SVM_amazon_review_dataset(1)(1) (1)

April 7, 2020

SVM_amazon_review _dataset

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
import sklearn
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.model_selection import train_test_split
import sqlite3
from tqdm import tqdm
import  warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings('ignore', 'Solver terminated early.*')
import string
from scipy import interp
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 confusion_matrix
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.calibration import CalibratedClassifierCV , calibration_curve
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve,auc
from sklearn.svm import SVC
import pickle
%matplotlib inline
```

[2]:
```
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

```
[3]: 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]
```

```
[4]: import nltk
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
nltk.download('stopwords')
nltk.download('wordnet')
lis = 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!
```

```
[5]: 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 :- is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.**

```
[6]: 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 lis)):
                   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
```

[7]:
```
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')
```

[8]:
```
X = data_without_dup['final_string']
y = data_without_dup['Score']
```

[9]:
```
X_train= X[0:250000]
y_train = y[0:250000]
X_test= X[250000:280000]
```

```
y_test = y[250000:280000]
```

BOW as vectorizer with Standardscaler

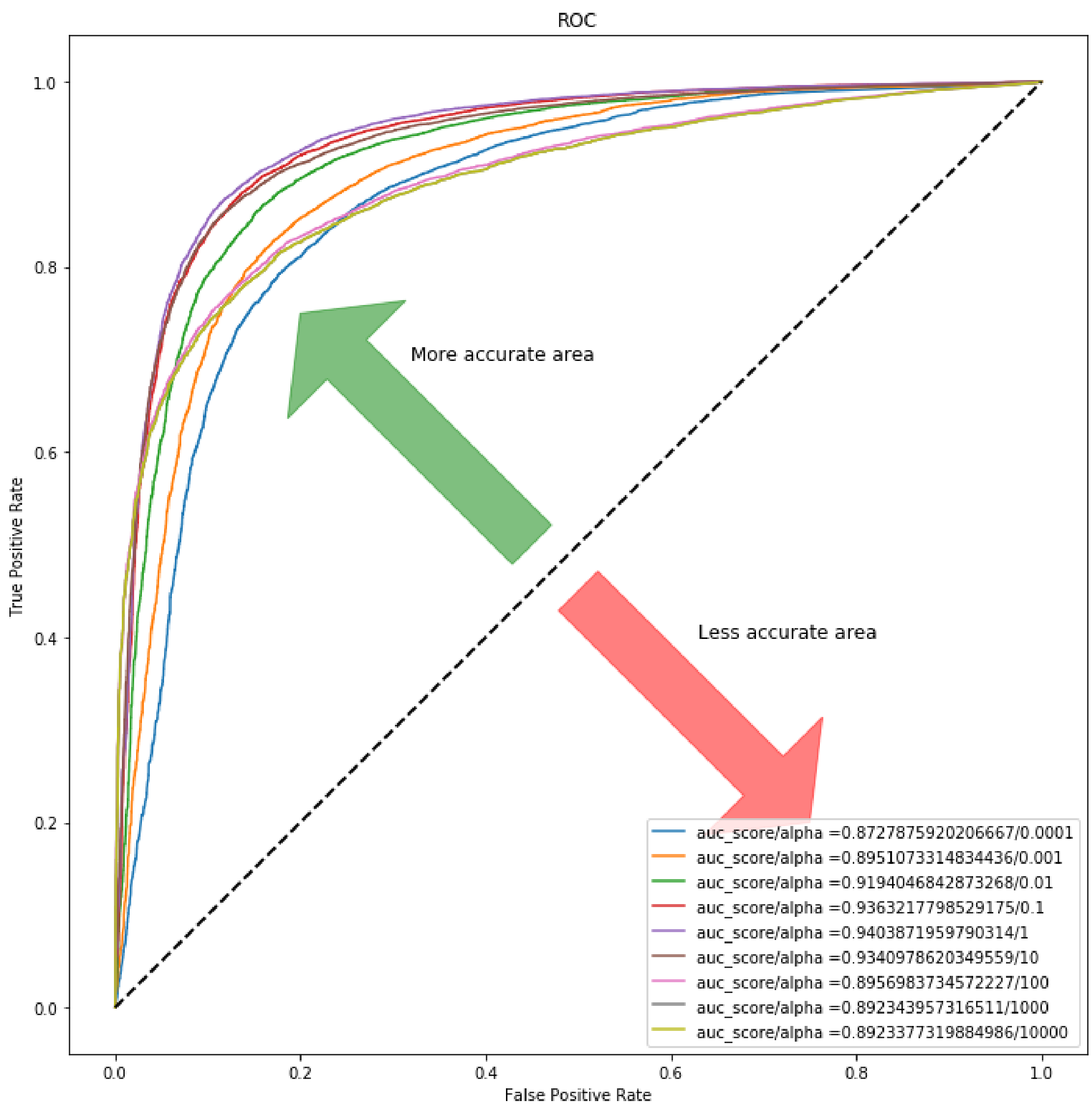
[17]:
```
count_vect = CountVectorizer()
X_train_bow = count_vect.fit_transform(X_train)
X_test_bow = count_vect.transform(X_test)
count_vec = StandardScaler(with_mean=False)
X_train_bow = count_vec.fit_transform(X_train_bow)
X_test_bow = count_vec.transform(X_test_bow)
```

```
/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)
```

Linear SVM

[28]:
```
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)
    )

tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
for i in acparam:
    classifier =␣
 ↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=i,class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,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))
```

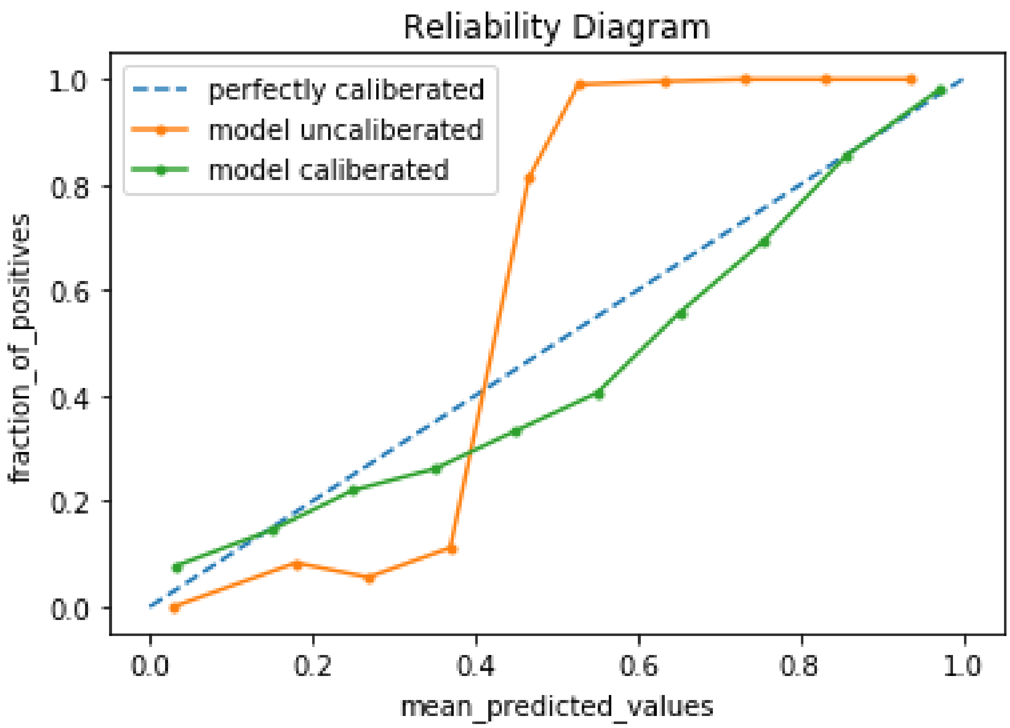
```
    #print("auc value for {} is {}".format(i,auc))
    plt.plot(fpr,tpr,label="auc_score/alpha ="+str(auc) +"/"+str(i))
    plt.legend(loc=4)
#     plt.plot(fpr,tpr,label="auc_score/alpha ="+str(auc) +"/"+str(i))
#     plt.legend(loc=4)
#     plt.show()
plt.plot([0,1],[0,1],linestyle = '--',lw = 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()
```



Reliability Diagram

[18]:
```
classifier =␣
↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=1,class_weight='balanced')
classifier.fit(X_train_bow,y_train)
coef = classifier.coef_
probs = classifier.decision_function(X_test_bow)
model = CalibratedClassifierCV(classifier,cv=5,method ='sigmoid')
model.fit(X_train_bow,y_train)
mod_probs = model.predict_proba(X_test_bow)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
```

```
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```



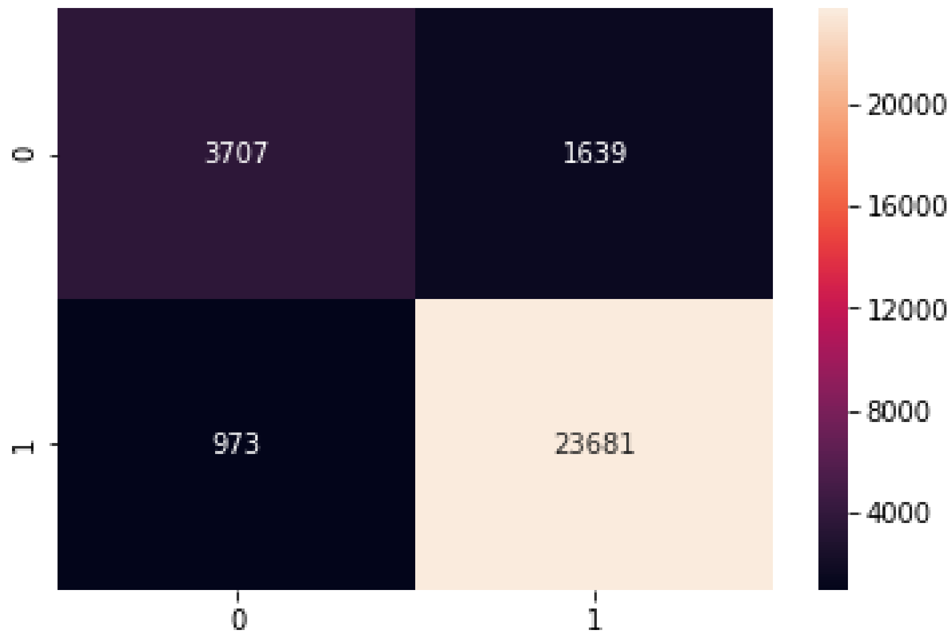
[20]:
```
class_labels = model.classes_
feature_names =count_vect.get_feature_names()
topn_class1 = sorted(zip(coef, feature_names),reverse=False)[:10]
topn_class2 = sorted(zip(coef, feature_names),reverse=True)[:10]
print("Important words in negative reviews")
for coef, feat in topn_class1:
    print(class_labels[0], coef, feat)
print("-----------------------------------------")
print("Important words in positive reviews")
for coef, feat in topn_class2:
    print(class_labels[1], coef, feat)
```

```
Important words in negative reviews
```

```
0 -0.0623597689116973 disappointed
0 -0.04407459856468149 worst
0 -0.04079761544860912 terrible
0 -0.0394333247551307 disappointing
0 -0.03836276939405171 bad
0 -0.03827092091242986 awful
0 -0.037717573995211466 horrible
0 -0.03619549020627634 money
0 -0.03573347397507956 thought
0 -0.034798940956997225 unfortunately
-----------------------------------------
Important words in positive reviews
1 0.0986137913348786 great
1 0.07869037563397005 love
1 0.06580537488193834 best
1 0.054316747186839306 delicious
1 0.05114921058067525 good
1 0.04331899000896731 excellent
1 0.042684723906926084 perfect
1 0.04194137537362303 favorite
1 0.03792207264371975 wonderful
1 0.0377737501305404 highly
```

[27]:

```
y_pred = model.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 linear SVM classifier for alpha = %d is %f%%' % (1,␣
 ↪acc))
```




```
The accuracy of the linear SVM classifier for alpha = 1 is 91.293333%
```

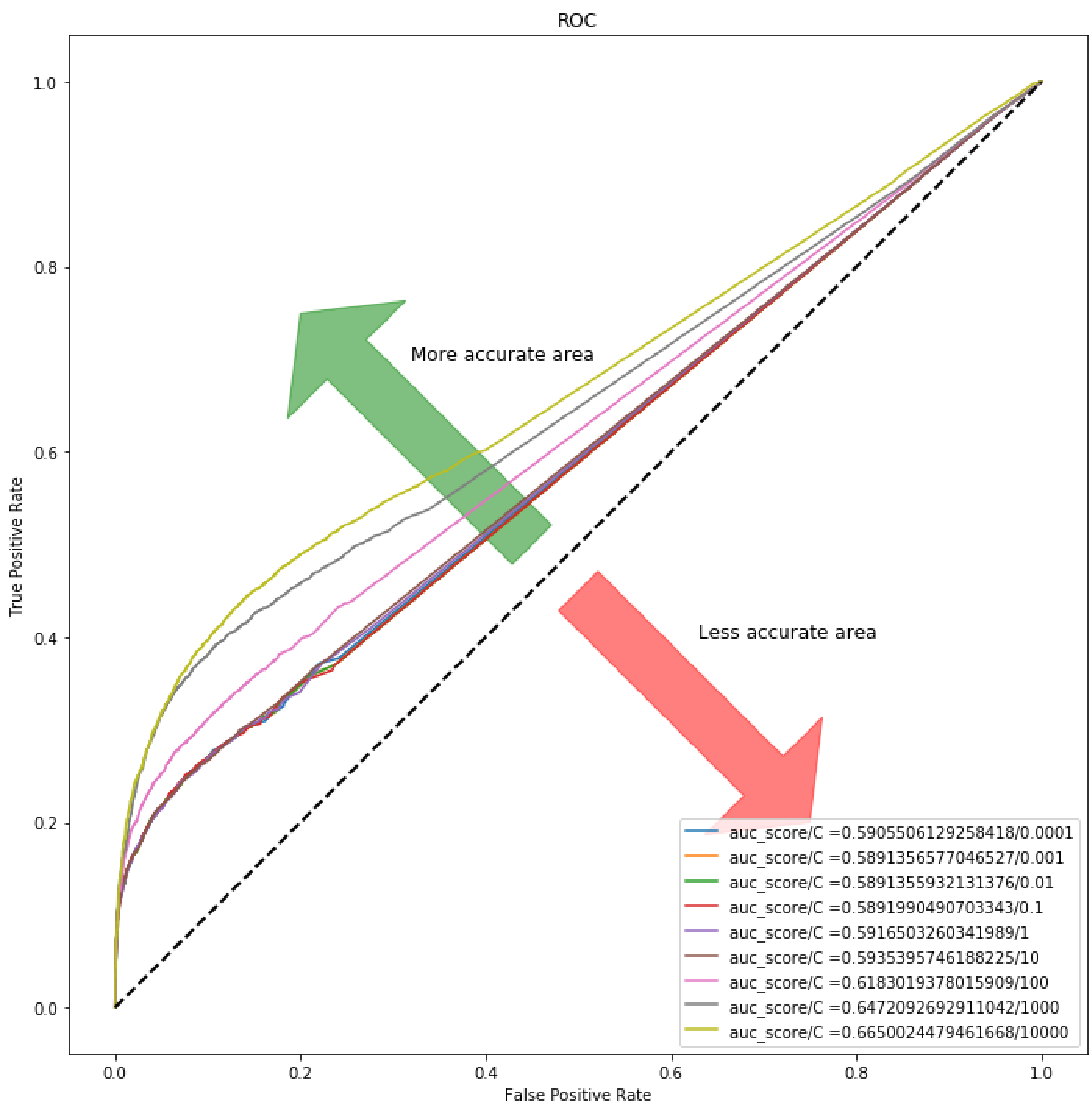
[28]:
```
y_pred = model.predict(X_test_bow)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.69      0.74      5346
           1       0.94      0.96      0.95     24654

   micro avg       0.91      0.91      0.91     30000
   macro avg       0.86      0.83      0.84     30000
weighted avg       0.91      0.91      0.91     30000
```

RBF Kernel SVM

[49]:
```
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(
```

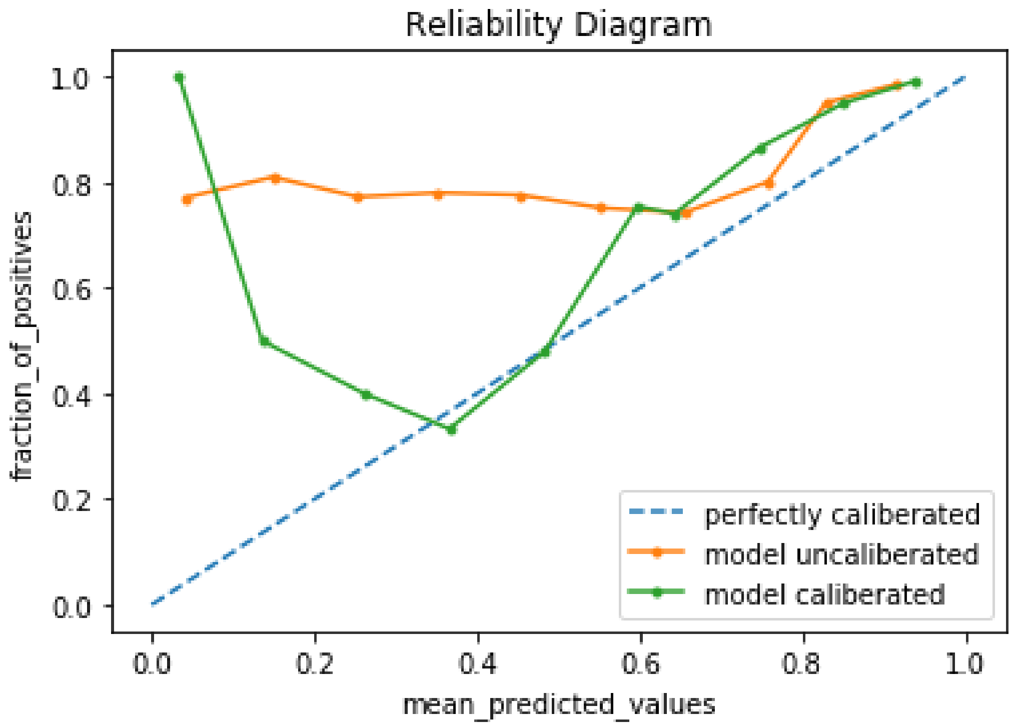
```
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
i = 1
for i in acparam:
    classifier =␣
 ↪SVC(max_iter=1000,tol=1e-3,C=i,kernel='rbf',class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,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()
```



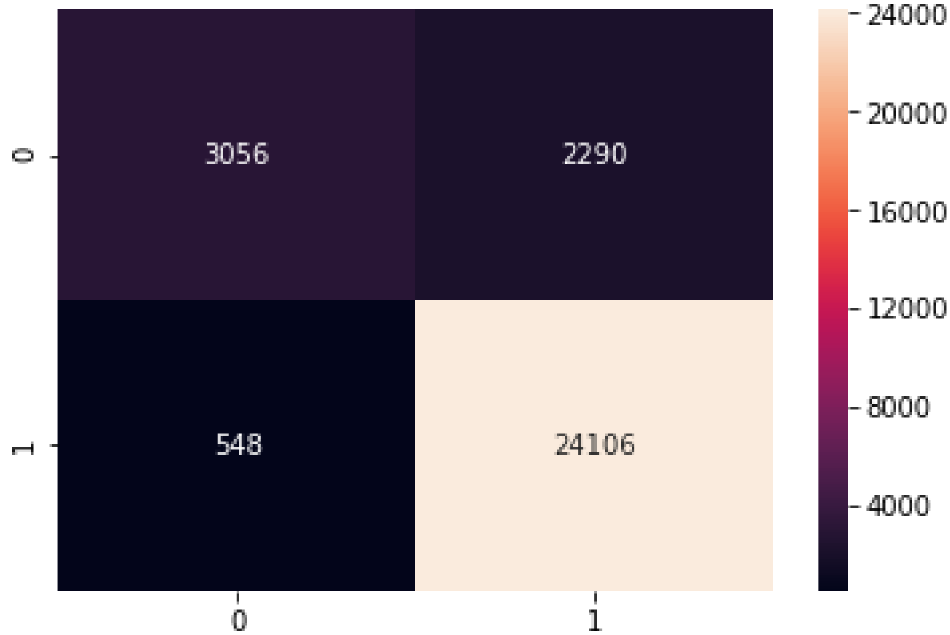
[11]:
```
classifier =␣
↪SVC(max_iter=1000,tol=1e-3,C=10000,kernel='rbf',class_weight='balanced')
classifier.fit(X_train_bow,y_train)
probs = classifier.decision_function(X_test_bow)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_bow,y_train)
mod_probs = model.predict_proba(X_test_bow)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
```

```
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```

```
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```



[44]:
```
y_pred = model.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 linear SVM classifier for C = %d is %f%%' % (10000,␣
 ↪acc))
```




```
The accuracy of the linear SVM classifier for C = 10000 is 90.540000%
```

[51]:
```python
y_pred = model.predict(X_test_bow)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

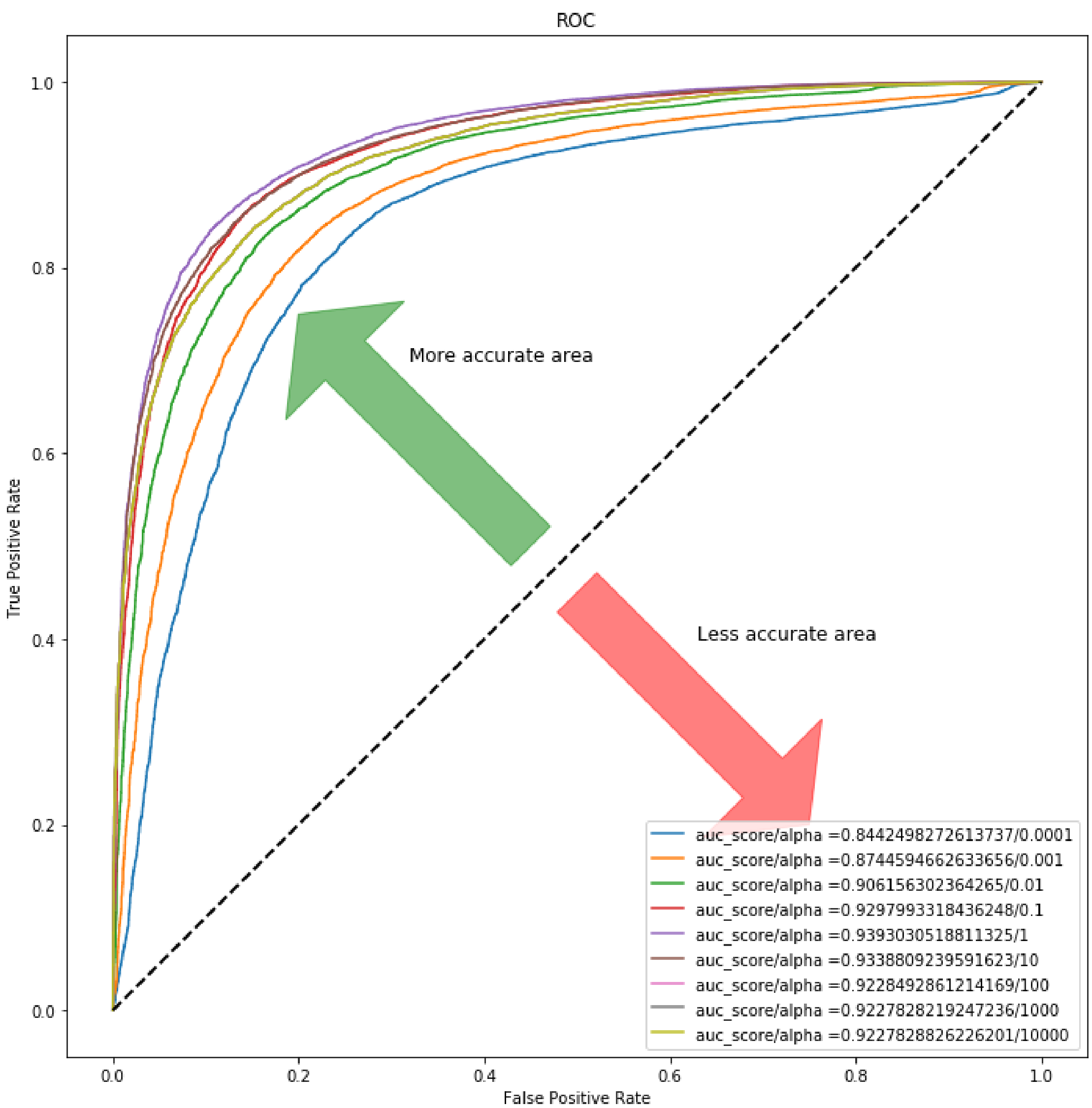
```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      5346
           1       0.82      1.00      0.90     24654

   micro avg       0.82      0.82      0.82     30000
   macro avg       0.41      0.50      0.45     30000
weighted avg       0.68      0.82      0.74     30000
```

tfidf as vectorizer with Standardscaler

[21]:
```python
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)
```

[13]:

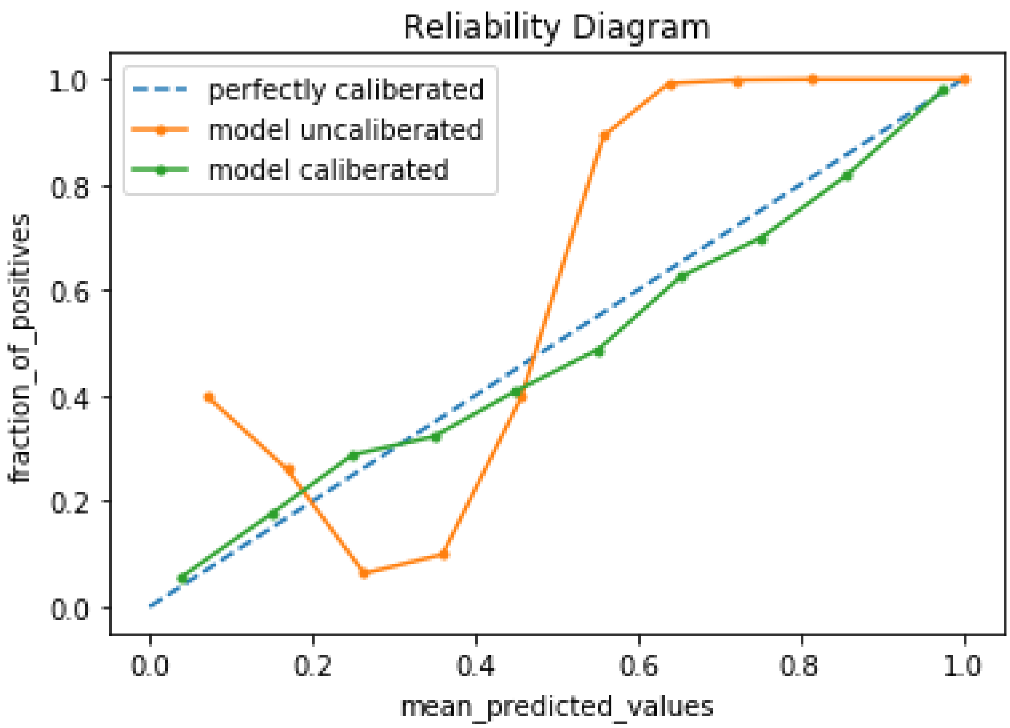
```python
acparam = [10**-4, 10**-3, 10**-2, 10**-1, 1 , 10, 100, 1000, 10000]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
i = 1
for i in acparam:
    classifier =␣
 ↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=i,class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,method ='sigmoid')
    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)
    tprs.append(interp(mean_fpr, fpr, tpr))
    #print("auc value for {} is {}".format(i,auc))
    plt.plot(fpr,tpr,label="auc_score/alpha ="+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()
```



Linear SVM

[23]:
```
classifier =␣
↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=1,class_weight='balanced')
classifier.fit(X_train_tfidf,y_train)
coef = classifier.coef_
probs = classifier.decision_function(X_test_tfidf)
model = CalibratedClassifierCV(classifier,cv=5,method ='sigmoid')
model.fit(X_train_tfidf,y_train)
mod_probs = model.predict_proba(X_test_tfidf)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
```

```
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```

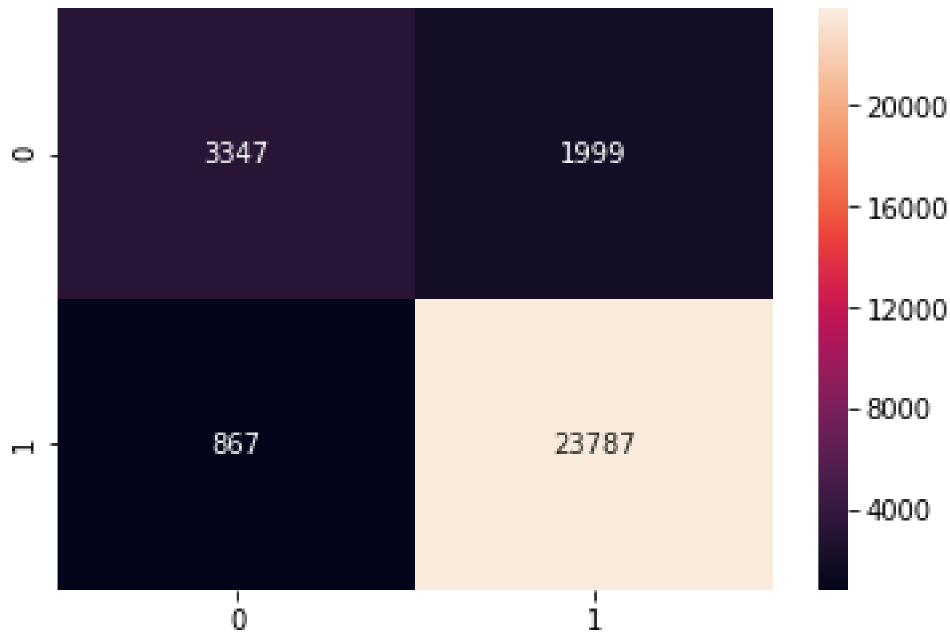


[25]:
```
class_labels = model.classes_
feature_names =tfidf_vec.get_feature_names()
topn_class1 = sorted(zip(coef, feature_names),reverse=False)[:10]
topn_class2 = sorted(zip(coef, feature_names),reverse=True)[:10]
print("Important words in negative reviews")
for coef, feat in topn_class1:
    print(class_labels[0], coef, feat)
print("-----------------------------------------")
print("Important words in positive reviews")
for coef, feat in topn_class2:
    print(class_labels[1], coef, feat)
```

```
Important words in negative reviews
```

```
0 -0.055012375312047807 disappointed
0 -0.04058444275252361 worst
0 -0.036902078762866795 terrible
0 -0.0349645694722509 disappointing
0 -0.03496417187139286 bad
0 -0.03451755640304197 horrible
0 -0.03377887606017093 awful
0 -0.032561847151554496 thought
0 -0.032526395176782574 unfortunately
0 -0.030602582371442463 didnt
-----------------------------------------
Important words in positive reviews
1 0.07830608173821682 great
1 0.06439161451581411 love
1 0.05622844115490847 best
1 0.04787791021102894 good
1 0.04449678399514948 delicious
1 0.03678310801238328 perfect
1 0.03552206094557603 excellent
1 0.03500014156903399 favorite
1 0.03195405346585583 nice
1 0.031552236311105986 wonderful
```

[43]:
```
y_pred = model.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 linear SVM classifier for alpha = %d is %f%%' % (1,␣
 ↪acc))
```




```
The accuracy of the linear SVM classifier for alpha = 1 is 90.446667%
```

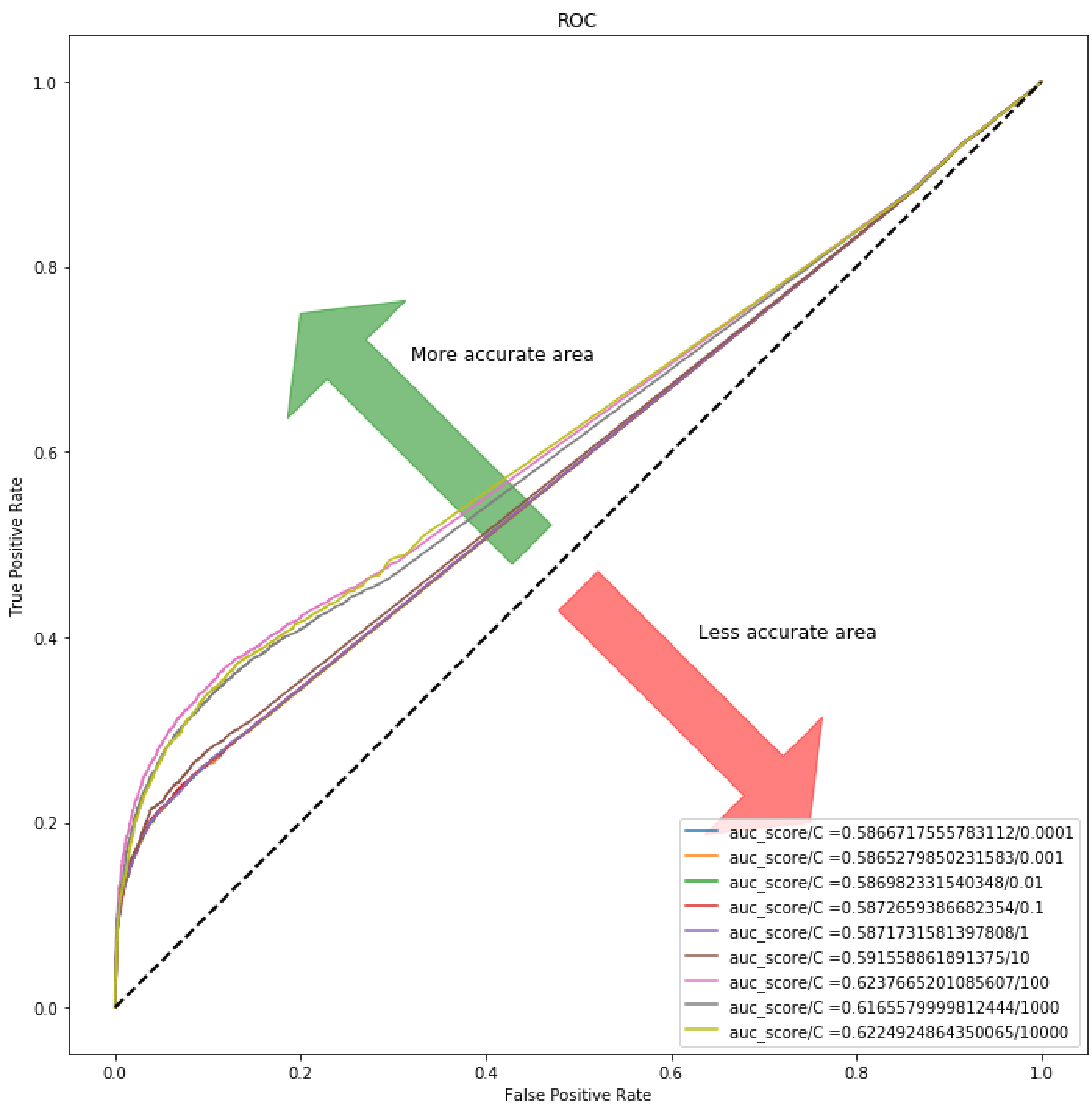
[57]:
```
y_pred = model.predict(X_test_tfidf)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.65      0.71      5346
           1       0.93      0.96      0.94     24654

   micro avg       0.90      0.90      0.90     30000
   macro avg       0.85      0.81      0.83     30000
weighted avg       0.90      0.90      0.90     30000
```

RBF Kernel SVM

[52]:
```
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)
i = 1
for i in acparam:
    classifier =␣
 ↪SVC(max_iter=1000,tol=1e-3,C=i,kernel='rbf',class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,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)
    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()
```

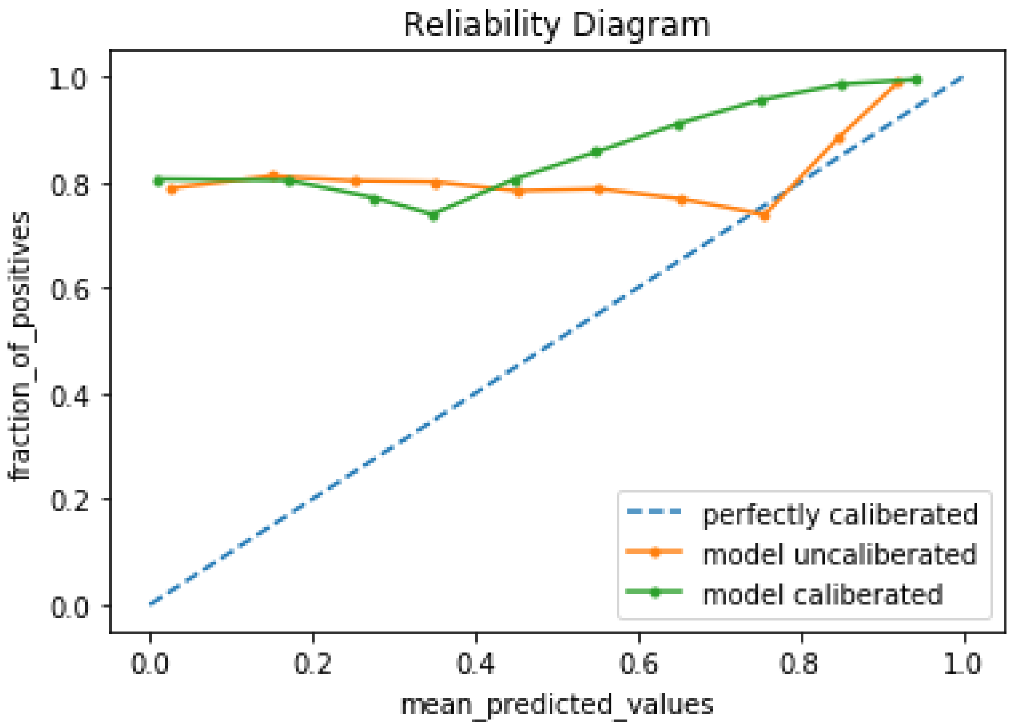


[14]:

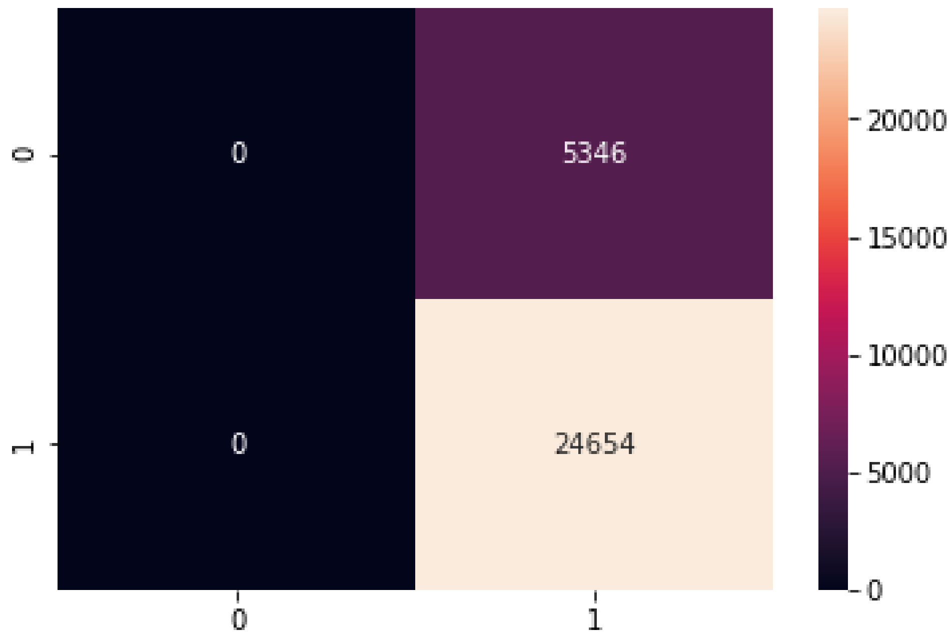
```
classifier =␣
↪SVC(max_iter=1000,tol=1e-3,C=100,kernel='rbf',class_weight='balanced')
classifier.fit(X_train_tfidf,y_train)
probs = classifier.decision_function(X_test_tfidf)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_tfidf,y_train)
mod_probs = model.predict_proba(X_test_tfidf)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
```

```python
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```

```
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```



[54]:
```
y_pred = model.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 linear SVM classifier for C = %d is %f%%' % (100,␣
 ↪acc))
```




```
The accuracy of the linear SVM classifier for C = 100 is 82.180000%
```

[55]:
```
y_pred = model.predict(X_test_tfidf)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      5346
           1       0.82      1.00      0.90     24654

   micro avg       0.82      0.82      0.82     30000
   macro avg       0.41      0.50      0.45     30000
weighted avg       0.68      0.82      0.74     30000
```

Word2Vec as vectorizer tranformation procedure

[11]:
```
# list_of_sent = []

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

[12]:
```
# from gensim.models import Word2Vec
# from gensim.models import KeyedVectors
```

```
[ ]: # mod = KeyedVectors.load_word2vec_format("/home/niranjan/Downloads/
     ↪GoogleNews-vectors-negative300.bin", binary=True)
```

```
[ ]: # 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)
```

```
[ ]: # 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)
```

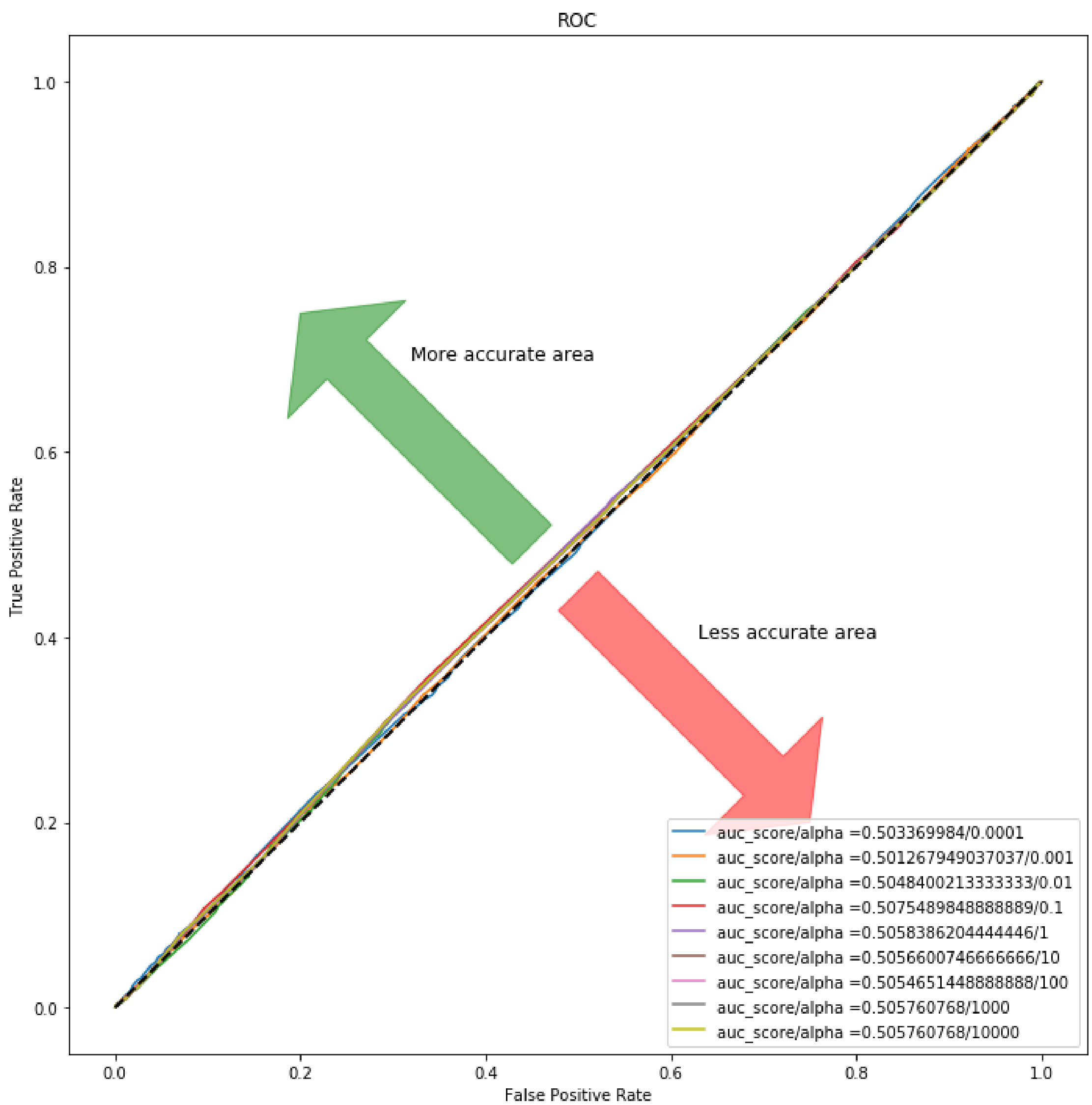
importing Average-w2v tranformed datasets as pickle objects

```
[15]: import pickle
      pickle_out1 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
      ↪X_train_avg_w2v","rb")
      pickle_out2 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
      ↪X_cv_avg_w2v","rb")
      X_train_avg_w2v = pickle.load(pickle_out1)
      X_test_avg_w2v = pickle.load(pickle_out2)
      pickle_out1.close()
      pickle_out2.close()
```

[16]:
```
y_train = y[0:100000]
y_test= y[100000:120000]
```

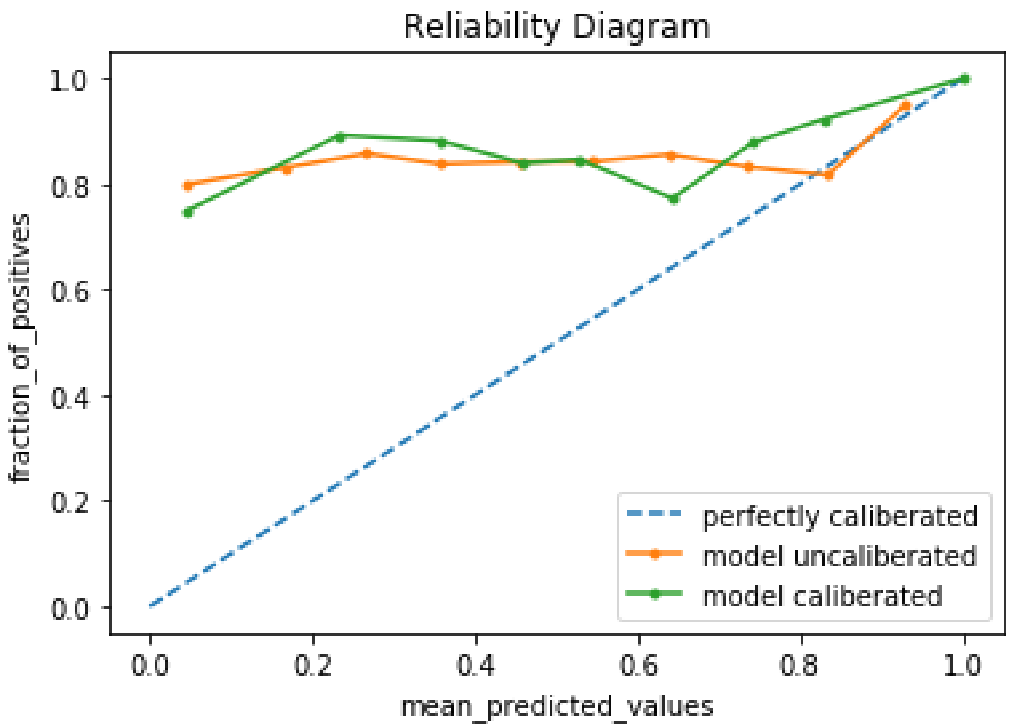
Linear SVM

[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)
    )

tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
i = 1
for i in acparam:
    classifier =␣
 ↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=i,class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,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/alpha ="+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()
```

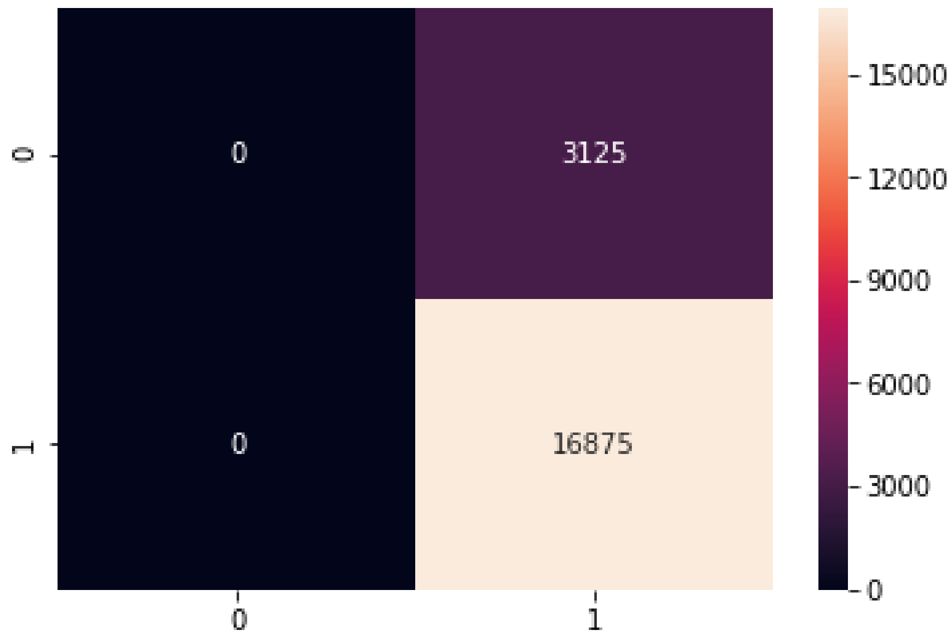


[17]:
```
classifier =␣
 ↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=1,class_weight='balanced')
classifier.fit(X_train_avg_w2v,y_train)
probs = classifier.decision_function(X_test_avg_w2v)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_avg_w2v,y_train)
mod_probs = model.predict_proba(X_test_avg_w2v)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
```

```
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```



[63]:
```
y_pred = model.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 linear SVM classifier for alpha = %d is %f%%' % (1,␣
↪acc))
```




```
The accuracy of the linear SVM classifier for alpha = 1 is 84.375000%
```

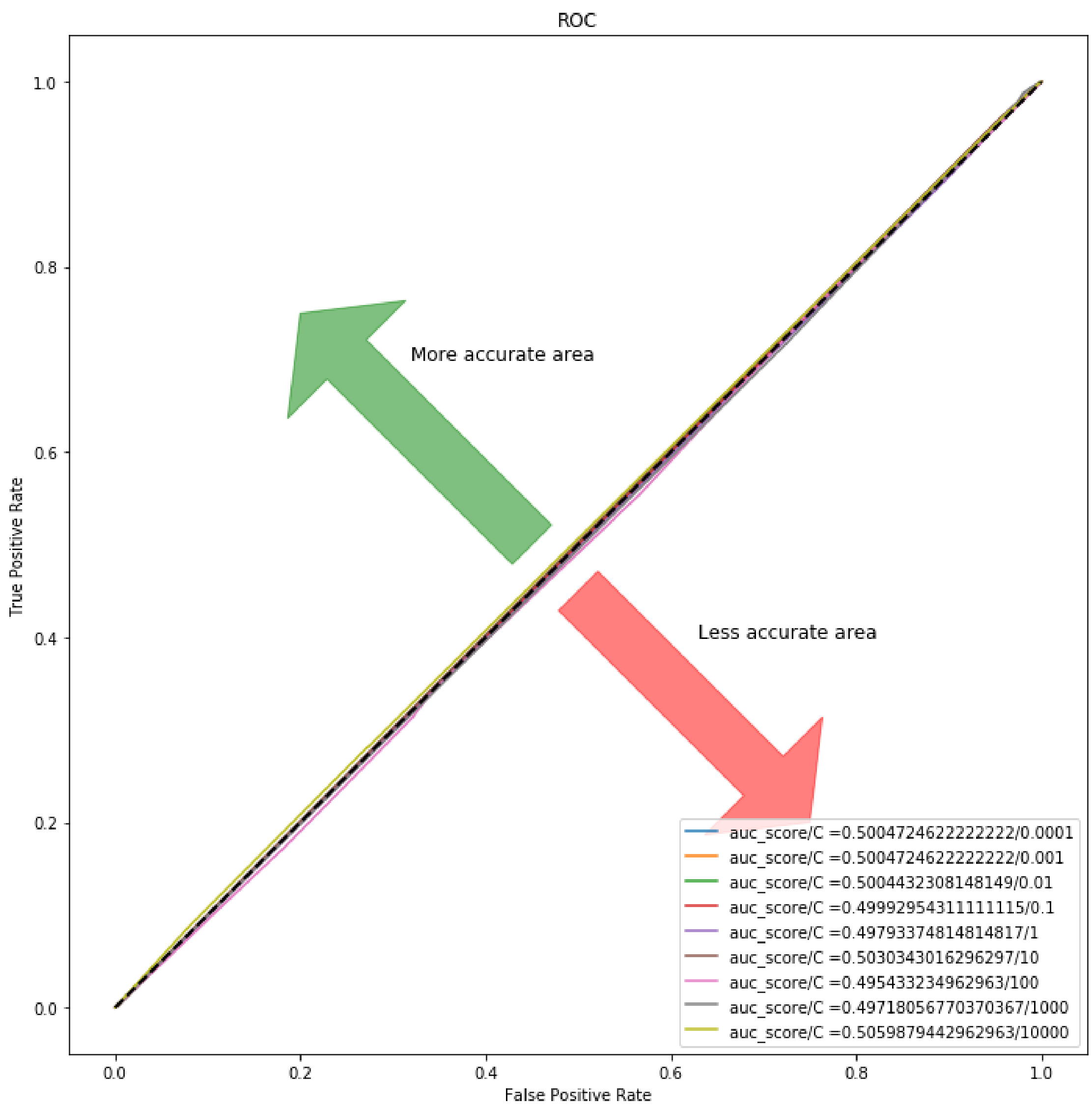
[64]:
```
y_pred = model.predict(X_test_avg_w2v)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3125
           1       0.84      1.00      0.92     16875

   micro avg       0.84      0.84      0.84     20000
   macro avg       0.42      0.50      0.46     20000
weighted avg       0.71      0.84      0.77     20000
```

RBF Kernel SVM

[65]:
```
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)
    )

tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
i = 1
for i in acparam:
    classifier =␣
↪SVC(max_iter=1000,tol=1e-3,C=i,kernel='rbf',class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,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()
```

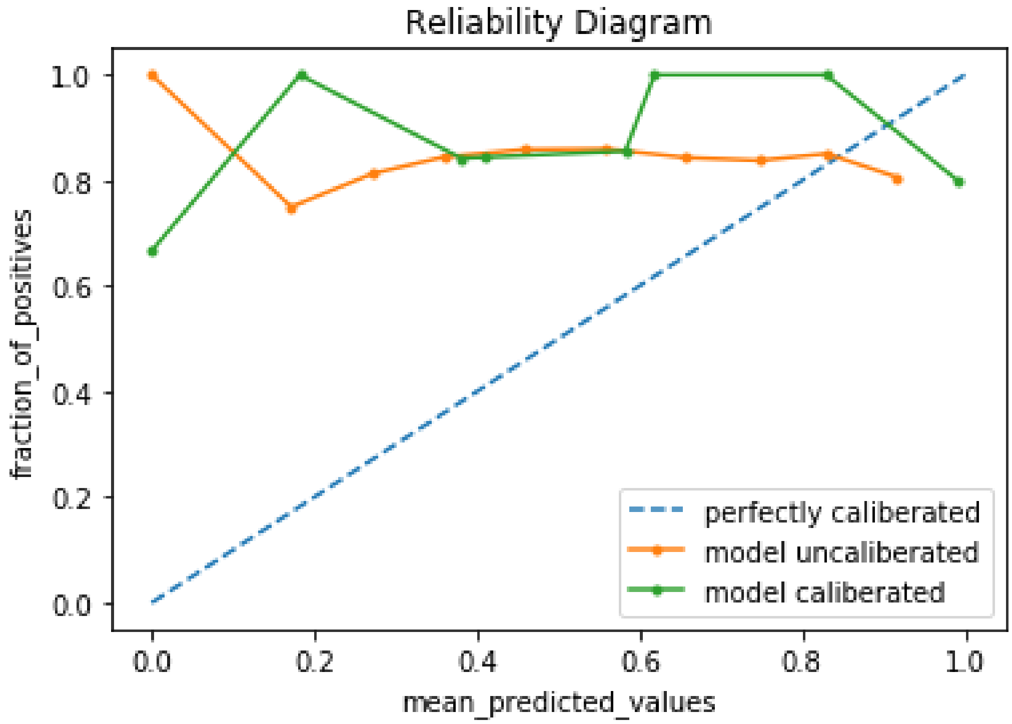


Reliability Diagram : Observed frequency of an event plotted against the Forecast probability of an event.

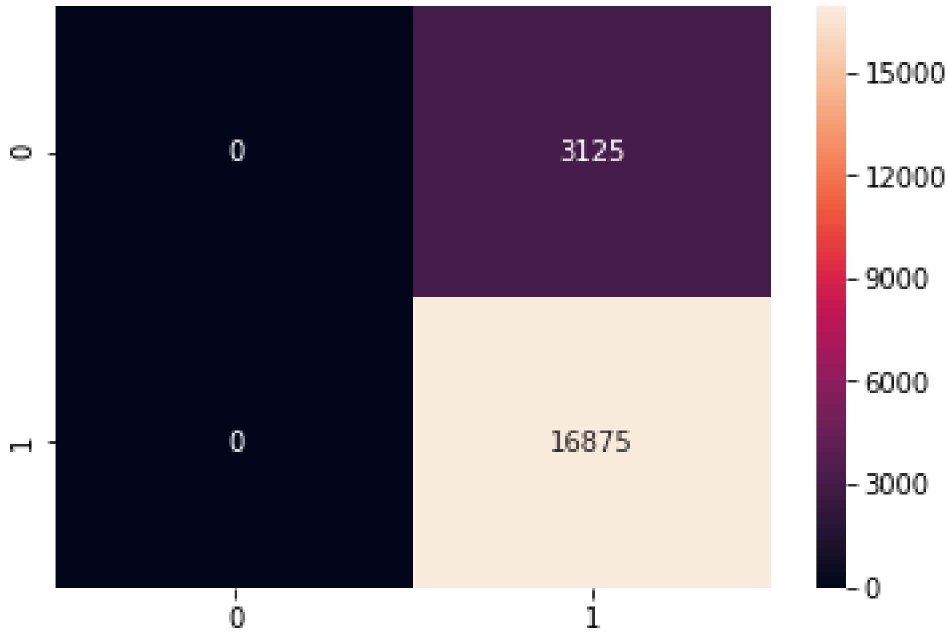
[18]:
```
classifier =␣
  ↪SVC(max_iter=1000,tol=1e-3,C=10000,kernel='rbf',class_weight='balanced')
classifier.fit(X_train_avg_w2v,y_train)
probs = classifier.decision_function(X_test_avg_w2v)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_avg_w2v,y_train)
mod_probs = model.predict_proba(X_test_avg_w2v)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
```

```
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```

```
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
/home/niranjan/anaconda3/lib/python3.6/site-packages/sklearn/svm/base.py:244:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```



[67]:
```
y_pred = model.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 linear SVM classifier for alpha = %d is %f%%' %␣
↪(10000, acc))
```



The accuracy of the linear SVM classifier for alpha = 10000 is 84.375000%

[68]:
```
y_pred = model.predict(X_test_avg_w2v)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3125
           1       0.84      1.00      0.92     16875

   micro avg       0.84      0.84      0.84     20000
   macro avg       0.42      0.50      0.46     20000
weighted avg       0.71      0.84      0.77     20000
```

tfidf-Word2Vec transformation procedure

[ ]:
```
# # 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
```

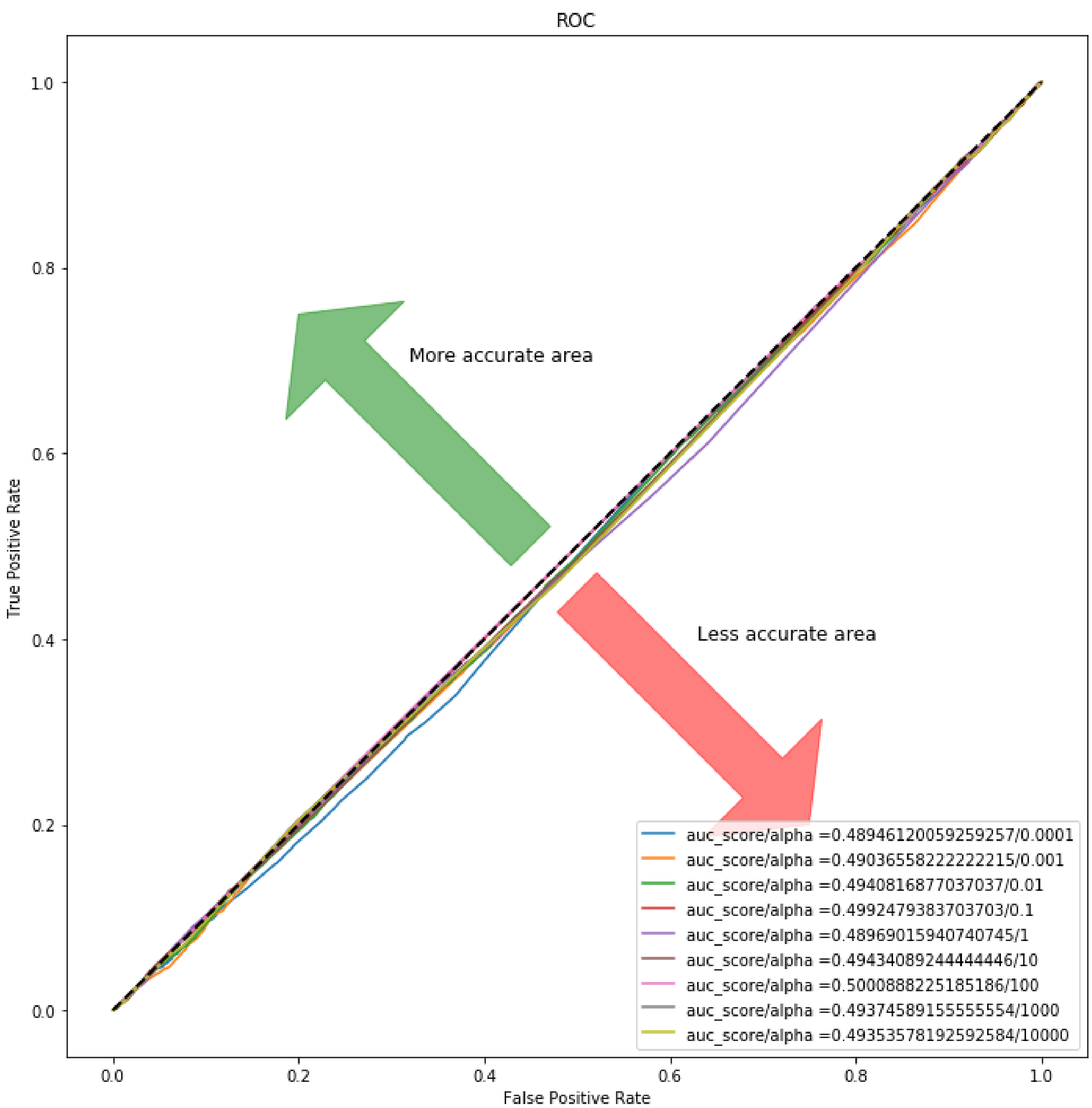
Importing tfidf-w2v processed datasets as pickle object

[19]:
```
import pickle
pickle_out1 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
 ↪tfidf_train_vectors","rb")
pickle_out2 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
 ↪tfidf_test_vectors","rb")
tfidf_train_vectors = pickle.load(pickle_out1)
tfidf_test_vectors = pickle.load(pickle_out2)
pickle_out1.close()
pickle_out2.close()
```

LinearSVM with tfidf-w2v as vectorizer

[24]:
```
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)
    )

tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
i = 1
for i in acparam:
    classifier =␣
 ↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=i,class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
    model.fit(tfidf_train_vectors,y_train)
    mod_probs = model.predict_proba(tfidf_test_vectors)[:,1]
```

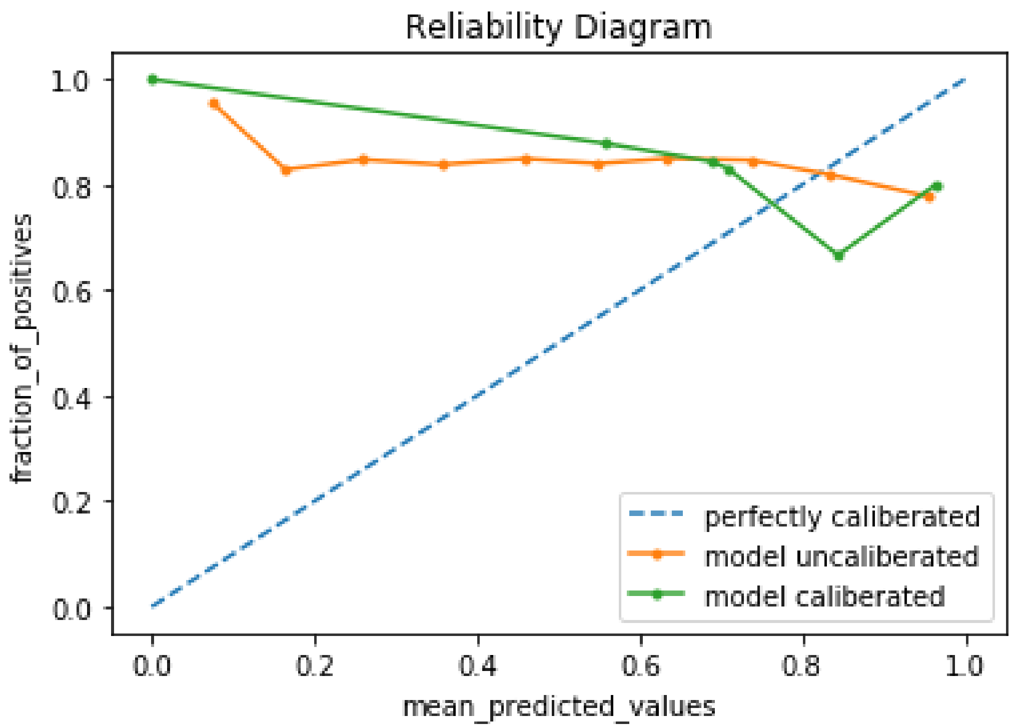
```
    fpr, tpr, thresholds = metrics.roc_curve(y_test,  mod_probs)
    auc = metrics.roc_auc_score(y_test, mod_probs)
    tprs.append(interp(mean_fpr, fpr, tpr))
    #print("auc value for {} is {}".format(i,auc))
    plt.plot(fpr,tpr,label="auc_score/alpha ="+str(auc) +"/"+str(i))
    plt.legend(loc=4)
#     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()
```



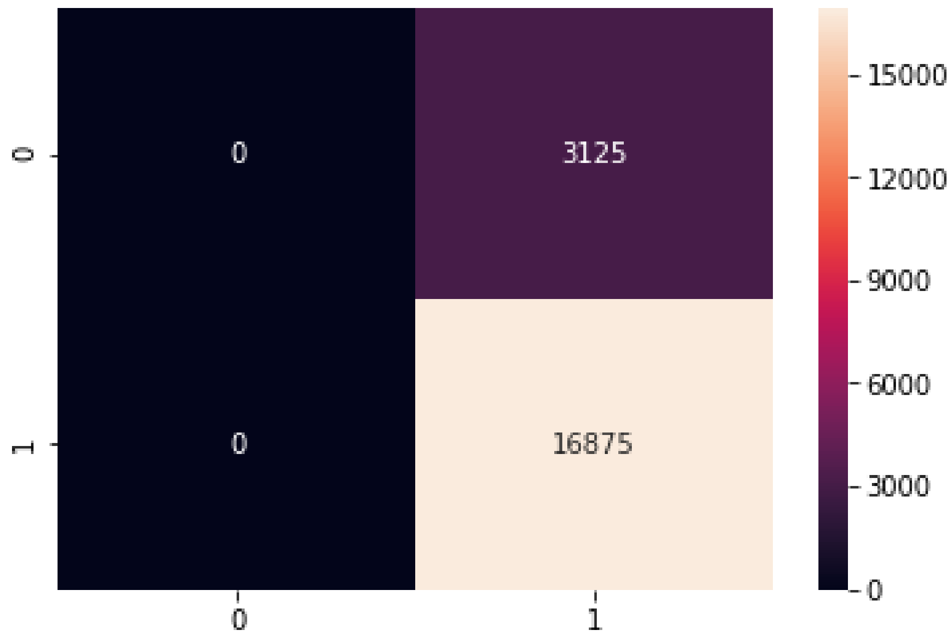
Reliability Diagram : Observed frequency of an event plotted against the Forecast probability of an event.

[70]:
```
classifier =␣
↪SGDClassifier(max_iter=1000,tol=1e-3,alpha=100,class_weight='balanced')
classifier.fit(tfidf_train_vectors,y_train)
probs = classifier.decision_function(tfidf_test_vectors)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(tfidf_train_vectors,y_train)
mod_probs = model.predict_proba(tfidf_test_vectors)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
```

```
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```



[71]:
```
y_pred = model.predict(tfidf_test_vectors)
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 linear SVM classifier for alpha = %s is %f%%' %␣
↪("100", acc))
```




```
The accuracy of the linear SVM classifier for alpha = 100 is 84.375000%
```

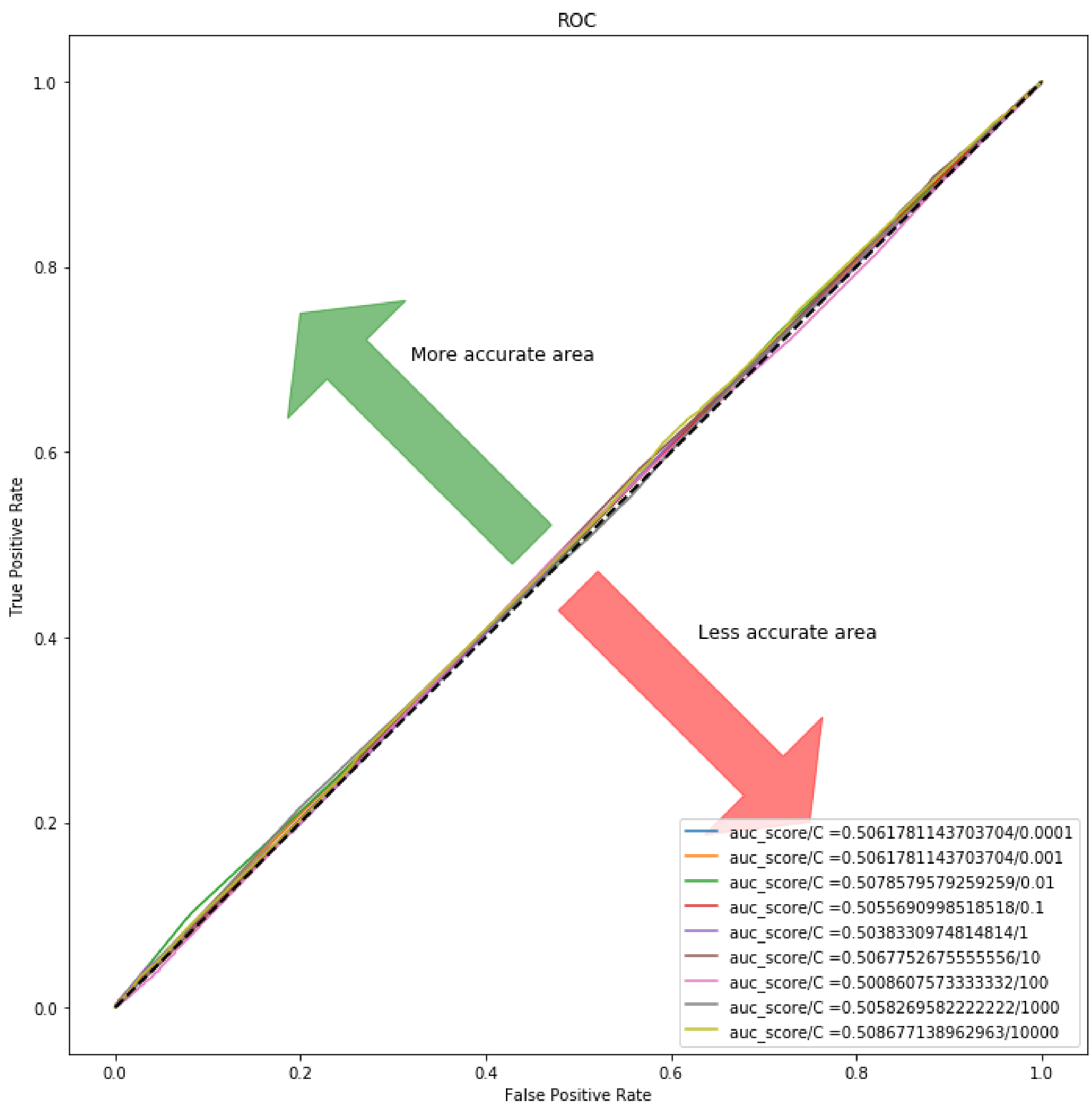
[72]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3125
           1       0.84      1.00      0.92     16875

   micro avg       0.84      0.84      0.84     20000
   macro avg       0.42      0.50      0.46     20000
weighted avg       0.71      0.84      0.77     20000
```

RBF Kernel SVM

[73]:
```python
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)
```

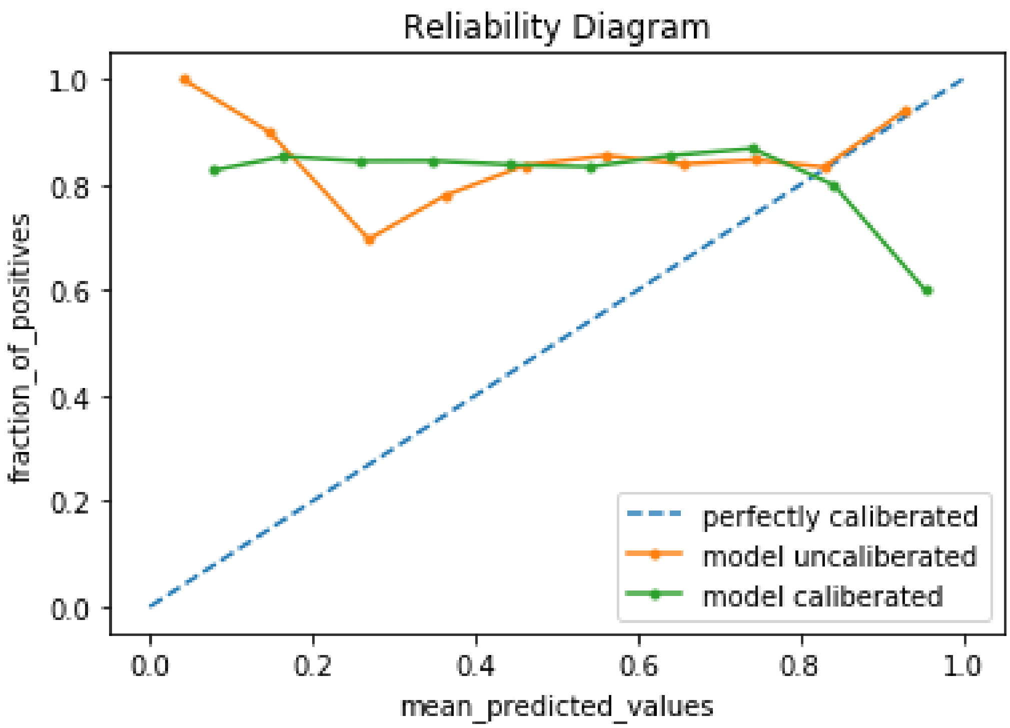
```
    )

tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
i = 1
for i in acparam:
    classifier =␣
 ↪SVC(max_iter=1000,tol=1e-3,C=i,kernel='rbf',class_weight='balanced')
    model = CalibratedClassifierCV(classifier,cv=5,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()
```



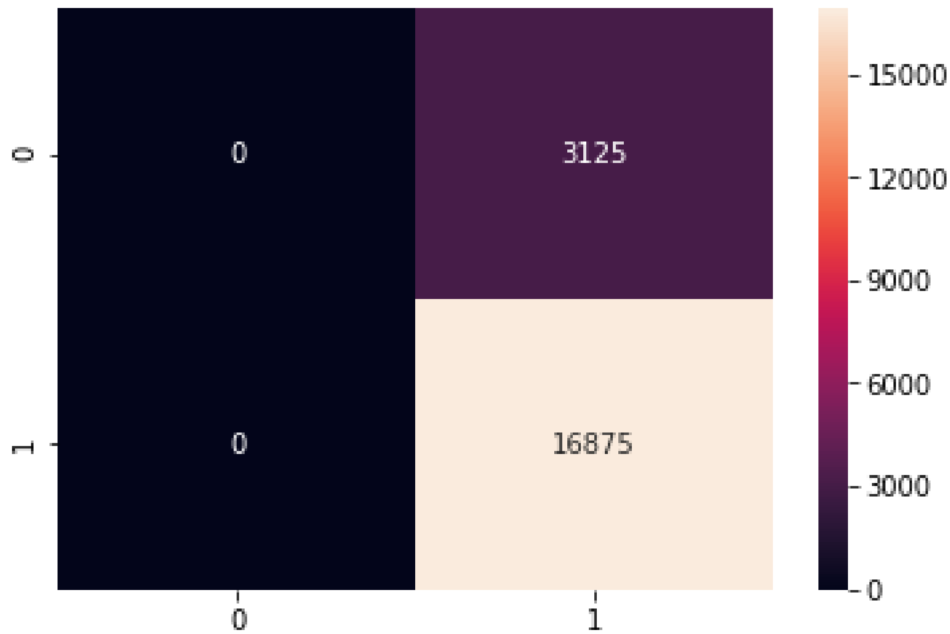
Reliability Diagram : Observed frequency of an event plotted against the Forecast probability of an event.

[77]:
```
classifier =␣
↪SVC(max_iter=1000,tol=1e-3,C=10000,kernel='rbf',class_weight='balanced')
classifier.fit(tfidf_train_vectors,y_train)
probs = classifier.decision_function(tfidf_test_vectors)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(tfidf_train_vectors,y_train)
mod_probs = model.predict_proba(tfidf_test_vectors)[:,1]
#reliability diagram
fop, mpv = calibration_curve(y_test, probs, n_bins=10,normalize=True)
fop1, mpv1 = calibration_curve(y_test, mod_probs, n_bins=10,normalize=True)
```

```
# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--',label='perfectly caliberated')
# plot model reliability
plt.plot(mpv, fop, marker='.',label='model uncaliberated')
plt.plot(mpv1, fop1, marker='.',label='model caliberated')
plt.title("Reliability Diagram")
plt.xlabel("mean_predicted_values")
plt.ylabel("fraction_of_positives")
plt.legend()
plt.show()
```



[75]:
```
y_pred = model.predict(tfidf_test_vectors)
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 linear SVM classifier for C = %s is %f%%' %␣
↪("10000", acc))
```




```
The accuracy of the linear SVM classifier for C = 10000 is 84.375000%
```

[76]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3125
           1       0.84      1.00      0.92     16875

   micro avg       0.84      0.84      0.84     20000
   macro avg       0.42      0.50      0.46     20000
weighted avg       0.71      0.84      0.77     20000
```

[26]:
```
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","Kernel","alpha","C","f1_score","Precision","recall","Auc
y.add_row(["BOW","SGDClassifier","Linear","1"," ","0.91","0.91",".91",".9405"])
y.add_row(["BOW","SVC","RBF"," ","10000","0.82","0.82",".82",".6650"])
```

```
y.add_row(["tfidf","SGDClassifier","Linear","1"," ","0.91","0.91","0.91",".
↪9395"])
y.add_row(["tfidf","SVC","RBF"," ","100","0.82","0.82","0.82",".6237"])
y.add_row(["Word2Vec","SGDClassifier","Linear",".001"," ","0.80","0.75","0.
↪87",".5114"])
y.add_row(["Word2Vec","SVC","RBF"," ",".01","0.80","0.75","0.87",".5005"])
y.add_row(["tfidf-Word2Vec","SGDClassifier","Linear",".001"," ","0.80","0.
↪75","0.87",".5073"])
y.add_row(["tfidf-Word2Vec","SVC","RBF"," ",".01","0.80","0.75","0.87",".4907"])
print(y)
```

```
+-------------------------------------------+-----------------------------------
-+
|       important_features_class_[0]        |     important_features_class_[1]
|
+-------------------------------------------+-----------------------------------
-+
| (-0.055012375312047807, 'disappointed')  |    (0.07830608173821682, 'great')
|
|     (-0.04058444275252361, 'worst')       |    (0.06439161451581411, 'love')
|
|   (-0.036902078762866795, 'terrible')     |    (0.05622844115490847, 'best')
|
|  (-0.0349645694722509, 'disappointing')  |    (0.04787791021102894, 'good')
|
|      (-0.03496417187139286, 'bad')        |  (0.04449678399514948, 'delicious')
|
|    (-0.03451755640304197, 'horrible')     |   (0.03678310801238328, 'perfect')
|
|     (-0.03377887606017093, 'awful')       |  (0.03552206094557603, 'excellent')
|
|    (-0.032561847151554496, 'thought')     |  (0.03500014156903399, 'favorite')
|
| (-0.032526395176782574, 'unfortunately') |    (0.03195405346585583, 'nice')
|
|     (-0.030602582371442463, 'didnt')      | (0.031552236311105986, 'wonderful')
|
+-------------------------------------------+-----------------------------------
-+
+----------------+---------------+--------+-------+-------+----------+----------
-+--------+-------+
|   Vectorizer   |     Model     | Kernel | alpha |   C   | f1_score | Precision
| recall |  Auc  |
+----------------+---------------+--------+-------+-------+----------+----------
-+--------+-------+
|      BOW       | SGDClassifier | Linear |   1   |       |   0.91   |    0.91
|  .91   | .9405 |
```

BOW	SVC	RBF		10000	0.82	0.82	.82	.6650
tfidf	SGDClassifier	Linear	1		0.91	0.91	0.91	.9395
tfidf	SVC	RBF		100	0.82	0.82	0.82	.6237
Word2Vec	SGDClassifier	Linear	.001		0.80	0.75	0.87	.5114
Word2Vec	SVC	RBF		.01	0.80	0.75	0.87	.5005
tfidf-Word2Vec	SGDClassifier	Linear	.001		0.80	0.75	0.87	.5073
tfidf-Word2Vec	SVC	RBF		.01	0.80	0.75	0.87	.4907