logisticRegression (1)

April 7, 2020

[16]:
```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.calibration import CalibratedClassifierCV , calibration_curve
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, KFold
import sqlite3
from tqdm import tqdm
import matplotlib.patches as patches
import  warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings('ignore', 'Solver terminated early.*')
import string
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import roc_curve,auc
from scipy import interp
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,␣
↪StratifiedKFold
from sklearn.metrics import f1_score
import pickle
%matplotlib inline
```

[17]:
```
con = sqlite3.connect("/home/niranjan/Downloads/database.sqlite")
data = pd.read_sql_query("select * from Reviews where Score!=3",con)
data['Score'] = [1 if i>3 else 0 for i in data['Score']]
```

Removing Duplicate Data

[18]:
```
df = data.sort_values(by= 'Time',ascending=True,inplace=False,kind='quicksort')
data_without_dup = df.
 ↪drop_duplicates(subset={'UserId','ProfileName','Time','Text'},␣
 ↪inplace=False,keep='first')
data_without_dup = data_without_dup[data_without_dup.
 ↪HelpfulnessNumerator<=data_without_dup.HelpfulnessDenominator]
```

[19]:
```
import nltk
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
nltk.download('stopwords')
nltk.download('wordnet')
stop = list(stopwords.words('english'))
lem = WordNetLemmatizer()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/niranjan/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/niranjan/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[20]:
```
import re
def clean_html(words):
    tag = re.compile(r'<.?>')
    cleanSent = re.sub(tag,'', words)
    return cleanSent


def punch_remove(words):
    tag = re.compile(r'[^a-zA-Z]')
    cleanSent = re.sub(tag,'',words)
    return cleanSent

```

****——————-data cleaning——————-****

---

removal of html tags, symbols other than alphabets, stopwords and performing lemmatization as part of data pre-processing.

Lemmatization :-Process of grouping together the inflected forms of a word, it can be analysed as a single item, identified by the word's lemma, or dictionary form.**

[21]:
```
final_data = []
str1 = ' '
positive_words = []
```

```
negative_words = []
i=0
for sen in data_without_dup['Text'].values:
    filter_word = []
    pos_word = []
    neg_word = []
    sent= clean_html(sen)
    for word in sent.split():
        cleanwords = punch_remove(word)
        for cleanword in cleanwords.split():
            if((len(cleanword) >2) & (cleanword.isalpha())):
                if((cleanword.lower() not in stop)):
                   w = (lem.lemmatize(cleanword.lower())).encode('utf-8')
                   filter_word.append(w)
                   if data_without_dup['Score'].values[i] == 1 :
                       pos_word.append(w)
                   else :
                       neg_word.append(w)
                else :
                   continue
            else :
                continue
    str1 = b" ".join(filter_word)
    str2 = b" ".join(pos_word)
    str3 = b" ".join(neg_word)
    final_data.append(str1)
    positive_words.append(str2)
    negative_words.append(str3)
    i = i + 1
```

[8]:
```
data_without_dup['final_string'] = final_data
data_without_dup['Positive_string'] = positive_words
data_without_dup['Negative_string'] = negative_words
data_without_dup['final_string'] = data_without_dup['final_string'].str.
 ↪decode('utf8')
data_without_dup['Positive_string'] = data_without_dup['Positive_string'].str.
 ↪decode('utf8')
data_without_dup['Negative_string'] = data_without_dup['Negative_string'].str.
 ↪decode('utf8')
```

[9]:
```
X = data_without_dup['final_string']
y =data_without_dup['Score']
```

[10]:
```
X_train= X[0:100000]
y_train = y[0:100000]
X_test= X[100000:120000]
y_test = y[100000:120000]
```

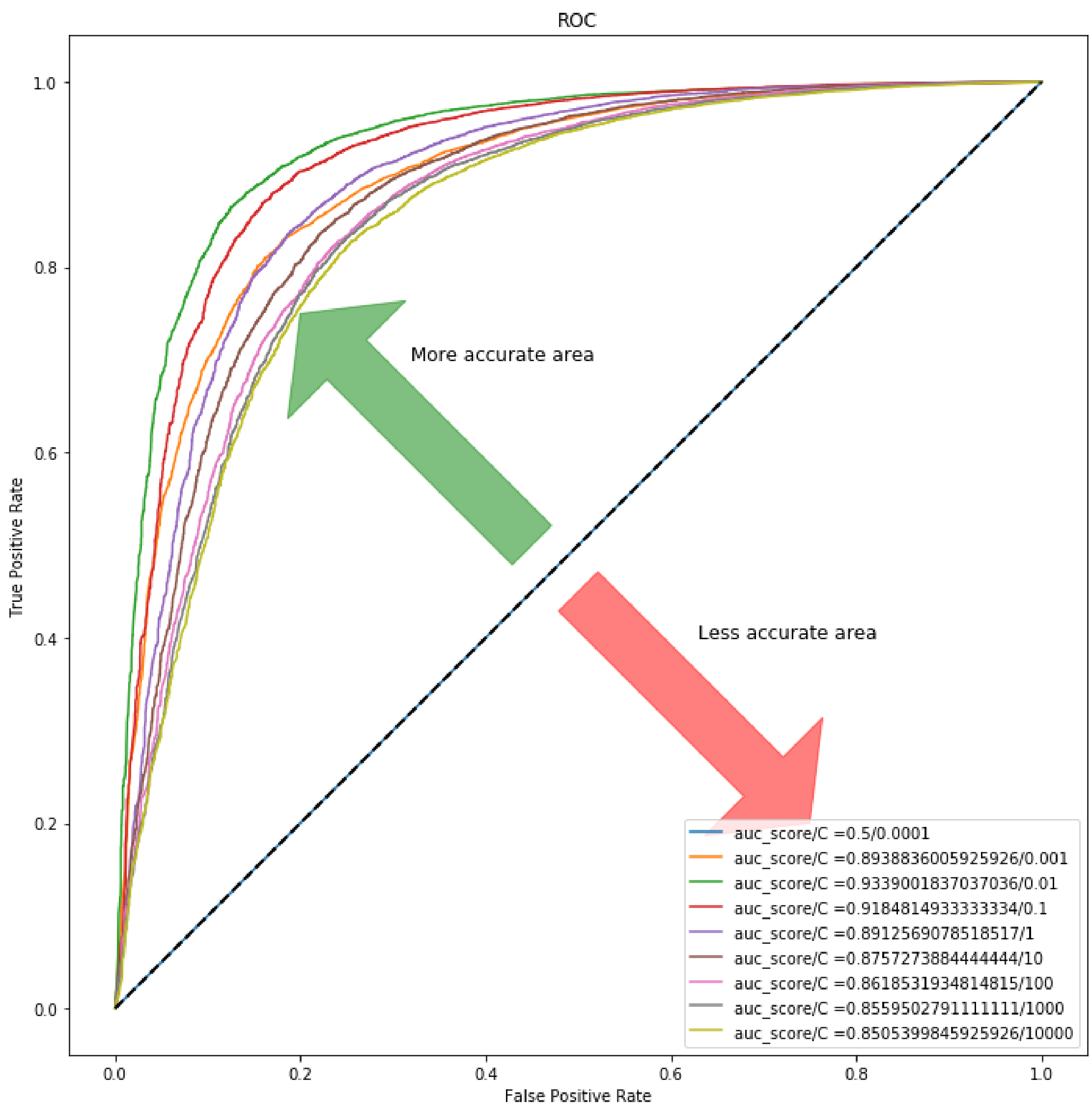
BOW as vectorizer with Standardscaler

[14]:
```
count_vect = CountVectorizer()
X_train_bow = count_vect.fit_transform(X_train)
X_test_bow = count_vect.transform(X_test)
count_vec = StandardScaler(with_mean=False)
X_train_bow = count_vec.fit_transform(X_train_bow)
X_test_bow = count_vec.transform(X_test_bow)
```

```
/home/niranjan/anaconda3/lib/python3.6/site-
packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input
dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/home/niranjan/anaconda3/lib/python3.6/site-
packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input
dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/home/niranjan/anaconda3/lib/python3.6/site-
packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input
dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

[67]:
```
acparam = [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10, 100, 1000, 10000]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
     patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
)
ax1.add_patch(
     patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
)
mean_fpr = np.linspace(0,1,100)
for i in acparam:
    clf = LogisticRegression(penalty='l1',n_jobs=1,class_weight='balanced',C=i)
    model = CalibratedClassifierCV(clf,cv=3,method ='isotonic')
    model.fit(X_train_bow,y_train)
    mod_probs = model.predict_proba(X_test_bow)[:,1]
    fpr, tpr, thresholds = metrics.roc_curve(y_test,  mod_probs)
    auc = metrics.roc_auc_score(y_test, mod_probs)
    tprs.append(interp(mean_fpr, fpr, tpr))
    #print("auc value for {} is {}".format(i,auc))
    plt.plot(fpr,tpr,label="auc_score/C ="+str(auc) +"/"+str(i))
    plt.legend(loc=4)
#     plt.plot(fpr,tpr,label="auc_score/alpha ="+str(auc) +"/"+str(i))
#     plt.legend(loc=4)
#     plt.show()
plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
# mean_tpr = str(np.mean(tprs, axis=0))
```

```
# mean_auc = auc(mean_fpr,mean_tpr)
#plt.plot(mean_fpr, mean_tpr, color='blue',label=r'Mean ROC (AUC = %0.2f )' %␣
 ↪(mean_auc),lw=2, alpha=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")
plt.text(0.32,0.7,'More accurate area',fontsize = 12)
plt.text(0.63,0.4,'Less accurate area',fontsize = 12)
plt.show()
```



[170]:
```
classifier = LogisticRegression(penalty='l1',n_jobs=1,C=0.01)
classifier.fit(X_train_bow,y_train)
```

[170]:
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=1,
          penalty='l1', random_state=None, solver='warn', tol=0.0001,
          verbose=0, warm_start=False)
```

[171]:
```
w = classifier.coef_
print(w.shape)
print(np.count_nonzero(w))
```

```
(1, 88598)
8153
```

[172]:
```
X_train_bow.data = X_train_bow.data + 0.001
```

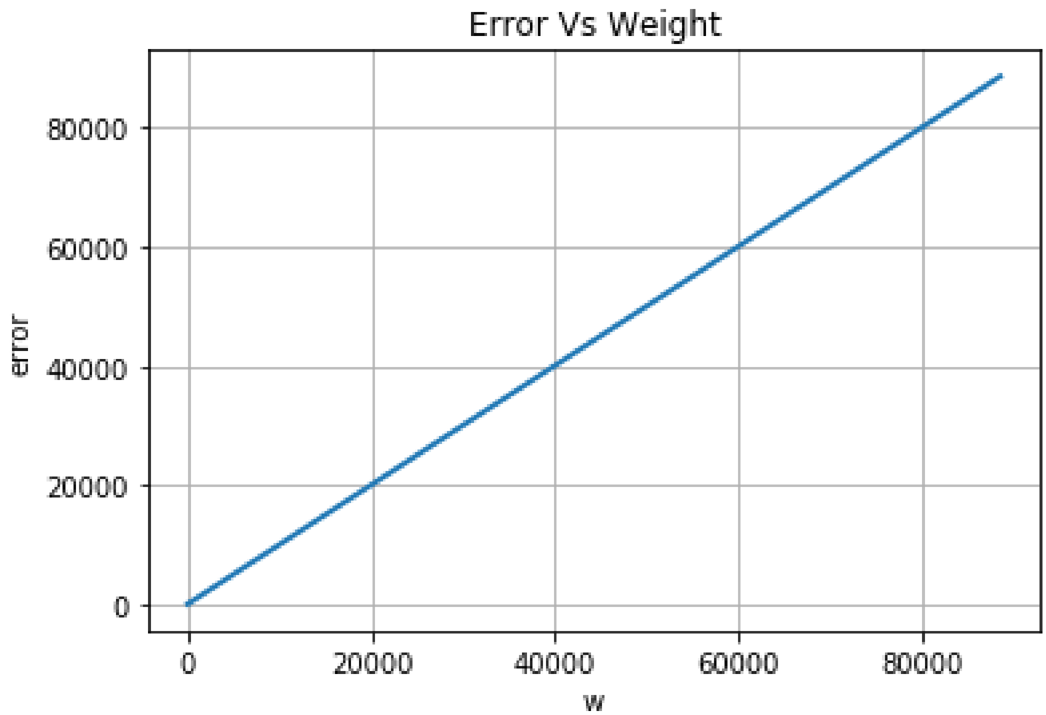
[173]:
```
classifier.fit(X_train_bow,y_train)
w_new = classifier.coef_
print(np.count_nonzero(w_new))
print(len(w))
```

```
8185
1
```

0.1 Perturbation:

addition of random noise(epsilon), here epsilon(here epsilon value is .001) to existing dataset to observe variation in statistical behaviour of data.

[175]:
```
w = w + 10**-6
w_new = w_new + 10**-6
weight_change = np.absolute(((w-w_new)) + 100 /( w + 100))
weight_change_cum = np.cumsum(weight_change)
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(weight_change_cum, linewidth=2)
plt.axis('tight')
plt.xlabel('w')
plt.ylabel('error')
plt.title("Error Vs Weight")
plt.grid()
plt.show()
```

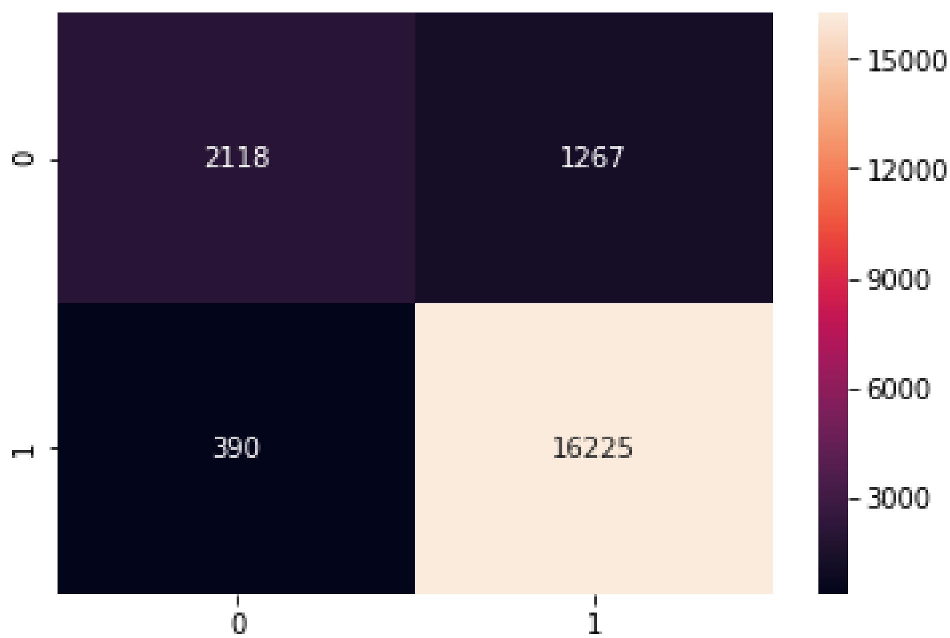


[176]:
```
print("0-percentile weight change is {}".format(np.percentile(weight_change,0)))
print("10-percentile weight change is {}".format(np.
 ↪percentile(weight_change,10)))
print("20-percentile weight change is {}".format(np.
 ↪percentile(weight_change,20)))
print("30-percentile weight change is {}".format(np.
 ↪percentile(weight_change,30)))
print("40-percentile weight change is {}".format(np.
 ↪percentile(weight_change,40)))
print("50-percentile weight change is {}".format(np.
 ↪percentile(weight_change,50)))
print("60-percentile weight change is {}".format(np.
 ↪percentile(weight_change,60)))
print("70-percentile weight change is {}".format(np.
 ↪percentile(weight_change,70)))
print("80-percentile weight change is {}".format(np.
 ↪percentile(weight_change,80)))
print("90-percentile weight change is {}".format(np.
 ↪percentile(weight_change,90)))
print("100-percentile weight change is {}".format(np.
 ↪percentile(weight_change,100)))
```

```
0-percentile weight change is 0.9821830968646674
```

```
10-percentile weight change is 0.9999999800000003
20-percentile weight change is 0.9999999800000003
30-percentile weight change is 0.9999999800000005
40-percentile weight change is 0.9999999800000005
50-percentile weight change is 0.9999999800000005
60-percentile weight change is 0.9999999800000005
70-percentile weight change is 0.9999999800000005
80-percentile weight change is 0.9999999800000003
90-percentile weight change is 0.9999999800000003
100-percentile weight change is 1.0224150296316836
```

[44]:
```
y_pred = classifier.predict(X_test_bow)
acc = f1_score(y_pred,y_test,average='micro')*100
df = pd.DataFrame(confusion_matrix(y_test,y_pred))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
print('\nThe accuracy of the logistics regression classifier for lambda = %d is␣
↪%f%%' % (100, acc))
```




```
The accuracy of the logistics regression classifier for lambda = 100 is
91.715000%
```

[80]:
```
y_pred = classifier.predict(X_test_bow)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.54      0.66      3125
           1       0.92      0.98      0.95     16875

   micro avg       0.91      0.91      0.91     20000
   macro avg       0.88      0.76      0.80     20000
weighted avg       0.91      0.91      0.90     20000
```

[18]:
```
clf = LogisticRegression(C=0.01,penalty='l1')
clf.fit(X_train_bow,y_train)
pred = clf.predict(X_test_bow)
# evaluate accuracy
acc = f1_score(y_test, pred,average='micro')*100
print("The accuracy of the logistic regression classifier for alpha {lamda} is␣
 ↪{b}%".format(lamda = 100, b = acc))
print("----------------------------------------------------------")
class_labels = clf.classes_
feature_names =count_vect.get_feature_names()
topn_class1 = sorted(zip(clf.predict_log_proba(X_test_bow)[:,0],␣
 ↪feature_names))[:10]
topn_class2 = sorted(zip(clf.predict_log_proba(X_test_bow)[:,1],␣
 ↪feature_names))[:10]
print("Important words in negative reviews")
for coef, feat in topn_class1:
    print(class_labels[0], coef, feat)
print("-----------------------------------------")
print("Important words in positive reviews")
for coef, feat in topn_class2:
    print(class_labels[1], coef, feat)
```

```
The accuracy of the logistic regression classifier for alpha 100 is 91.13%
----------------------------------------------------------
Important words in negative reviews
0 -27.02000410645287 bounced
0 -22.3249720641345 comparision
0 -22.200629809414384 cokebr
0 -21.488486657746066 coverbr
0 -20.750353969774594 decadant
0 -20.52818912271801 coagulant
0 -19.636253931429245 cutie
0 -19.466055619763058 dauchand
0 -19.179290258201863 anetto
0 -18.64453479944347 arthritisbr
-----------------------------------------
Important words in positive reviews
```

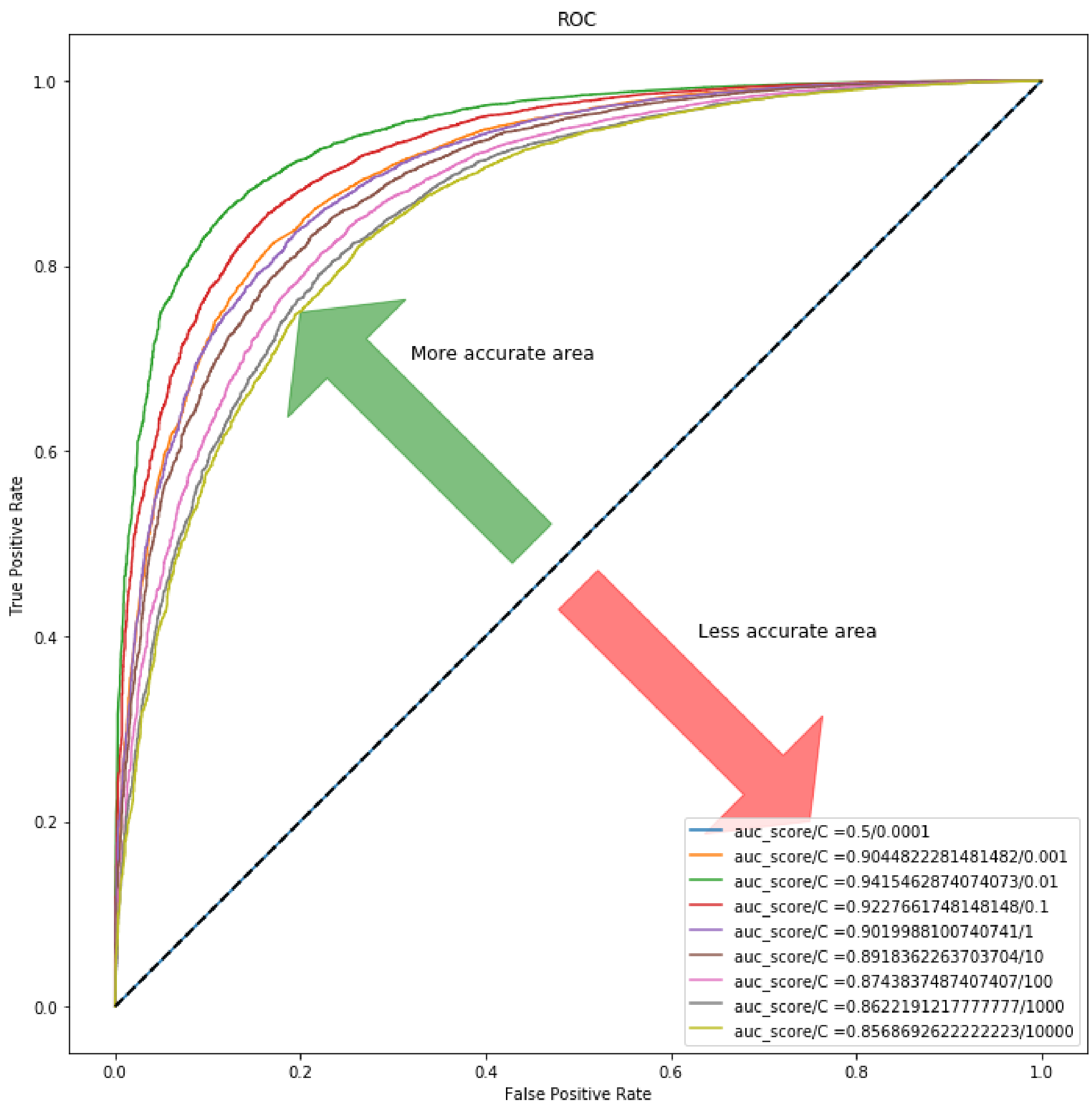
```
1 -23.900573869573332 chipaddicted
1 -21.776566857304122 caseit
1 -20.096166391200427 cleaningthe
1 -17.0018530193198 bitterish
1 -16.725721466566704 dealsbr
1 -16.407771498405396 citronmandarinpeach
1 -13.877218744770458 bombaybr
1 -13.684158403513871 buttah
1 -13.4772072785796 amaze
1 -13.1434941640128 cornpuff
```

tfidf as vectorizer with Standardscaler

[11]:
```
tfidf_vec = TfidfVectorizer()
X_train_tfidf = tfidf_vec.fit_transform(X_train)
X_test_tfidf = tfidf_vec.transform(X_test)
vec = StandardScaler(with_mean=False)
X_train_tfidf = vec.fit_transform(X_train_tfidf)
X_test_tfidf = vec.transform(X_test_tfidf)
```

[19]:
```
acparam = [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10, 100, 1000, 10000]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
     patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
)
ax1.add_patch(
     patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
)
mean_fpr = np.linspace(0,1,100)
for i in acparam:
    clf = LogisticRegression(penalty='l1',n_jobs=1,class_weight='balanced',C=i)
    model = CalibratedClassifierCV(clf,cv=3,method ='isotonic')
    model.fit(X_train_tfidf,y_train)
    mod_probs = model.predict_proba(X_test_tfidf)[:,1]
    fpr, tpr, thresholds = metrics.roc_curve(y_test,  mod_probs)
    auc = metrics.roc_auc_score(y_test, mod_probs)
    #print("auc value for {} is {}".format(i,auc))
    plt.plot(fpr,tpr,label="auc_score/C ="+str(auc) +"/"+str(i))
    plt.legend(loc=4)
#     plt.plot(fpr,tpr,label="auc_score/alpha ="+str(auc) +"/"+str(i))
#     plt.legend(loc=4)
#     plt.show()
plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
# mean_tpr = str(np.mean(tprs, axis=0))
# mean_auc = auc(mean_fpr,mean_tpr)
#plt.plot(mean_fpr, mean_tpr, color='blue',label=r'Mean ROC (AUC = %0.2f )' %␣
 ↪(mean_auc),lw=2, alpha=1)
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")
plt.text(0.32,0.7,'More accurate area',fontsize = 12)
plt.text(0.63,0.4,'Less accurate area',fontsize = 12)
plt.show()
```



finding best params using GridsearchCV

[16]:
```
acparam = {'C':[10**-4, 10**-3, 10**-2, 10**-1, 1 , 10, 100, 1000, 10000]}
clf = LogisticRegression(penalty='l1',n_jobs=1,class_weight='balanced')
model = GridSearchCV(clf,param_grid=acparam,cv=3,scoring='roc_auc')
```

```
model.fit(X_train_tfidf,y_train)
```

[16]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=1, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

[17]:
```
print(model.best_estimator_)
```

```
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=1, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

0.2 Perturbation:

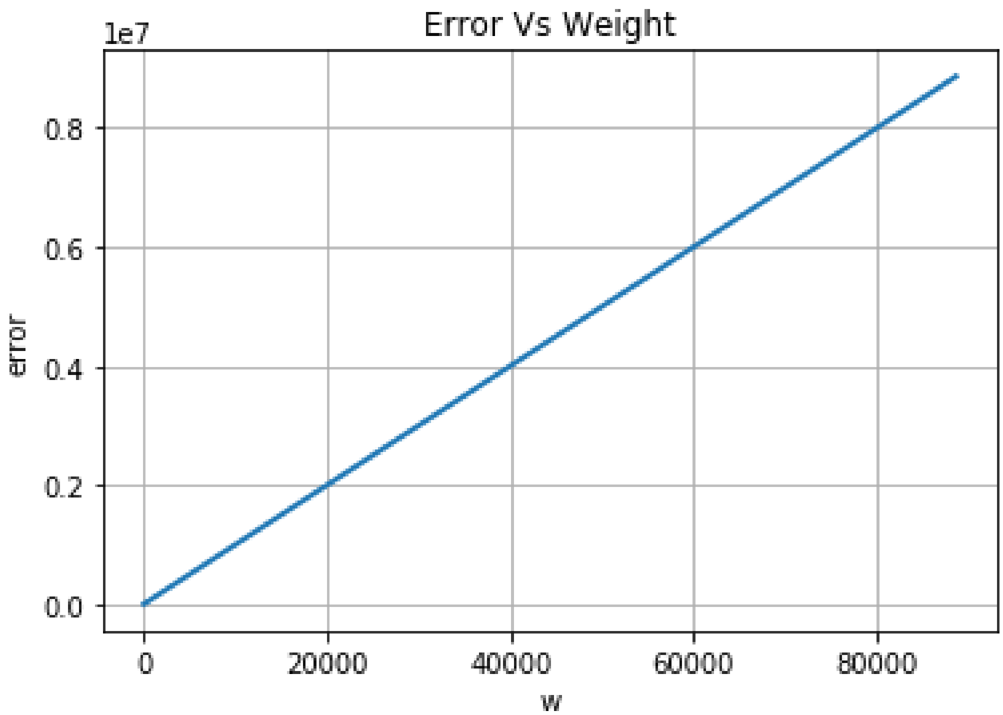
addition of random noise(epsilon), here epsilon(here epsilon value is .001) to existing dataset to observe variation in statistical behaviour of data.

[177]:
```
w = classifier.coef_
```

[178]:
```
X_train_tfidf.data = X_train_tfidf.data + 0.001
```

[179]:
```
classifier.fit(X_train_tfidf,y_train)
w_new = classifier.coef_
```

[180]:
```
w = w + 10**-6
w_new = w_new + 10**-6
weight_change = np.absolute(((w-w_new)) + 100 /( w + 100))*100
weight_change_cum = np.cumsum(weight_change)
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(weight_change_cum, linewidth=2)
plt.axis('tight')
plt.xlabel('w')
plt.ylabel('error')
plt.title("Error Vs Weight")
plt.grid()
plt.show()
```



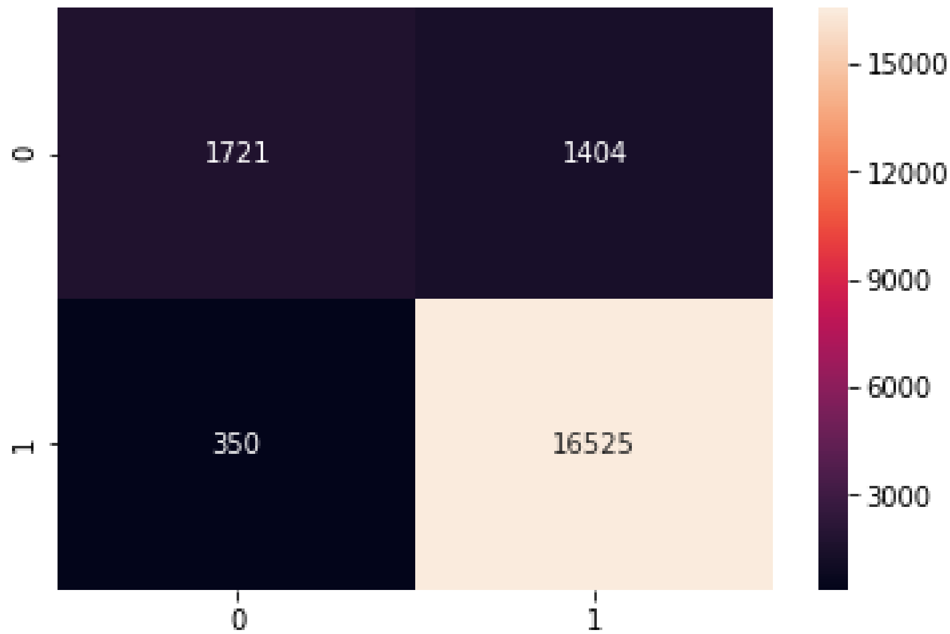
[181]:
```
print("0-percentile weight change is {}".format(np.percentile(weight_change,0)))
print("10-percentile weight change is {}".format(np.
 ↪percentile(weight_change,10)))
print("20-percentile weight change is {}".format(np.
 ↪percentile(weight_change,20)))
print("30-percentile weight change is {}".format(np.
 ↪percentile(weight_change,30)))
print("40-percentile weight change is {}".format(np.
 ↪percentile(weight_change,40)))
print("50-percentile weight change is {}".format(np.
 ↪percentile(weight_change,50)))
print("60-percentile weight change is {}".format(np.
 ↪percentile(weight_change,60)))
print("70-percentile weight change is {}".format(np.
 ↪percentile(weight_change,70)))
print("80-percentile weight change is {}".format(np.
 ↪percentile(weight_change,80)))
print("90-percentile weight change is {}".format(np.
 ↪percentile(weight_change,90)))
print("100-percentile weight change is {}".format(np.
 ↪percentile(weight_change,100)))
```

```
0-percentile weight change is 95.49140707655877
```

```
10-percentile weight change is 99.99999900000002
20-percentile weight change is 99.99999900000002
30-percentile weight change is 99.99999900000002
40-percentile weight change is 99.99999900000003
50-percentile weight change is 99.99999900000002
60-percentile weight change is 99.99999900000003
70-percentile weight change is 99.99999900000002
80-percentile weight change is 99.99999900000002
90-percentile weight change is 99.99999900000002
100-percentile weight change is 103.57210785605871
```

[91]:

```
y_pred = classifier.predict(X_test_tfidf)
acc = f1_score(y_pred,y_test,average='micro')*100
df = pd.DataFrame(confusion_matrix(y_test,y_pred))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
print('\nThe accuracy of the logistics regression classifier for lambda = %d is␣
 ↪%f%%' % (100, acc))
```




```
The accuracy of the logistics regression classifier for lambda = 100 is
91.230000%
```

[92]:

```
y_pred = classifier.predict(X_test_tfidf)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.55      0.66      3125
           1       0.92      0.98      0.95     16875

   micro avg       0.91      0.91      0.91     20000
   macro avg       0.88      0.76      0.81     20000
weighted avg       0.91      0.91      0.90     20000
```

[19]:

```
clf = LogisticRegression(C=0.01,penalty='l1')
clf.fit(X_train_tfidf,y_train)
pred = clf.predict(X_test_tfidf)
# evaluate accuracy
acc = f1_score(y_test, pred,average='micro')*100
print("The accuracy of the logistic regression classifier for alpha {lamda} is␣
 ↪{b}%".format(lamda = 100, b = acc))
print("----------------------------------------------------------")
class_labels = clf.classes_
feature_names =count_vect.get_feature_names()
topn_class1 = sorted(zip(clf.predict_log_proba(X_test_tfidf)[:,0],␣
 ↪feature_names))[:10]
topn_class2 = sorted(zip(clf.predict_log_proba(X_test_tfidf)[:,1],␣
 ↪feature_names))[:10]
print("Important words in negative reviews")
for coef, feat in topn_class1:
    print(class_labels[0], coef, feat)
print("-----------------------------------------")
print("Important words in positive reviews")
for coef, feat in topn_class2:
    print(class_labels[1], coef, feat)
```

```
The accuracy of the logistic regression classifier for alpha 100 is 91.24%
----------------------------------------------------------
Important words in negative reviews
0 -13.71481969424643 chlosterol
0 -13.630163662172007 clem
0 -13.537794064582307 abyssinian
0 -13.39758381540121 beige
0 -13.174760918546946 antagonizers
0 -13.131372769216608 bitingnuts
0 -13.108545550395194 creameran
0 -12.726170435488719 caramelsbr
0 -12.509086336915287 concoction
0 -12.482053969086861 consistencythey
-----------------------------------------
Important words in positive reviews
```

```
1 -10.545641701570243 breadthen
1 -10.072484555124836 delallos
1 -9.43550725004765 cage
1 -9.341389020456583 buckshot
1 -9.28492571903222 alteration
1 -9.219861194921391 cantalope
1 -9.148721793549845 decedant
1 -9.023448579157128 deform
1 -8.988829458437898 againill
1 -8.947648405963761 cleaningthe
```

Word2Vec as vectorizer

[14]:
```python
list_of_sent = []

for sent in data_without_dup['final_string'].values:
  list_of_sent.append(sent.split())
```

[15]:
```python
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

[ ]:
```python
model = KeyedVectors.load_word2vec_format('/home/niranjan/Downloads/data',␣
 ↪binary=True)
```

[149]:
```python
X_train_avg_w2v = []
w2v_model = Word2Vec(list_of_sent[0:100000],min_count=5,size=150,workers=4)
w2v_words = list(w2v_model.wv.vocab)

for sent in list_of_sent[0:100000]:
    sent_vec = np.zeros(150)
    count_words = 0
    for words in sent:
        if words in  w2v_words:
            vec = w2v_model.wv[words]
            sent_vec += vec
            count_words +=1
    if count_words !=0:
        sent_vec = sent_vec/count_words
    X_train_avg_w2v.append(sent_vec)
```

[150]:
```python
X_test_avg_w2v = []
w2v_model = Word2Vec(list_of_sent[100000:120000],min_count=5,size␣
 ↪=150,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in list_of_sent[100000:120000]:
    sent_vec = np.zeros(150)
    count_words = 0
    for words in sent:
```

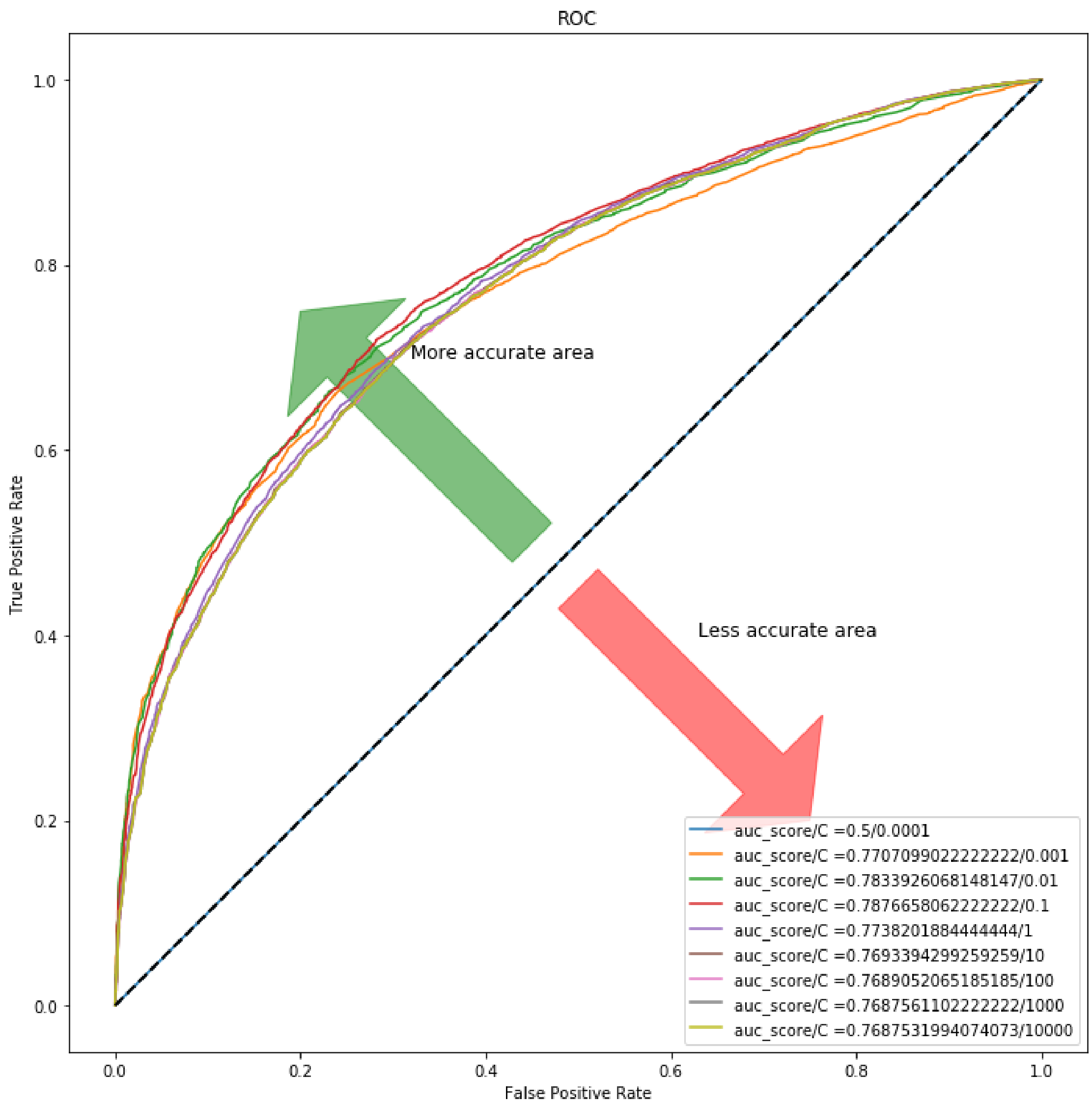
```
        if words in  w2v_words:
            vec = w2v_model.wv[words]
            sent_vec += vec
            count_words +=1
    if count_words !=0:
        sent_vec = sent_vec/count_words
    X_test_avg_w2v.append(sent_vec)
```

Importing tranformed average_W2V pickle

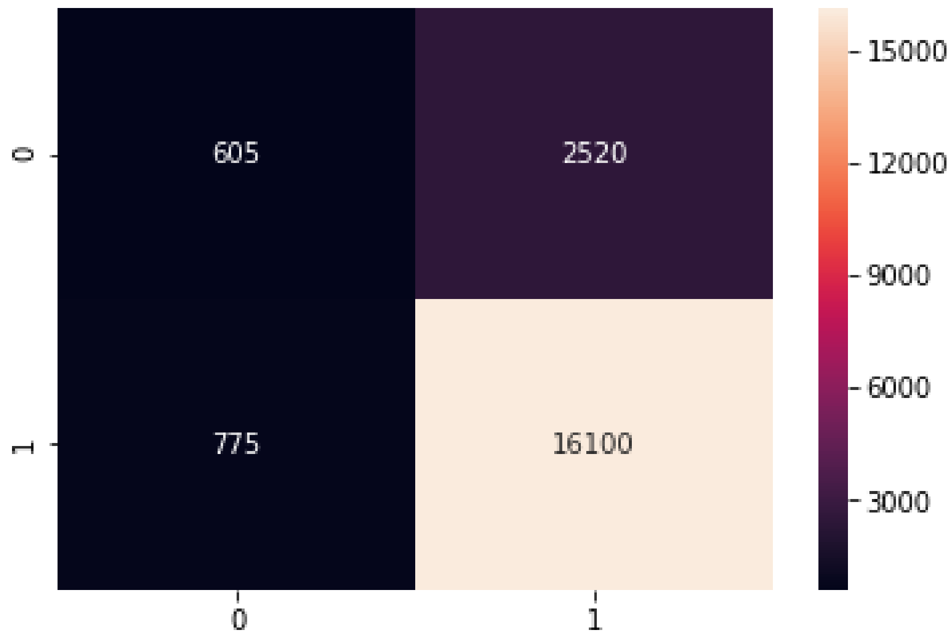
[151]:
```
y_train = y[0:100000]
y_test = y[100000:120000]
```

[152]:
```
acparam = [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10, 100, 1000, 10000]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
     patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
)
ax1.add_patch(
     patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
)
mean_fpr = np.linspace(0,1,100)
for i in acparam:
    clf = LogisticRegression(penalty='l1',n_jobs=1,class_weight='balanced',C=i)
    model = CalibratedClassifierCV(clf,cv=3,method ='isotonic')
    model.fit(X_train_avg_w2v,y_train)
    mod_probs = model.predict_proba(X_test_avg_w2v)[:,1]
    fpr, tpr, thresholds = metrics.roc_curve(y_test,  mod_probs)
    auc = metrics.roc_auc_score(y_test, mod_probs)
    tprs.append(interp(mean_fpr, fpr, tpr))
    #print("auc value for {} is {}".format(i,auc))
    plt.plot(fpr,tpr,label="auc_score/C ="+str(auc) +"/"+str(i))
    plt.legend(loc=4)
#     plt.plot(fpr,tpr,label="auc_score/alpha ="+str(auc) +"/"+str(i))
#     plt.legend(loc=4)
#     plt.show()
plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
# mean_tpr = str(np.mean(tprs, axis=0))
# mean_auc = auc(mean_fpr,mean_tpr)
#plt.plot(mean_fpr, mean_tpr, color='blue',label=r'Mean ROC (AUC = %0.2f )' %␣
 ↪(mean_auc),lw=2, alpha=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")
plt.text(0.32,0.7,'More accurate area',fontsize = 12)
```

```
plt.text(0.63,0.4,'Less accurate area',fontsize = 12)
plt.show()
```



[154]:
```
y_pred = classifier.predict(X_test_avg_w2v)
acc = f1_score(y_pred,y_test,average='micro')*100
df = pd.DataFrame(confusion_matrix(y_test,y_pred))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
print('\nThe accuracy of the logistics regression classifier for lambda = %d is␣
↪%f%%' % (10, acc))
```




```
The accuracy of the logistics regression classifier for lambda = 10 is
83.525000%
```

[157]:
```
y_pred = classifier.predict(X_test_avg_w2v)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.44      0.19      0.27      3125
           1       0.86      0.95      0.91     16875

   micro avg       0.84      0.84      0.84     20000
   macro avg       0.65      0.57      0.59     20000
weighted avg       0.80      0.84      0.81     20000
```

tfidf-word2vec

[ ]:
```
# TF-IDF weighted Word2Vec
vec = TfidfVectorizer()
vec.fit_transform(X_train[0:100000],y_train[0:100000])
tfidf_feat = vec.get_feature_names() # tfidf words/col-names
dictionary = dict(zip(vec.get_feature_names(), list(vec.idf_)))
```

```
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val =␣
↪tfidf
tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is stored in␣
↪this list
row=0;
w2v_model = Word2Vec(list_of_sent[0:100000],min_count=5,size=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in tqdm(list_of_sent[0:100000]): # for each review/sentence
    sent_vec = np.zeros(100) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            if word in tfidf_feat:
                vect = w2v_model.wv[word]
#               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vect * tf_idf)
                weight_sum += tf_idf
            else:
                break
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_train_vectors.append(sent_vec)
    row += 1
```

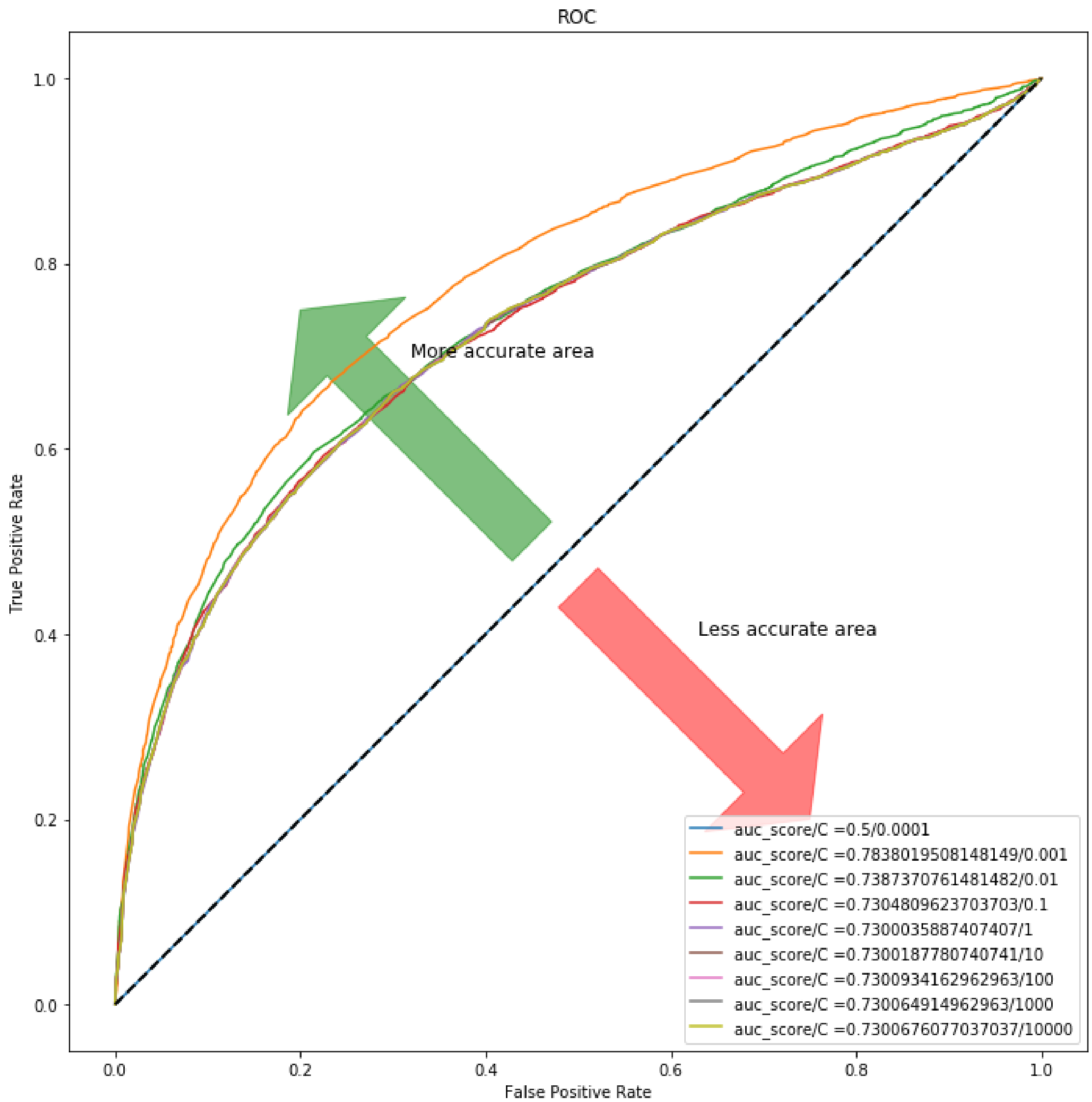
[ ]:
```
# TF-IDF weighted Word2Vec
vec = TfidfVectorizer()
vec.fit_transform(X[100000:120000],y[100000:120000])
tfidf_feat = vec.get_feature_names() # tfidf words/col-names
dictionary = dict(zip(vec.get_feature_names(), list(vec.idf_)))
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val =␣
↪tfidf
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is stored in␣
↪this list
row=0;
w2v_model = Word2Vec(list_of_sent[100000:120000],min_count=5,size=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in tqdm(list_of_sent[100000:120000]): # for each review/sentence
    sent_vec = np.zeros(100) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            if word in tfidf_feat:
                vect = w2v_model.wv[word]
```

```
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vect * tf_idf)
                weight_sum += tf_idf
            else:
                break
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_test_vectors.append(sent_vec)
    row += 1
```

[136]:
```
y_train = y[0:100000]
y_test = y[100000:120000]
```

[137]:
```
acparam = [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10, 100, 1000, 10000]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
     patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
)
ax1.add_patch(
     patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
)
mean_fpr = np.linspace(0,1,100)
for i in acparam:
    clf = LogisticRegression(penalty='l1',n_jobs=1,class_weight='balanced',C=i)
    model = CalibratedClassifierCV(clf,cv=3,method ='isotonic')
    model.fit(tfidf_train_vectors,y_train)
    mod_probs = model.predict_proba(tfidf_test_vectors)[:,1]
    fpr, tpr, thresholds = metrics.roc_curve(y_test,  mod_probs)
    auc = metrics.roc_auc_score(y_test, mod_probs)
    tprs.append(interp(mean_fpr, fpr, tpr))
    #print("auc value for {} is {}".format(i,auc))
    plt.plot(fpr,tpr,label="auc_score/C ="+str(auc) +"/"+str(i))
    plt.legend(loc=4)
#     plt.plot(fpr,tpr,label="auc_score/alpha ="+str(auc) +"/"+str(i))
#     plt.legend(loc=4)
#     plt.show()
plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
# mean_tpr = str(np.mean(tprs, axis=0))
# mean_auc = auc(mean_fpr,mean_tpr)
#plt.plot(mean_fpr, mean_tpr, color='blue',label=r'Mean ROC (AUC = %0.2f )' %␣
 ↪(mean_auc),lw=2, alpha=1)
plt.xlabel('False Positive Rate')
```

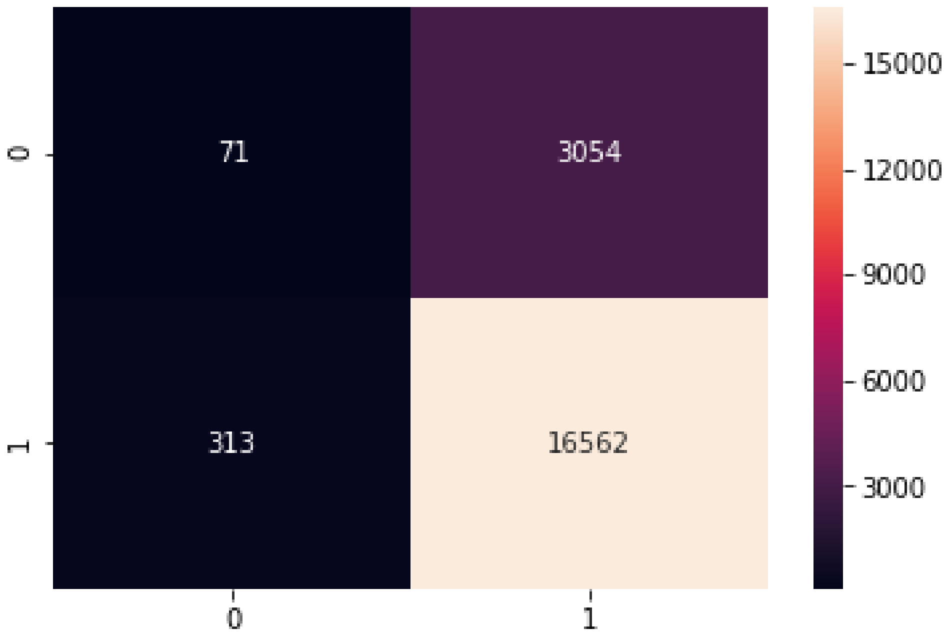
```
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")
plt.text(0.32,0.7,'More accurate area',fontsize = 12)
plt.text(0.63,0.4,'Less accurate area',fontsize = 12)
plt.show()
```



[ ]:
```
classifier = LogisticRegression(penalty='l1',n_jobs=1,C=.001)
classifier.fit(tfidf_train_vectors,y_train)
```

[145]:
```
y_pred = classifier.predict(tfidf_test_vectors)
acc = f1_score(y_pred,y[100000:120000],average='micro')*100
```

```
df = pd.DataFrame(confusion_matrix(y[100000:120000],y_pred))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
print('\nThe accuracy of the logistics regression classifier for lambda = %d is␣
↪%f%%' % (1000, acc))
```




```
The accuracy of the logistics regression classifier for lambda = 1000 is
83.165000%
```

[147]:
```
y_pred = classifier.predict(tfidf_test_vectors)
from sklearn.metrics import classification_report
print(classification_report(y[100000:120000], y_pred))
```

```
              precision    recall  f1-score   support

           0       0.18      0.02      0.04      3125
           1       0.84      0.98      0.91     16875

   micro avg       0.83      0.83      0.83     20000
   macro avg       0.51      0.50      0.47     20000
weighted avg       0.74      0.83      0.77     20000
```

[20]:

```python
from prettytable import PrettyTable
x = PrettyTable()
x.add_column("important_features_class_[0]",topn_class1)
x.add_column("important_features_class_[1]",topn_class2)
print(x)
y = PrettyTable()
y.field_names=["Vectorizer","Model","C","f1_score(micro␣
↪average)","Precision","recall"]
y.add_row(["BOW","LogisticRegression","100","0.91","0.91","0.91"])
y.add_row(["tfidf","LogisticRegression","100","0.91","0.91","0.91"])
y.add_row(["Word2Vec","LogisticRegression","10","0.84","0.84","0.84"])
y.add_row(["tfidf-Word2Vec","LogisticRegression",".001","0.83","0.83","0.83"])

print(y)
```

```
+------------------------------------------+------------------------------------
-+
|       important_features_class_[0]       |     important_features_class_[1]
|
+------------------------------------------+------------------------------------
-+
|    (-13.71481969424643, 'chlosterol')    |  (-10.545641701570243, 'breadthen')
|
|      (-13.630163662172007, 'clem')       |  (-10.072484555124836, 'delallos')
|
|   (-13.537794064582307, 'abyssinian')    |     (-9.43550725004765, 'cage')
|
|      (-13.39758381540121, 'beige')       |   (-9.341389020456583, 'buckshot')
|
|  (-13.174760918546946, 'antagonizers')   |  (-9.28492571903222, 'alteration')
|
|   (-13.131372769216608, 'bitingnuts')    |  (-9.219861194921391, 'cantalope')
|
|    (-13.108545550395194, 'creameran')    |   (-9.148721793549845, 'decedant')
|
|   (-12.726170435488719, 'caramelsbr')    |    (-9.023448579157128, 'deform')
|
|   (-12.509086336915287, 'concoction')    |   (-8.988829458437898, 'againill')
|
| (-12.482053969086861, 'consistencythey') | (-8.947648405963761, 'cleaningthe')
|
+------------------------------------------+------------------------------------
-+
+----------------+--------------------+------+-------------------------+--------
---+--------+
|   Vectorizer   |       Model        |  C   | f1_score(micro average) |
Precision | recall |
```

BOW	LogisticRegression	100	0.91	0.91	0.91
tfidf	LogisticRegression	100	0.91	0.91	0.91
Word2Vec	LogisticRegression	10	0.84	0.84	0.84
tfidf-Word2Vec	LogisticRegression	1000	0.83	0.83	0.83