BaseLine Modelling

1.Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from datetime import datetime
from dateutil.parser import parse
import pickle
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
import re
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from bs4 import BeautifulSoup
import pickle
from scipy.sparse import hstack
from tqdm import tqdm
import os
import pickle
from scipy.sparse import csr_matrix
from sklearn.metrics import accuracy_score, confusion_matrix,roc_auc_score, f1_score,r
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import Input, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Dense,Concatenate, Activation, Flatten
from tensorflow.keras.layers import  BatchNormalization, Dropout, LSTM
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.layers import Embedding, Conv1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.metrics import log_loss
%load_ext tensorboard
import datetime
```

2.Importing Preprocessed data

Importing Preprocessed data

In [4]:

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

In [ ]:

```
data.shape
```

Out[6]:

```
(118698, 17)
```

In [5]:

```
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]:

```
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 [ ]:

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

In [ ]:

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

Out[5]:

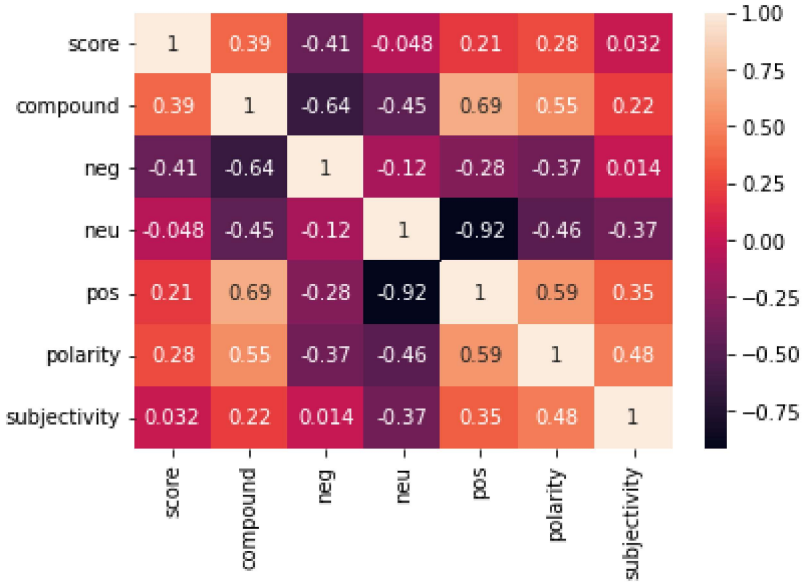
	ReviewTitle	CompleteReview	score	compound	neg	neu	pos	polarity	subjectivity
0	Good Company for Every employee	Good company for every Engineers dream, Full M...	1	0.8625	0.0	0.654	0.346	0.483333	0.65
1	Productive	I am just pass out bsc in chemistry Typical da...	1	0.7264	0.0	0.804	0.196	0.406667	0.54

In [ ]:

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

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9905ed5d0>
```



From above we can see that neu and 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 [ ]:

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

```

In [ ]:

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

In [ ]:

```
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
            'won', "won't", 'wouldn', "wouldn't"]

def preprocess_text(text_data):
    #stemmer=SnowballStemmer(language="english")
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sentance = re.sub(r"http\S+", "", sentance)
        sentance = BeautifulSoup(sentance, 'lxml').get_text()
        sentance = re.sub("\S*\d\S*", "", sentance).strip()
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        sent=sent.lower().strip()
        #sent=" ".join(stemmer.stem(e) for e in sent.split())
        preprocessed_text.append(sent)
    return preprocessed_text

```

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 Managemen...	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 [ ]:

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

In [ ]:

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

Train Test Split

In [ ]:

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

In [ ]:

```
print("Shape of X_train is",X_train.shape)
print("Shape of X_test is ",X_test.shape)
print("Shape of y_train is",y_train.shape)
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 [ ]:

```
X_train.reset_index(drop=True,inplace=True)
y_train.reset_index(drop=True,inplace=True)
X_test.reset_index(drop=True,inplace=True)
y_test.reset_index(drop=True,inplace=True)
```

In [ ]:

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

Normalizing Numerical Features

In [ ]:

```
normalizer = MinMaxScaler()
#normalizer.fit(X_train['compound'].values.reshape(1,-1))
X_train_compound_norm = normalizer.fit_transform(X_train['compound'].values.reshape(-1,
X_test_compound_norm = normalizer.transform(X_test['compound'].values.reshape(-1,1))

normalizer = MinMaxScaler()
#normalizer.fit(X_train['neg'].values.reshape(1,-1))
X_train_neg_norm = normalizer.fit_transform(X_train['neg'].values.reshape(-1,1))
X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(-1,1))

normalizer = MinMaxScaler()
#normalizer.fit(X_train['pos'].values.reshape(1,-1))
X_train_pos_norm = normalizer.fit_transform(X_train['pos'].values.reshape(-1,1))
X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(-1,1))

normalizer = MinMaxScaler()
#normalizer.fit(X_train['polarity'].values.reshape(1,-1))
X_train_polarity_norm = normalizer.fit_transform(X_train['polarity'].values.reshape(-1,
X_test_polarity_norm = normalizer.transform(X_test['polarity'].values.reshape(-1,1))


normalizer = MinMaxScaler()
#normalizer.fit(X_train['subjectivity'].values.reshape(1,-1))
X_train_subjectivity_norm = normalizer.fit_transform(X_train['subjectivity'].values.res
X_test_subjectivity_norm = normalizer.transform(X_test['subjectivity'].values.reshape(-
```

Standard Scaler

In [ ]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_compound_norm = sc.fit_transform(X_train['compound'].values.reshape(-1,1))
X_test_compound_norm = sc.transform(X_test['compound'].values.reshape(-1,1))

sc = StandardScaler()
X_train_neg_norm = sc.fit_transform(X_train['neg'].values.reshape(-1,1))
X_test_neg_norm = sc.transform(X_test['neg'].values.reshape(-1,1))

sc = StandardScaler()
X_train_pos_norm = sc.fit_transform(X_train['pos'].values.reshape(-1,1))
X_test_pos_norm = sc.transform(X_test['pos'].values.reshape(-1,1))

sc = StandardScaler()
X_train_polarity_norm = sc.fit_transform(X_train['polarity'].values.reshape(-1,1))
X_test_polarity_norm = sc.transform(X_test['polarity'].values.reshape(-1,1))


sc = StandardScaler()
X_train_subjectivity_norm = sc.fit_transform(X_train['subjectivity'].values.reshape(-1,
X_test_subjectivity_norm = sc.transform(X_test['subjectivity'].values.reshape(-1,1))

X_train_numerical=pd.DataFrame({'compound':X_train_compound_norm.ravel(),'neg':X_train_
                                'polarity':X_train_polarity_norm.ravel(),'subjectivity'

X_test_numerical=pd.DataFrame({'compound':X_test_compound_norm.ravel(),'neg':X_test_neg
                               'polarity':X_test_polarity_norm.ravel(),'subjectivity':

X_train_numeric_concatenate=X_train_numerical[['compound','neg','pos','polarity','subje
X_test_numeric_concatenate=X_test_numerical[['compound','neg','pos','polarity','subject
```

4.Text to Vector

4.1 BOW

In [ ]:

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

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

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

```
X_tr_bow = hstack((X_train_bow,X_train_compound_norm,X_train_neg_norm,X_train_pos_norm,
X_te_bow = hstack((X_test_bow,X_test_compound_norm,X_test_neg_norm,X_test_pos_norm,X_te

print("Final Data matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_te_bow.shape, y_test.shape)
print("="*100)
```

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

In [ ]:

```
X_tr_bow1 = hstack((X_train_bow, X_train.drop('process_review',axis=1).values)).tocsr()
X_te_bow1= hstack((X_test_bow,X_test.drop('process_review',axis=1).values)).tocsr()

print("Final Data matrix")
print(X_tr_bow1.shape, y_train.shape)
print(X_te_bow1.shape, y_test.shape)
print("="*100)
```

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

4.2 TFIDF

In [ ]:

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

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

In [ ]:

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

X_tr_tfidf = hstack((X_train_tfidf,X_train_compound_norm,X_train_neg_norm,X_train_pos_r
X_te_tfidf = hstack((X_test_tfidf,X_test_compound_norm,X_test_neg_norm,X_test_pos_norm,

print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

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

In [ ]:

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

X_tr_tfidf1 = hstack((X_train_tfidf, X_train.drop('process_review',axis=1).values)).toc
X_te_tfidf1 = hstack((X_test_tfidf,X_test.drop('process_review',axis=1).values)).tocsr(

print("Final Data matrix")
print(X_tr_tfidf1.shape, y_train.shape)
print(X_te_tfidf1.shape, y_test.shape)
print("="*100)
```

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

In [ ]:

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

4.3 Avg W2V

In [ ]:

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

In [ ]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this li
for sentence in tqdm(X_train['process_review'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|██████████| 94958/94958 [00:06<00:00, 14744.15it/s]

94958
300
```

In [ ]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this lis
for sentence in tqdm(X_test['process_review'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|██████████| 23740/23740 [00:01<00:00, 16173.68it/s]

23740
300
```

In [ ]:

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

In [ ]:

```
avg_w2v_vectors_train
```

Out[49]:

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

In [ ]:

```
X_tr_avgw2v = hstack((csr_matrix(avg_w2v_vectors_train),X_train_compound_norm,X_train_r
X_te_avgw2v = hstack((csr_matrix(avg_w2v_vectors_test),X_test_compound_norm,X_test_neg_

print("Final Data matrix")
print(X_tr_avgw2v.shape, y_train.shape)
print(X_te_avgw2v.shape, y_test.shape)
print("="*100)
```

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

4.4 TFIDF W2V

In [ ]:

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

In [ ]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_train['process_review'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((ser
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████| 94958/94958 [00:20<00:00, 4533.28it/s]

94958
300
```

In [ ]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this l
for sentence in tqdm(X_test['process_review'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((ser
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|██████████| 23740/23740 [00:05<00:00, 4147.91it/s]

23740
300
```

In [ ]:

```
tfidf_w2v_vectors_train=np.array(tfidf_w2v_vectors_train)
tfidf_w2v_vectors_test=np.array(tfidf_w2v_vectors_test)
```

In [ ]:

```
X_tr_tfidfw2v = hstack((csr_matrix(tfidf_w2v_vectors_train), X_train_compound_norm,X_tr
X_te_tfidfw2v = hstack((csr_matrix(tfidf_w2v_vectors_test),X_test_compound_norm,,X_test

print("Final Data matrix")
print(X_tr_avgw2v.shape, y_train.shape)
print(X_te_avgw2v.shape, y_test.shape)
print("="*100)
```

4.5 Tokenizer

In [ ]:

```
token=Tokenizer()
token.fit_on_texts(X_train['process_review'])
X_train_text_token=token.texts_to_sequences(X_train['process_review'])
X_test_text_token=token.texts_to_sequences(X_test['process_review'])
```

In [ ]:

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

```
Maximum sequence length is  183
```

In [ ]:

```
vocab_size_text=max(token.word_index.values())+1
X_train_text_padded=pad_sequences(X_train_text_token,maxlen=183,padding='post')
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 [ ]:

```
#Finding best alpha
classifier=MultinomialNB(class_prior=[0.5,0.5])
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
clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=T
clf.fit(X_tr_bow, y_train)

```

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 [ ]:

```
#Printing Best AUC score and best alpha
print("The best alpha value is ",clf.best_params_)
print("The best AUC score is ",clf.best_score_)


```

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

In [ ]:

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

In [ ]:

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

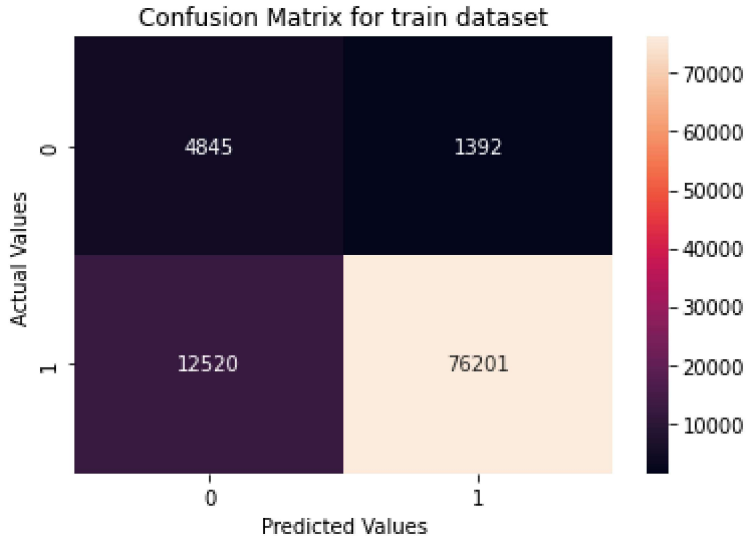
```
Accuracy for BOW Data for Naive Bayes 0.8557708508845829
ROC AUC score for BOW Data for Naive Bayes 0.8032480109486999
f1 score for BOW Data for Naive Bayes 0.9179683756588404
log loss for Bow data for Naive Bayes 4.981511837183812
```

In [ ]:

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

Out[60]:

```
Text(33.0, 0.5, 'Actual Values')
```

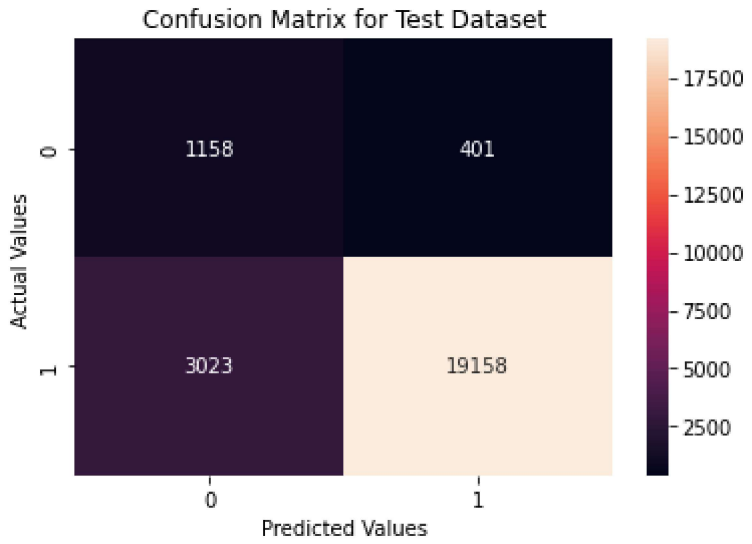


In [ ]:

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

Out[61]:

```
Text(33.0, 0.5, 'Actual Values')
```



Predicting 10 Sample points and comparing it with ground thruth

In [ ]:

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

5.1.2TFIDF

In [ ]:

```
#Finding best alpha
classifier=MultinomialNB(class_prior=[0.5,0.5])
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
clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=T
clf.fit(X_tr_tfidf, y_train)
classifier_tfidf= MultinomialNB(alpha=clf.best_params_['alpha'],class_prior=[0.5,0.5])
classifier_tfidf.fit(X_tr_tfidf, y_train)

y_pred=classifier_tfidf.predict(X_te_tfidf)
```

In [ ]:

```
print("Accuracy for Tfidf Data for Naive Bayes",accuracy_score(y_test,y_pred))
print("ROC AUC score for Tfidf Data for Naive Bayes",roc_auc_score(y_test,y_pred))
print("f1 score for Tfidf Data for Naive Bayes",f1_score(y_test,y_pred))
print("log loss for Tfidf data for Naive Bayes",log_loss(y_test,y_pred))
```

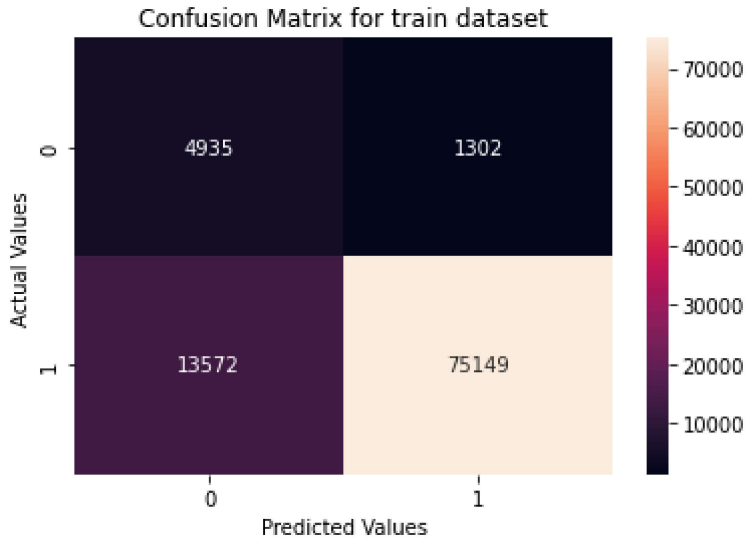
```
Accuracy for Tfidf Data for Naive Bayes 0.8612468407750632
ROC AUC score for Tfidf Data for Naive Bayes 0.8100547426316098
f1 score for Tfidf Data for Naive Bayes 0.9212790364209922
log loss for Tfidf data for Naive Bayes 4.79237740895698
```

In [ ]:

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

Out[66]:

```
Text(33.0, 0.5, 'Actual Values')
```

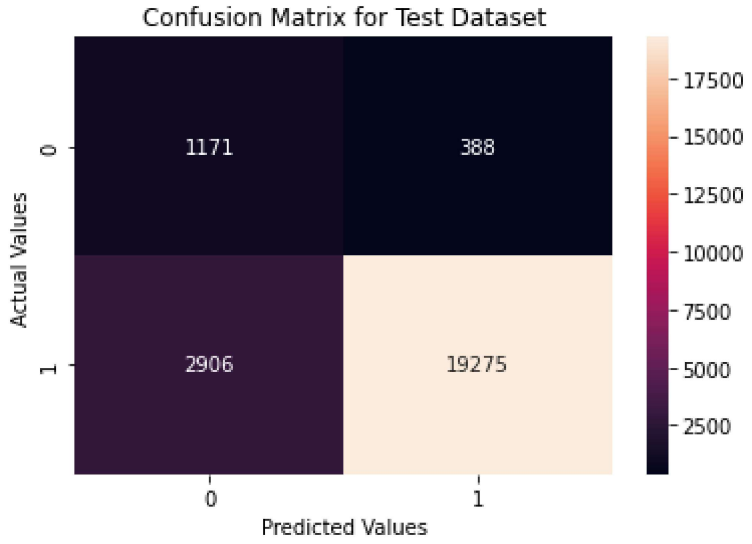


In [ ]:

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

Out[65]:

```
Text(33.0, 0.5, 'Actual Values')
```



Comparing Ground thruth and prediction

In [ ]:

```
y_pred_sample=y_pred[:10]
y_test_sample=y_test[:10].values
compare_df=pd.DataFrame({"Ground Thruth":y_test_sample,"Prediction":y_pred_sample})
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 Tfidf W2V can't be used with Naive Bayes as it contains negative values

In [ ]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model",  "Accuracy","AUC","F1 score"]
x.add_row(["BOW", "Naive Bayes", 0.844, 0.80,0.91])
x.add_row(["TFIDF", "Naive Bayes", 0.857, 0.80,0.91])

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 [ ]:

```
classifier=LogisticRegression()
parameters = {'C':[100, 10, 1.0, 0.1, 0.01]}
clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=T
clf.fit(X_tr_bow1, y_train)
classifier_bow_lr= LogisticRegression(C=clf.best_params_['C'])
classifier_bow_lr.fit(X_tr_bow1, y_train)
y_pred_train=classifier_bow_lr.predict(X_tr_bow1)
y_pred_test=classifier_bow_lr.predict(X_te_bow1)
```

In [ ]:

```
print("Accuracy for BOW Data for Logistic Regressions",accuracy_score(y_test,y_pred_tes
print("ROC AUC score for BOW Data for Logistic Regression",roc_auc_score(y_test,y_pred_
print("f1 score for BOW Data for Logistic Regression",f1_score(y_test,y_pred_test))
print("log loss for Bow data for Logistic Regression",log_loss(y_test,y_pred_test))
```

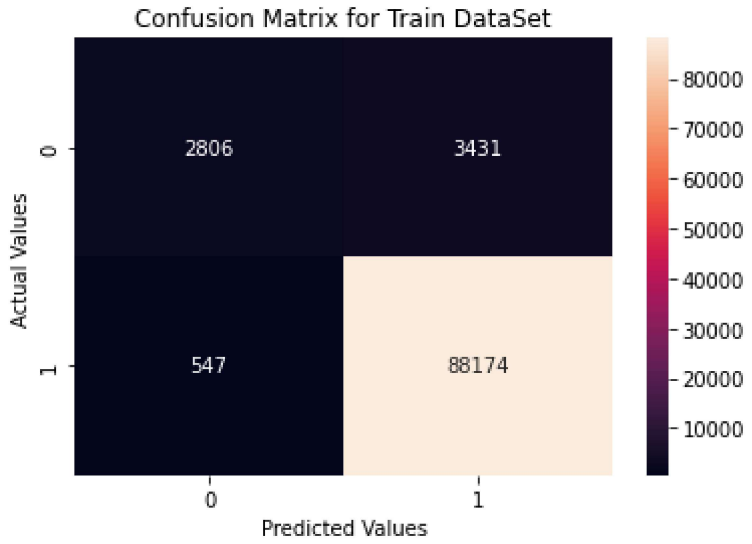
```
Accuracy for BOW Data for Logistic Regressions 0.9534540859309183
ROC AUC score for BOW Data for Logistic Regression 0.6998742979323502
f1 score for BOW Data for Logistic Regression 0.9755005210296434
log loss for Bow data for Logistic Regression 1.6076700060995952
```

In [ ]:

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

Out[70]:

```
Text(33.0, 0.5, 'Actual Values')
```

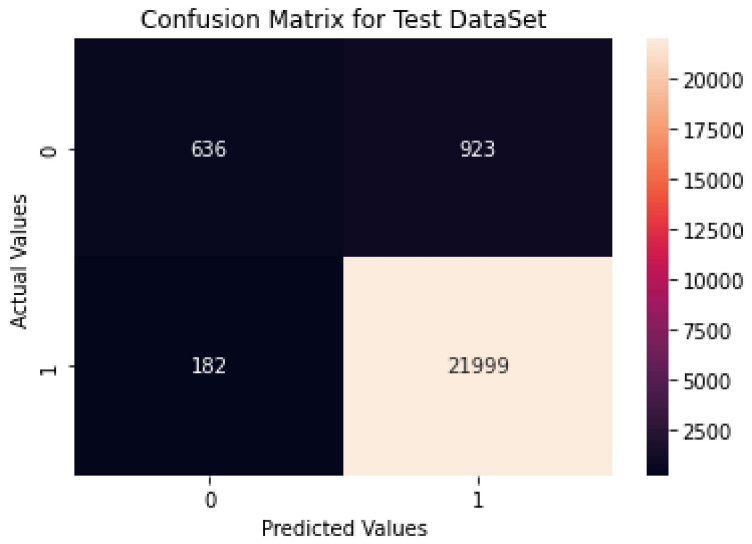


In [ ]:

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

Out[71]:

```
Text(33.0, 0.5, 'Actual Values')
```



In [ ]:

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

```
classifier=LogisticRegression()
parameters = {'C':[100, 10, 1.0, 0.1, 0.01]}
clf = GridSearchCV(classifier, parameters, cv=3, scoring='roc_auc',return_train_score=T
clf.fit(X_tr_tfidf1, y_train)
classifier_tfidf_lr= LogisticRegression(C=clf.best_params_['C'])
classifier_tfidf_lr.fit(X_tr_tfidf1, y_train)
y_pred_train=classifier_tfidf_lr.predict(X_tr_tfidf1)
y_pred_test=classifier_tfidf_lr.predict(X_te_tfidf1)
```

In [ ]:

```
print("Accuracy for Tfidf Data for Logistic Regressions",accuracy_score(y_test,y_pred_t
print("ROC AUC score for Tfidf Data for Logistic Regression",roc_auc_score(y_test,y_pre
print("f1 score for Tfidf Data for Logistic Regression",f1_score(y_test,y_pred_test))
print("log loss for Tfidf data for Logistic Regression",log_loss(y_test,y_pred_test))
```

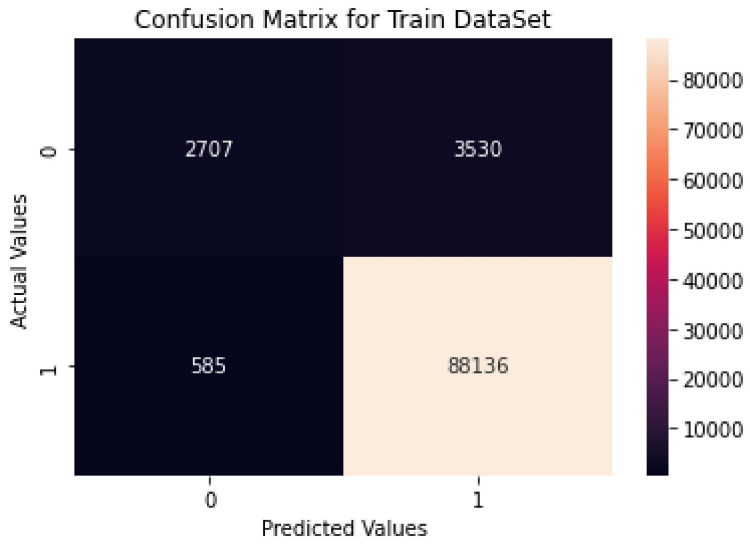
```
Accuracy for Tfidf Data for Logistic Regressions 0.9537489469250211
ROC AUC score for Tfidf Data for Logistic Regression 0.6958576183194425
f1 score for Tfidf Data for Logistic Regression 0.9756745978997741
log loss for Tfidf data for Logistic Regression 1.5974863396968844
```

In [ ]:

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

Out[75]:

```
Text(33.0, 0.5, 'Actual Values')
```

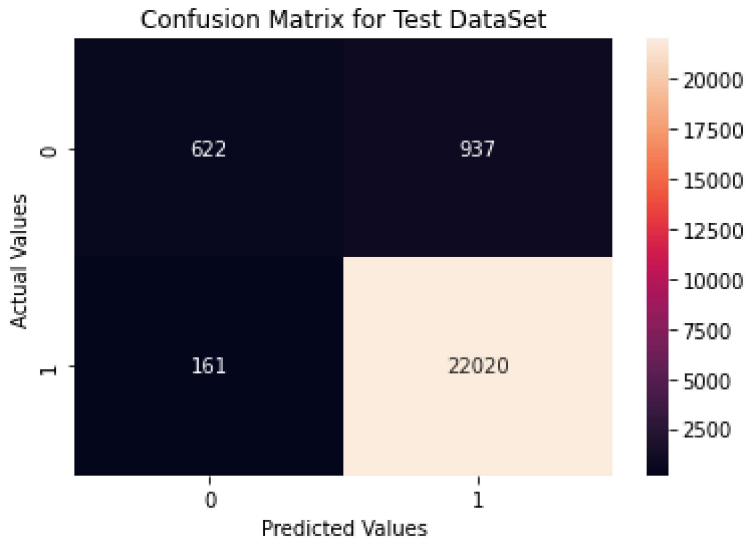


In [ ]:

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

Out[76]:

```
Text(33.0, 0.5, 'Actual Values')
```



In [ ]:

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

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

In [ ]:

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

In [ ]:

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

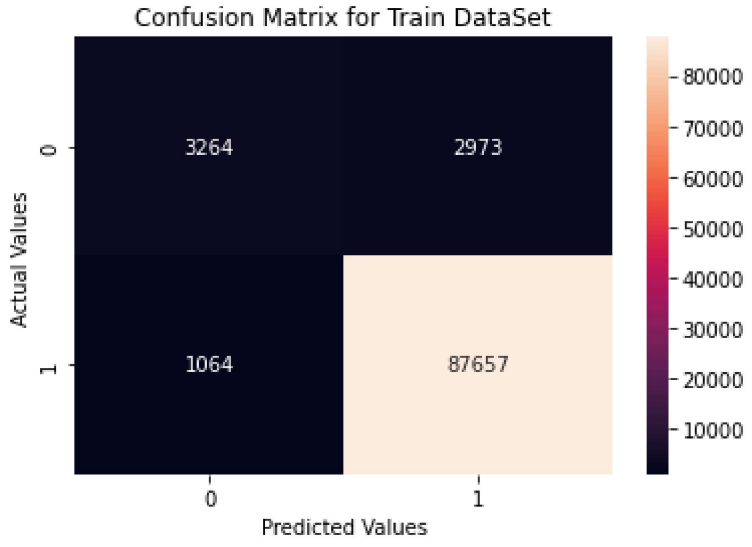
```
Accuracy for BOW Data for SVC 0.9449452401010952
ROC AUC score for BOW Data for SVC 0.7018807363605608
f1 score for BOW Data for SVC 0.9708629645317342
log loss for Bow data for SVC 1.9015543886029016
```

In [ ]:

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

Out[20]:

```
Text(33.0, 0.5, 'Actual Values')
```

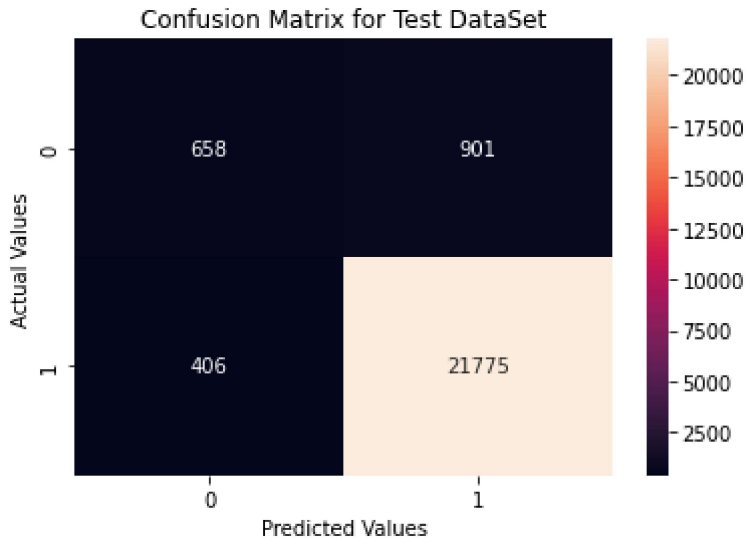


In [ ]:

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

Out[21]:

```
Text(33.0, 0.5, 'Actual Values')
```



In [ ]:

```
y_pred_sample=y_pred_test[:10]
y_test_sample=y_test[:10].values
compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
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 Tfidf

In [ ]:

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

In [ ]:

```
print("Accuracy for Tfidf Data for SVC",accuracy_score(y_test,y_pred_test))
print("ROC AUC score for tfidf Data for SVC",roc_auc_score(y_test,y_pred_test))
print("f1 score for tfidf Data for SVC",f1_score(y_test,y_pred_test))
print("log loss for tfidf Data for SVC",log_loss(y_test,y_pred_test))
```

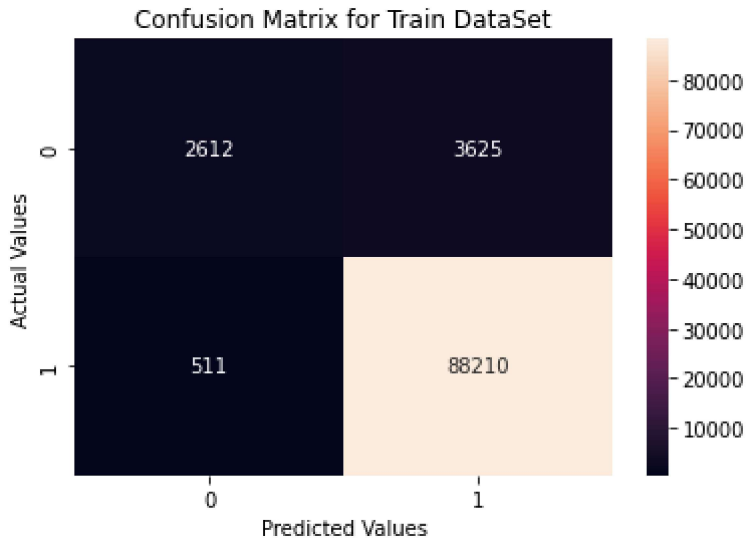
```
Accuracy for Tfidf Data for SVC 0.9530328559393428
ROC AUC score for tfidf Data for SVC 0.6856345798557029
f1 score for tfidf Data for SVC 0.9753247615464625
log loss for tfidf Data for SVC 1.6222203576172165
```

In [ ]:

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

Out[25]:

```
Text(33.0, 0.5, 'Actual Values')
```

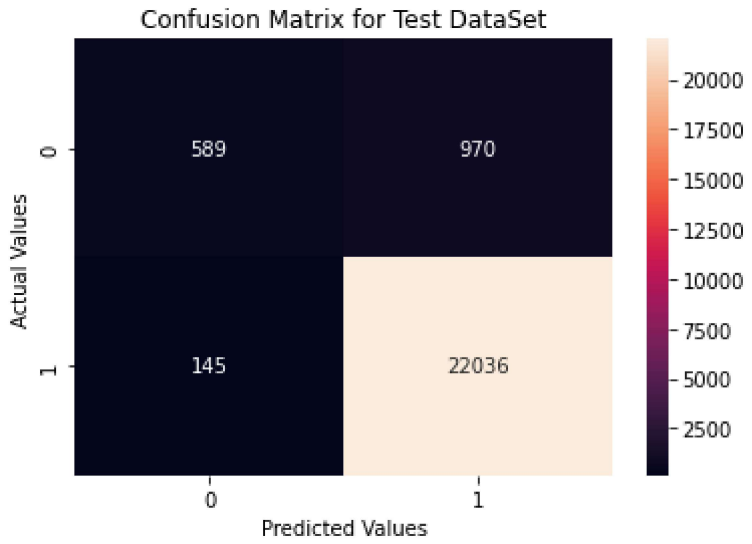


In [ ]:

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

Out[26]:

```
Text(33.0, 0.5, 'Actual Values')
```



In [ ]:

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

```
classifier=LGBMClassifier(boosting_type='gbdt')
parameters={'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],'n_estimators':[5,10,5
clf=RandomizedSearchCV(classifier,param_distributions=parameters,scoring='roc_auc',cv=3
clf.fit(X_tr_bow1, y_train)
classifier_bow_lgbm= LGBMClassifier(learning_rate=clf.best_params_['learning_rate'],n_e
classifier_bow_lgbm.fit(X_tr_bow1, y_train)
y_pred_train=classifier_bow_lgbm.predict(X_tr_bow1)
y_pred_test=classifier_bow_lgbm.predict(X_te_bow1)

```

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

In [ ]:

```
print("Accuracy for bow Data for SGD",accuracy_score(y_test,y_pred_test))
print("ROC AUC score for bow Data for SGD",roc_auc_score(y_test,y_pred_test))
print("f1 score for bow Data for SGD",f1_score(y_test,y_pred_test))
print("logloss for bow Data for SGD",log_loss(y_test,y_pred_test))
```

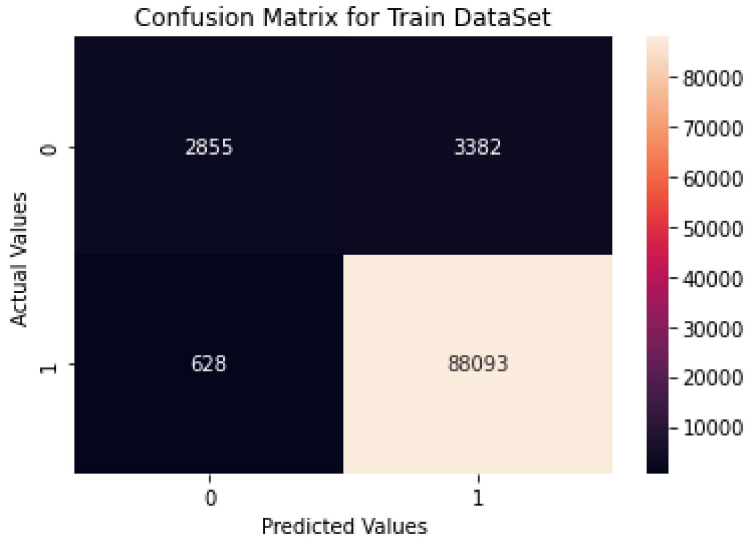
```
Accuracy for bow Data for SGD 0.9519797809604044
ROC AUC score for bow Data for SGD 0.6919290961449331
f1 score for bow Data for SGD 0.9747318024647574
logloss for bow Data for SGD 1.6585915041686854
```

In [ ]:

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

Out[30]:

```
Text(33.0, 0.5, 'Actual Values')
```

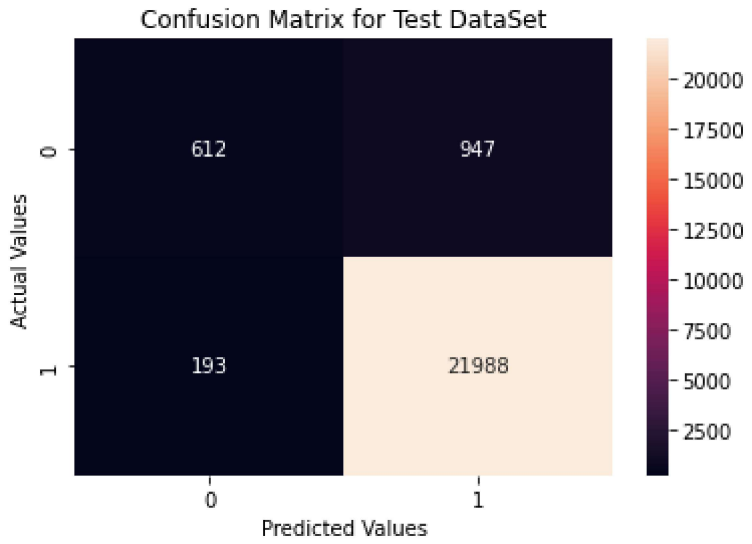


In [ ]:

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

Out[31]:

```
Text(33.0, 0.5, 'Actual Values')
```



In [ ]:

```
y_pred_sample=y_pred_test[:10]
y_test_sample=y_test[:10].values
compare_df=pd.DataFrame({"Ground Truth":y_test_sample,"Prediction":y_pred_sample})
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 Tfidf

In [ ]:

```
classifier=LGBMClassifier(boosting_type='gbdt')
parameters={'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],'n_estimators':[5,10,
clf=RandomizedSearchCV(classifier,param_distributions=parameters,scoring='roc_auc',cv=
clf.fit(X_tr_tfidf1, y_train)
classifier_tfidf_lgbm= LGBMClassifier(learning_rate=clf.best_params_['learning_rate'],
classifier_tfidf_lgbm.fit(X_tr_tfidf1, y_train)
y_pred_train=classifier_tfidf_lgbm.predict(X_tr_tfidf1)
y_pred_test=classifier_tfidf_lgbm.predict(X_te_tfidf1)
```

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

In [ ]:

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

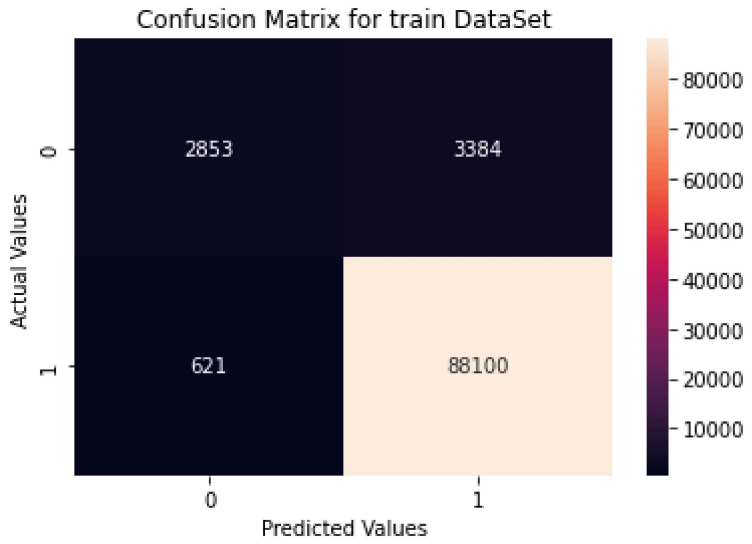
```
Accuracy for tfidf Data for SGD 0.951642796967144
ROC AUC score for tfidf Data for SGD 0.6908542318418884
f1 score for tfidf Data for SGD 0.9745533537261162
logloss for tfidf Data for SGD 1.6702306200048933
```

In [ ]:

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

Out[35]:

```
Text(33.0, 0.5, 'Actual Values')
```

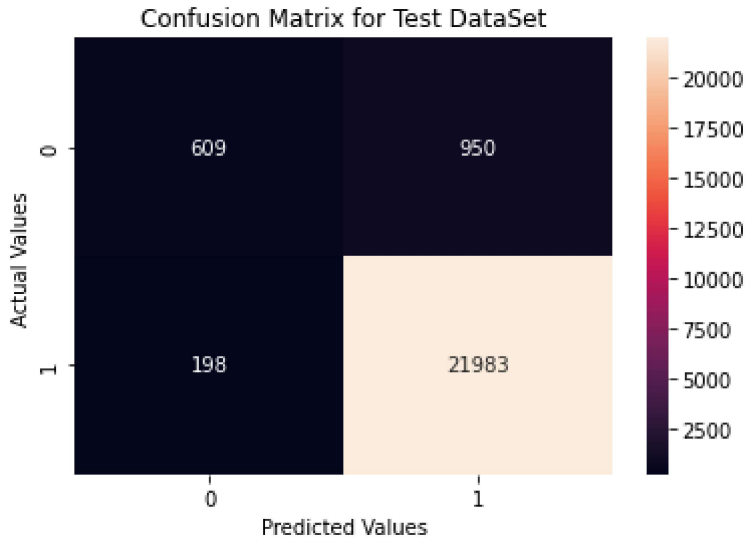


In [ ]:

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

Out[36]:

```
Text(33.0, 0.5, 'Actual Values')
```



In [ ]:

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

```
classifier_bow_rf= RandomForestClassifier(n_estimators=100,max_depth=250)
classifier_bow_rf.fit(X_tr_bow1, y_train)
y_pred_train=classifier_bow_rf.predict(X_tr_bow1)
y_pred_test=classifier_bow_rf.predict(X_te_bow1)

```

In [ ]:

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

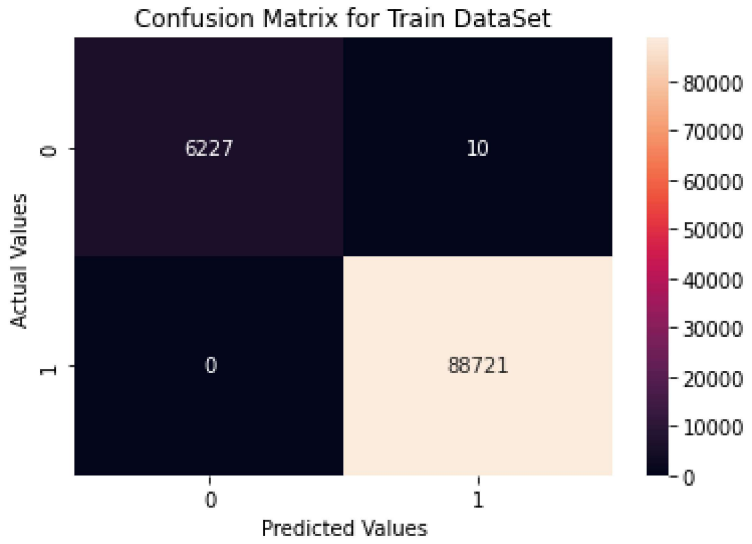
```
Accuracy for bow Data for Random Forest 0.9504633529907329
ROC AUC score for bow Data for Random Forest 0.6493728676187593
f1 score for bow Data for Random Forest 0.9740740740740741
log loss for bow data for Random Forest 1.7109717861340183
```

In [ ]:

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

Out[41]:

```
Text(33.0, 0.5, 'Actual Values')
```

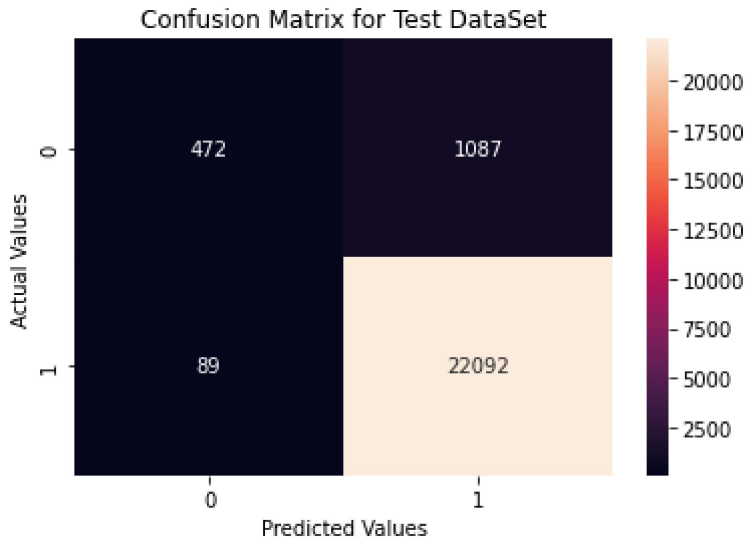


In [ ]:

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

Out[40]:

```
Text(33.0, 0.5, 'Actual Values')
```



In [ ]:

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

5.5.2 TFIDF

In [ ]:

```
classifier_tfidf_rf= RandomForestClassifier(n_estimators=100,max_depth=250)
classifier_tfidf_rf.fit(X_tr_tfidf1, y_train)
y_pred_train=classifier_tfidf_rf.predict(X_tr_tfidf1)
y_pred_test=classifier_tfidf_rf.predict(X_te_tfidf1)

```

In [ ]:

```
print("Accuracy for Tfidf Data for Random Forest",accuracy_score(y_test,y_pred_test))
print("ROC AUC score for Tfidf Data for Random Forest",roc_auc_score(y_test,y_pred_test
print("f1 score for Tfidf Data for Random Forest",f1_score(y_test,y_pred_test))
print("log loss for Tfidf Data for Random Forest", log_loss(y_test,y_pred_test))
```

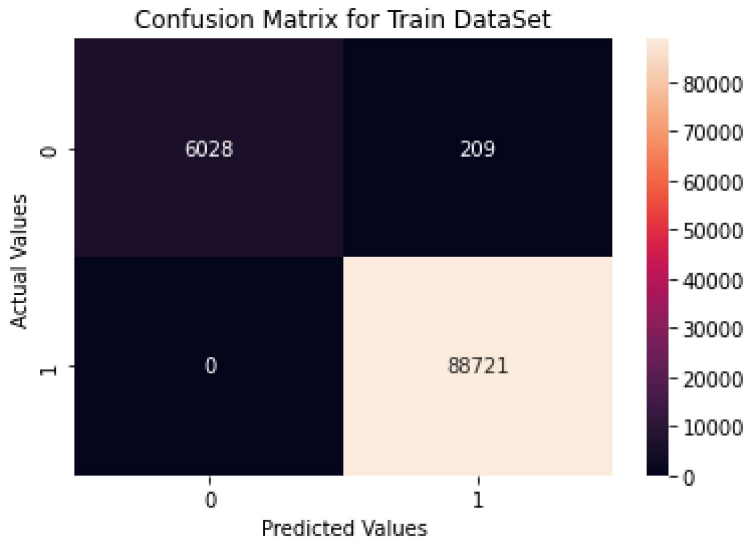
```
Accuracy for Tfidf Data for Random Forest 0.9503791069924179
ROC AUC score for Tfidf Data for Random Forest 0.6466441946410977
f1 score for Tfidf Data for Random Forest 0.9740391396332861
log loss for Tfidf Data for Random Forest 1.713881842964966
```

In [ ]:

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

Out[45]:

```
Text(33.0, 0.5, 'Actual Values')
```

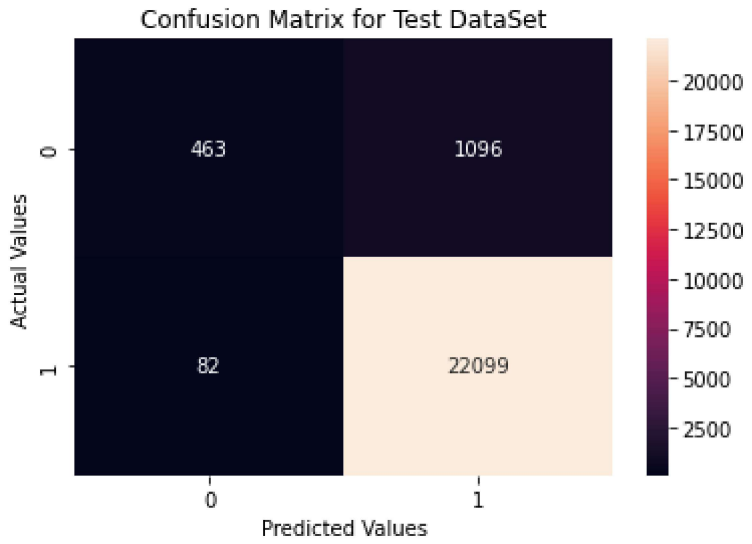


In [ ]:

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

Out[46]:

```
Text(33.0, 0.5, 'Actual Values')
```



5.6 MLP

In [ ]:

```
#https://medium.com/@thongonary/how-to-compute-f1-score-for-each-epoch-in-keras-a1acd17
class Metrics(tf.keras.callbacks.Callback):
  def __init__(self):
        self.validation_data = (X_te_bow1,y_test)
  def on_train_begin(self, logs={}):
    self.val_f1s = []
    self.val_auc=[]

  def on_epoch_end(self, epoch, logs={}):
    val_predict = (np.asarray(self.model.predict(self.validation_data[0]))).round()
    val_targ = self.validation_data[1]
    _val_f1 = f1_score(val_targ, val_predict)
    _val_auc=roc_auc_score(val_targ,val_predict)
    self.val_auc.append(_val_auc)
    self.val_f1s.append(_val_f1)
    print(" — val_f1: %f — val_auc: %f" %(_val_f1,_val_auc))
```

5.6.1 BOW

In [ ]:

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

In [ ]:

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

In [ ]:

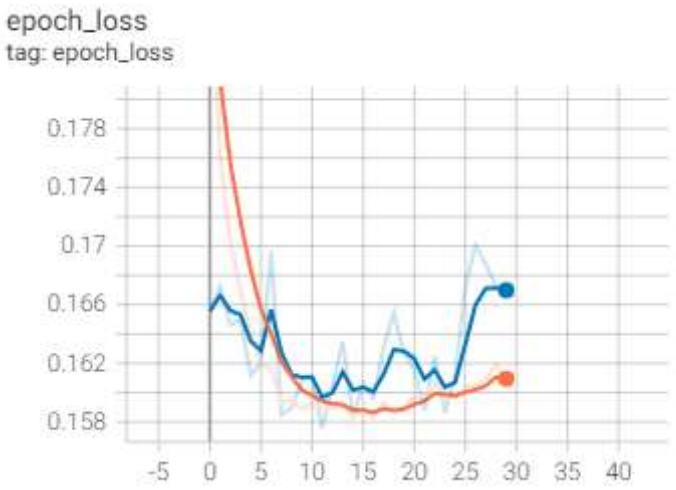
```
metrics=Metrics()
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
                             save_best_only=True, mode='auto')

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,
callbacks_list=[metrics,checkpoint,tensorboard_callback]
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 - accu
racy: 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 - accu
racy: 0.9515 — val_f1: 0.975107 — val_auc: 0.676739

Epoch 2: val_accuracy improved from 0.95084 to 0.95257, saving model to mo
del_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
```

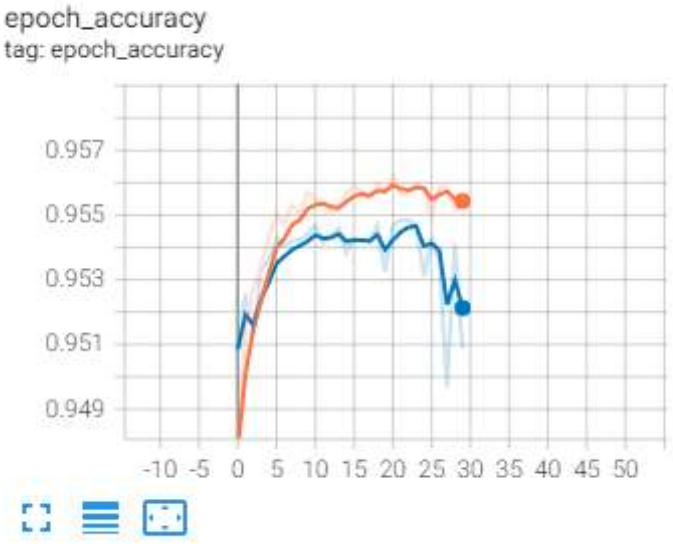
In [ ]:

```
%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 [ ]:

```
#https://medium.com/@thongonary/how-to-compute-f1-score-for-each-epoch-in-keras-a1acd17
class Metrics(tf.keras.callbacks.Callback):
  def __init__(self):
        self.validation_data = (X_te_tfidf1,y_test)
  def on_train_begin(self, logs={}):
    self.val_f1s = []
    self.val_auc=[]

  def on_epoch_end(self, epoch, logs={}):
    val_predict = (np.asarray(self.model.predict(self.validation_data[0]))).round()
    val_targ = self.validation_data[1]
    _val_f1 = f1_score(val_targ, val_predict)
    _val_auc=roc_auc_score(val_targ,val_predict)
    self.val_auc.append(_val_auc)
    self.val_f1s.append(_val_f1)
    print(" — val_f1: %f — val_auc: %f" %(_val_f1,_val_auc))
```

In [ ]:

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

In [ ]:

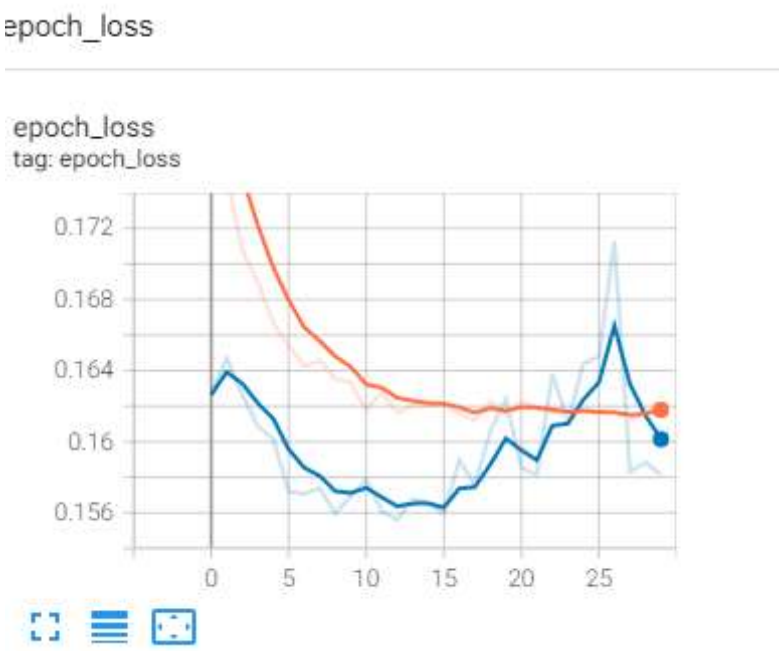
```
metrics=Metrics()
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
                             save_best_only=True, mode='auto')

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,
callbacks_list=[metrics,checkpoint,tensorboard_callback]
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_en
d` time: 0.0046s). Check your callbacks.
5930/5935 [============================>.] - ETA: 0s - loss: 0.1864 - accu
racy: 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 - accu
racy: 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 [ ]:

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

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

<IPython.core.display.Javascript object>
```



)

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 [ ]:

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

In [ ]:

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

Building Model

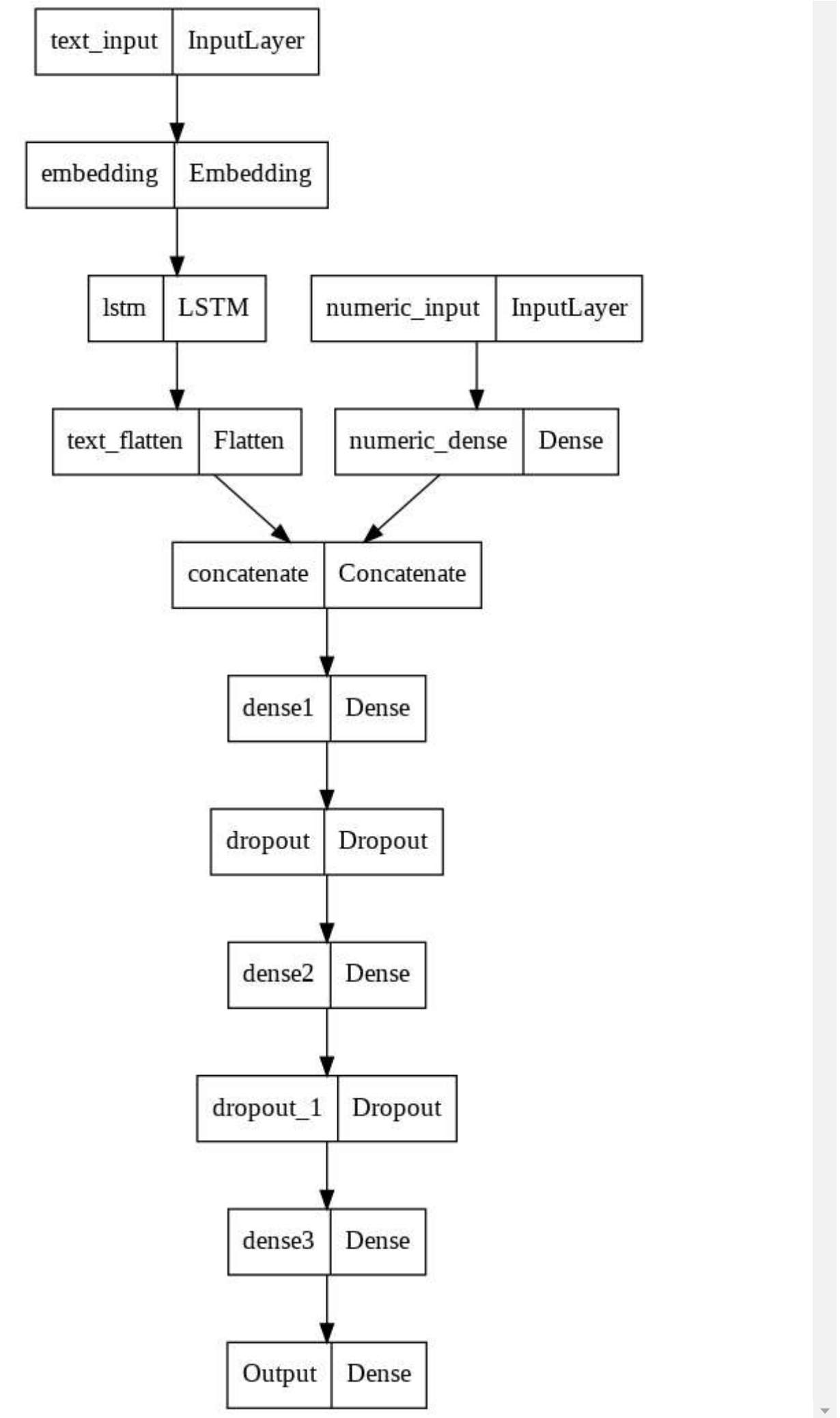
In [ ]:

```
#Model
#Text Model
pad_length=183
input_text=Input(shape=(pad_length,),name='text_input')
embedd_text=Embedding(input_dim=vocab_size_text,output_dim=300,
                      input_length=pad_length,weights=[embedding_matrix],trainable=Fals
lstm=LSTM(units=100,return_sequences=True)(embedd_text)
flatten_text=Flatten(name='text_flatten')(lstm)
text_model=Model(input_text,flatten_text)

#Numerical
input_numeric=Input(shape=(5,),name='numeric_input')
dense_numeric=Dense(units=16,activation='relu',kernel_initializer="glorot_uniform",name
numeric_model=Model(input_numeric,dense_numeric)

#Concatenating
concatenate_layer=Concatenate(axis=-1)([text_model.output,numeric_model.output])

#dense Layer1 after concatenate
dense1=Dense(units=256,activation='relu',kernel_initializer="glorot_uniform",
             name='dense1')(concatenate_layer)

#Dropout Layer1
dropout1=Dropout(0.3)(dense1)

#Dense layer2 after concatenate
dense2=Dense(units=128,activation='relu',kernel_initializer="glorot_uniform",
             name='dense2')(dropout1)

#Dropout Layer2
dropout2=Dropout(0.3)(dense2)

#Dense Layer3
dense3=Dense(units=64,activation='relu',kernel_initializer="glorot_uniform",
            name='dense3')(dropout2)

#Output Layer
output=Dense(units=2,activation='softmax',kernel_initializer="glorot_uniform",name="Out

model1=Model([text_model.input,numeric_model.input],output)
```

In [ ]:

```
1 plot_model(model1)
```

Out[22]:


text_input
InputLayer
embedding
Embedding
lstm
LSTM
numeric_input
InputLayer
text_flatten
Flatten
numeric_dense
Dense
concatenate
Concatenate
dense1
Dense
dropout
Dropout
dense2
Dense
dropout_1
Dropout
dense3
Dense
Output
Dense

Compiling the model

In [ ]:

```
from tensorflow.keras.callbacks import Callback
class RocCallback(Callback):
    def __init__(self,training_data,validation_data):
        self.x = training_data[0]
        self.y = training_data[1]
        self.x_val = validation_data[0]
        self.y_val = validation_data[1]


    def on_train_begin(self, logs={}):
        return

    def on_train_end(self, logs={}):
        return

    def on_epoch_begin(self, epoch, logs={}):
        return

    def on_epoch_end(self, epoch, logs={}):
        y_pred_train = self.model.predict(self.x)
        roc_train = roc_auc_score(self.y, y_pred_train)
        y_pred_val = self.model.predict(self.x_val)
        roc_val = roc_auc_score(self.y_val, y_pred_val)
        print('\rroc-auc_train: %s - roc-auc_val: %s' % (str(round(roc_train,4)),str(rc
        return

    def on_batch_begin(self, batch, logs={}):
        return

    def on_batch_end(self, batch, logs={}):
        return

```

In [ ]:

```
#https://stackoverflow.com/questions/53500047/stop-training-in-keras-when-accuracy-is-c
class MyThresholdCallback(tf.keras.callbacks.Callback):
    def __init__(self, threshold):
        super(MyThresholdCallback, self).__init__()
        self.threshold = threshold

    def on_epoch_end(self, epoch, logs=None):
        accuracy = logs["val_accuracy"]
        if accuracy >= self.threshold:
            self.model.stop_training = True
```

In [ ]:

```
model1.compile(optimizer=Adam(learning_rate=1e-4),loss='categorical_crossentropy',
               metrics=['accuracy'])

X_train_complete=[np.array(X_train_text_padded),np.array(X_train_numeric_concatenate)]


X_test_complete=[np.array(X_test_text_padded),np.array(X_test_numeric_concatenate)]

filepath="model_save/weights_lstm-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
                             save_best_only=True, mode='auto')

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,

roc = RocCallback(training_data=(X_train_complete, y_train_encoded),validation_data=(X_
early_stop=MyThresholdCallback(threshold=0.956)
callbacks_list=[roc,checkpoint,early_stop,tensorboard_callback]

model1.fit(X_train_complete,y_train_encoded,epochs=50,validation_data=(X_test_complete,
           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
   4/2968 [..............................] - ETA: 1:02 - loss: 1.6093 - ac
curacy: 0.8516   WARNING:tensorflow:Callback method `on_train_batch_end` i
s 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 mo
del_save/weights_lstm-02-0.9150.hdf5
2968/2968 [==============================] - 58s 20ms/step - loss: 0.7385
```

In [ ]:

```
%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

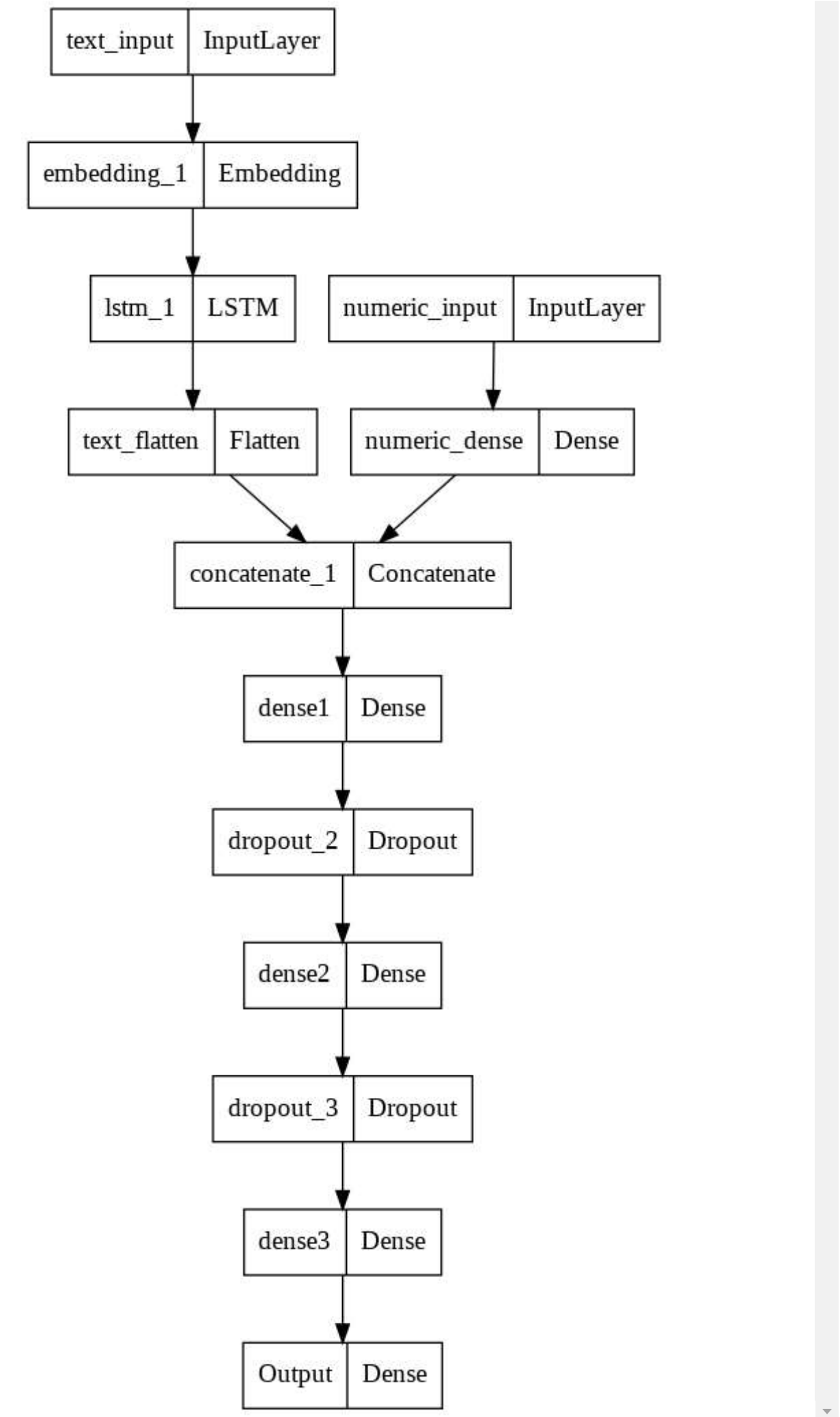
In [ ]:

```
#Model
#Text Model
pad_length=183
input_text=Input(shape=(pad_length,),name='text_input')
embedd_text=Embedding(input_dim=vocab_size_text,output_dim=300,
                      input_length=pad_length)(input_text)
lstm=LSTM(units=100,return_sequences=True)(embedd_text)
flatten_text=Flatten(name='text_flatten')(lstm)
text_model=Model(input_text,flatten_text)

#Numerical
input_numeric=Input(shape=(5,),name='numeric_input')
dense_numeric=Dense(units=16,activation='relu',kernel_initializer="glorot_uniform",name
numeric_model=Model(input_numeric,dense_numeric)

#Concatenating
concatenate_layer=Concatenate(axis=-1)([text_model.output,numeric_model.output])

#dense Layer1 after concatenate
dense1=Dense(units=256,activation='relu',kernel_initializer="glorot_uniform",
             name='dense1')(concatenate_layer)

#Dropout Layer1
dropout1=Dropout(0.3)(dense1)

#Dense layer2 after concatenate
dense2=Dense(units=128,activation='relu',kernel_initializer="glorot_uniform",
             name='dense2')(dropout1)

#Dropout Layer2
dropout2=Dropout(0.3)(dense2)

#Dense Layer3
dense3=Dense(units=64,activation='relu',kernel_initializer="glorot_uniform",
            name='dense3')(dropout2)

#Output Layer
output=Dense(units=2,activation='softmax',kernel_initializer="glorot_uniform",name="Out

model2=Model([text_model.input,numeric_model.input],output)
```

In [ ]:

```
1 plot_model(model2)
```

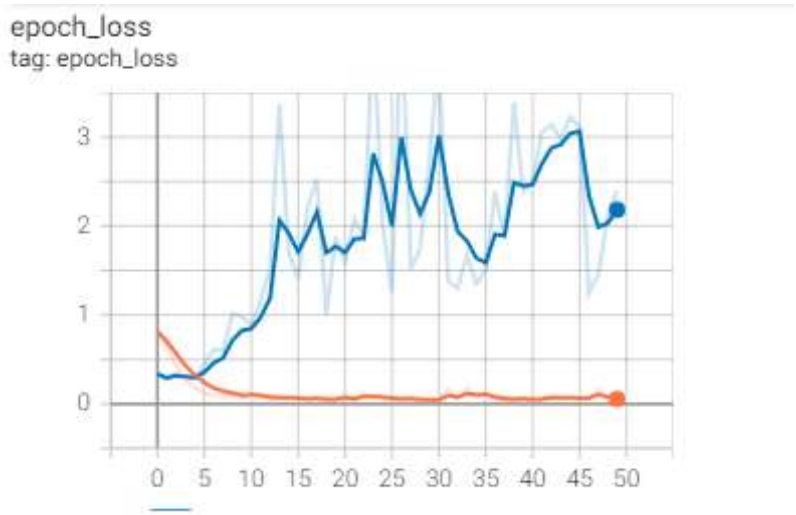
Out[31]:

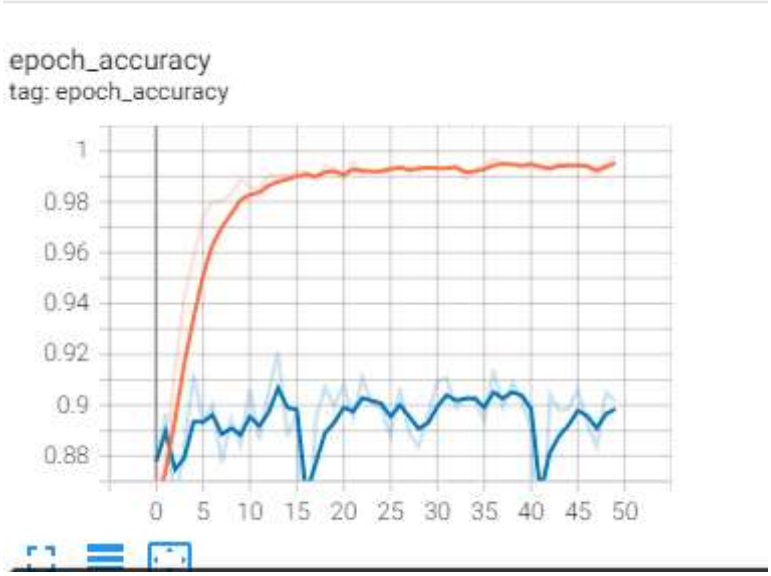

text_input
InputLayer
embedding_1
Embedding
lstm_1
LSTM
numeric_input
InputLayer
text_flatten
Flatten
numeric_dense
Dense
concatenate_1
Concatenate
dense1
Dense
dropout_2
Dropout
dense2
Dense
dropout_3
Dropout
dense3
Dense
Output
Dense

In [ ]:

```
model2.compile(optimizer=Adam(learning_rate=1e-3),loss='categorical_crossentropy',
               metrics=['accuracy'])

X_train_complete=[np.array(X_train_text_padded),np.array(X_train_numeric_concatenate)]


X_test_complete=[np.array(X_test_text_padded),np.array(X_test_numeric_concatenate)]


filepath="model_save/weights_lstm-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1,
                             save_best_only=True, mode='auto')

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir,histogram_freq=1,

roc = RocCallback(training_data=(X_train_complete, y_train_encoded),validation_data=(X_
early_stop=MyThresholdCallback(threshold=0.956)
callbacks_list=[roc,checkpoint,early_stop,tensorboard_callback]

model2.fit(X_train_complete,y_train_encoded,epochs=50,validation_data=(X_test_complete,
           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 - ac
curacy: 0.8594WARNING:tensorflow:Callback method `on_train_batch_end` is s
low 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 mo
del_save/weights_lstm-02-0.8968.hdf5
2968/2968 [==============================] - 197s 66ms/step - loss: 0.6361
```





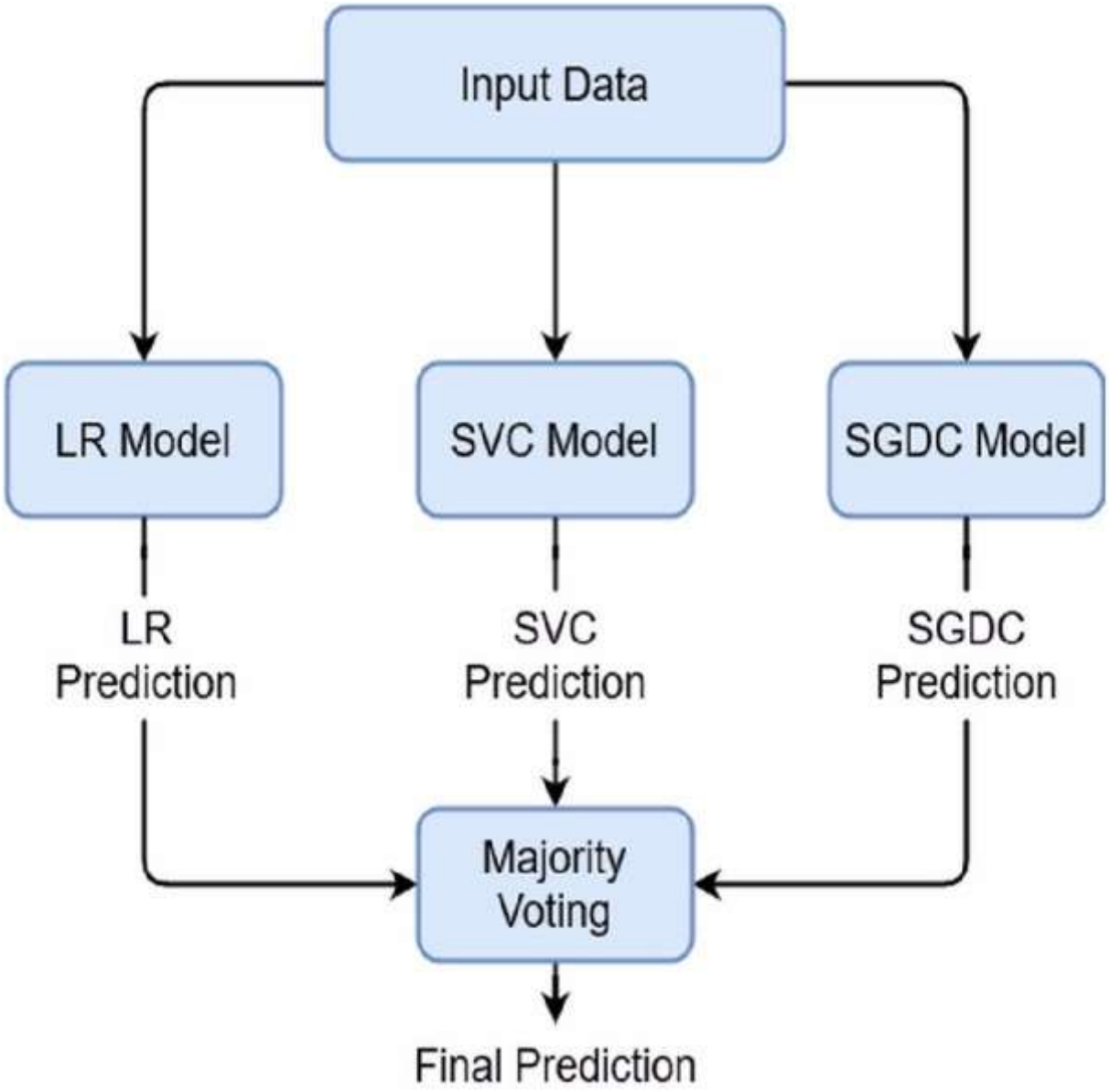
In [ ]:

```
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 [ ]:

```
#Prediction made by Logistic Regression model trained on Tfidf data
y_pred_lr=classifier_tfidf_lr.predict(X_te_tfidf1)

#Prediction made by SVC model trained on Tfidf data
y_pred_svc=classifier_tfidf_svc.predict(X_te_tfidf1)

#Prediction made by SGD model trained on Tfidf data
y_pred_sgd=classifier_tfidf_lgbm.predict(X_te_tfidf1)
```

In [ ]:

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

In [ ]:

```
y_pred_sgd.shape
```

Out[104]:

```
(23740,)
```

In [ ]:

```
#https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list/
from collections import Counter

def most_frequent(List):
    occurence_count = Counter(List)
    return occurence_count.most_common(1)[0][0]
```

In [ ]:

```
majority_class=[]
for i in range(y_pred_sgd.shape[0]):
  each_point_list=[]
  each_point_list.append(y_pred_lr[i])
  each_point_list.append(y_pred_svc[i])
  each_point_list.append(y_pred_sgd[i])
  majority_class.append(most_frequent(each_point_list))
```

In [ ]:

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

In [ ]:

```
print("Accuracy for tfidf Data for Stacking Classifier",accuracy_score(y_test,majority_
print("ROC AUC score for tfidf Data for Stacking Classifier",roc_auc_score(y_test,majo
print("f1 score for tfidf Data for Stacking Classifier",f1_score(y_test,majority_class
print("log loss for tfidf data for Stacking Classifier",f1_score(y_test,majority_class
```

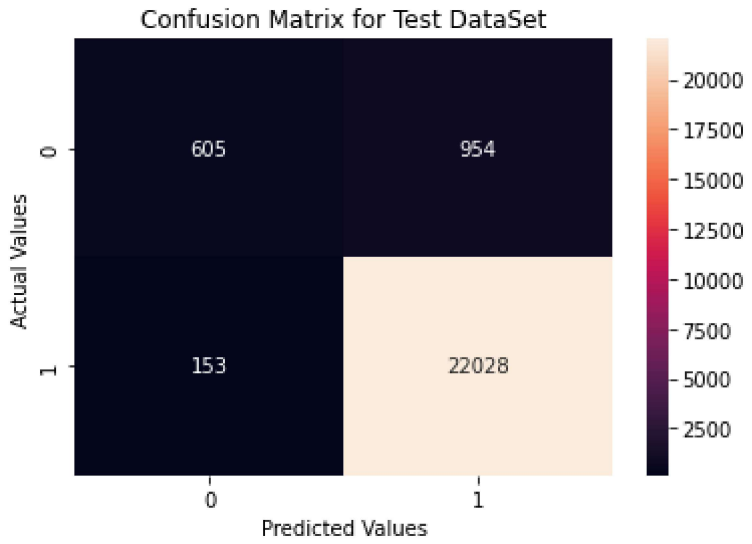
```
Accuracy for tfidf Data for Stacking Classifier 0.9533698399326032
ROC AUC score for tfidf Data for Stacking Classifier 0.6905857398829545
f1 score for tfidf Data for Stacking Classifier 0.9754887850674225
log loss for tfidf data for Stacking Classifier 0.9754887850674225
```

In [ ]:

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

Out[53]:

```
Text(33.0, 0.5, 'Actual Values')
```



6 Results & Conclusion

In [ ]:

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

In [ ]:

```
1 print(x)
```

```
+-------------------------------------+-------------------------+--------
-------+---------------+----------+----------+
|                Model                |       Vectorizer        | Test Ac
curacy | Test AUC Score | F1 Score | log loss |
+-------------------------------------+-------------------------+--------
-------+---------------+----------+----------+
|             Naive Bayes             |           BOW           |     85.
57%    |      0.8      |   0.92   |   4.98   |
|             Naive Bayes             |          TFIDF          |     86.
12%    |      0.81     |   0.92   |   4.79   |
|         logistic Regression         |           BOW           |     95.
34%    |      0.69     |   0.97   |   1.6    |
|         logistic Regression         |          TFIDF          |     95.
37%    |      0.69     |   0.97   |   1.59   |
|       Support vector Classifier     |           BOW           |     94.
49%    |      0.7      |   0.97   |   1.9    |
|       Support vector Classifier     |          TFIDF          |     95.
3%     |      0.68     |   0.97   |   1.62   |
| Stochastic Gradient Descent(LightGBM) |           BOW           |     95.
2%     |      0.69     |   0.97   |   1.65   |
| Stochastic Gradient Descent(LightGBM) |          TFIDF          |     95.
16%    |      0.69     |   0.97   |   1.67   |
|             Random Forest           |           BOW           |     95.
04%    |      0.65     |   0.97   |   1.71   |
|             Random Forest           |          TFIDF          |     95.
04%    |      0.64     |   0.97   |   1.71   |
|         Multi layer perceptron      |           BOW           |     95.
48%    |      0.71     |   0.97   |   0.16   |
|         Multi layer perceptron      |          TFIDF          |     95.
4%     |      0.7      |   0.97   |   0.15   |
|                LSTM                 | Pretrained Glove Vectors |     94.
63%    |      0.85     |    -     |  0.376   |
|                LSTM                 | Embedded training layer  |     92.
1%     |      0.83     |    -     |   3.37   |
|          Stacking Classifier        |          TFIDF          |     95.
3%     |      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
```