Amazon fine food- DT

December 26, 2018

1 Amazon fine food- DT

```
In [6]: import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns

        con = sqlite3.connect('./final.sqlite')

        review= pd.read_sql_query('''select * from Reviews''',con)

In [7]: review=review.sort_values('Time')

In [8]: review.shape

Out[8]: (9982, 11)

In [9]: labels=review.Score
        review=review.drop(['Score'],axis=1)
```

2 Bow

```
In [10]: from sklearn.feature_extraction.text import TfidfTransformer
         from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn import datasets, neighbors
         from matplotlib.colors import ListedColormap

         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.metrics import confusion_matrix
         from sklearn import metrics
         from sklearn.metrics import roc_curve, auc
         from nltk.stem.porter import PorterStemmer
```

In [11]:
```
per=int(0.7*review.shape[0])
a=review[0:per]
b=review[per:]
label=labels[0:per]
label1=labels[per:]
```

In [12]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit_transform(a['Text'].values)
```

Out[12]:
```
<6987x19546 sparse matrix of type '<class 'numpy.int64'>'
        with 382513 stored elements in Compressed Sparse Row format>
```

In [13]:
```
dtrain=count_vect.transform(a['Text'])
dtest=count_vect.transform(b['Text'])
```

In [14]:
```
def partition(x):
    if x =='negative':
        return 0
    return 1
```

In [15]:
```
#changing reviews with score less than 3 to be positive and vice-versa
actualScore = label
positiveNegative = actualScore.map(partition)
label= positiveNegative
actualScore = label1
positiveNegative = actualScore.map(partition)
label1= positiveNegative
```

In [16]:
```
# ============================== loading libraries ==================================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
# ===================================================================================
```

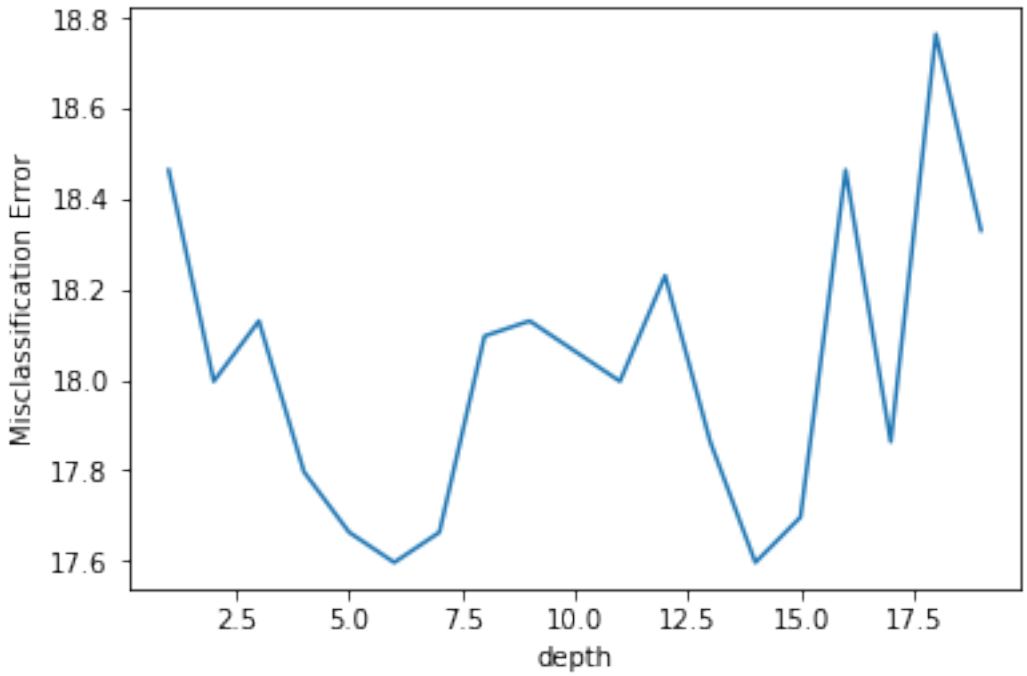
```
c:\users\vishal\appdata\local\programs\python\python36\lib\site-packages\sklearn\cross_validati
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [17]:
```
X_test, y_test = dtest, label1

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(dtrain, label, test_size=0
```

In [10]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection  import GridSearchCV
!pip install tqdm
import tqdm
from sklearn import tree
```

```
Requirement already satisfied: tqdm in c:\users\vishal\appdata\local\programs\python\python36\l
```

In [94]:
```
#a=166.81
scores = []
sparsity=[]
neighbors=range(1,20)
for a in neighbors:
    dt_optimal = DecisionTreeClassifier(max_depth=a)

    # fitting the model
    dt_optimal.fit(X_tr, y_tr)

    # predict the response
    pred = dt_optimal.predict(X_test)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    scores.append(acc)

MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('depth')
plt.ylabel('Misclassification Error')
plt.show()
```




```
In [95]: from sklearn.model_selection import StratifiedShuffleSplit

         grid_values = {'max_depth': range(1,50)}
         grid = GridSearchCV(DecisionTreeClassifier(),param_grid=grid_values)
         grid.fit(X_tr, y_tr)

         print("The best parameters are %s with a score of %0.2f"
               % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'max_depth': 2} with a score of 0.85
```

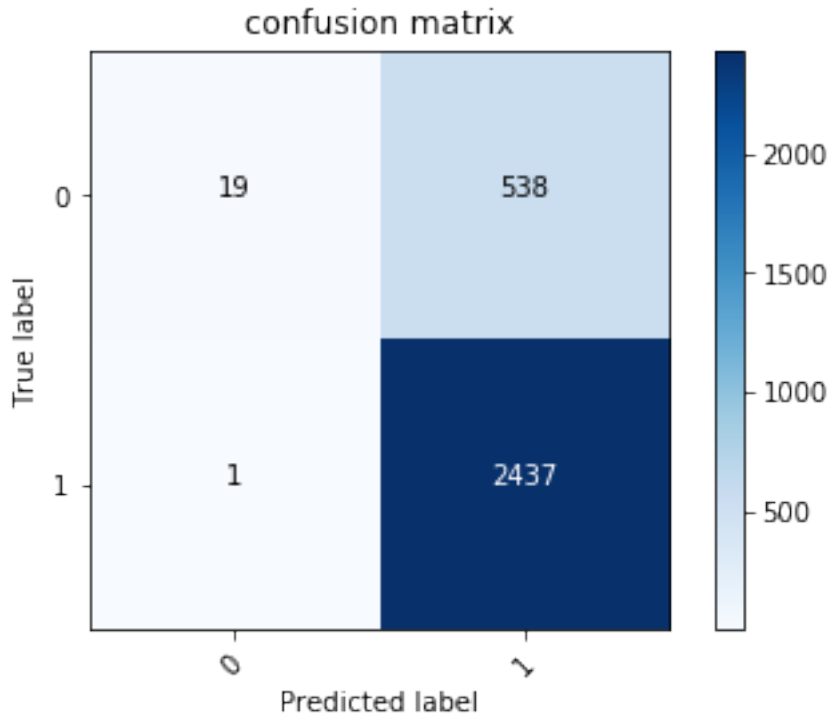
```
In [97]: sv = DecisionTreeClassifier(max_depth=2)
         sv.fit(X_tr,y_tr)
         pred = sv.predict(X_cv)
         acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
         print('\n****CV accuracy for max depth=2  is %d%%' % (acc))
```

```
****CV accuracy for max depth=2  is 85%
```

```
In [98]: sv = DecisionTreeClassifier(max_depth=2)
         sv.fit(X_tr,y_tr)
         pred = sv.predict(X_test)
         acc = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\n****test accuracy for max depth=2  is %d%%' % (acc))
```

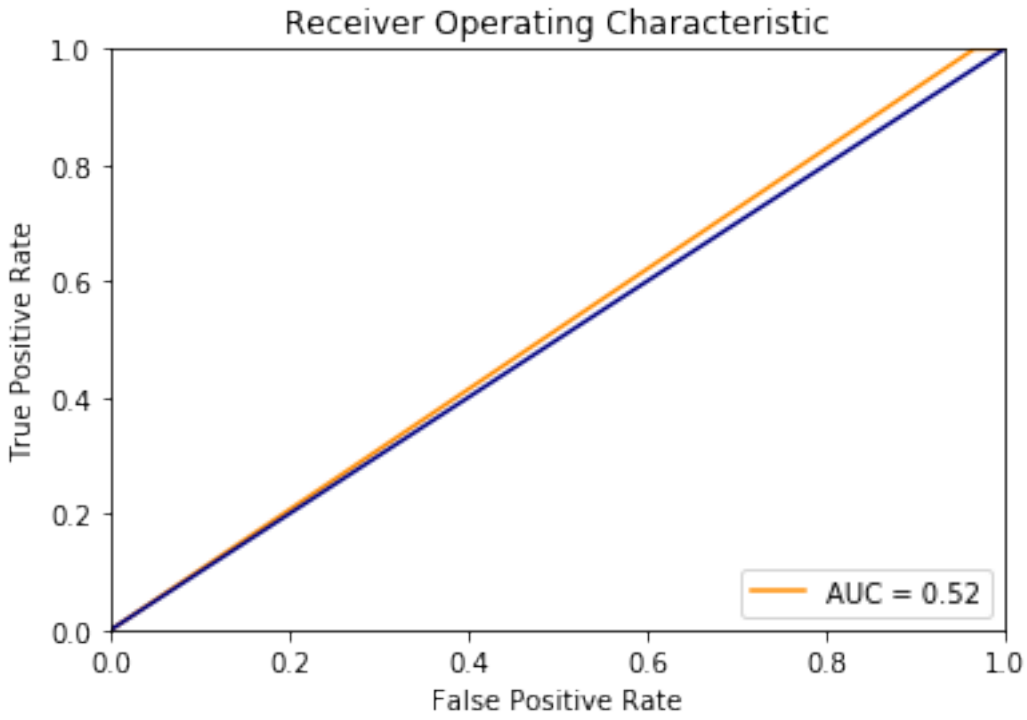
****test accuracy for max depth=2 is 82%

In [99]:
```
import itertools

confusion = confusion_matrix(y_test, pred)
plt.imshow(confusion,cmap=plt.cm.Blues)
plt.title('confusion matrix')

plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, rotation=45)
plt.yticks(tick_marks)

fmt = 'd'
thresh = confusion.max() / 2
for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
    plt.text(j, i, format(confusion[i, j], fmt),
             horizontalalignment="center",
             color="white" if confusion[i, j] > thresh else "black")


plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.show()
```



```
In [131]: import numpy as np
          from sklearn import metrics

          fpr, tpr, thresholds = metrics.roc_curve(y_test, pred)
          roc_auc = metrics.auc(fpr, tpr)
          print(roc_auc)
          plt.title('Receiver Operating Characteristic')
          plt.plot(fpr, tpr, color='darkorange', label = 'AUC = %0.2f' % roc_auc)
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1], color='navy')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```

```
0.5168505691600526
```




```
In [5]: import numpy as np
        np.logspace(-3, 3, 7)
```

```
Out[5]: array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])
```

3 TF-IDF

```
In [11]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
         tf_idf_vect.fit_transform(a['Text'].values)
```

```
Out[11]: <6987x223262 sparse matrix of type '<class 'numpy.float64'>'
                with 897925 stored elements in Compressed Sparse Row format>
```

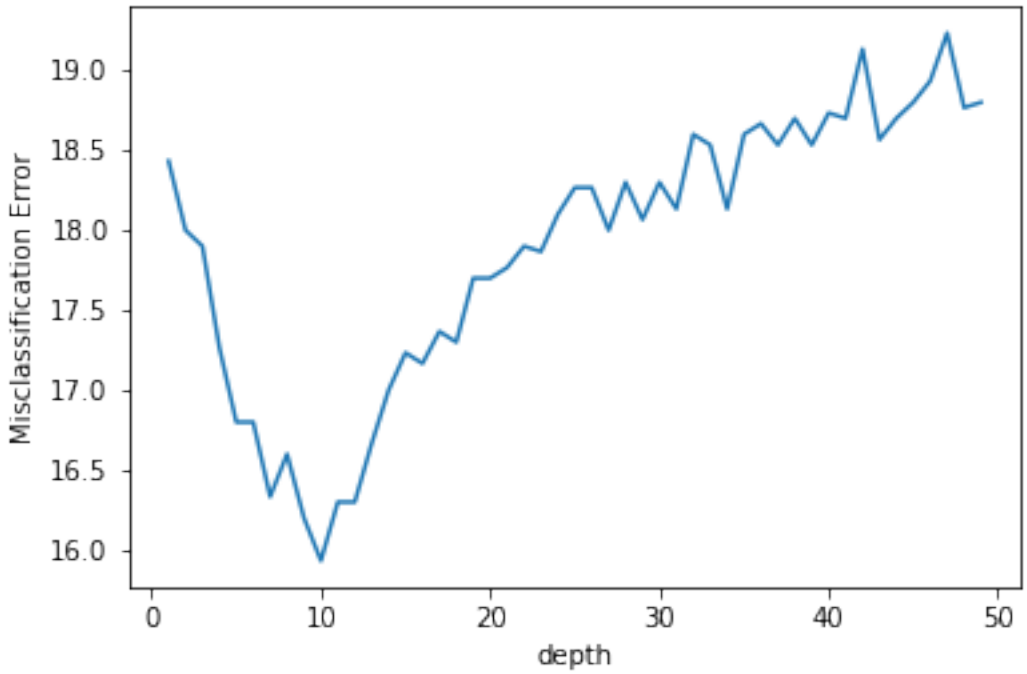
```
In [138]: dtrain=tf_idf_vect.transform(a['Text'])
          dtest=tf_idf_vect.transform(b['Text'])
```

```
In [139]: def partition(x):
              if x =='negative':
                  return 0
              return 1

          #changing reviews with score less than 3 to be positive and vice-versa
          actualScore = label
          positiveNegative = actualScore.map(partition)
          label= positiveNegative
          actualScore = label1
          positiveNegative = actualScore.map(partition)
          label1= positiveNegative
```

```
In [140]: X_test, y_test = dtest, label1

          # split the train data set into cross validation train and cross validation test
          X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(dtrain, label, test_size=(
```

```
In [168]: #a=166.81
          scores = []
          sparsity=[]
          neighbors=range(1,50)
          for a in neighbors:
              dt_optimal = DecisionTreeClassifier(max_depth=a)

              # fitting the model
              dt_optimal.fit(X_tr, y_tr)

              # predict the response
              pred = dt_optimal.predict(X_test)

              # evaluate accuracy
              acc = accuracy_score(y_test, pred) * 100
              scores.append(acc)

          MSE = [100 - x for x in scores]

          plt.plot(neighbors, MSE)
```

```
plt.grid()
plt.xlabel('depth')
plt.ylabel('Misclassification Error')
plt.show()
```



In [170]:
```
from sklearn.model_selection import StratifiedShuffleSplit

grid_values = {'max_depth': range(1,20)}
grid = GridSearchCV(DecisionTreeClassifier(),param_grid=grid_values)
grid.fit(X_tr, y_tr)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'max_depth': 6} with a score of 0.86
```

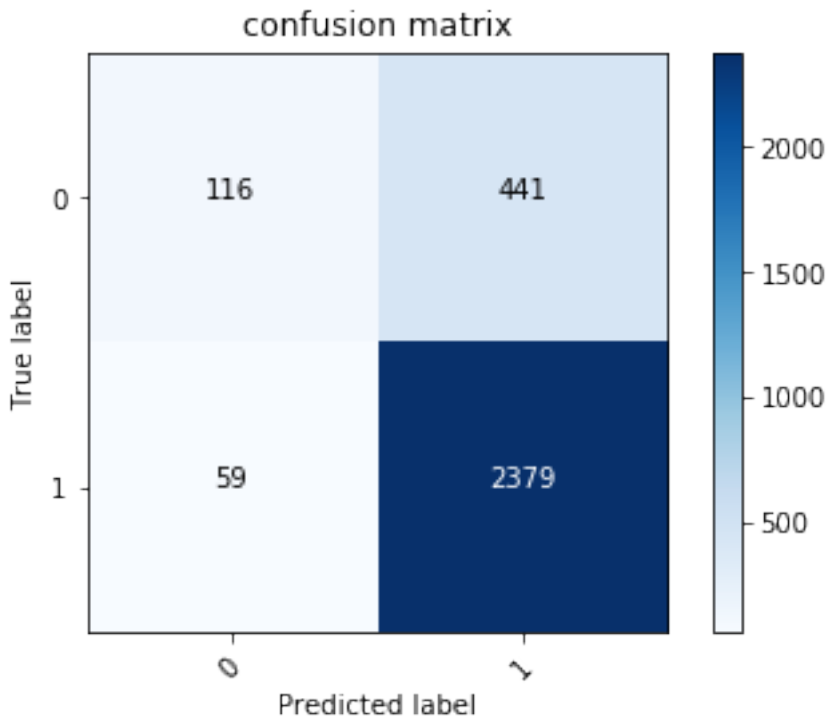
In [171]:
```
sv = DecisionTreeClassifier(max_depth=6)
sv.fit(X_tr,y_tr)
pred = sv.predict(X_cv)
acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
print('\n****CV accuracy for max depth=6  is %d%%' % (acc))
```

```
****CV accuracy for max depth=6  is 86%
```

In [172]:
```
sv = DecisionTreeClassifier(max_depth=6)
sv.fit(X_tr,y_tr)
pred = sv.predict(X_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****test accuracy for max depth=6  is %d%%' % (acc))
```

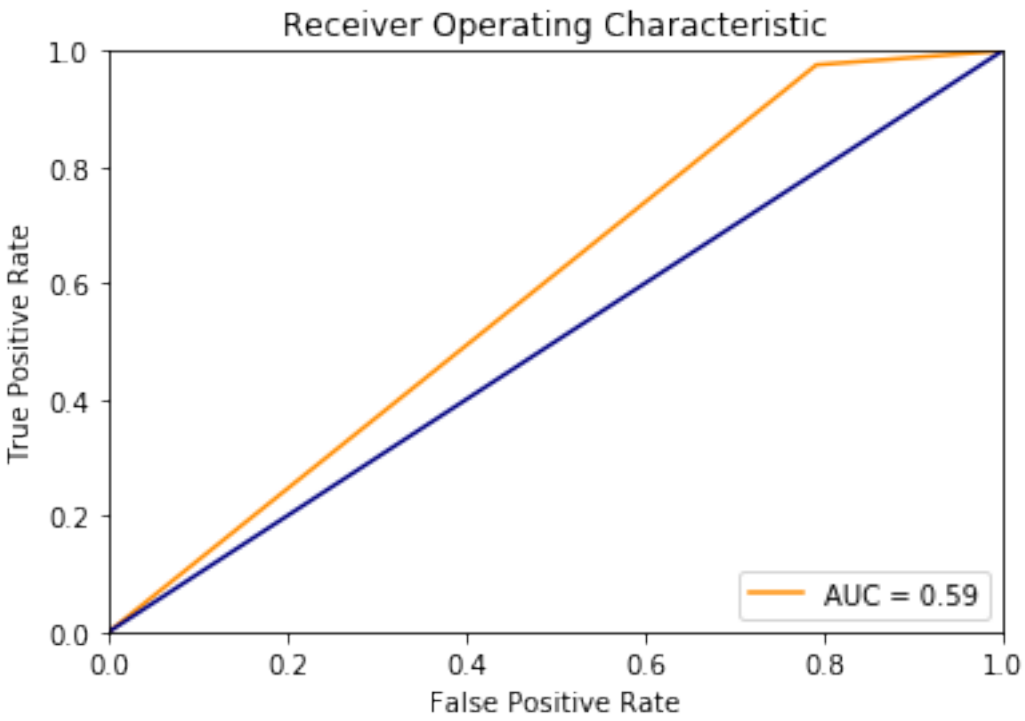
```
****test accuracy for max depth=6  is 83%
```

In [173]:
```
import itertools

confusion = confusion_matrix(y_test, pred)
plt.imshow(confusion,cmap=plt.cm.Blues)
plt.title('confusion matrix')

plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, rotation=45)
plt.yticks(tick_marks)

fmt = 'd'
thresh = confusion.max() / 2
for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
    plt.text(j, i, format(confusion[i, j], fmt),
             horizontalalignment="center",
             color="white" if confusion[i, j] > thresh else "black")


plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.show()
```



In [174]:
```
import numpy as np
from sklearn import metrics

fpr, tpr, thresholds = metrics.roc_curve(y_test, pred)
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color='darkorange', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], color='navy')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
0.5920291818793696
```



4 W2V

```
In [12]: import codecs
         import glob
         import logging
         import multiprocessing
         import os
         import pprint

         import gensim.models.word2vec as w2v
         import sklearn.manifold

         import re

         model = w2v.Word2Vec.load("trained/food2vec1.w2v")
```

```
c:\users\vishal\appdata\local\programs\python\python36\lib\site-packages\gensim\utils.py:1197:
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
In [13]: model
```

```
Out[13]: <gensim.models.word2vec.Word2Vec at 0x23973564b70>
```

In [14]:
```
# Train your own Word2Vec model using your own text corpus
import gensim
import re
i=0
list_of_sent=[]

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

def cleanpunc(sentence): #function to clean the word of any punctuation or special ch
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned


for sent in a['Text'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

In [15]:
```
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [16]:
```
words = list(w2v_model.wv.vocab)
print(len(words))
```

```
5830
```

In [17]:
```
all_word_vectors_matrix = w2v_model.wv.syn0
```

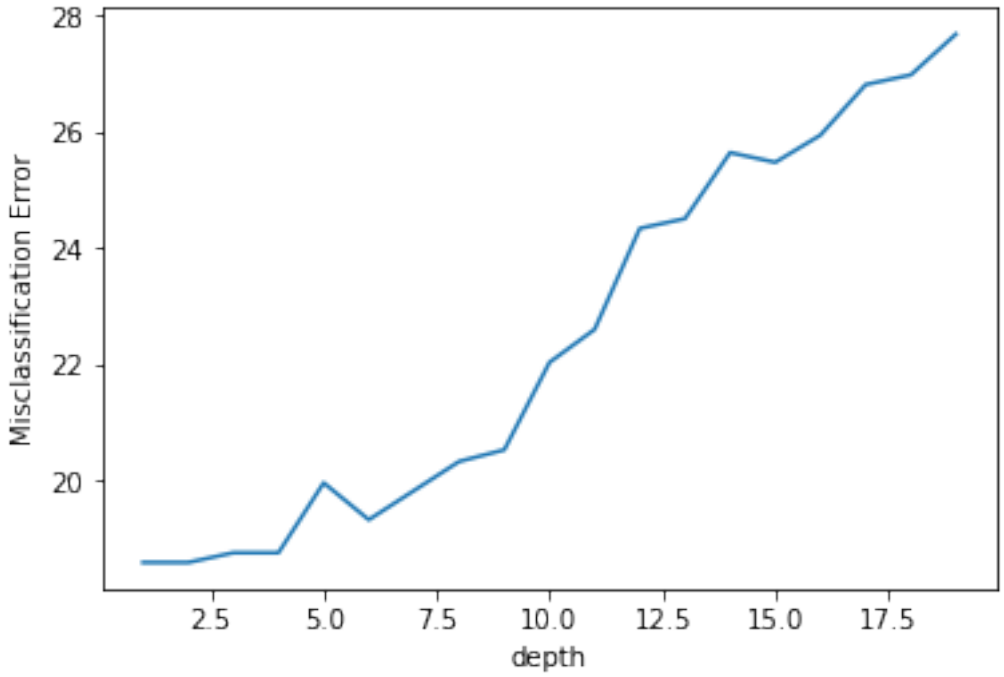
```
c:\users\vishal\appdata\local\programs\python\python36\lib\site-packages\ipykernel_launcher.py
  """Entry point for launching an IPython kernel.
```

In [17]:
```
count=0
temp=np.zeros((1,50))
revect=np.zeros((1,50))
```

In [164]:
```
np.seterr(divide='ignore', invalid='ignore')
 count=0
```

```
            for i in review['Text']:
                for j in cleanpunc(i).split():
                    if j in words:
                        temp=np.add(temp, all_word_vectors_matrix[words.index(j)])
                        count+=1
                if count!=0:
                    temp=[i/count for i in temp]
                count=0
                revect=np.concatenate((revect,temp))
                temp=np.zeros((1,50))

In [165]: revect=np.delete(revect,0,axis=0)

In [166]: revect.shape

Out[166]: (9982, 50)

In [169]: per=int(0.7*revect.shape[0])
          a=revect[0:per]
          b=revect[per:]

In [175]: actualScore = label
          positiveNegative = actualScore.map(partition)
          label= positiveNegative
          actualScore = label1
          positiveNegative = actualScore.map(partition)
          label1= positiveNegative

In [177]: X_test, y_test = b, label1

          # split the train data set into cross validation train and cross validation test
          X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(a, label, test_size=0.3)

In [178]: #a=166.81
          scores = []
          sparsity=[]
          neighbors=range(1,20)
          for a in neighbors:
              dt_optimal = DecisionTreeClassifier(max_depth=a)

              # fitting the model
              dt_optimal.fit(X_tr, y_tr)

              # predict the response
              pred = dt_optimal.predict(X_test)

              # evaluate accuracy
              acc = accuracy_score(y_test, pred) * 100
              scores.append(acc)
```

```
MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('depth')
plt.ylabel('Misclassification Error')
plt.show()
```



In [179]:
```
from sklearn.model_selection import StratifiedShuffleSplit

grid_values = {'max_depth': range(1,20)}
grid = GridSearchCV(DecisionTreeClassifier(),param_grid=grid_values)
grid.fit(X_tr, y_tr)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'max_depth': 1} with a score of 0.85
```

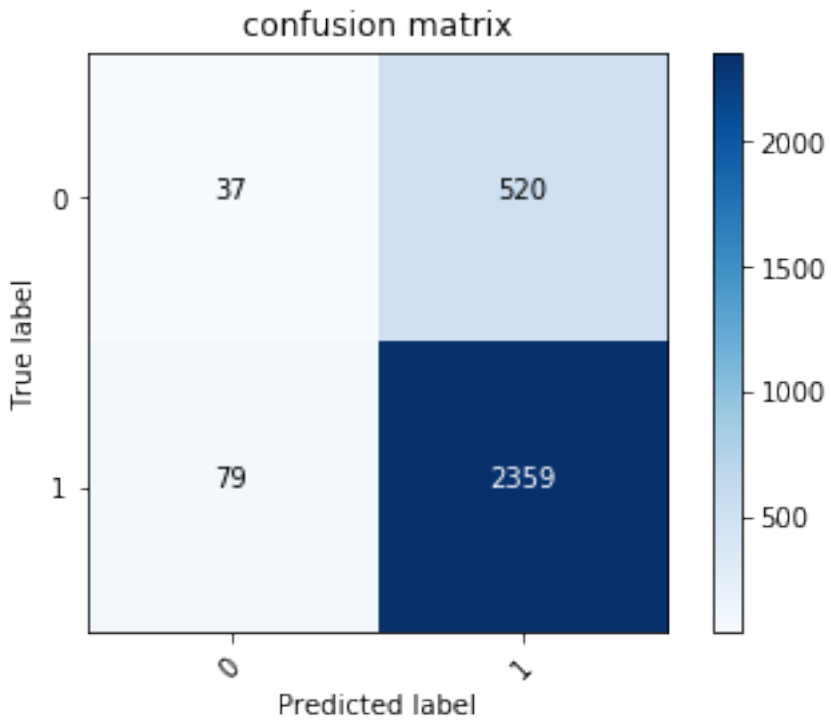
In [180]:
```
sv = DecisionTreeClassifier(max_depth=1)
sv.fit(X_tr,y_tr)
pred = sv.predict(X_cv)
acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
print('\n****CV accuracy for max depth=1  is %d%%' % (acc))
```

```
****CV accuracy for max depth=1  is 85%
```

```
In [181]: sv = DecisionTreeClassifier(max_depth=5)
          sv.fit(X_tr,y_tr)
          pred = sv.predict(X_test)
          acc = accuracy_score(y_test, pred, normalize=True) * float(100)
          print('\n****test accuracy for max depth=1  is %d%%' % (acc))
```

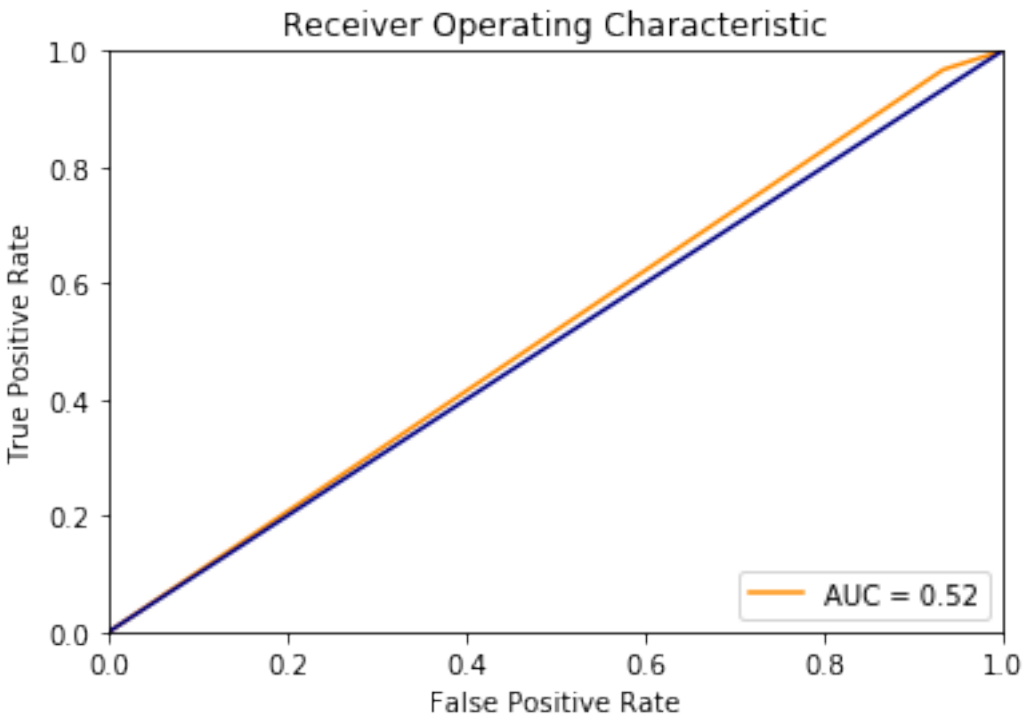
```
****test accuracy for max depth=1  is 80%
```

```
In [182]: import itertools

          confusion = confusion_matrix(y_test, pred)
          plt.imshow(confusion,cmap=plt.cm.Blues)
          plt.title('confusion matrix')

          plt.colorbar()
          tick_marks = np.arange(2)
          plt.xticks(tick_marks, rotation=45)
          plt.yticks(tick_marks)

          fmt = 'd'
          thresh = confusion.max() / 2
          for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
              plt.text(j, i, format(confusion[i, j], fmt),
                       horizontalalignment="center",
                       color="white" if confusion[i, j] > thresh else "black")


          plt.ylabel('True label')
          plt.xlabel('Predicted label')

          plt.show()
```




```
In [183]: import numpy as np
          from sklearn import metrics

          fpr, tpr, thresholds = metrics.roc_curve(y_test, pred)
          roc_auc = metrics.auc(fpr, tpr)
          print(roc_auc)
          plt.title('Receiver Operating Characteristic')
          plt.plot(fpr, tpr, color='darkorange', label = 'AUC = %0.2f' % roc_auc)
          plt.legend(loc = 'lower right')
          plt.plot([0, 1], [0, 1], color='navy')
          plt.xlim([0, 1])
          plt.ylim([0, 1])
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.show()
```

```
0.5170118397662387
```



5 Tfidf- W2V

```
In [18]: # TF-IDF weighted Word2Vec
         from sklearn.feature_extraction.text import TfidfTransformer
         from sklearn.feature_extraction.text import TfidfVectorizer

In [19]: tfidf_feat = tf_idf_vect.get_feature_names()

In [48]: np.seterr(divide='ignore', invalid='ignore')
         revect=np.zeros((1,50))
         for i in tqdm.tqdm(review['Text']):
             sent_vec = np.zeros((1,50))
             weight_sum =0;

             for j in cleanpunc(i).split():
                 if j in w2v_model.wv and j in tf_idf_vect.vocabulary_:
                     vec = w2v_model.wv[j]

                     #print(tfidf_feat.index(word))
                     # obtain the tf_idfidf of a word in a sentence/review
                     tf_idf = tfidf_feat.index(j)

                     sent_vec += (vec * tf_idf)
```

```
                    weight_sum += tf_idf

            sent_vec /= weight_sum
            revect=np.concatenate((revect,sent_vec))
```

```
100%|| 9982/9982 [1:18:22<00:00,  2.12it/s]
```

```
In [49]: revect=np.delete(revect,0,axis=0)
```

```
In [74]: revect[378]
```

```
Out[74]: array([ 0.35760214, -0.0375447 , -0.61522902,  0.22957666,  0.37983019,
                 0.25043158,  0.00530838, -0.55306081,  0.0721186 , -0.24519114,
                -0.11105667, -0.03496286,  0.30415614, -0.18254267,  0.1095003 ,
                -0.40943566, -0.15800385,  0.1444826 , -0.35447146, -0.15232457,
                 0.21691892,  0.3807691 , -0.03972623, -0.21448196, -0.00836947,
                 0.1209112 ,  0.70938468,  0.44282689,  0.33717179,  0.62536209,
                -0.45028976,  0.26676877, -0.79960853, -0.3381092 ,  0.144765  ,
                 0.11670489,  0.29700065, -0.07014012, -0.31835951,  0.38804883,
                 0.26344096,  0.09399787,  0.57581403, -0.46837896,  0.62700151,
                -0.14371044, -0.20018703, -0.28529516, -0.30279676,  0.42823893])
```

```
In [51]: per=int(0.7*revect.shape[0])
         a=revect[0:per]
         b=revect[per:]
```

```
In [73]: np.where(np.isnan(X_tr))
```

```
Out[73]: (array([ 378,  378,  378, ..., 4881, 4881, 4881], dtype=int64),
          array([ 0,  1,  2, ..., 47, 48, 49], dtype=int64))
```

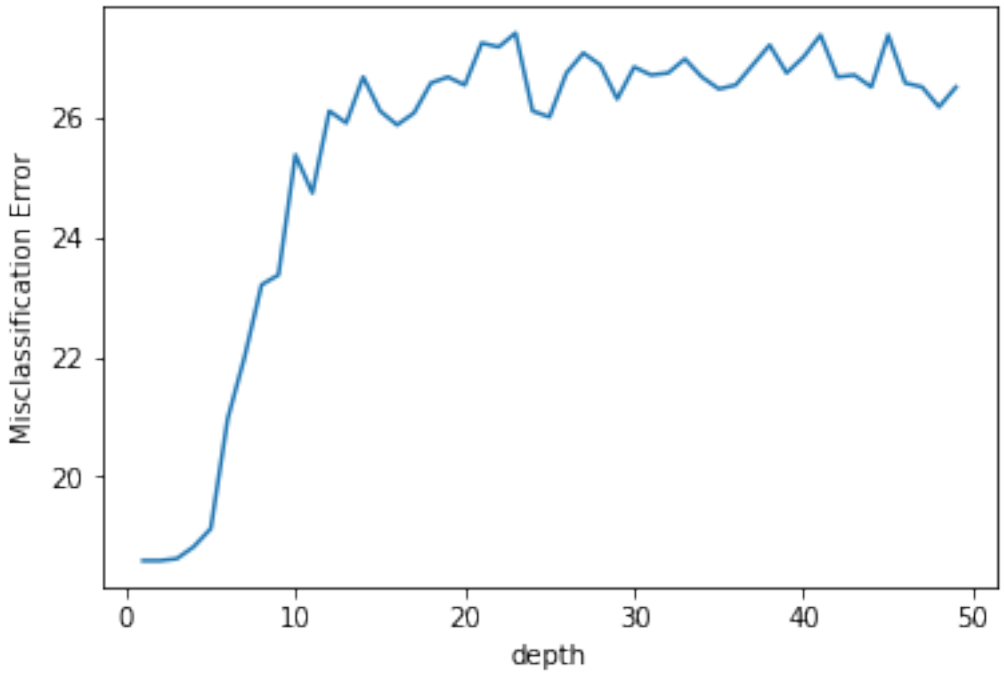
```
In [77]: from sklearn.preprocessing import Imputer
         trainDataVecs = Imputer().fit_transform(X_tr)
         testdatavecs=Imputer().fit_transform(X_test)
```

```
In [84]: cvdatavecs=Imputer().fit_transform(X_cv)
```

```
In [56]: # split the data set into train and test
         X_test, y_test = b, label1

         # split the train data set into cross validation train and cross validation test
         X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(a, label, test_size=0.3)
```

```
In [79]: #a=166.81
         scores = []
         sparsity=[]
         neighbors=range(1,50)
         for a in neighbors:
             dt_optimal = DecisionTreeClassifier(max_depth=a)
```

```
    # fitting the model
    dt_optimal.fit(trainDataVecs, y_tr)

    # predict the response
    pred = dt_optimal.predict(testdatavecs)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    scores.append(acc)

MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('depth')
plt.ylabel('Misclassification Error')
plt.show()
```




```
In [82]: from sklearn.model_selection import StratifiedShuffleSplit

grid_values = {'max_depth': range(1,50)}
grid = GridSearchCV(DecisionTreeClassifier(),param_grid=grid_values)
grid.fit(trainDataVecs, y_tr)
```

```
        print("The best parameters are %s with a score of %0.2f"
              % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'max_depth': 1} with a score of 0.85
```

```
In [85]: sv = DecisionTreeClassifier(max_depth=1)
         sv.fit(trainDataVecs,y_tr)
         pred = sv.predict(cvdatavecs)
         acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
         print('\n****CV accuracy for max depth=1  is %d%%' % (acc))
```

```
****CV accuracy for max depth=1  is 85%
```

```
In [87]: sv = DecisionTreeClassifier(max_depth=5)
         sv.fit(trainDataVecs,y_tr)
         pred = sv.predict(testdatavecs)
         acc = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\n****test accuracy for max depth=1  is %d%%' % (acc))
```

```
****test accuracy for max depth=1  is 80%
```

```
In [130]: mp={}
          count=0
          for i in sv.feature_importances_:
              count+=1
              if i!=0:
                  print(count)
                  mp.update({count:i})
```

```
1
8
12
14
18
19
21
24
26
29
31
32
39
40
41
```

```
43
44
46
```

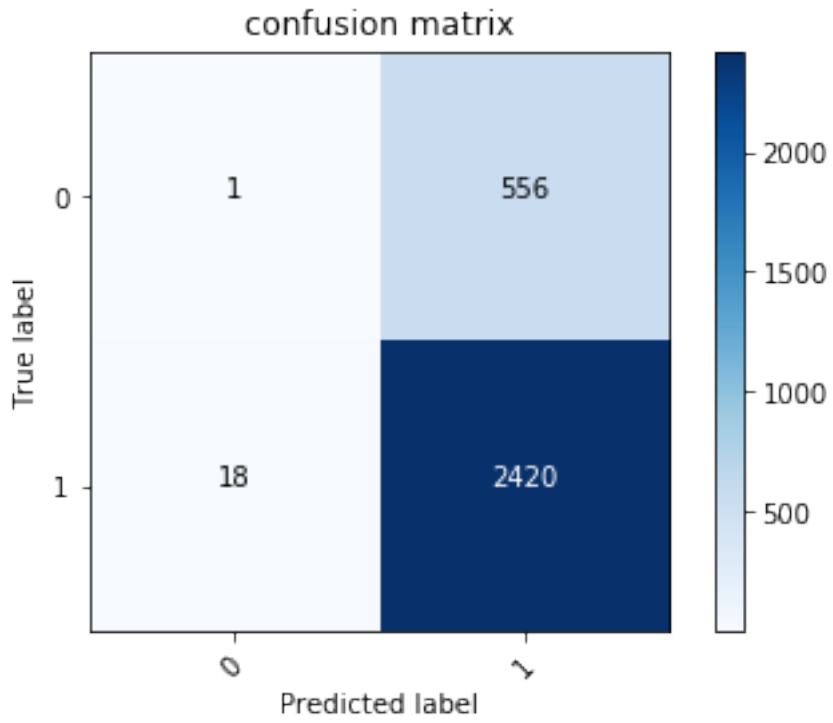
```
In [133]: import operator
          print(mp)
          sorted_d = [(k,v) for k,v in .items()]
          sorted_d
```

```
{1: 0.08910319547759818, 8: 0.014754114479474181, 12: 0.010530363906229037, 14: 0.015889046362
```

```
Out[133]: [(1, 0.08910319547759818),
           (8, 0.014754114479474181),
           (12, 0.010530363906229037),
           (14, 0.015889046362510657),
           (18, 0.019113206008185703),
           (19, 0.0205342952306605),
           (21, 0.06987487101663552),
           (24, 0.017639217132513995),
           (26, 0.009292792636342735),
           (29, 0.058760600361833414),
           (31, 0.03181867161266615),
           (32, 0.015546834661352016),
           (39, 0.028981605495340994),
           (40, 0.2941153982417443),
           (41, 0.07224288770582883),
           (43, 0.025147751874211633),
           (44, 0.1392728769103012),
           (46, 0.06738227088657088)]
```

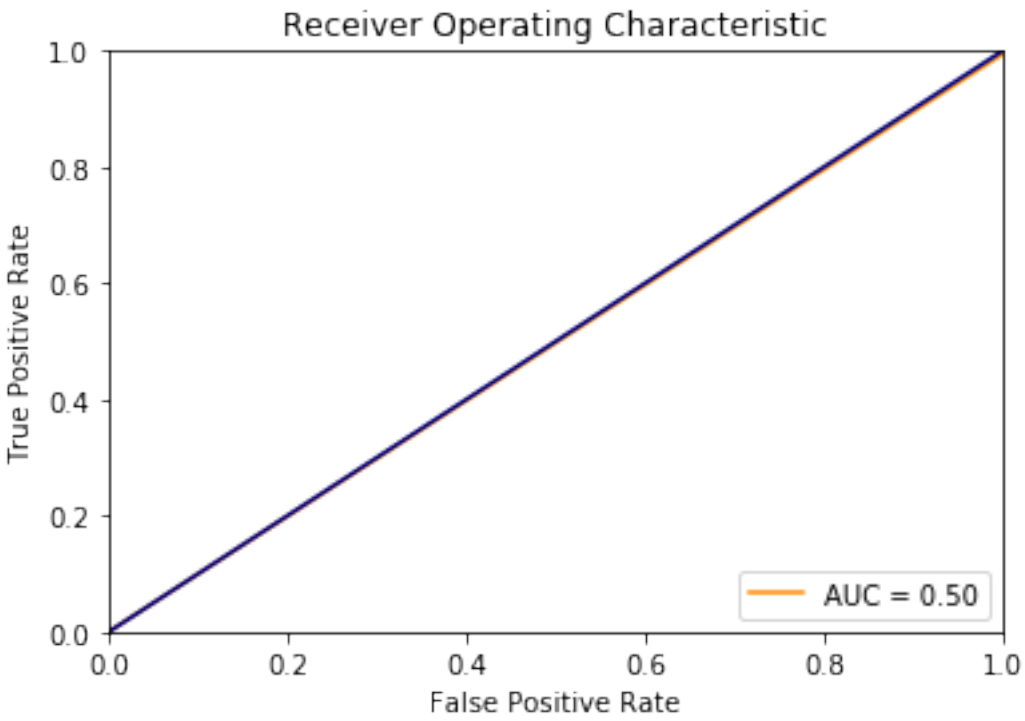
```
In [88]: import itertools

         confusion = confusion_matrix(y_test, pred)
         plt.imshow(confusion,cmap=plt.cm.Blues)
         plt.title('confusion matrix')

         plt.colorbar()
         tick_marks = np.arange(2)
         plt.xticks(tick_marks, rotation=45)
         plt.yticks(tick_marks)

         fmt = 'd'
         thresh = confusion.max() / 2
         for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
             plt.text(j, i, format(confusion[i, j], fmt),
                      horizontalalignment="center",
                      color="white" if confusion[i, j] > thresh else "black")
```

```
plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.show()
```



In [89]:
```
import numpy as np
from sklearn import metrics

fpr, tpr, thresholds = metrics.roc_curve(y_test, pred)
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, color='darkorange', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], color='navy')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
0.4972061156170331
```



6 Conclusion

```
In [93]: import pandas as pd
         list=['model','CV','best depth','Train metric','Test metric','AUC']
         conclu = [('BoW','Grid',2,'85%', '82%','0.52'),
                        ('tf-idf','Grid',6,'86%', '83%','0.59'),
                        ('avg-W2Vec','Grid', 1,'85%','80%','0.52'),
                       ('tfidf-W2Vec','Grid', 1,'85%','80%','0.50')
                        ]
         df = pd.DataFrame(conclu, columns=list)
         df
```

Out[93]:

	model	CV	best depth	Train metric	Test metric	AUC
0	BoW	Grid	2	85%	82%	0.52
1	tf-idf	Grid	6	86%	83%	0.59
2	avg-W2Vec	Grid	1	85%	80%	0.52
3	tfidf-W2Vec	Grid	1	85%	80%	0.50

6.0.1 From the above observation table we can see that tf-idf is working well as it's AUC=0.59