randomforest_XGBoost (1)

April 7, 2020

[36]:
```
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
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
import sklearn
import pydotplus
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 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
```

[37]:
```
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

[38]:
```
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)
```

[39]:
```
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!
```

[40]:
```
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.**

[41]:
```
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
```

[42]:
```
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')
```

[43]:
```
X = data_without_dup['final_string']
y = data_without_dup['Score']
```

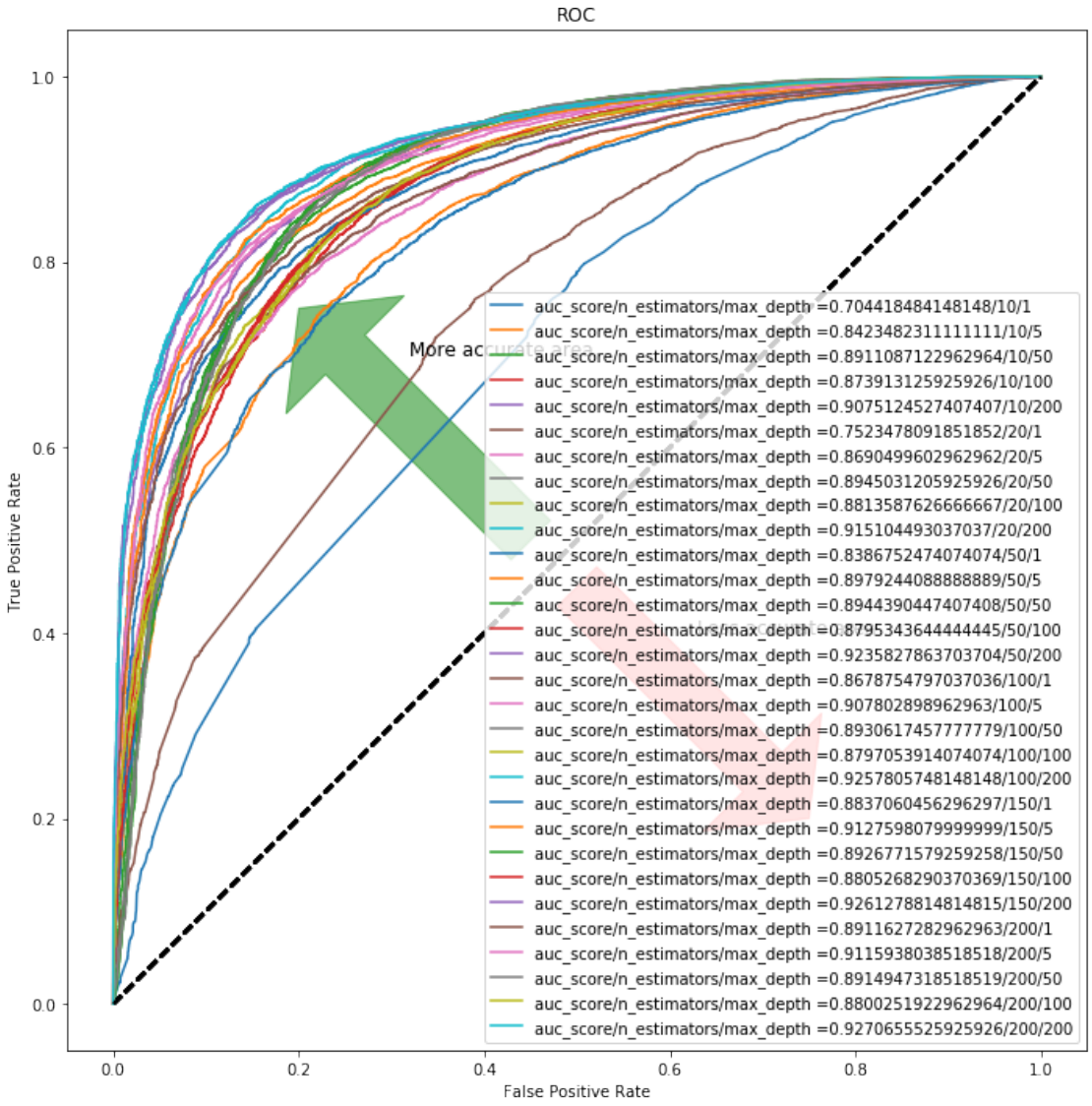
[44]:
```
X_train= X[0:100000]
y_train = y[0:100000]
X_test= X[100000:120000]
```

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

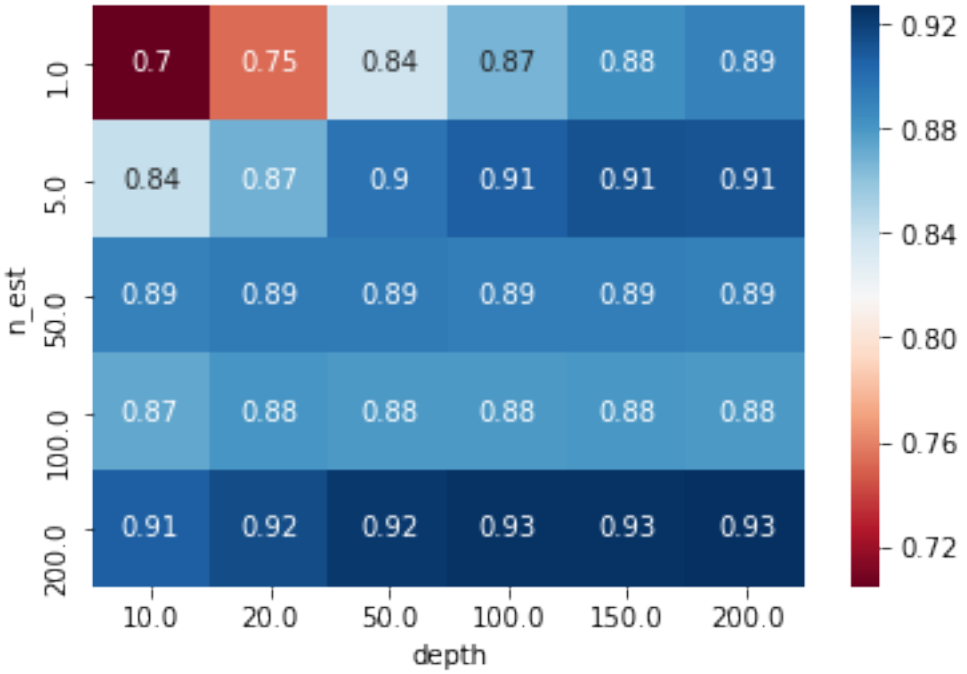
[45]:
```
count_vect = CountVectorizer()
X_train_bow = count_vect.fit_transform(X_train)
X_test_bow = count_vect.transform(X_test)
count_vec = StandardScaler(with_mean=False)
X_train_bow = count_vec.fit_transform(X_train_bow)
X_test_bow = count_vec.transform(X_test_bow)
```

[184]:
```
depth = [1,5,50,100,200]
n_est = [10, 20, 50, 100,150, 200]
auc_lis=[]
depth_lis = []
n_est_lis = []
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 n_est:
    for j in depth:
        classifier =␣
↪RandomForestClassifier(n_estimators=i,max_depth=j,class_weight='balanced',n_jobs=-1,␣
↪bootstrap=True)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        n_est_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/n_estimators/max_depth ="+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()
```

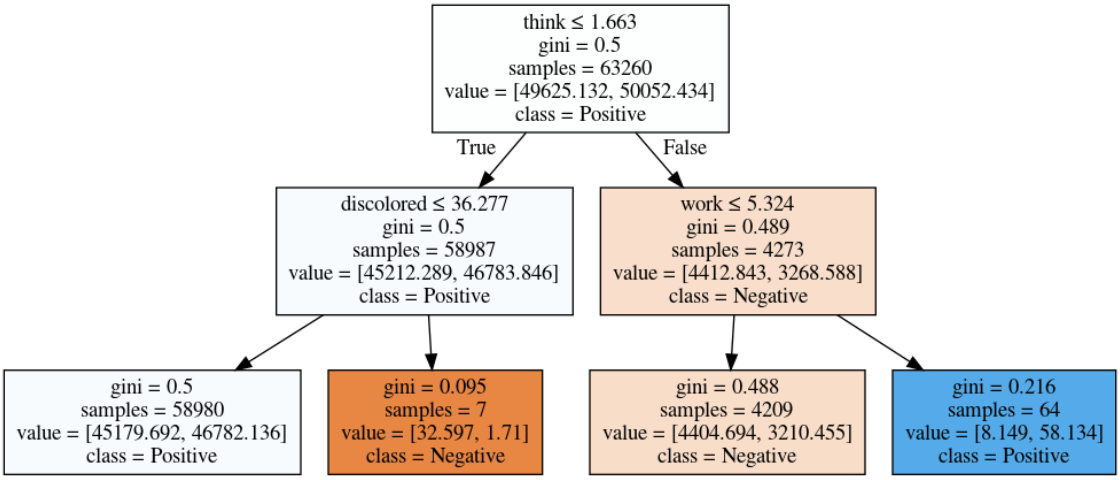


[186]:
```
X_bow = np.array(n_est_lis)
Y_bow = np.array(depth_lis)
Z_bow = np.array(auc_lis)

df_bow = pd.DataFrame.from_dict(np.array([X_bow,Y_bow,Z_bow]).T)
df_bow.columns = ['n_est','depth','auc']
df_bow['auc'] = pd.to_numeric(df_bow['auc'])
pivotted= df_bow.pivot('n_est','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```

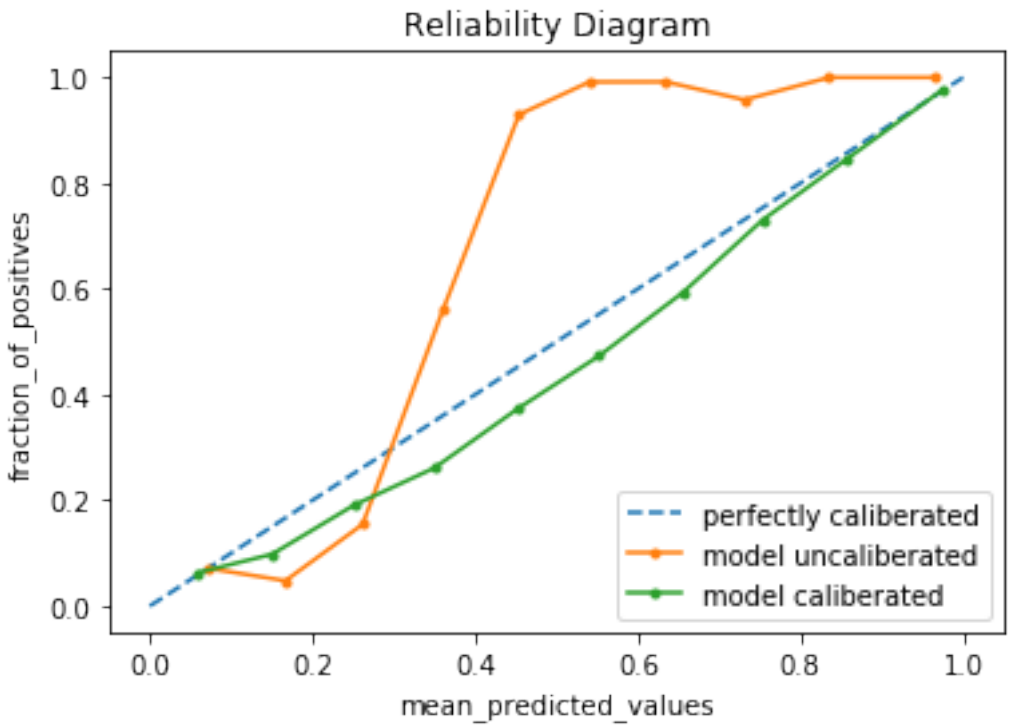


[192]:
```
classifier =␣
↪RandomForestClassifier(max_depth=2,class_weight='balanced',n_estimators=200)
classifier.fit(X_train_bow,y_train)
est = classifier.estimators_[5]
feature_name = count_vect.get_feature_names()
target = ['Negative','Positive']
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(est,out_file=None, 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())
```

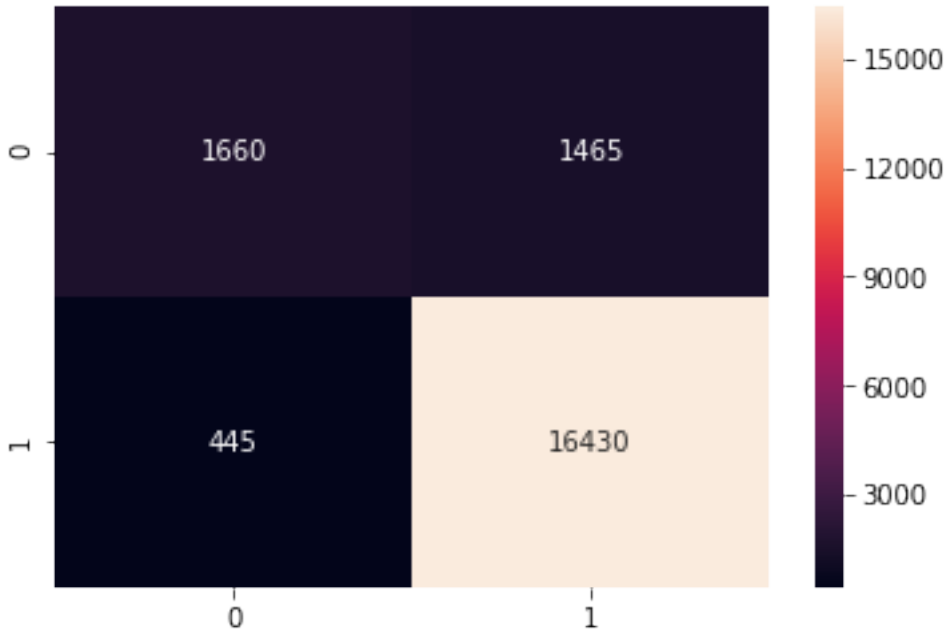
[192]:



[46]:
```
classifier = RandomForestClassifier(n_estimators=200,max_depth=20,␣
↪class_weight='balanced',bootstrap=True)
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()
```



[195]:
```
classifier = RandomForestClassifier(n_estimators=200,max_depth=20,␣
 ↪bootstrap=True, 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␣
 ↪n_estimators = %d is %f%%' % (20,200, acc))
```



The accuracy of the DecisionTreeClassifier for depth = 20 and n_estimators = 200 is 90.450000%

[196]:
```
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      0.53      0.63      3125
           1       0.92      0.97      0.95     16875

   micro avg       0.90      0.90      0.90     20000
   macro avg       0.85      0.75      0.79     20000
weighted avg       0.90      0.90      0.90     20000
```

[55]:
```
importance = classifier.feature_importances_
class_labels = model.classes_
feature_names =count_vect.get_feature_names()
topn_class1_dtree = sorted(zip(importance, feature_names),reverse=False)[:20]
topn_class2_dtree = sorted(zip(importance, feature_names),reverse=True)[:20]
print("Important words in negative reviews")
for importanc, feat in topn_class1_dtree:
    print(class_labels[0], importanc, feat)
```

```
print("----------------------------------------")
print("Important words in positive reviews")
for importanc, feat in topn_class2_dtree:
    print(class_labels[1], importanc, feat)
```

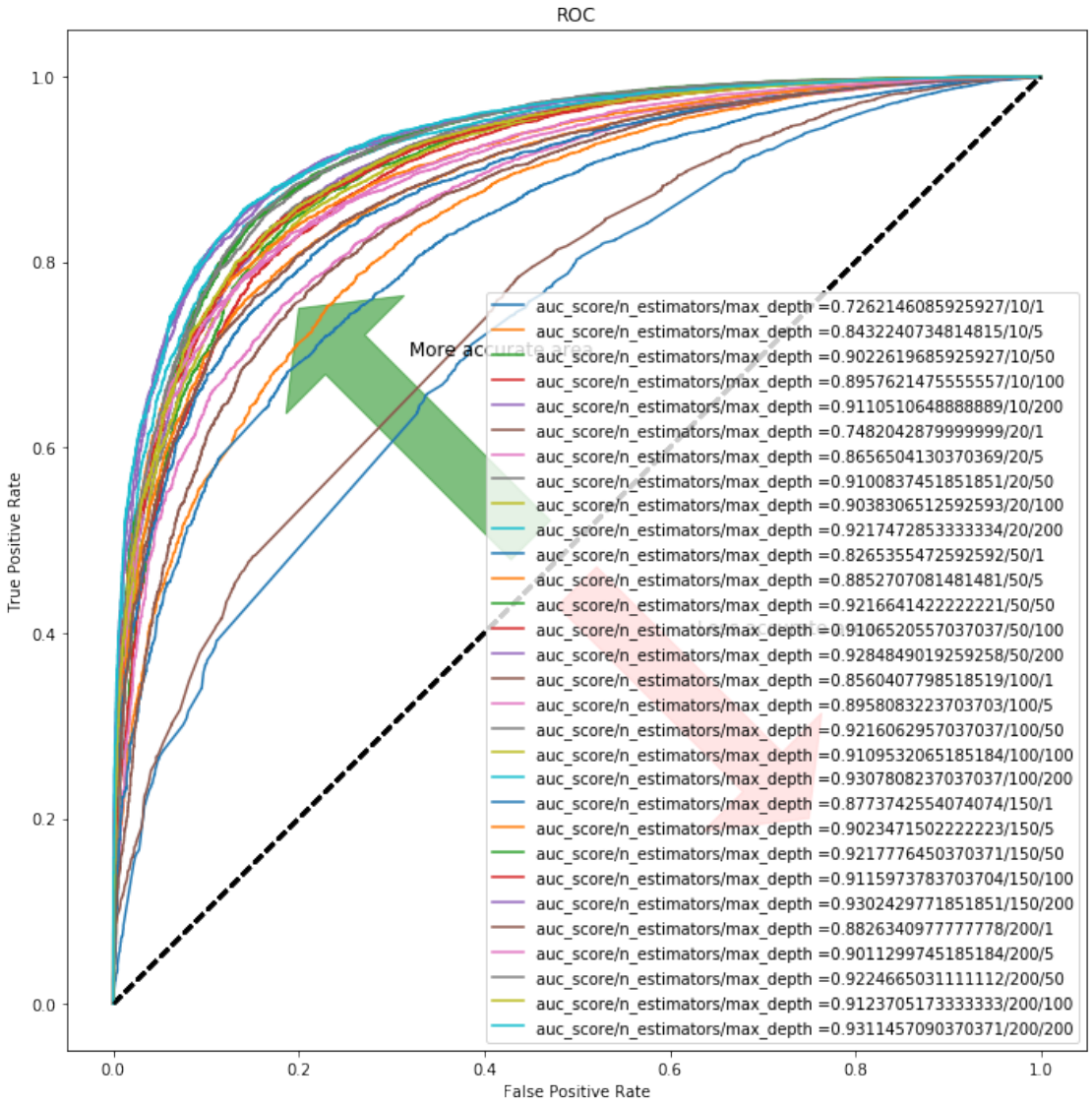
```
Important words in negative reviews
0 0.0 aa
0 0.0 aaa
0 0.0 aaaa
0 0.0 aaaaa
0 0.0 aaaaaaaaaaaaaaa
0 0.0 aaaaaaaaagghh
0 0.0 aaaaaabr
0 0.0 aaaaah
0 0.0 aaaaahhhhhhhhhhhhhhhthe
0 0.0 aaaah
0 0.0 aaah
0 0.0 aaahhhhhh
0 0.0 aaahs
0 0.0 aachen
0 0.0 aacurate
0 0.0 aad
0 0.0 aadp
0 0.0 aaf
0 0.0 aafco
0 0.0 aafter
----------------------------------------
Important words in positive reviews
1 0.006829901 delicious
1 0.0059515843 worst
1 0.0051996135 best
1 0.0049646627 great
1 0.0049589006 perfect
1 0.0045655733 highly
1 0.0043366887 threw
1 0.0042989836 waste
1 0.004289796 excellent
1 0.0040165526 easy
1 0.003805688 yummy
1 0.0038038422 favorite
1 0.0035729995 awesome
1 0.0035460214 gross
1 0.0034753508 horrible
1 0.003406402 wonderful
1 0.0032368226 terrible
1 0.0031637086 hooked
1 0.003051933 money
```

```
1 0.0030434837 yuck
```

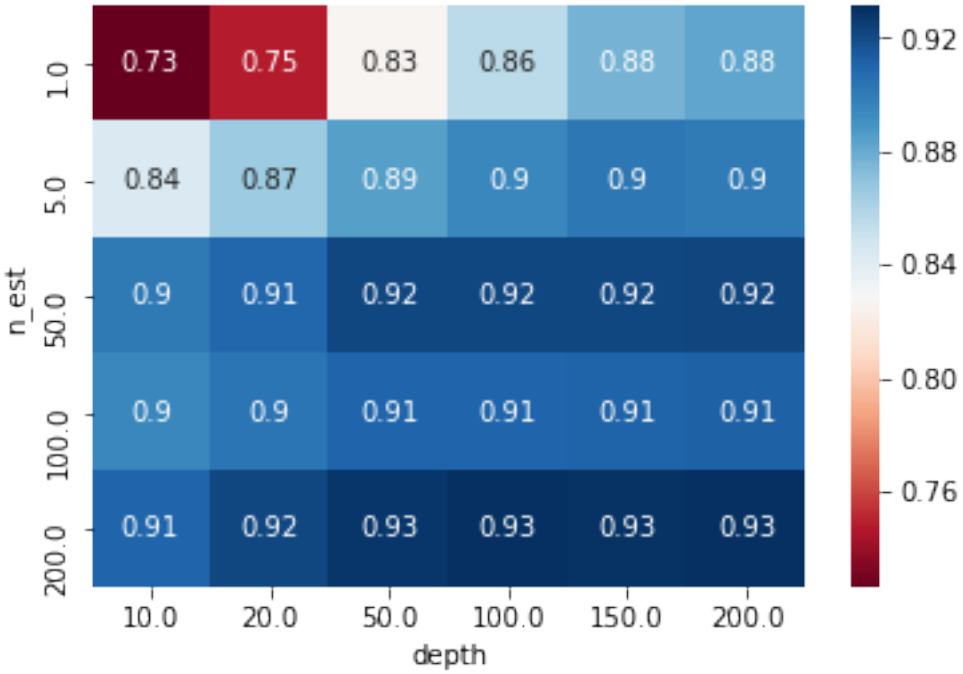
[48]:
```
tfidf_vec = TfidfVectorizer()
X_train_tfidf = tfidf_vec.fit_transform(X_train)
X_test_tfidf = tfidf_vec.transform(X_test)
vec = StandardScaler(with_mean=False)
X_train_tfidf = vec.fit_transform(X_train_tfidf)
X_test_tfidf = vec.transform(X_test_tfidf)
```

[199]:
```
depth = [1,5,50,100,200]
n_est = [10, 20, 50, 100,150, 200]
auc_lis=[]
depth_lis = []
n_est_lis = []
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 n_est:
    for j in depth:
        classifier =␣
 ↪RandomForestClassifier(n_estimators=i,max_depth=j,class_weight='balanced',n_jobs=-1,␣
 ↪bootstrap=True)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        n_est_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/n_estimators/max_depth ="+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()
```

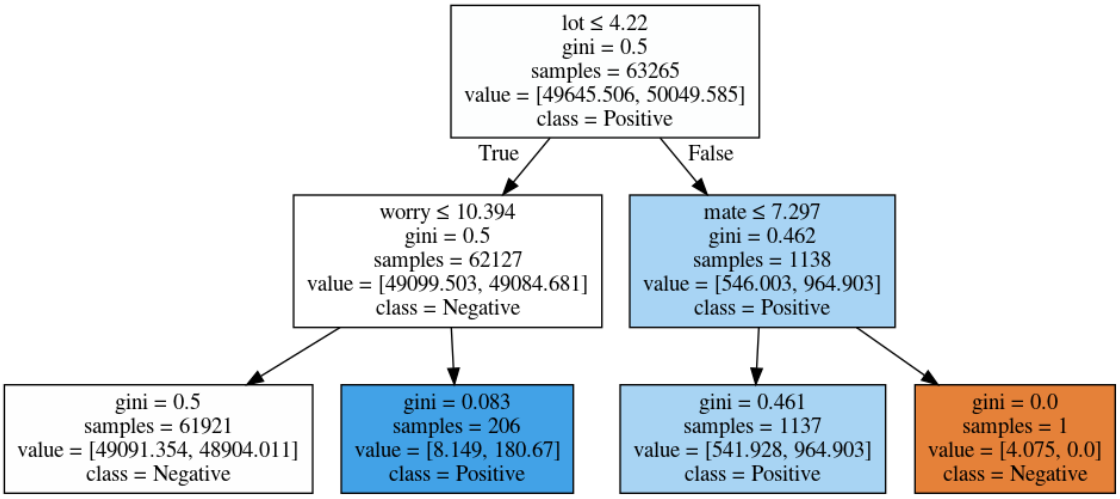


[200]:
```
X_tfidf = np.array(n_est_lis)
Y_tfidf = np.array(depth_lis)
Z_tfidf = np.array(auc_lis)

df_tfidf = pd.DataFrame.from_dict(np.array([X_tfidf,Y_tfidf,Z_tfidf]).T)
df_tfidf.columns = ['n_est','depth','auc']
df_tfidf['auc'] = pd.to_numeric(df_tfidf['auc'])
pivotted= df_tfidf.pivot('n_est','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```

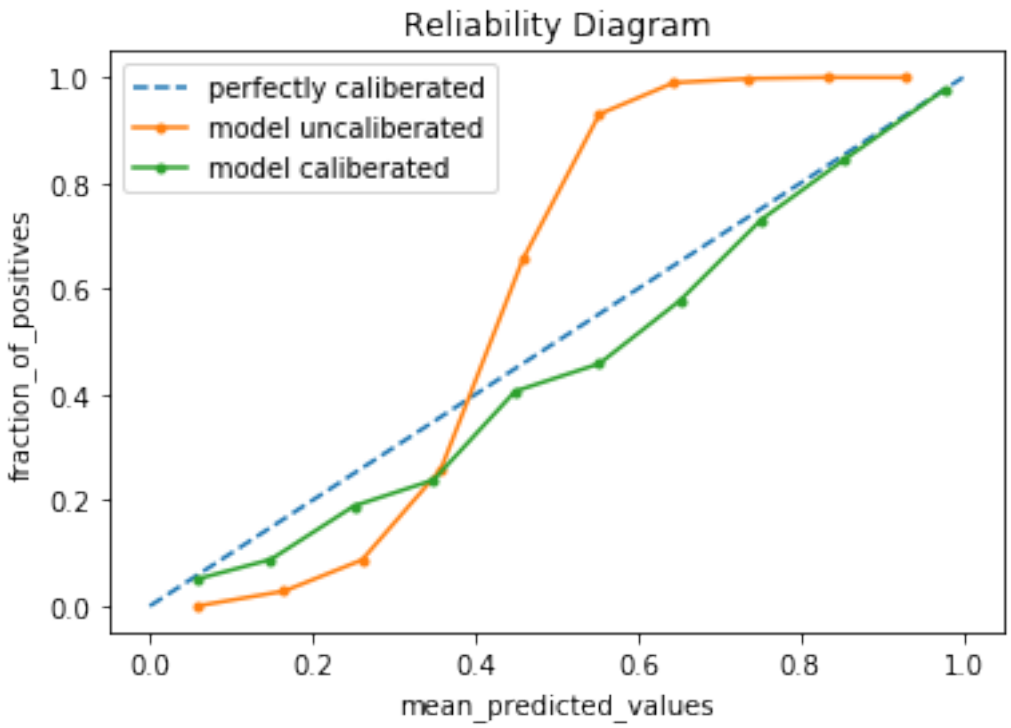


[201]:
```
classifier =␣
↪RandomForestClassifier(n_estimators=200,max_depth=2,class_weight='balanced')
classifier.fit(X_train_tfidf,y_train)
est = classifier.estimators_[5]
feature_name = count_vect.get_feature_names()
target = ['Negative','Positive']
from IPython.display import Image
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(est,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())
```

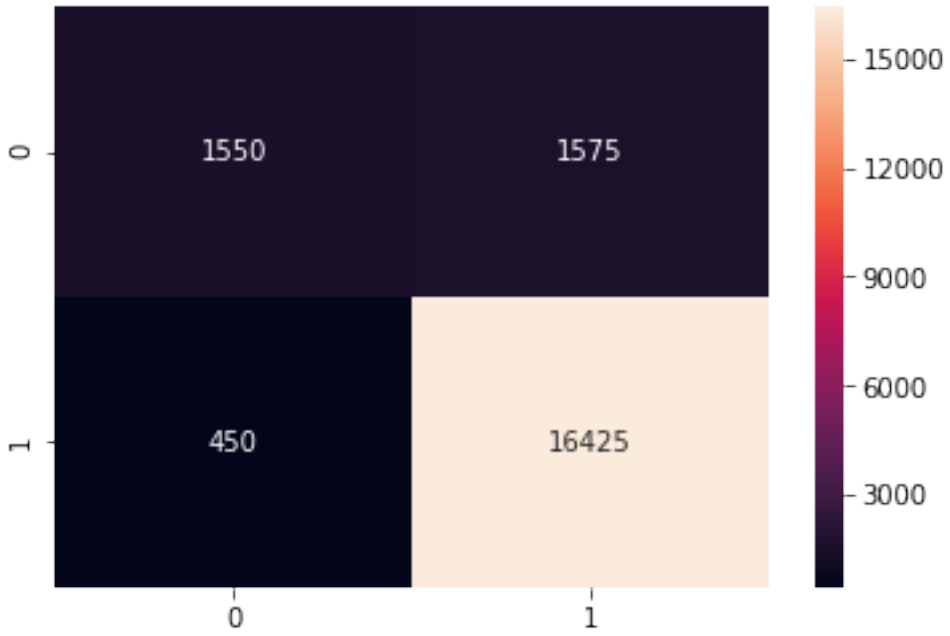
[201]:



[18]:
```
classifier = RandomForestClassifier(n_estimators=200,max_depth=20,␣
 ↪class_weight='balanced',bootstrap=True)
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_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()
```



[203]:
```
classifier = RandomForestClassifier(n_estimators=200,max_depth=20,␣
 ↪bootstrap=True, 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␣
 ↪n_estimators = %d is %f%%' % (20,200, acc))
```



The accuracy of the DecisionTreeClassifier for depth = 20 and n_estimators = 200 is 89.875000%

[208]:
```
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      0.53      0.63      3125
           1       0.92      0.98      0.95     16875

   micro avg       0.90      0.90      0.90     20000
   macro avg       0.86      0.75      0.79     20000
weighted avg       0.90      0.90      0.90     20000
```

[207]:
```
coef = classifier.feature_importances_
class_labels = model.classes_
feature_names =tfidf_vec.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)
```

```
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 aaaaaaaaaaaaaaa
0 0.0 aaaaaaaaagghh
0 0.0 aaaaaaabr
0 0.0 aaaaah
0 0.0 aaaaahhhhhhhhhhhhhhhthe
0 0.0 aaaah
0 0.0 aaah
0 0.0 aaahhhhhh
0 0.0 aaahs
0 0.0 aachen
0 0.0 aacurate
0 0.0 aad
0 0.0 aadp
0 0.0 aaf
0 0.0 aafco
0 0.0 aafter
0 0.0 aaghbr
----------------------------------------
Important words in positive reviews
1 0.028298974725658965 great
1 0.026023852193331747 love
1 0.011971531149919406 delicious
1 0.011113770649705163 best
1 0.010498264644342304 perfect
1 0.009632776836157026 highly
1 0.009506993908913932 would
1 0.008969033439677764 waste
1 0.008873008024555564 favorite
1 0.008703546510161897 money
1 0.008441521205061099 thought
1 0.00842245105822927 terrible
1 0.008069391521763603 easy
1 0.008014864586896544 product
1 0.007844958936829127 worst
1 0.00783701438978076 wonderful
1 0.007599996434595793 bad
1 0.006904125003644765 good
1 0.006809918540410611 always
```

1 0.006580332073711011 return

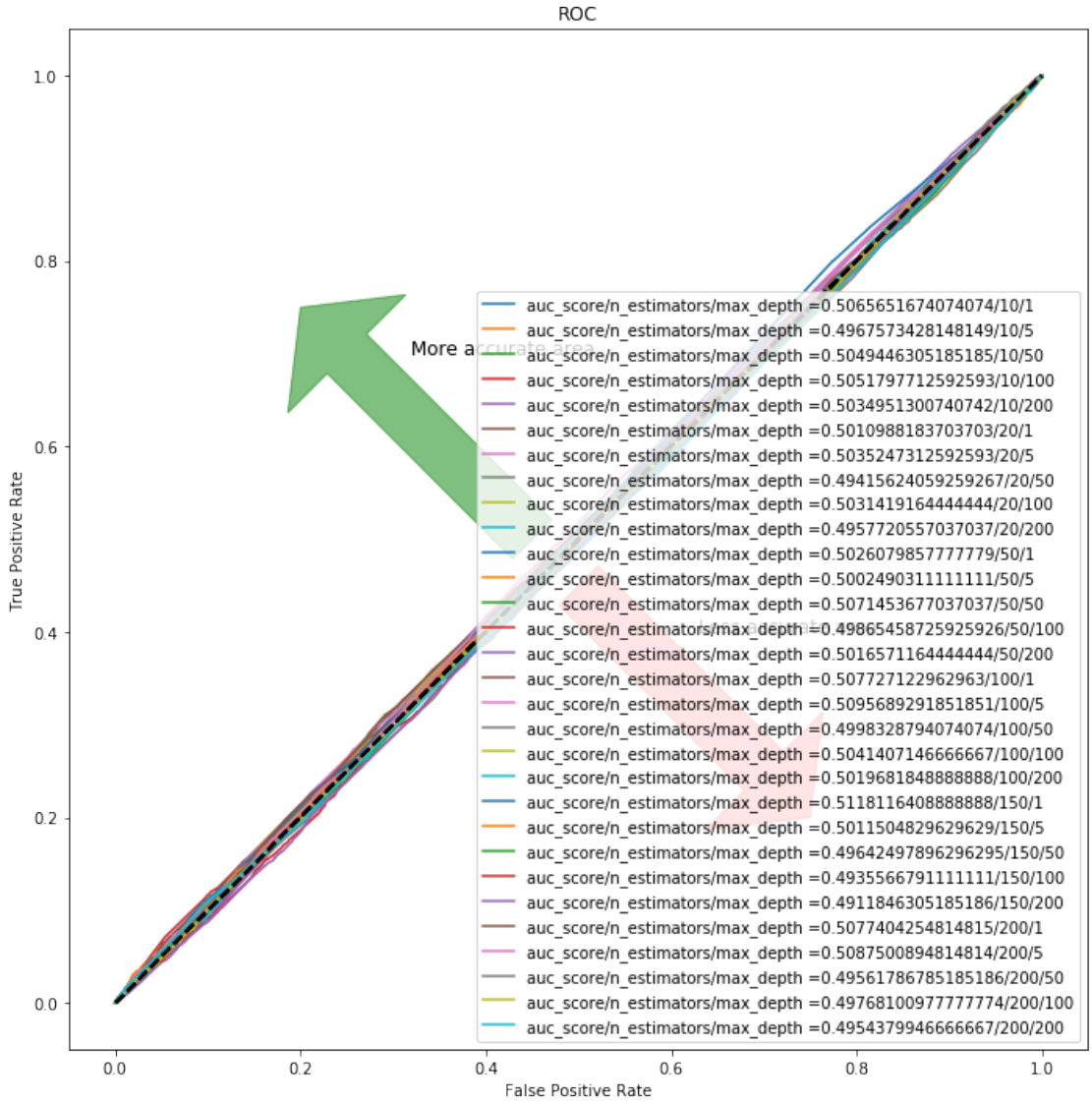
Word2Vec as vectorizer

[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()
```

[29]:
```
y_train = y[0:100000]
y_test= y[100000:120000]
```

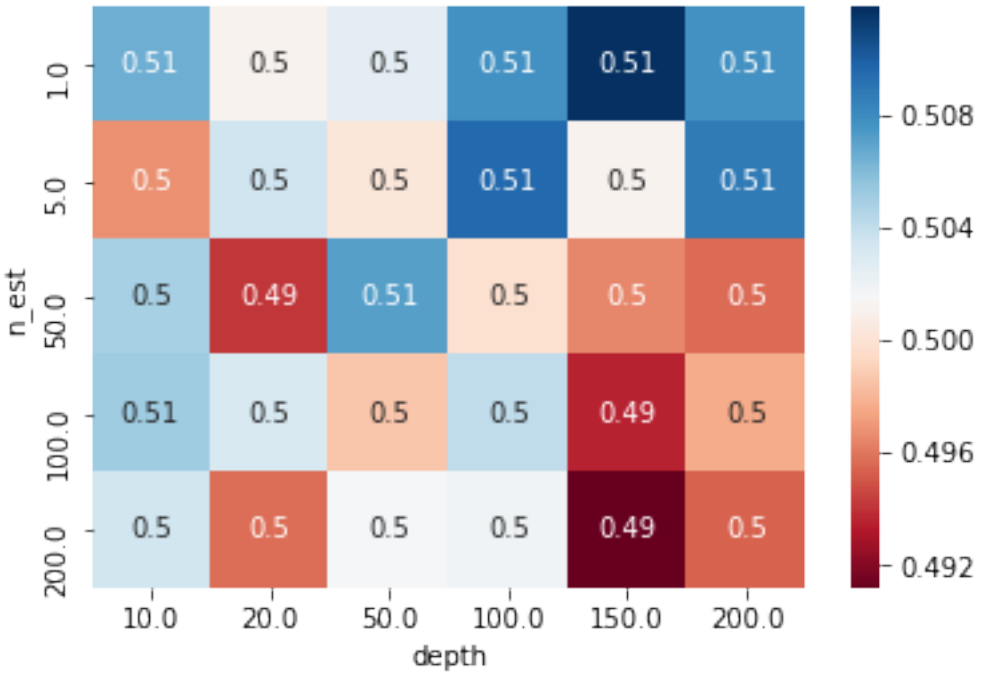
[214]:
```
depth = [1,5,50,100,200]
n_est = [10, 20, 50, 100,150, 200]
auc_lis=[]
depth_lis = []
n_est_lis = []
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    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 n_est:
    for j in depth:
        classifier =␣
 ↪RandomForestClassifier(n_estimators=i,max_depth=j,class_weight='balanced',n_jobs=-1,␣
 ↪bootstrap=True)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        n_est_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/n_estimators/max_depth ="+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()
```



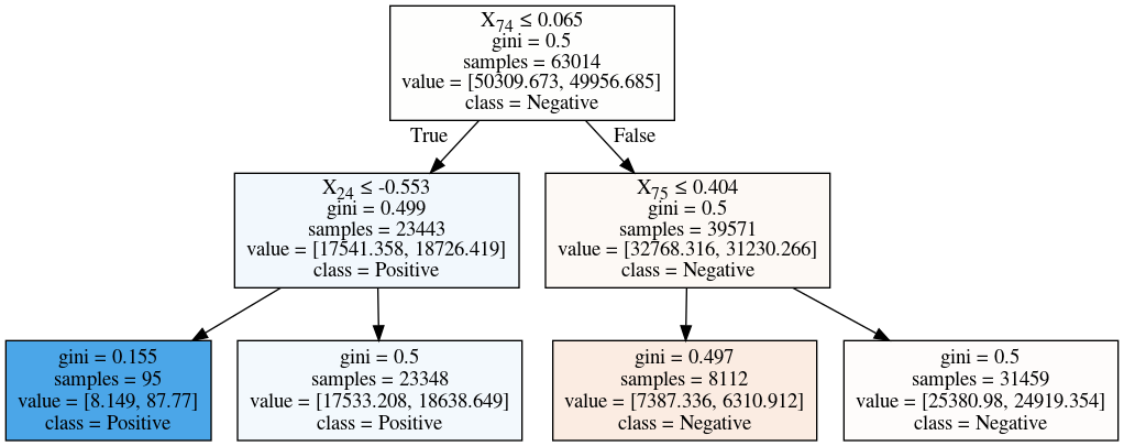
[215]:
```
X_avg_w2v = np.array(n_est_lis)
Y_avg_w2v = np.array(depth_lis)
Z_avg_w2v = np.array(auc_lis)
```

```
df_avg_w2v = pd.DataFrame.from_dict(np.array([X_avg_w2v,Y_avg_w2v,Z_avg_w2v]).T)
df_avg_w2v.columns = ['n_est','depth','auc']
df_avg_w2v['auc'] = pd.to_numeric(df_avg_w2v['auc'])
pivotted= df_avg_w2v.pivot('n_est','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```

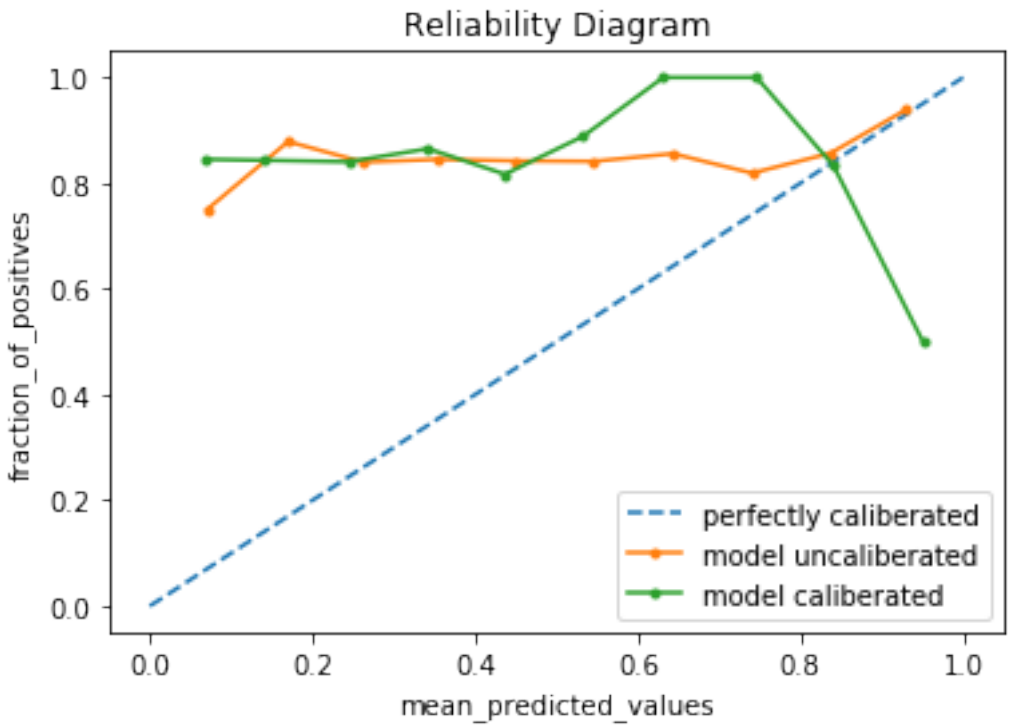


[217]:
```
classifier =␣
↪RandomForestClassifier(max_depth=2,class_weight='balanced',n_estimators=200)
classifier.fit(X_train_avg_w2v,y_train)
est = classifier.estimators_[5]
target = ['Negative','Positive']
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(est,out_file=None, class_names=target , filled =␣
↪True,special_characters=True)
# Draw graph
graph = pydotplus.graph_from_dot_data(graph)
# Show graph
Image(graph.create_png())
```

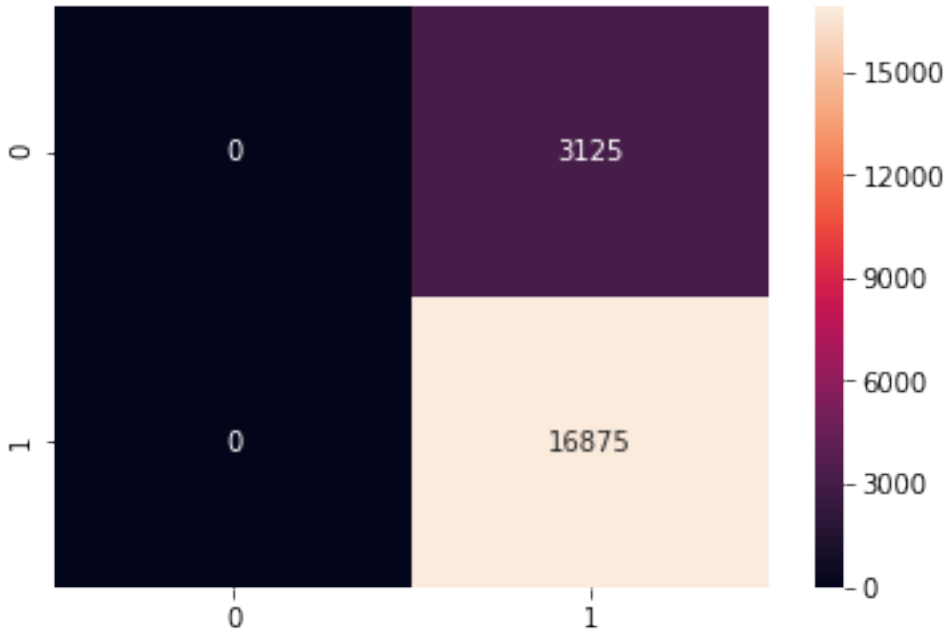
[217]:



[218]:
```
classifier = RandomForestClassifier(n_estimators=200,max_depth=20,␣
↪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()
```



[13]:
```
classifier = RandomForestClassifier(n_estimators=1,max_depth=10,␣
 ↪bootstrap=True, class_weight='balanced')
classifier.fit(X_train_avg_w2v,y_train)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_avg_w2v,y_train)
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 DecisionTreeClassifier for depth = %d and␣
 ↪n_estimators = %d is %f%%' % (1,10, acc))
```



The accuracy of the DecisionTreeClassifier for depth = 1 and n_estimators = 10 is 84.375000%

[219]:
```
y_pred = model.predict(X_test_avg_w2v)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

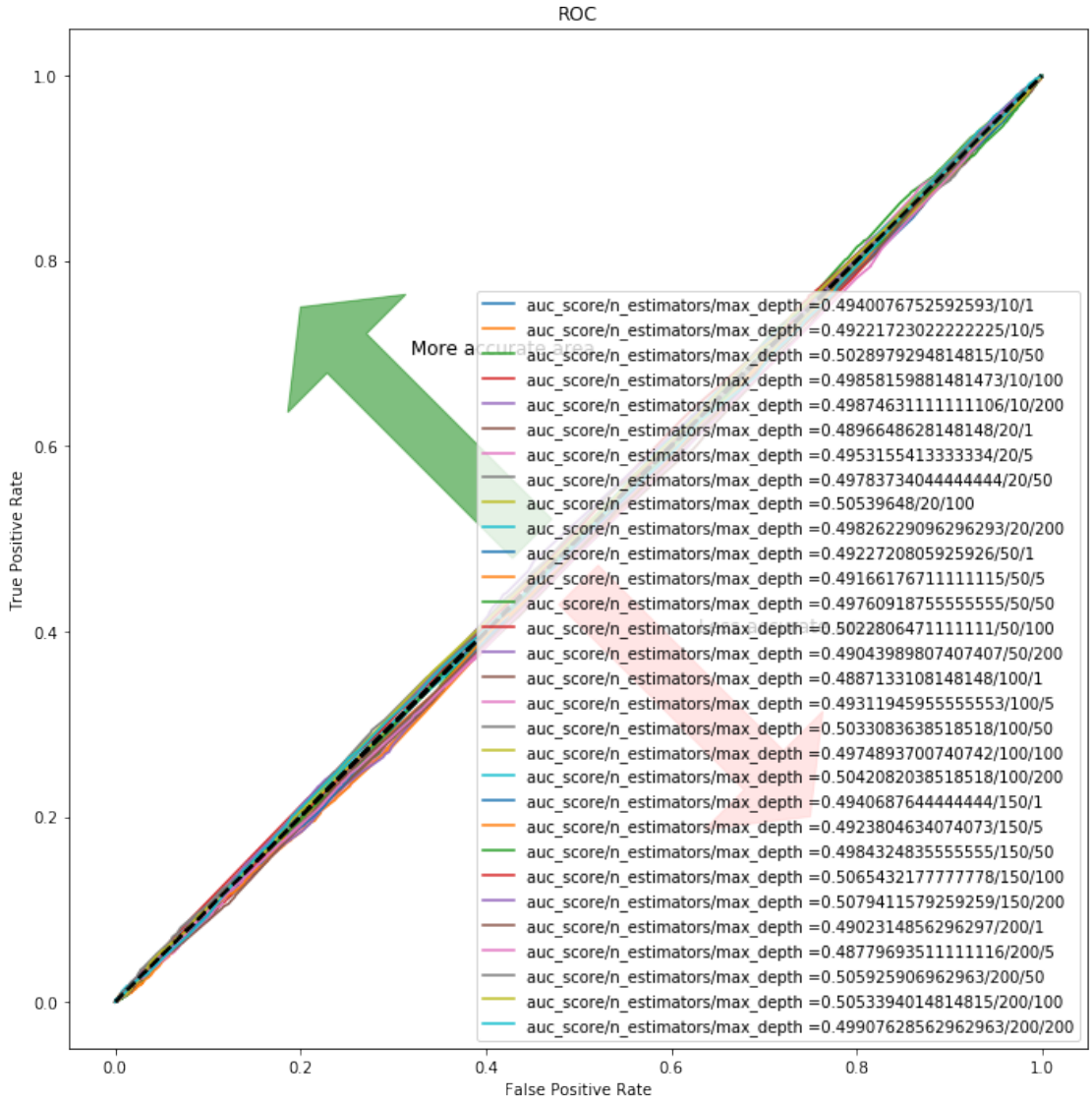
```
              precision    recall  f1-score   support

           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

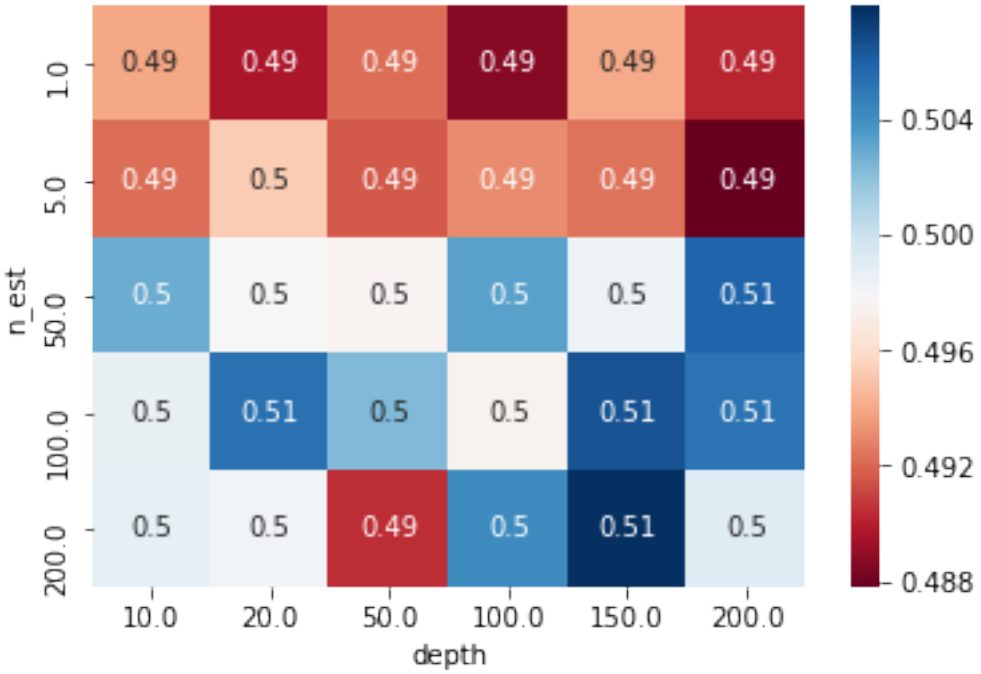
[23]:
```
import pickle
pickle_out1 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
 ↪tfidf_train_vectors","rb")
pickle_out2 = open("/home/niranjan/Downloads/UBUNTU 18_1/AppliedAI/
 ↪tfidf_test_vectors","rb")
tfidf_train_vectors = pickle.load(pickle_out1)
```

```
tfidf_test_vectors = pickle.load(pickle_out2)
pickle_out1.close()
pickle_out2.close()
```

[221]:
```
depth = [1,5,50,100,200]
n_est = [10, 20, 50, 100,150, 200]
auc_lis=[]
depth_lis = []
n_est_lis = []
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 n_est:
    for j in depth:
        classifier =␣
 ↪RandomForestClassifier(n_estimators=i,max_depth=j,class_weight='balanced',n_jobs=-1,␣
 ↪bootstrap=True)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        n_est_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/n_estimators/max_depth ="+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()
```

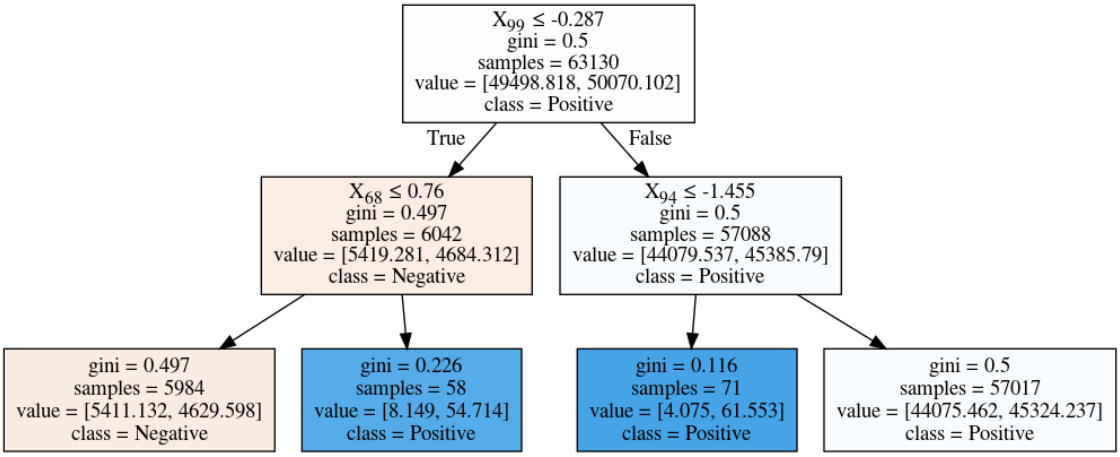


[222]:
```
X_tfidf_w2v = np.array(n_est_lis)
Y_tfidf_w2v = np.array(depth_lis)
Z_tfidf_w2v = np.array(auc_lis)

df_tfidf_w2v = pd.DataFrame.from_dict(np.
 ↪array([X_tfidf_w2v,Y_tfidf_w2v,Z_tfidf_w2v]).T)
df_tfidf_w2v.columns = ['n_est','depth','auc']
df_tfidf_w2v['auc'] = pd.to_numeric(df_tfidf_w2v['auc'])
pivotted= df_tfidf_w2v.pivot('n_est','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```

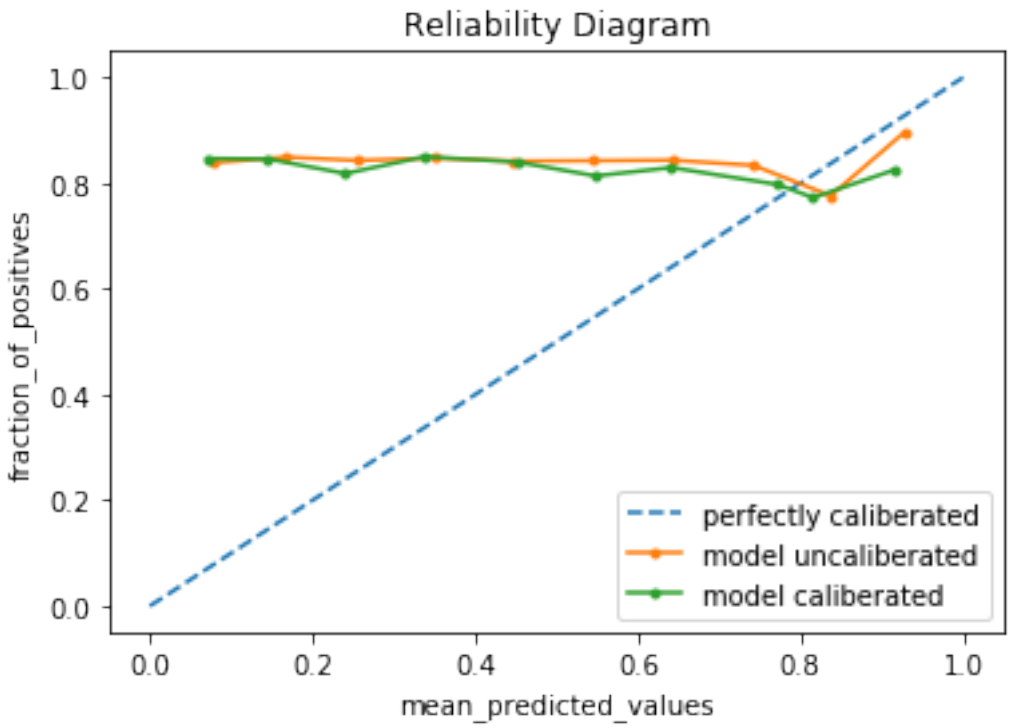


[223]:
```
classifier =␣
↪RandomForestClassifier(max_depth=2,class_weight='balanced',n_estimators=100)
classifier.fit(tfidf_train_vectors,y_train)
est = classifier.estimators_[5]
target = ['Negative','Positive']
from sklearn.tree import export_graphviz
graph = tree.export_graphviz(est,out_file=None, class_names=target, filled =␣
↪True,special_characters=True)
# Draw graph
graph = pydotplus.graph_from_dot_data(graph)
# Show graph
Image(graph.create_png())
```

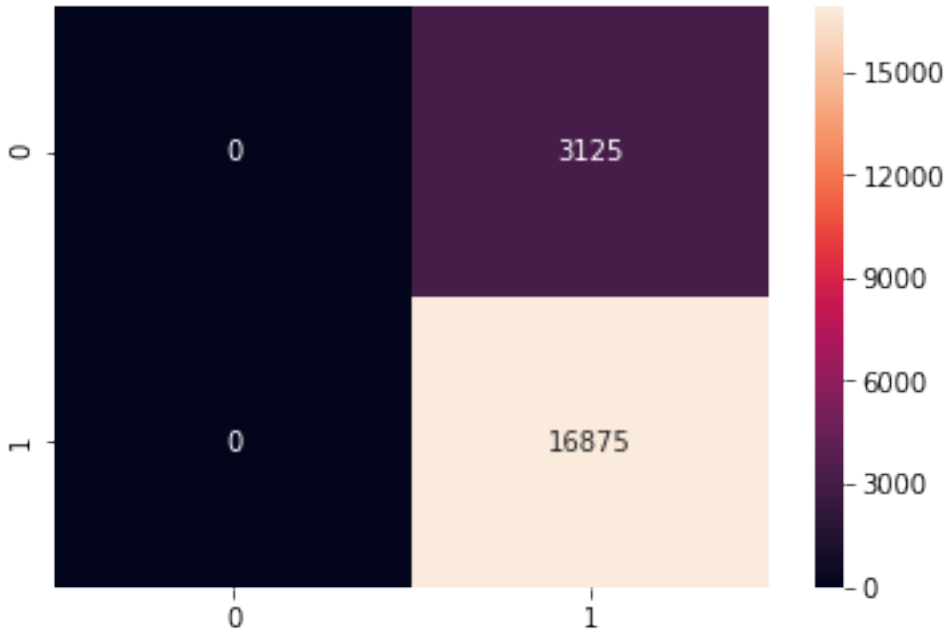
[223]:



[235]:
```
classifier = RandomForestClassifier(max_depth=20,␣
 ↪class_weight='balanced',n_estimators=100)
classifier.fit(tfidf_train_vectors,y_train)
#coef = classifier.coef_
probs = classifier.predict_proba(tfidf_test_vectors)[:,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()
```



[15]:
```
classifier = RandomForestClassifier(max_depth=20, n_estimators=100,␣
 ↪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␣
 ↪n_estimators = %d is %f%%' % (20,100, acc))
```



The accuracy of the DecisionTreeClassifier for depth = 20 and n_estimators = 100 is 84.375000%

[225]:
```
y_pred = model.predict(tfidf_test_vectors)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

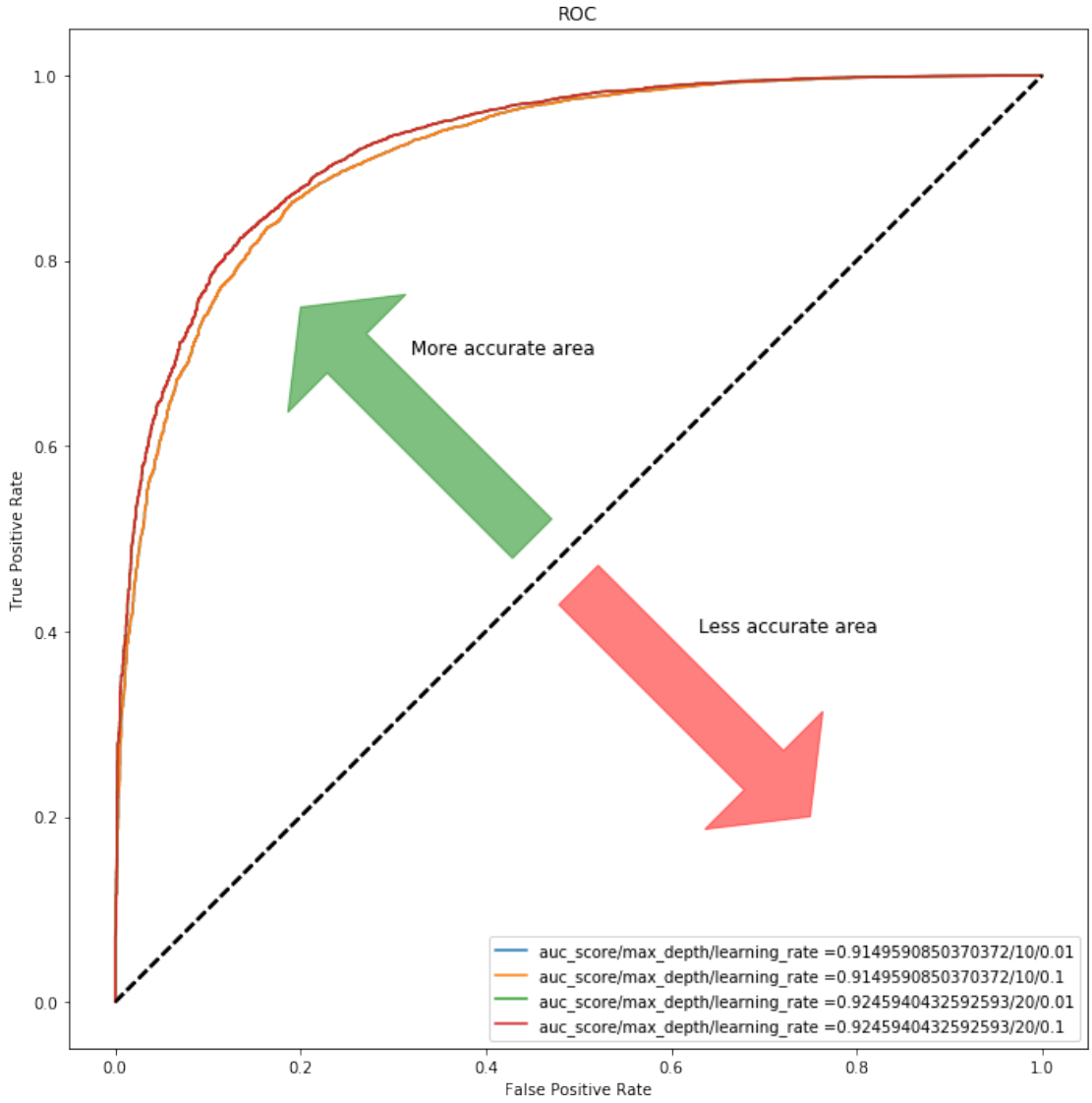
```
              precision    recall  f1-score   support

           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
```

XGBoost

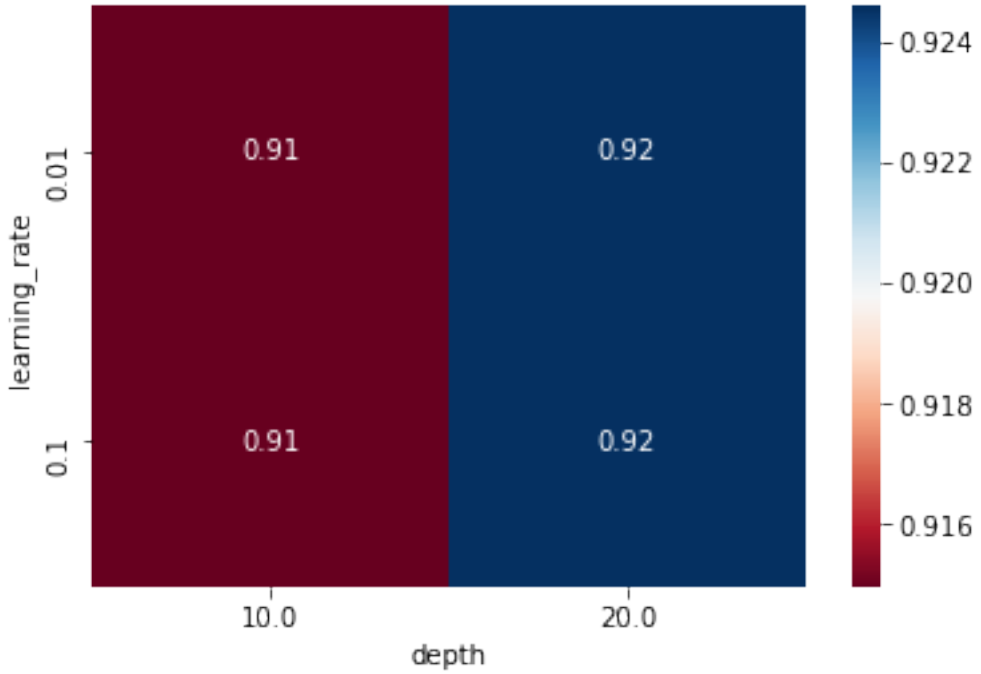
[49]:
```
import xgboost as xgb
```

[23]:
```
depth = [10,20]
learn_rate = [.01,.1]
auc_lis=[]
learn_rate_lis = []
```

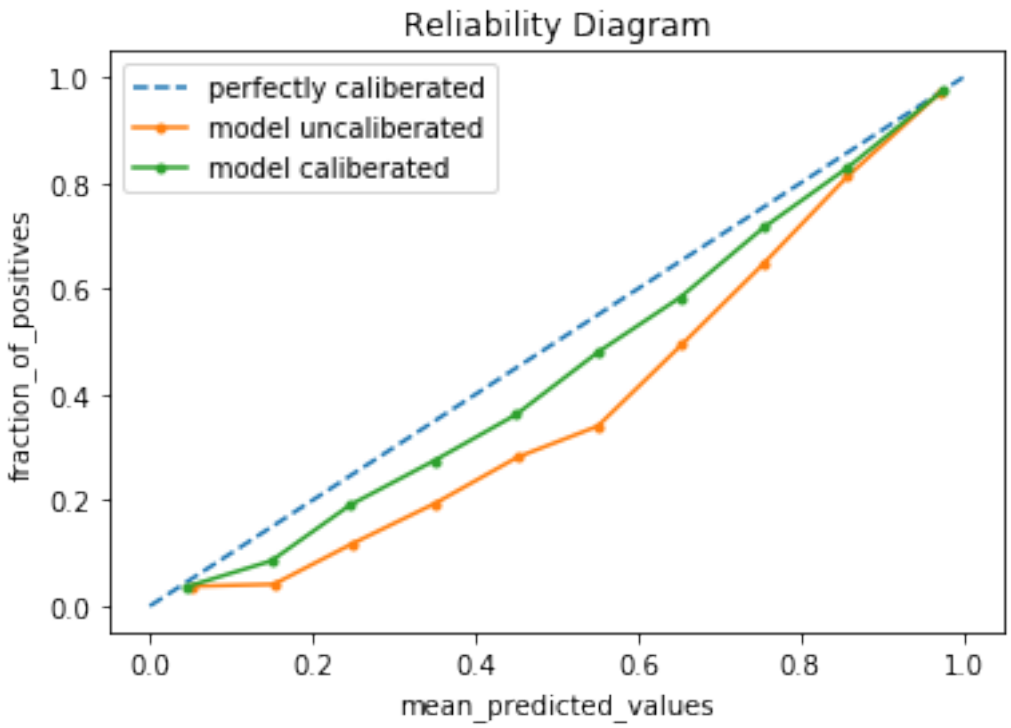
```
depth_lis = []
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

mean_fpr = np.linspace(0,1,100)
for i in depth:
    for j in learn_rate:
        classifier = xgb.XGBClassifier(booster='gbtree',max_depth=i,eta=j)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        learn_rate_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/max_depth/learning_rate ="+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()
```



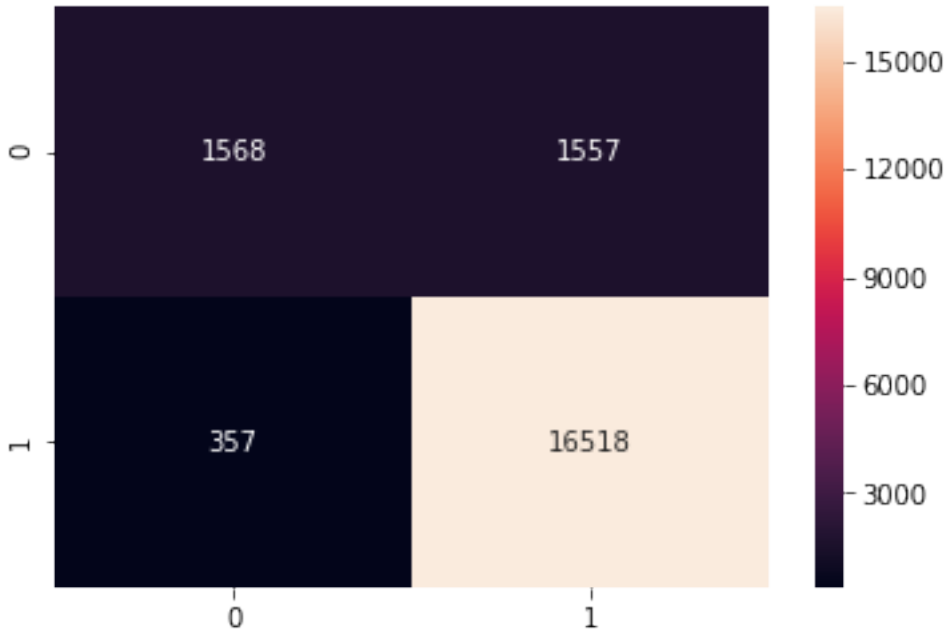
[25]:
```
X_bow = np.array(learn_rate_lis)
Y_bow = np.array(depth_lis)
Z_bow = np.array(auc_lis)

df_bow = pd.DataFrame.from_dict(np.array([X_bow,Y_bow,Z_bow]).T)
df_bow.columns = ['learning_rate','depth','auc']
df_bow['auc'] = pd.to_numeric(df_bow['auc'])
pivotted= df_bow.pivot('learning_rate','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```



[50]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=20,eta=.01)
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()
```



[35]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=20,eta=.01)
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 learning␣
 ↪rate = %d is %f%%' % (20,.01, acc))
```




```
The accuracy of the DecisionTreeClassifier for depth = 20 and learning rate = 0
is 90.430000%
```

[36]:
```
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       0.81      0.50      0.62      3125
           1       0.91      0.98      0.95     16875

   micro avg       0.90      0.90      0.90     20000
   macro avg       0.86      0.74      0.78     20000
weighted avg       0.90      0.90      0.89     20000
```

[54]:
```
importance = classifier.feature_importances_
class_labels = model.classes_
feature_names =count_vect.get_feature_names()
topn_class1_xgb = sorted(zip(importance, feature_names),reverse=False)[:20]
topn_class2_xgb = sorted(zip(importance, feature_names),reverse=True)[:20]
print("Important words in negative reviews")
for importanc, feat in topn_class1_xgb:
    print(class_labels[0], importanc, feat)
```

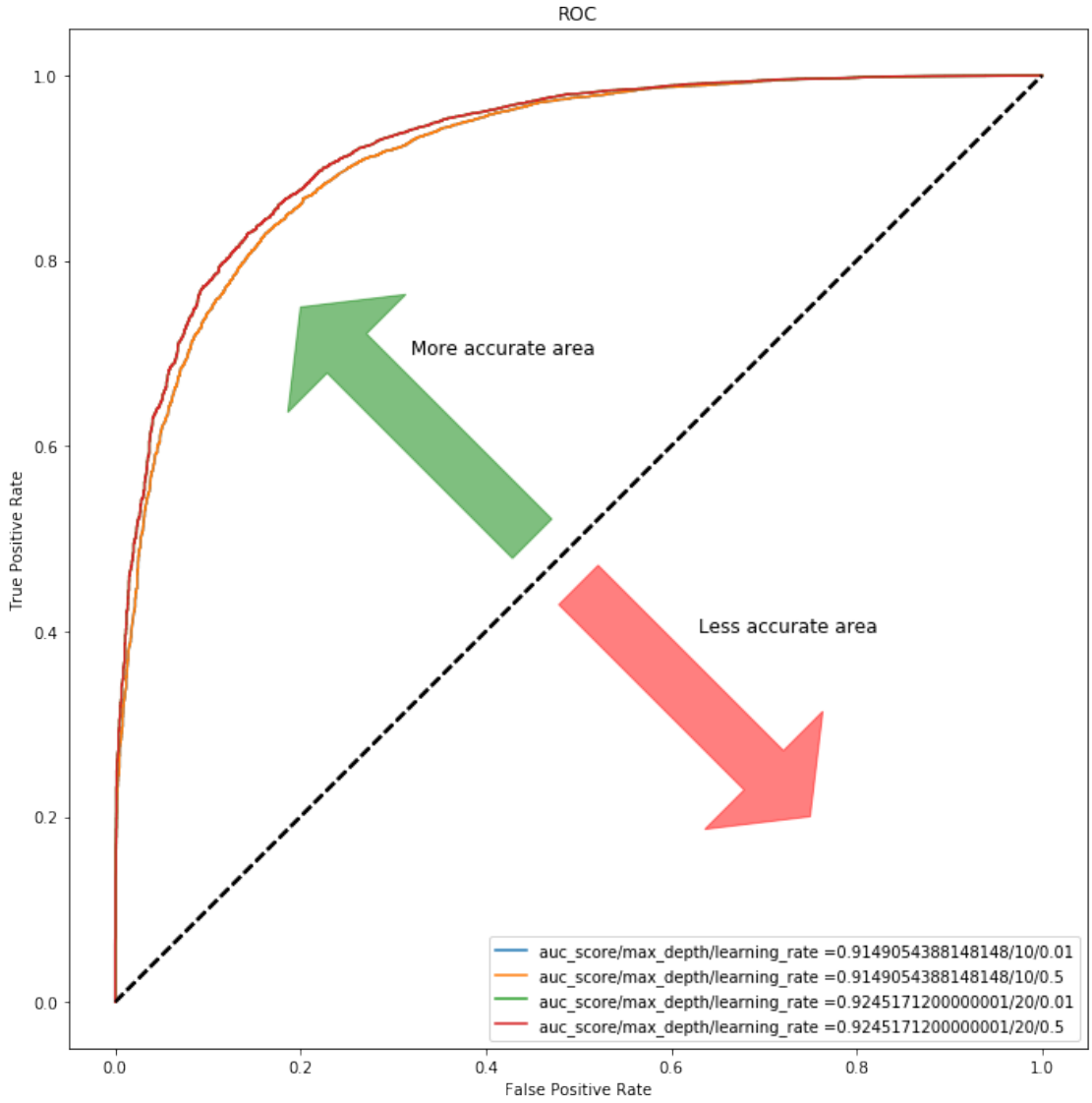
```
print("----------------------------------------")
print("Important words in positive reviews")
for importanc, feat in topn_class2_xgb:
    print(class_labels[1], importanc, feat)
```

```
Important words in negative reviews
0 0.0 aa
0 0.0 aaa
0 0.0 aaaa
0 0.0 aaaaa
0 0.0 aaaaaaaaaaaaaaa
0 0.0 aaaaaaaaagghh
0 0.0 aaaaaabr
0 0.0 aaaaah
0 0.0 aaaaahhhhhhhhhhhhhhhthe
0 0.0 aaaah
0 0.0 aaah
0 0.0 aaahhhhhh
0 0.0 aaahs
0 0.0 aachen
0 0.0 aacurate
0 0.0 aad
0 0.0 aadp
0 0.0 aaf
0 0.0 aafco
0 0.0 aafter
----------------------------------------
Important words in positive reviews
1 0.006829901 delicious
1 0.0059515843 worst
1 0.0051996135 best
1 0.0049646627 great
1 0.0049589006 perfect
1 0.0045655733 highly
1 0.0043366887 threw
1 0.0042989836 waste
1 0.004289796 excellent
1 0.0040165526 easy
1 0.003805688 yummy
1 0.0038038422 favorite
1 0.0035729995 awesome
1 0.0035460214 gross
1 0.0034753508 horrible
1 0.003406402 wonderful
1 0.0032368226 terrible
1 0.0031637086 hooked
1 0.003051933 money
```

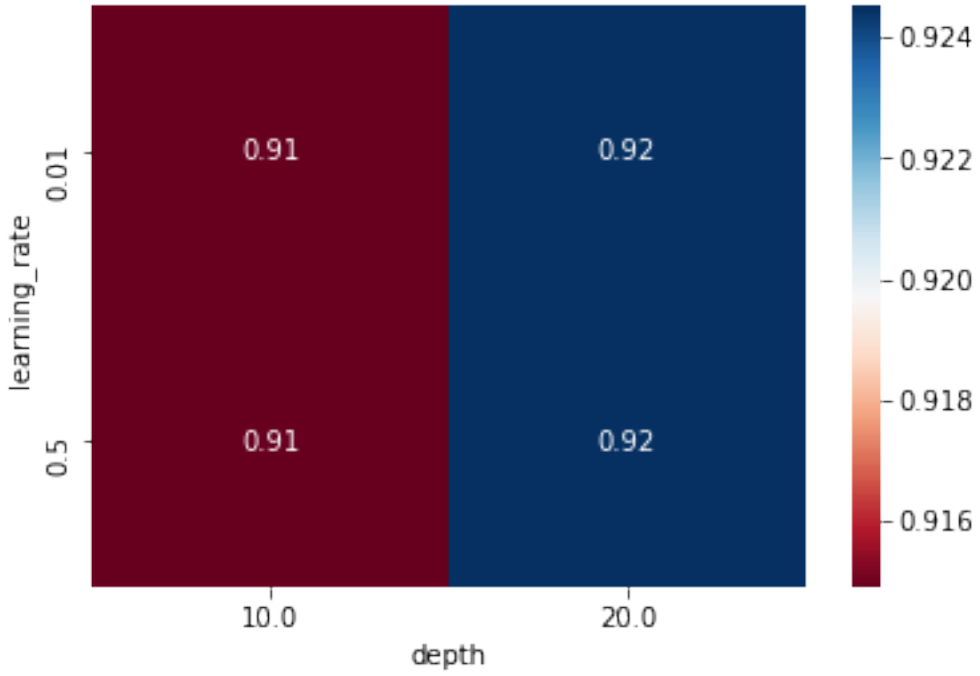
1 0.0030434837 yuck

tfidf as a vectorizer

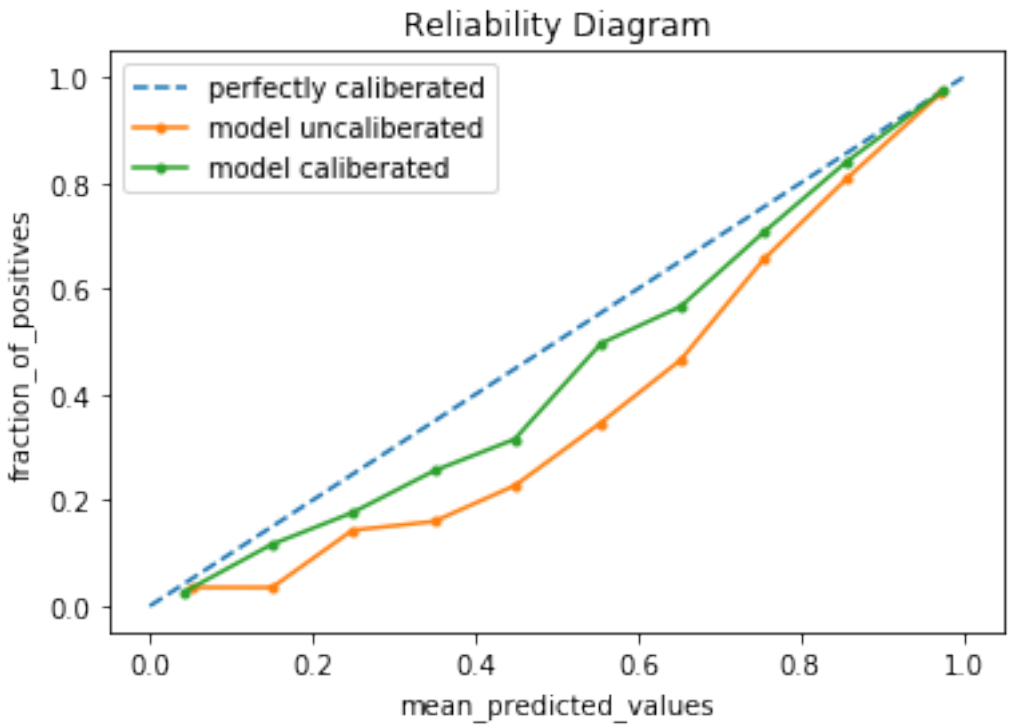
[39]:
```
depth = [10,20]
learn_rate = [.01,.5]
auc_lis=[]
learn_rate_lis = []
depth_lis = []
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

mean_fpr = np.linspace(0,1,100)
for i in depth:
    for j in learn_rate:
        classifier = xgb.XGBClassifier(booster='gbtree',max_depth=i,eta=j)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        learn_rate_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/max_depth/learning_rate ="+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()
```



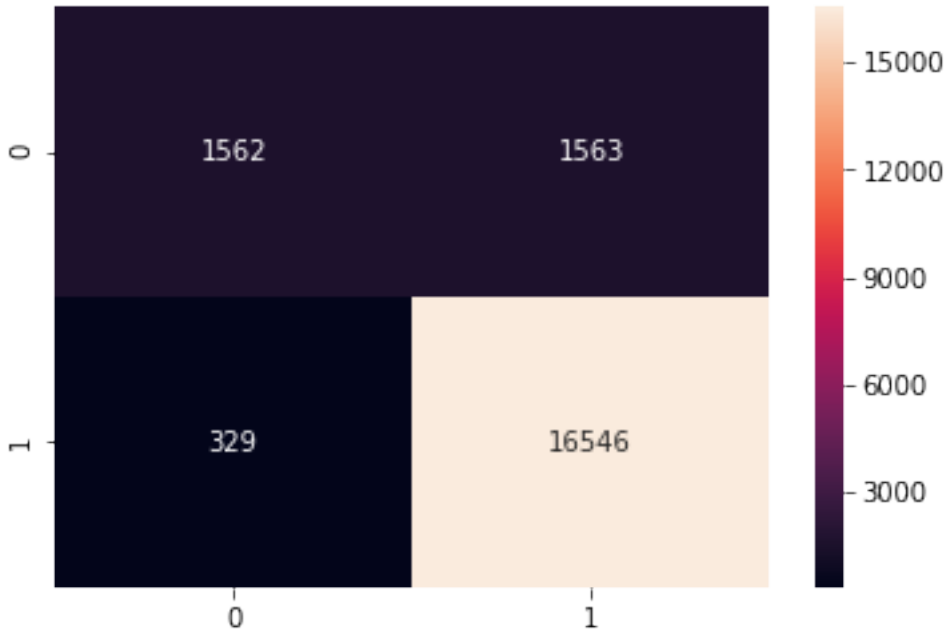
[40]:
```
X_tfidf = np.array(learn_rate_lis)
Y_tfidf = np.array(depth_lis)
Z_tfidf = np.array(auc_lis)

df_tfidf = pd.DataFrame.from_dict(np.array([X_tfidf,Y_tfidf,Z_tfidf]).T)
df_tfidf.columns = ['learning_rate','depth','auc']
df_tfidf['auc'] = pd.to_numeric(df_tfidf['auc'])
pivotted= df_tfidf.pivot('learning_rate','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```



[41]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=20,eta=.01)
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_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()
```



[19]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=20,eta=.01)
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 learning␣
 ↪rate = %d is %f%%' % (20,.01, acc))
```




```
The accuracy of the DecisionTreeClassifier for depth = 20 and learning rate = 0
is 90.540000%
```

[42]:
```
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       0.83      0.50      0.62      3125
           1       0.91      0.98      0.95     16875

   micro avg       0.91      0.91      0.91     20000
   macro avg       0.87      0.74      0.78     20000
weighted avg       0.90      0.91      0.90     20000
```

[43]:
```
coef = classifier.feature_importances_
class_labels = model.classes_
feature_names =tfidf_vec.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)
```

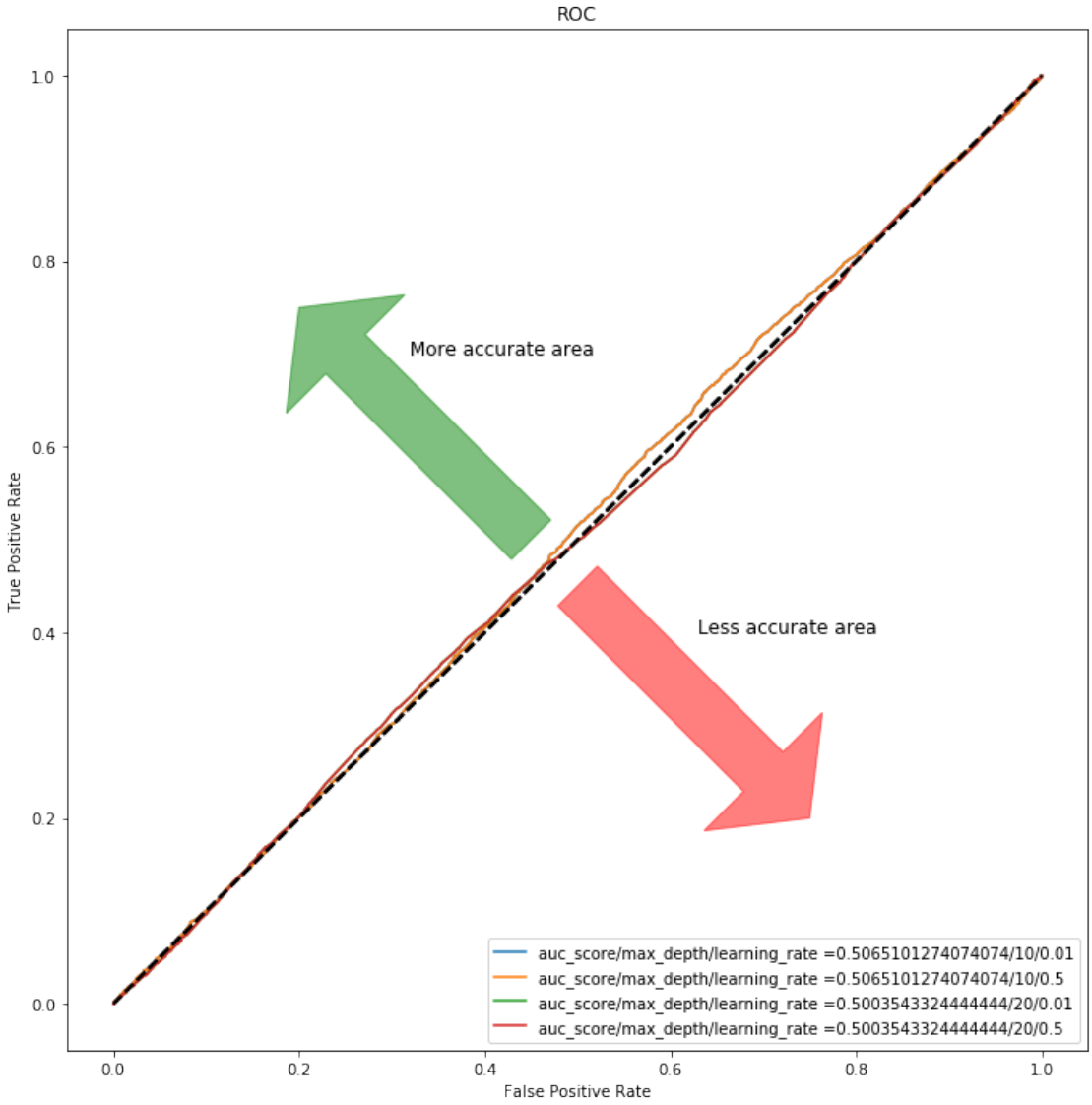
```python
print("----------------------------------------")
print("Important words in positive reviews")
for coef, feat in topn_class2:
    print(class_labels[1], coef, feat)
```

```
Important words in negative reviews
0 0.0 aa
0 0.0 aaa
0 0.0 aaaa
0 0.0 aaaaa
0 0.0 aaaaaaaaaaaaaaa
0 0.0 aaaaaaaaagghh
0 0.0 aaaaaabr
0 0.0 aaaaah
0 0.0 aaaaahhhhhhhhhhhhhhhthe
0 0.0 aaaah
0 0.0 aaah
0 0.0 aaahhhhhh
0 0.0 aaahs
0 0.0 aachen
0 0.0 aacurate
0 0.0 aad
0 0.0 aadp
0 0.0 aaf
0 0.0 aafco
0 0.0 aafter
----------------------------------------
Important words in positive reviews
1 0.0068835136 worst
1 0.0046489057 waste
1 0.0044691307 threw
1 0.004339722 gross
1 0.0037219487 fell
1 0.0036671592 perfect
1 0.003599539 return
1 0.0035968006 excellent
1 0.0035444552 hooked
1 0.0034700911 yuck
1 0.0034109915 highly
1 0.0033880905 horrible
1 0.0033838314 delicious
1 0.003365124 ripoff
1 0.003334188 description
1 0.003185331 amazing
1 0.003145375 email
1 0.0031071145 terrible
1 0.0029951467 fantastic
```

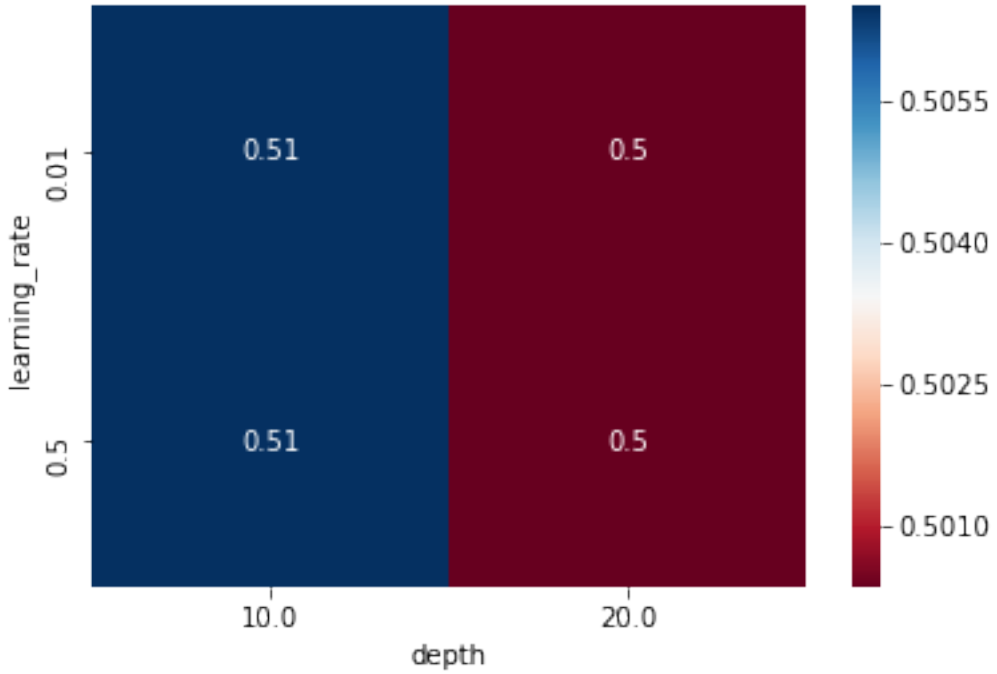
1 0.0028791972 pleasantly

word2vec as vectorizer

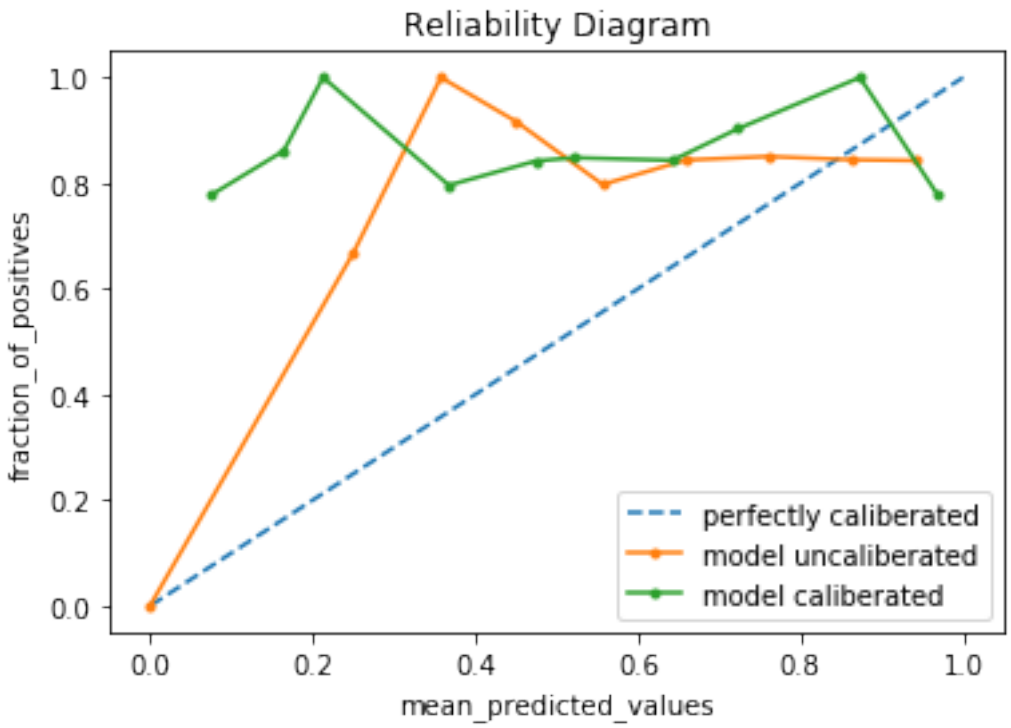
[47]:
```
depth = [10,20]
learn_rate = [.01,.5]
auc_lis=[]
learn_rate_lis = []
depth_lis = []
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    patches.Arrow(0.5,0.45,0.25,-0.25,width=0.3,color='red',alpha = 0.5)
    )

mean_fpr = np.linspace(0,1,100)
for i in depth:
    for j in learn_rate:
        classifier = xgb.XGBClassifier(booster='gbtree',max_depth=i,eta=j)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        learn_rate_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/max_depth/learning_rate ="+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()
```



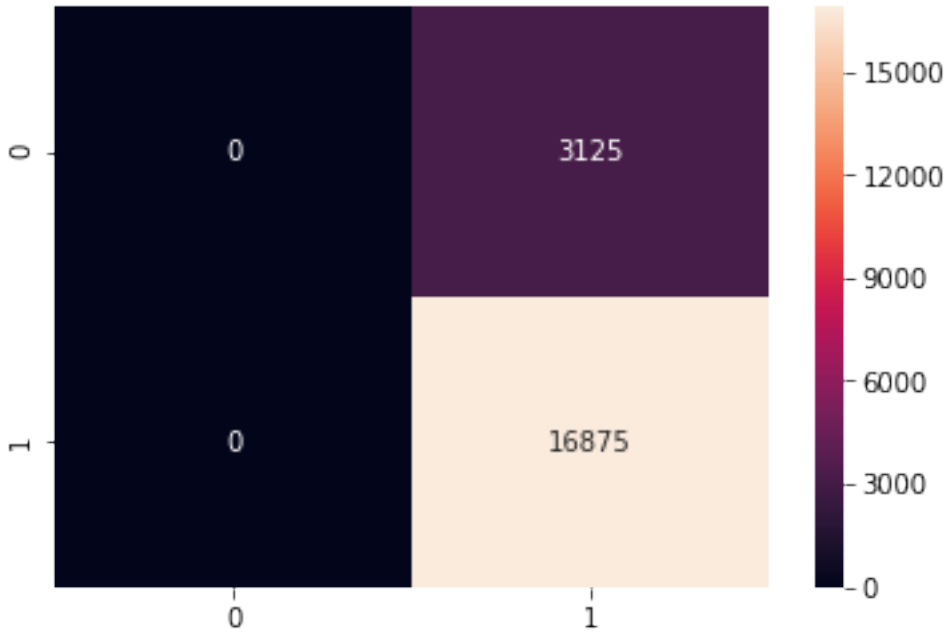
[48]:
```
X_avg_w2v = np.array(learn_rate_lis)
Y_avg_w2v = np.array(depth_lis)
Z_avg_w2v = np.array(auc_lis)

df_avg_w2v = pd.DataFrame.from_dict(np.array([X_avg_w2v,Y_avg_w2v,Z_avg_w2v]).T)
df_avg_w2v.columns = ['learning_rate','depth','auc']
df_avg_w2v['auc'] = pd.to_numeric(df_avg_w2v['auc'])
pivotted= df_avg_w2v.pivot('learning_rate','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```



[50]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=20,eta=.01)
classifier.fit(np.array(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()
```



[31]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=10,eta=.01)
classifier.fit(np.array(X_train_avg_w2v),y_train)
model = CalibratedClassifierCV(classifier,cv=5,method ='isotonic')
model.fit(X_train_avg_w2v,y_train)
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 DecisionTreeClassifier for depth = %d and learning␣
 ↪rate = %d is %f%%' % (10,.01, acc))
```



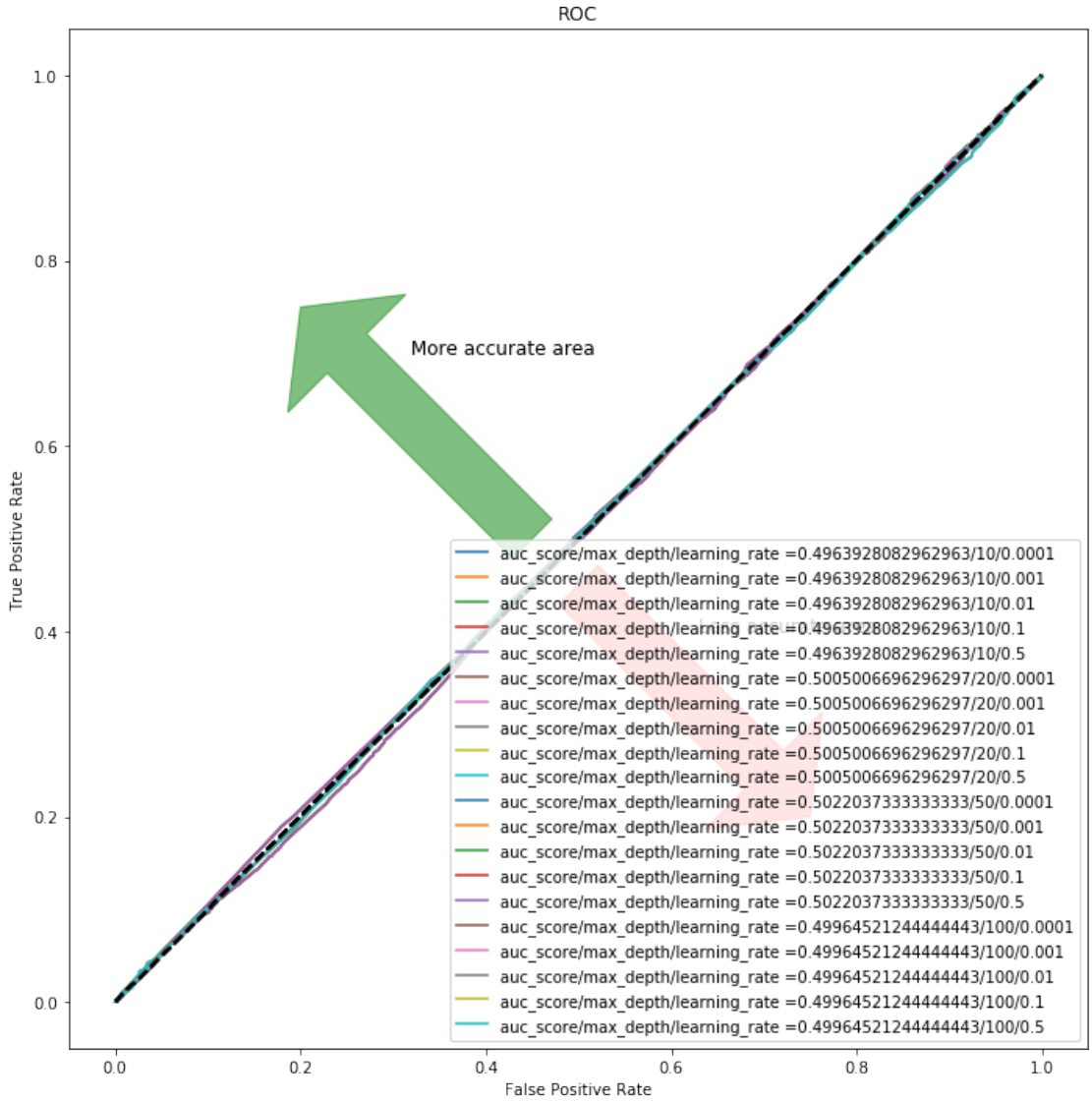

```
The accuracy of the DecisionTreeClassifier for depth = 10 and learning rate = 0
is 84.375000%
```

[32]:
```
y_pred = model.predict(X_test_avg_w2v)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

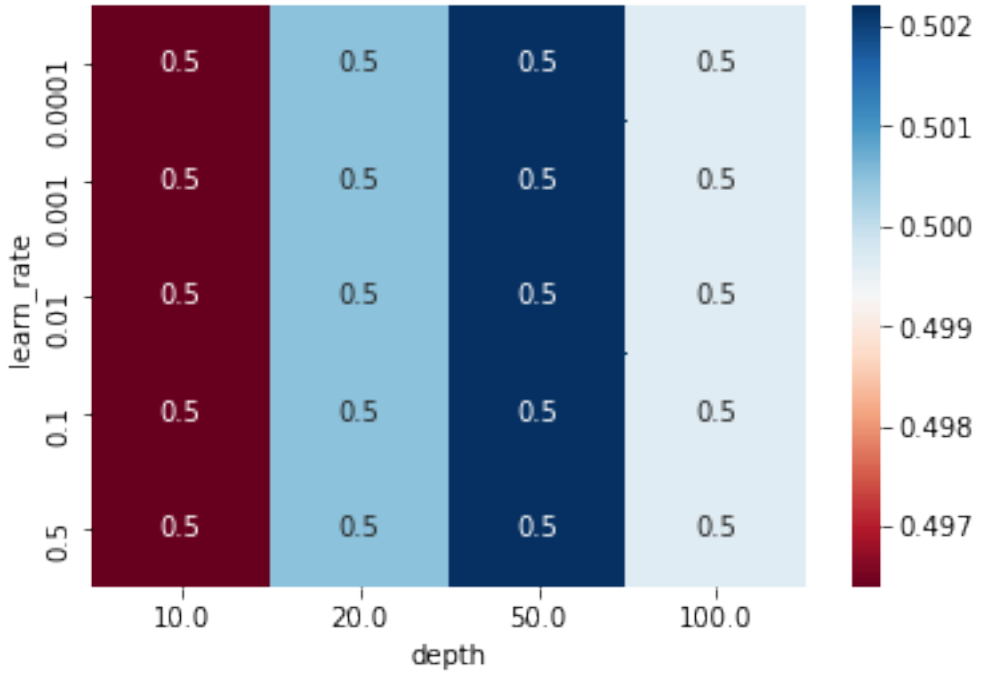
```
/home/niranjan/anaconda3/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
/home/niranjan/anaconda3/lib/python3.6/site-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

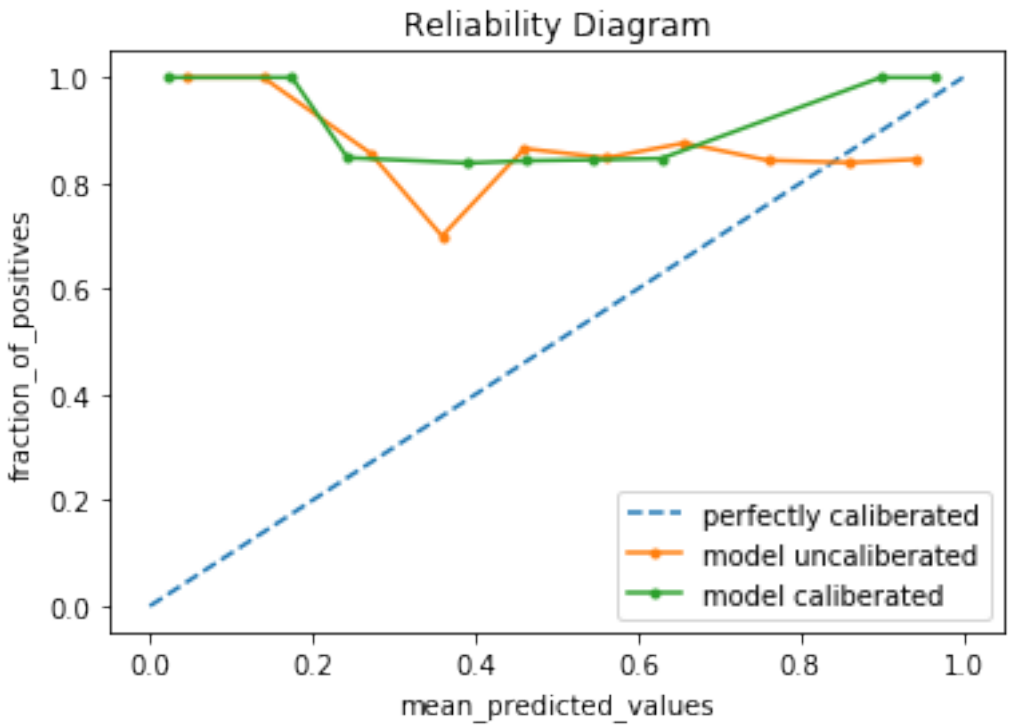
[232]:
```
depth = [10,20,50,100]
learn_rate = [0.0001,.001,.01,.1,.5]
auc_lis=[]
learn_rate_lis = []
depth_lis = []
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
ax1.add_patch(
    patches.Arrow(0.45,0.5,-0.25,0.25,width=0.3,color='green',alpha = 0.5)
    )
ax1.add_patch(
    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 learn_rate:
        classifier = xgb.XGBClassifier(booster='gbtree',max_depth=i,eta=j)
        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)
        auc_lis.append(auc)
        depth_lis.append(i)
        learn_rate_lis.append(j)
        plt.plot(fpr,tpr,label="auc_score/max_depth/learning_rate ="+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()
```



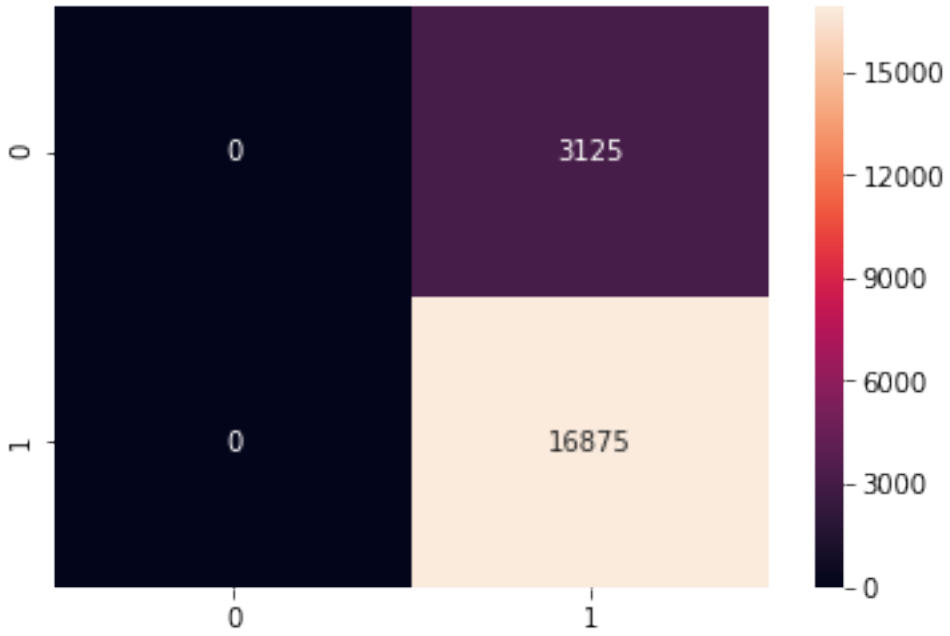
[234]:
```
X_tfidf_w2v = np.array(learn_rate_lis)
Y_tfidf_w2v = np.array(depth_lis)
Z_tfidf_w2v = np.array(auc_lis)

df_tfidf_w2v = pd.DataFrame.from_dict(np.
 ↪array([X_tfidf_w2v,Y_tfidf_w2v,Z_tfidf_w2v]).T)
df_tfidf_w2v.columns = ['learn_rate','depth','auc']
df_tfidf_w2v['auc'] = pd.to_numeric(df_tfidf_w2v['auc'])
pivotted= df_tfidf_w2v.pivot('learn_rate','depth','auc')
sns.heatmap(pivotted,cmap='RdBu',annot=True)
plt.show()
```



[254]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=50,eta=.01)
classifier.fit(np.array(tfidf_train_vectors),y_train)
# #coef = classifier.coef_
probs = classifier.predict_proba(tfidf_test_vectors)[:,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()
```



[33]:
```
classifier = xgb.XGBClassifier(booster='gbtree',max_depth=10,eta=.01)
classifier.fit(np.array(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 learning␣
 ↪rate = %d is %f%%' % (10,.01, acc))
```



The accuracy of the DecisionTreeClassifier for depth = 10 and learning rate = 0 is 84.375000%

[34]:
```
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)
```

[62]:
```
from prettytable import PrettyTable
x = PrettyTable()
y = PrettyTable()
z = PrettyTable()
k = PrettyTable()
v = PrettyTable()
u = PrettyTable()
v = PrettyTable()



x.add_column("important_features_class_[0] for␣
 ↪DecisionTreeClassifier",topn_class1_dtree)
x.add_column("important_features_class_[1] for␣
 ↪DecisionTreeClassifier",topn_class2_dtree)



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.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])



v.add_column("important_features_class_[0] for XGBClassifier",topn_class1_xgb)
v.add_column("important_features_class_[1] for XGBClassifier",topn_class2_xgb)

k.
 ↪field_names=["Vectorizer","Model","depth","learning_ratee","Auc_score","f1_score(micro␣
 ↪average)"]
```

```
k.add_row(["BOW","XGBClassifier","20",".01",".9245","0.90"])
k.add_row(["tfidf","XGBClassifier","20",".01",".9245","0.91"])
k.add_row(["Word2Vec","XGBClassifier","10",".01",".5065","0.84"])
k.add_row(["tfidf-Word2Vec","XGBClassifier","10",".01",".4963","0.84"])



u.add_row(["BOW","XGBClassifier",0.90,0.90])
u.add_row(["tfidf","XGBClassifier",0.91,0.91])
u.add_row(["Word2Vec","XGBClassifier",0.84,0.84])
u.add_row(["tfidf-Word2Vec","XGBClassifier",0.84,0.84])

print(x)
print(y)
print(z)
print("--------------------------------XGBClassifier-----------------------------------------

print(v)
print(k)
print(u)
```

```
+-----------------------------------------------------------+---------------------
-----------------------------------+
| important_features_class_[0] for DecisionTreeClassifier |
important_features_class_[1] for DecisionTreeClassifier |
+-----------------------------------------------------------+---------------------
-----------------------------------+
|                        (0.0, 'aa')                        |
(0.006829901, 'delicious')               |
|                        (0.0, 'aaa')                       |
(0.0059515843, 'worst')                  |
|                       (0.0, 'aaaa')                       |
(0.0051996135, 'best')                  |
|                       (0.0, 'aaaaa')                      |
(0.0049646627, 'great')                  |
|                  (0.0, 'aaaaaaaaaaaaaa')                  |
(0.0049589006, 'perfect')                |
|                    (0.0, 'aaaaaaaaagghh')                 |
(0.0045655733, 'highly')                 |
|                      (0.0, 'aaaaaabr')                    |
(0.0043366887, 'threw')                  |
|                       (0.0, 'aaaaah')                     |
(0.0042989836, 'waste')                  |
|            (0.0, 'aaaaahhhhhhhhhhhhhhhhthe')              |
(0.004289796, 'excellent')               |
|                       (0.0, 'aaaah')                      |
```

```
(0.0040165526, 'easy')                  |
|                      (0.0, 'aaah')                       |
(0.003805688, 'yummy')                  |
|                    (0.0, 'aaahhhhhh')                    |
(0.0038038422, 'favorite')               |
|                      (0.0, 'aaahs')                      |
(0.0035729995, 'awesome')                |
|                     (0.0, 'aachen')                      |
(0.0035460214, 'gross')                  |
|                    (0.0, 'aacurate')                     |
(0.0034753508, 'horrible')               |
|                       (0.0, 'aad')                       |
(0.003406402, 'wonderful')               |
|                      (0.0, 'aadp')                       |
(0.0032368226, 'terrible')               |
|                       (0.0, 'aaf')                       |
(0.0031637086, 'hooked')                 |
|                      (0.0, 'aafco')                      |
(0.003051933, 'money')                   |
|                     (0.0, 'aafter')                      |
(0.0030434837, 'yuck')                   |
+----------------------------------------------------------+----------------------
-----------------------------------+
+---------------+------------------------+-------+------------------+---------
--+-------------------------+
|   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  |
+---------------+------------------------+-----------+--------+
------------------------------XGBClassifier-------------------------------------
```

```
-----------------
+-----------------------------------------------+-----------------------------
-----------------+
| important_features_class_[0] for XGBClassifier | important_features_class_[1]
for XGBClassifier |
+-----------------------------------------------+-----------------------------
-----------------+
|                  (0.0, 'aa')                  |           (0.006829901,
'delicious')           |
|                  (0.0, 'aaa')                 |            (0.0059515843,
'worst')             |
|                 (0.0, 'aaaa')                 |             (0.0051996135,
'best')             |
|                 (0.0, 'aaaaa')                |             (0.0049646627,
'great')              |
|            (0.0, 'aaaaaaaaaaaaaaa')           |           (0.0049589006,
'perfect')            |
|             (0.0, 'aaaaaaaaagghh')            |            (0.0045655733,
'highly')             |
|               (0.0, 'aaaaaabr')               |            (0.0043366887,
'threw')              |
|                (0.0, 'aaaaah')                |            (0.0042989836,
'waste')              |
|       (0.0, 'aaaaahhhhhhhhhhhhhhhhthe')       |           (0.004289796,
'excellent')           |
|                 (0.0, 'aaaah')                |             (0.0040165526,
'easy')             |
|                 (0.0, 'aaah')                 |             (0.003805688,
'yummy')              |
|               (0.0, 'aaahhhhhh')              |           (0.0038038422,
'favorite')           |
|                 (0.0, 'aahs')                 |           (0.0035729995,
'awesome')             |
|                (0.0, 'aachen')                |            (0.0035460214,
'gross')              |
|               (0.0, 'aacurate')               |           (0.0034753508,
'horrible')           |
|                  (0.0, 'aad')                 |           (0.003406402,
'wonderful')           |
|                 (0.0, 'aadp')                 |           (0.0032368226,
'terrible')           |
|                  (0.0, 'aaf')                 |            (0.0031637086,
'hooked')             |
|                 (0.0, 'aafco')                |             (0.003051933,
'money')              |
|                (0.0, 'aafter')                |             (0.0030434837,
'yuck')             |
+-----------------------------------------------+-----------------------------
```

```
------------------+
+----------------+---------------+-------+---------------+-----------+---------
----------------+
|   Vectorizer   |     Model     | depth | learning_ratee | Auc_score |
f1_score(micro average) |
+----------------+---------------+-------+---------------+-----------+---------
----------------+
|      BOW       | XGBClassifier |   20  |      .01       |   .9245   |
0.90          |
|     tfidf      | XGBClassifier |   20  |      .01       |   .9245   |
0.91          |
|    Word2Vec    | XGBClassifier |   10  |      .01       |   .5065   |
0.84          |
| tfidf-Word2Vec | XGBClassifier |   10  |      .01       |   .4963   |
0.84          |
+----------------+---------------+-------+---------------+-----------+---------
----------------+
+----------------+---------------+---------+---------+
|    Field 1     |    Field 2    | Field 3 | Field 4 |
+----------------+---------------+---------+---------+
|      BOW       | XGBClassifier |   0.9   |   0.9   |
|     tfidf      | XGBClassifier |   0.91  |   0.91  |
|    Word2Vec    | XGBClassifier |   0.84  |   0.84  |
| tfidf-Word2Vec | XGBClassifier |   0.84  |   0.84  |
+----------------+---------------+---------+---------+
```