DecisionTrees

April 7, 2020

[1]:

```
import numpy as np
import pandas as pd
import pydotplus
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
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from graphviz import Source
import sklearn
from sklearn.model_selection import train_test_split
from IPython.display import Image
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 cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.calibration import CalibratedClassifierCV , calibration_curve
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve,auc
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]
df_x = data_without_dup.drop(['Score'],axis=1)
```

[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

[12]:

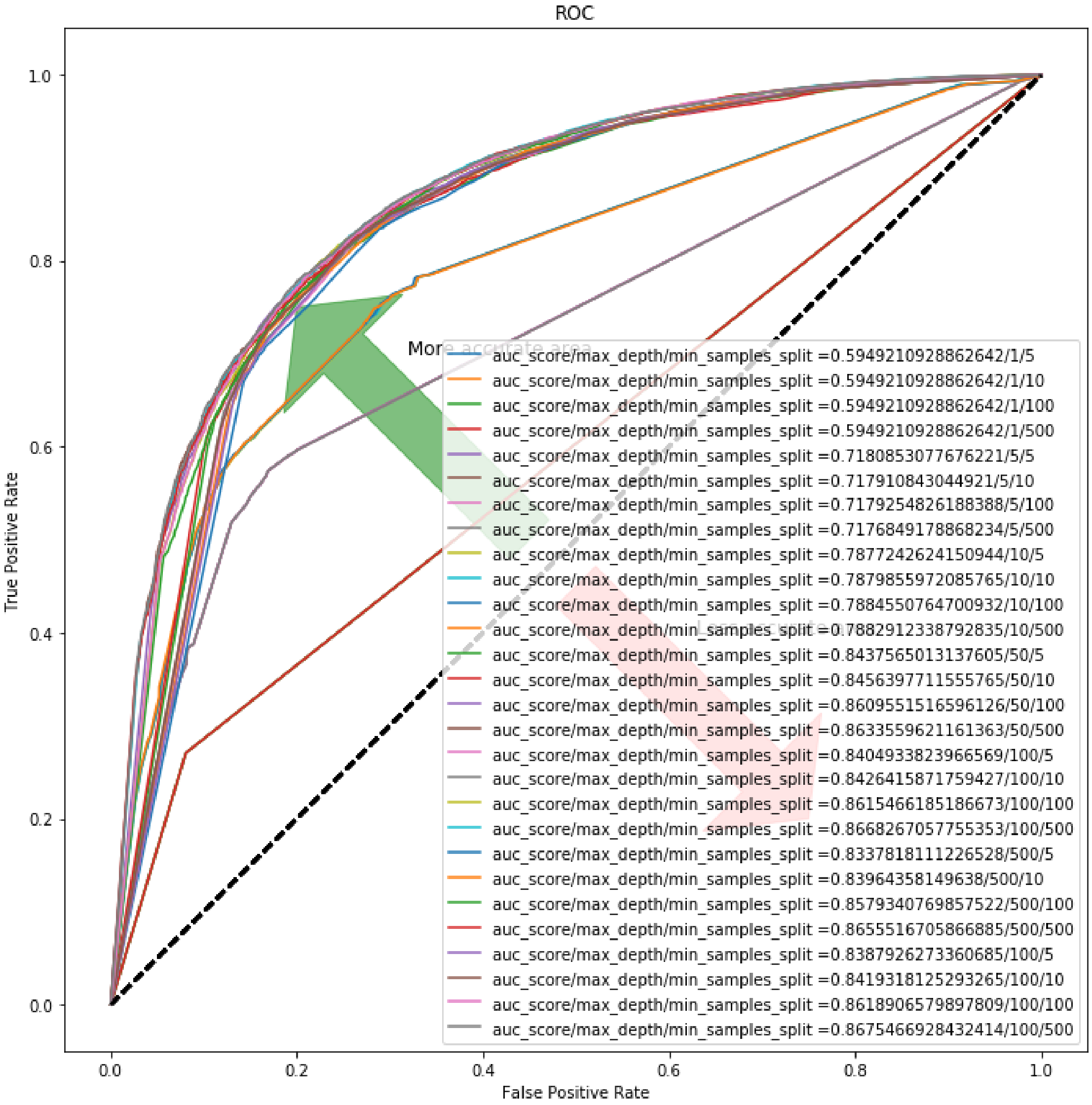
```
count_vect = CountVectorizer()
X_train_bow = count_vect.fit_transform(X_train)
X_test_bow = count_vect.transform(X_test)
count_vec = StandardScaler(with_mean=False)
X_train_bow = count_vec.fit_transform(X_train_bow)
X_test_bow = count_vec.transform(X_test_bow)
```

```
/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)
```

Hyper parameter tuning

[11]:

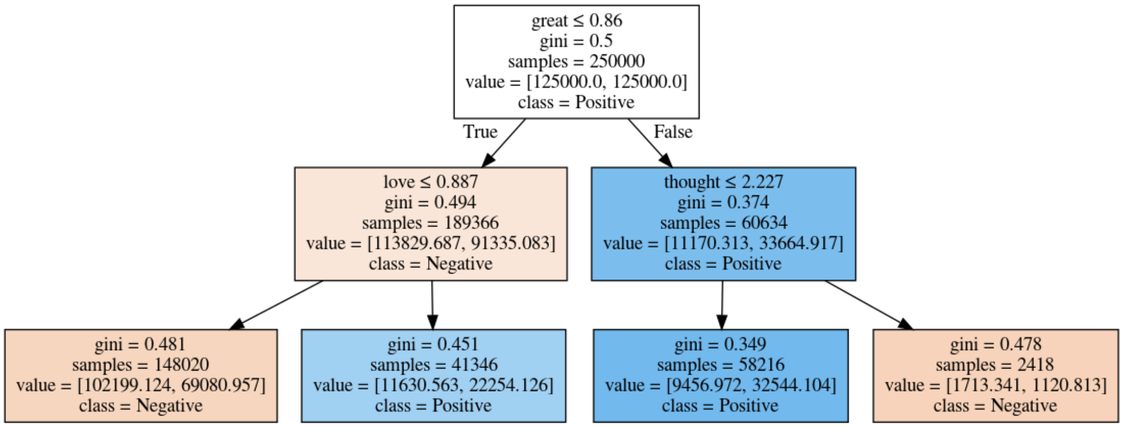
```
depth = [1, 5, 10, 50, 100, 500, 100]
min_samples_spl = [5, 10, 100, 500]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

mean_fpr = np.linspace(0,1,100)
for i in depth:
    for j in min_samples_spl:
        classifier = DecisionTreeClassifier(max_depth=i, min_samples_split=j,␣
 ↪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)
        plt.plot(fpr,tpr,label="auc_score/max_depth/min_samples_split␣
↪="+str(auc) +"/"+str(i)+"/"+str(j))
        plt.legend(loc=4)
        plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
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()
```



[32]:

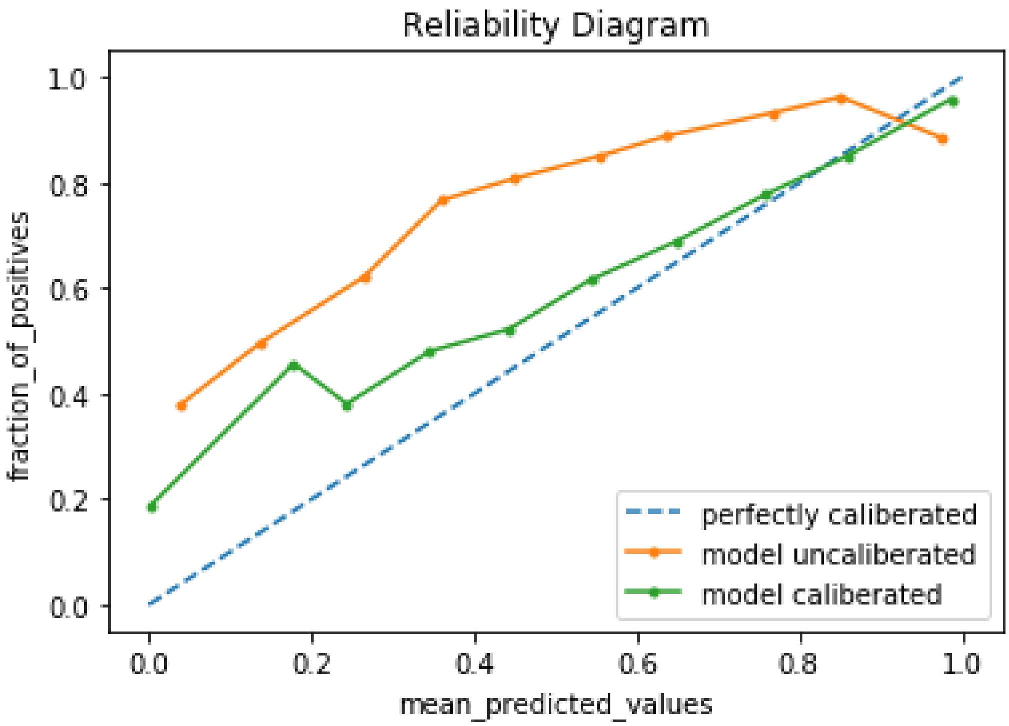
```
import graphviz
import pydotplus
from IPython.display import Image
classifier = DecisionTreeClassifier(max_depth=2,class_weight='balanced')
classifier.fit(X_train_bow,y_train)
feature_name = count_vect.get_feature_names()
target = ['Negative','Positive']
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(classifier,out_file=None, class_names=target␣
 ↪,feature_names = feature_name, filled = True,special_characters=True)
# Draw graph
graph = pydotplus.graph_from_dot_data(graph)
# Show graph
Image(graph.create_png())
```

[32]:



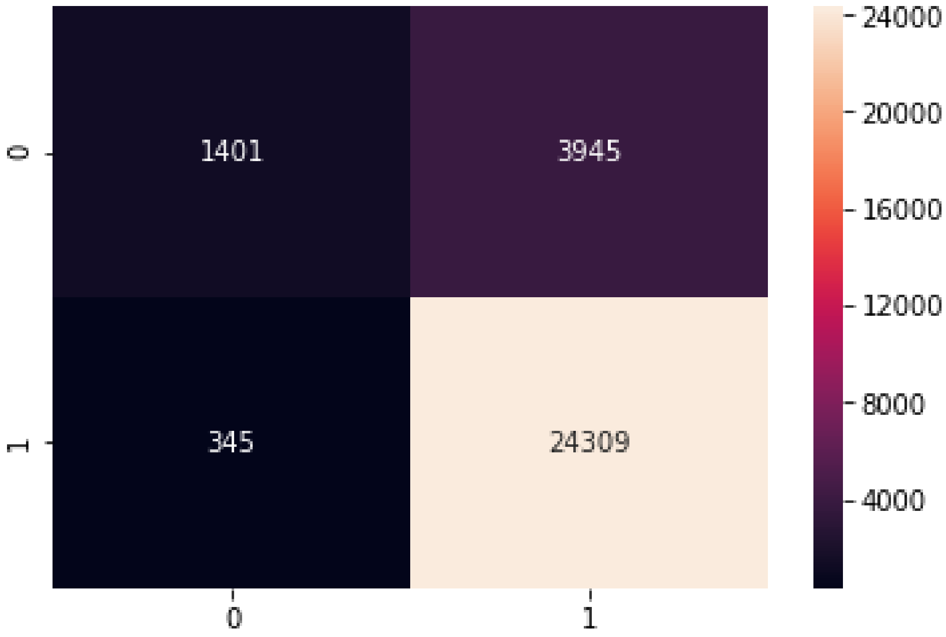
[40]:

```
classifier = DecisionTreeClassifier(max_depth=15, class_weight='balanced')
classifier.fit(X_train_bow,y_train)
#coef = classifier.coef_
probs = classifier.predict_proba(X_test_bow)[:,1]
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()
```



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

[13]:
```
classifier = DecisionTreeClassifier(max_depth=100, min_samples_split=500,␣
 ↪class_weight='balanced')
classifier.fit(X_train_bow,y_train)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_bow,y_train)
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 DecisionTreeClassifier for depth = %d and␣
 ↪min_samples_split = %d is %f%%' % (100,500, acc))
```



```
The accuracy of the DecisionTreeClassifier for depth = 100 and min_samples_split
= 500 is 85.700000%
```

[14]:
```
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       0.80      0.26      0.40      5346
           1       0.86      0.99      0.92     24654

   micro avg       0.86      0.86      0.86     30000
   macro avg       0.83      0.62      0.66     30000
weighted avg       0.85      0.86      0.83     30000
```

[15]:
```
coef = classifier.feature_importances_
class_labels = model.classes_
feature_names =count_vect.get_feature_names()
topn_class1 = sorted(zip(coef, feature_names),reverse=False)[:20]
topn_class2 = sorted(zip(coef, feature_names),reverse=True)[:20]
print("Important words in negative reviews")
for coef, feat in topn_class1:
    print(class_labels[0], coef, feat)
```

```python
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 0.0 aa
0 0.0 aaa
0 0.0 aaaa
0 0.0 aaaaa
0 0.0 aaaaaaaaaaa
0 0.0 aaaaaaaaaaaaaaa
0 0.0 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabr
0 0.0 aaaaaaaaaaaaaaaaaaargh
0 0.0 aaaaaaaaagghh
0 0.0 aaaaaaahhhhhh
0 0.0 aaaaaaarrrrrggghhh
0 0.0 aaaaaabr
0 0.0 aaaaaah
0 0.0 aaaaaahhh
0 0.0 aaaaaahhhhbr
0 0.0 aaaaaahhhhhyaaaaaa
0 0.0 aaaaaawwwwwwwwww
0 0.0 aaaaah
0 0.0 aaaaahhhhhhhhhhhhhhhhthe
0 0.0 aaaaawill
-----------------------------------------
Important words in positive reviews
1 0.0990350154071816 great
1 0.05358931950541775 love
1 0.04721148244145879 best
1 0.036247023890801 delicious
1 0.03024856779790211 disappointed
1 0.026546502488069848 bad
1 0.022347812312398498 perfect
1 0.020542713887296185 thought
1 0.019491600028405522 good
1 0.019195535082647958 favorite
1 0.018327058204032076 excellent
1 0.016600775553961878 would
1 0.014625911816804999 money
1 0.01213348699346732 worst
1 0.011647719525502632 wonderful
1 0.011115672121797858 awful
1 0.010046053897243981 highly
1 0.009916758070424038 review
1 0.009036311856948726 unfortunately
```

```
1 0.009013909009281071 nice
```

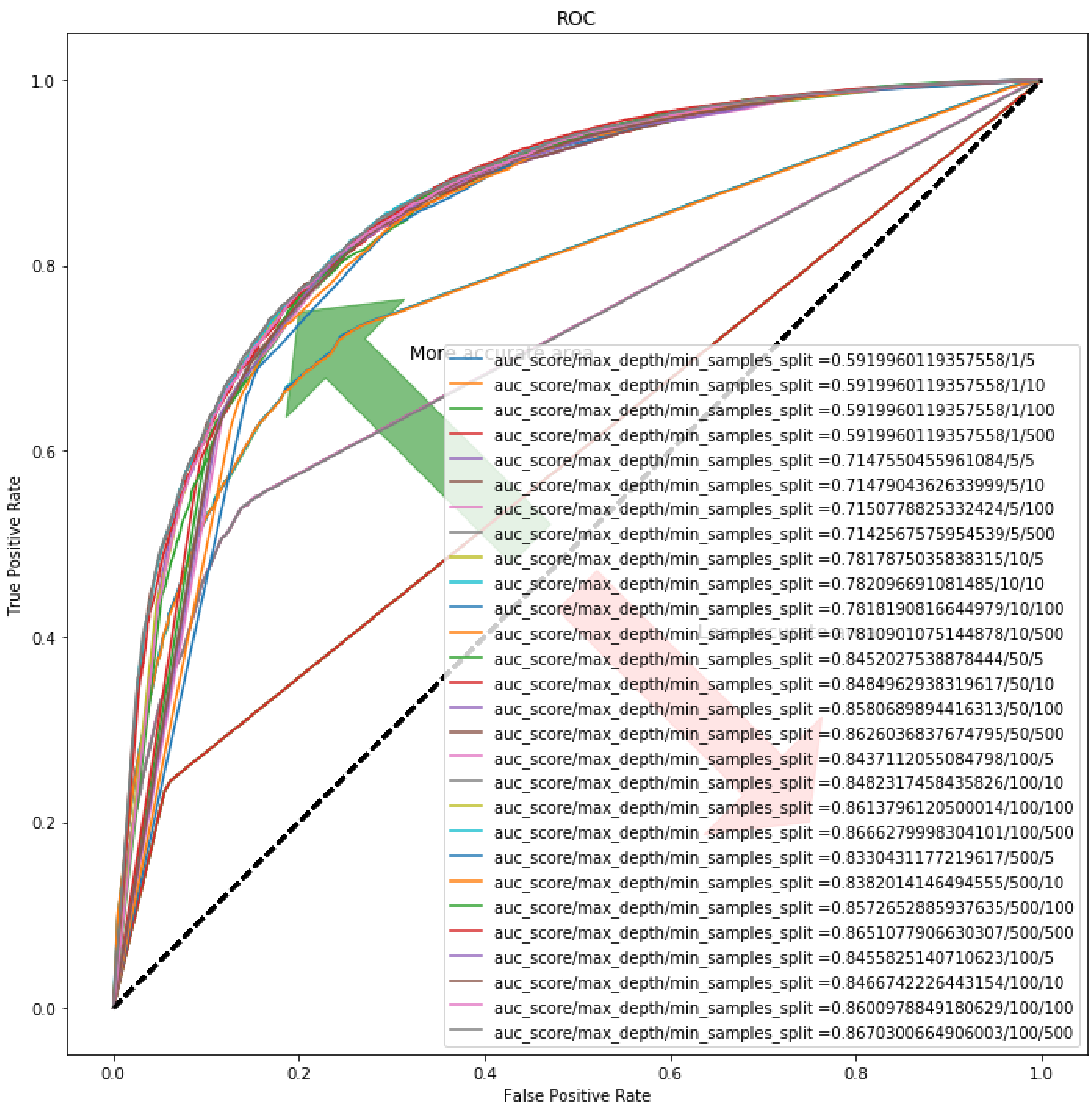
tfidf as vectorizer with Standardscaler

[127]:

```
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)
```

[46]:

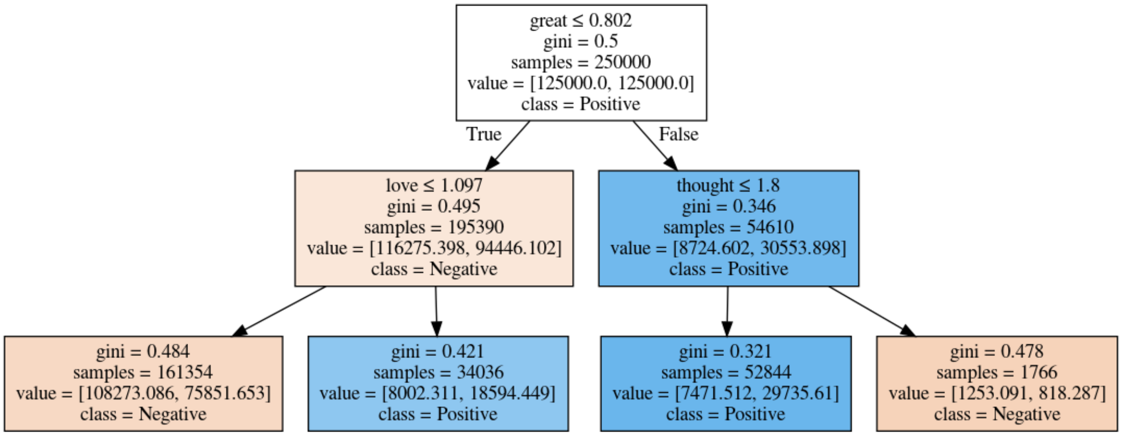
```
depth = [1, 5, 10, 50, 100, 500, 100]
min_samples_spl = [5, 10, 100, 500]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

mean_fpr = np.linspace(0,1,100)
for i in depth:
    for j in min_samples_spl:
        classifier = DecisionTreeClassifier(max_depth=i, min_samples_split=j,␣
 ↪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)
        plt.plot(fpr,tpr,label="auc_score/max_depth/min_samples_split␣
 ↪="+str(auc) +"/"+str(i)+"/"+str(j))
        plt.legend(loc=4)
        plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
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()
```



[47]:

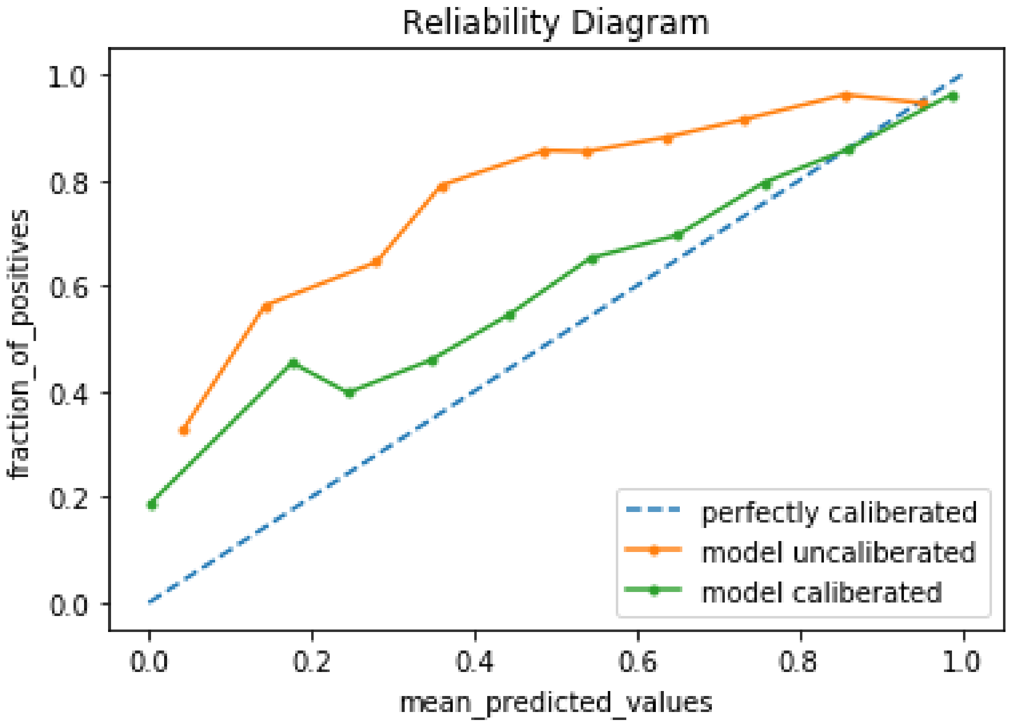
```
classifier = DecisionTreeClassifier(max_depth=2,class_weight='balanced')
classifier.fit(X_train_tfidf,y_train)
feature_name = count_vect.get_feature_names()
target = ['Negative','Positive']
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(classifier,out_file=None, class_names=target␣
 ↪,feature_names = feature_name, filled = True,special_characters=True)
# Draw graph
graph = pydotplus.graph_from_dot_data(graph)
# Show graph
Image(graph.create_png())
```

[47]:

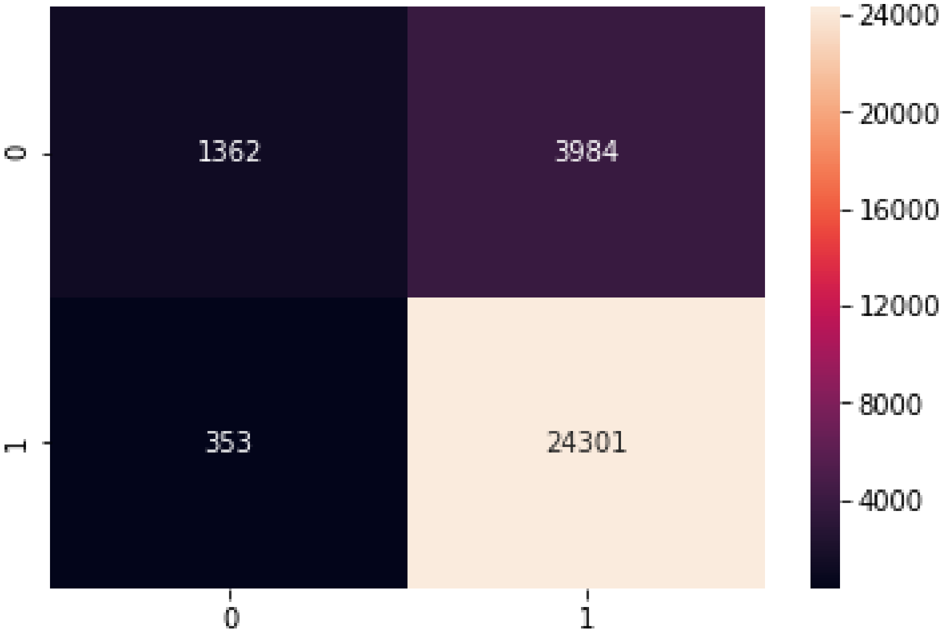


[48]:

```
classifier = DecisionTreeClassifier(max_depth=15, class_weight='balanced')
classifier.fit(X_train_tfidf,y_train)
#coef = classifier.coef_
probs = classifier.predict_proba(X_test_tfidf)[:,1]
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_bow,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()
```



```
[49]: classifier = DecisionTreeClassifier(max_depth=100, min_samples_split=500,␣
       ↪class_weight='balanced')
      classifier.fit(X_train_tfidf,y_train)
      model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
      model.fit(X_train_tfidf,y_train)
      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 DecisionTreeClassifier for depth = %d and␣
       ↪min_samples_split = %d is %f%%' % (100,500, acc))
```



```
The accuracy of the DecisionTreeClassifier for depth = 100 and min_samples_split
= 500 is 85.543333%
```

[50]:

```python
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       0.79      0.25      0.39      5346
           1       0.86      0.99      0.92     24654

   micro avg       0.86      0.86      0.86     30000
   macro avg       0.83      0.62      0.65     30000
weighted avg       0.85      0.86      0.82     30000
```

[56]:

```python
importance = classifier.feature_importances_
class_labels = model.classes_
feature_names =tfidf_vec.get_feature_names()
topn_class1 = sorted(zip(importance, feature_names),reverse=False)[:20]
topn_class2 = sorted(zip(importance, feature_names),reverse=True)[:20]
print("Important words in negative reviews")
for importanc, feat in topn_class1:
    print(class_labels[0], importanc, feat)
print("-----------------------------------------")
print("Important words in positive reviews")
for importanc, feat in topn_class2:
    print(class_labels[1], importanc, feat)
```

```
Important words in negative reviews
0 0.0 aa
0 0.0 aaa
0 0.0 aaaa
0 0.0 aaaaa
0 0.0 aaaaaaaaaaa
0 0.0 aaaaaaaaaaaaaaa
0 0.0 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabr
0 0.0 aaaaaaaaaaaaaaaaaaaargh
0 0.0 aaaaaaaaagghh
0 0.0 aaaaaaahhhhhh
0 0.0 aaaaaaarrrrrggghhh
0 0.0 aaaaaabr
0 0.0 aaaaaah
0 0.0 aaaaaahhh
0 0.0 aaaaaahhhhbr
0 0.0 aaaaaahhhhhyaaaaaa
0 0.0 aaaaaawwwwwwwwww
0 0.0 aaaaah
0 0.0 aaaaahhhhhhhhhhhhhhhhthe
0 0.0 aaaaawill
-----------------------------------------
Important words in positive reviews
1 0.10418003844166326 great
1 0.055505336973219896 love
1 0.05124971894124606 best
1 0.03777943866638621 delicious
1 0.026288969232898297 disappointed
1 0.023825810017042738 perfect
1 0.023316608928900848 good
1 0.02036001477438885 favorite
1 0.02014687423002695 bad
1 0.018050280727092333 thought
1 0.017882415172827775 excellent
1 0.013602705438262717 wonderful
1 0.01174013265024442 highly
1 0.011696928089392907 money
1 0.01162862553202851 easy
1 0.01096842614565546 nice
1 0.010109408694509759 awful
1 0.009656989254082569 worst
1 0.0077253510765387156 taste
```

```
1 0.007634554669081282 terrible
```

Word2Vec as vectorizer

[63]:
```
list_of_sent = []

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

[64]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

[66]:
```
mod = KeyedVectors.load_word2vec_format("/home/niranjan/Downloads/GoogleNews-vectors-negative300.bin", binary=True)
```

[101]:
```
X_train_avg_w2v = []
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 list_of_sent[0:100000]:
    sent_vec = np.zeros(100)
    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)
```

[102]:
```
X_test_avg_w2v = []
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 list_of_sent[100000:120000]:
    sent_vec = np.zeros(100)
    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)
```

[28]:

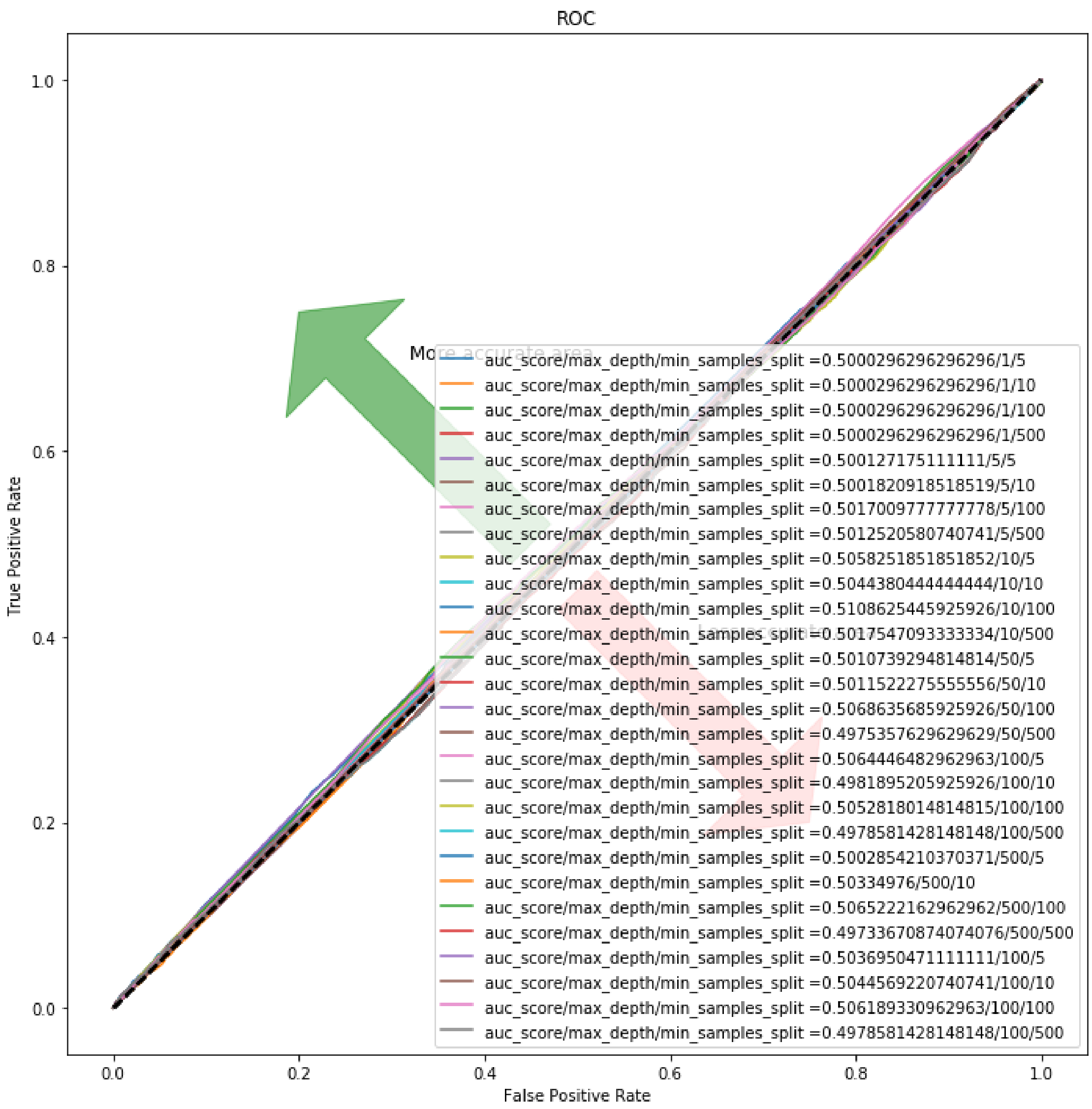
```
import pickle
pickle_out1 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
 ↪X_train_avg_w2v","rb")
pickle_out2 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
 ↪X_cv_avg_w2v","rb")
X_train_avg_w2v = pickle.load(pickle_out1)
X_test_avg_w2v = pickle.load(pickle_out2)
pickle_out1.close()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

[19]:

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

[59]:

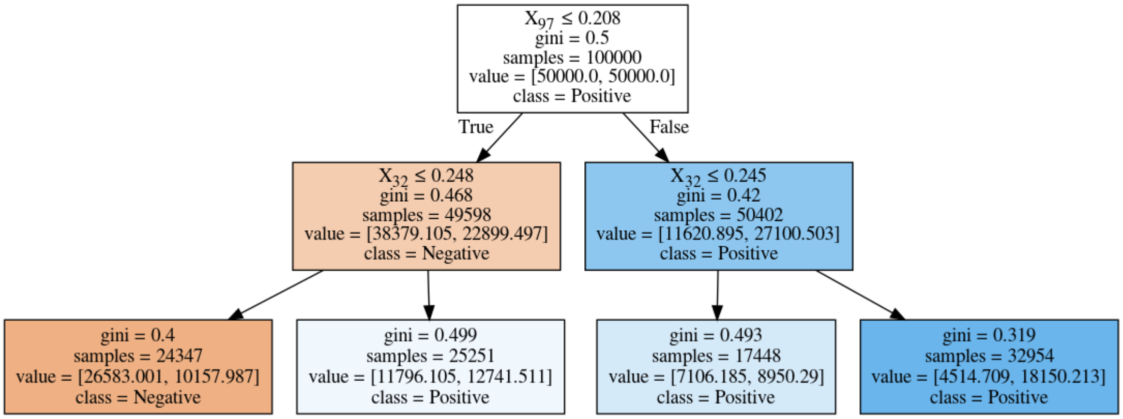
```
depth = [1, 5, 10, 50, 100, 500, 100]
min_samples_spl = [5, 10, 100, 500]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

mean_fpr = np.linspace(0,1,100)
for i in depth:
    for j in min_samples_spl:
        classifier = DecisionTreeClassifier(max_depth=i, min_samples_split=j,␣
 ↪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)
        plt.plot(fpr,tpr,label="auc_score/max_depth/min_samples_split␣
 ↪="+str(auc) +"/"+str(i)+"/"+str(j))
        plt.legend(loc=4)
        plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
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()
```



[104]:

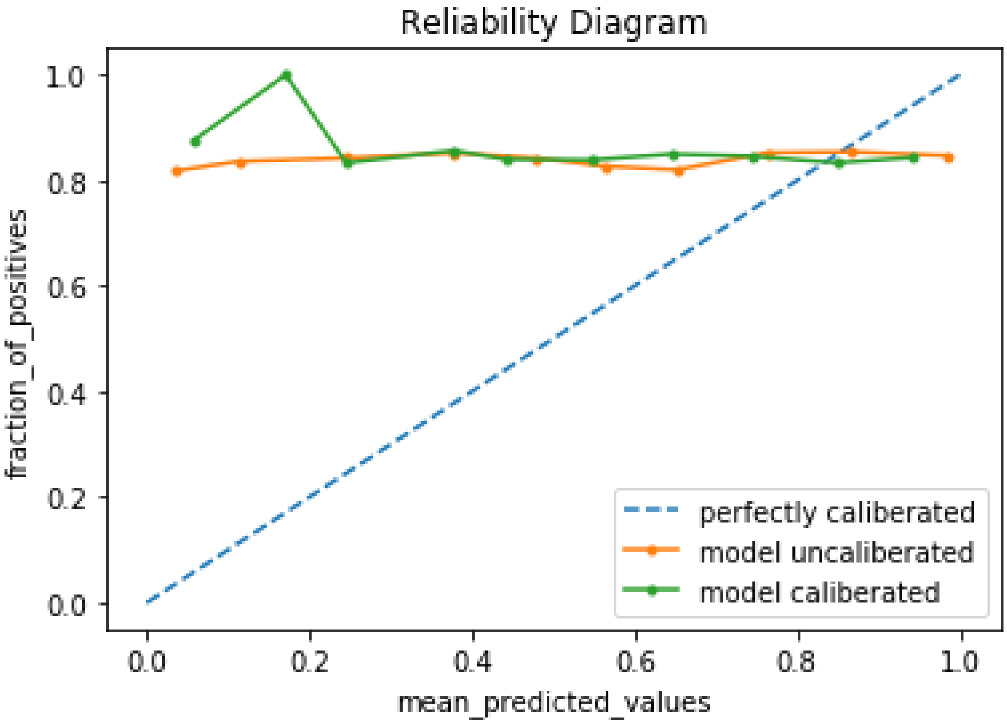
```
classifier = DecisionTreeClassifier(max_depth=2,class_weight='balanced')
classifier.fit(X_train_avg_w2v,y_train)
#feature_name = count_vect.get_feature_names()
target = ['Negative','Positive']
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(classifier,out_file=None, class_names=target ,filled = True,special_characters=True)
# Draw graph
graph = pydotplus.graph_from_dot_data(graph)
# Show graph
Image(graph.create_png())
```

[104]:



[29]:

```
classifier = DecisionTreeClassifier(max_depth=15, class_weight='balanced')
classifier.fit(X_train_avg_w2v,y_train)
#coef = classifier.coef_
probs = classifier.predict_proba(X_test_avg_w2v)[:,1]
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()
```



[30]:

```
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

[105]:

```
# 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
```

```
100%|      | 100000/100000 [2:06:45<00:00,  8.29it/s]
```

[107]:

```
# 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
```

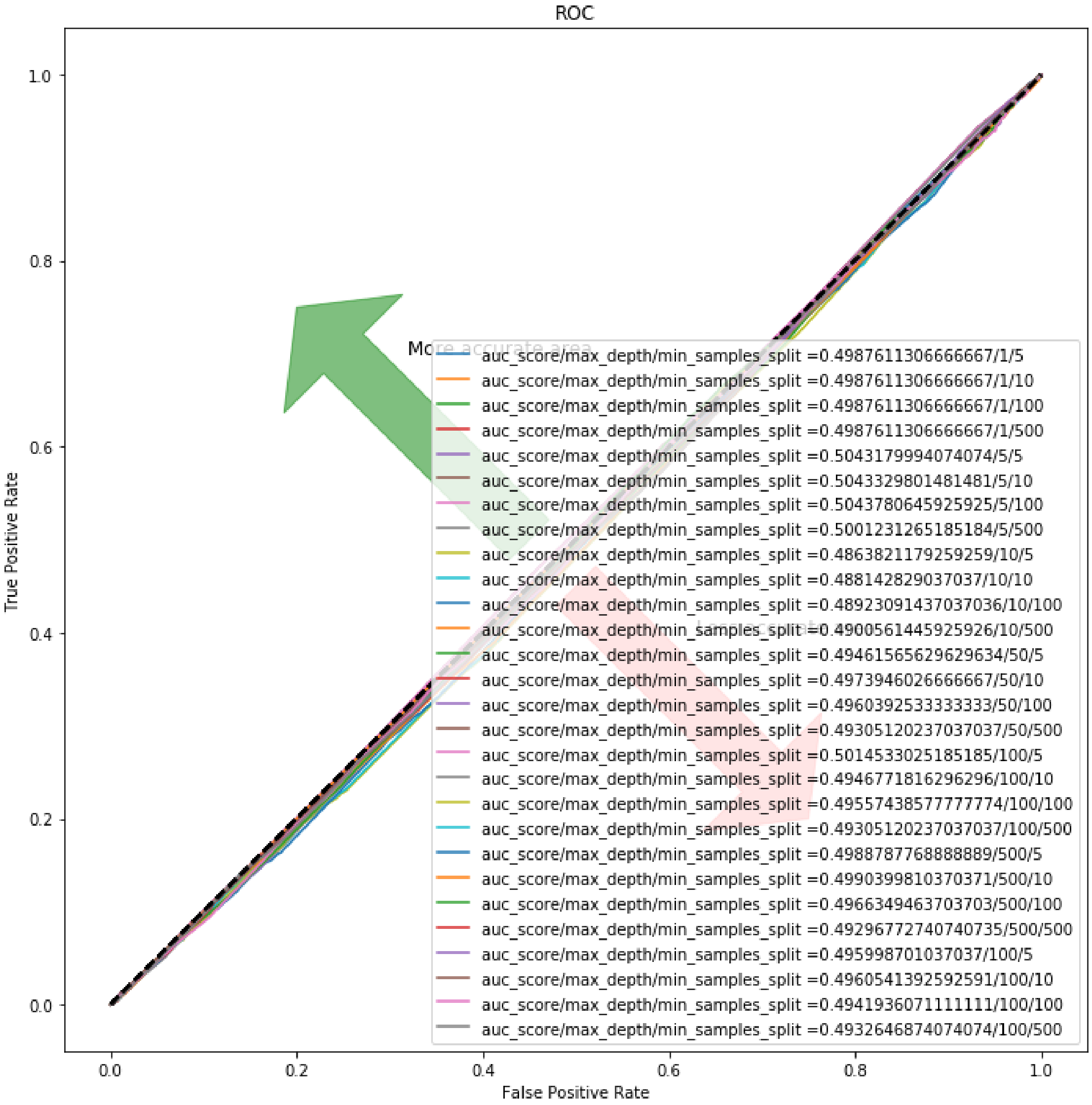
```
100%|      | 20000/20000 [07:21<00:00, 45.26it/s]
```

[4]:

```
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()
```

[84]:

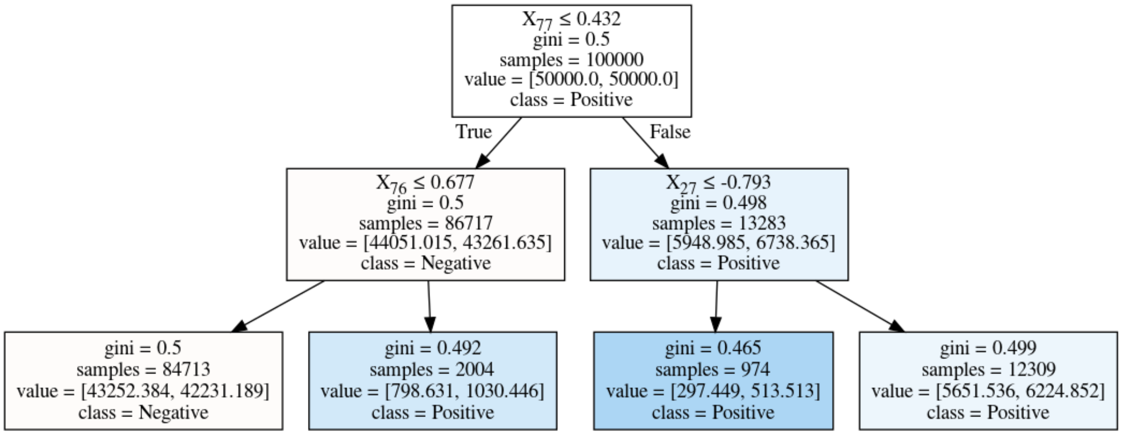
```
depth = [1, 5, 10, 50, 100, 500, 100]
min_samples_spl = [5, 10, 100, 500]
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

mean_fpr = np.linspace(0,1,100)
for i in depth:
    for j in min_samples_spl:
        classifier = DecisionTreeClassifier(max_depth=i, min_samples_split=j,␣
 ↪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)
        plt.plot(fpr,tpr,label="auc_score/max_depth/min_samples_split␣
↪="+str(auc) +"/"+str(i)+"/"+str(j))
        plt.legend(loc=4)
        plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
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()
```



[24]:

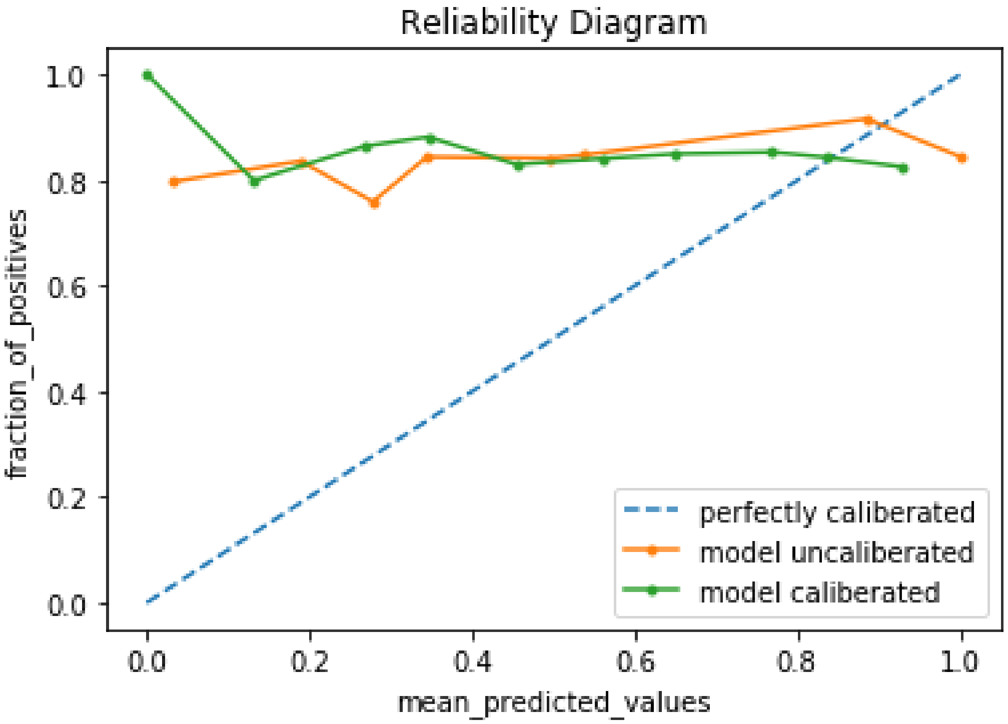
```
classifier = DecisionTreeClassifier(max_depth=2,class_weight='balanced')
classifier.fit(tfidf_train_vectors,y_train)
target = ['Negative','Positive']
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(classifier,out_file=None, class_names=target,␣
 ↪filled = True,special_characters=True)
# Draw graph
graph = pydotplus.graph_from_dot_data(graph)
# Show graph
Image(graph.create_png())
```

[24]:

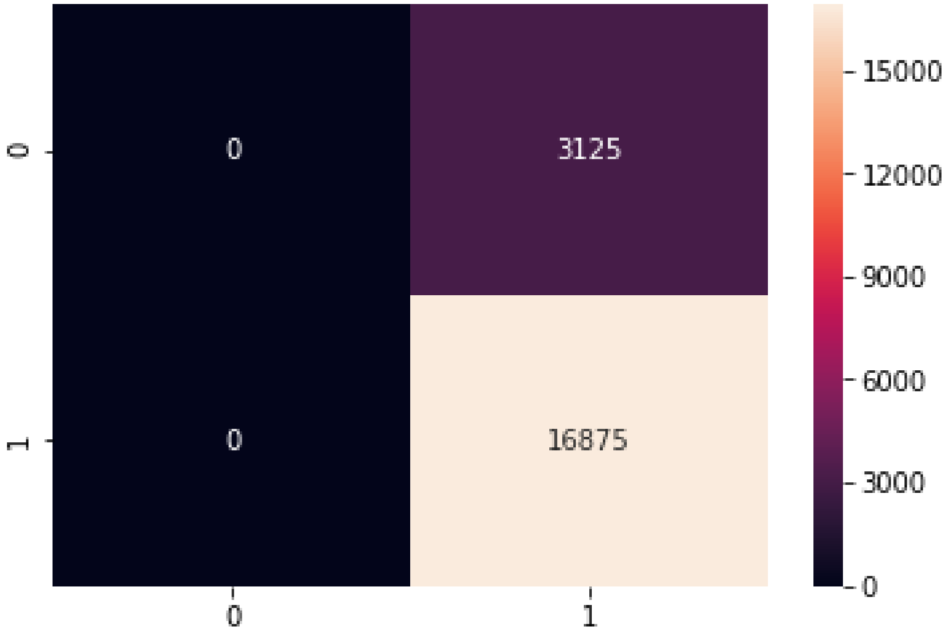


[94]:

```
classifier = DecisionTreeClassifier(max_depth=15, class_weight='balanced')
classifier.fit(tfidf_train_vectors,y_train)
#coef = classifier.coef_
probs = classifier.predict_proba(X_test_avg_w2v)[:,1]
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()
```



```
[124]: classifier = DecisionTreeClassifier(max_depth=100, min_samples_split=500,␣
        ↪class_weight='balanced')
       classifier.fit(tfidf_train_vectors,y_train)
       model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
       model.fit(tfidf_train_vectors,y_train)
       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 DecisionTreeClassifier for depth = %d and␣
        ↪min_samples_split = %d is %f%%' % (100,500, acc))
```



```
The accuracy of the DecisionTreeClassifier for depth = 100 and min_samples_split
= 500 is 84.375000%
```

[125]:

```
y_pred = model.predict(tfidf_test_vectors)
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


/home/niranjan/anaconda3/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/niranjan/anaconda3/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
```

```
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/niranjan/anaconda3/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

[19]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.add_column("important_features_class_[0]",topn_class1)
x.add_column("important_features_class_[1]",topn_class2)
y = PrettyTable()
y.
 ↪field_names=["Vectorizer","Model","depth","min_samples_split","Auc_score","f1_score(micro␣
 ↪average)"]
y.add_row(["BOW","DecisionTreeClassifier","100","500",".8668","0.86"])
y.add_row(["tfidf","DecisionTreeClassifier","100","500",".8666","0.86"])
y.add_row(["Word2Vec","DecisionTreeClassifier","10","100",".5108","0.84"])
y.add_row(["tfidf-Word2Vec","DecisionTreeClassifier","5","500",".5001","0.84"])
z = PrettyTable()
z.field_names = ["Vectorizer","Model","Precision", "Recall"]
z.add_row(["BOW","DecisionTreeClassifier",0.86,0.86])
z.add_row(["tfidf","DecisionTreeClassifier",0.86,0.86])
z.add_row(["Word2Vec","DecisionTreeClassifier",0.84,0.84])
z.add_row(["tfidf-Word2Vec","DecisionTreeClassifier",0.84,0.84])

print(x)
print(y)
print(z)
```

```
+----------------------------------------------------------+---------------------
--------------------+
|               important_features_class_[0]               |
important_features_class_[1]      |
+----------------------------------------------------------+---------------------
--------------------+
|                        (0.0, 'aa')                       |
(0.0990350154071816, 'great')      |
|                        (0.0, 'aaa')                      |
(0.05358931950541775, 'love')      |
|                       (0.0, 'aaaa')                      |
(0.04721148244145879, 'best')      |
|                       (0.0, 'aaaaa')                     |
(0.036247023890801, 'delicious')    |
|                    (0.0, 'aaaaaaaaaaa')                  |
(0.03024856779790211, 'disappointed')  |
|                  (0.0, 'aaaaaaaaaaaaaaa')                |
(0.026546502488069848, 'bad')      |
| (0.0, 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabr') |
(0.022347812312398498, 'perfect')    |
|             (0.0, 'aaaaaaaaaaaaaaaaaaaargh')             |
(0.020542713887296185, 'thought')    |
|                  (0.0, 'aaaaaaaaagghh')                  |
(0.019491600028405522, 'good')     |
|                  (0.0, 'aaaaaaahhhhhh')                  |
(0.019195535082647958, 'favorite')   |
|                (0.0, 'aaaaaaarrrrggghhh')                |
(0.018327058204032076, 'excellent')   |
|                    (0.0, 'aaaaaabr')                     |
(0.016600775553961878, 'would')     |
|                     (0.0, 'aaaaaah')                     |
(0.014625911816804999, 'money')     |
|                    (0.0, 'aaaaaahhh')                    |
(0.01213348699346732, 'worst')     |
|                  (0.0, 'aaaaaahhhhbr')                   |
(0.011647719525502632, 'wonderful')   |
|                (0.0, 'aaaaaahhhhyaaaaaa')                |
(0.011115672121797858, 'awful')     |
|                (0.0, 'aaaaaawwwwwwwwww')                 |
(0.010046053897243981, 'highly')    |
|                     (0.0, 'aaaaah')                      |
(0.009916758070424038, 'review')    |
|            (0.0, 'aaaaahhhhhhhhhhhhhhhhthe')             |
(0.009036311856948726, 'unfortunately') |
|                    (0.0, 'aaaaawill')                    |
(0.00901390900928107, 'nice')      |
+----------------------------------------------------------+--------------------
---------------------+
+----------------+------------------------+-------+-------------------+---------
--+-------------------------+
|   Vectorizer   |         Model          | depth | min_samples_split |
Auc_score | f1_score(micro average) |
+----------------+------------------------+-------+-------------------+---------
--+-------------------------+
|      BOW       | DecisionTreeClassifier |  100  |        500        |   .8668
|           0.86          |
|     tfidf      | DecisionTreeClassifier |  100  |        500        |   .8666
|           0.86          |
|    Word2Vec    | DecisionTreeClassifier |   10  |        100        |   .5108
|           0.84          |
| tfidf-Word2Vec | DecisionTreeClassifier |   5   |        500        |   .5001
|           0.84          |
+----------------+------------------------+-------+-------------------+---------
--+-------------------------+
+----------------+------------------------+-----------+--------+
|   Vectorizer   |         Model          | Precision | Recall |
+----------------+------------------------+-----------+--------+
|      BOW       | DecisionTreeClassifier |    0.86   |  0.86  |
|     tfidf      | DecisionTreeClassifier |    0.86   |  0.86  |
|    Word2Vec    | DecisionTreeClassifier |    0.84   |  0.84  |
| tfidf-Word2Vec | DecisionTreeClassifier |    0.84   |  0.84  |
+----------------+------------------------+-----------+--------+
```