In [1]:
```
# importing the libaries

import pandas as pd
import numpy as np
import os

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns

from plotly import tools
import plotly.offline as py
import plotly.figure_factory as ff
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go

from nltk.corpus import stopwords
from nltk.util import ngrams

from wordcloud import WordCloud

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split , GridSearchCV , cross_val_
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.metrics import classification_report , confusion_matrix , f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix , roc_curve , accuracy_score , preci

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.naive_bayes import MultinomialNB , GaussianNB
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier

from collections import defaultdict
from collections import Counter
plt.style.use('ggplot')
stop=set(stopwords.words('english'))

import re
from nltk.tokenize import word_tokenize
import gensim
import string

from tqdm import tqdm
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding,LSTM,Dense,SpatialDropout1D
from keras.initializers import Constant
```

```
from keras.optimizers import Adam

import folium
from folium import plugins

from spellchecker import SpellChecker
```

```
Using TensorFlow backend.
```

In [2]:
```
# Load the train and test dataset

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
submission = pd.read_csv('submission.csv')
```

In [3]:
```
train.shape
```

Out[3]:
```
(7613, 5)
```

In [4]:
```
train.isnull().sum()
```

Out[4]:
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

In [5]:
```
train.head(10)
```

Out[5]:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1
5	8	NaN	NaN	#RockyFire Update => California Hwy. 20 closed...	1
6	10	NaN	NaN	#flood #disaster Heavy rain causes flash flood...	1
7	13	NaN	NaN	I'm on top of the hill and I can see a fire in...	1
8	14	NaN	NaN	There's an emergency evacuation happening now ...	1
9	15	NaN	NaN	I'm afraid that the tornado is coming to our a...	1

In [6]:
```
new_train = train
new_train['keyword'].fillna("No keyword",inplace=True)
new_train.isnull().sum()
```

Out[6]:
```
id             0
keyword        0
location    2533
text           0
target         0
dtype: int64
```

In [7]:
```
# drop the location column both from train and test part

# train.drop(['location'] , axis = 1 , inplace = True)
# test.drop(['location'] , axis = 1 , inplace = True)
```

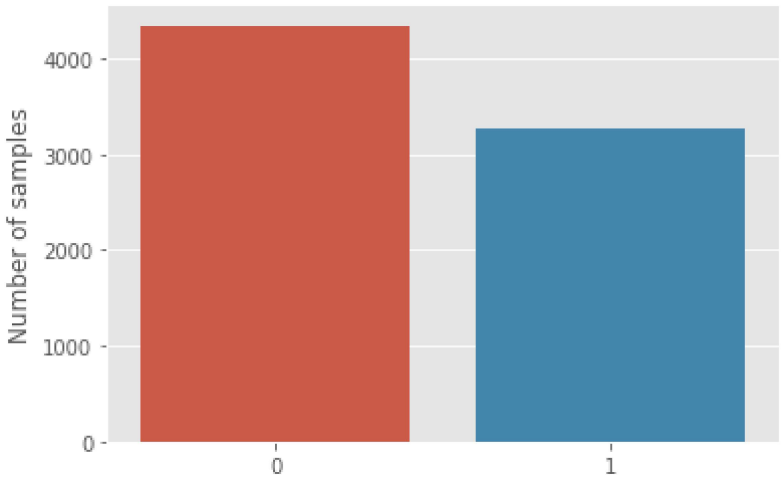
In [8]:
```
train.shape
```

Out[8]:
```
(7613, 5)
```

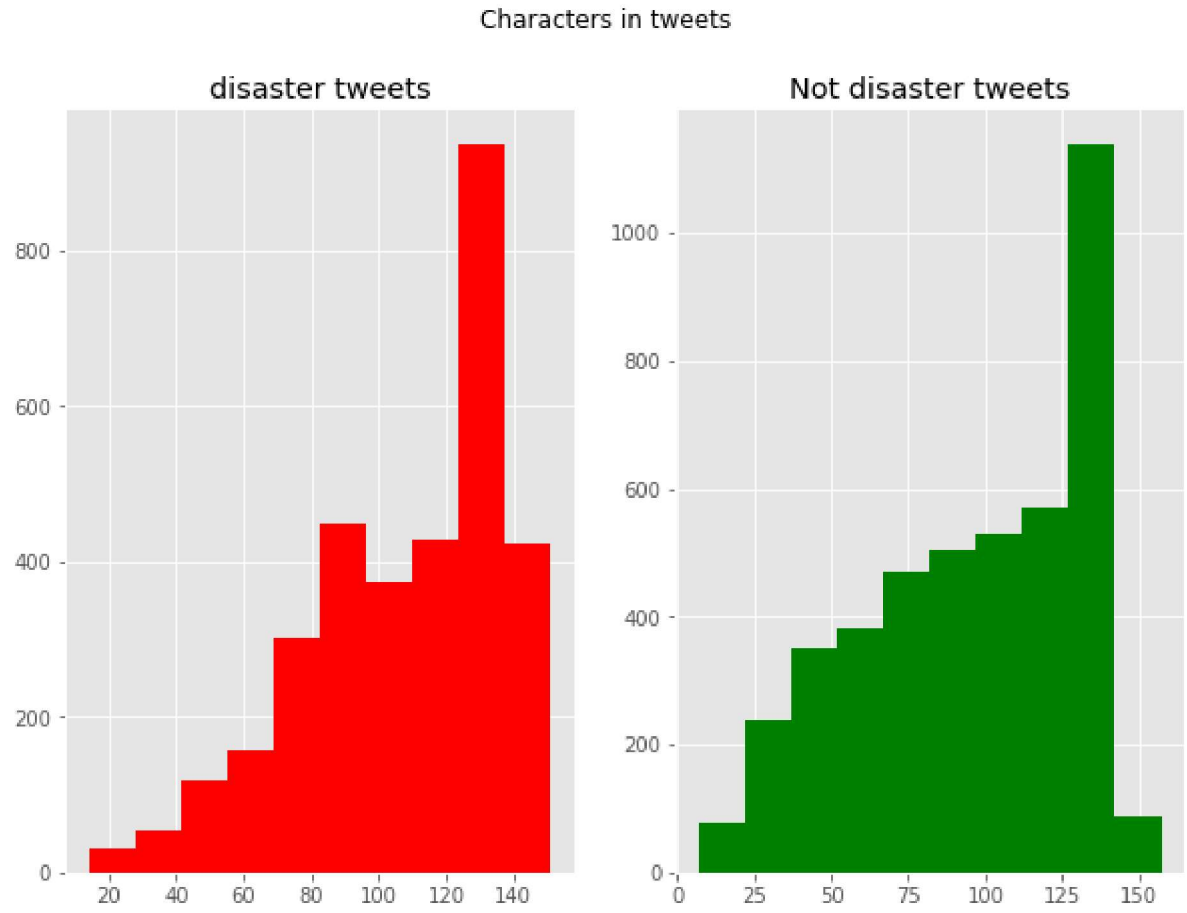
EDA

In [9]:
```
x=train.target.value_counts()
sns.barplot(x.index,x)
plt.gca().set_ylabel('Number of samples')
```

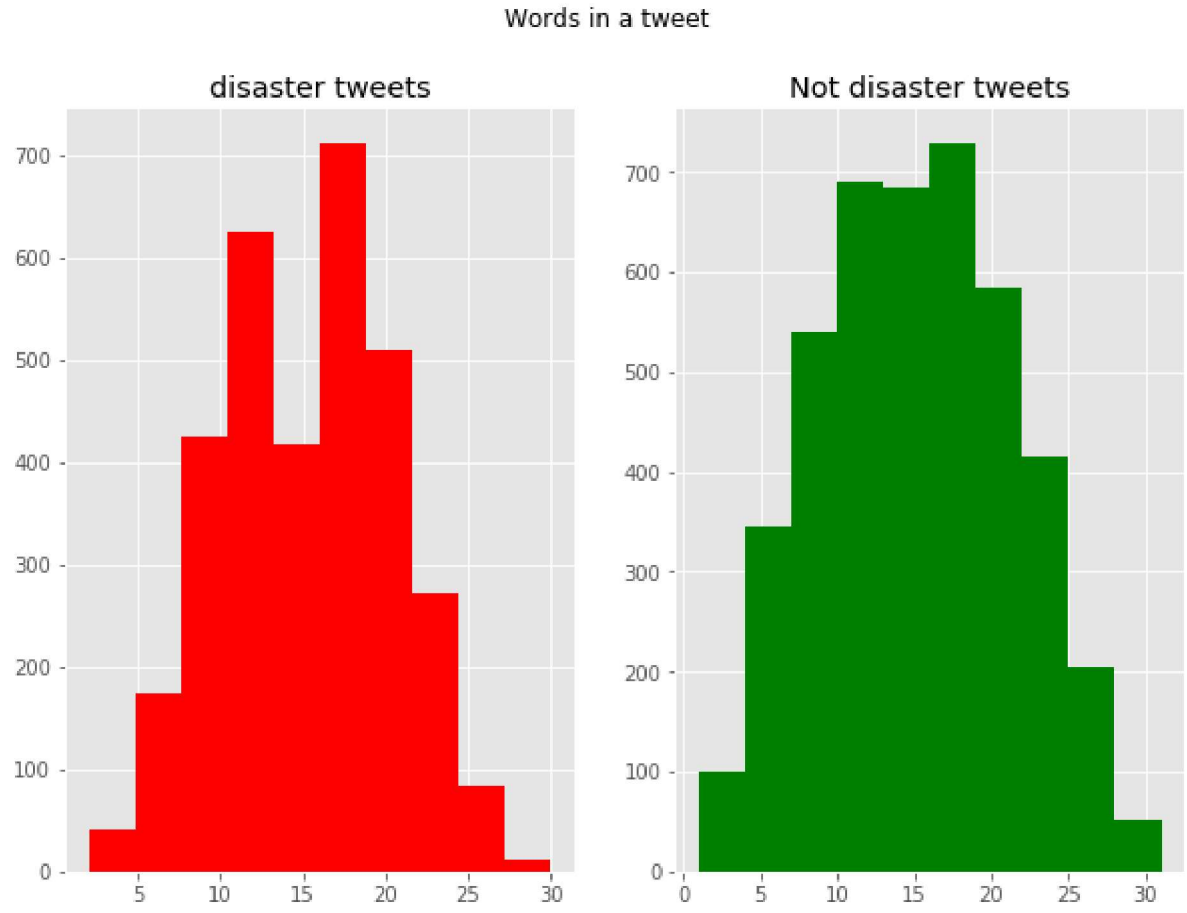
Out[9]:
```
Text(0, 0.5, 'Number of samples')
```



In [10]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,7))
tweet_len=train[train['target']==1]['text'].str.len()
ax1.hist(tweet_len,color='red')
ax1.set_title('disaster tweets')
tweet_len=train[train['target']==0]['text'].str.len()
ax2.hist(tweet_len,color='green')
ax2.set_title('Not disaster tweets')
fig.suptitle('Characters in tweets')
plt.show()
```

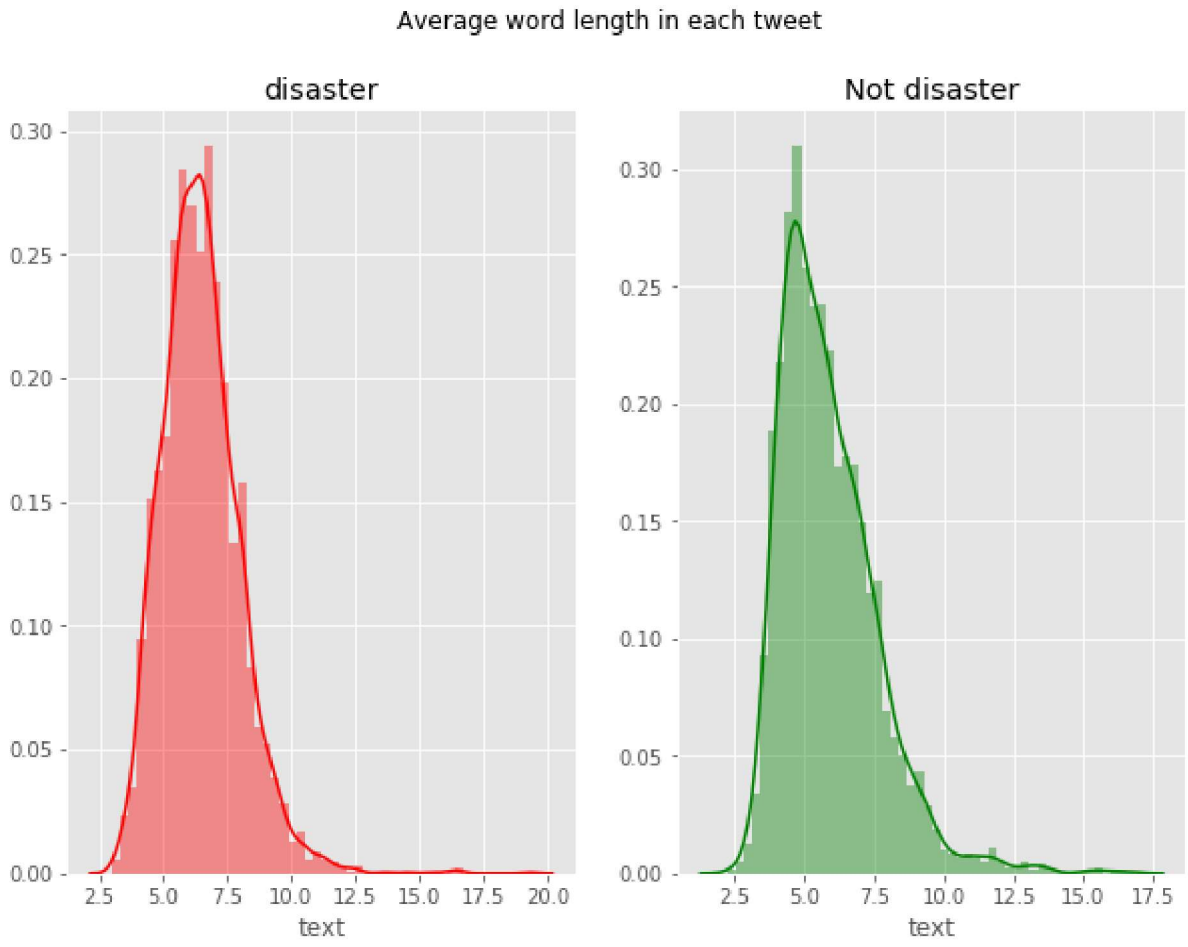


In [11]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,7))
tweet_len=train[train['target']==1]['text'].str.split().map(lambda x: len(x))
ax1.hist(tweet_len,color='red')
ax1.set_title('disaster tweets')
tweet_len=train[train['target']==0]['text'].str.split().map(lambda x: len(x))
ax2.hist(tweet_len,color='green')
ax2.set_title('Not disaster tweets')
fig.suptitle('Words in a tweet')
plt.show()
```



In [12]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(10,7))
word=train[train['target']==1]['text'].str.split().apply(lambda x : [len(i) for
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax1,color='red')
ax1.set_title('disaster')
word=train[train['target']==0]['text'].str.split().apply(lambda x : [len(i) for
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax2,color='green')
ax2.set_title('Not disaster')
fig.suptitle('Average word length in each tweet')
```

Out[12]:
```
Text(0.5, 0.98, 'Average word length in each tweet')
```



In [13]:
```
# Tweets per location

cnt_ = train['location'].value_counts()
cnt_.reset_index()
cnt_ = cnt_[:20,]
trace1 = go.Bar(
                x = cnt_.index,
                y = cnt_.values,
                name = "Number of tweets in dataset according to location",
                marker = dict(color = 'rgba(200, 74, 55, 0.5)',
                              line=dict(color='rgb(0,0,0)',width=1.5)),
                )

data = [trace1]
layout = go.Layout(barmode = "group",title = 'Number of tweets depending on loca
fig = go.Figure(data = data, layout = layout)
py.iplot(fig)
```

In [14]:
```
#Number of tweets depending on location per class

train1_df = train[train["target"]==1]
train0_df = train[train["target"]==0]
cnt_1 = train1_df['location'].value_counts()
cnt_1.reset_index()
cnt_1 = cnt_1[:20,]

cnt_0 = train0_df['location'].value_counts()
cnt_0.reset_index()
cnt_0 = cnt_0[:20,]

trace1 = go.Bar(
                x = cnt_1.index,
                y = cnt_1.values,
                name = "Number of real disaster tweets",
                marker = dict(color = 'rgba(255, 74, 55, 0.5)',
                              line=dict(color='rgb(0,0,0)',width=1.5)),
                )
trace0 = go.Bar(
                x = cnt_0.index,
                y = cnt_0.values,
                name = "Number of unreal disaster tweets",
                marker = dict(color = 'rgba(79, 82, 97, 0.5)',
                              line=dict(color='rgb(0,0,0)',width=1.5)),
                )


data = [trace0,trace1]
layout = go.Layout(barmode = 'stack',title = 'Number of tweets depending on loca
fig = go.Figure(data = data, layout = layout)
py.iplot(fig)
```

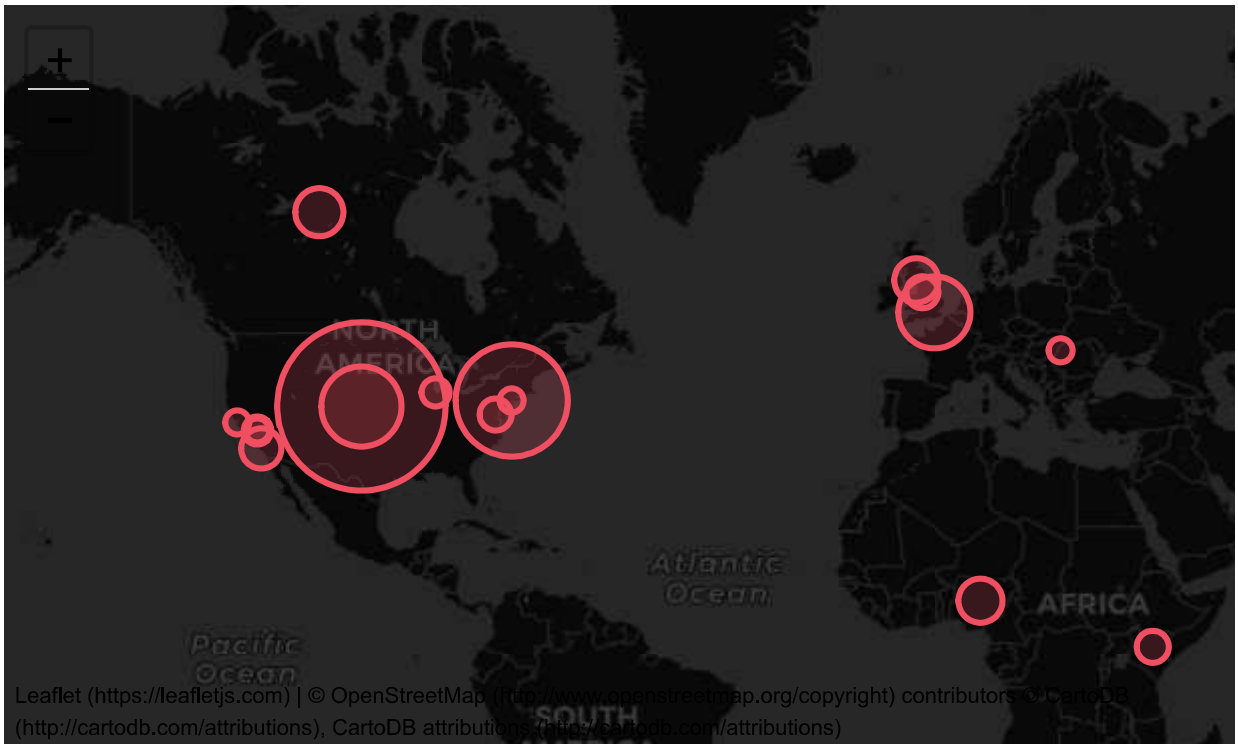
Visualize per location using map

In [15]:
```
from geopy.geocoders import Nominatim
from geopy.extra.rate_limiter import RateLimiter


df = train['location'].value_counts()[:20,]
df = pd.DataFrame(df)
df = df.reset_index()
df.columns = ['location', 'counts']
geolocator = Nominatim(user_agent="specify_your_app_name_here")
geocode = RateLimiter(geolocator.geocode, min_delay_seconds=1)
dictt_latitude = {}
dictt_longitude = {}
for i in df['location'].values:
    print(i)
    location = geocode(i)
    dictt_latitude[i] = location.latitude
    dictt_longitude[i] = location.longitude
df['latitude']= df['location'].map(dictt_latitude)
df['longitude'] = df['location'].map(dictt_longitude)
```

```
USA
New York
United States
London
Canada
Nigeria
UK
Los Angeles, CA
India
Mumbai
Washington, DC
Kenya
Worldwide
Australia
Chicago, IL
California
Everywhere
New York, NY
California, USA
San Francisco
```

In [16]:
```
map1 = folium.Map(location=[10.0, 10.0], tiles='CartoDB dark_matter', zoom_start=
markers = []
for i, row in df.iterrows():
    loss = row['counts']
    if row['counts'] > 0:
        count = row['counts']*0.4
    folium.CircleMarker([float(row['latitude']), float(row['longitude'])], radius=
map1
```

Out[16]:



It is clear that most of the tweets from america region.

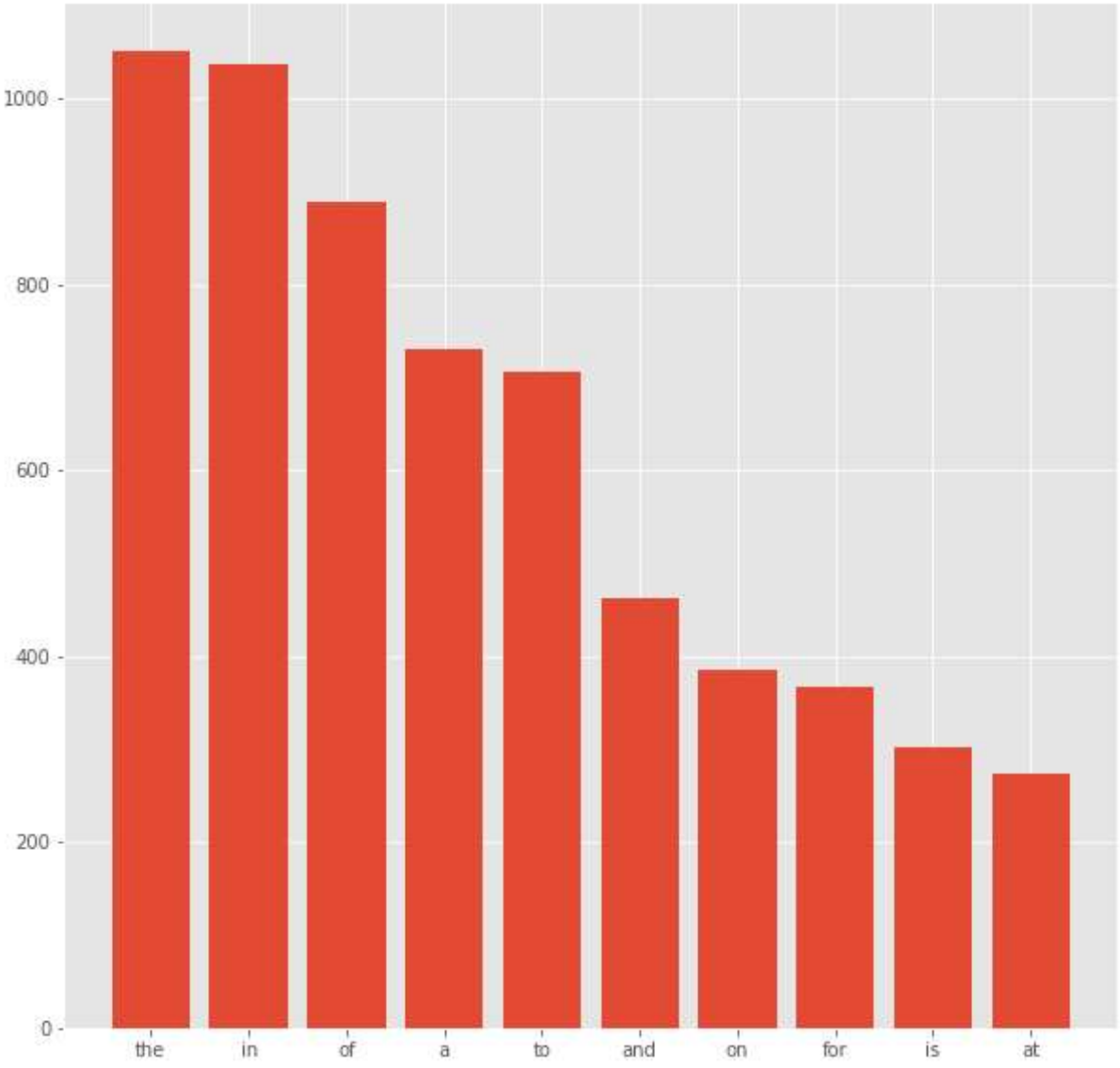
Common Stopwords in tweets

In [17]:
```
def create_corpus(target):
    corpus =[]

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

    return corpus
```

In [18]:
```
corpus = create_corpus(1)
```

In [19]:
```
dic = defaultdict(int)
for word in corpus:
    if word in stop:
        dic[word]+=1

top = sorted(dic.items(), key = lambda x:x[1] , reverse = True)[:10]
```

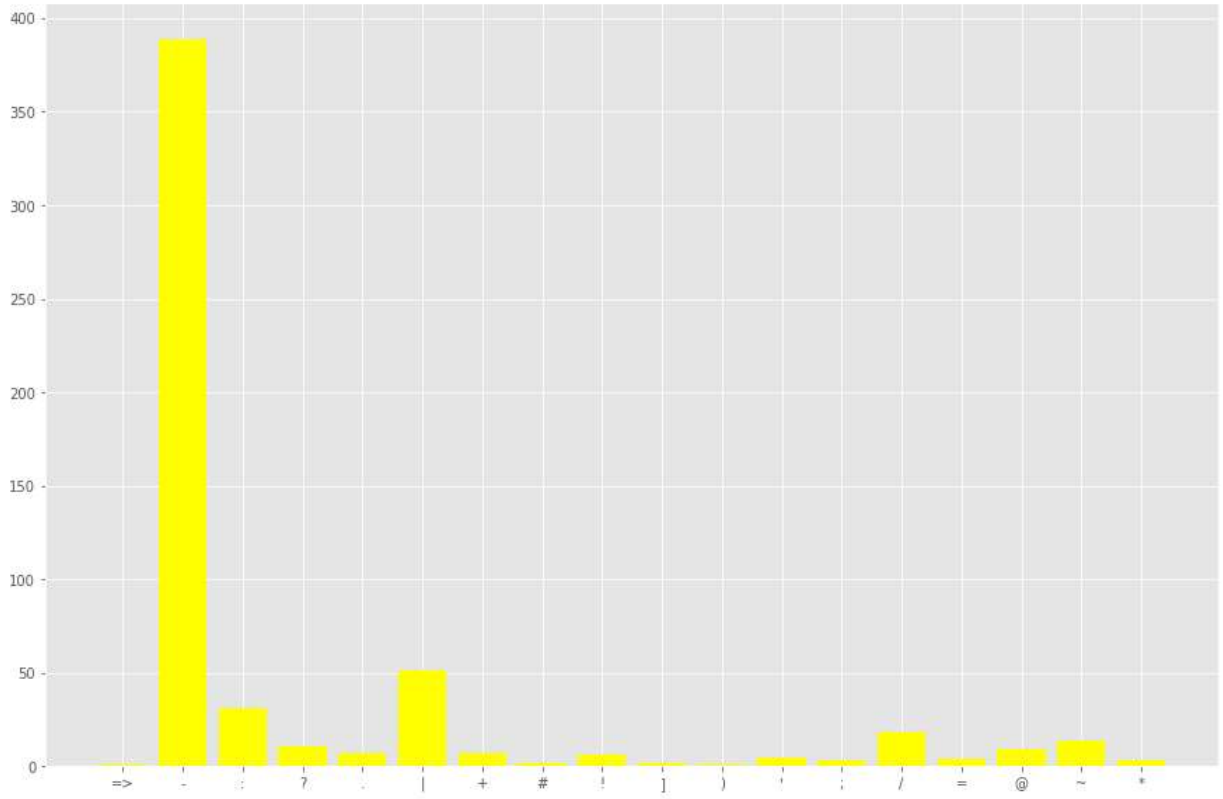
In [20]:
```
x , y = zip(*top)
plt.figure(figsize = (10 , 10))
plt.bar(x , y);
```



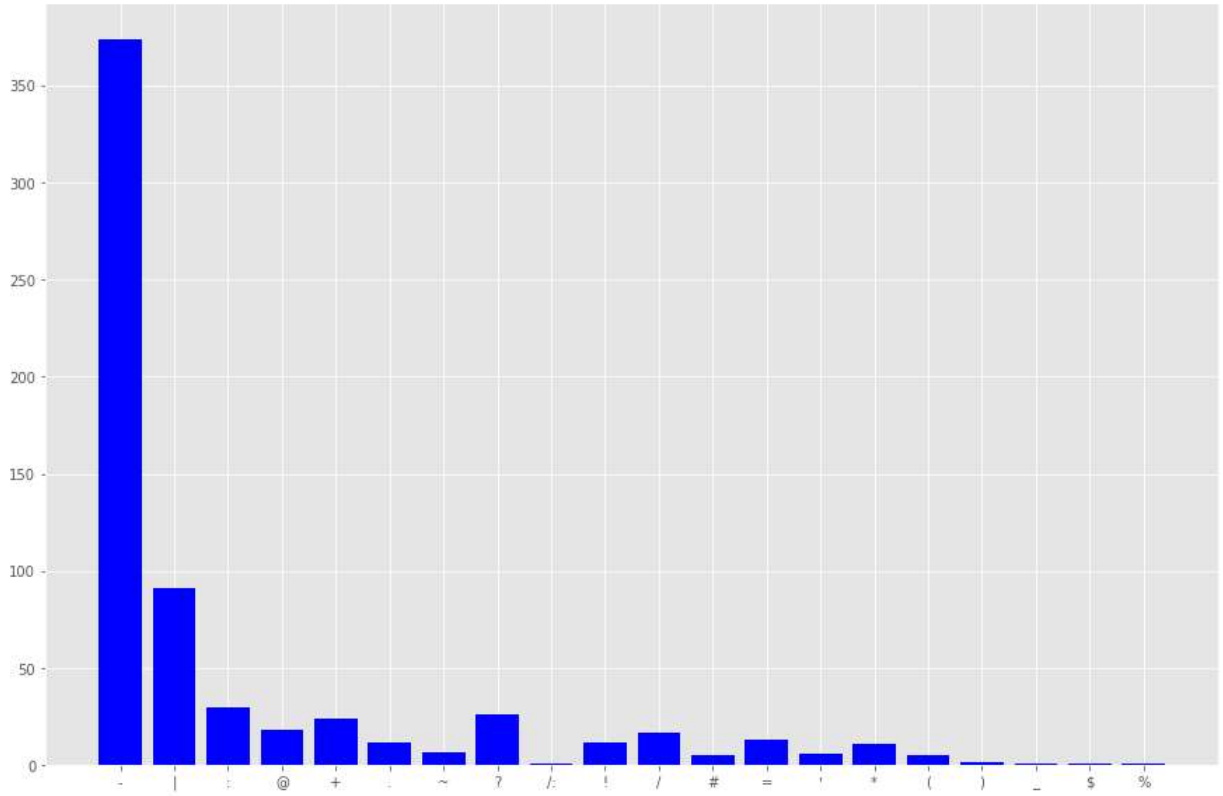
Analyzing Punctuations

In [21]:
```
plt.figure(figsize=(15 , 10))

corpus = create_corpus(1)

dic=defaultdict(int)
import string
special = string.punctuation
for i in (corpus):
    if i in special:
        dic[i]+=1

x,y=zip(*dic.items())
plt.bar(x,y , color = 'yellow')
```

Out[21]: <BarContainer object of 18 artists>



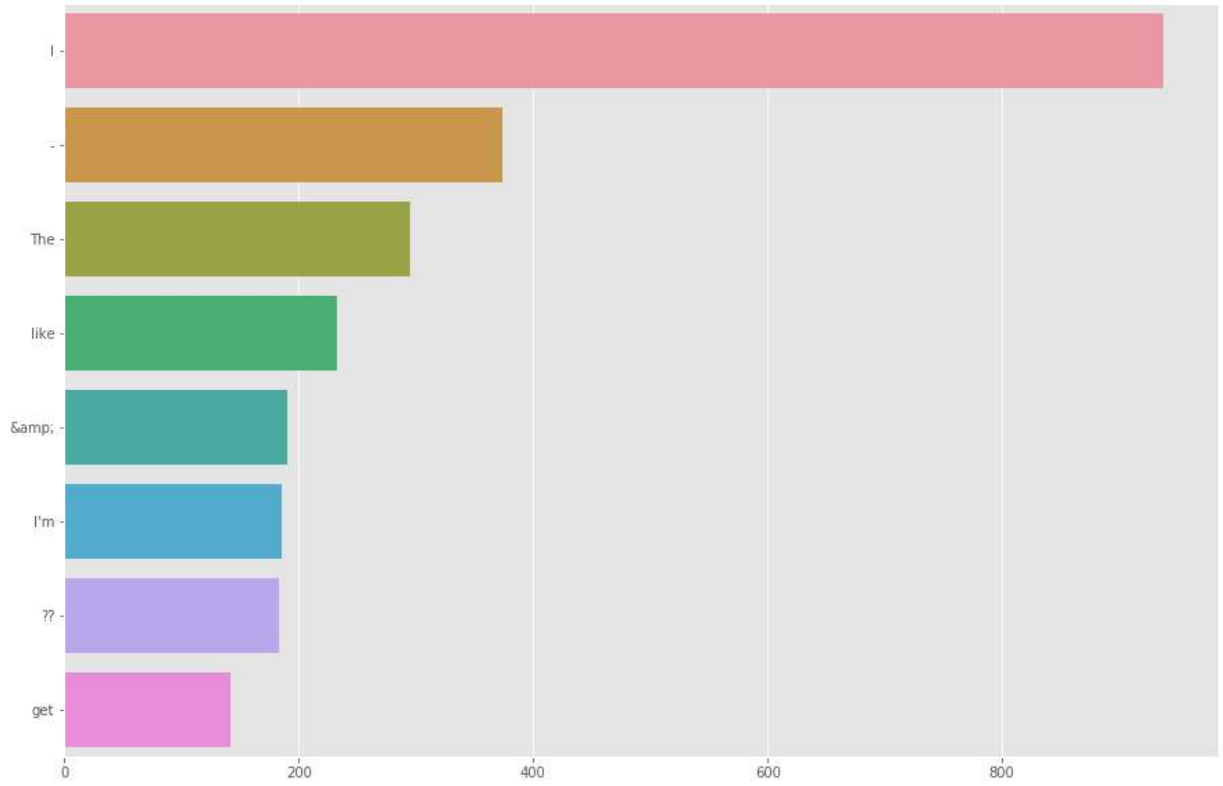
In [22]:
```
plt.figure(figsize=(15 , 10))
corpus = create_corpus(0)

dic=defaultdict(int)
import string
special = string.punctuation
for i in (corpus):
    if i in special:
        dic[i]+=1

x,y=zip(*dic.items())
plt.bar(x , y , color = 'blue');
```



Common Words

In [23]:
```
counter=Counter(corpus)
most=counter.most_common()

x=[]
y=[]

for word,count in most[:40]:
    if (word not in stop) :
        x.append(word)
        y.append(count)
```

In [24]:
```
plt.figure(figsize=(15 , 10))
sns.barplot(x = y , y = x);
```



In [10]:
```
df = pd.concat([train , test])
```

Data Preprocessing

1 . Removing stop words(Optional)

2 . Remove Punctuations

3 . Remove Html

4 . Remove Emojis

5 . Spelling Corrections

6 . Removing Urls

In [11]:
```
## Concate both train and test part
df_new = pd.concat([train , test])
```

In [12]:
```
df_new.shape
```

Out[12]:
```
(10876, 5)
```

In [13]:
```
df_new.isnull().sum()
```

Out[13]:
```
id             0
keyword       26
location    3638
text           0
target      3263
dtype: int64
```

In [14]:
```
sns.barplot(y = 'target' , data = df_new)
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19e148f93c8>
```

In [15]:
```
df_new.head(20)
```

Out[15]:

	id	keyword	location	text	target
0	1	No keyword	NaN	Our Deeds are the Reason of this #earthquake M...	1.0
1	4	No keyword	NaN	Forest fire near La Ronge Sask. Canada	1.0
2	5	No keyword	NaN	All residents asked to 'shelter in place' are ...	1.0
3	6	No keyword	NaN	13,000 people receive #wildfires evacuation or...	1.0
4	7	No keyword	NaN	Just got sent this photo from Ruby #Alaska as ...	1.0
5	8	No keyword	NaN	#RockyFire Update => California Hwy. 20 closed...	1.0
6	10	No keyword	NaN	#flood #disaster Heavy rain causes flash flood...	1.0
7	13	No keyword	NaN	I'm on top of the hill and I can see a fire in...	1.0
8	14	No keyword	NaN	There's an emergency evacuation happening now ...	1.0
9	15	No keyword	NaN	I'm afraid that the tornado is coming to our a...	1.0
10	16	No keyword	NaN	Three people died from the heat wave so far	1.0
11	17	No keyword	NaN	Haha South Tampa is getting flooded hah- WAIT ...	1.0
12	18	No keyword	NaN	#raining #flooding #Florida #TampaBay #Tampa 1...	1.0
13	19	No keyword	NaN	#Flood in Bago Myanmar #We arrived Bago	1.0
14	20	No keyword	NaN	Damage to school bus on 80 in multi car crash ...	1.0
15	23	No keyword	NaN	What's up man?	0.0
16	24	No keyword	NaN	I love fruits	0.0
17	25	No keyword	NaN	Summer is lovely	0.0
18	26	No keyword	NaN	My car is so fast	0.0
19	28	No keyword	NaN	What a gooooooooaaaaaal!!!!!!	0.0

In [16]:
```
df_new.tail()
```

Out[16]:

	id	keyword	location	text	target
3258	10861	NaN	NaN	EARTHQUAKE SAFETY LOS ANGELES □ÛÒ SAFETY FASTE...	NaN
3259	10865	NaN	NaN	Storm in RI worse than last hurricane. My city...	NaN
3260	10868	NaN	NaN	Green Line derailment in Chicago http://t.co/U...	NaN
3261	10874	NaN	NaN	MEG issues Hazardous Weather Outlook (HWO) htt...	NaN
3262	10875	NaN	NaN	#CityofCalgary has activated its Municipal Eme...	NaN

In [17]:
```
df_new['text'][0]
```

Out[17]:
```
0    Our Deeds are the Reason of this #earthquake M...
0                    Just happened a terrible car crash
Name: text, dtype: object
```

In [18]:
```
df_new['text'][12]
```

Out[18]:
```
12    #raining #flooding #Florida #TampaBay #Tampa 1...
12                                 No don't tell me that!
Name: text, dtype: object
```

In [19]:
```
train['text'][0]
```

Out[19]:
```
'Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all'
```

Removing URLS

In [20]:
```
def remove_URL(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'',text)
```

In [21]:
```
train['text'] = train['text'].apply(lambda x : remove_URL(x))
df_new['text'] = df_new['text'].apply(lambda x : remove_URL(x))
```

Remove HTML tags

In [22]:
```
def remove_html(text):
    html=re.compile(r'<.*?>')
    return html.sub(r'',text)
```

In [23]:
```
train['text'] = train['text'].apply(lambda x : remove_html(x))
df_new['text'] = df_new['text'].apply(lambda x : remove_URL(x))
```

Remove Emoji's

In [24]:
```
def remove_emoji(text):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
```

In [25]:
```
df_new['text']=df_new['text'].apply(lambda x: remove_emoji(x))
train['text'] = train['text'].apply(lambda x : remove_emoji(x))
```

In [26]:
```
df_new['text'][12]
```

Out[26]:
```
12    #raining #flooding #Florida #TampaBay #Tampa 1...
12                                 No don't tell me that!
Name: text, dtype: object
```

Remove Punctuations

In [27]:
```
def remove_punct(text):
    table=str.maketrans('','',string.punctuation)
    return text.translate(table)
```

In [28]:
```
df_new['text'] = df_new['text'].apply(lambda x : remove_punct(x))
train['text'] = train['text'].apply(lambda x : remove_punct(x))
```

In [29]:
```
df_new['text'][12]
```

Out[29]:
```
12    raining flooding Florida TampaBay Tampa 18 or ...
12                                  No dont tell me that
Name: text, dtype: object
```

In [30]:
```
train['text'][12]
```

Out[30]:
```
'raining flooding Florida TampaBay Tampa 18 or 19 days Ive lost count '
```

In [31]:
```
train.head()
```

Out[31]:

	id	keyword	location	text	target
0	1	No keyword	NaN	Our Deeds are the Reason of this earthquake Ma...	1
1	4	No keyword	NaN	Forest fire near La Ronge Sask Canada	1
2	5	No keyword	NaN	All residents asked to shelter in place are be...	1
3	6	No keyword	NaN	13000 people receive wildfires evacuation orde...	1
4	7	No keyword	NaN	Just got sent this photo from Ruby Alaska as s...	1

In [32]:
```
train.tail()
```

Out[32]:

	id	keyword	location	text	target
7608	10869	No keyword	NaN	Two giant cranes holding a bridge collapse int...	1
7609	10870	No keyword	NaN	ariaahrary TheTawniest The out of control wild...	1
7610	10871	No keyword	NaN	M194 0104 UTC5km S of Volcano Hawaii	1
7611	10872	No keyword	NaN	Police investigating after an ebike collided w...	1
7612	10873	No keyword	NaN	The Latest More Homes Razed by Northern Califo...	1

Spelling corrections

In [33]:
```
spell = SpellChecker()
def correct_spellings(text):
    corrected_text = []
    misspelled_words = spell.unknown(text.split())
    for word in text.split():
        if word in misspelled_words:
            corrected_text.append(spell.correction(word))
        else:
            corrected_text.append(word)
    return " ".join(corrected_text)
```

In [34]:
```
# df_new['text'] = df_new['text'].apply(lambda x : correct_spellings(x))
# train['text'] = train['text'].apply(lambda x : correct_spellings(x))
```

In [35]:
```
train.tail()
```

Out[35]:

	id	keyword	location	text	target
7608	10869	No keyword	NaN	Two giant cranes holding a bridge collapse int...	1
7609	10870	No keyword	NaN	ariaahrary TheTawniest The out of control wild...	1
7610	10871	No keyword	NaN	M194 0104 UTC5km S of Volcano Hawaii	1
7611	10872	No keyword	NaN	Police investigating after an ebike collided w...	1
7612	10873	No keyword	NaN	The Latest More Homes Razed by Northern Califo...	1

In [36]:
```
df_new.head()
```

Out[36]:

	id	keyword	location	text	target
0	1	No keyword	NaN	Our Deeds are the Reason of this earthquake Ma...	1.0
1	4	No keyword	NaN	Forest fire near La Ronge Sask Canada	1.0
2	5	No keyword	NaN	All residents asked to shelter in place are be...	1.0
3	6	No keyword	NaN	13000 people receive wildfires evacuation orde...	1.0
4	7	No keyword	NaN	Just got sent this photo from Ruby Alaska as s...	1.0

Convert Text to TFIDF-W2V

In [37]:
```
x = train.drop(['keyword' , 'location' , 'target'] , axis = 1)
y = train['target']
```

In [38]:
```
x
```

Out[38]:

	id	text
0	1	Our Deeds are the Reason of this earthquake Ma...
1	4	Forest fire near La Ronge Sask Canada
2	5	All residents asked to shelter in place are be...
3	6	13000 people receive wildfires evacuation orde...
4	7	Just got sent this photo from Ruby Alaska as s...
...	...	...
7608	10869	Two giant cranes holding a bridge collapse int...
7609	10870	ariaahrary TheTawniest The out of control wild...
7610	10871	M194 0104 UTC5km S of Volcano Hawaii
7611	10872	Police investigating after an ebike collided w...
7612	10873	The Latest More Homes Razed by Northern Califo...

7613 rows × 2 columns

In [39]:
```
test
```

Out[39]:

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan
...	...	...	...	...
3258	10861	NaN	NaN	EARTHQUAKE SAFETY LOS ANGELES □ÛÒ SAFETY FASTE...
3259	10865	NaN	NaN	Storm in RI worse than last hurricane. My city...
3260	10868	NaN	NaN	Green Line derailment in Chicago http://t.co/U...
3261	10874	NaN	NaN	MEG issues Hazardous Weather Outlook (HWO) htt...
3262	10875	NaN	NaN	#CityofCalgary has activated its Municipal Eme...

3263 rows × 4 columns

In [40]:
```
final_test = test.drop(['keyword' , 'location'] , axis = 1)
final_test
```

Out[40]:

	id	text
0	0	Just happened a terrible car crash
1	2	Heard about #earthquake is different cities, s...
2	3	there is a forest fire at spot pond, geese are...
3	9	Apocalypse lighting. #Spokane #wildfires
4	11	Typhoon Soudelor kills 28 in China and Taiwan
...	...	...
3258	10861	EARTHQUAKE SAFETY LOS ANGELES □ÛÒ SAFETY FASTE...
3259	10865	Storm in RI worse than last hurricane. My city...
3260	10868	Green Line derailment in Chicago http://t.co/U...
3261	10874	MEG issues Hazardous Weather Outlook (HWO) htt...
3262	10875	#CityofCalgary has activated its Municipal Eme...

3263 rows × 2 columns

In [41]:
```
final_test.shape
```

Out[41]:
```
(3263, 2)
```

In [42]:
```
x_train , x_test , y_train , y_test = train_test_split(x , y , test_size = 0.20
```

In [43]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
```

```
(6090, 2)
(1523, 2)
(6090,)
```

TFIDF-W2V on Text

In [45]:
```
tfidf_vect = TfidfVectorizer()
X_train_tfidf_w2v = tfidf_vect.fit_transform(x_train["text"])
X_test_tfidf_w2v = tfidf_vect.transform(x_test["text"])
test_tfidf_w2v = tfidf_vect.transform(final_test["text"])

dictionary = dict(zip(tfidf_vect.get_feature_names(), list(tfidf_vect.idf_)))
```

In [46]:
```
print(X_train_tfidf_w2v.shape)
print(X_test_tfidf_w2v.shape)
print(test_tfidf_w2v.shape)
```

```
(6090, 15510)
(1523, 15510)
(3263, 15510)
```

X_Train

In [56]:
```
import gensim
i=0
list_of_sent_x_train=[]
for sent in tqdm(x_train['text'].values):
    filtered_sentence=[]
    sent = remove_html(sent)
    for w in sent.split():
        for cleaned_words in remove_punct(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent_x_train.append(filtered_sentence)
```

```
100%|██████████| 6090/6090 [00:00<00:00, 9938.83it/s]
```

In [57]:
```
w2v_model = gensim.models.Word2Vec(list_of_sent_x_train)

w2v_words = list(w2v_model.wv.vocab)
print(len(w2v_words))
```

```
2268
```

In [58]:
```
# TF-IDF weighted Word2Vec

tfidf_feat = tfidf_vect.get_feature_names()

tfidf_sent_vectors_x_train = []
row=0;

for sent in tqdm(list_of_sent_x_train):
    sent_vec = np.zeros(100)
    weight_sum =0

    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))

            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf

    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_x_train.append(sent_vec)
```

```
100%|██████████| 6090/6090 [00:20<00:00, 300.70it/s]
```

X_TEST

In [59]:
```
import gensim
i=0
list_of_sent_x_test = []
for sent in tqdm(x_test['text'].values):
    filtered_sentence=[]
    sent=remove_html(sent)
    for w in sent.split():
        for cleaned_words in remove_punct(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent_x_test.append(filtered_sentence)
```

```
100%|██████████| 1523/1523 [00:00<00:00, 9665.04it/s]
```

In [60]:
```
# TF-IDF weighted Word2Vec

tfidf_feat = tfidf_vect.get_feature_names()

tfidf_sent_vectors_x_test = []
row=0;

for sent in tqdm(list_of_sent_x_test):
    sent_vec = np.zeros(100)
    weight_sum =0

    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))

            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf

    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_x_test.append(sent_vec)
```

```
100%|██████████| 1523/1523 [00:04<00:00, 330.43it/s]
```

TEST

In [61]:
```
import gensim
i=0

list_of_sent_test=[]
for sent in tqdm(test['text'].values):
    filtered_sentence=[]
    sent=remove_html(sent)
    for w in sent.split():
        for cleaned_words in remove_punct(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent_test.append(filtered_sentence)
```

```
100%|██████████| 3263/3263 [00:00<00:00, 9636.47it/s]
```

In [62]:
```
# TF-IDF weighted Word2Vec

tfidf_feat = tfidf_vect.get_feature_names()

tfidf_sent_vectors_test = []
row=0;

for sent in tqdm(list_of_sent_test):
    sent_vec = np.zeros(100)
    weight_sum =0

    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))

            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf

    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
```

```
100%|██████████| 3263/3263 [00:11<00:00, 272.52it/s]
```

In [63]:
```
x_train_w2v = tfidf_sent_vectors_x_train
x_test_w2v = tfidf_sent_vectors_x_test
test_w2v = tfidf_sent_vectors_test
```

Apply DIfferent ML model on TFIDF-W2V

Logistic Regression on TFIDF-W2V

In [66]:
```
#Standardising the train and test data

sc = StandardScaler()
X_train = sc.fit_transform(x_train_w2v)
X_test = sc.transform(x_test_w2v)
Test = sc.transform(test_w2v)
```

In [67]:
```
tuned_parameters = [10**-4, 10**-2, 10**0, 10**2, 10**4]

cv_scores = []

for i in tqdm(tuned_parameters):
    model = LogisticRegression(penalty = 'l1' , C = i , n_jobs = -1)
    scores = cross_val_score(model , X_train , y_train , cv = 10 , scoring = 'ac
    cv_scores.append(scores.mean())
```

```
100%|██████████| 5/5 [00:03<00:00,  1.25it/s]
```

In [68]:
```
optimal_C = tuned_parameters[cv_scores.index(max(cv_scores))]
print('\nThe optimal value of C is %.3f.' % optimal_C)
```

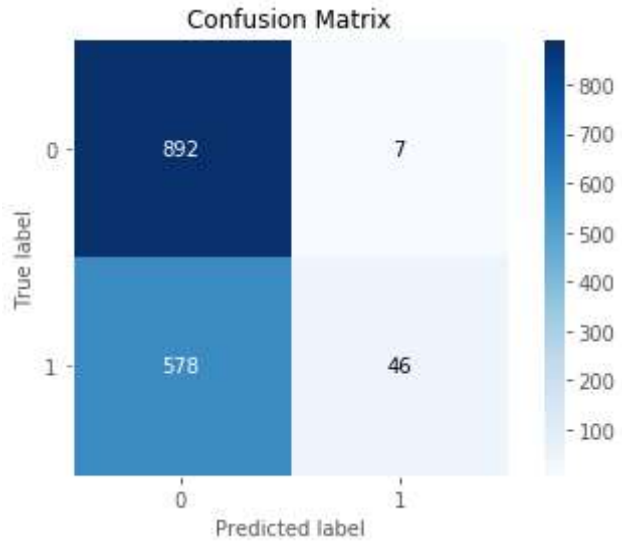
```
The optimal value of C is 0.000.
```

In [69]:
```
lr = LogisticRegression(penalty = 'l2' , C = 0.0001 , n_jobs = -1)
lr.fit(X_train , y_train)
pred = lr.predict(X_test)

ac = accuracy_score(y_test , pred)
print(ac*100)
```

```
61.58896913985554
```

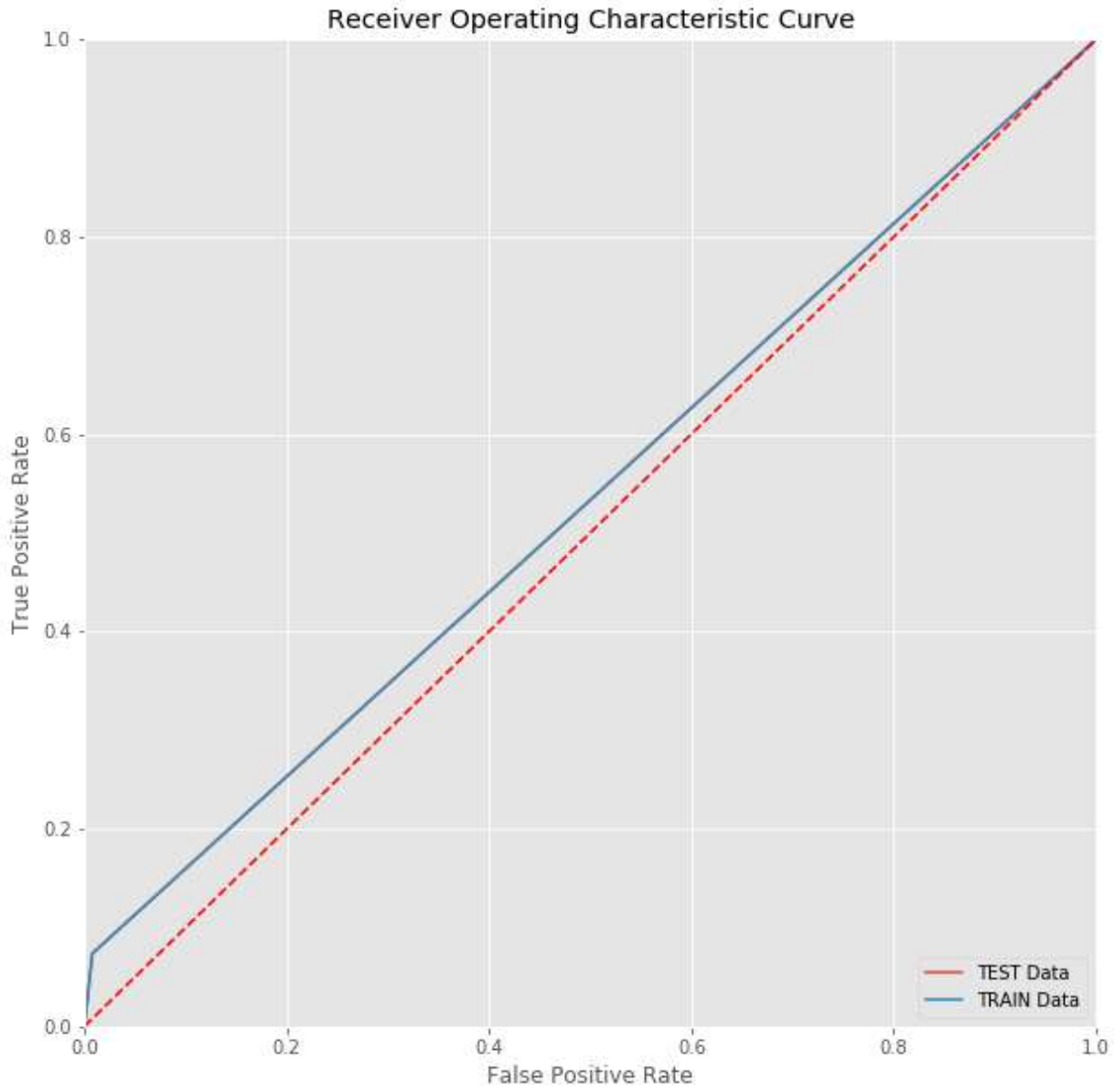
In [70]:
```
from sklearn import metrics
import scikitplot as skplt

print("***Test Data Report***")
print("Best C = ",optimal_C)
fpr, tpr, threshold = roc_curve(y_test , lr.predict(X_test))
auc = metrics.auc(fpr, tpr)
print("AUC = ",auc*100)
skplt.metrics.plot_confusion_matrix(y_test, pred)
plt.show()
```

```
***Test Data Report***
Best C =  0.0001
AUC =  53.296575967599324
```



In [71]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test , lr.predict(X_test))
fpr2, tpr2, threshold2 = metrics.roc_curve(y_test , lr.predict(X_test))

roc_auc = metrics.auc(fpr, tpr)
roc_auc2 = metrics.auc(fpr2, tpr2)
```

In [72]:
```
# plot ROC-curve
plt.figure(figsize = (10 , 10))
plt.title('Receiver Operating Characteristic Curve')
plt.gca()
plt.plot(fpr, tpr, label = 'AUC = %0.2f' % roc_auc)
plt.plot(fpr2, tpr2, label = 'AUC = %0.2f' % roc_auc2)
plt.legend(['TEST Data', 'TRAIN Data'],loc = 'lower right')

plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [76]:
```
# feature_name = count_vect.get_feature_names()
# w = lr.coef_
# weight=w.reshape(-1)
# sorted_feature = np.argsort(weight)
# top_20_positive_feature=sorted_feature[:-20:-1]
```

In [75]:
```
# print("Positive feature top 20 :")
# print("-------")
# for i in top_20_positive_feature:
#     print("%s\t-->\t%f"%(feature_name[i],weight[i]))
```

In [77]:
```
# w = lr.coef_
# weight=w.reshape(-1)
# sorted_feature = np.argsort(weight)
# feature_name = count_vect.get_feature_names()
# top_20_negative_feature = sorted_feature[:20]
```

In [73]:
```
print("Negative feature top 20 :")
print("-------")
for i in top_20_negative_feature:
    print("%s\t -->\t%f  "%(feature_name[i],weight[i]))
```

```
Negative feature top 20 :
-------
you      -->    -0.023155
my       -->    -0.019741
im       -->    -0.014935
me       -->    -0.012642
body     -->    -0.012451
just     -->    -0.012423
love     -->    -0.011289
your     -->    -0.010415
full     -->    -0.010302
bags     -->    -0.009893
panic    -->    -0.009826
like     -->    -0.009813
harm     -->    -0.009613
ruin     -->    -0.009397
so       -->    -0.009258
blazing  -->    -0.009179
lol      -->    -0.009080
nowplaying       -->    -0.009073
wrecked  -->    -0.009067
youtube  -->    -0.008984
```

In the above list it is clear that there are so many stop words that affects our model.we have to remove the stopwords.

Apply KNN on TFIDF-W2V

In [78]:
```
knn = KNeighborsClassifier()
knn.fit(X_train , y_train)

y_pred_knn = knn.predict(X_test)

ac = accuracy_score(y_test , y_pred_knn)
print(ac*100)
```

```
67.10439921208142
```

In [75]:
```
n_folds = 5
parameters = {
        'n_neighbors': range (2 , 50 , 2)
        }
```

In [76]:
```
knn = KNeighborsClassifier()

tree = GridSearchCV(estimator = knn , param_grid = parameters , cv = n_folds , n_
tree.fit(X_train , y_train)

score1 = tree.cv_results_

print(pd.DataFrame(score1).head())
print(tree.best_params_)
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.004988      0.000631         0.533524        0.009570
1       0.004796      0.000750         0.645834        0.046200
2       0.005581      0.000799         0.577961        0.018822
3       0.009375      0.007291         0.574671        0.039541
4       0.007781      0.003301         0.679563        0.171710

  param_n_neighbors              params  split0_test_score  \
0                 2   {'n_neighbors': 2}           0.666667
1                 4   {'n_neighbors': 4}           0.639573
2                 6   {'n_neighbors': 6}           0.639573
3                 8   {'n_neighbors': 8}           0.621511
4                10  {'n_neighbors': 10}           0.613300

   split1_test_score  split2_test_score  split3_test_score  ...  \
0           0.647783           0.638752           0.653530  ...
1           0.647783           0.615764           0.643678  ...
2           0.637110           0.615764           0.641215  ...
3           0.636289           0.609195           0.624795  ...
4           0.623153           0.605911           0.619048  ...

   mean_test_score  std_test_score  rank_test_score  split0_train_score  \
0         0.650082        0.009608                1            0.717159
1         0.633990        0.012369                2            0.649836
2         0.629392        0.012228                3            0.646757
3         0.620361        0.010063                4            0.628284
4         0.614286        0.006175                5            0.616174

   split1_train_score  split2_train_score  split3_train_score  \
0            0.728243            0.726806            0.722496
1            0.712849            0.659278            0.656199
2            0.661125            0.653941            0.651683
3            0.659688            0.635263            0.628489
4            0.634647            0.624179            0.622126

   split4_train_score  mean_train_score  std_train_score
0            0.718801          0.722701         0.004325
1            0.658251          0.667282         0.023018
2            0.631773          0.649056         0.009802
3            0.623768          0.635099         0.012831
4            0.621716          0.623768         0.006051

[5 rows x 21 columns]
{'n_neighbors': 2}
```

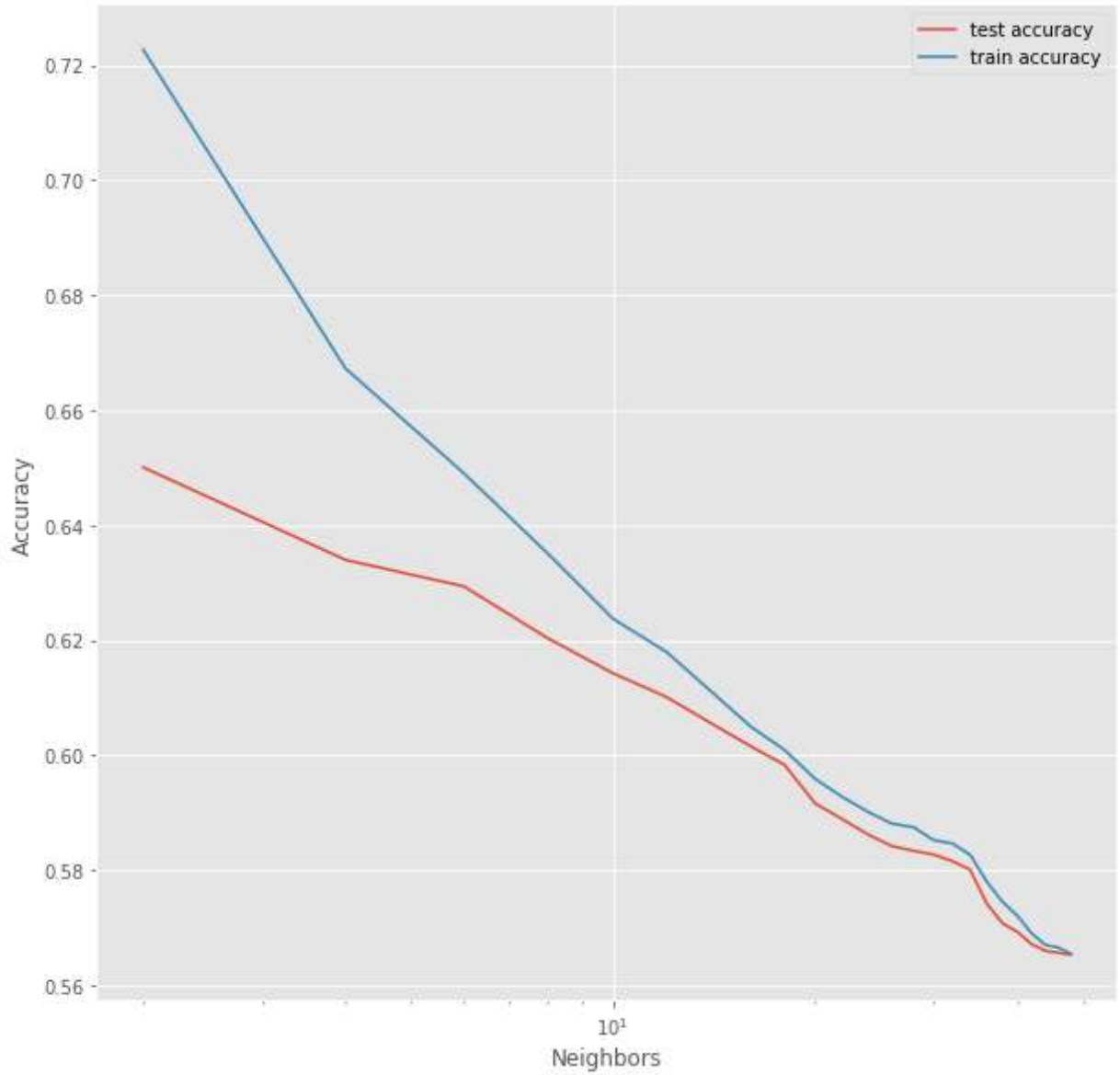
In [77]:
```
score1 = pd.DataFrame(score1)
score1.head()
```

Out[77]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	params
0	0.004988	0.000631	0.533524	0.009570	2	{'n_neighbors': 2}
1	0.004796	0.000750	0.645834	0.046200	4	{'n_neighbors': 4}
2	0.005581	0.000799	0.577961	0.018822	6	{'n_neighbors': 6}
3	0.009375	0.007291	0.574671	0.039541	8	{'n_neighbors': 8}
4	0.007781	0.003301	0.679563	0.171710	10	{'n_neighbors': 10}

5 rows × 21 columns

In [78]:
```
plt.figure(figsize=(10 , 10))
plt.plot(score1['param_n_neighbors'], score1['mean_test_score'])
plt.plot(score1['param_n_neighbors'], score1['mean_train_score'])
plt.xlabel('Neighbors')
plt.ylabel('Accuracy')
plt.legend(['test accuracy', 'train accuracy'] , loc='upper right')
plt.xscale('log')
```



In [79]:
```
knn = KNeighborsClassifier(n_neighbors = 12 , algorithm = 'kd_tree')
knn.fit(X_train , y_train)

y_pred_knn = knn.predict(X_test)

ac_knn = accuracy_score(y_test , y_pred_knn)
print('After Cross-validation: ',ac*100)
```

```
After Cross-validation:  67.10439921208142
```

Naive-Bayes on TFIDF-W2V

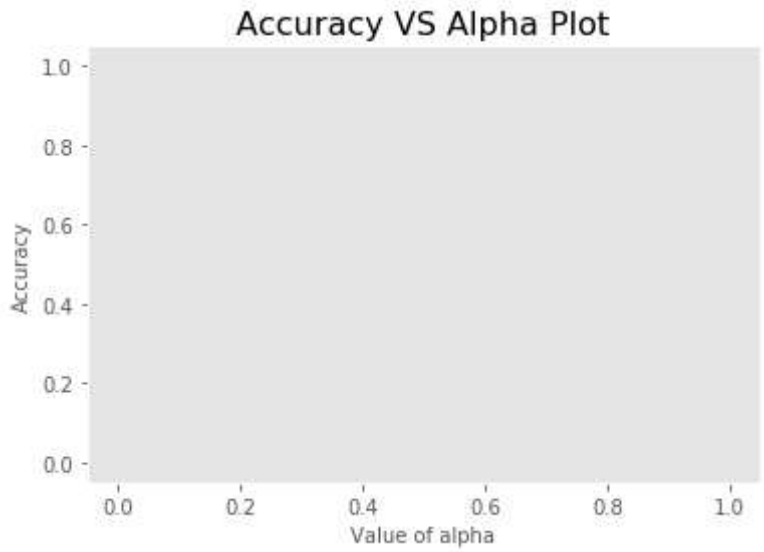
In [80]:
```
alpha = []
i = 0.001

while(i <= 1000):
    alpha.append(np.round(i,3))
    i *= 3

cv_scores = []

for k in tqdm(alpha):
    model = MultinomialNB(alpha = k)
    scores = cross_val_score(model , X_train , y_train , cv = 10 , scoring = 'f1_
    cv_scores.append(scores.mean())
```

```
100%|██████████| 13/13 [00:01<00:00,  9.77it/s]
```

In [81]:
```
optimal_alpha = alpha[cv_scores.index(max(cv_scores))]
print('\nThe optimal value of alpha is %.3f.' % optimal_alpha)
```

```
The optimal value of alpha is 0.001.
```

In [82]:
```
# plot accuracy vs alpha
plt.plot(alpha, cv_scores)
plt.xlabel('Value of alpha',size=10)
plt.ylabel('Accuracy',size=10)
plt.title('Accuracy VS Alpha Plot',size=16)
plt.grid()
plt.show()
print("\n*******Train Data Report******");
print("\nAlpha values :\n",alpha)
print("\nF1 Score for each value of alpha :\n ", np.round(cv_scores,5)*100)
```

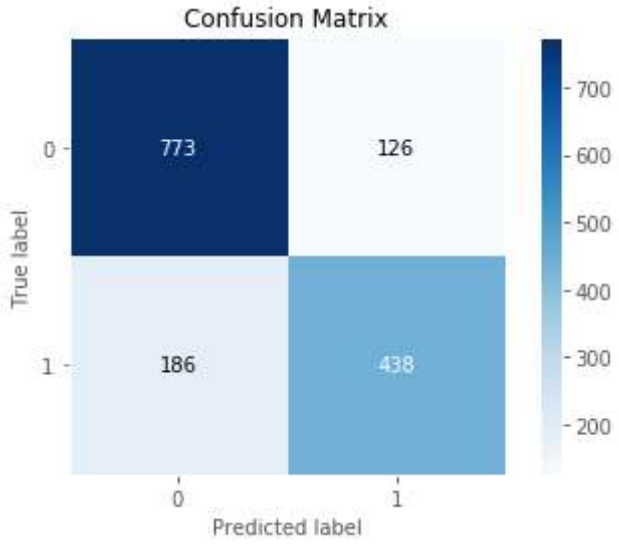



```
*******Train Data Report******

Alpha values :
 [0.001, 0.003, 0.009, 0.027, 0.081, 0.243, 0.729, 2.187, 6.561, 19.683, 59.04
9, 177.147, 531.441]

F1 Score for each value of alpha :
  [nan nan nan nan nan nan nan nan nan nan nan nan nan]
```

In [15]:
```
model_bow_multinomial = MultinomialNB(alpha = 0.001)

model_bow_multinomial.fit(X_train , y_train)

pred = model_bow_multinomial.predict(X_test)
```

In [89]:
```
print("***Test Data Report***")
print('Accuracy = ', accuracy_score(y_test, pred)*100)
print("f1_score = ",f1_score(y_test, pred, average='macro')*100)

print("precision_score = " , precision_score(y_test , pred, average='macro')*100
print("recall_score = " , recall_score(y_test, pred, average='macro')*100)
skplt.metrics.plot_confusion_matrix(y_test, pred)

plt.show()
```

```
***Test Data Report***
Accuracy =  79.51411687458962
f1_score =  78.47256200324016
precision_score =  79.13218556563797
recall_score =  78.08836741678789
```



In [91]:
```
print(classification_report(y_test , pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       899
           1       0.78      0.70      0.74       624

    accuracy                           0.80      1523
   macro avg       0.79      0.78      0.78      1523
weighted avg       0.79      0.80      0.79      1523
```

SVM on TFIDF-W2V

In [85]:
```
sv = SVC()
sv.fit(X_train , y_train)

y_pred_svm = sv.predict(X_test)

ac_svr = accuracy_score(y_test , y_pred_svm)
print(ac_svr*100)
```

```
72.09455022980958
```

In [94]:
```
folds = KFold(n_splits = 5, shuffle = True, random_state = 4)
params = {"C": [0.01 , 0.1, 1, 10, 100, 1000]}

model = SVC()

model_cv_C = GridSearchCV(estimator = model, param_grid = params, cv = folds , v
model_cv_C.fit(X_train, y_train)


cv_results = pd.DataFrame(model_cv_C.cv_results_)
cv_results
```

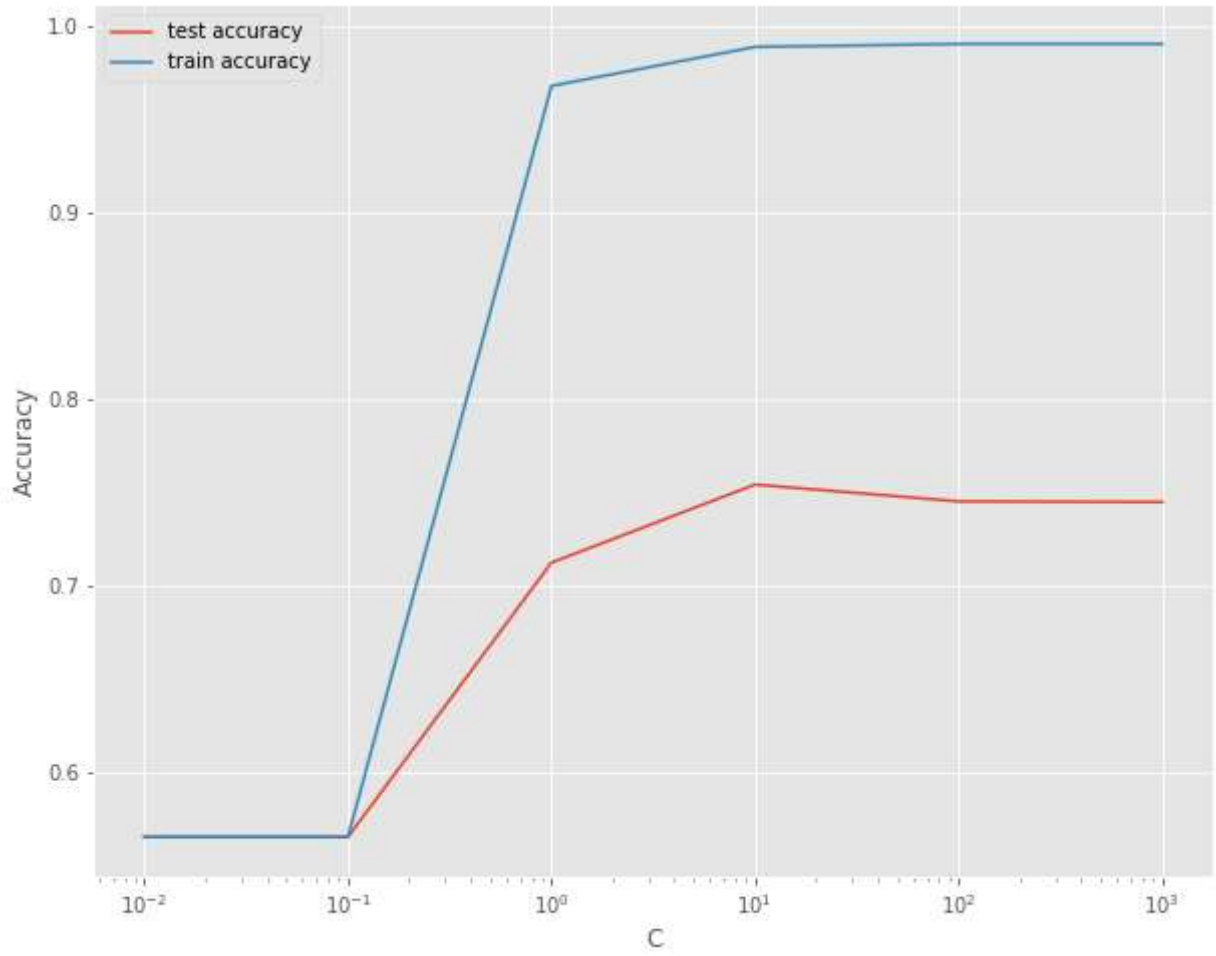
```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.4min finished
```

Out[94]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_sco
0	10.318077	0.154108	2.092425	0.150773	0.01	{'C': 0.01}	0.5812
1	8.707073	0.506377	1.864736	0.101443	0.1	{'C': 0.1}	0.5812
2	8.150981	0.161615	2.023904	0.075840	1	{'C': 1}	0.7282
3	9.221630	0.423916	2.007184	0.148073	10	{'C': 10}	0.7446
4	9.154882	0.449905	1.517774	0.054401	100	{'C': 100}	0.7380
5	7.713616	0.630339	1.490521	0.087230	1000	{'C': 1000}	0.7405

6 rows × 21 columns

In [95]:
```
plt.figure(figsize=(10 , 8))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
```



In [96]:
```
folds = KFold(n_splits = 5, shuffle = True, random_state = 4)
gamma = {'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}

model = SVC()

model_cv_g = GridSearchCV(estimator = model, param_grid = gamma, cv = folds , ve
model_cv_g.fit(X_train, y_train)

cv_results_g = pd.DataFrame(model_cv_g.cv_results_)
cv_results_g
```

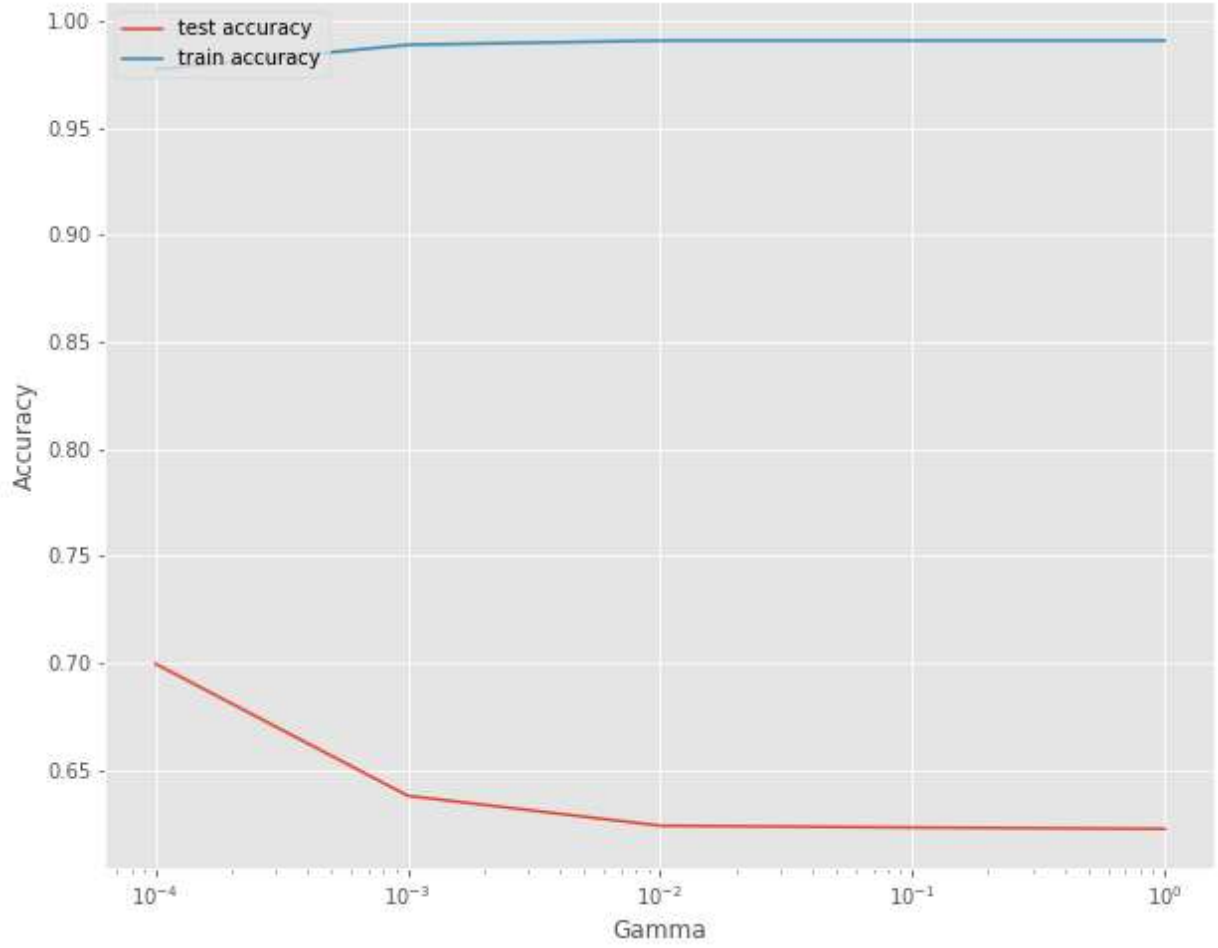
```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:  2.2min finished
```

Out[96]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_gamma	params	split0_te
0	11.685300	1.286822	2.357774	0.158978	1	{'gamma': 1}	
1	9.331086	0.451355	2.162767	0.141042	0.1	{'gamma': 0.1}	
2	7.412320	0.530985	1.748685	0.060593	0.01	{'gamma': 0.01}	
3	7.572654	0.099630	1.672800	0.082468	0.001	{'gamma': 0.001}	
4	11.188125	1.043557	2.344826	0.688058	0.0001	{'gamma': 0.0001}	

5 rows × 21 columns

In [99]:
```
plt.figure(figsize=(10 , 8))
plt.plot(cv_results_g['param_gamma'], cv_results_g['mean_test_score'])
plt.plot(cv_results_g['param_gamma'], cv_results_g['mean_train_score'])
plt.xlabel('Gamma')
plt.ylabel('Accuracy')
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
```



In [100]:
```
print(model_cv_C.best_params_)
print(model_cv_g.best_params_)
```

```
{'C': 10}
{'gamma': 0.0001}
```

In [101]:
```
sv = SVC(C = 10 , gamma = 0.0001)
sv.fit(X_train , y_train)

y_pred_svm = sv.predict(X_test)

ac_svr = accuracy_score(y_test , y_pred_svm)
print(ac_svr*100)
```

```
76.49376231122784
```

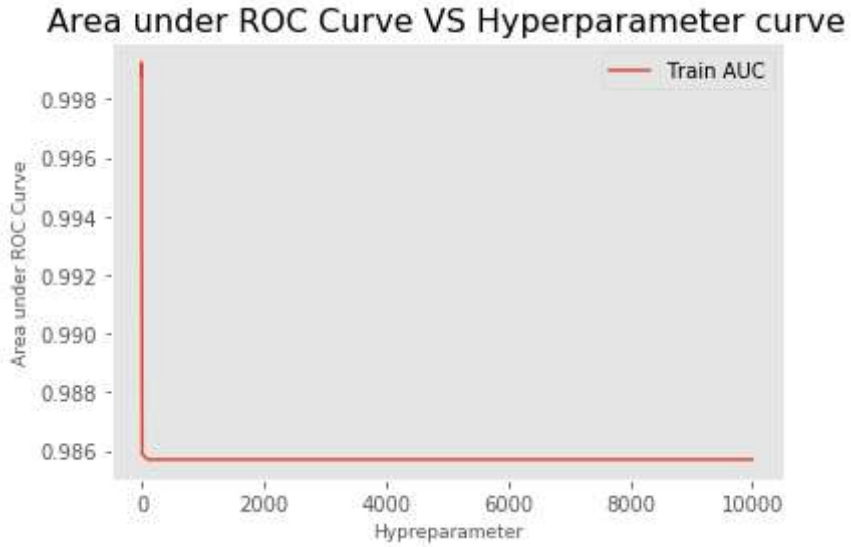
In [153]:
```
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score

alpha = [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]
auc_score=[]


for i in tqdm(alpha):
    model = linear_model.SGDClassifier(alpha=i, loss='hinge', class_weight='bala
    model.fit(X_train, y_train)

    Cal_CV_model = CalibratedClassifierCV(model, method="sigmoid",cv=10)
    Cal_CV_model.fit(X_train, y_train)
    predict_y = Cal_CV_model.predict_proba(X_train)
    preds = predict_y[:,1]
    roc_auc = roc_auc_score(y_train, preds)
    auc_score.append(roc_auc)
```

```
100%|██████████| 9/9 [00:03<00:00,  2.72it/s]
```

In [154]:
```
# determining best value of alpha
optimal_alpha = alpha[auc_score.index(max(auc_score))]
print('\nThe optimal value of alpha is %.3f.' % optimal_alpha)
```

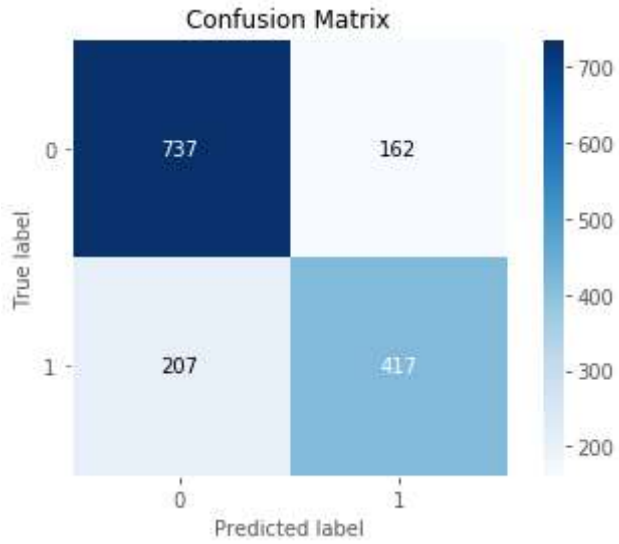
```
The optimal value of alpha is 0.010.
```

In [155]:
```
# plot accuracy vs alpha
plt.plot(alpha, auc_score,label="Train AUC")
plt.xlabel('Hypreparameter',size=9)
plt.ylabel('Area under ROC Curve',size=9)
plt.title('Area under ROC Curve VS Hyperparameter curve',size=16)
plt.legend(loc='best')
plt.grid()
plt.show()
```



In [157]:
```
model = linear_model.SGDClassifier(alpha=optimal_alpha, loss='hinge', class_weigl
model.fit(X_train, y_train)

lr = CalibratedClassifierCV(model, method="sigmoid",cv=10)
lr.fit(X_train, y_train)
pred = lr.predict(X_test)


print("***Test Data Report***")
print("Best alpha = ",optimal_alpha)
fpr, tpr, threshold = metrics.roc_curve(y_test, lr.predict(X_test))
auc = metrics.auc(fpr, tpr)
print("AUC = ",auc*100)
skplt.metrics.plot_confusion_matrix(y_test, pred)
plt.show()
```

```
***Test Data Report***
Best alpha =  0.01
AUC =  74.40345041499101
```



Decision Tree on TFIDF-W2V

In [86]:
```
dt = DecisionTreeClassifier()
dt.fit(X_train , y_train)


y_pred_dt = dt.predict(X_test)

ac = accuracy_score(y_test , y_pred_dt)
print(ac*100)
```

```
62.96782665791202
```

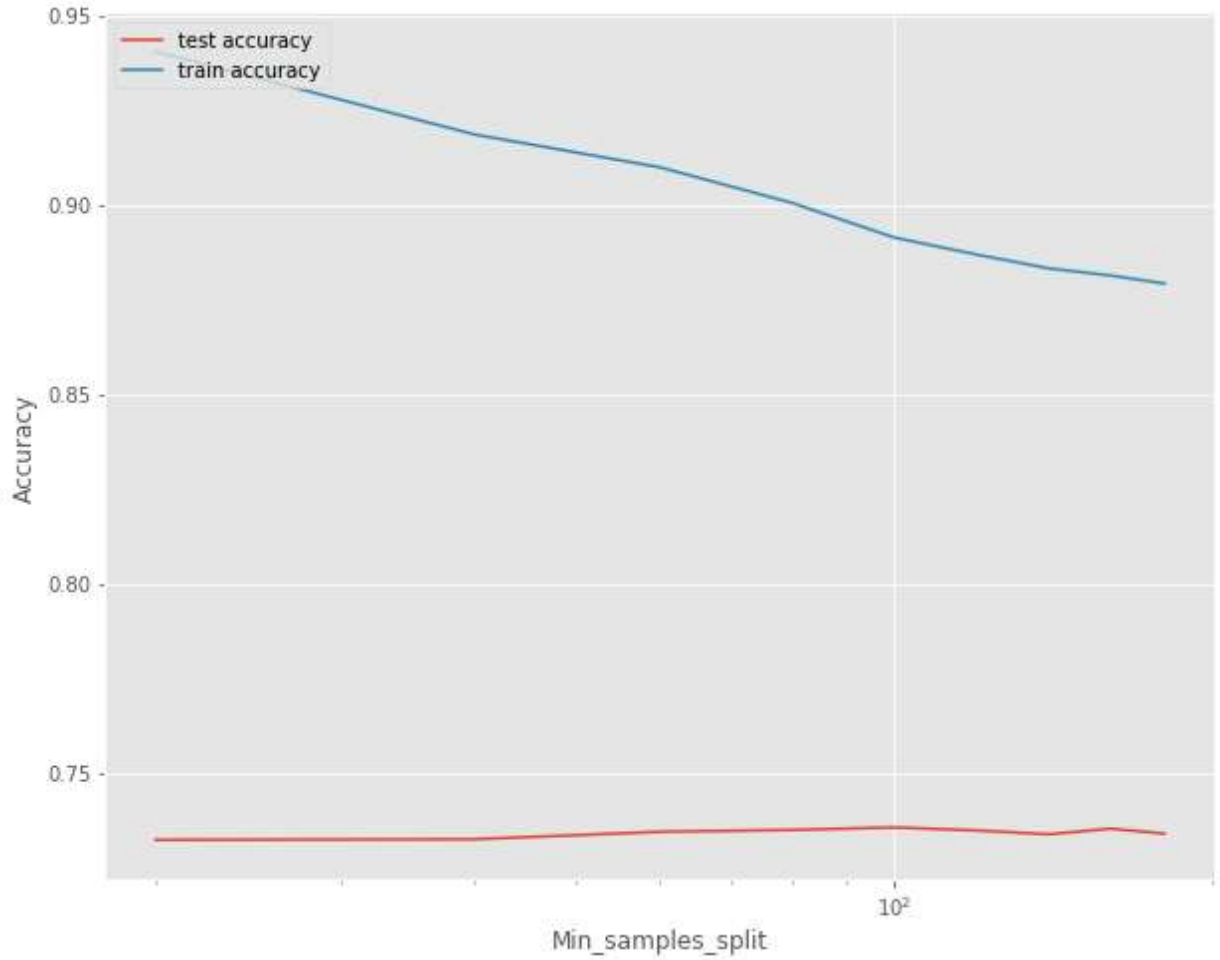
In [106]:
```
n_folds = 5
parameters = {'min_samples_split': range(20 , 200 , 20)}

dtree = DecisionTreeClassifier(random_state = 100)

tree_split = GridSearchCV(dtree, parameters , cv=n_folds , n_jobs = -1 , return_
tree_split.fit(X_train, y_train)


cv_results_ms = pd.DataFrame(tree_split.cv_results_)
cv_results_ms
```

Out[106]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	
0	2.022468	0.120813	0.003654	0.003723	20	{'min_sa
1	1.873793	0.107680	0.003213	0.003935	40	{'min_sa
2	1.928503	0.072935	0.001608	0.003217	60	{'min_sa
3	1.960126	0.109326	0.000406	0.000811	80	{'min_sa
4	2.077347	0.267628	0.004845	0.003957	100	{'min_sa
5	1.758382	0.070353	0.004053	0.003394	120	{'min_sa
6	1.711813	0.073921	0.003245	0.003907	140	{'min_sa
7	1.769603	0.118734	0.002822	0.002720	160	{'min_sa
8	1.570176	0.327149	0.002022	0.003132	180	{'min_sa

9 rows × 21 columns

In [107]:
```
plt.figure(figsize=(10 , 8))
plt.plot(cv_results_ms['param_min_samples_split'] , cv_results_ms['mean_test_sco
plt.plot(cv_results_ms['param_min_samples_split'] , cv_results_ms['mean_train_sc
plt.xlabel('Min_samples_split')
plt.ylabel('Accuracy')
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
```



In [113]:
```
n_folds = 5
parameters = {'min_samples_leaf': range (1 , 20)}

dtree = DecisionTreeClassifier()

tree_ml = GridSearchCV(dtree , parameters , cv=n_folds , n_jobs = -1 , return_tr
tree_ml.fit(X_train, y_train)


cv_results_ml = pd.DataFrame(tree_ml.cv_results_)
cv_results_ml
```

Out[113]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	
0	2.881057	0.206584	0.002193	0.001322	1	{'min_s
1	2.157800	0.104069	0.003392	0.003070	2	{'min_s
2	1.880401	0.067884	0.002589	0.000789	3	{'min_s
3	1.683492	0.110542	0.002400	0.001361	4	{'min_s
4	1.391826	0.089664	0.002595	0.000798	5	{'min_s
5	1.545468	0.122933	0.001797	0.001164	6	{'min_s
6	1.565450	0.135951	0.001399	0.000487	7	{'min_s
7	1.218413	0.114981	0.001596	0.000798	8	{'min_s
8	0.952453	0.039265	0.001995	0.000631	9	{'min_s
9	0.941085	0.072197	0.014960	0.016760	10	{'min_s
10	0.820804	0.037238	0.001198	0.000399	11	{'min_s
11	0.967914	0.109197	0.007393	0.011788	12	{'min_s
12	0.793572	0.057685	0.001796	0.000747	13	{'min_s
13	0.731141	0.069673	0.011472	0.018531	14	{'min_s
14	0.819528	0.067857	0.001196	0.000747	15	{'min_s
15	0.755467	0.095441	0.003193	0.003051	16	{'min_s
16	0.581071	0.041602	0.002601	0.001349	17	{'min_s

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	
17	0.624482	0.037352	0.001796	0.000746	18	{'min_s
18	0.578357	0.073950	0.001792	0.000745	19	{'min_s

19 rows × 21 columns

In [16]:
```
plt.figure(figsize=(10 , 8))
plt.plot(cv_results_ml['param_min_samples_leaf'] , cv_results_ml['mean_test_scor
plt.plot(cv_results_ml['param_min_samples_leaf'] , cv_results_ml['mean_train_sco
plt.xlabel('Min_samples_Leaf')
plt.ylabel('Accuracy')
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
```

In [117]:
```
n_folds = 5
parameters = {'max_depth':range (1,20)}

dtree = DecisionTreeClassifier()

tree_d = GridSearchCV(dtree , parameters , cv=n_folds , n_jobs = -1 , return_tra
tree_d.fit(X_train, y_train)


cv_results_d = pd.DataFrame(tree_d.cv_results_)
cv_results_d
```

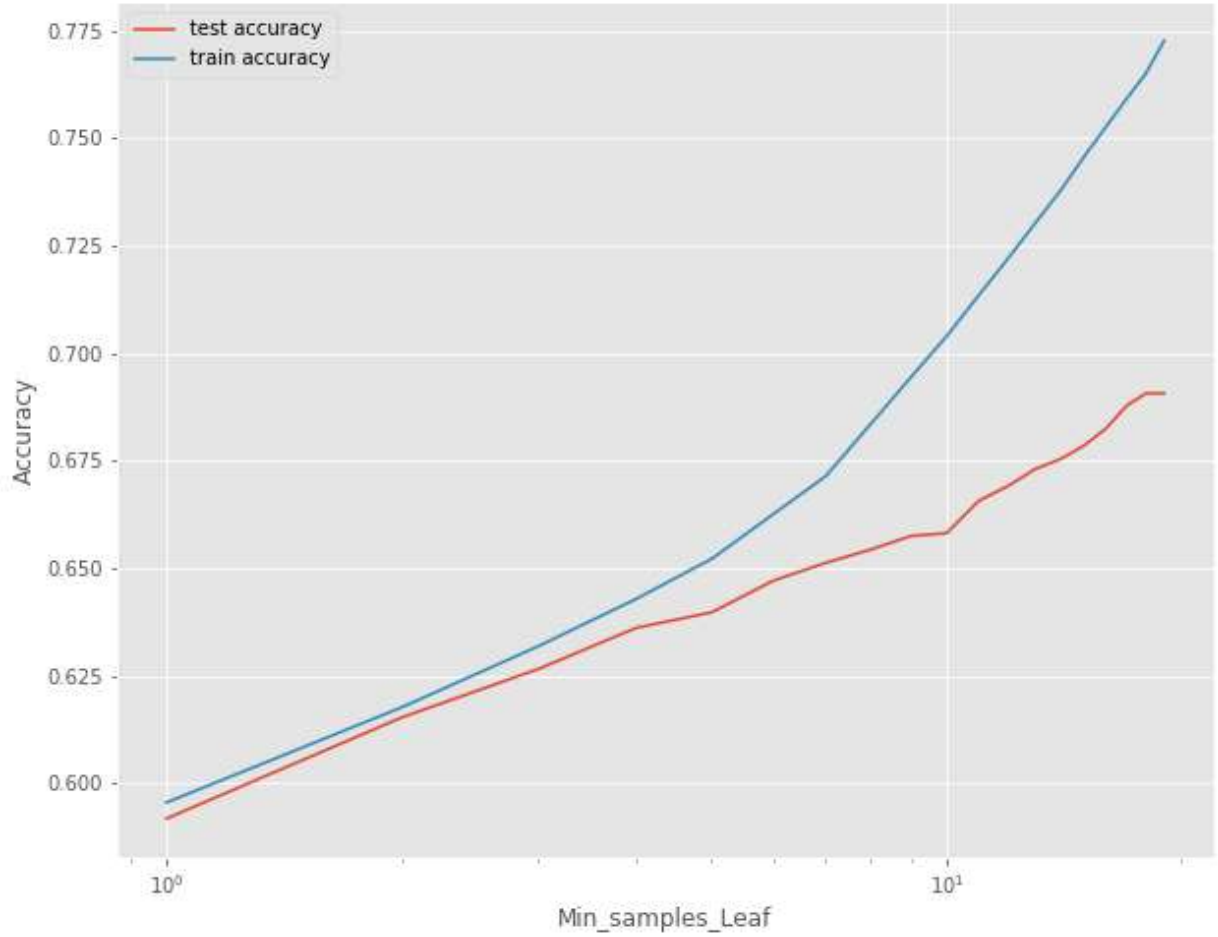
Out[117]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params
0	0.170145	0.015624	0.001197	0.000747	1	{'max_depth': 1}
1	0.170552	0.008231	0.000999	0.000894	2	{'max_depth': 2}
2	0.173044	0.014575	0.000599	0.000798	3	{'max_depth': 3}
3	0.243511	0.011576	0.001397	0.000488	4	{'max_depth': 4}
4	0.263797	0.020012	0.003591	0.004306	5	{'max_depth': 5}
5	0.276311	0.032506	0.001197	0.000978	6	{'max_depth': 6}
6	0.261819	0.029855	0.000598	0.000798	7	{'max_depth': 7}
7	0.301386	0.039834	0.001806	0.003136	8	{'max_depth': 8}
8	0.306118	0.010263	0.002604	0.002740	9	{'max_depth': 9}
9	0.356868	0.018729	0.001596	0.000798	10	{'max_depth': 10}
10	0.402870	0.024998	0.002004	0.000631	11	{'max_depth': 11}
11	0.409479	0.039242	0.000997	0.000631	12	{'max_depth': 12}
12	0.434244	0.039103	0.002805	0.002653	13	{'max_depth': 13}
13	0.402074	0.025530	0.001005	0.000900	14	{'max_depth': 14}
14	0.443128	0.027260	0.001011	0.000652	15	{'max_depth': 15}
15	0.499075	0.051335	0.001197	0.000746	16	{'max_depth': 16}
16	0.573369	0.042712	0.006782	0.012090	17	{'max_depth': 17}
17	0.522663	0.030784	0.001602	0.000494	18	{'max_depth': 18}

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params
18	0.529827	0.020076	0.002393	0.002864	19	{'max_depth': 19}

19 rows × 21 columns

In [130]:
```
plt.figure(figsize=(10 , 8))
plt.plot(cv_results_d['param_max_depth'] , cv_results_d['mean_test_score'])
plt.plot(cv_results_d['param_max_depth'] , cv_results_d['mean_train_score'])
plt.xlabel('Min_samples_Leaf')
plt.ylabel('Accuracy')
plt.legend(['test accuracy', 'train accuracy'], loc='upper left')
plt.xscale('log')
```



In [122]:
```
print(tree_split.best_params_)
print(tree_ml.best_params_)
print(tree_d.best_params_)
```

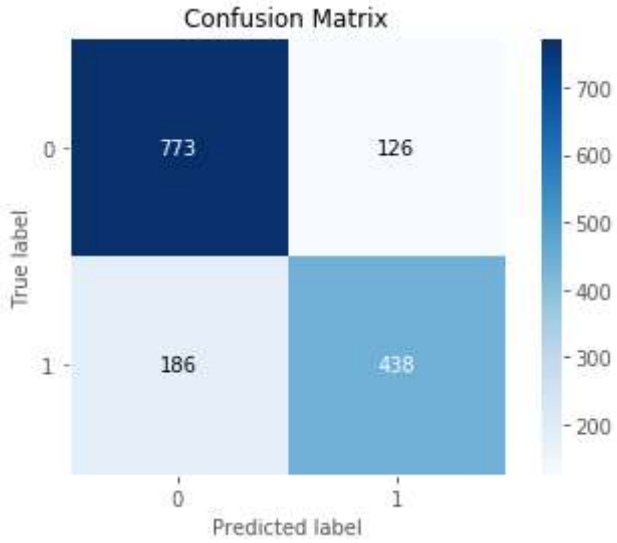
```
{'min_samples_leaf': 5}
{'min_samples_leaf': 1}
{'max_depth': 18}
```

In [87]:
```
dt = DecisionTreeClassifier(max_depth = 20 , min_samples_leaf = 10 , min_samples_
dt.fit(X_train , y_train)


y_pred_dt = dt.predict(X_test)

ac = accuracy_score(y_test , y_pred_dt)
print(ac*100)
```

```
66.57912015758372
```

In [148]:
```
optimal_depth = 20
optimal_split = 80

print("***Test Data Report***")
print("Best max_depth = ",optimal_depth)
print("Best min_samples_split = ",optimal_split)
fpr, tpr, threshold = metrics.roc_curve(y_test, lr.predict(X_test))
auc = metrics.auc(fpr, tpr)
print("AUC = ",auc*100)
skplt.metrics.plot_confusion_matrix(y_test, pred)
plt.show()
```

```
***Test Data Report***
Best max_depth =  20
Best min_samples_split =  80
AUC =  75.35072801688486
```



Random Forest on TFIDF-W2V

In [88]:
```
max_depths = [2,4,6,9,11]
base_learners = [1, 5, 10, 50, 100]
param_grid = {'max_depth': max_depths,'n_estimators':base_learners}


model = GridSearchCV(RandomForestClassifier(class_weight='balanced'), param_grid
model.fit(X_train, y_train)

print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_train, y_train))
```

```
Model with best parameters :
 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                        criterion='gini', max_depth=11, max_features='auto',
                        max_leaf_nodes=None, max_samples=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=100,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
Accuracy of the model :  0.9823638705186446
```

In [89]:
```
y_pred_rf = model.predict(X_test)
ac = accuracy_score(y_test , y_pred_rf)
ac
```

Out[89]:
```
0.7347340774786605
```

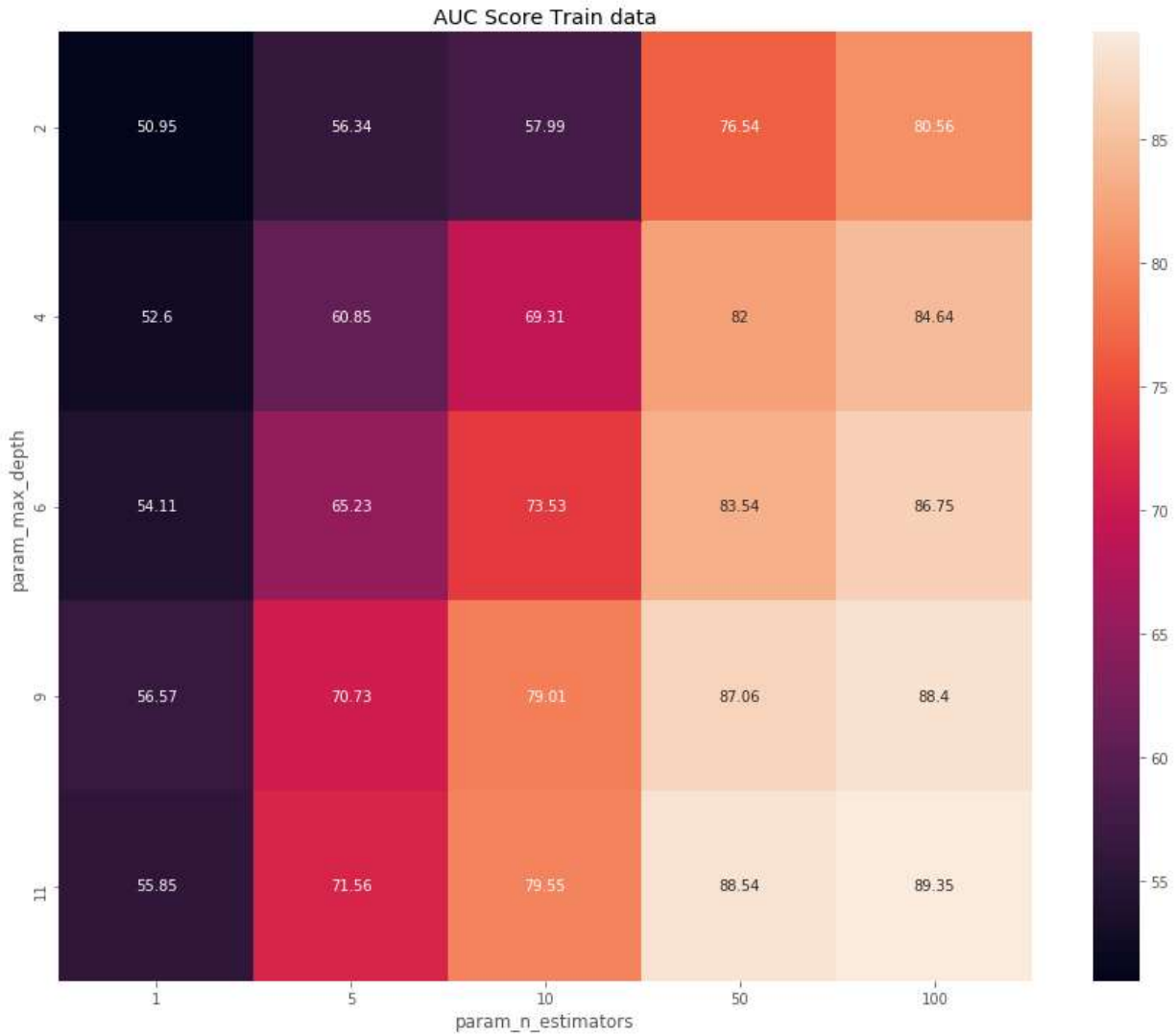
In [160]:
```
dataframe = pd.DataFrame(model.cv_results_)
dataframe.head()
```

Out[160]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estim
0	0.016784	0.003406	0.000675	0.000955	2	
1	0.043731	0.009535	0.012799	0.005317	2	
2	0.074657	0.012385	0.009410	0.000942	2	
3	0.287726	0.008659	0.035086	0.006827	2	
4	0.587622	0.009146	0.065997	0.007233	2	

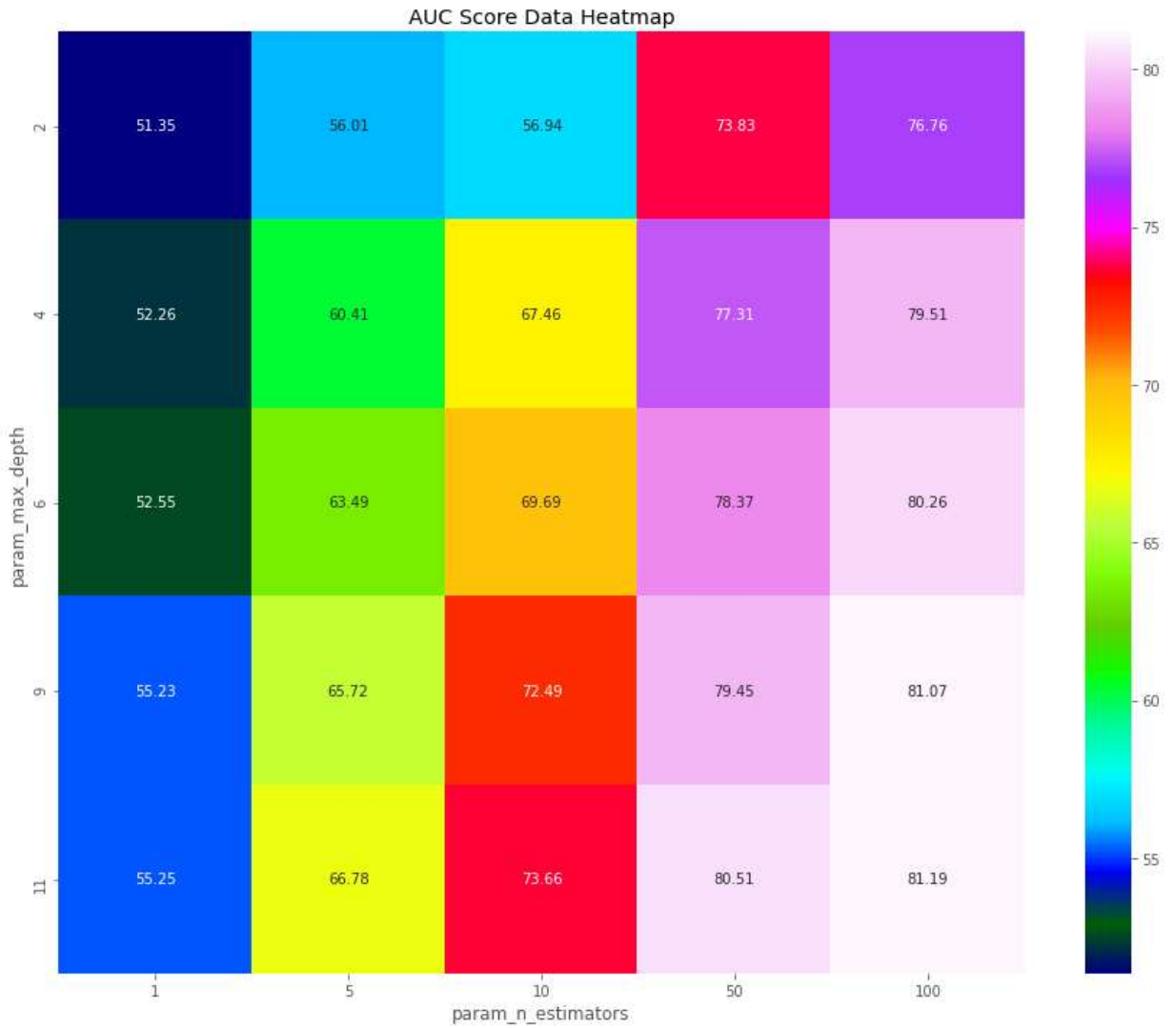
In [166]:
```
# Train Data Auc Score Vs hyperparameter Heatmap
max_scores = dataframe.groupby(['param_max_depth',
                                'param_n_estimators']).max()
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
plt.figure(figsize = (15 , 12))
sns.heatmap(max_scores.mean_train_score*100, annot=True, fmt='.4g');
ax = plt.axes()
ax.set_title('AUC Score Train data')
plt.show()
```

```
C:\Users\Mahmudur Limon\Anaconda3\envs\gputest\lib\site-packages\ipykernel_launcher.py:7: MatplotlibDeprecationWarning:

Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
```



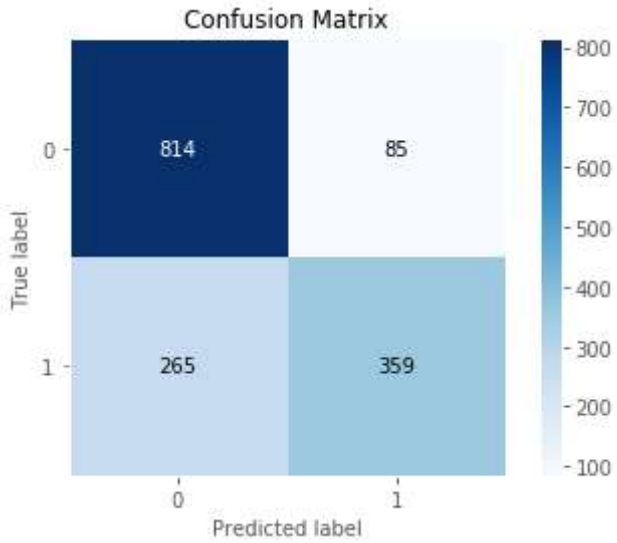
In [165]:
```
# CV Data Auc Score Vs hyperparameter Heatmap
max_scores = dataframe.groupby(['param_max_depth',
                                'param_n_estimators']).max()
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
plt.figure(figsize = (15 , 12))
sns.heatmap(max_scores.mean_test_score*100, annot=True, fmt='.4g' , cmap = 'gist_
ax = plt.axes()
ax.set_title('AUC Score Data Heatmap')
plt.show()
```

```
C:\Users\Mahmudur Limon\Anaconda3\envs\gputest\lib\site-packages\ipykernel_launcher.py:7: MatplotlibDeprecationWarning:

Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a future version, a new instance will always be created and returned.  Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
```



In [169]:
```
optimal_depth = 11
optimal_estimators = 100

lr = RandomForestClassifier(n_estimators=optimal_estimators, max_depth=optimal_d
lr.fit(X_train_BOW,y_train)
pred = lr.predict(X_test_BOW)

print("***Test Data Report***")
print("Best max_depth = ",optimal_depth)
print("Best Base Learners = ",optimal_estimators)
fpr, tpr, threshold = metrics.roc_curve(y_test, lr.predict(X_test))
auc = metrics.auc(fpr, tpr)
print("AUC = ",auc*100)
skplt.metrics.plot_confusion_matrix(y_test, pred)
plt.show()
```

```
***Test Data Report***
Best max_depth =  11
Best Base Learners =  100
AUC =  74.05646587376287
```



In [90]:
```
rf = RandomForestClassifier()
rf.fit(X_train , y_train)

y_pred_rf = rf.predict(X_test)

ac_rf = accuracy_score(y_test , y_pred_rf)
print(ac_rf*100)
```

```
72.94812869336835
```

In [228]:
```
test_pred = rf.predict(test_BOW)
```

In [229]:
```
# test_pred = rf.predict(test_BOW)
# submission['target'] = test_pred.astype(int)
# submission.to_csv('submission02.csv', index=False)
```

In [174]:
```
n_folds = 10
parameters = {'max_depth': range(2, 30, 2)}

rf = RandomForestClassifier(random_state = 100)
grd_search_max = GridSearchCV(rf , parameters , cv = n_folds , n_jobs = -1 , retu

grd_search_max.fit(X_train , y_train)
print('Best parameter for max_depth: ',grd_search_max.best_params_)

scores = grd_search_max.cv_results_
pd.DataFrame(scores).head()
```

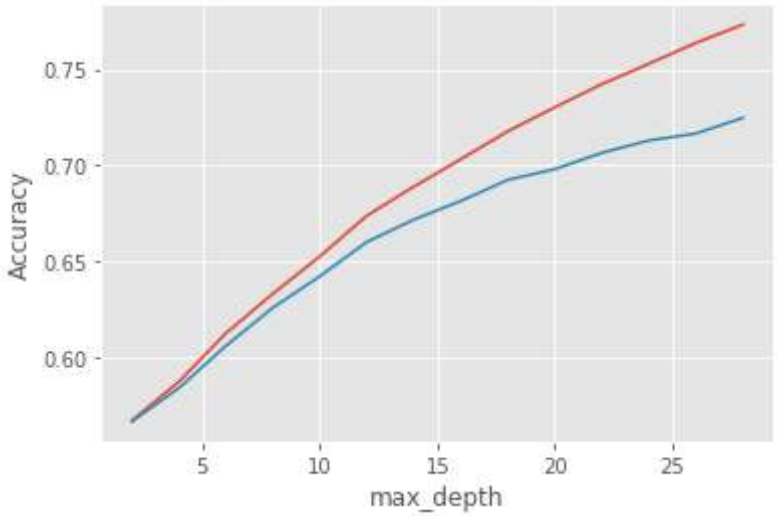
```
Best parameter for max_depth:  {'max_depth': 28}
```

Out[174]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	s
0	1.097105	0.101208	0.054257	0.011452	2	{'max_depth': 2}	
1	1.320912	0.090097	0.050891	0.016362	4	{'max_depth': 4}	
2	1.509562	0.060344	0.037270	0.007206	6	{'max_depth': 6}	
3	1.859384	0.052908	0.034113	0.005775	8	{'max_depth': 8}	
4	2.560092	0.079024	0.051421	0.013828	10	{'max_depth': 10}	

5 rows × 31 columns

In [175]:
```
plt.figure()
plt.plot(scores["param_max_depth"],scores["mean_train_score"],label="Training ac
plt.plot(scores["param_max_depth"],scores["mean_test_score"],label="Test accurac
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.show()
```



In [177]:
```
n_folds = 5
parameters = {'n_estimators': range(50, 500, 50)}

rf = RandomForestClassifier(random_state = 100)
grd_search_est = GridSearchCV(rf , parameters , cv = n_folds , n_jobs = -1 , ret

grd_search_est.fit(X_train, y_train)
print('Best parameter for n_estimators: ',grd_search_est.best_params_)

scores_1 = grd_search_est.cv_results_
pd.DataFrame(scores_1).head()
```

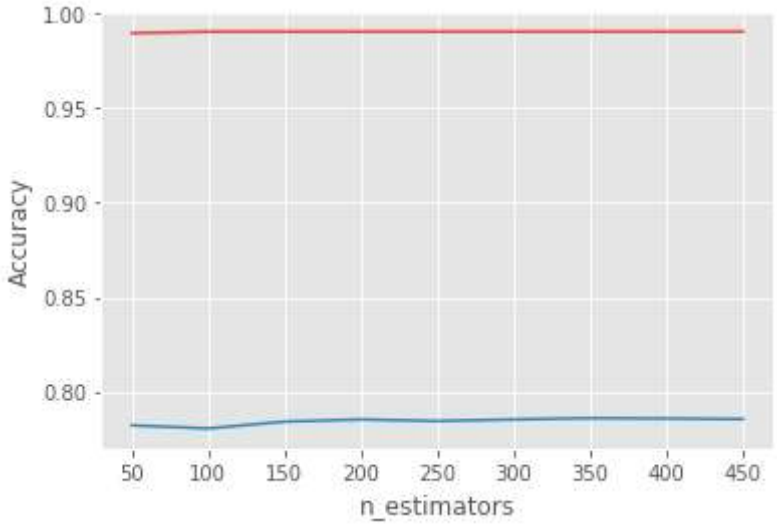
```
Best parameter for n_estimators:  {'n_estimators': 350}
```

Out[177]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	params
0	16.495535	0.501997	0.091412	0.012973	50	{'n_estimators' 50
1	34.319227	1.058878	0.229507	0.038290	100	{'n_estimators' 100
2	50.603185	0.384209	0.289535	0.035433	150	{'n_estimators' 150
3	66.416147	2.983965	0.445283	0.053172	200	{'n_estimators' 200
4	71.182656	8.838264	0.355182	0.004933	250	{'n_estimators' 250

5 rows × 21 columns

In [178]:
```
plt.figure()
plt.plot(scores_1["param_n_estimators"],scores_1["mean_train_score"],label="Trai
plt.plot(scores_1["param_n_estimators"],scores_1["mean_test_score"],label="Test 
plt.xlabel('n_estimators')
plt.ylabel('Accuracy')
plt.show()
```



In [179]:
```
n_folds = 5
parameters = {'min_samples_leaf': range(30, 400, 50)}

rf = RandomForestClassifier(random_state = 100)
grd_search_lf = GridSearchCV(rf , parameters , cv = n_folds , return_train_score

grd_search_lf.fit(X_train, y_train)
print('Best parameter for min_samples_leaf: ',grd_search_lf.best_params_)

scores_2 = grd_search_lf.cv_results_
pd.DataFrame(scores_2).head()
```

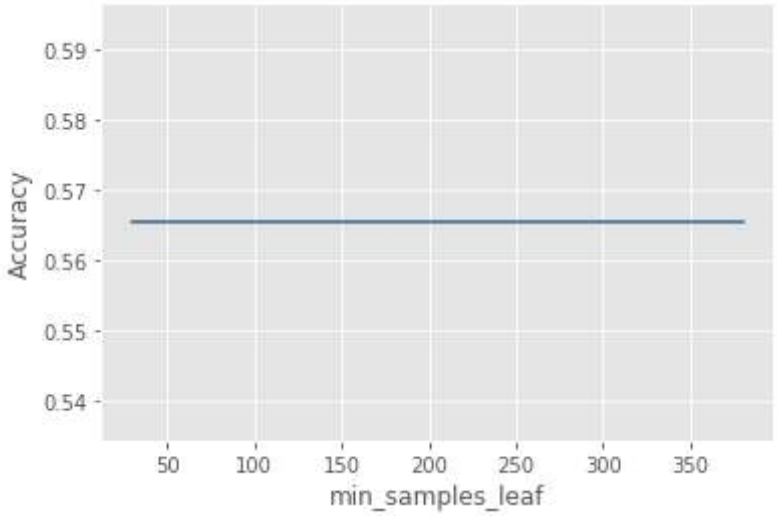
```
Best parameter for min_samples_leaf:  {'min_samples_leaf': 30}
```

Out[179]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_leaf	
0	1.286695	0.315587	0.101946	0.083947	30	{'min_sar
1	0.847230	0.133691	0.088366	0.043182	80	{'min_sar
2	0.556950	0.024576	0.045469	0.006044	130	{'min_sar
3	0.666235	0.397856	0.044280	0.015913	180	{'min_sar
4	0.697134	0.398707	0.046674	0.034540	230	{'min_sar

5 rows × 21 columns

In [180]:
```
plt.figure()
plt.plot(scores_2["param_min_samples_leaf"],scores_2["mean_train_score"],label="
plt.plot(scores_2["param_min_samples_leaf"],scores_2["mean_test_score"],label="T
plt.xlabel('min_samples_leaf')
plt.ylabel('Accuracy')
plt.show()
```



In [184]:
```
n_folds = 5
parameters = {'min_samples_split': range(10, 400, 50)}

rf = RandomForestClassifier()
grd_search_sp = GridSearchCV(rf , parameters , cv = n_folds , n_jobs = -1, retur

grd_search_sp.fit(X_train, y_train)
print('Best parameter for min_samples_split: ',grd_search_sp.best_params_)

score_3 = grd_search_sp.cv_results_
pd.DataFrame(score_3).head()
```

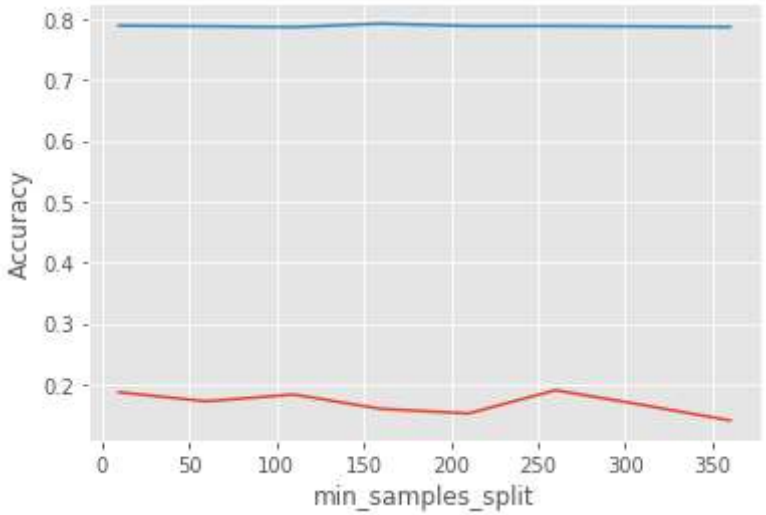
```
Best parameter for min_samples_split:  {'min_samples_split': 160}
```

Out[184]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	
0	21.406171	0.767308	0.187100	0.008430	10	{'min_sa
1	17.413395	0.232521	0.172140	0.007559	60	{'min_sa
2	16.887572	0.341539	0.183394	0.032281	110	{'min_sa
3	17.409879	0.796128	0.159774	0.019540	160	{'min_sa
4	17.294000	0.458271	0.152191	0.012641	210	{'min_sa

5 rows × 21 columns

In [185]:
```
plt.figure()
plt.plot(score_3["param_min_samples_split"],score_3["mean_score_time"],label="Tr
plt.plot(score_3["param_min_samples_split"],score_3["mean_test_score"],label="Te
plt.xlabel('min_samples_split')
plt.ylabel('Accuracy')
plt.show()
```



In [186]:
```
print('Best parameter for max_depth: ',grd_search_max.best_params_)
print('Best parameter for min_samples_leaf: ',grd_search_lf.best_params_)
print('Best parameter for min_samples_split: ',grd_search_sp.best_params_)
print('Best parameter for n_estimators: ',grd_search_est.best_params_)
```

```
Best parameter for max_depth:  {'max_depth': 28}
Best parameter for min_samples_leaf:  {'min_samples_leaf': 30}
Best parameter for min_samples_split:  {'min_samples_split': 160}
Best parameter for n_estimators:  {'n_estimators': 350}
```

In [194]:
```
new_rf = RandomForestClassifier(n_estimators = 350 , max_depth = 28 , min_sample
new_rf.fit(X_train , y_train)

y_pred_rf = new_rf.predict(X_test)

ac_rf = accuracy_score(y_test , y_pred_rf)
print(ac_rf*100)
```

```
59.02823374917925
```

GBDT on TFIDF-W2V

In [196]:
```
learn_rate = {'learning_rate': [0.001 , 0.01 , 0.1 , 1.0 , 1.3]}
folds = 5
```

In [200]:
```
GBC = GradientBoostingClassifier()

grd_search_lr = GridSearchCV(GBC , cv = folds , param_grid = learn_rate , n_jobs
grd_search_lr.fit(X_train, y_train)
print('Best parameter for max_depth: ' , grd_search_lr.best_params_)

cv_results_gbdt = pd.DataFrame(grd_search_lr.cv_results_)
cv_results_gbdt.head()
```

```
Best parameter for max_depth:  {'learning_rate': 1.0}
```

Out[200]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param
0	20.821945	0.444296	0.005187	0.000746	0.001	{'learning_rate 0.001
1	19.947538	0.141624	0.008776	0.002555	0.01	{'learning_rate 0.01
2	19.750963	0.283580	0.006586	0.001736	0.1	{'learning_rate 0.1
3	21.138318	0.932057	0.006097	0.000482	1	{'learning_rate 1.0
4	18.426071	2.278891	0.004915	0.001023	1.3	{'learning_rate 1.3

In [203]:
```
sub_sample = {"subsample": [0.3, 0.6, 0.9]}
folds = 5
```

In [204]:
```
GBC = GradientBoostingClassifier(max_depth=2, n_estimators=200)

grd_search_ss = GridSearchCV(GBC , cv = folds , param_grid = sub_sample)
grd_search_ss.fit(X_train, y_train)
print('Best parameter for max_depth: ',grd_search_ss.best_params_)

cv_results_sub = pd.DataFrame(grd_search_lr.cv_results_)
cv_results_sub.head()
```

```
Best parameter for max_depth:  {'subsample': 0.6}
```

Out[204]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param
0	20.821945	0.444296	0.005187	0.000746	0.001	{'learning_rate 0.001
1	19.947538	0.141624	0.008776	0.002555	0.01	{'learning_rate 0.01
2	19.750963	0.283580	0.006586	0.001736	0.1	{'learning_rate 0.1
3	21.138318	0.932057	0.006097	0.000482	1	{'learning_rate 1.0
4	18.426071	2.278891	0.004915	0.001023	1.3	{'learning_rate 1.3

In [217]:
```
#GBC = GradientBoostingClassifier(subsample = 0.6 , learning_rate = 0.1 , n_estir

GBC = GradientBoostingClassifier(subsample = 0.6 , learning_rate = 1.0 ,  n_estir
GBC.fit(X_train , y_train)

y_gbr = GBC.predict(X_test)

ac_gbr = accuracy_score(y_test , y_gbr)
print(ac_gbr*100)
```

```
77.93827971109653
```

XGB on TFIDF-W2V

In [91]:
```
classifier = XGBClassifier()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

ac_xg = accuracy_score(y_test , y_pred)
print(ac_xg*100)
```

```
71.37229152987524
```

In [92]:
```
max_depths = [2,4,6,9,11]
base_learners = [1, 5, 10, 50, 100]
param_grid = {'max_depth': max_depths,'n_estimators':base_learners}

model = GridSearchCV(xgb.XGBClassifier(scale_pos_weight=1), param_grid, scoring =
model.fit(X_train, y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_train, y_train))
```

```
Model with best parameters :
 XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=9,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
              validate_parameters=False, verbosity=None)
Accuracy of the model :  0.9994301935531442
```

In [17]:
```
optimal_depth = 9
optimal_estimators = 100


xg_model = xgb.XGBClassifier(max_depth=optimal_depth, n_estimators=optimal_estima
xg_model.fit(X_train,y_train)

pred = xg_model.predict(X_test)

print("***Test Data Report***")
print("Best max_depth = ",optimal_depth)
print("Best Base Learners = ",optimal_estimators)
fpr, tpr, threshold = metrics.roc_curve(y_test, lr.predict(X_test))
auc = metrics.auc(fpr, tpr)
print("AUC = ",auc*100)
skplt.metrics.plot_confusion_matrix(y_test, pred)
plt.show()
```

In [230]:
```
# test_pred = xg_model.predict(test_BOW)

# submission['target'] = test_pred.astype(int)
# submission.to_csv('submission02xgb.csv', index=False)
```