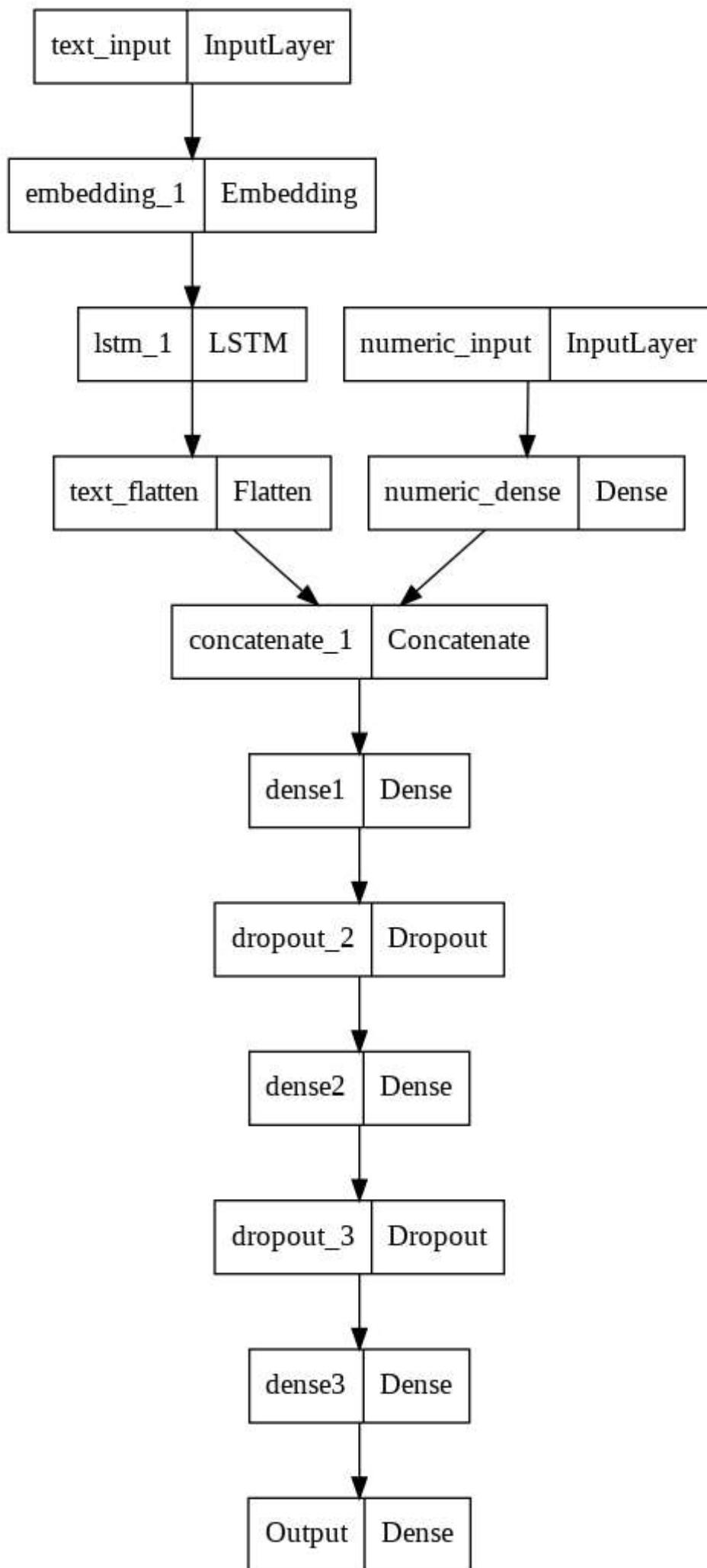
In [ ]:

```
1 #Model
2 #Text Model
3 pad_length=183
4 input_text=Input(shape=(pad_length,),name='text_input')
5 embedd_text=Embedding(input_dim=vocab_size_text,output_dim=300,
6                         input_length=pad_length)(input_text)
7 lstm=LSTM(units=100,return_sequences=True)(embedd_text)
8 flatten_text=Flatten(name='text_flatten')(lstm)
9 text_model=Model(input_text,flatten_text)
10
11 #Numerical
12 input_numeric=Input(shape=(5,),name='numeric_input')
13 dense_numeric=Dense(units=16,activation='relu',kernel_initializer="glorot_uniform",name='dense1')(input_numeric)
14 numeric_model=Model(input_numeric,dense_numeric)
15
16 #Concatenating
17 concatenate_layer=Concatenate(axis=-1)([text_model.output,numeric_model.output])
18
19 #dense Layer1 after concatenate
20 dense1=Dense(units=256,activation='relu',kernel_initializer="glorot_uniform",
21               name='dense1')(concatenate_layer)
22
23 #Dropout Layer1
24 dropout1=Dropout(0.3)(dense1)
25
26 #Dense Layer2 after concatenate
27 dense2=Dense(units=128,activation='relu',kernel_initializer="glorot_uniform",
28               name='dense2')(dropout1)
29
30 #Dropout Layer2
31 dropout2=Dropout(0.3)(dense2)
32
33 #Dense Layer3
34 dense3=Dense(units=64,activation='relu',kernel_initializer="glorot_uniform",
35               name='dense3')(dropout2)
36
37 #Output Layer
38 output=Dense(units=2,activation='softmax',kernel_initializer="glorot_uniform",name="Output")
39
40 model2=Model([text_model.input,numeric_model.input],output)
41
42
43
```

In [ ]:

```
1 plot_model(model2)
```

Out[31]:



In [ ]:

```

1 model2.compile(optimizer=Adam(learning_rate=1e-3),loss='categorical_crossentropy',
2                 metrics=['accuracy'])
3
4 X_train_complete=[np.array(X_train_text_padded),np.array(X_train_numeric_concatenate)]
5
6
7 X_test_complete=[np.array(X_test_text_padded),np.array(X_test_numeric_concatenate)]
8
9
10 filepath="model_save/weights_lstm-{epoch:02d}-{val_accuracy:.4f}.hdf5"
11 checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
12                               save_best_only=True, mode='auto')
13
14 logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
15 tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,
16
17 roc = RocCallback(training_data=(X_train_complete, y_train_encoded),validation_data=(X_
18 early_stop=MyThresholdCallback(threshold=0.956)
19 callbacks_list=[roc,checkpoint,early_stop,tensorboard_callback]
20
21 model2.fit(X_train_complete,y_train_encoded,epochs=50,validation_data=(X_test_complete,
22             class_weight={0:14,1:1},batch_size=32,callbacks=[callbacks_list])

```

WARNING:tensorflow: `write\_grads` will be ignored in TensorFlow 2.0 for the  
`TensorBoard` Callback.

Epoch 1/50

6/2968 [.....] - ETA: 3:22 - loss: 1.1705 - accuracy: 0.8594  
WARNING:tensorflow:Callback method `on\_train\_batch\_end` is slow compared to the batch time (batch time: 0.0283s vs `on\_train\_batch\_end` time: 0.0713s). Check your callbacks.  
roc-auc\_train: 0.9273 - roc-auc\_val: 0.8946

Epoch 1: val\_accuracy improved from -inf to 0.87784, saving model to model\_save/weights\_lstm-01-0.8778.hdf5

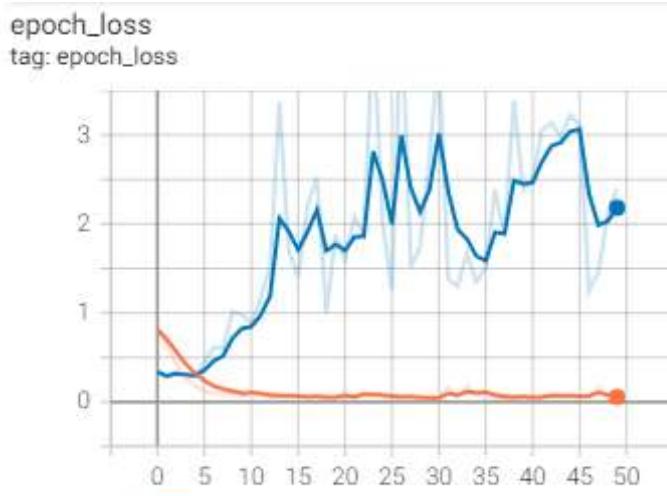
2968/2968 [=====] - 210s 66ms/step - loss: 0.8155 - accuracy: 0.8621 - val\_loss: 0.3352 - val\_accuracy: 0.8778

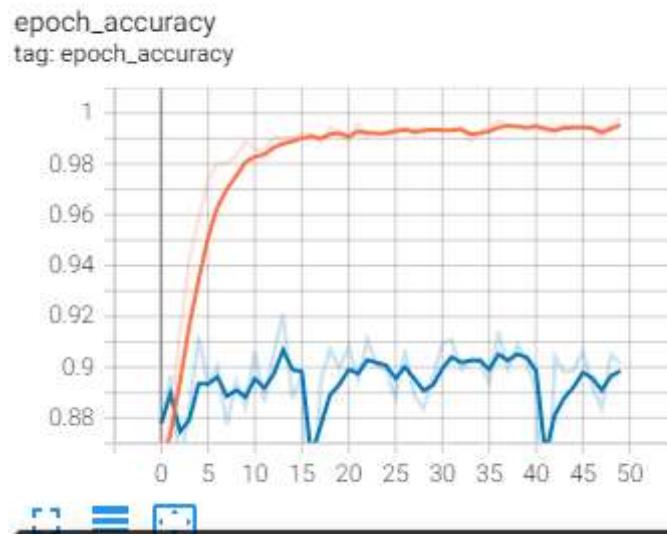
Epoch 2/50

roc-auc\_train: 0.9707 - roc-auc\_val: 0.8801

Epoch 2: val\_accuracy improved from 0.87784 to 0.89684, saving model to model\_save/weights\_lstm-02-0.8968.hdf5

2968/2968 [=====] - 197s 66ms/step - loss: 0.6361





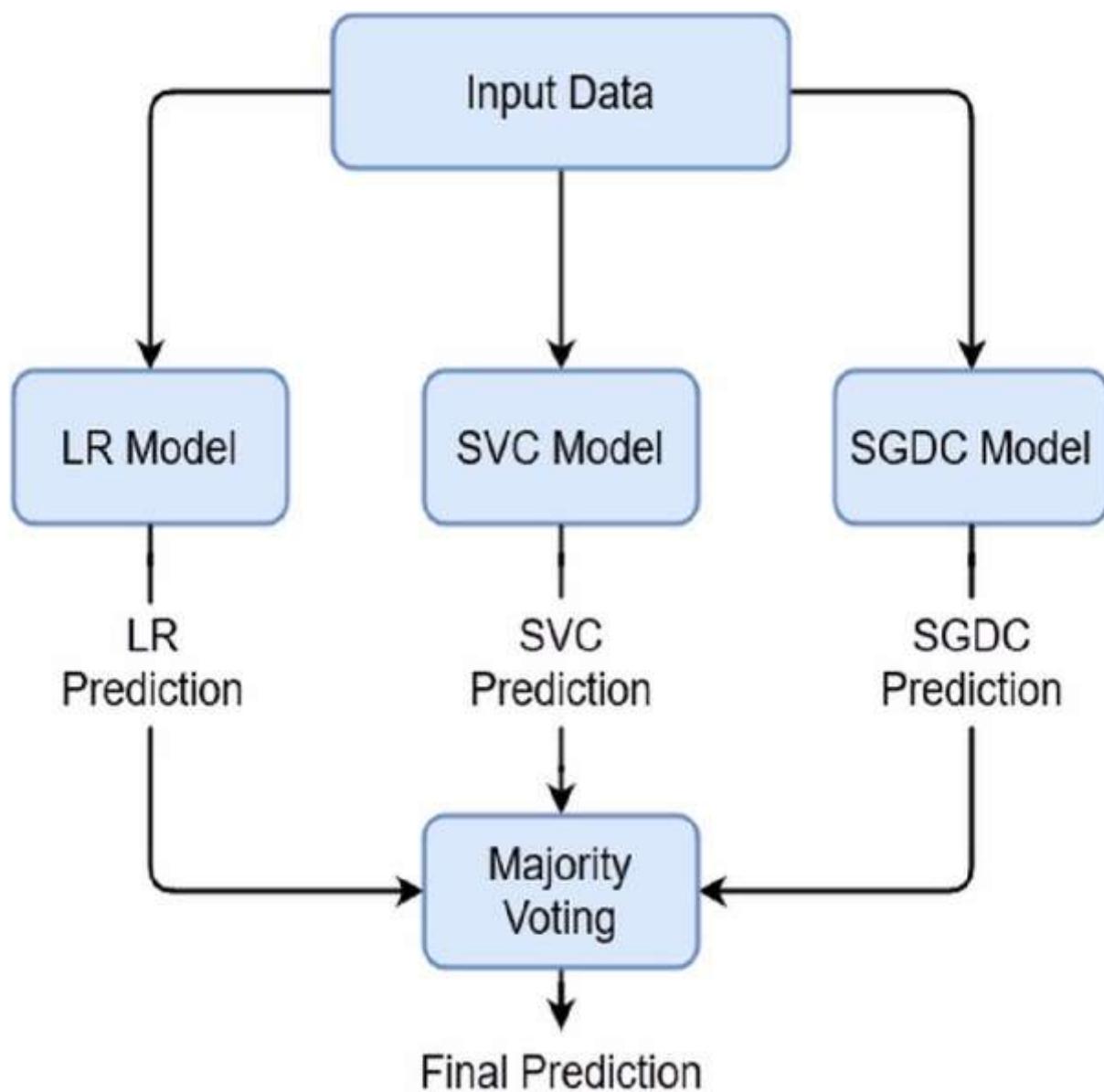
In [ ]:

```
1 %tensorboard --logdir logs
```

<IPython.core.display.Javascript object>

**Best Validation Accuracy achieved is 92.1% and Best AUC is 0.83 and Validation loss is 3.374**

## 5.8 Stacking Classifier



From above we can see that models are performing better with TFIDF data.Hence we are proceeding with models trained with tfidf data

In [ ]:

```

1 #Prediction made by Logistic Regression model trained on Tfifdf data
2 y_pred_lr=classifier_tfidf_lr.predict(X_te_tfidf1)
3
4 #Prediction made by SVC model trained on Tfifdf data
5 y_pred_svc=classifier_tfidf_svc.predict(X_te_tfidf1)
6
7 #Prediction made by SGD model trained on Tfifdf data
8 y_pred_sgd=classifier_tfidf_lgbm.predict(X_te_tfidf1)
  
```

In [ ]:

```

1 import pickle
2 with open("stack_models.pkl","wb") as f:
3     pickle.dump([classifier_tfidf_lr,classifier_tfidf_svc,classifier_tfidf_lgbm],f)
  
```

In [ ]:

```
1 y_pred_sgd.shape
```

Out[104]:

(23740,)

In [ ]:

```
1 #https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list/
2 from collections import Counter
3
4 def most_frequent(List):
5     occurence_count = Counter(List)
6     return occurence_count.most_common(1)[0][0]
```

In [ ]:

```
1 majority_class=[]
2 for i in range(y_pred_sgd.shape[0]):
3     each_point_list=[]
4     each_point_list.append(y_pred_lr[i])
5     each_point_list.append(y_pred_svc[i])
6     each_point_list.append(y_pred_sgd[i])
7     majority_class.append(most_frequent(each_point_list))
```

In [ ]:

```
1 majority_class=np.array(majority_class)
```

In [ ]:

```
1 print("Accuracy for tfidf Data for Stacking Classifier",accuracy_score(y_test,majority_
2 print("ROC AUC score for tfidf Data for Stacking Classifier",roc_auc_score(y_test,major
3 print("f1 score for tfidf Data for Stacking Classifier",f1_score(y_test,majority_class)
4 print("log loss for tfidf data for Stacking Classifier",f1_score(y_test,majority_class)
```

Accuracy for tfidf Data for Stacking Classifier 0.9533698399326032  
 ROC AUC score for tfidf Data for Stacking Classifier 0.6905857398829545  
 f1 score for tfidf Data for Stacking Classifier 0.9754887850674225  
 log loss for tfidf data for Stacking Classifier 0.9754887850674225

In [ ]:

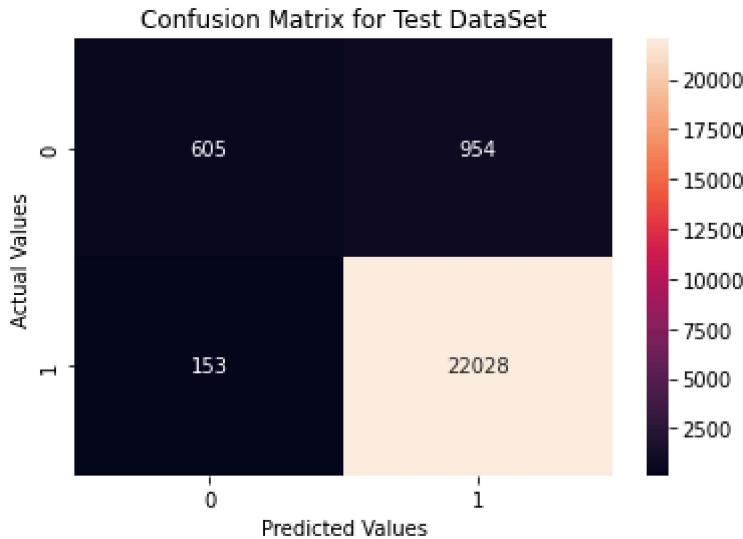
```

1 plt.title("Confusion Matrix for Test DataSet")
2 sns.heatmap(pd.DataFrame(confusion_matrix(y_test,majority_class)),annot=True, fmt=".7g")
3 plt.xlabel("Predicted Values")
4 plt.ylabel("Actual Values")

```

Out[53]:

Text(33.0, 0.5, 'Actual Values')



## 6 Results & Conclusion

In [ ]:

```

1 from prettytable import PrettyTable
2 x = PrettyTable()
3 x.field_names = ["Model", "Vectorizer", "Test Accuracy", "Test AUC Score", "F1 Score", "loc"]
4 x.add_row(["Naive Bayes", "BOW", "85.57%", "0.80", "0.92", "4.98"])
5 x.add_row(["Naive Bayes", "TFIDF", "86.12%", "0.81", "0.92", "4.79"])
6 x.add_row(["logistic Regression", "BOW", "95.34%", "0.69", "0.97", "1.6"])
7 x.add_row(["logistic Regression", "TFIDF", "95.37%", "0.69", "0.97", "1.59"])
8 x.add_row(["Support vector Classifier", "BOW", "94.49%", "0.70", "0.97", "1.9"])
9 x.add_row(["Support vector Classifier", "TFIDF", "95.3%", "0.68", "0.97", "1.62"])
10 x.add_row(["Stochastic Gradient Descent(LightGBM)", "BOW", "95.2%", "0.69", "0.97", "1.65"])
11 x.add_row(["Stochastic Gradient Descent(LightGBM)", "TFIDF", "95.16%", "0.69", "0.97", "1.67"])
12 x.add_row(["Random Forest", "BOW", "95.04%", "0.65", "0.97", "1.71"])
13 x.add_row(["Random Forest", "TFIDF", "95.04%", "0.64", "0.97", "1.71"])
14 x.add_row(["Multi layer perceptron", "BOW", "95.48%", "0.71", "0.97", "0.16"])
15 x.add_row(["Multi layer perceptron", "TFIDF", "95.4%", "0.70", "0.97", "0.15"])
16 x.add_row(["LSTM", "Pretrained Glove Vectors", "94.63%", "0.85", "-", "0.376"])
17 x.add_row(["LSTM", "Embedded training layer", "92.1%", "0.83", "-", "3.37"])
18 x.add_row(["Stacking Classifier", "TFIDF", "95.3%", "0.69", "0.97", "0.97"])

```

In [ ]:

```
1 print(x)
```

		Model		Vectorizer	Test Ac
		accuracy	Test AUC Score	F1 Score	log loss
57%		Naive Bayes	0.8	0.92	4.98
12%		Naive Bayes	0.81	0.92	4.79
34%		logistic Regression	0.69	0.97	1.6
37%		logistic Regression	0.69	0.97	1.59
49%		Support vector Classifier	0.7	0.97	1.9
3%		Support vector Classifier	0.68	0.97	1.62
2%		Stochastic Gradient Descent(LightGBM)	0.69	0.97	1.65
16%		Stochastic Gradient Descent(LightGBM)	0.69	0.97	1.67
04%		Random Forest	0.65	0.97	1.71
04%		Random Forest	0.64	0.97	1.71
48%		Multi layer perceptron	0.71	0.97	0.16
4%		Multi layer perceptron	0.7	0.97	0.15
63%		LSTM	0.85	-	0.376
1%		LSTM	0.83	-	3.37
3%		Stacking Classifier	0.69	0.97	0.97

## Conclusion

1. The Best accuracy achieved is 95.48 % by MLP on BOW Data
2. The Best AUC achieved is 0.85 by LSTM model on Pretrained Glove Vectors
3. The Best model is MLP on Bow data with an accuracy of 95.48% and AUC of 0.85 and logloss of 0.16

In [ ]:

```
1
```

