In [2]:
```
# =============================== Loading libraries ===============================
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pickle
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
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import classification_report
from prettytable import PrettyTable
import random
from scipy.stats import uniform
from sklearn.metrics import roc_curve, auc
from sklearn.learning_curve import validation_curve
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import precision_score, recall_score,roc_auc_score
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
from sklearn.calibration import CalibratedClassifierCV
import joblib
from sklearn.svm import SVC
from sklearn import svm
from sklearn import linear_model
from scipy import stats
import scikitplot as skplt

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

#import nltk
#nltk.download('stopwords')

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

#from gensim.models import KeyedVectors
#model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin.g

#import gensim
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from sklearn.decomposition import TruncatedSVD
from sklearn import tree
import graphviz

#import os
#os.environ["PATH"] += os.pathsep + 'C:/Users/AbhiShek/Anaconda3/Lib/site-packag

# ==============================================================================
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the m
odel_selection module into which all the refactored classes and functions are m
oved. Also note that the interface of the new CV iterators are different from t
hat of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\learning_curve.py:22: Dep
recationWarning: This module was deprecated in version 0.18 in favor of the mod
el_selection module into which all the functions are moved. This module will be
removed in 0.20
  DeprecationWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarnin
g: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [3]:

```
fileObject = open("./train_to_file3.pkl",'rb') # we open the file for reading
X_train = pickle.load(fileObject) # load the object from the file

fileObject = open("./x_test_to_file3.pkl",'rb') # we open the file for reading
X_test = pickle.load(fileObject) # load the object from the file

fileObject = open("./y_train_to_file3.pkl",'rb') # we open the file for reading
y_train = pickle.load(fileObject) # load the object from the file

fileObject = open("./y_test_to_file3.pkl",'rb') # we open the file for reading
y_test = pickle.load(fileObject) # load the object from the file
```

In [4]:
```
print(np.shape(X_train))
print(np.shape(X_test))
print(np.shape(y_train))
print(np.shape(y_test))
```

```
(70000, 11)
(30000, 11)
(70000,)
(30000,)
```

BoW

In [5]:
```
#Appling BoW to fit and transform
count_vect =  CountVectorizer()
bow_NB = count_vect.fit(X_train[:,9])
train_bow_nstd = count_vect.transform(X_train[:,9])
test_bow_nstd = count_vect.transform(X_test[:,9])

print("the type of count vectorizer ",type(train_bow_nstd))
print("the number of unique words ", test_bow_nstd.get_shape()[1])

print(train_bow_nstd.shape)
print(test_bow_nstd.shape)
print(y_train.shape)
print(y_test.shape)
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words  50158
(70000, 50158)
(30000, 50158)
(70000,)
(30000,)
```

In [6]:
```
# Column Standardization of the BoW non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(train_bow_nstd)
train_bow = std_scal.transform(train_bow_nstd)
test_bow = std_scal.transform(test_bow_nstd)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [7]:
```
clf_dtree = tree.DecisionTreeClassifier()
clf_dtree = clf_dtree.fit(train_bow, y_train)
clf_dtree
```

Out[7]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [8]:
```
parameter = {
        'max_depth': (10, 50, 100, 500),
        'min_samples_split': (5, 10, 50, 100)
    }

gsearch_dt = GridSearchCV(estimator = clf_dtree,
                          param_grid= parameter,
                          cv=3,
                          scoring='f1')
gsearch_dt.fit(train_bow, y_train)

print(gsearch_dt)
results = gsearch_dt.cv_results_

# summarize the results of the grid search
print("\nBest score: ",gsearch_dt.best_score_)
NB_OPTIMAL_clf = gsearch_dt.best_estimator_

best_max_depth_bow = gsearch_dt.best_estimator_.max_depth
print("\nOptimal value of Hyperparameter, max_depth : ",best_max_depth_bow)

best_min_samples_split_bow = gsearch_dt.best_estimator_.min_samples_split
print("\nOptimal value of Hyperparameter, min_samples_split : ",best_min_samples_
```

```
GridSearchCV(cv=3, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': (10, 50, 100, 500), 'min_samples_split': (5, 1
0, 50, 100)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)

Best score:  0.9401310501841424

Optimal value of Hyperparameter, max_depth :  10

Optimal value of Hyperparameter, min_samples_split :  100
```

In [11]:
```
joblib.dump(results,"results.pkl")
```

Out[11]:
```
['results.pkl']
```

In [12]:
```
results = joblib.load("results.pkl")
```

In [9]:
```
results
```

Out[9]:
```
{'mean_fit_time': array([ 16.3110857 ,  16.64579741,  13.74764299,  12.3496239
2,
         70.8750759 ,  88.93902961,  75.33119925,  69.67070786,
        135.66119162, 125.24920209, 118.05657617, 113.17246819,
        160.41656399, 156.24411011, 143.60378257, 115.96239297]),
 'std_fit_time': array([0.96940904, 1.01507751, 0.92042447, 0.23798104, 1.01021
57 ,
        2.22835542, 3.30128648, 3.03134913, 3.05537011, 3.92056701,
        1.65869858, 2.22295289, 6.50952195, 4.34062752, 9.9097419 ,
        4.74610445]),
 'mean_score_time': array([0.13271681, 0.13827046, 0.12509616, 0.12499436, 0.14
242188,
        0.16300591, 0.15599672, 0.17466346, 0.17833591, 0.17233086,
        0.17087579, 0.1633265 , 0.18667348, 0.19000642, 0.21827443,
        0.1632483 ]),
 'std_score_time': array([1.26570569e-02, 3.21237660e-03, 1.13928551e-04, 1.597
74052e-05,
        2.52805026e-03, 1.15247754e-02, 1.41400001e-03, 4.02562737e-03,
        6.79366455e-03, 4.91713159e-03, 2.16365238e-02, 1.81356792e-02,
        1.08686603e-02, 4.97057453e-03, 2.17238962e-02, 9.88158889e-03]),
 'param_max_depth': masked_array(data=[10, 10, 10, 10, 50, 50, 50, 50, 100, 10
0, 100, 100,
                    500, 500, 500, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 50, 100, 5, 10, 50, 100,
5, 10, 50, 100, 5, 10,
                    50, 100],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 50},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 50},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 100, 'min_samples_split': 5},
  {'max_depth': 100, 'min_samples_split': 10},
  {'max_depth': 100, 'min_samples_split': 50},
  {'max_depth': 100, 'min_samples_split': 100},
  {'max_depth': 500, 'min_samples_split': 5},
  {'max_depth': 500, 'min_samples_split': 10},
  {'max_depth': 500, 'min_samples_split': 50},
  {'max_depth': 500, 'min_samples_split': 100}],
 'split0_test_score': array([0.94196439, 0.94206479, 0.94176619, 0.94165781, 0.
93164919,
        0.93104764, 0.93101065, 0.93167672, 0.92846358, 0.92768511,
        0.92850978, 0.92695807, 0.92476014, 0.92568104, 0.92612258,
        0.92638966]),
```

```
 'split1_test_score': array([0.93931767, 0.93905991, 0.93975455, 0.94025333, 0.
9308098 ,
        0.93129297, 0.93348914, 0.93364466, 0.92661497, 0.92726487,
        0.92875471, 0.92944167, 0.92493562, 0.925836  , 0.92719404,
        0.9285183 ]),
 'split2_test_score': array([0.93842215, 0.93885244, 0.9384508 , 0.93848195, 0.
92906124,
        0.92779014, 0.9288857 , 0.9314143 , 0.92445022, 0.92438026,
        0.92432003, 0.92786964, 0.92338885, 0.92253436, 0.92288011,
        0.92553805]),
 'mean_test_score': array([0.93990144, 0.93999241, 0.93999054, 0.94013105, 0.93
050676,
        0.9300436 , 0.9311285 , 0.93224522, 0.92650962, 0.92644343,
        0.92719486, 0.92808978, 0.92436154, 0.92468381, 0.92539892,
        0.92681533]),
 'std_test_score': array([0.00150387, 0.00146787, 0.00136375, 0.00129942, 0.001
07804,
        0.00159656, 0.00188118, 0.00099533, 0.00164015, 0.00146892,
        0.00203524, 0.00102581, 0.00069151, 0.00152119, 0.00183398,
        0.00125335]),
 'rank_test_score': array([ 4,  2,  3,  1,  7,  8,  6,  5, 12, 13, 10,  9, 16,
15, 14, 11]),
 'split0_train_score': array([0.94813652, 0.94771114, 0.94652363, 0.94466924,
0.98473228,
        0.98078838, 0.96847835, 0.96105546, 0.99282091, 0.98751257,
        0.97539058, 0.96722601, 0.99515053, 0.99031347, 0.97697217,
        0.96938441]),
 'split1_train_score': array([0.94769259, 0.94731439, 0.94567118, 0.94406997,
0.98460838,
        0.98075009, 0.96735613, 0.96050379, 0.99316745, 0.98833024,
        0.97438008, 0.96730272, 0.99538016, 0.99038005, 0.97578931,
        0.96895494]),
 'split2_train_score': array([0.94761722, 0.94735494, 0.94614445, 0.94490978,
0.98481052,
        0.98110669, 0.96976468, 0.96358858, 0.99221169, 0.98684752,
        0.97541083, 0.96935973, 0.99564974, 0.99020574, 0.97799541,
        0.97163387]),
 'mean_train_score': array([0.94781544, 0.94746015, 0.94611309, 0.94454966, 0.9
8471706,
        0.98088172, 0.96853305, 0.96171594, 0.99273335, 0.98756344,
        0.97506049, 0.96796282, 0.99539348, 0.99029976, 0.97691896,
        0.96999107]),
 'std_train_score': array([2.29109340e-04, 1.78241543e-04, 3.48716753e-04, 3.53
121380e-04,
        8.32237147e-05, 1.59845763e-04, 9.84043537e-04, 1.34317118e-03,
        3.95068817e-04, 6.06386019e-04, 4.81199460e-04, 9.88264607e-04,
        2.04021225e-04, 7.18215617e-05, 9.01421895e-04, 1.17478679e-03])}
```

In [65]:
```
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_test_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
```

```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

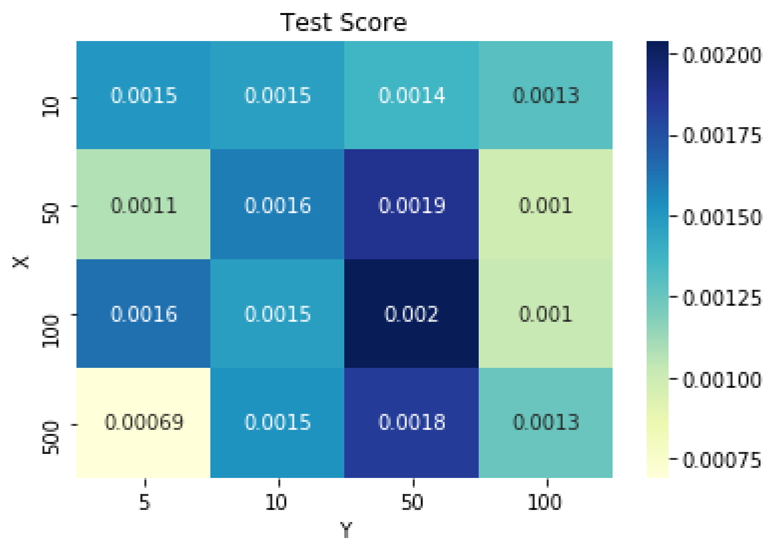
In [68]:
```
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

Out[68]:

	X	Y	Z
0	10	5	0.001504
1	10	10	0.001468
2	10	50	0.001364
3	10	100	0.001299
4	50	5	0.001078
5	50	10	0.001597
6	50	50	0.001881
7	50	100	0.000995
8	100	5	0.001640
9	100	10	0.001469
10	100	50	0.002035
11	100	100	0.001026
12	500	5	0.000692
13	500	10	0.001521
14	500	50	0.001834
15	500	100	0.001253

In [71]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Test Score')
```

Out[71]:
```
Text(0.5, 1.0, 'Test Score')
```



In [73]:

```python
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_train_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

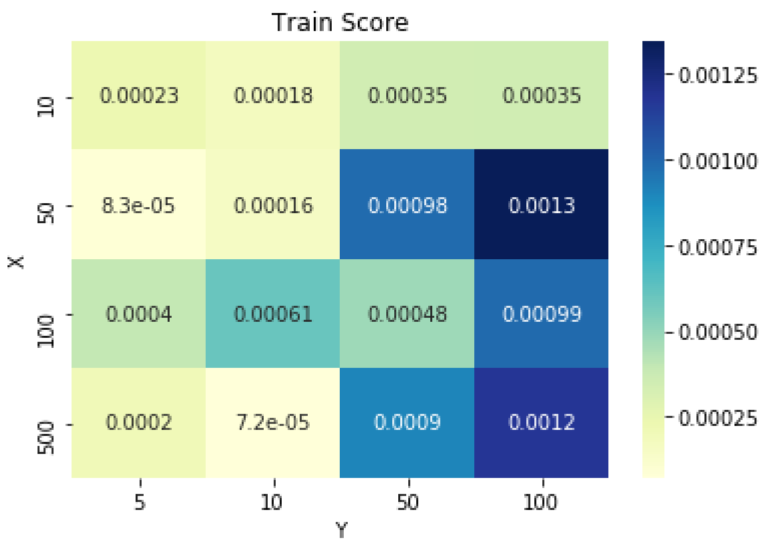
```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

Out[73]:

	X	Y	Z
0	10	5	0.000229
1	10	10	0.000178
2	10	50	0.000349
3	10	100	0.000353
4	50	5	0.000083
5	50	10	0.000160
6	50	50	0.000984
7	50	100	0.001343
8	100	5	0.000395

In [74]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Train Score')
```

Out[74]: Text(0.5, 1.0, 'Train Score')



Decision Tree on BoW with Best Parameters

In [35]:
```
clf_dtree_best = tree.DecisionTreeClassifier(max_depth=10, min_samples_split=100
clf_dtree_best.fit(train_bow, y_train)

y_pred_test = clf_dtree_best.predict(test_bow)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_te
roc_auc_best = auc(false_positive_rate, true_positive_rate)

joblib.dump(clf_dtree_best,"clf_dtree_best.pkl")
joblib.dump(y_pred_test,"y_pred_test.pkl")
joblib.dump(roc_auc_best,"roc_auc_best.pkl")
```

Out[35]: ['roc_auc_best.pkl']

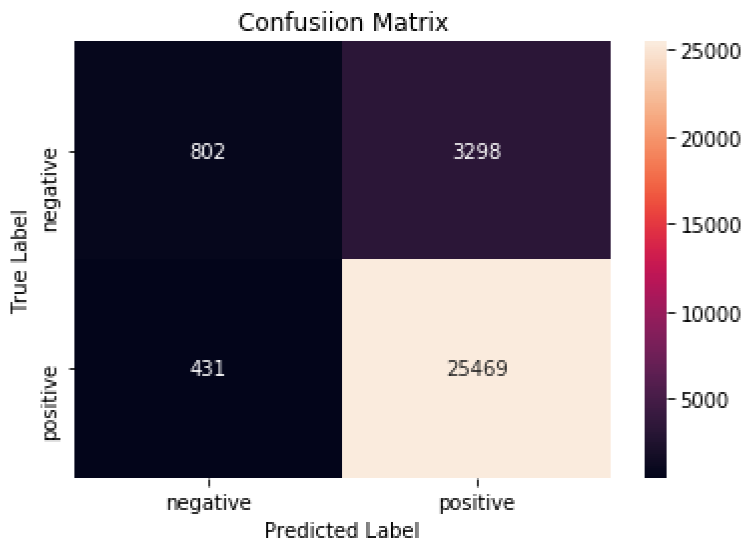
In [77]:
```
clf_dtree_best = joblib.load("clf_dtree_best.pkl")
y_pred_test = joblib.load("y_pred_test.pkl")
roc_auc_best = joblib.load("roc_auc_best.pkl")
roc_auc_best
```

Out[77]: 0.5894844147283171

In [37]:
```
# Confusion Matrix on Test Data
#y_pred = np.argmax(pred_test, axis=1)
cm_bow = confusion_matrix(y_test, y_pred_test)
cm_bow
```

Out[37]:
```
array([[  802,  3298],
       [  431, 25469]], dtype=int64)
```

In [38]:
```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_bow, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

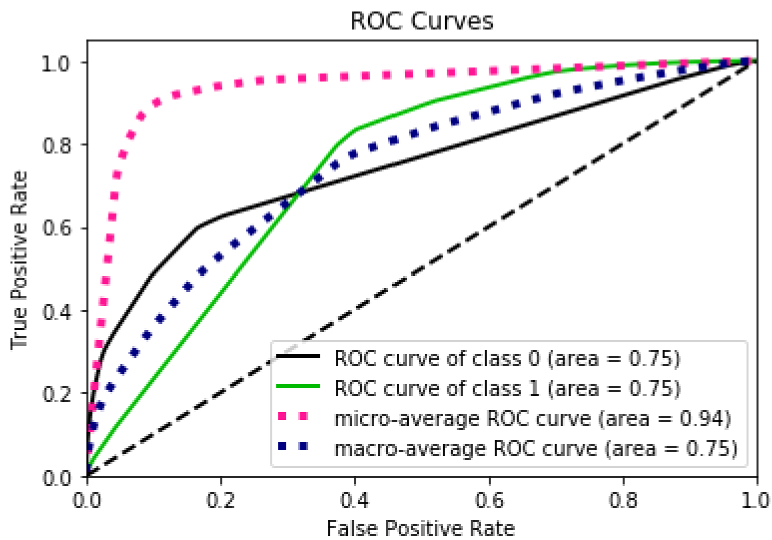


In [40]:
```
y_pred_train_proba = clf_dtree_best.predict_proba(train_bow)
y_pred_test_proba = clf_dtree_best.predict_proba(test_bow)
```

In [41]:
```
#Plotting ROC curve over Train Data
skplt.metrics.plot_roc_curve(y_train,y_pred_train_proba)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

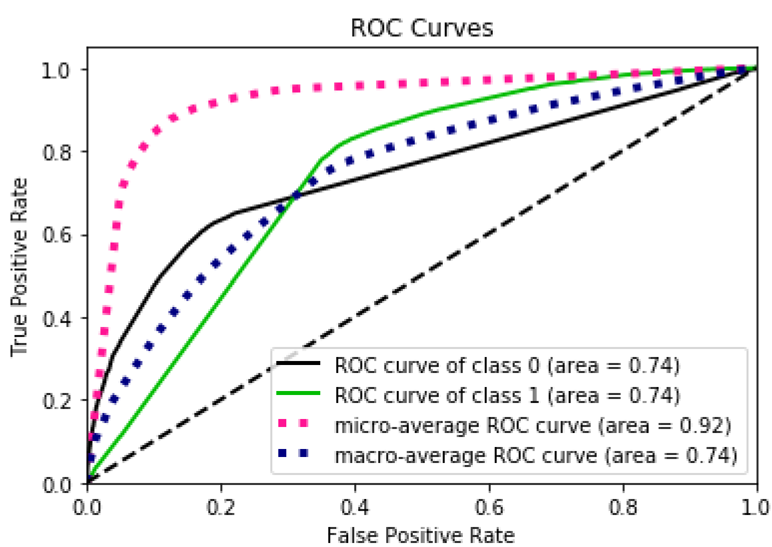
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x2ec6bc9bb00>



In [42]:
```
#Plotting ROC curve over Test Data
skplt.metrics.plot_roc_curve(y_test,y_pred_test_proba)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x2ec6bd45f28>

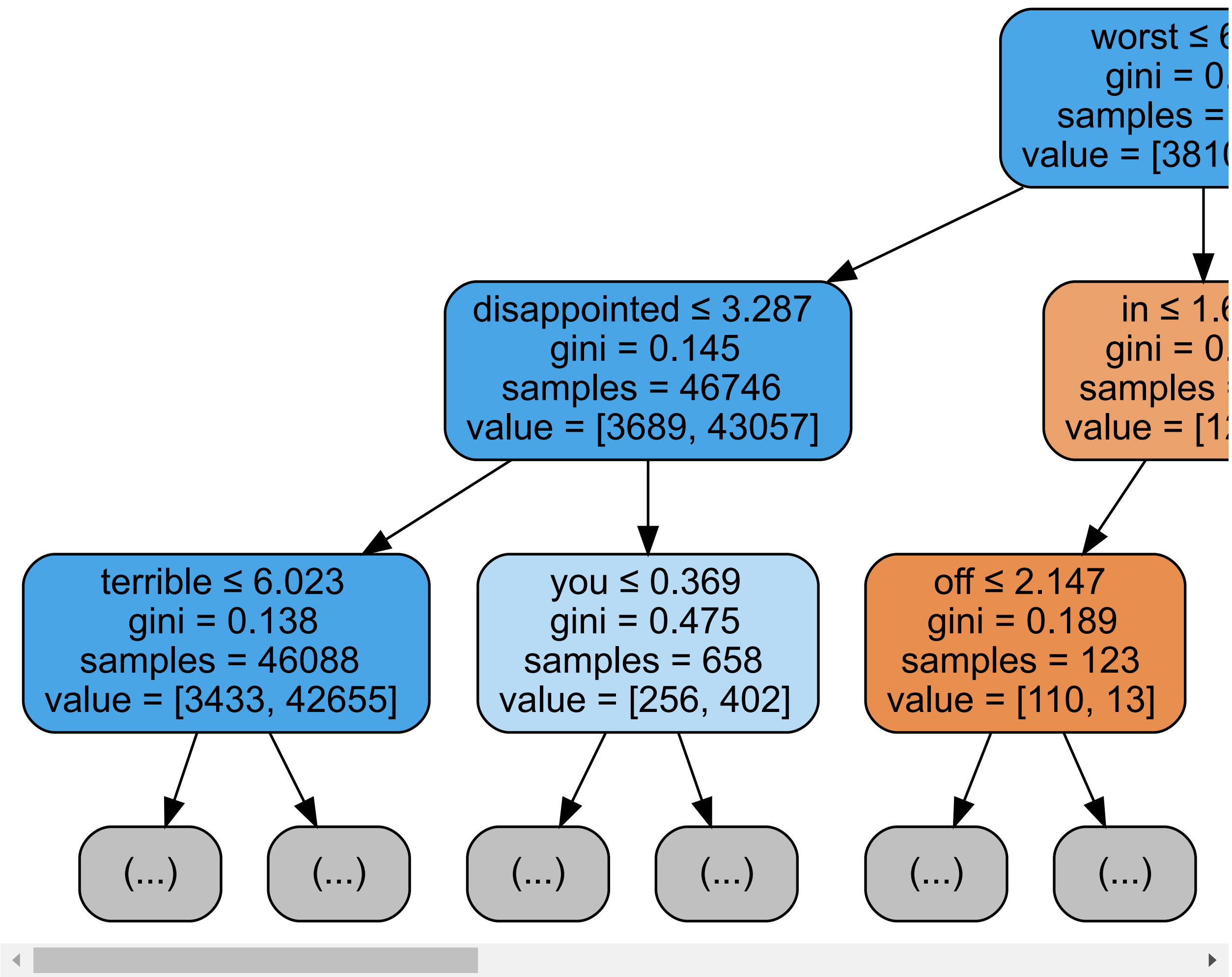


In [49]:
```
def most_informative_feature_for_binary_classification(vectorizer, classifier, n
    class_labels = classifier.classes_
    feature_names = vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.feature_importances_, feature_names))[:n
    topn_class2 = sorted(zip(classifier.feature_importances_, feature_names))[-n
    #print(dict(zip(iris_pd.columns, clf.feature_importances_)))

    print("Top 20 important features:\n ")
    for coef, feat in reversed(topn_class2):
        print (coef,"--> ", feat)


most_informative_feature_for_binary_classification(count_vect, clf_dtree_best)
```

```
Top 20 important features:

0.13675912498792062 -->  not
0.07565956141379283 -->  great
0.0721215165990680З -->  worst
0.06566557770479665 -->  disappointed
0.06231627614015801 -->  was
0.0468568030225928 -->  money
0.04382670380998059 -->  awful
0.042428174003224954 -->  terrible
0.034679778079664686 -->  horrible
0.031638564349793195 -->  waste
0.027899457864880444 -->  best
0.026820522891220576 -->  disappointing
0.025529565497729605 -->  delicious
0.02261988297114902 -->  bad
0.021128847276721156 -->  and
0.02033381360079511 -->  disappointment
0.018377359046007857 -->  find
0.017325097116191065 -->  good
0.017150603179352856 -->  you
0.014484344991899347 -->  love
```

In [50]:
```
dot_data = tree.export_graphviz(clf_dtree_best, out_file=None,
                      feature_names=count_vect.get_feature_names(),
                      filled=True, rounded=True,
                      max_depth = 3,
                      special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[50]:



TDIDF

In [78]:
```
#tf-idf on train data
tf_idf_vect = TfidfVectorizer(ngram_range=(1,1)) #considering only uni-gram as I
train_tfidf_nstd = tf_idf_vect.fit_transform(X_train[:,9]) #sparse matrix
test_tfidf_nstd = tf_idf_vect.transform(X_test[:,9])
print(train_tfidf_nstd.shape)
print(test_tfidf_nstd.shape)
```

```
(70000, 50158)
(30000, 50158)
```

In [79]:
```
# Column Standardization of the tfidf non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(train_tfidf_nstd)
train_tfidf = std_scal.transform(train_tfidf_nstd)
test_tfidf = std_scal.transform(test_tfidf_nstd)
```

In [80]:
```
clf_dtree_tfidf = tree.DecisionTreeClassifier()
clf_dtree_tfidf = clf_dtree_tfidf.fit(train_tfidf, y_train)
clf_dtree_tfidf
```

Out[80]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [82]:
```
parameter = {
        'max_depth': (10, 50, 100, 500),
        'min_samples_split': (5, 10, 50, 100)
    }

gsearch_dt_tfidf = GridSearchCV(estimator = clf_dtree_tfidf,
                          param_grid= parameter,
                          cv=3,
                          scoring='f1')
gsearch_dt_tfidf.fit(train_tfidf, y_train)

print(gsearch_dt_tfidf)
results_tfidf = gsearch_dt_tfidf.cv_results_

# summarize the results of the grid search
print("\nBest score: ",gsearch_dt_tfidf.best_score_)
NB_OPTIMAL_clf_tfidf = gsearch_dt_tfidf.best_estimator_

best_max_depth_tfidf = gsearch_dt_tfidf.best_estimator_.max_depth
print("\nOptimal value of Hyperparameter, max_depth : ",best_max_depth_tfidf)

best_min_samples_split_tfidf = gsearch_dt_tfidf.best_estimator_.min_samples_spli
print("\nOptimal value of Hyperparameter, min_samples_split : ",best_min_samples_
```

```
GridSearchCV(cv=3, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': (10, 50, 100, 500), 'min_samples_split': (5, 1
0, 50, 100)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)

Best score:  0.9406953146056071

Optimal value of Hyperparameter, max_depth :  10

Optimal value of Hyperparameter, min_samples_split :  100
```

In [83]:
```
joblib.dump(results_tfidf,"results_tfidf.pkl")
```

Out[83]:
```
['results_tfidf.pkl']
```

In [84]:
```
results_tfidf = joblib.load("results_tfidf.pkl")
results_tfidf
```

Out[84]:
```
{'mean_fit_time': array([ 23.21834135,  22.94935322,  21.895353  ,  21.0510441
5,
        101.5016168 , 111.03950222, 109.17370653, 126.04858645,
        198.12915691, 182.98127143, 164.11317499, 152.27093101,
        236.86794758, 248.61721786, 193.44268179, 197.97029328]),
 'std_fit_time': array([ 0.62632641,  0.66681435,  0.47643973,  0.44130969,  0.
72555859,
        12.72471449, 12.31967425, 14.23149979, 17.76721227, 17.63056785,
        12.44513891,  1.89577197,  8.31369432, 16.71417197,  9.30418931,
         2.26798842]),
 'mean_score_time': array([0.19200667, 0.1943326 , 0.19500383, 0.19233147, 0.21
832633,
        0.26770425, 0.23200425, 0.24234343, 0.27033631, 0.25466291,
        0.23966281, 0.23534139, 0.25633709, 0.25499527, 0.22766511,
        0.23599792]),
 'std_score_time': array([0.00489834, 0.00329913, 0.00293429, 0.00579326, 0.013
88622,
        0.0880107 , 0.02031939, 0.01810829, 0.01692974, 0.03493051,
        0.00872895, 0.0103956 , 0.02071435, 0.01639093, 0.00419114,
        0.01202712]),
 'param_max_depth': masked_array(data=[10, 10, 10, 10, 50, 50, 50, 50, 100, 10
0, 100, 100,
                    500, 500, 500, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 50, 100, 5, 10, 50, 100,
5, 10, 50, 100, 5, 10,
                    50, 100],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 50},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 50},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 100, 'min_samples_split': 5},
  {'max_depth': 100, 'min_samples_split': 10},
  {'max_depth': 100, 'min_samples_split': 50},
  {'max_depth': 100, 'min_samples_split': 100},
  {'max_depth': 500, 'min_samples_split': 5},
  {'max_depth': 500, 'min_samples_split': 10},
  {'max_depth': 500, 'min_samples_split': 50},
  {'max_depth': 500, 'min_samples_split': 100}],
 'split0_test_score': array([0.94165643, 0.94136189, 0.9414192 , 0.94143298, 0.
93148462,
        0.93126524, 0.93214012, 0.93184433, 0.92745466, 0.9267506 ,
        0.92871239, 0.92917601, 0.92222571, 0.92328542, 0.92550594,
```

```
        0.92597617]),
 'split1_test_score': array([0.94045554, 0.94070031, 0.94079845, 0.94102671, 0.
92853051,
        0.92913311, 0.92924167, 0.930614  , 0.92539248, 0.92412498,
        0.9246727 , 0.92644049, 0.92144374, 0.92      , 0.92197622,
        0.92310293]),
 'split2_test_score': array([0.93941847, 0.93959514, 0.93948087, 0.93962622, 0.
9268666 ,
        0.92568829, 0.92693681, 0.92823608, 0.9218897 , 0.92145081,
        0.91976384, 0.92281709, 0.91842271, 0.91633312, 0.91725459,
        0.9189294 ]),
 'mean_test_score': array([0.94051016, 0.94055246, 0.94056618, 0.94069531, 0.92
896061,
        0.92869558, 0.92943957, 0.93023149, 0.92491232, 0.92410883,
        0.92438304, 0.92614457, 0.92069741, 0.91987289, 0.92157897,
        0.92266955]),
 'std_test_score': array([0.00091446, 0.00072881, 0.00080819, 0.00077393, 0.001
90968,
        0.00229771, 0.00212886, 0.00149769, 0.00229713, 0.00216367,
        0.00365898, 0.00260445, 0.00163981, 0.0028397 , 0.0033803 ,
        0.00289311]),
 'rank_test_score': array([ 4,  3,  2,  1,  7,  8,  6,  5, 10, 12, 11,  9, 15,
16, 14, 13]),
 'split0_train_score': array([0.9498227 , 0.94957409, 0.94794806, 0.94621211,
0.9848256 ,
        0.98110762, 0.96986242, 0.96526834, 0.99292479, 0.98773452,
        0.97644322, 0.970347  , 0.99679027, 0.99174218, 0.98041748,
        0.97478376]),
 'split1_train_score': array([0.9486663 , 0.94828385, 0.94703588, 0.94604212,
0.98469424,
        0.98087628, 0.9690427 , 0.96430071, 0.99191804, 0.98755267,
        0.97440678, 0.969268  , 0.99714386, 0.99214363, 0.97839747,
        0.97285073]),
 'split2_train_score': array([0.94948256, 0.94912273, 0.94814352, 0.94648841,
0.98421097,
        0.98102754, 0.97001116, 0.96407157, 0.99260548, 0.98826663,
        0.9759595 , 0.96968597, 0.99720501, 0.99236307, 0.98003285,
        0.97432715]),
 'mean_train_score': array([0.94932385, 0.94899355, 0.94770916, 0.94624755, 0.9
8457694,
        0.98100381, 0.96963876, 0.96454687, 0.99248277, 0.98785128,
        0.97560317, 0.96976699, 0.99704638, 0.99208296, 0.97961593,
        0.97398722]),
 'std_train_score': array([4.85254149e-04, 5.34597936e-04, 4.82715086e-04, 1.83
911117e-04,
        2.64278390e-04, 9.59196336e-05, 4.25829367e-04, 5.18658146e-04,
        4.20059384e-04, 3.02940480e-04, 8.68715421e-04, 4.44211841e-04,
        1.82807821e-04, 2.57080018e-04, 8.75778768e-04, 8.24952311e-04])}
```

In [85]:
```
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_train_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

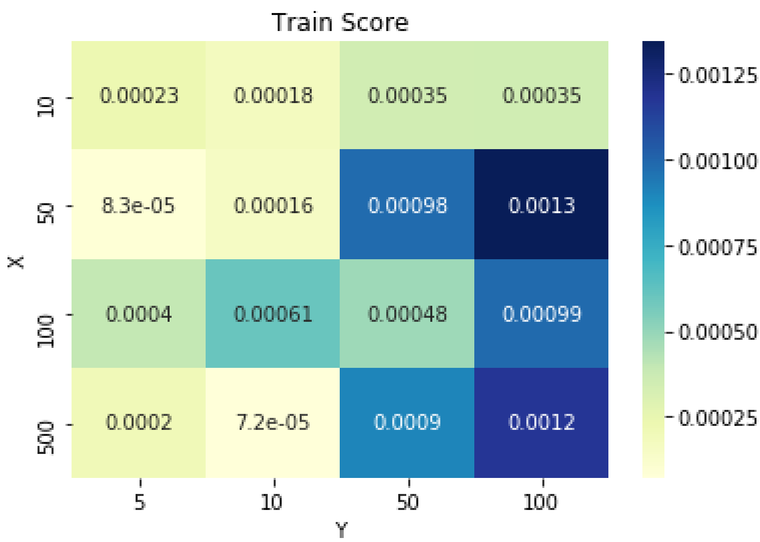
```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

Out[85]:

	X	Y	Z
0	10	5	0.000229
1	10	10	0.000178
2	10	50	0.000349
3	10	100	0.000353
4	50	5	0.000083
5	50	10	0.000160
6	50	50	0.000984
7	50	100	0.001343
8	100	5	0.000395
9	100	10	0.000606
10	100	50	0.000481
11	100	100	0.000988
12	500	5	0.000204
13	500	10	0.000072
14	500	50	0.000901
15	500	100	0.001175

In [86]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Train Score')
```

Out[86]: Text(0.5, 1.0, 'Train Score')



In [87]:
```
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_test_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

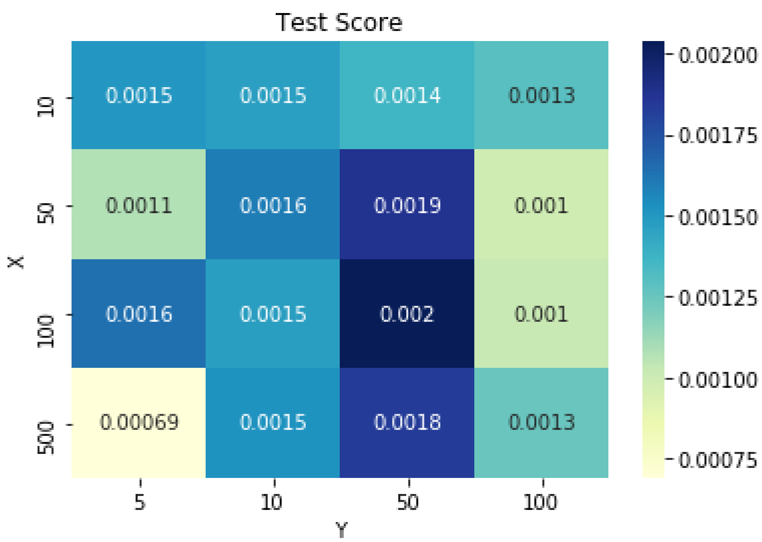
```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

Out[87]:

	X	Y	Z
0	10	5	0.001504
1	10	10	0.001468
2	10	50	0.001364
3	10	100	0.001299
4	50	5	0.001078
5	50	10	0.001597
6	50	50	0.001881
7	50	100	0.000995
8	100	5	0.001640
9	100	10	0.001469
10	100	50	0.002035
11	100	100	0.001026
12	500	5	0.000692
13	500	10	0.001521
14	500	50	0.001834
15	500	100	0.001253

In [88]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Test Score')
```

Out[88]:
```
Text(0.5, 1.0, 'Test Score')
```



Decision Tree on TFIDF with Best Parameters

In [10]:
```
clf_dtree_tfidf_best = tree.DecisionTreeClassifier(max_depth=10, min_samples_spl
clf_dtree_tfidf_best.fit(train_tfidf, y_train)

y_pred_test_tfidf = clf_dtree_tfidf_best.predict(test_tfidf)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_t
roc_auc_tfidf_best = auc(false_positive_rate, true_positive_rate)

joblib.dump(clf_dtree_tfidf_best,"clf_dtree_tfidf_best.pkl")
joblib.dump(y_pred_test_tfidf,"y_pred_test_tfidf.pkl")
joblib.dump(roc_auc_tfidf_best,"roc_auc_tfidf_best.pkl")
```

Out[10]:
```
['roc_auc_tfidf_best.pkl']
```

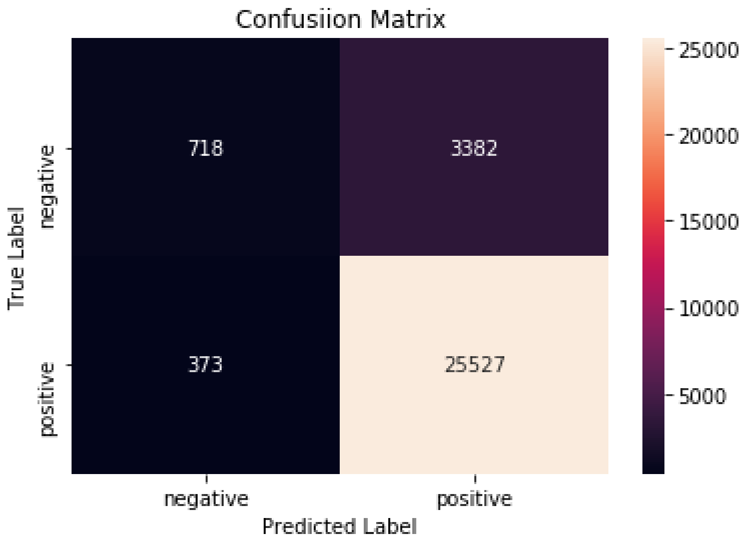
In [16]:
```
clf_dtree_tfidf_best = joblib.load("clf_dtree_tfidf_best.pkl")
y_pred_test_tfidf = joblib.load("y_pred_test_tfidf.pkl")
roc_auc_tfidf_best = joblib.load("roc_auc_tfidf_best.pkl")
roc_auc_tfidf_best
```

Out[16]:
```
0.580360203408984
```

In [17]:
```
# Confusion Matrix on Test Data
#y_pred = np.argmax(pred_test, axis=1)
cm_tfidf = confusion_matrix(y_test, y_pred_test_tfidf)
cm_tfidf
```

Out[17]:
```
array([[  718,  3382],
       [  373, 25527]], dtype=int64)
```

In [18]:
```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_tfidf, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

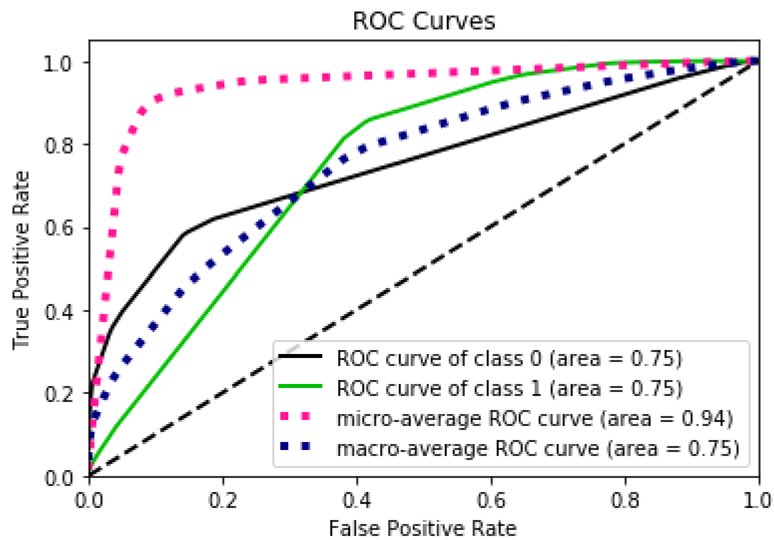


In [20]:
```
y_pred_train_proba_tfidf = clf_dtree_tfidf_best.predict_proba(train_tfidf)
y_pred_test_proba_tfidf = clf_dtree_tfidf_best.predict_proba(test_tfidf)
```

In [21]:
```
#Plotting ROC curve over Train Data
skplt.metrics.plot_roc_curve(y_train,y_pred_train_proba_tfidf)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

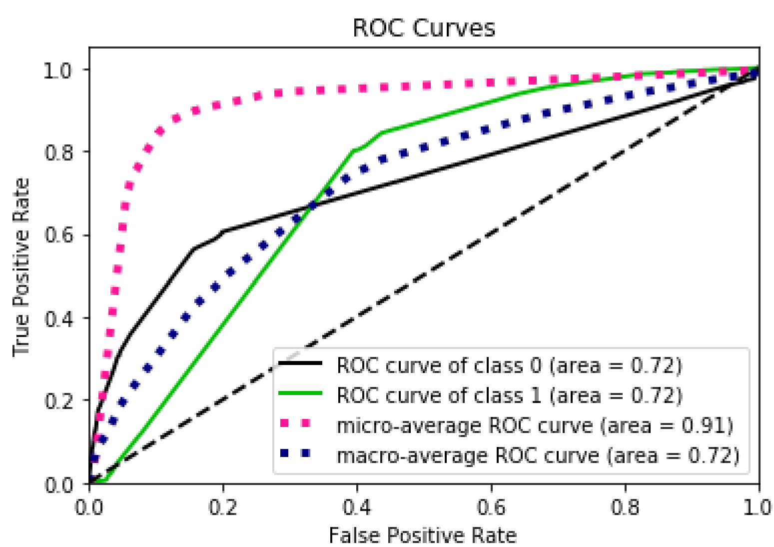
Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x238cff96f60>
```



In [22]:
```
#Plotting ROC curve over Test Data
skplt.metrics.plot_roc_curve(y_test,y_pred_test_proba_tfidf)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x238cffdca90>
```

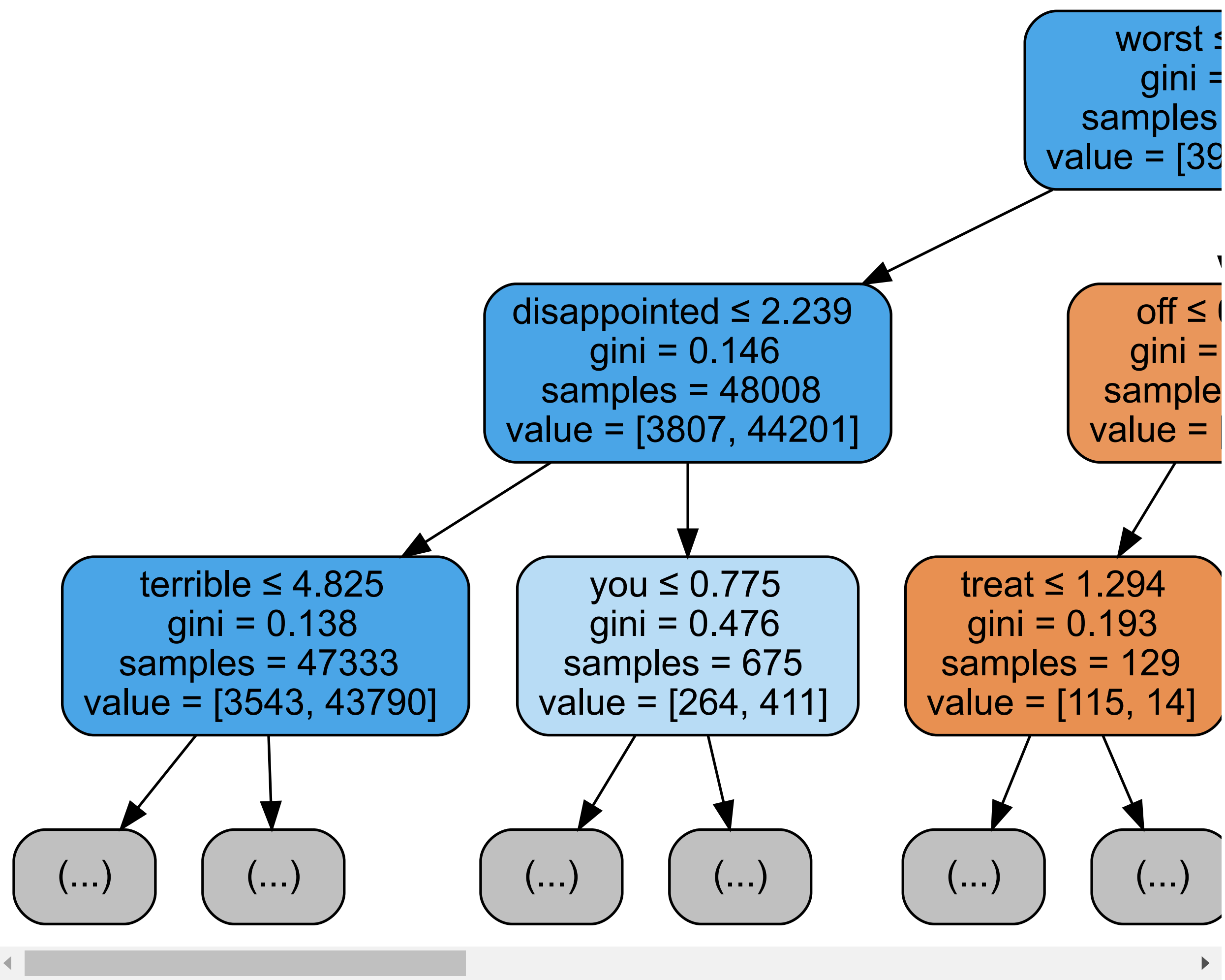


In [24]:
```
def most_informative_feature_for_binary_classification(vectorizer, classifier, n
    class_labels = classifier.classes_
    feature_names = vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.feature_importances_, feature_names))[:n
    topn_class2 = sorted(zip(classifier.feature_importances_, feature_names))[-n
    #print(dict(zip(iris_pd.columns, clf.feature_importances_)))

    print("Top 20 important features:\n ")
    for coef, feat in reversed(topn_class2):
        print (coef,"--> ", feat)


most_informative_feature_for_binary_classification(tf_idf_vect, clf_dtree_tfidf_l
```

```
Top 20 important features:

0.14057903019407447 -->  not
0.06801588980901073 -->  was
0.06736153811238069 -->  great
0.06716416874074672 -->  worst
0.05425180173271436 -->  disappointed
0.04404892349196213 -->  awful
0.0394325782602613 -->  terrible
0.03430715220625825 -->  money
0.02895438236575219 -->  best
0.0288257685212685 -->  horrible
0.027650118010705358 -->  waste
0.025572538557227752 -->  delicious
0.022483611636000985 -->  bad
0.021530291947656512 -->  disappointment
0.018453866245291606 -->  disappointing
0.016281934544395527 -->  did
0.016266072762773025 -->  you
0.013295538243558292 -->  find
0.011720581691207339 -->  love
0.01111688871246896 -->  would
```

In [70]:
```
dot_data = tree.export_graphviz(clf_dtree_tfidf_best, out_file=None,
                                feature_names=count_vect.get_feature_names(),
                                filled=True, rounded=True,
                                max_depth = 3,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[70]:



avgW2W

In [91]:
```
fileObject = open("./final_to_file3.pkl",'rb') # we open the file for reading
final = pickle.load(fileObject) # load the object from the file
```

In [92]:
```
#w2v
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sent=[]
for sent in final['CleanedText'].values:
    list_of_sent.append(sent.split())

print(type(list_of_sent))
print(final['CleanedText'].values[0])
print("*******************************************************************")
print(list_of_sent[0])
```

```
<class 'list'>
witti littl book make son laugh loud recit car drive along alway sing refrain h
es learn whale india droop love new word book introduc silli classic book will
bet son still abl recit memori colleg
*****************************************************************
['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'car', 'dri
ve', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn', 'whale', 'india', 'dr
oop', 'love', 'new', 'word', 'book', 'introduc', 'silli', 'classic', 'book', 'w
ill', 'bet', 'son', 'still', 'abl', 'recit', 'memori', 'colleg']
```

In [93]:
```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

In [94]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =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:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
#print(len(sent_vectors[0]))
print(type(sent_vectors))
```

```
100000
<class 'list'>
```

In [95]:
```
# create design matrix X and target vector y
X = np.array(sent_vectors[::]) # end index is exclusive
y = np.array(final['Score']) # showing you two ways of indexing a pandas df
```

In [96]:
```
X_train_nstd = X[0:70000:1]
X_test_nstd = X[70000:100000:1]

y_train_nstd = y[0:70000:1]
y_test_nstd =y[70000:100000:1]

print(X_train_nstd.shape)
print(X_test_nstd.shape)
print(y_train_nstd.shape)
print(y_test_nstd.shape)
```

```
(70000, 50)
(30000, 50)
(70000,)
(30000,)
```

In [97]:
```
# Column Standardization of the tfidf non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(X_train_nstd)
train_avgw2v = std_scal.transform(X_train_nstd)
test_avgw2v = std_scal.transform(X_test_nstd)
```

tfidf-W-W2V

In [98]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [99]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = t

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
row=0;
for sent in (list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # 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:
            vec = 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 += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [100]:
```
print(len(tfidf_sent_vectors))
print(np.shape(tfidf_sent_vectors))
print(type(tfidf_sent_vectors))
```

```
100000
(100000, 50)
<class 'list'>
```

In [101]:
```
# create design matrix X and target vector y
X = np.array(tfidf_sent_vectors[::]) # end index is exclusive
y = np.array(final['Score']) # showing you two ways of indexing a pandas df
```

In [102]:
```
#taking 40K data into consideration
X_train_nstd = X[0:70000:1]
X_test_nstd = X[70000:100000:1]

y_train_nstd = y[0:70000:1]
y_test_nstd =y[70000:100000:1]

print(X_train_nstd.shape)
print(X_test_nstd.shape)
print(y_train_nstd.shape)
print(y_test_nstd.shape)
```

```
(70000, 50)
(30000, 50)
(70000,)
(30000,)
```

In [103]:
```
# Column Standardization of the tfidf non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(X_train_nstd)
train_tfidfww2v = std_scal.transform(X_train_nstd)
test_tfidfww2v = std_scal.transform(X_test_nstd)
```

Decision tree on avgw2v

In [104]:
```
clf_dtree_avgw2v = tree.DecisionTreeClassifier()
clf_dtree_avgw2v = clf_dtree_avgw2v.fit(train_avgw2v, y_train)
clf_dtree_avgw2v
```

Out[104]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [105]:
```
parameter = {
        'max_depth': (10, 50, 100, 500),
        'min_samples_split': (5, 10, 50, 100)
    }

gsearch_dt_avgw2v = GridSearchCV(estimator = clf_dtree_avgw2v,
                          param_grid= parameter,
                          cv=3,
                          scoring='f1')
gsearch_dt_avgw2v.fit(train_avgw2v, y_train)

print(gsearch_dt_avgw2v)
results_avgw2v = gsearch_dt_avgw2v.cv_results_

# summarize the results of the grid search
print("\nBest score: ",gsearch_dt_avgw2v.best_score_)
NB_OPTIMAL_clf_avgw2v = gsearch_dt_avgw2v.best_estimator_

best_max_depth_avgw2v = gsearch_dt_avgw2v.best_estimator_.max_depth
print("\nOptimal value of Hyperparameter, max_depth : ",best_max_depth_avgw2v)

best_min_samples_split_avgw2v = gsearch_dt_avgw2v.best_estimator_.min_samples_sp
print("\nOptimal value of Hyperparameter, min_samples_split : ",best_min_samples_
```

```
GridSearchCV(cv=3, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': (10, 50, 100, 500), 'min_samples_split': (5, 1
0, 50, 100)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)

Best score:  0.9339276150267604

Optimal value of Hyperparameter, max_depth :  10

Optimal value of Hyperparameter, min_samples_split :  100
```

In [106]:
```
joblib.dump(results_avgw2v,"results_avgw2v.pkl")
```

Out[106]:
```
['results_avgw2v.pkl']
```

In [107]:
```
results_avgw2v = joblib.load("results_avgw2v.pkl")
results_avgw2v
```

Out[107]:
```
{'mean_fit_time': array([ 9.67353535,  9.64587641,  9.60054286,  9.66454252, 1
5.72911191,
        15.92145435, 15.54411936, 15.25178615, 15.72878416, 15.68577886,
        15.42479499, 15.09579301, 15.71378469, 15.79211775, 15.38444805,
        15.23078354]),
 'std_fit_time': array([0.0521318 , 0.01124283, 0.02904043, 0.09989732, 0.02941
685,
        0.2005213 , 0.05197705, 0.02513229, 0.09296609, 0.04558629,
        0.06668414, 0.02271469, 0.01791504, 0.1870347 , 0.08803648,
        0.20450028]),
 'mean_score_time': array([0.03633261, 0.03665423, 0.03766783, 0.03633356, 0.04
400015,
        0.04200157, 0.04099997, 0.04067755, 0.04233281, 0.04266715,
        0.04166071, 0.04132549, 0.04232748, 0.04300618, 0.04200149,
        0.04133272]),
 'std_score_time': array([4.71034627e-04, 9.37212143e-04, 2.35600671e-03, 4.869
36587e-04,
        1.41411111e-03, 1.36730278e-06, 8.10467325e-07, 4.68070019e-04,
        4.71595499e-04, 4.72890535e-04, 4.65492410e-04, 1.23963243e-03,
        4.75154943e-04, 9.87256378e-06, 1.78416128e-06, 4.74356437e-04]),
 'param_max_depth': masked_array(data=[10, 10, 10, 10, 50, 50, 50, 50, 100, 10
0, 100, 100,
                    500, 500, 500, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 50, 100, 5, 10, 50, 100,
5, 10, 50, 100, 5, 10,
                    50, 100],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 50},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 50},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 100, 'min_samples_split': 5},
  {'max_depth': 100, 'min_samples_split': 10},
  {'max_depth': 100, 'min_samples_split': 50},
  {'max_depth': 100, 'min_samples_split': 100},
  {'max_depth': 500, 'min_samples_split': 5},
  {'max_depth': 500, 'min_samples_split': 10},
  {'max_depth': 500, 'min_samples_split': 50},
  {'max_depth': 500, 'min_samples_split': 100}],
 'split0_test_score': array([0.93267792, 0.93174256, 0.93367492, 0.93416972, 0.
90804372,
        0.90859214, 0.91940861, 0.92761996, 0.90838293, 0.90847557,
        0.91924805, 0.92721149, 0.90693399, 0.90757085, 0.91904566,
```

```
        0.92740819]),
 'split1_test_score': array([0.93233011, 0.93265306, 0.93268228, 0.93386243, 0.
91075827,
        0.91287196, 0.92373859, 0.92774525, 0.91130664, 0.9137545 ,
        0.923528  , 0.92808871, 0.911693  , 0.91334096, 0.92339863,
        0.92763972]),
 'split2_test_score': array([0.92760127, 0.92725796, 0.92961234, 0.93375068, 0.
90506064,
        0.90658356, 0.91814534, 0.92610205, 0.90513921, 0.90558814,
        0.91846303, 0.92643389, 0.90564092, 0.90548339, 0.91824994,
        0.92630278]),
 'mean_test_score': array([0.93086979, 0.93055121, 0.93198987, 0.93392762, 0.90
795422,
        0.90934921, 0.92043083, 0.92715576, 0.90827626, 0.90927273,
        0.92041301, 0.92724469, 0.90808929, 0.90879838, 0.92023139,
        0.9271169 ]),
 'std_test_score': array([0.00231553, 0.00235813, 0.0017293 , 0.00017718, 0.002
32689,
        0.00262243, 0.0023951 , 0.00074683, 0.00251895, 0.0033812 ,
        0.0022258 , 0.00067598, 0.00260229, 0.00332318, 0.00226299,
        0.00058337]),
 'rank_test_score': array([ 3,  4,  2,  1, 16, 11,  8,  6, 14, 12,  9,  5, 15,
13, 10,  7]),
 'split0_train_score': array([0.95940077, 0.95841669, 0.95318042, 0.9493514 ,
0.99498305,
        0.98625776, 0.96077913, 0.95257237, 0.99491005, 0.98638059,
        0.96078314, 0.95265864, 0.99498354, 0.98618687, 0.96087761,
        0.95257016]),
 'split1_train_score': array([0.95917567, 0.95883605, 0.95357618, 0.95016556,
0.9948354 ,
        0.98725999, 0.96256015, 0.95441033, 0.99490831, 0.98730857,
        0.96255398, 0.95430386, 0.99498171, 0.98723384, 0.9626546 ,
        0.95440975]),
 'split2_train_score': array([0.96072508, 0.95976868, 0.95434966, 0.95039804,
0.99517402,
        0.98684018, 0.96224737, 0.9542059 , 0.9950891 , 0.98679282,
        0.96227866, 0.95419458, 0.99508946, 0.98664364, 0.96233375,
        0.95417783]),
 'mean_train_score': array([0.95976717, 0.95900714, 0.95370209, 0.94997167, 0.9
9499749,
        0.98678598, 0.96186222, 0.95372953, 0.99496915, 0.98682733,
        0.96187192, 0.95371902, 0.99501824, 0.98668811, 0.96195532,
        0.95371925]),
 'std_train_score': array([6.83545186e-04, 5.65052015e-04, 4.85570589e-04, 4.48
746577e-04,
        1.38614894e-04, 4.10948156e-04, 7.76430386e-04, 8.22484357e-04,
        8.48190304e-05, 3.79629328e-04, 7.78051224e-04, 7.51133050e-04,
        5.03666307e-05, 4.28576749e-04, 7.73231570e-04, 8.18027600e-04])}
```

In [108]:
```
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_train_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

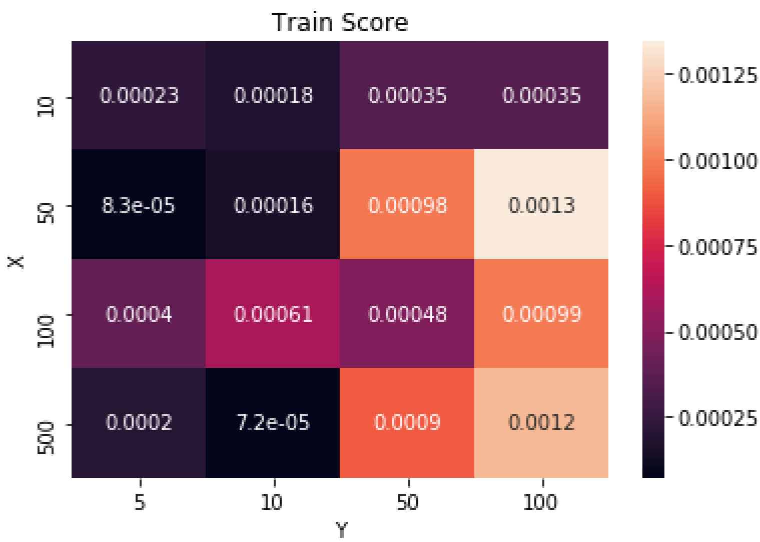
```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

Out[108]:

	X	Y	Z
0	10	5	0.000229
1	10	10	0.000178
2	10	50	0.000349
3	10	100	0.000353
4	50	5	0.000083
5	50	10	0.000160
6	50	50	0.000984
7	50	100	0.001343
8	100	5	0.000395
9	100	10	0.000606
10	100	50	0.000481
11	100	100	0.000988
12	500	5	0.000204
13	500	10	0.000072
14	500	50	0.000901
15	500	100	0.001175

In [110]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True)
ax.set_title('Train Score')
```

Out[110]: Text(0.5, 1.0, 'Train Score')



In [111]:
```
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_train_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

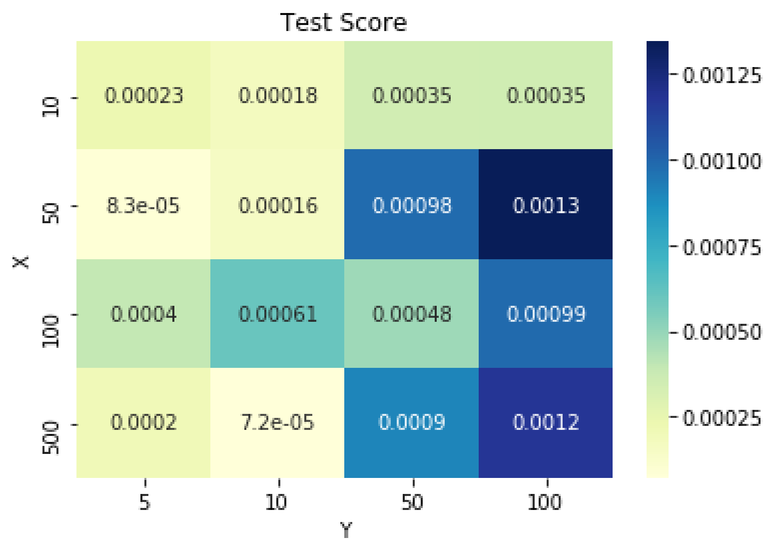
```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

Out[111]:

	X	Y	Z
0	10	5	0.000229
1	10	10	0.000178
2	10	50	0.000349
3	10	100	0.000353
4	50	5	0.000083
5	50	10	0.000160
6	50	50	0.000984
7	50	100	0.001343
8	100	5	0.000395
9	100	10	0.000606
10	100	50	0.000481
11	100	100	0.000988
12	500	5	0.000204
13	500	10	0.000072
14	500	50	0.000901
15	500	100	0.001175

In [112]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Test Score')
```

Out[112]: Text(0.5, 1.0, 'Test Score')



Decision tree n avgW2V with best parameters

In [49]:
```
clf_dtree_avgw2v_best = tree.DecisionTreeClassifier(max_depth=10, min_samples_sp
clf_dtree_avgw2v_best.fit(train_avgw2v, y_train)

y_pred_test_avgw2v = clf_dtree_avgw2v_best.predict(test_avgw2v)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_t
roc_auc_avgw2v_best = auc(false_positive_rate, true_positive_rate)

joblib.dump(clf_dtree_avgw2v_best,"clf_dtree_avgw2v_best.pkl")
joblib.dump(y_pred_test_avgw2v,"y_pred_test_avgw2v.pkl")
joblib.dump(roc_auc_avgw2v_best,"roc_auc_avgw2v_best.pkl")
```

Out[49]: ['roc_auc_avgw2v_best.pkl']

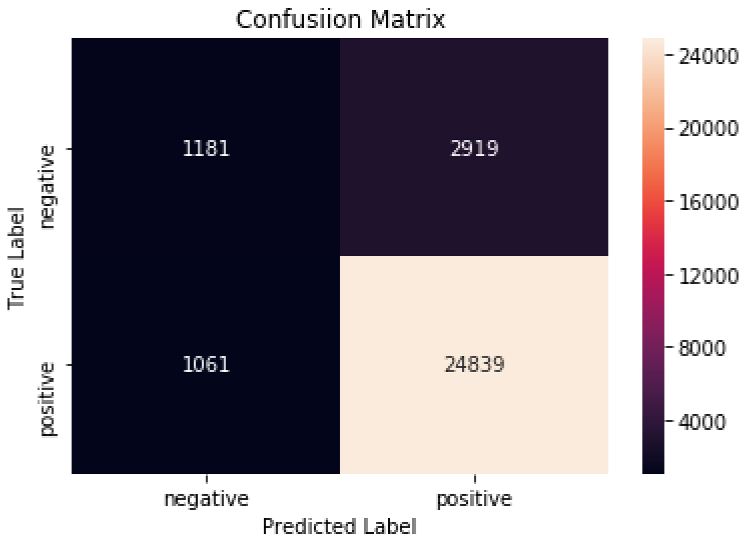
In [50]:
```
clf_dtree_avgw2v_best = joblib.load("clf_dtree_avgw2v_best.pkl")
y_pred_test_avgw2v = joblib.load("y_pred_test_avgw2v.pkl")
roc_auc_avgw2v_best = joblib.load("roc_auc_avgw2v_best.pkl")
roc_auc_avgw2v_best
```

Out[50]: 0.623541764761277

In [51]:
```
# Confusion Matrix on Test Data
#y_pred = np.argmax(pred_test, axis=1)
cm_avgw2v = confusion_matrix(y_test, y_pred_test_avgw2v)
cm_avgw2v
```

Out[51]:
```
array([[ 1181,  2919],
       [ 1061, 24839]], dtype=int64)
```

In [52]:
```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_avgw2v, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

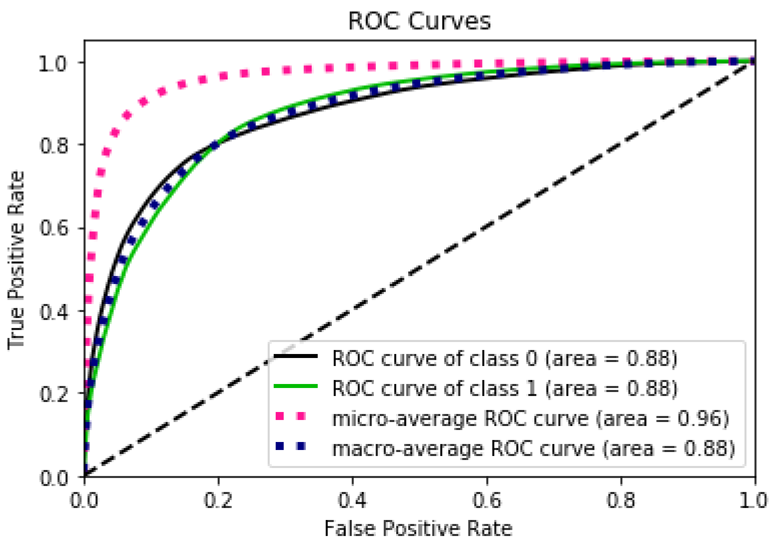


In [53]:
```
y_pred_train_proba_avgw2v = clf_dtree_avgw2v_best.predict_proba(train_avgw2v)
y_pred_test_proba_avgw2v = clf_dtree_avgw2v_best.predict_proba(test_avgw2v)
```

In [54]:
```
#Plotting ROC curve over Train Data
skplt.metrics.plot_roc_curve(y_train,y_pred_train_proba_avgw2v)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

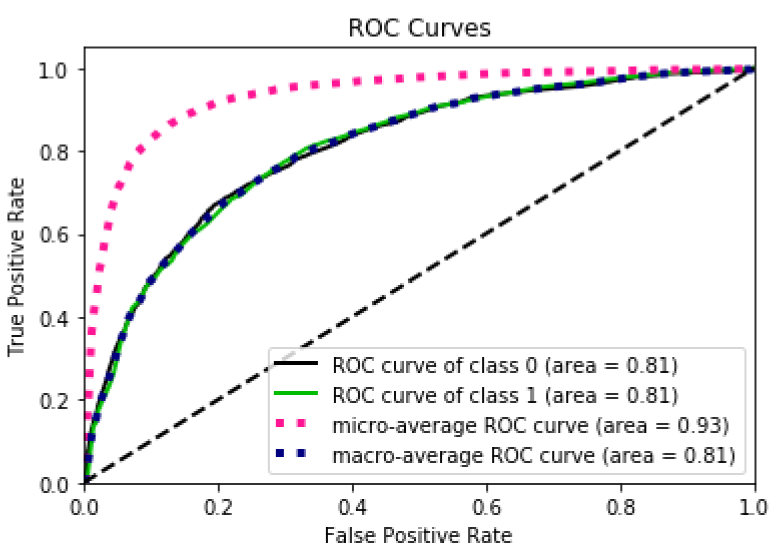
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x238803db048>



In [55]:
```
#Plotting ROC curve over Test Data
skplt.metrics.plot_roc_curve(y_test,y_pred_test_proba_avgw2v)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x238863bbe80>



Decision tree on avgW2V

In [114]:
```
clf_dtree_tfidfww2v = tree.DecisionTreeClassifier()
clf_dtree_tfidfww2v = clf_dtree_tfidfww2v.fit(train_tfidfww2v, y_train)
clf_dtree_tfidfww2v
```

Out[114]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [115]:
```
parameter = {
        'max_depth': (10, 50, 100, 500),
        'min_samples_split': (5, 10, 50, 100)
    }

gsearch_dt_tfidfww2v = GridSearchCV(estimator = clf_dtree_tfidfww2v,
                          param_grid= parameter,
                          cv=3,
                          scoring='f1')
gsearch_dt_tfidfww2v.fit(train_tfidfww2v, y_train)

print(gsearch_dt_tfidfww2v)
results_tfidfww2v = gsearch_dt_tfidfww2v.cv_results_

# summarize the results of the grid search
print("\nBest score: ",gsearch_dt_tfidfww2v.best_score_)
NB_OPTIMAL_clf_tfidfww2v = gsearch_dt_tfidfww2v.best_estimator_

best_max_depth_tfidfww2v = gsearch_dt_tfidfww2v.best_estimator_.max_depth
print("\nOptimal value of Hyperparameter, max_depth : ",best_max_depth_avgw2v)

best_min_samples_split_tfidfww2v = gsearch_dt_tfidfww2v.best_estimator_.min_samp
print("\nOptimal value of Hyperparameter, min_samples_split : ",best_min_samples_
```

```
GridSearchCV(cv=3, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': (10, 50, 100, 500), 'min_samples_split': (5, 1
0, 50, 100)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)

Best score:  0.9325425934731919

Optimal value of Hyperparameter, max_depth :  10

Optimal value of Hyperparameter, min_samples_split :  100
```

In [116]:
```
joblib.dump(results_tfidfww2v,"results_tfidfww2v.pkl")
```

Out[116]:
```
['results_tfidfww2v.pkl']
```

In [117]:
```
results_tfidfww2v = joblib.load("results_tfidfww2v.pkl")
results_tfidfww2v
```

Out[117]:
```
{'mean_fit_time': array([11.4971749 , 10.17852147, 11.56268986, 10.60651414, 1
8.02674754,
        20.75989413, 22.00454299, 19.30385288, 20.54829375, 19.843069  ,
        19.61579458, 18.9970003 , 20.56338533, 19.86815532, 19.46697593,
        19.08710965]),
 'std_fit_time': array([0.95208276, 0.25577852, 0.23045816, 0.84617185, 0.42394
321,
        0.39921845, 1.54595274, 0.26827836, 0.36000856, 0.62303393,
        0.34115417, 0.30680656, 1.15234423, 0.42790657, 0.33383946,
        0.50872494]),
 'mean_score_time': array([0.04000092, 0.03933922, 0.04466605, 0.03700034, 0.04
567258,
        0.05366858, 0.06066028, 0.05200203, 0.05100083, 0.05066737,
        0.05233264, 0.04901385, 0.04966966, 0.05266627, 0.04967101,
        0.05032818]),
 'std_score_time': array([0.00216333, 0.00170099, 0.00880744, 0.00081585, 0.002
49328,
        0.00590545, 0.01083399, 0.00141422, 0.0021602 , 0.00124632,
        0.00758384, 0.00354946, 0.00170332, 0.00205568, 0.00189078,
        0.0017106 ]),
 'param_max_depth': masked_array(data=[10, 10, 10, 10, 50, 50, 50, 50, 100, 10
0, 100, 100,
                    500, 500, 500, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 50, 100, 5, 10, 50, 100,
5, 10, 50, 100, 5, 10,
                    50, 100],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 50},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 50},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 100, 'min_samples_split': 5},
  {'max_depth': 100, 'min_samples_split': 10},
  {'max_depth': 100, 'min_samples_split': 50},
  {'max_depth': 100, 'min_samples_split': 100},
  {'max_depth': 500, 'min_samples_split': 5},
  {'max_depth': 500, 'min_samples_split': 10},
  {'max_depth': 500, 'min_samples_split': 50},
  {'max_depth': 500, 'min_samples_split': 100}],
 'split0_test_score': array([0.92881276, 0.92879418, 0.93039706, 0.93329897, 0.
90230575,
        0.90415789, 0.91633293, 0.92592329, 0.90171454, 0.90403113,
        0.91653612, 0.9258864 , 0.90141398, 0.90367467, 0.91634504,
```

```
        0.92583193]),
 'split1_test_score': array([0.93105486, 0.93137811, 0.93215743, 0.93305754, 0.
90662629,
        0.90634766, 0.9192687 , 0.92754105, 0.90732374, 0.90538723,
        0.92027017, 0.92743581, 0.90562902, 0.90681652, 0.91961322,
        0.92714069]),
 'split2_test_score': array([0.92973049, 0.92962187, 0.93058884, 0.93127124, 0.
89868101,
        0.89995829, 0.91438761, 0.92336502, 0.89758784, 0.89945012,
        0.91461622, 0.92343794, 0.89956289, 0.89990921, 0.91473167,
        0.92335412]),
 'mean_test_score': array([0.92986602, 0.92993137, 0.93104777, 0.93254259, 0.90
253768,
        0.90348795, 0.91666308, 0.92560979, 0.9022087 , 0.90295618,
        0.91714083, 0.92558672, 0.90220195, 0.9034668 , 0.91689664,
        0.92544225]),
 'std_test_score': array([0.00092034, 0.00107735, 0.00078854, 0.00090436, 0.003
24776,
        0.0026511 , 0.00200631, 0.0017192 , 0.00398997, 0.0025402 ,
        0.00234747, 0.00164581, 0.00253838, 0.0028237 , 0.00203068,
        0.00157022]),
 'rank_test_score': array([ 4,  3,  2,  1, 14, 11, 10,  5, 15, 13,  8,  6, 16,
12,  9,  7]),
 'split0_train_score': array([0.95416608, 0.95329505, 0.94856241, 0.94628893,
0.99431307,
        0.98580043, 0.958557  , 0.95153789, 0.99417863, 0.98570769,
        0.95856483, 0.95151881, 0.99425225, 0.98561011, 0.95861696,
        0.95151766]),
 'split1_train_score': array([0.9543261 , 0.95366973, 0.94919569, 0.94684935,
0.9943371 ,
        0.98627439, 0.96013636, 0.95183263, 0.99436099, 0.98629837,
        0.95993474, 0.95177839, 0.99438516, 0.98637529, 0.9601738 ,
        0.95188104]),
 'split2_train_score': array([0.95491047, 0.95404557, 0.95000704, 0.94683041,
0.99482369,
        0.98597574, 0.9604047 , 0.95212194, 0.99477534, 0.98584808,
        0.96037218, 0.95211074, 0.99477521, 0.98590557, 0.9604295 ,
        0.95213206]),
 'mean_train_score': array([0.95446755, 0.95367012, 0.94925505, 0.94665623, 0.9
9449129,
        0.98601685, 0.95969935, 0.95183082, 0.99443832, 0.98595138,
        0.95962392, 0.95180265, 0.99447087, 0.98596366, 0.95974009,
        0.95184359]),
 'std_train_score': array([0.00031993, 0.0003064 , 0.00059126, 0.00025984, 0.00
023525,
        0.00019567, 0.00081516, 0.00023844, 0.00024967, 0.00025197,
        0.00076989, 0.00024226, 0.00022194, 0.00031507, 0.000801  ,
        0.00025222])}
```

In [118]:
```
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_train_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

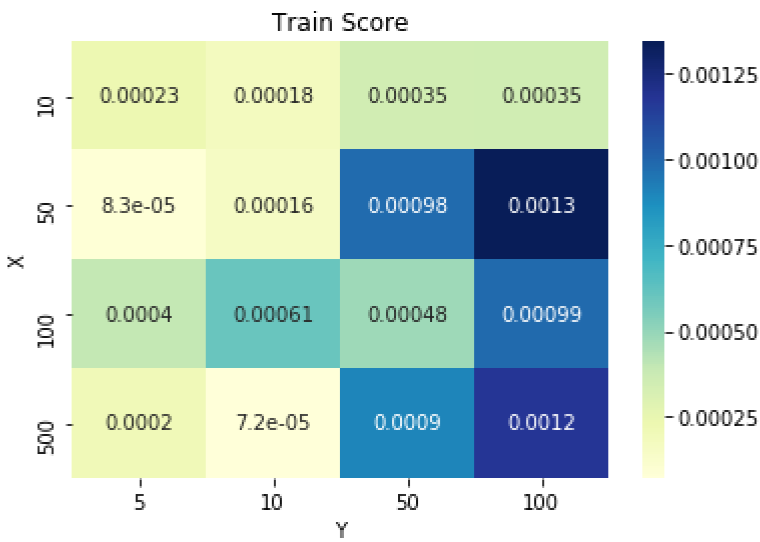
```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

Out[118]:

	X	Y	Z
0	10	5	0.000229
1	10	10	0.000178
2	10	50	0.000349
3	10	100	0.000353
4	50	5	0.000083
5	50	10	0.000160
6	50	50	0.000984
7	50	100	0.001343
8	100	5	0.000395
9	100	10	0.000606
10	100	50	0.000481
11	100	100	0.000988
12	500	5	0.000204
13	500	10	0.000072
14	500	50	0.000901
15	500	100	0.001175

In [119]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Train Score')
```

Out[119]: Text(0.5, 1.0, 'Train Score')



In [120]:
```
X = [10,10,10,10, 50,50,50,50, 100,100,100,100, 500,500,500,500]
Y = [5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,5, 10, 50, 100,]
Z = results['std_train_score']
print(type(X))
X_df = pd.DataFrame(X)
Y_df = pd.DataFrame(Y)
Z_df = pd.DataFrame(Z)
print(type(X_df))
X_df.reindex(columns=[*X_df.columns.tolist(), 'Y'],fill_value=1)
X_df['Y']=Y_df.values
X_df.reindex(columns=[*X_df.columns.tolist(), 'Z'],fill_value=1)
X_df['Z']=Z_df.values
X_df.columns = ['X', 'Y', 'Z']
X_df
```

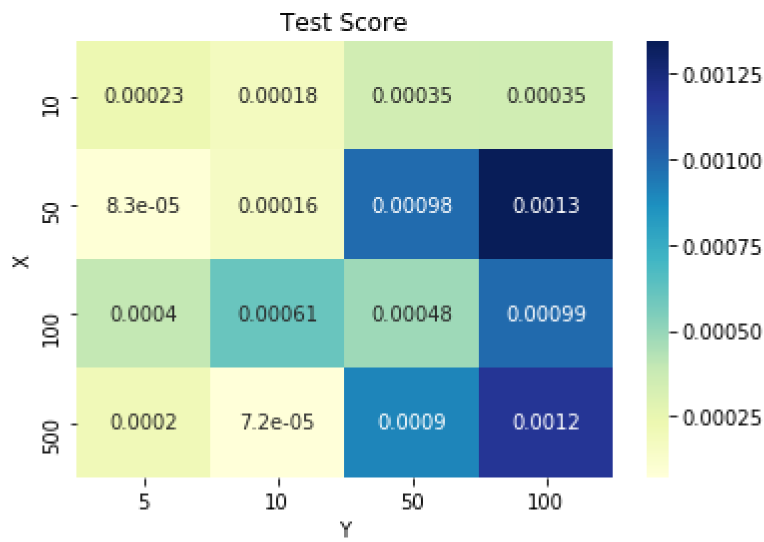
```
<class 'list'>
<class 'pandas.core.frame.DataFrame'>
```

Out[120]:

	X	Y	Z
0	10	5	0.000229
1	10	10	0.000178
2	10	50	0.000349
3	10	100	0.000353
4	50	5	0.000083
5	50	10	0.000160
6	50	50	0.000984
7	50	100	0.001343
8	100	5	0.000395
9	100	10	0.000606
10	100	50	0.000481
11	100	100	0.000988
12	500	5	0.000204
13	500	10	0.000072
14	500	50	0.000901
15	500	100	0.001175

In [121]:
```
plot_data = X_df.pivot("X", "Y", "Z")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Test Score')
```

Out[121]: Text(0.5, 1.0, 'Test Score')



Decision tree on tfidf-W-W2V with best parameters

In [59]:
```
clf_dtree_tfidfww2v_best = tree.DecisionTreeClassifier(max_depth=10, min_samples_
clf_dtree_tfidfww2v_best.fit(train_tfidfww2v, y_train)

y_pred_test_tfidfww2v = clf_dtree_tfidfww2v_best.predict(test_tfidfww2v)

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_te
roc_auc_tfidfww2v_best = auc(false_positive_rate, true_positive_rate)

joblib.dump(clf_dtree_tfidfww2v_best,"clf_dtree_tfidfww2v_best.pkl")
joblib.dump(y_pred_test_tfidfww2v,"y_pred_test_tfidfww2v.pkl")
joblib.dump(roc_auc_tfidfww2v_best,"roc_auc_tfidfww2v_best.pkl")
```

Out[59]: ['roc_auc_tfidfww2v_best.pkl']

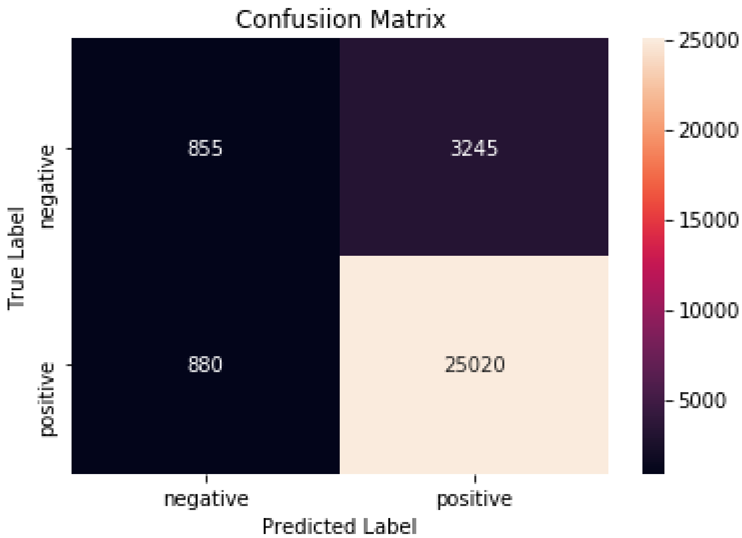
In [60]:
```
clf_dtree_tfidfww2v_best = joblib.load("clf_dtree_tfidfww2v_best.pkl")
y_pred_test_tfidfww2v = joblib.load("y_pred_test_tfidfww2v.pkl")
roc_auc_tfidfww2v_best = joblib.load("roc_auc_tfidfww2v_best.pkl")
roc_auc_tfidfww2v_best
```

Out[60]: 0.5872798756945099

In [61]:
```
# Confusion Matrix on Test Data
#y_pred = np.argmax(pred_test, axis=1)
cm_tfidfww2v = confusion_matrix(y_test, y_pred_test_tfidfww2v)
cm_tfidfww2v
```

Out[61]:
```
array([[  855,  3245],
       [  880, 25020]], dtype=int64)
```

In [62]:
```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm_tfidfww2v, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

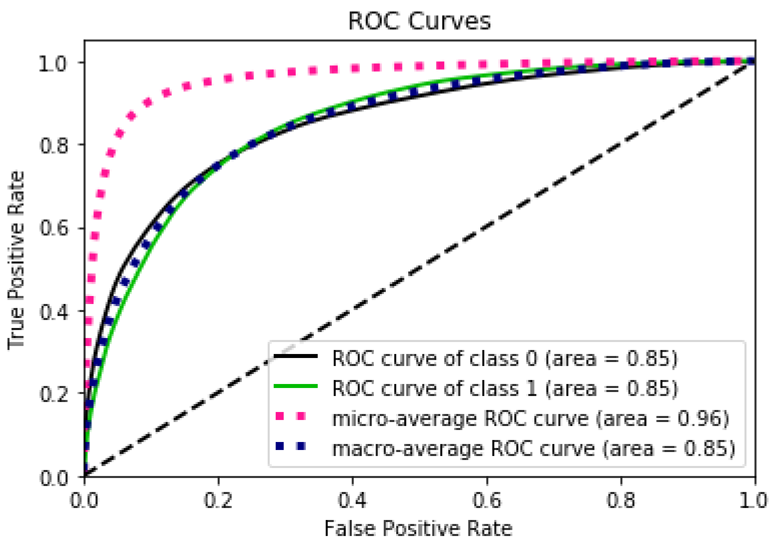


In [63]:
```
y_pred_train_proba_tfidfww2v = clf_dtree_tfidfww2v_best.predict_proba(train_tfid
y_pred_test_proba_tfidfww2v = clf_dtree_tfidfww2v_best.predict_proba(test_tfidfw
```

In [64]:
```
#Plotting ROC curve over Train Data
skplt.metrics.plot_roc_curve(y_train,y_pred_train_proba_tfidfww2v)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

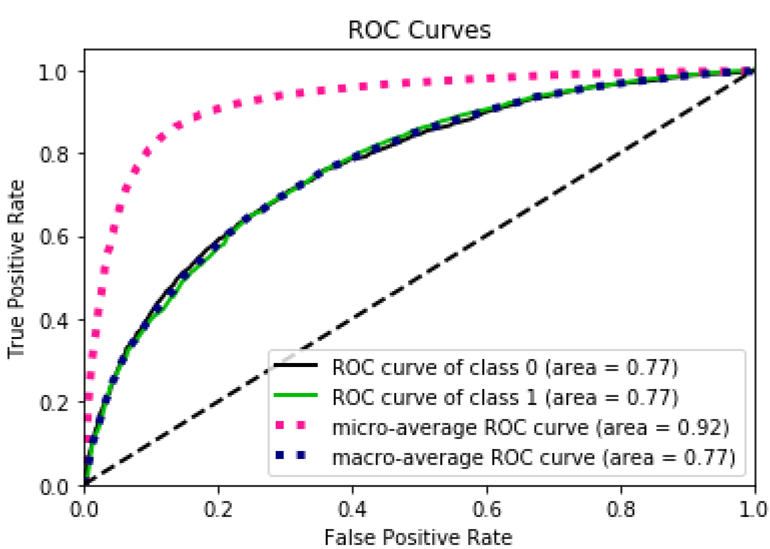
Out[64]: `<matplotlib.axes._subplots.AxesSubplot at 0x238862f0160>`



In [65]:
```
#Plotting ROC curve over Test Data
skplt.metrics.plot_roc_curve(y_test,y_pred_test_proba_tfidfww2v)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77:
DeprecationWarning: Function plot_roc_curve is deprecated; This will be removed
in v0.5.0. Please use scikitplot.metrics.plot_roc instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Out[65]: `<matplotlib.axes._subplots.AxesSubplot at 0x238865179b0>`



In [78]:
```
x = PrettyTable()
x.field_names = ["Paramters/Models","BoW", "TFIDF", "AvgW2V", "TFIDF-W-W2V"]
#x.field_names = ["Kernel = Linear"]

x.add_row(["max_depth : ",best_max_depth_bow,best_max_depth_tfidf,best_max_depth_
x.add_row(["min_samples_split ",best_min_samples_split_bow,best_min_samples_spli
x.add_row(["AUC Score: ",roc_auc_best, roc_auc_tfidf_best, roc_auc_avgw2v_best, r

print(x)
```

```
+--------------------+--------------------+-------------------+----------------
---+--------------------+
|  Paramters/Models  |        BoW         |       TFIDF       |       AvgW2V
|    TFIDF-W-W2V     |
+--------------------+--------------------+-------------------+----------------
---+--------------------+
|    max_depth :     |         10         |         10        |         10
|         10         |
| min_samples_split  |        100         |         5         |        100
|        100         |
|    AUC Score:      | 0.5894844147283171 | 0.580360203408984 | 0.6235417647612
77 | 0.5872798756945099 |
+--------------------+--------------------+-------------------+----------------
---+--------------------+
```

In [ ]: