In [1]:

```
%matplotlib inline

#import all the modules
import sqlite3
import numpy as np
import pandas as pd
import nltk
import seaborn as sns
import pickle
from prettytable import PrettyTable

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn import metrics
#from sklearn.metrics import roc_curve,auc
#from sklearn.manifold import TSNE
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.naive_bayes import BernoulliNB,MultinomialNB
from sklearn.model_selection import train_test_split,GridSearchCV,TimeSeriesSplit
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn import cross_validation
import warnings
warnings.filterwarnings('ignore')
```

```
e:\sofs\python3.6\lib\site-packages\sklearn\cross_validation.py:41: Deprec
ationWarning: 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 moved. Also note that the interface of the new CV iterators are differ
ent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
import pickle
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"))
```

In [3]:

```
def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
```

In [4]:

```
conn=sqlite3.connect('D:/Applied AI Course/final2.sqlite')
conn.cursor()
conn.commit()
conn.text_factory=str
#final_data.to_sql('Reviews',conn,schema=None,if_exists='replace')
```

In [5]:

```
fd=pd.read_sql_query("""SELECT * FROM REVIEWS""",conn)
```

In [6]:

```
conn2=sqlite3.connect('D:/Applied AI Course/final.sqlite')
```

In [7]:

```
label_df=pd.read_sql_query("""SELECT * FROM REVIEWS""",conn2)
```

In [8]:

```
label_df.head(3)
```

Out[8]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1

In [8]:

```
label_df=label_df.sort_values('Time',axis=0,inplace=False,kind='quicksort')
```

In [9]:

```
fd=fd.sort_values('Time',axis=0,inplace=False,kind='quicksort')
```

In [10]:

```
fd.head(3)
```

Out[10]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNu
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0

In [11]:

```
label_df.shape
```

Out[11]:

(364173, 12)

In [12]:

```
fd.shape
```

Out[12]:

(364173, 11)

Sampleset data

In [13]:

```
d_pos=label_df[label_df["Score"] == 'Positive'].sample(n=50000)
d_neg=label_df[label_df["Score"] == 'Negative'].sample(n=50000)
finald=pd.concat([d_pos,d_neg])
finald.shape
```

Out[13]:

```
(100000, 12)
```

In [14]:

```
finald.head(2)
final_d=finald.sort_values(by='Time')
final_d.head(3)
```

Out[14]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNu
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0

Bag of Words

In [70]:

```
X=final_d["CleanedText"]
X.shape
```

Out[70]:

```
(100000,)
```

In [71]:

```
y=final_d["Score"]
y.shape
```

Out[71]:

```
(100000,)
```

In [72]:

```
#split the data into train and test fo bag of words

X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.3,random_state=0, shuffl
e=False)
#split train into cross val train and cross val test
X_t,X_cv,Y_t,Y_cv=train_test_split(X_train,Y_train,test_size=0.3)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

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

In [73]:

```
BoW=CountVectorizer()
#final_count=count_vect.fit_transform(test_data["CleanedText"].values)
X_train = BoW.fit_transform(X_train)
X_test = BoW.transform(X_test)
#stdscaler=StandardScaler(with_mean=False)
#X_train=stdscaler.fit_transform(X_train)
#X_test=stdscaler.fit(X_train).transform(X_test)
savetofile(BoW,"D:/Applied AI Course/BoW")
print("the type of count vectorizer is:",type(X_train))
#final_count.get_shape()
print(X_train.shape,X_test.shape)
```

```
the type of count vectorizer is: <class 'scipy.sparse.csr.csr_matrix'>
(70000, 44592) (30000, 44592)
```

In [ ]:

```
openfromfile("D:/Applied AI Course/BoW")
```

In [19]:

```
print(X_train.shape,Y_train.shape,X_test.shape,Y_test.shape)
```

```
(70000, 44578) (70000,) (30000, 44578) (30000,)
```

In [74]:

```
from sklearn import preprocessing
X_train=preprocessing.normalize(X_train)
X_test=preprocessing.normalize(X_test)
```

In [75]:

```
print(X_train.shape,X_test.shape)
```

```
(70000, 44592) (30000, 44592)
```

In [76]:

```
tcv=TimeSeriesSplit(n_splits=10)
for train,cv in tcv.split(X_train):
    print(X_train[train].shape,X_train[cv].shape)
```

```
(6370, 44592) (6363, 44592)
(12733, 44592) (6363, 44592)
(19096, 44592) (6363, 44592)
(25459, 44592) (6363, 44592)
(31822, 44592) (6363, 44592)
(38185, 44592) (6363, 44592)
(44548, 44592) (6363, 44592)
(50911, 44592) (6363, 44592)
(57274, 44592) (6363, 44592)
(63637, 44592) (6363, 44592)
```

Naive Bayes

Find the best alpha using GridSearch cross validation for Laplace smoothing along with Time series splitting since the data is time-series based

In [77]:

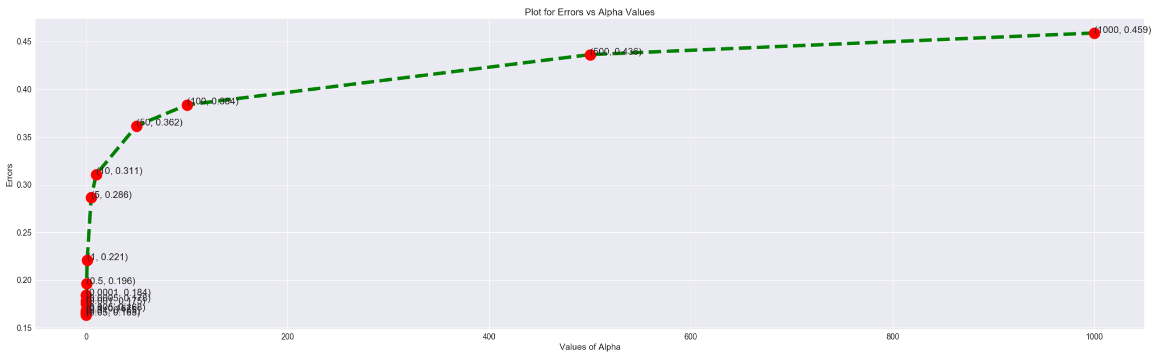
```
#find the best alpha based on cv accuracy for bow
#l_a={'alpha':[1000,500,100,50,10,5,0.1,0.05,0.001]}
#alpha_set=list(range(float(1e-0,20,1e-4)))
#for i in alpha_set:
    # instantiate learning model (alpha = 10)
nB = MultinomialNB()
#alphas = np.logspace(-5, 4, 100)
#alpha_vals = list(np.arange(10e-6 , 10e-2 , 0.005))
#dict(alpha=numpy.linspace(0,2,20)[1:])
#np.array( [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10] )
param_grid = {'alpha': [1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0
001]} #params we need to try on classifier
#myList = np.arange(0.00001, 0.001, 0.00005)
#np.linspace(0.001, 10, 10)
tcv=TimeSeriesSplit(n_splits=10)
gsv=GridSearchCV(nB,param_grid,cv=tcv,verbose=1)
gsv.fit(X_train,Y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))

#nB = BernoulliNB()
#cls=GridSearchCV(nB,l_a,cv=10)
    # fitting the model on crossvalidation train
#bNB.fit(X_train, Y_train)

    # predict the response on the crossvalidation train
#pred = bNB.predict(X_cv)

    # evaluate CV accuracy
#acc = accuracy_score(Y_cv, pred, normalize=True) * float(100)
#print("Best HyperParameter: ",cls.best_params_)
#print("Best Accuracy: %.2f%%"%(cls.best_score_*100))
#test accuracy
#nB = BernoulliNB(alpha=1000)
#nB.fit(X_train,Y_train)
#pred = nB.predict(X_test)
#acc = accuracy_score(Y_test, pred, normalize=True) * float(100)
#print('\n****Test accuracy for alpha = 1000 is %d%%' % (acc))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits

[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   59.0s finished

Best HyperParameter:  {'alpha': 0.05}
Best Accuracy: 83.67%
```

In [78]:

```
   # instantiate learning model (alpha = 10)
import matplotlib.pyplot as plt

cv_result = gsv.cv_results_
mts = cv_result["mean_test_score"]        #list that will hold the mean of cross valida
tion accuracy scores for each alpha
alphas = cv_result["params"]
alphas=np.array(alphas)
alphas
alpha_values = []                         #list that will hold all the alpha values tha
t the grid search cross validator tried.
for i in range(0,len(alphas)):
    alpha_values.append(alphas[i]["alpha"])

mse = [1 - x for x in mts]
optimal_alpha = alpha_values[mse.index(min(mse))]
print('The optimal value of alpha is : {}'.format(optimal_alpha))
plt.figure(figsize=(35,10))
plt.plot(alpha_values , mse, color='green', linestyle='dashed', linewidth=6, marker='o'
, markerfacecolor='red', markersize=20)
for xy in zip(alpha_values, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('Plot for Errors vs Alpha Values')
plt.xlabel('Values of Alpha')
plt.ylabel('Errors')
plt.show()
```

```
The optimal value of alpha is : 0.05
```



In [79]:

```
#BernoulliNB with optimal k and test accuracy for bag of words

nB_opt=MultinomialNB(alpha=optimal_alpha)
#fit the model
nB_opt.fit(X_train,Y_train)
#predict the model
prediction=nB_opt.predict(X_test)

#the accuracy score
acc_score=accuracy_score(Y_test,prediction)* 100
print('\n the accuracy score for bag of words model with optimal a=%.2f is %f%%' % (opt
imal_alpha,(acc_score)))
print('#'*100)
print("Number of mislabeled points out of a total %d points : %d" % (X_train.shape[0],(
Y_test != prediction).sum()))
print('%'*50)
training_accuracy = nB_opt.score(X_train, Y_train)
training_error = 1 - training_accuracy
test_accuracy = accuracy_score(Y_test, prediction)
test_error = 1 - test_accuracy

print("training error:%.2f%%" %training_error)
print('#'*100)
print("training accuracy:%.2f%%" %training_accuracy)
print('#'*100)
print("test error:%.2f%%" %test_error)
print('#'*100)
print("test accuracy:%.2f%%" %test_accuracy)
```

```
 the accuracy score for bag of words model with optimal a=0.05 is 85.60333
3%
##########################################################################
##########################
Number of mislabeled points out of a total 70000 points : 4319
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
training error:0.11%
##########################################################################
##########################
training accuracy:0.89%
##########################################################################
##########################
test error:0.14%
##########################################################################
##########################
test accuracy:0.86%
```

Feature importance

In [80]:

```
#pred_proba = nB_opt.predict_proba(X_test)
#words = np.take(count_vect.get_feature_names(), pred_proba.argmax(axis=1))

feat_name=np.array(BoW.get_feature_names())
f_cnt=nB_opt.feature_count_
log_prob = nB_opt.feature_log_prob_
feature_prob = pd.DataFrame(log_prob, columns=feat_name).T
#sorted_idx=nB_opt.coef_[0].argsort()
#print("smallest coeffecient is: \n {} \n".format(feat_name[sorted_idx[:10]]))
#print("largest coefficient is: \n {} \n".format(feat_name[sorted_idx[:-11:-1]]))
top_positive = feature_prob[1].sort_values(ascending=False)[:10]
top_negative = feature_prob[0].sort_values(ascending=False)[:10]

class_count = nB_opt.class_count_
pos_points_prob_sort = nB_opt.feature_log_prob_[1, :].argsort()
neg_points_prob_sort = nB_opt.feature_log_prob_[0, :].argsort()

df_res1=pd.DataFrame(top_positive)
df_res2=pd.DataFrame(top_negative)

#print(df_res1)
#print(df_res2)
#df_res1=df_res1.join(df_res2, lsuffix='_positive', rsuffix='_negative')
#print(df_res1)
#print(df_res1)
#pt=PrettyTable()
#pt.field_name=["Positive","Negative"]
#pt.add_column("Positive",top_positive[:10])
#pt.add_row([top_positive[:11]])
#pt.add_column("Negative",top_negative[:10])
#pt.add_row([top_negative[:11]])
#pt.add_row("Top positive",feature_prob[:10])
#print(pt)
print("_"*101)
print("Top 10 words with feature importance")
print("_"*101)
print("     positive")
print("_"*101)
print(df_res1)
print("_"*101)
print("     negative")
print("_"*101)
print(df_res2)
```

```
Top 10 words with feature importance
```

```
     positive
```

	1
like	-4.430782
love	-4.435004
tast	-4.436963
great	-4.450149
good	-4.512257
flavor	-4.622111
use	-4.727671
product	-4.755062
one	-4.802277
tri	-4.912813

```
     negative
```

	0
tast	-4.103294
like	-4.233568
product	-4.395340
one	-4.714888
flavor	-4.726163
would	-4.835597
tri	-4.862831
good	-4.953607
buy	-5.008081
coffe	-5.045937

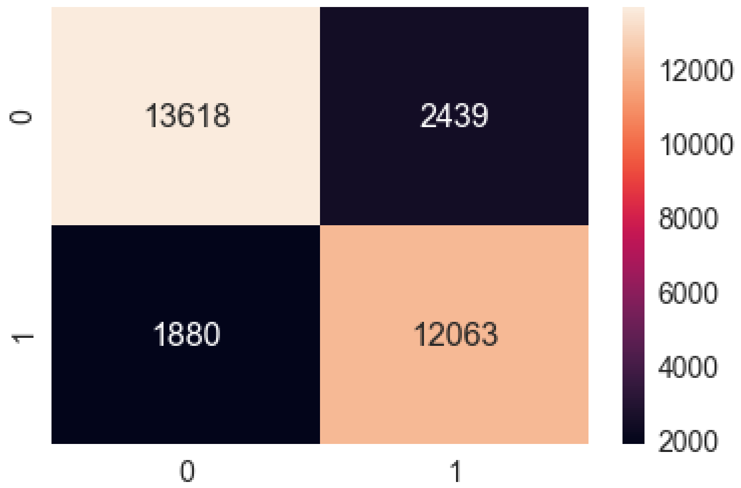
Confusion matrix

In [81]:

```
conf_matr_df = pd.DataFrame(confusion_matrix(Y_test, prediction), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[81]:

<matplotlib.axes._subplots.AxesSubplot at 0x37a61e10>



In [82]:

```
from sklearn.metrics import classification_report,precision_score,recall_score,f1_score
print(classification_report(Y_test,prediction))
```

```
             precision    recall  f1-score   support

   Negative       0.88      0.85      0.86     16057
   Positive       0.83      0.87      0.85     13943

avg / total       0.86      0.86      0.86     30000
```

In [83]:

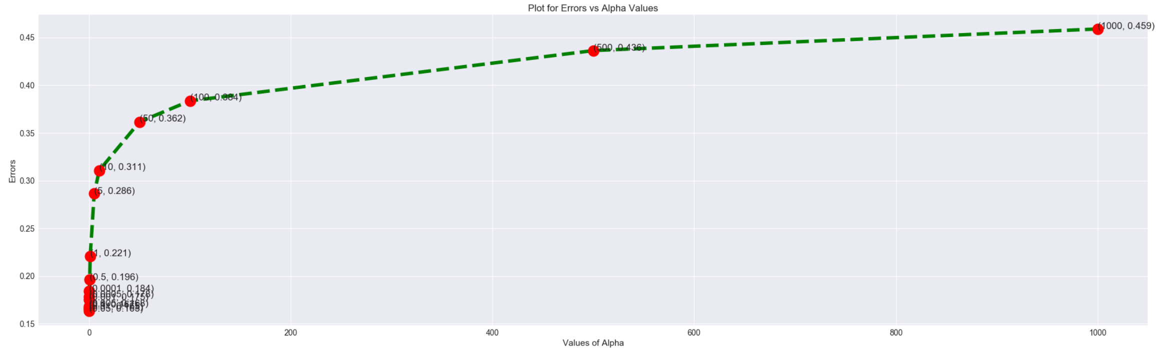
```
bnB = BernoulliNB()
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.00
01]} #params we need to try on classifier
#myList = np.arange(0.00001, 0.001, 0.00005)
tcv=TimeSeriesSplit(n_splits=10)
b_gsv=GridSearchCV(nB,param_grid,cv=tcv,verbose=1)
b_gsv.fit(X_train,Y_train)
print("Best HyperParameter: ",b_gsv.best_params_)
print("Best Accuracy: %.2f%%"%(b_gsv.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits

[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   56.0s finished

Best HyperParameter:  {'alpha': 0.05}
Best Accuracy: 83.67%
```

In [84]:

```
import matplotlib.pyplot as plt

cv_result = b_gsv.cv_results_
mts = cv_result["mean_test_score"]        #list that will hold the mean of cross valida
tion accuracy scores for each alpha
alphas = cv_result["params"]
alphas=np.array(alphas)
alphas
alpha_values = []                         #list that will hold all the alpha values tha
t the grid search cross validator tried.
for i in range(0,len(alphas)):
    alpha_values.append(alphas[i]["alpha"])

mse = [1 - x for x in mts]
m_optimal_alpha = alpha_values[mse.index(min(mse))]
print('The optimal value of alpha is : {}'.format(m_optimal_alpha))
plt.figure(figsize=(35,10))
plt.plot(alpha_values , mse, color='green', linestyle='dashed', linewidth=6, marker='o'
, markerfacecolor='red', markersize=20)
for xy in zip(alpha_values, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('Plot for Errors vs Alpha Values')
plt.xlabel('Values of Alpha')
plt.ylabel('Errors')
plt.show()
```

```
The optimal value of alpha is : 0.05
```



In [85]:

```
bnB_opt=BernoulliNB(alpha=m_optimal_alpha)
#fit the model
bnB_opt.fit(X_train,Y_train)
#predict the model
prediction=bnB_opt.predict(X_test)

#the accuracy score
b_acc_score=accuracy_score(Y_test,prediction)* 100
print('\n the accuracy score for bag of words model with optimal a=%.2f is %f%%' % (m_o
ptimal_alpha,b_acc_score))
print('#'*100)
print("Number of mislabeled points out of a total %d points : %d" % (X_train.shape[0],(
Y_test != prediction).sum()))
print('%'*50)
training_accuracy = bnB_opt.score(X_train, Y_train)
training_error = 1 - training_accuracy
test_accuracy = accuracy_score(Y_test, prediction)
test_error = 1 - test_accuracy

print("training error:%.2f%%" %training_error)
print('#'*100)
print("training accuracy:%.2f%%" %training_accuracy)
print('#'*100)
print("test error:%.2f%%" %test_error)
print('#'*100)
print("test accuracy:%.2f%%" %test_accuracy)
```

```
 the accuracy score for bag of words model with optimal a=0.05 is 83.35333
3%
##########################################################################
##########################
Number of mislabeled points out of a total 70000 points : 4994
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
training error:0.11%
##########################################################################
##########################
training accuracy:0.89%
##########################################################################
##########################
test error:0.17%
##########################################################################
##########################
test accuracy:0.83%
```

#Feature importance for BernoulliNB

In [86]:

```
feat_name=np.array(BoW.get_feature_names())
f_cnt=bnB_opt.feature_count_
log_prob = bnB_opt.feature_log_prob_
feature_prob = pd.DataFrame(log_prob, columns=feat_name).T
top_positive = feature_prob[1].sort_values(ascending=False)[:10]
top_negative = feature_prob[0].sort_values(ascending=False)[:10]

class_count = bnB_opt.class_count_
pos_points_prob_sort = bnB_opt.feature_log_prob_[1, :].argsort()
neg_points_prob_sort = bnB_opt.feature_log_prob_[0, :].argsort()

df_res1=pd.DataFrame(top_positive)
df_res2=pd.DataFrame(top_negative)


print("_"*101)
print("Top 10 words with feature importance")
print("_"*101)
print("     positive")
print("_"*101)
print(df_res1)
print("_"*101)
print("     negative")
print("_"*101)
print(df_res2)
```

```
Top 10 words with feature importance
```

```
     positive
```

```
                1
like    -1.188386
tast    -1.214855
love    -1.263692
great   -1.289337
good    -1.294080
flavor  -1.437414
one     -1.478649
use     -1.497564
tri     -1.546608
product -1.583752
```

```
     negative
```

```
                0
tast    -0.982607
like    -1.000314
product -1.210859
one     -1.347205
would   -1.417623
tri     -1.462002
flavor  -1.543126
good    -1.545748
buy     -1.619292
get     -1.668067
```

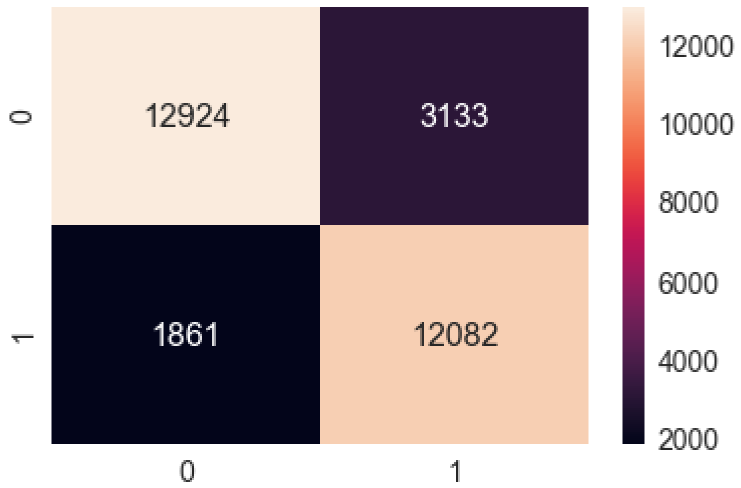
Confusion matrix

In [87]:

```
conf_matr_df = pd.DataFrame(confusion_matrix(Y_test, prediction), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[87]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x39b1c048>
```



In [88]:

```
from sklearn.metrics import classification_report,precision_score,recall_score,f1_score
print(classification_report(Y_test,prediction))
```

```
             precision    recall  f1-score   support

   Negative       0.87      0.80      0.84     16057
   Positive       0.79      0.87      0.83     13943

avg / total       0.84      0.83      0.83     30000
```

Observation: with optimal alpha being 0.05 the Bernoulli Naive Bayes for BagOfWords approach has precision 86% and recall 8% with optimal alpha being 0.05 the Multinomial Naive Bayes for BoW has precision and recall at 84% and 83% respectively

Tf_IDF

In [89]:

```
X=final_d["CleanedText"]
X.shape
```

Out[89]:

(100000,)

In [90]:

```
y=final_d["Score"]
y.shape
```

Out[90]:

(100000,)

In [91]:

```
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.30,random_state=None, sh
uffle=False)
#split train into cross val train and cross val test
X_t,X_cv,Y_t,Y_cv=train_test_split(X_train,Y_train,test_size=0.3)
```

In [92]:

```
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

(70000,) (30000,) (70000,) (30000,)

In [95]:

```
TfIdf=TfidfVectorizer()
X_train=TfIdf.fit_transform(X_train)
X_test=TfIdf.transform(X_test)
#stdscler=StandardScaler(with_mean=False)
#X_train=stdscaler.fit_transform(final_tfidf_vect)
#X_test=stdscaler.fit(final_tfidf_vect).transform(final_test_tfidf_vect)
savetofile(TfIdf,"D:/Applied AI Course/TfIdf")
print(X_train.shape,X_test.shape)
```

(70000, 44592) (30000, 44592)

In [ ]:

```
openfromfile("D:/Applied AI Course/TfIdf")
```

In [96]:

```
X_train=preprocessing.normalize(X_train)
X_test=preprocessing.normalize(X_test)
print(X_train.shape,X_test.shape)
```

(70000, 44592) (30000, 44592)

In [97]:

```
tcv=TimeSeriesSplit(n_splits=10)
for train,cv in tcv.split(X_train):
    print(X_train[train].shape,X_train[cv].shape)
```

```
(6370, 44592) (6363, 44592)
(12733, 44592) (6363, 44592)
(19096, 44592) (6363, 44592)
(25459, 44592) (6363, 44592)
(31822, 44592) (6363, 44592)
(38185, 44592) (6363, 44592)
(44548, 44592) (6363, 44592)
(50911, 44592) (6363, 44592)
(57274, 44592) (6363, 44592)
(63637, 44592) (6363, 44592)
```

To find the best alpha using grid search cross validation with time series splitting

In [98]:

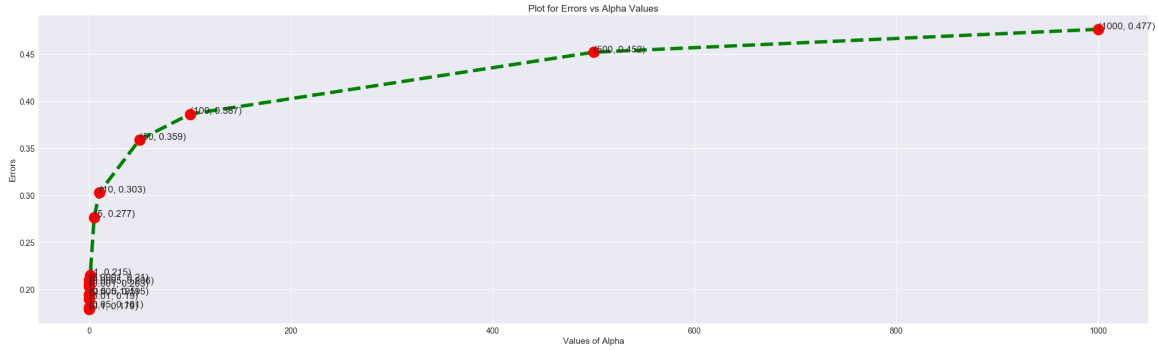
```
mnB = MultinomialNB()
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.00
01]} #params we need to try on classifier
#myList = np.arange(0.00001, 0.001, 0.00005)
tcv=TimeSeriesSplit(n_splits=10)
gsv_t=GridSearchCV(mnB,param_grid,cv=tcv,verbose=1)
gsv_t.fit(X_train,Y_train)
print("Best HyperParameter: ",gsv_t.best_params_)
print("Best Accuracy: %.2f%%"%(gsv_t.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits

[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   57.8s finished

Best HyperParameter:  {'alpha': 0.1}
Best Accuracy: 82.10%
```

In [99]:

```
import matplotlib.pyplot as plt

cv_result = gsv_t.cv_results_
mts = cv_result["mean_test_score"]        #list that will hold the mean of cross valida
tion accuracy scores for each alpha
alphas = cv_result["params"]
alphas=np.array(alphas)
alphas
alpha_values = []                         #list that will hold all the alpha values tha
t the grid search cross validator tried.
for i in range(0,len(alphas)):
    alpha_values.append(alphas[i]["alpha"])

mse = [1 - x for x in mts]
optimal_alpha_tfidf = alpha_values[mse.index(min(mse))]
print('The optimal value of alpha is : {}'.format(optimal_alpha_tfidf))
plt.figure(figsize=(35,10))
plt.plot(alpha_values , mse, color='green', linestyle='dashed', linewidth=6, marker='o'
, markerfacecolor='red', markersize=20)
for xy in zip(alpha_values, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('Plot for Errors vs Alpha Values')
plt.xlabel('Values of Alpha')
plt.ylabel('Errors')
plt.show()

#l_a={'alpha':[1000,500,100,50,10,5,0.1,0.05,0.001]}
#alpha_set=[1e-3, 1e-2,1e-1, 1e-0, 1e2, 1e3, 1e4]
#for i in alpha_set:
    # instantiate learning model (alpha = 10)
#nB = BernoulliNB()
#cls=GridSearchCV(nB,l_a,cv=10)
    # fitting the model on crossvalidation train
#cls.fit(X_train, Y_train)

    # predict the response on the crossvalidation train
#pred = cls.predict(X_cv)

    # evaluate CV accuracy
#acc = accuracy_score(Y_cv, pred, normalize=True) * float(100)
#print("Best HyperParameter: ",cls.best_params_)
#print("Best Accuracy: %.2f%%"%(cls.best_score_*100))
#test accuracy
#nB = BernoulliNB(alpha=1000)
#nB.fit(X_train,Y_train)
#pred = nB.predict(X_test)
#acc = accuracy_score(Y_test, pred, normalize=True) * float(100)
#print('\n****Test accuracy for alpha = 1000 is %d%%' % (acc))
```

```
The optimal value of alpha is : 0.1
```



In [100]:

```
mnB_opt_tf=MultinomialNB(alpha=optimal_alpha_tfidf)
#fit the model
mnB_opt_tf.fit(X_train,Y_train)
#predict the model
prediction=mnB_opt_tf.predict(X_test)

#the accuracy score
m_acc_score=accuracy_score(Y_test,prediction)* 100
print('\n the accuracy score for TfIdf model with optimal a=%.2f is %f%%' % (optimal_al
pha_tfidf,(m_acc_score)))
print('#'*100)
print("Number of mislabeled points out of a total %d points : %d" % (X_train.shape[0],(
Y_test != prediction).sum()))
print('%'*50)
training_accuracy = mnB_opt_tf.score(X_train, Y_train)
training_error_tf = 1 - training_accuracy
test_accuracy = accuracy_score(Y_test, prediction)
test_error_tf = 1 - test_accuracy

print("training error:%.2f%%" %training_error_tf)
print('#'*100)
print("training accuracy:%.2f%%" %training_accuracy)
print('#'*100)
print("test error:%.2f%%" %test_error_tf)
print('#'*100)
print("test accuracy:%.2f%%" %test_accuracy)
```

```
 the accuracy score for TfIdf model with optimal a=0.10 is 83.830000%
##########################################################################
##########################
Number of mislabeled points out of a total 70000 points : 4851
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
training error:0.10%
##########################################################################
##########################
training accuracy:0.90%
##########################################################################
##########################
test error:0.16%
##########################################################################
##########################
test accuracy:0.84%
```

Feature importance

In [101]:

```
feat_name_tf=np.array(TfIdf.get_feature_names())
f_cnt=mnB_opt_tf.feature_count_
log_prob = mnB_opt_tf.feature_log_prob_
feature_prob = pd.DataFrame(log_prob, columns=feat_name_tf).T
#sorted_idx=nB_opt.coef_[0].argsort()
#print("smallest coeffecient is: \n {} \n".format(feat_name[sorted_idx[:10]]))
#print("largest coefficient is: \n {} \n".format(feat_name[sorted_idx[:-11:-1]]))
top_positive = feature_prob[1].sort_values(ascending=False)[:10]
top_negative = feature_prob[0].sort_values(ascending=False)[:10]
df_res1=pd.DataFrame(top_positive)
df_res2=pd.DataFrame(top_negative)
print("_"*101)
print("Top 10 words with feature importance")
print("_"*101)
print("     positive  ")
print("_"*101)
print(df_res1)
print("_"*101)
print("     negative  ")
print("_"*101)
print(df_res2)
```

```
_____________________________________________________________________________________________________
Top 10 words with feature importance
_____________________________________________________________________________________________________
     positive  
_____________________________________________________________________________________________________
                1
great   -4.986396
love    -5.022042
good    -5.157499
tea     -5.191400
tast    -5.220942
like    -5.224462
flavor  -5.236926
coffe   -5.284627
use     -5.333968
product -5.418744
_____________________________________________________________________________________________________
     negative  
_____________________________________________________________________________________________________
                0
tast    -4.886878
like    -5.029320
product -5.086899
flavor  -5.349372
coffe   -5.371320
one     -5.390101
would   -5.405713
tri     -5.489964
order   -5.535339
buy     -5.542741
```

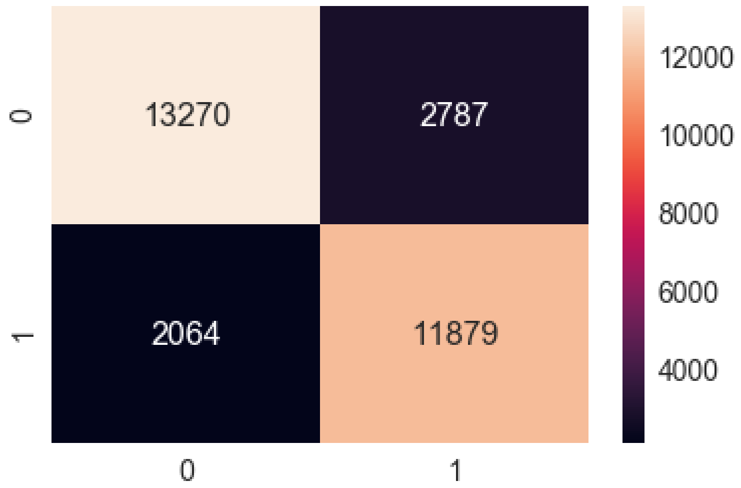
Confusion matrix

In [102]:

```
conf_matr_df = pd.DataFrame(confusion_matrix(Y_test, prediction), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[102]:

<matplotlib.axes._subplots.AxesSubplot at 0x347b99e8>



In [103]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction))
```

```
             precision    recall  f1-score   support

   Negative       0.87      0.83      0.85     16057
   Positive       0.81      0.85      0.83     13943

avg / total       0.84      0.84      0.84     30000
```

BernoulliNB

In [104]:

```
tcv=TimeSeriesSplit(n_splits=10)
for train,cv in tcv.split(X_train):
    print(X_train[train].shape,X_train[cv].shape)
```

```
(6370, 44592) (6363, 44592)
(12733, 44592) (6363, 44592)
(19096, 44592) (6363, 44592)
(25459, 44592) (6363, 44592)
(31822, 44592) (6363, 44592)
(38185, 44592) (6363, 44592)
(44548, 44592) (6363, 44592)
(50911, 44592) (6363, 44592)
(57274, 44592) (6363, 44592)
(63637, 44592) (6363, 44592)
```

In [105]:

```
nB = BernoulliNB()
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.00
01]} #params we need to try on classifier
tcv=TimeSeriesSplit(n_splits=10)
gsv_t_b=GridSearchCV(nB,param_grid,cv=tcv,verbose=1)
gsv_t_b.fit(X_train,Y_train)
print("Best HyperParameter: ",gsv_t_b.best_params_)
print("Best Accuracy: %.2f%%"%(gsv_t_b.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

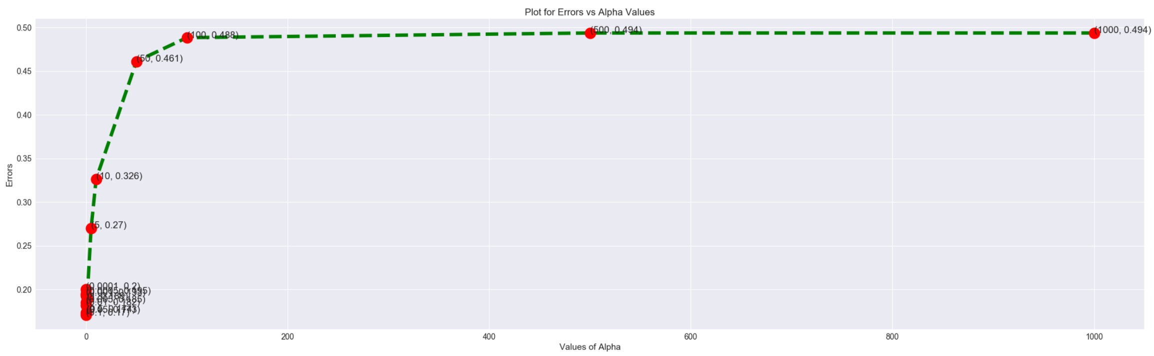
```
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:  1.1min finished
```

```
Best HyperParameter:  {'alpha': 0.1}
Best Accuracy: 82.95%
```

In [106]:

```
import matplotlib.pyplot as plt

cv_result = gsv_t_b.cv_results_
mts = cv_result["mean_test_score"]        #list that will hold the mean of cross valida
tion accuracy scores for each alpha
alphas = cv_result["params"]
alphas=np.array(alphas)
alphas
alpha_values = []                         #list that will hold all the alpha values tha
t the grid search cross validator tried.
for i in range(0,len(alphas)):
    alpha_values.append(alphas[i]["alpha"])

mse = [1 - x for x in mts]
btf_optimal_alpha = alpha_values[mse.index(min(mse))]
print('The optimal value of alpha is : {}'.format(btf_optimal_alpha))
plt.figure(figsize=(35,10))
plt.plot(alpha_values , mse, color='green', linestyle='dashed', linewidth=6, marker='o'
, markerfacecolor='red', markersize=20)
for xy in zip(alpha_values, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('Plot for Errors vs Alpha Values')
plt.xlabel('Values of Alpha')
plt.ylabel('Errors')
plt.show()
```

```
The optimal value of alpha is : 0.1
```



In [107]:

```
nB_opt_tf=BernoulliNB(alpha=btf_optimal_alpha)
#fit the model
nB_opt_tf.fit(X_train,Y_train)
#predict the model
prediction=nB_opt_tf.predict(X_test)

#the accuracy score
acc_score=accuracy_score(Y_test,prediction)* 100
print('\n the accuracy score for TfIDf model with optimal a=%.2f is %f%%' %(btf_optimal
_alpha,(acc_score)))

print("Number of mislabeled points out of a total %d points : %d" % (X_train.shape[0],(
Y_test != prediction).sum()))

training_accuracy = nB_opt_tf.score(X_train, Y_train)
training_error_tfidf = 1 - training_accuracy
test_accuracy = accuracy_score(Y_test, prediction)
test_error_tfidf = 1 - test_accuracy

print("training error:%.2f%%" %training_error_tfidf)
print('#'*100)
print("training accuracy:%.2f%%" %training_accuracy)
print('#'*100)
print("test error:%.2f%%" %test_error_tfidf)
print('#'*100)
print("test accuracy:%.2f%%" %test_accuracy)
```

```
 the accuracy score for TfIDf model with optimal a=0.10 is 83.603333%
Number of mislabeled points out of a total 70000 points : 4919
training error:0.11%
###########################################################################
##########################
training accuracy:0.89%
###########################################################################
##########################
test error:0.16%
###########################################################################
##########################
test accuracy:0.84%
```

In [108]:

```
feat_name_tf=np.array(TfIdf.get_feature_names())
f_cnt=nB_opt_tf.feature_count_
log_prob = nB_opt_tf.feature_log_prob_
feature_prob = pd.DataFrame(log_prob, columns=feat_name_tf).T
#sorted_idx=nB_opt.coef_[0].argsort()
#print("smallest coeffecient is: \n {} \n".format(feat_name[sorted_idx[:10]]))
#print("largest coefficient is: \n {} \n".format(feat_name[sorted_idx[:-11:-1]]))
top_positive = feature_prob[1].sort_values(ascending=False)[:10]
top_negative = feature_prob[0].sort_values(ascending=False)[:10]

df_res1=pd.DataFrame(top_positive)
df_res2=pd.DataFrame(top_negative)


print("_"*101)
print("Top 10 words with feature importance")
print("_"*101)
print("     positive    ")
print(df_res1)
print("_"*101)
print("     negative    ")
print(df_res2)
print("_"*101)
```

```
_____________________________________________________________________________________________________
Top 10 words with feature importance
_____________________________________________________________________________________________________
     positive    
                1
like    -1.188385
tast    -1.214853
love    -1.263690
great   -1.289335
good    -1.294078
flavor  -1.437411
one     -1.478646
use     -1.497560
tri     -1.546604
product -1.583748
_____________________________________________________________________________________________________
     negative    
                0
tast    -0.982606
like    -1.000313
product -1.210857
one     -1.347202
would   -1.417620
tri     -1.461999
flavor  -1.543122
good    -1.545744
buy     -1.619288
get     -1.668062
_____________________________________________________________________________________________________
```

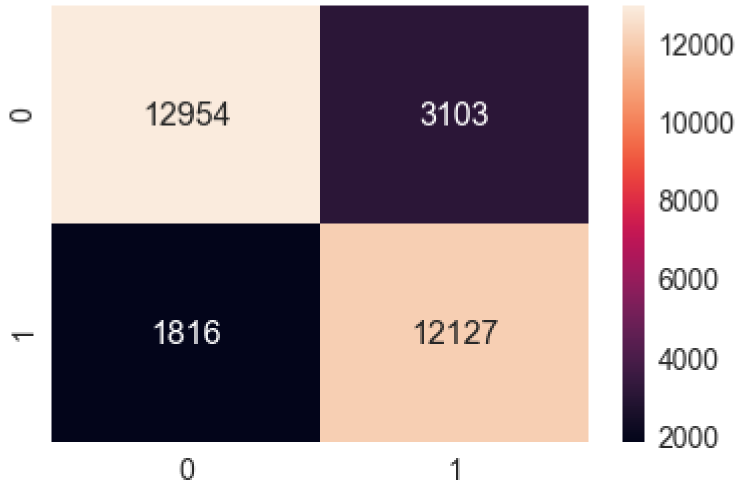
Confusion matrix

In [109]:

```
conf_matr_df = pd.DataFrame(confusion_matrix(Y_test, prediction), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[109]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x42ab64e0>
```



In [110]:

```
from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction))
```

```
             precision    recall  f1-score   support

   Negative       0.88      0.81      0.84     16057
   Positive       0.80      0.87      0.83     13943

avg / total       0.84      0.84      0.84     30000
```

Observation: with alpha being 0.1 for BernoulliNB we get precision 84% and recall at 84% while for MultinomialNB optimal alpha being 0.1 we get precision and recall at 84%

Word2Vec

In [112]:

```
import gensim
from gensim.models import word2vec,KeyedVectors
```

In [113]:

```
X=final_d["Text"]
X.shape
```

Out[113]:

```
(100000,)
```

In [114]:

```
y=final_d["Score"]
y.shape
```

Out[114]:

```
(100000,)
```

In [115]:

```
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.30,random_state=None, sh
uffle=False)
#split train into cross val train and cross val test
X_t,X_cv,Y_t,Y_cv=train_test_split(X_train,Y_train,test_size=0.3, shuffle=False)
```

In [116]:

```
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

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

In [117]:

```
i=0
list_of_sentence=[]
for sent in X_train.values:
    filtered_sentence=[]
    list_of_sentence.append(sent.split())
    #sent=cleanhtml(sent)
    #for w in sent.split():
      #   for cleaned in cleanpunc(w).split():
      #      if(cleaned.isalpha()):
       #         filtered_sentence.append(cleaned.lower())
        #    else:
         #       continue
#list_of_sentence.append(filtered_sentence)
#print(X["CleanedText"].values[0])
#print('###########')
#print(list_of_sentence[0])
W2V_Tr=gensim.models.Word2Vec(list_of_sentence,min_count=5,size=50,workers=4)
savetofile(W2V_Tr,"D:/Applied AI Course/W2V_Tr")
words=list(W2V_Tr.wv.vocab)
print(len(words))
```

```
35134
```

In [118]:

```
W2V_Tr.wv.most_similar('like')
```

Out[118]:

```
[('like,', 0.7047081589698792),
 ('like.', 0.6462217569351196),
 ('miss', 0.6058076620101929),
 ('resemble', 0.5741134285926819),
 ('enjoy', 0.5489007234573364),
 ('like...', 0.5378314256668091),
 ('crave', 0.5366167426109314),
 ('know,', 0.5362352728843689),
 ('mean', 0.5354884266853333),
 ('prefer', 0.5214476585388184)]
```

In [119]:

```
#word2vec for test
i=0
list_of_sentences=[]
for sent in X_test.values:
    filtered_sentences=[]
    list_of_sentences.append(sent.split())
    #sent=cleanhtml(sent)
    #for w in sent.split():
     #   for cleaned in cleanpunc(w).split():
      #      if(cleaned.isalpha()):
       #         filtered_sentence.append(cleaned.lower())
        #    else:
         #       continue
#list_of_sentences.append(filtered_sentence)
#print(X_train.values[0])
#print('##########')
print(list_of_sentences[0])
W2V_test=gensim.models.Word2Vec(list_of_sentences,min_count=5,size=50,workers=4)
savetofile(W2V_test,"D:/Applied AI Course/W2V_test")
words_test=list(W2V_test.wv.vocab)
print(len(words_test))
```

```
['I', 'was', 'surprised', 'that', 'these', 'Habichuelas', 'con', 'Dulce',
'do', 'not', 'taste', 'subpar', 'at', 'all!', 'I', 'actually', 'think', 't
hey', 'taste', 'about', 'average,', 'but', "I'm", 'disappointed', 'by', 't
he', 'lack', 'of', 'yams,', 'raisins,', 'and', 'other', 'tasty', 'compleme
nts', 'that', 'are', 'usually', 'found', 'in', 'this', 'dish!']
20614
```

In [120]:

```
from tqdm import tqdm
```

In [ ]:

```
import re


def cleanhtml(sentence):
    cleantext = re.sub('<.*>', '', sentence)
    return cleantext


def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|#|@|.|,|)|(|\|/]', r'', sentence)
    return cleaned
```

In [ ]:

```
print(train_w2v_words.shape, test_w2v_words.shape)
```

Avg-Word2Vec

In [121]:

```
sent_vectors = []
for sent in tqdm(list_of_sentence): # 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
        try:
            vec = W2V_Tr.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass

    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))


train_vectors=np.nan_to_num(sent_vectors)
```

```
100%|██████████████████████████████████████████| 70000/70000 [00:21<00:00, 3320.15
it/s]

70000
50
```

In [122]:

```
test_vectors = []
for sent in tqdm(list_of_sentences):
    sent_vec = np.zeros(50)
    cnt_words = 0
    for word in sent:
        try:
            vec = W2V_Tr.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    test_vectors.append(sent_vec)
test_vectors = np.nan_to_num(test_vectors)
```

```
100%|██████████████████████████████████| 30000/30000 [00:09<00:00, 3248.11
it/s]
```

In [123]:

```
X_train=np.array(train_vectors)
X_test=np.array(test_vectors)
```

In [124]:

```
X_train.shape
```

Out[124]:

```
(70000, 50)
```

In [125]:

```
Y_train.shape
```

Out[125]:

```
(70000,)
```

In [ ]:

```
#model=word2vec.Word2Vec.load('w2vmodel')
```

In [126]:

```
tcv=TimeSeriesSplit(n_splits=10)
for train,cv in tcv.split(X_train):
    print(X_train[train].shape,X_train[cv].shape)
```

```
(6370, 50) (6363, 50)
(12733, 50) (6363, 50)
(19096, 50) (6363, 50)
(25459, 50) (6363, 50)
(31822, 50) (6363, 50)
(38185, 50) (6363, 50)
(44548, 50) (6363, 50)
(50911, 50) (6363, 50)
(57274, 50) (6363, 50)
(63637, 50) (6363, 50)
```

In [127]:

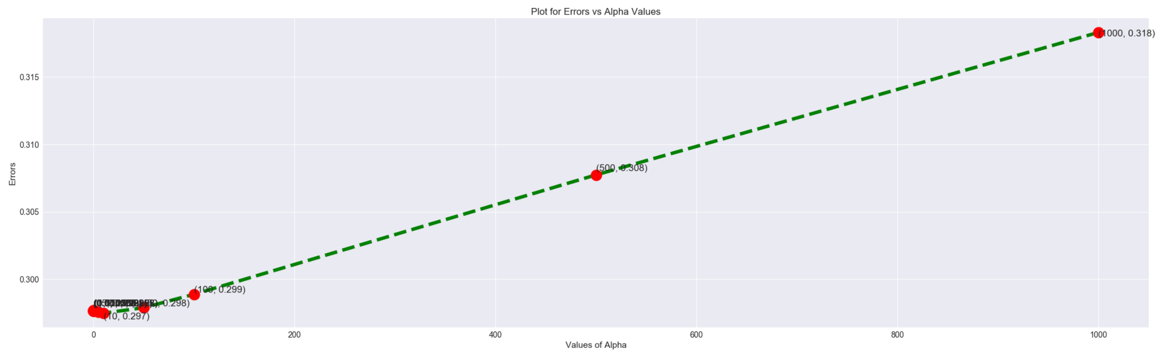
```
mnB = BernoulliNB()
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.00
01]} #params we need to try on classifier
#myList = np.arange(0.00001, 0.001, 0.00005)
tcv=TimeSeriesSplit(n_splits=10)
gsv_awt=GridSearchCV(mnB,param_grid,cv=tcv,verbose=1,n_jobs=3,scoring='accuracy')
gsv_awt.fit(X_train,Y_train)
print("Best HyperParameter: ",gsv_awt.best_params_)
print("Best Accuracy: %.2f%%"%(gsv_awt.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits

[Parallel(n_jobs=3)]: Done  44 tasks      | elapsed:    8.6s
[Parallel(n_jobs=3)]: Done 150 out of 150 | elapsed:   27.3s finished

Best HyperParameter:  {'alpha': 10}
Best Accuracy: 70.25%
```

In [128]:

```
import matplotlib.pyplot as plt

cv_result = gsv_awt.cv_results_
mts = cv_result["mean_test_score"]        #list that will hold the mean of cross valida
tion accuracy scores for each alpha
alphas = cv_result["params"]
alphas=np.array(alphas)
alphas
alpha_values = []                         #list that will hold all the alpha values tha
t the grid search cross validator tried.
for i in range(0,len(alphas)):
    alpha_values.append(alphas[i]["alpha"])

mse = [1 - x for x in mts]
aw2v_optimal_alpha = alpha_values[mse.index(min(mse))]
print('The optimal value of alpha is : {}'.format(aw2v_optimal_alpha))
plt.figure(figsize=(35,10))
plt.plot(alpha_values , mse, color='green', linestyle='dashed', linewidth=6, marker='o'
, markerfacecolor='red', markersize=20)
for xy in zip(alpha_values, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('Plot for Errors vs Alpha Values')
plt.xlabel('Values of Alpha')
plt.ylabel('Errors')
plt.show()
```

```
The optimal value of alpha is : 10
```



In [129]:

```
nB_opt_aw2v=BernoulliNB(alpha=aw2v_optimal_alpha)
#fit the model
nB_opt_aw2v.fit(X_train,Y_train)
#predict the model
prediction=nB_opt_aw2v.predict(X_test)

#the accuracy score
acc_score=accuracy_score(Y_test,prediction)* 100
print('\n the accuracy score for Weighted TfIdf model with optimal a=%.2f is %f%%' %(aw
2v_optimal_alpha,(acc_score)))

print("Number of mislabeled points out of a total %d points : %d" % (X_train.shape[0],(
Y_test != prediction).sum()))


training_accuracy = nB_opt_aw2v.score(X_train, Y_train)
training_error_aw2v = 1 - training_accuracy
test_accuracy = accuracy_score(Y_test, prediction)
test_error_aw2v = 1 - test_accuracy

print("training error:%.2f%%" %training_error)
print('#'*100)
print("training accuracy:%.2f%%" %training_accuracy)
print('#'*100)
print("test error:%.2f%%" %test_error)
print('#'*100)
print("test accuracy:%.2f%%" %test_accuracy)
```

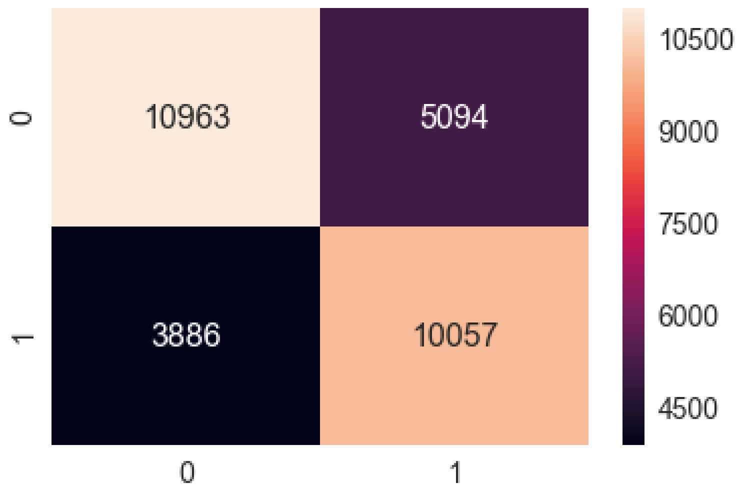
```
 the accuracy score for Weighted TfIdf model with optimal a=10.00 is 70.06
6667%
Number of mislabeled points out of a total 70000 points : 8980
training error:0.11%
##########################################################################
##########################
training accuracy:0.71%
##########################################################################
##########################
test error:0.17%
##########################################################################
##########################
test accuracy:0.70%
```

confusion matrix

In [130]:

```
conf_matr_df = pd.DataFrame(confusion_matrix(Y_test, prediction), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df, annot=True,annot_kws={"size": 16}, fmt='g')

print("-----------------------------------------------------------------------------------
-------------------------------------")

from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction))
```

```
-----------------------------------------------------------------------------
-------------------------------------------
             precision    recall  f1-score   support

   Negative       0.74      0.68      0.71     16057
   Positive       0.66      0.72      0.69     13943

avg / total       0.70      0.70      0.70     30000
```



Weighted Tf-IDf Word2Vec

In [131]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final_d['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 [132]:

```
X=final_d["CleanedText"]
X.shape
```

Out[132]:

```
(100000,)
```

In [133]:

```
y=final_d["Score"]
y.shape
```

Out[133]:

(100000,)

In [134]:

```
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.30,random_state=None, sh
uffle=False)
#split train into cross val train and cross val test
X_t,X_cv,Y_t,Y_cv=train_test_split(X_train,Y_train,test_size=0.3,random_state=None, shu
ffle=False)
```

In [135]:

```
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

(70000,) (30000,) (70000,) (30000,)

In [136]:

```
X_train = model.fit_transform(X_train)
X_test = model.transform(X_test)
#stdscaler=StandardScaler(with_mean=False)
#X_train=stdscaler.fit_transform(X_train)
#X_test=stdscaler.fit(X_train).transform(X_test)
#savetofile(tf_idf_vect,"D:/Applied AI Course/Tfidfweightedwrd2vec")
```

In [137]:

```
X_train=preprocessing.normalize(X_train)
X_test=preprocessing.normalize(X_test)
```

In [138]:

```
savetofile(model,"D:/Applied AI Course/model")
```

In [139]:

```
print(X_train.shape,X_test.shape)
```

(70000, 44592) (30000, 44592)

In [140]:

```
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 = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this lis
t
row=0;
for sent in tqdm(list_of_sentence): # 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
        try:
            vec = W2V_Tr.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)
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass

    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████| 70000/70000 [00:27<00:00, 2536.23
it/s]
```

In [141]:

```
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 = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in thi
s list
row=0;
for sent in tqdm(list_of_sentences): # 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
        try:
            vec = W2V_Tr.wv[word]
            tf_idf = dictionary[word]*sent.count(word)
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass

#             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

    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████| 30000/30000 [00:12<00:00, 2463.08
it/s]
```

In [145]:

```
X_train=np.array(tfidf_sent_vectors)
X_test=np.array(tfidf_sent_vectors_test)
```

In [ ]:

```
X_train.shape
```

alpha value using GridSearch Cross-validation

In [146]:

```
tcv=TimeSeriesSplit(n_splits=10)
for train,cv in tcv.split(X_train):
    print(X_train[train].shape,X_train[cv].shape)
```

```
(6370, 50) (6363, 50)
(12733, 50) (6363, 50)
(19096, 50) (6363, 50)
(25459, 50) (6363, 50)
(31822, 50) (6363, 50)
(38185, 50) (6363, 50)
(44548, 50) (6363, 50)
(50911, 50) (6363, 50)
(57274, 50) (6363, 50)
(63637, 50) (6363, 50)
```

BernoulliNB

In [147]:

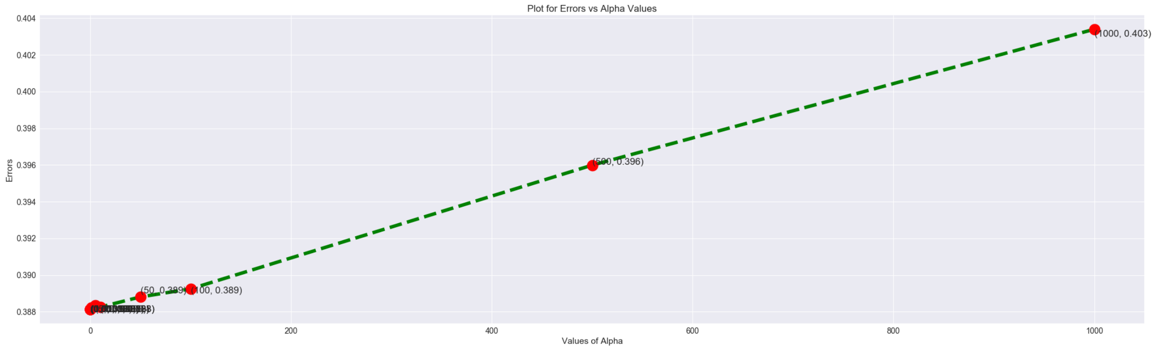
```
mnB = BernoulliNB()
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.00
01]} #params we need to try on classifier
#myList = np.arange(0.00001, 0.001, 0.00005)
tcv=TimeSeriesSplit(n_splits=10)
gsv_wt=GridSearchCV(mnB,param_grid,cv=tcv,verbose=1,n_jobs=3,scoring='accuracy')
gsv_wt.fit(X_train,Y_train)
print("Best HyperParameter: ",gsv_wt.best_params_)
print("Best Accuracy: %.2f%%"%(gsv_wt.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits

[Parallel(n_jobs=3)]: Done  44 tasks      | elapsed:    9.3s
[Parallel(n_jobs=3)]: Done 150 out of 150 | elapsed:   29.6s finished

Best HyperParameter:  {'alpha': 0.005}
Best Accuracy: 61.19%
```

In [148]:

```
import matplotlib.pyplot as plt

cv_result = gsv_wt.cv_results_
mts = cv_result["mean_test_score"]        #List that will hold the mean of cross valida
tion accuracy scores for each alpha
alphas = cv_result["params"]
alphas=np.array(alphas)
alphas
alpha_values = []                         #List that will hold all the alpha values tha
t the grid search cross validator tried.
for i in range(0,len(alphas)):
    alpha_values.append(alphas[i]["alpha"])

mse = [1 - x for x in mts]
tfw2v_optimal_alpha = alpha_values[mse.index(min(mse))]
print('The optimal value of alpha is : {}'.format(tfw2v_optimal_alpha))
plt.figure(figsize=(35,10))
plt.plot(alpha_values , mse, color='green', linestyle='dashed', linewidth=6, marker='o'
, markerfacecolor='red', markersize=20)
for xy in zip(alpha_values, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.title('Plot for Errors vs Alpha Values')
plt.xlabel('Values of Alpha')
plt.ylabel('Errors')
plt.show()
#l_a={'alpha':[1000,500,100,50,10,5,0.1,0.05,0.001]}
#alpha_set=[1e-3, 1e-2,1e-1, 1e-0, 1e2, 1e3, 1e4]
#for i in alpha_set:
    # instantiate learning model (alpha = 10)
#nB = BernoulliNB()
#cls=GridSearchCV(nB,l_a,cv=10)
    # fitting the model on crossvalidation train
#cls.fit(X_train, Y_train)

    # predict the response on the crossvalidation train
#pred = cls.predict(X_cv)

    # evaluate CV accuracy
#acc = accuracy_score(Y_cv, pred, normalize=True) * float(100)
#print("Best HyperParameter: ",cls.best_params_)
#print("Best Accuracy: %.2f%%"%(cls.best_score_*100))
#test accuracy
#nB = BernoulliNB(alpha=1)
#nB.fit(X_train,Y_train)
#pred = nB.predict(X_test)
#acc = accuracy_score(Y_test, pred, normalize=True) * float(100)
#print('\n****Test accuracy for alpha = 1000 is %d%%' % (acc))
```

```
The optimal value of alpha is : 0.005
```



In [150]:

```
nB_opt_tfw2v=BernoulliNB(alpha=tfw2v_optimal_alpha)
#fit the model
nB_opt_tfw2v.fit(X_train,Y_train)
#predict the model
prediction=nB_opt_tfw2v.predict(X_test)

#the accuracy score
acc_score=accuracy_score(Y_test,prediction)* 100
print('\n the accuracy score for Weighted TfIdf model with optimal a=%.4f is %f%%' %(tf
w2v_optimal_alpha,(acc_score)))

print("Number of mislabeled points out of a total %d points : %d" % (X_train.shape[0],(
Y_test != prediction).sum()))


training_accuracy = nB_opt_tfw2v.score(X_train, Y_train)
training_error_tfw2v = 1 - training_accuracy
test_accuracy = accuracy_score(Y_test, prediction)
test_error_tfw2v = 1 - test_accuracy

print("training error:%.2f%%" %training_error)
print('#'*100)
print("training accuracy:%.2f%%" %training_accuracy)
print('#'*100)
print("test error:%.2f%%" %test_error)
print('#'*100)
print("test accuracy:%.2f%%" %test_accuracy)
```

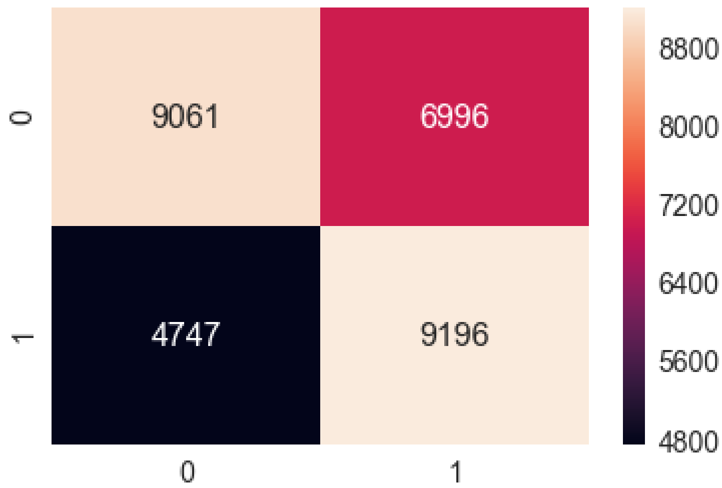
```
 the accuracy score for Weighted TfIdf model with optimal a=0.0050 is 60.8
56667%
Number of mislabeled points out of a total 70000 points : 11743
training error:0.11%
###########################################################################
##########################
training accuracy:0.62%
###########################################################################
##########################
test error:0.17%
###########################################################################
##########################
test accuracy:0.61%
```

Confusion matrix

In [151]:

```
conf_matr_df = pd.DataFrame(confusion_matrix(Y_test, prediction), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df, annot=True,annot_kws={"size": 16}, fmt='g')

print("-----------------------------------------------------------------------------------------------------------------------")

from sklearn.metrics import classification_report
print(classification_report(Y_test,prediction))
```

```
--------------------------------------------------------------------------
--------------------------------------------
             precision    recall  f1-score   support

   Negative       0.66      0.56      0.61     16057
   Positive       0.57      0.66      0.61     13943

avg / total       0.62      0.61      0.61     30000
```



Observations & Conclusions

In [152]:

```
from prettytable import PrettyTable

pr=PrettyTable()
hyp1=m_optimal_alpha
tr=np.round(training_error,3)
te=np.round(test_error,3)
hyp2=optimal_alpha_tfidf
tr1=np.round(training_error_tf,2)
te1=np.round(test_error_tf,2)
pr.field_names=["Model", "Hyperparameter(Alpha)", "Train error"," Test Error"]
pr.add_row(["BoW",hyp1,tr,te])
pr.add_row(["TFIDF",hyp2,tr1,te1])
pr.add_row(["(Bernoulli Tfidf)",btf_optimal_alpha,np.round(training_error_tfidf,3),np.r
ound(test_error_tfidf,3)])
pr.add_row(["AvgWord2Vec",aw2v_optimal_alpha,np.round(training_error_aw2v,3),np.round(t
est_error_aw2v,3)])
pr.add_row(["Weighted TFIDF Word2Vec",tfw2v_optimal_alpha,np.round(training_error_tfw2v
,3),np.round(test_error_tfw2v,3)])
print(pr)
```

```
+-------------------------+-----------------------+-------------+---------
----+
|          Model          | Hyperparameter(Alpha) | Train error |  Test Er
ror |
+-------------------------+-----------------------+-------------+---------
----+
|           BoW           |          0.05         |     0.11    |    0.166
    |
|          TFIDF          |          0.1          |     0.1     |     0.16
    |
|    (Bernoulli Tfidf)    |          0.1          |    0.113    |    0.164
    |
|       AvgWord2Vec       |           10          |    0.292    |    0.299
    |
| Weighted TFIDF Word2Vec |         0.005         |    0.383    |    0.391
    |
+-------------------------+-----------------------+-------------+---------
----+
```