Support Vector Machine on Amazon fine food dataset

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

To perform Support Vector Machine on different vectors like BOW, Tf-idf, Avg-W2vec & Tf-idf_W2vec.

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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 re

import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
```

```
#Importing Train and test dataset
train_data=pd.read_csv("E:/Applied AI assignments/Amazon_fine_train_data.csv")
test_data=pd.read_csv("E:/Applied AI assignments/Amazon_fine_test_data.csv")
```

```
train_data=train_data.astype(str)
test_data=test_data.astype(str)
```

```
train_data.shape
```

```
(80000, 13)
```

```
train_data['Score'].value_counts()
```

```
positive    70407
negative     9593
Name: Score, dtype: int64
```

```
test_data.shape
```

```
(20000, 13)
```

```
test_data['Score'].value_counts()
```

```
positive    17322
negative     2678
Name: Score, dtype: int64
```

```
#Train data
y_train = train_data['Score']
x_train = train_data['CleanedText']

#Test data
y_test = test_data['Score']
x_test = test_data['CleanedText']
```

```
#Replacing Positive score with 0 and negative score with 1
y_train.replace('negative',1,inplace=True)
y_train.replace('positive',0,inplace=True)

y_test.replace('negative',1,inplace=True)
y_test.replace('positive',0,inplace=True)
```

```
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn import cross_validation
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

Since SVM is computationally expensive therfore running SGD classifier on different featurization like BOW, Tfidf(bigram), Avg-W2V & Tfidf-W2V. For the featurization where we will get best performance metric(f1_score), we will run SVC classifier

RandomisedSearch CV

```
alpha_range = [0.000001,0.00001,0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1,2,3,4,5,6,7
epsilon_range=[0.0000001,0.000001,0.00001,0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
T= TimeSeriesSplit(n_splits=5)
weight=[None,'balanced']

param_distributions = dict(alpha=alpha_range,epsilon=epsilon_range,class_weight=weight)
print(param_distributions)

# instantiate and fit the grid
grid = RandomizedSearchCV(SGDClassifier(), param_distributions, cv=T, scoring='f1',n_iter=30, return_tra
```

```
{'alpha': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 15, 20, 30, 40, 50], 'epsilon': [1e-07, 1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1.0], 'class_weight': [None, 'balanced']}
```

Binary Bow

```
count_vect = CountVectorizer(binary=True)

#Train data
vocabulary = count_vect.fit(x_train) #in scikit-learn
Bow_x_train= count_vect.transform(x_train)
print("the type of count vectorizer ",type(Bow_x_train))
print("the shape of out text BOW vectorizer ",Bow_x_train.get_shape())
print("the number of unique words ", Bow_x_train.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (80000, 33433)
the number of unique words  33433
```

```
#Test data
Bow_x_test = count_vect.transform(x_test)
print("the type of count vectorizer ",type(Bow_x_test))
print("the shape of out text BOW vectorizer ",Bow_x_test.get_shape())
print("the number of unique words ", Bow_x_test.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (20000, 33433)
the number of unique words  33433
```

```
#Standardizing Bow_x_train and Bow_x_test
from sklearn.preprocessing import StandardScaler
Standard=StandardScaler(with_mean=False)
Bow_x_train = Standard.fit_transform(Bow_x_train)
Bow_x_test = Standard.transform(Bow_x_test)

print(Bow_x_train.shape)
print(Bow_x_test.shape)
```

```
(80000, 33433)
(20000, 33433)
```

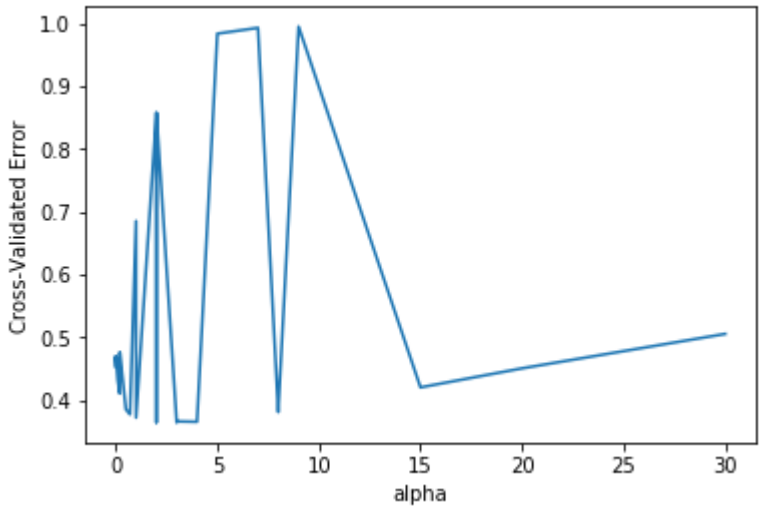
Fitting Randomised search CV on BOW

```
grid.fit(Bow_x_train, y_train)

# examine the best model
print(grid.best_score_)
print(grid.best_params_)
```

```
0.6363102574032368
{'epsilon': 0.4, 'class_weight': 'balanced', 'alpha': 3}
```

```
#Plotting alpha v/s CV_error
a=pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
a['alpha'] = [d.get('alpha') for d in a['params']]
b=a.sort_values(['alpha'])
CV_Error=1-b['mean_test_score']
alpha =b['alpha']


plt.plot(alpha,CV_Error)
plt.xlabel('alpha')
plt.ylabel('Cross-Validated Error')
```

```
Text(0,0.5,'Cross-Validated Error')
```



```
#{'epsilon': 0.4, 'class_weight': 'balanced', 'alpha': 3}
svm_optimal=SGDClassifier(alpha=3,epsilon=0.4,class_weight='balanced')

# fitting the model
svm_optimal.fit(Bow_x_train, y_train)

# predict the response
pred_bow = svm_optimal.predict(Bow_x_test)

# evaluate f1_score
f1_score = f1_score(y_test, pred_bow)

# Train & Test Error
print("The overall f1_score for the Train Data is : ", metrics.f1_score(y_train,svm_optimal.predict(Bow_
print("The overall f1_score for the Test Data is : ", metrics.f1_score(y_test,pred_bow))
```

```
The overall f1_score for the Train Data is :  0.7488488356795158
The overall f1_score for the Test Data is :  0.6735686274509805
```

Pertubation test

```
# Re-training the model after adding noise
Epsilon = np.random.normal(loc=0,scale =0.01)
Noise_Bow_x_train=Bow_x_train
Noise_Bow_x_train.data+=Epsilon
```

```
Noise_Bow_x_train.shape
```

```
(80000, 33433)
```

```
#{'epsilon': 0.4, 'class_weight': 'balanced', 'alpha': 3}
svm_optimal_noise=SGDClassifier(alpha=3,epsilon=0.4,class_weight='balanced')

# fitting the model
svm_optimal_noise.fit(Noise_Bow_x_train, y_train)

# predict the response
pred_bow = svm_optimal_noise.predict(Bow_x_test)

# evaluate f1_score
f1_score = f1_score(y_test, pred_bow)

# Train & Test Error
print("The overall f1_score for the Train Data is : ", metrics.f1_score(y_train,svm_optimal_noise.predi
print("The overall f1_score for the Test Data is : ", metrics.f1_score(y_test,pred_bow))
```

```
The overall f1_score for the Train Data is :  0.7488608482299334
The overall f1_score for the Test Data is :  0.6734597899357265
```

```
#Features
feature_names = np.array(vocabulary.get_feature_names())
feature_names.shape
```

```
(33433,)
```

```
#Weights before adding noise
svm_optimal.coef_.shape
```

```
(1, 33433)
```

```
#Weights after adding noise
svm_optimal_noise.coef_.shape
```

```
(1, 33433)
```

```
merge_arr = np.concatenate([svm_optimal.coef_, svm_optimal_noise.coef_], axis=0)
merge=pd.DataFrame(data=merge_arr,columns=feature_names).transpose()
merge[2]=((merge[1]-merge[0])/merge[0])*100
merge
```

	0	1	2
aaa	-0.000568	-0.000568	-0.006081
aaaaaaaaagghh	-0.000670	-0.000670	-0.004300
aaaaah	-0.000758	-0.000757	-0.006081
aaaaahhhhhhhhhhhhhhh	-0.000268	-0.000268	-0.004300
aaaah	-0.000134	-0.000268	99.991400
aaah	-0.000387	-0.000464	19.991062
aachen	0.001966	0.001966	-0.004300
aad	0.000000	0.000000	NaN
aadp	-0.000536	-0.000536	-0.004300
aafco	0.000438	0.000438	-0.010533
aagh	-0.000670	-0.000670	-0.004300
aah	-0.000852	-0.000852	-0.006081
aahh	-0.000670	-0.000670	-0.004300
aand	-0.000670	-0.000670	-0.004300
aardvark	-0.000670	-0.000670	-0.004300
aarrgh	0.003931	0.003931	-0.004300
ab	-0.000506	-0.000506	-0.011377
aback	-0.000867	-0.000915	5.445340
abandon	-0.000282	-0.000119	-57.928600
abaolut	-0.000670	-0.000670	-0.004300
abattoir	-0.000134	-0.000134	-0.004300
abba	-0.000773	-0.000773	-0.007448
abbey	-0.000568	-0.000663	16.659572
abbi	-0.001377	-0.001317	-4.357023

	0	1	2
abbott	-0.000095	-0.000095	-0.006081
abbrevi	-0.000077	-0.000155	99.985104
abc	-0.000460	-0.000391	-15.048432
abcstor	-0.000670	-0.000670	-0.004300
abd	-0.000473	-0.000473	-0.006081
abdomen	-0.000387	-0.000387	-0.007448
...	...	...	...
zot	-0.001004	-0.001004	-0.008600
zotz	-0.000599	-0.000599	-0.012162
zour	0.001788	0.001787	-0.008600
zout	-0.000947	-0.000947	-0.006081
zowi	-0.000402	-0.000402	-0.004300
zreport	-0.000670	-0.000670	-0.004300
zsweet	-0.000670	-0.000670	-0.004300
zuc	-0.000670	-0.000670	-0.004300
zucchini	-0.000297	-0.000363	22.550771
zuccini	-0.001620	-0.001569	-3.136021
zuccnini	0.000000	0.000000	NaN
zuchinni	-0.000670	-0.000670	-0.004300
zuke	0.000402	0.000512	27.418087
zulu	-0.000670	-0.000670	-0.004300
zum	0.000000	-0.000134	-inf
zummi	0.000000	-0.000134	-inf
zune	-0.000670	-0.000670	-0.004300
zupreem	-0.000268	-0.000268	-0.004300
zurich	-0.000670	-0.000670	-0.004300
zwar	-0.000134	-0.000134	-0.004300
zwieback	0.002064	0.002219	7.483452
zwiebeck	-0.000670	-0.000670	-0.004300

	0	1	2
zydeco	-0.000670	-0.000670	-0.004300
zzzzzs	-0.000663	-0.000568	-14.290927
zzzzzz	0.000000	0.000000	NaN
zzzzzzz	0.002948	0.002948	-0.004300
zzzzzzzz	-0.000268	-0.000268	-0.004300
zzzzzzzzzz	0.000000	0.000000	NaN
zzzzzzzzzzz	-0.000670	-0.000670	-0.004300
çay	-0.000268	-0.000402	49.993550

33433 rows × 3 columns

```
merge[merge[2]>30].shape
```

```
(1319, 3)
```

1319 features out of 33433 shows percentage change > 30 post pertubation test i.e 3.94%

We can say that our data isn't much affected by multicollinearity

```
#Features
feature_names = np.array(vocabulary.get_feature_names())
sorted_coef_index = svm_optimal.coef_[0].argsort()
```

```
#Top 20 positive features
p=feature_names[sorted_coef_index[:20]]

sp = ""
for i in p:
    sp += str(i)+","
print(sp)
```

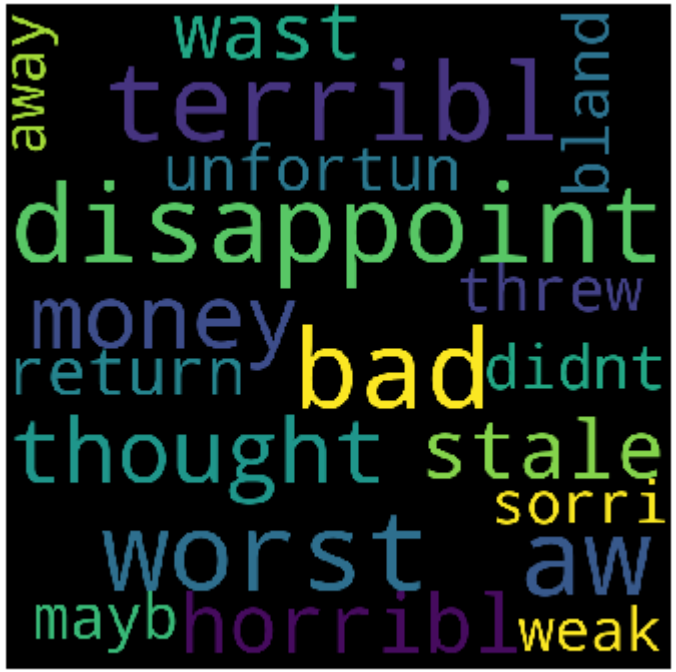
```
great,love,best,delici,perfect,favorit,excel,good,find,easi,nice,wonder,snack,make,thank,alway,tasti,keep,year,high,
```

```
#Top 20 negative features
n=feature_names[sorted_coef_index[:-21:-1]]

sn = ""
for i in n:
    sn += str(i)+","
print(sn)
```

```
disappoint,worst,terribl,bad,aw,thought,money,horribl,stale,would,wast,return,unfortun,threw,bland,didnt,mayb,sorri,weak,aw
ay,
```

```
print("************ Top 20 Negative words ******************")
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                min_font_size = 10).generate(sn)

# plot the WordCloud image
plt.figure(figsize = (5,5), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()


print("************ Top 20 Positive words ******************")
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                min_font_size = 10).generate(sp)

# plot the WordCloud image
plt.figure(figsize = (5,5), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

```
************ Top 20 Negative words ******************
```

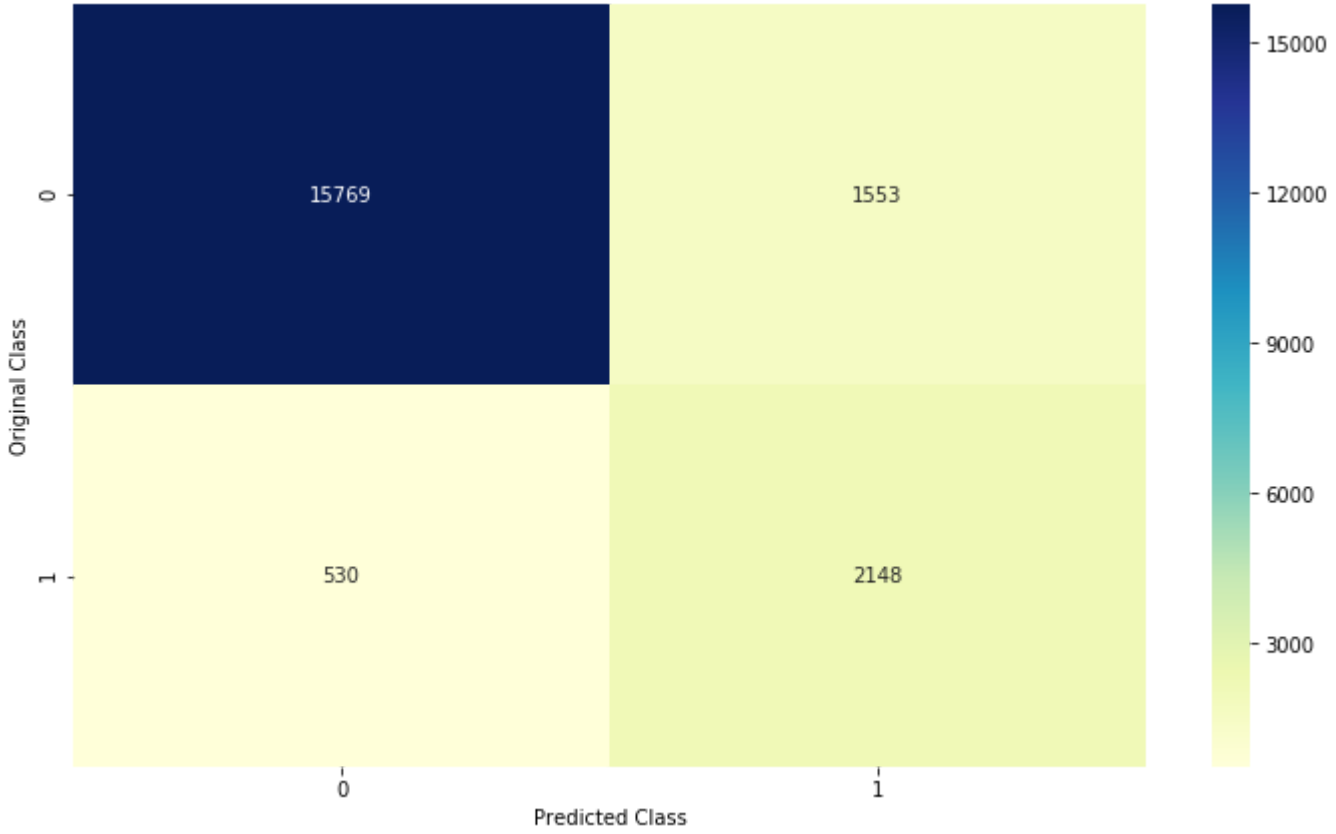


********** Top 20 Positive words **********

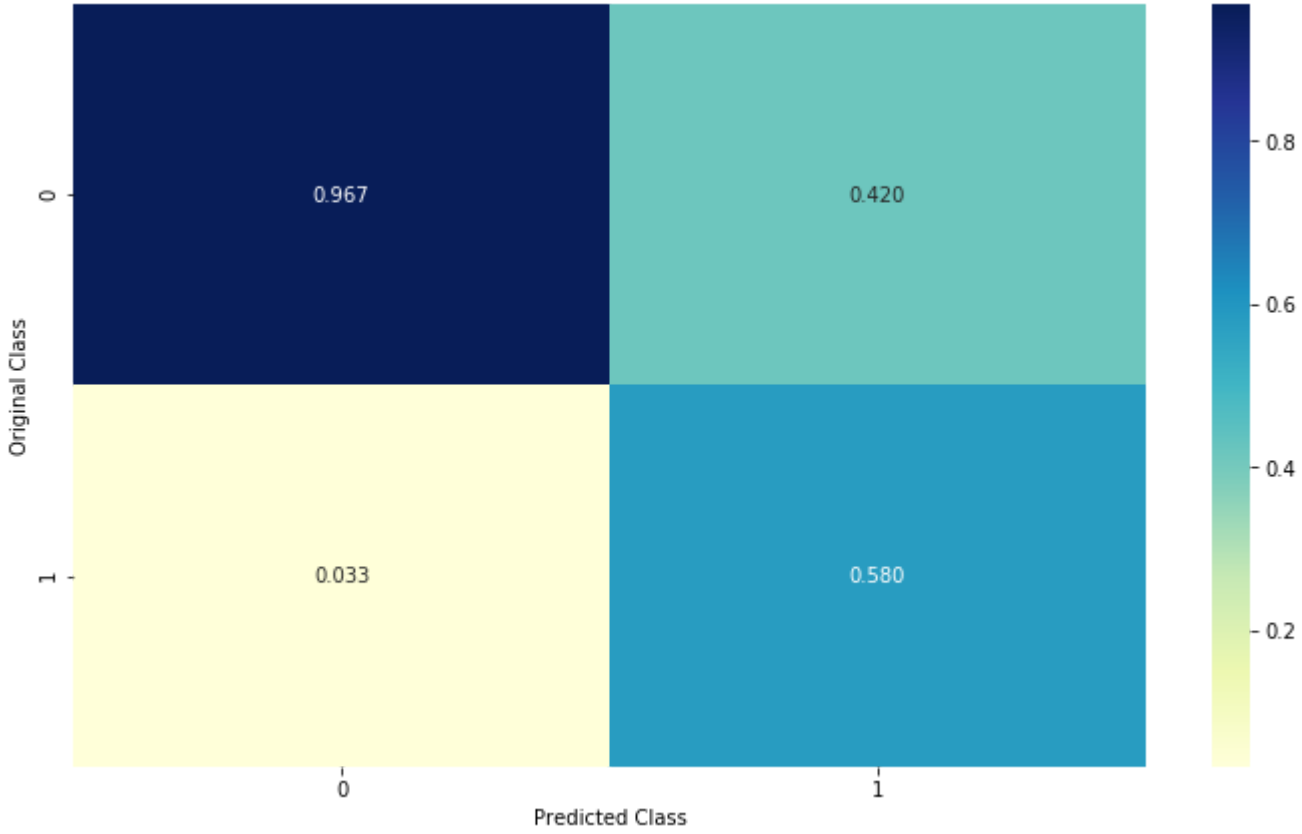

```
#Confusion matrix
C = confusion_matrix(y_test, pred_bow)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 print("-"*20, "Confusion matrix", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt="g", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
 plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

    # representing B in heatmap format
print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```

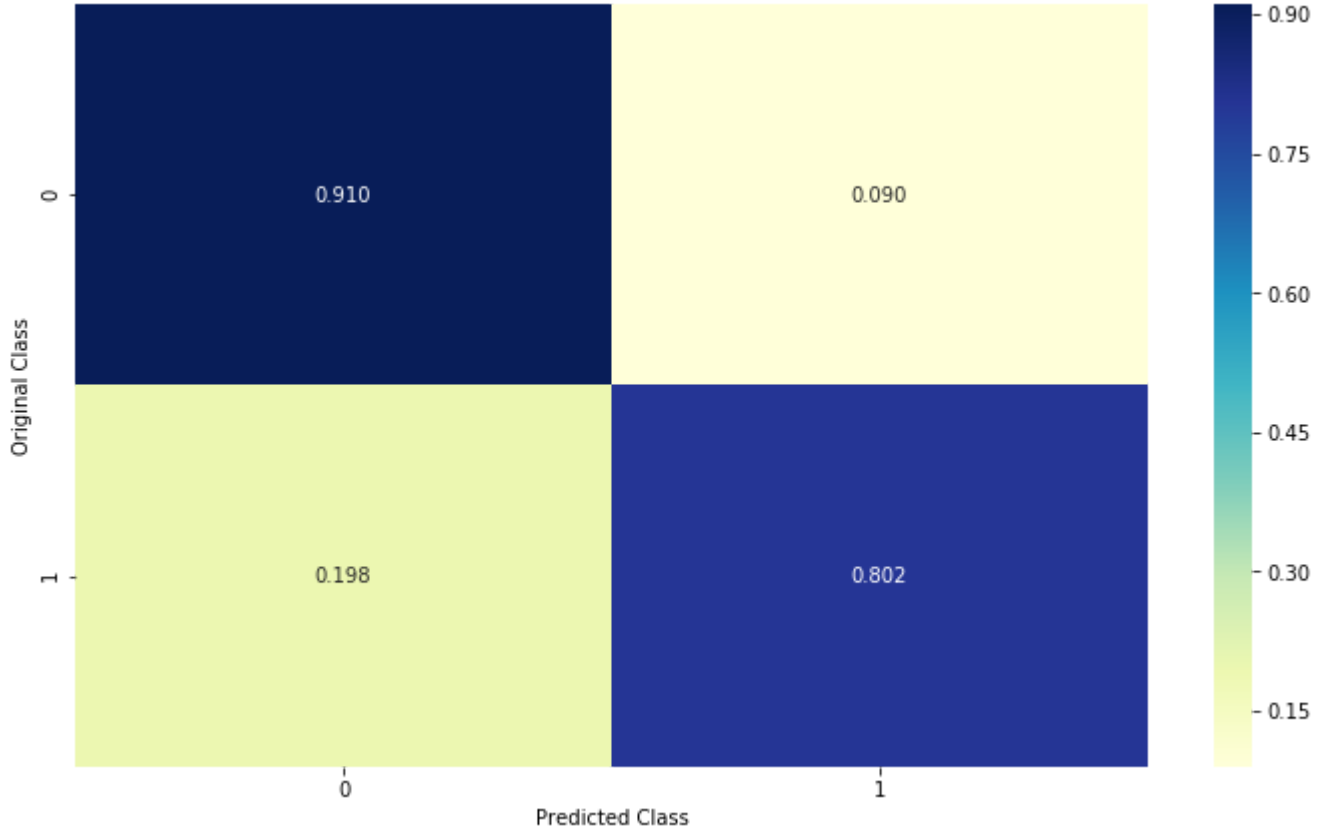
```
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



---------- Recall matrix (Row sum=1) ----------


0.910
0.090
0.198
0.802
Original Class
Predicted Class
0
1
0.90
0.75
0.60
0.45
0.30
0.15

Tf-idf

```
#Initiating Vectorizer
count_vect = TfidfVectorizer(ngram_range=(1,2))

#Train data
vocabulary = count_vect.fit(x_train)
Tfidf_x_train= count_vect.transform(x_train)
print("the type of count vectorizer ",type(Tfidf_x_train))
print("the shape of out text BOW vectorizer ",Tfidf_x_train.get_shape())
print("the number of unique words ", Tfidf_x_train.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (80000, 1013943)
the number of unique words  1013943
```

```
#Test data
Tfidf_x_test= count_vect.transform(x_test)
print("the type of count vectorizer ",type(Tfidf_x_test))
print("the shape of out text BOW vectorizer ",Tfidf_x_test.get_shape())
print("the number of unique words ", Tfidf_x_test.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (20000, 1013943)
the number of unique words  1013943
```

```
#Standardizing
from sklearn.preprocessing import StandardScaler
Standard=StandardScaler(with_mean=False)
Tfidf_x_train = Standard.fit_transform(Tfidf_x_train)
Tfidf_x_test = Standard.transform(Tfidf_x_test)

print(Tfidf_x_train.shape)
print(Tfidf_x_test.shape)
```

```
(80000, 1013943)
(20000, 1013943)
```

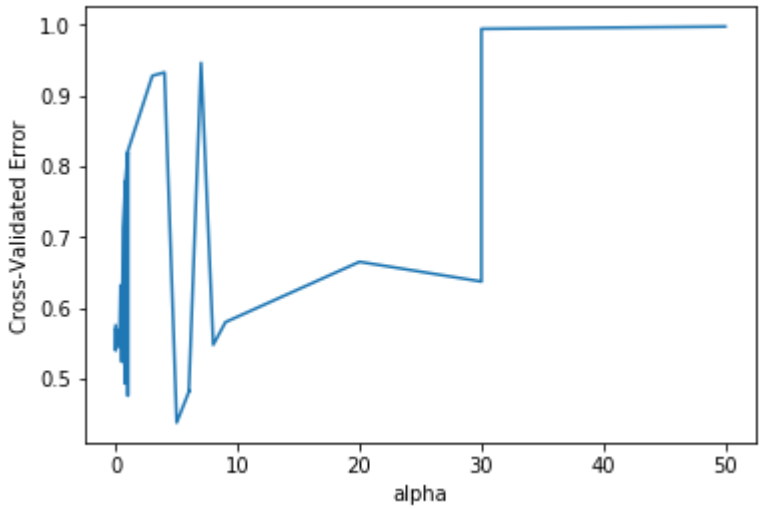
Fitting Grid Search on Tf-Idf

```
grid.fit(Tfidf_x_train, y_train)

# examine the best model
print(grid.best_score_)
print(grid.best_params_)
```

```
0.5625001303946727
{'epsilon': 0.2, 'class_weight': 'balanced', 'alpha': 5}
```

```
#Plotting alpha v/s CV_error
a=pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
a['alpha'] = [d.get('alpha') for d in a['params']]
b=a.sort_values(['alpha'])
CV_Error=1-b['mean_test_score']
alpha =b['alpha']


plt.plot(alpha,CV_Error)
plt.xlabel('alpha')
plt.ylabel('Cross-Validated Error')
```

```
Text(0,0.5,'Cross-Validated Error')
```



```

#{'epsilon': 0.2, 'class_weight': 'balanced', 'alpha': 5}
svm_optimal=SGDClassifier(alpha=5,epsilon=0.2,class_weight='balanced')

# fitting the model
svm_optimal.fit(Tfidf_x_train, y_train)

# predict the response
pred_tfidf = svm_optimal.predict(Tfidf_x_test)

# evaluate accuracy
f1_score = f1_score(y_test, pred_tfidf)

# Train & Test Error
print("The overall f1_score for the Train Data is : ", metrics.f1_score(y_train,svm_optimal.predict(Tfi
print("The overall f1_score for the Test Data is : ", metrics.f1_score(y_test,pred_tfidf))
```

```
The overall f1_score for the Train Data is :  0.9989060790748555
The overall f1_score for the Test Data is :  0.5788245279015488
```

Pertubation test

```
# Re-training the model after adding noise
Epsilon = np.random.normal(loc=0,scale =0.01)
Noise_Tfidf_x_train=Tfidf_x_train
Noise_Tfidf_x_train.data+=Epsilon
```

```
#{'epsilon': 0.2, 'class_weight': 'balanced', 'alpha': 5}
svm_optimal_noise=SGDClassifier(alpha=5,epsilon=0.2,class_weight='balanced')

# fitting the model
svm_optimal_noise.fit(Noise_Tfidf_x_train, y_train)

# predict the response
pred_tfidf = svm_optimal_noise.predict(Tfidf_x_test)

# evaluate accuracy
f1_score = f1_score(y_test, pred_tfidf)

# Train & Test Error
print("The overall f1_score for the Train Data is : ", metrics.f1_score(y_train,svm_optimal.predict(Noi
print("The overall f1_score for the Test Data is : ", metrics.f1_score(y_test,pred_tfidf))
```

```
The overall f1_score for the Train Data is :  0.9989060790748555
The overall f1_score for the Test Data is :  0.5773239736226335
```

```
#Features
feature_names = np.array(vocabulary.get_feature_names())
feature_names.shape
```

```
(1013943,)
```

```
svm_optimal.coef_.shape
```

```
(1, 1013943)
```

```
svm_optimal_noise.coef_.shape
```

```
(1, 1013943)
```

```
merge_arr = np.concatenate([svm_optimal.coef_, svm_optimal_noise.coef_], axis=0)
merge=pd.DataFrame(data=merge_arr,columns=feature_names).transpose()
merge[2]=((merge[1]-merge[0])/merge[0])*100
merge
```

	0	1	2
aaa	-0.000105	-0.000179	71.285216
aaa condit	-0.000080	-0.000161	99.996881
aaa perfect	-0.000080	-0.000080	-0.001559
aaaaaaaaagghh	-0.000161	-0.000161	-0.001559
aaaaah	-0.000113	-0.000113	-0.002219
aaaaah awak	-0.000080	-0.000080	-0.001559
aaaaah satisfi	-0.000080	-0.000080	-0.001559
aaaaahhhhhhhhhhhhhhh	-0.000080	-0.000080	-0.001559
aaaaahhhhhhhhhhhhhhhh angel	-0.000080	-0.000080	-0.001559
aaaah	-0.000080	-0.000080	-0.001559
aaaah snob	-0.000080	-0.000080	-0.001559
aaah	-0.000134	-0.000134	-0.002801
aaah inhal	-0.000080	-0.000080	-0.001559
aaah miss	-0.000080	-0.000080	-0.001559
aaah sip	-0.000080	-0.000080	-0.001559
aachen	0.000590	0.000590	-0.001559
aachen munich	0.000590	0.000590	-0.001559
aad	0.000000	0.000000	NaN
aad sausag	0.000000	0.000000	NaN
aadp	-0.000080	-0.000080	-0.001559
aafco	0.000142	0.000184	29.714666
aafco also	-0.000080	-0.000080	-0.001559
aafco certifi	0.000590	0.000590	-0.001559
aafco countri	-0.000080	-0.000080	-0.001559

	0	1	2
aafco definit	0.000590	0.000590	-0.001559
aafco dog	-0.000080	0.000000	-100.000000
aafco guidelin	0.000000	0.000000	NaN
aafco requir	-0.000080	-0.000080	-0.001559
aagh	-0.000080	-0.000080	-0.001559
aagh yelp	-0.000080	-0.000080	-0.001559
...	...	...	...
zum heal	-0.000080	-0.000080	-0.001559
zummi	-0.000080	-0.000080	-0.001559
zummi love	-0.000080	-0.000080	-0.001559
zummi tast	-0.000080	-0.000080	-0.001559
zummi tri	-0.000080	-0.000080	-0.001559
zune	-0.000241	-0.000241	-0.001559
zune video	-0.000241	-0.000241	-0.001559
zupreem	-0.000080	-0.000080	-0.001559
zupreem ferret	-0.000080	-0.000080	-0.001559
zurich	-0.000080	-0.000080	-0.001559
zurich schnatzlet	-0.000080	-0.000080	-0.001559
zwar	-0.000080	-0.000080	-0.001559
zwar billig	-0.000080	-0.000080	-0.001559
zwieback	0.000288	0.000288	-0.002321
zwieback toast	0.000288	0.000288	-0.002321
zwiebeck	-0.000161	-0.000161	-0.001559
zwiebeck toast	-0.000161	-0.000161	-0.001559
zydeco	-0.000161	-0.000161	-0.001559
zydeco saturday	-0.000161	-0.000161	-0.001559
zzzzzs	-0.000170	-0.000170	-0.002208
zzzzzs larg	-0.000161	-0.000161	-0.001559
zzzzzz	0.000000	0.000000	NaN

	0	1	2
zzzzzz say	0.000000	0.000000	NaN
zzzzzzz	0.000590	0.000590	-0.001559
zzzzzzz high	0.000590	0.000590	-0.001559
zzzzzzzz	-0.000080	-0.000080	-0.001559
zzzzzzzzzz	-0.000080	-0.000080	-0.001559
zzzzzzzzzz final	-0.000080	-0.000080	-0.001559
zzzzzzzzzzz	-0.000080	-0.000080	-0.001559
çay	-0.000080	-0.000080	-0.001559

1013943 rows × 3 columns

```
merge[merge[2]>30].shape
```

```
(30883, 3)
```

30883 features out of 1013943 shows percentage change > 30 post pertubation test i.e 3.04%

We can say that our data isn't much affected by multicollinearity

```
sorted_coef_index = svm_optimal.coef_[0].argsort()
```

```
#Top 20 positive features
p=feature_names[sorted_coef_index[:20]]

sp = ""
for i in p:
    sp += str(i)+","
print(sp)
```

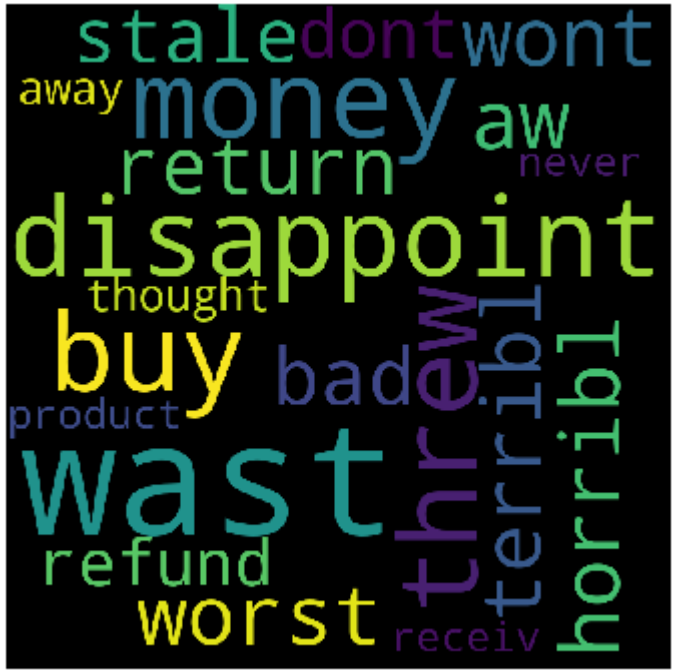
```
great,love,good,best,delici,excel,favorit,find,use,make,perfect,wonder,tasti,nice,flavor,price,easi,enjoy,high recommend,th
ank,
```

```
#Top 20 negative features
n=feature_names[sorted_coef_index[:-21:-1]]

sn = ""
for i in n:
    sn += str(i)+","
print(sn)
```

```
disappoint,worst,wast money,horribl,terribl,aw,wont buy,wast,threw,return,stale,refund,bland,money,wors,two star,disappoint
product,bad,never buy,disgust,
```

```
print("************ Top 20 Negative words ******************")
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                min_font_size = 10).generate(sn)

# plot the WordCloud image
plt.figure(figsize = (5,5), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()


print("************ Top 20 Positive words ******************")
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='black',
                min_font_size = 10).generate(sp)

# plot the WordCloud image
plt.figure(figsize = (5,5), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

```
************ Top 20 Negative words ******************
```

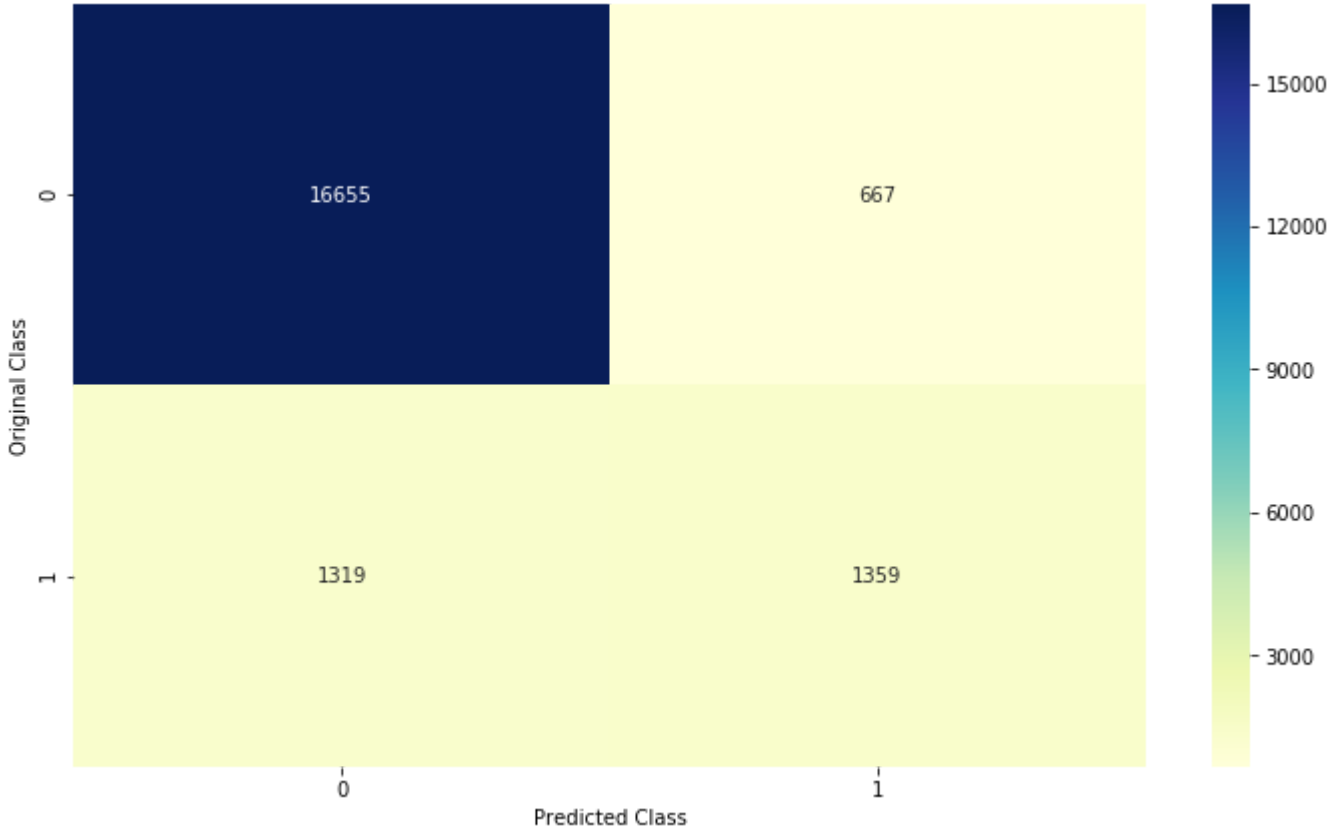


********** Top 20 Positive words **********

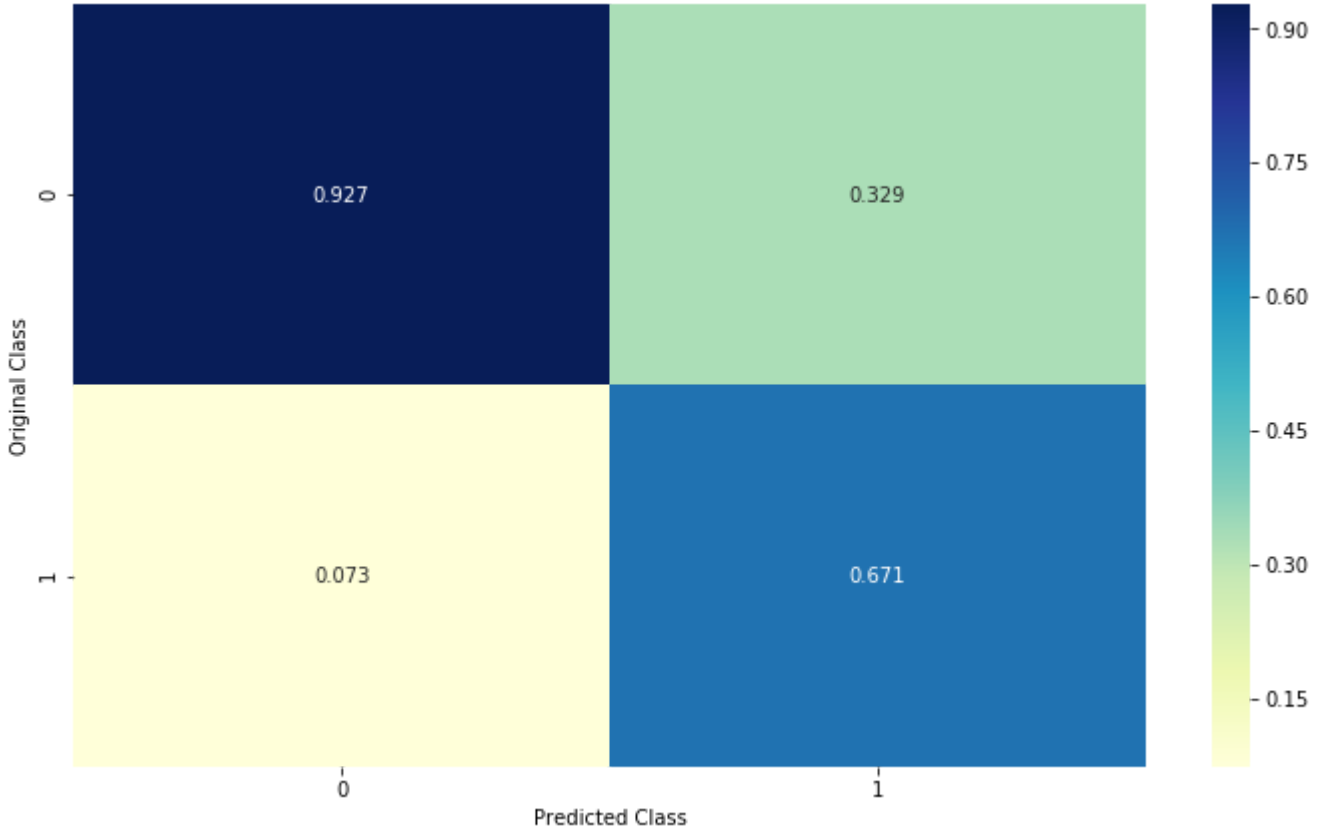

```
#Confusion matrix
C = confusion_matrix(y_test, pred_tfidf)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 print("-"*20, "Confusion matrix", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt="g", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
 plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

    # representing B in heatmap format
print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```

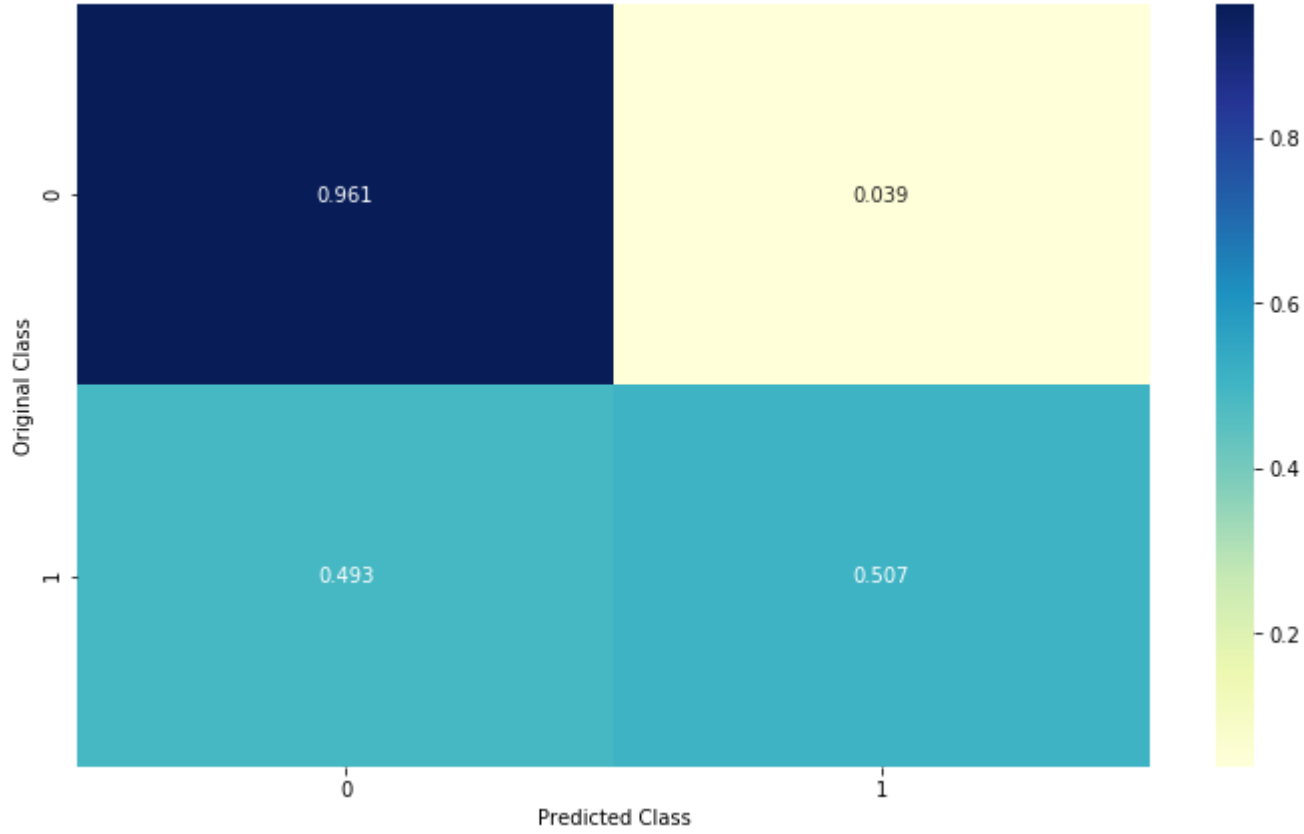
```
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



---------- Recall matrix (Row sum=1) ----------



Avg-W2Vec

```
#W2V list of Training data
i=0
list_of_sent_train=[]
for sent in train_data['CleanedText'].values:
    list_of_sent_train.append(sent.split())
```

```
#W2V List of Test data
i=0
list_of_sent_test=[]
for sent in test_data['CleanedText'].values:
    list_of_sent_test.append(sent.split())
```

```
#Training W2V train model
# min_count = 5 considers only words that occured atleast 5 times
w2v_model_train=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=6)
```

```
w2v_words_train = list(w2v_model_train.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_train))
print("sample words ", w2v_words_train[0:50])
```

```
number of words that occured minimum 5 times  11361
sample words  ['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'car', 'drive', 'along', 'alway', 'sing', 'refrai
n', 'hes', 'learn', 'whale', 'india', 'droop', 'love', 'new', 'word', 'introduc', 'silli', 'classic', 'will', 'bet', 'stil
l', 'abl', 'memori', 'colleg', 'rememb', 'see', 'show', 'air', 'televis', 'year', 'ago', 'child', 'sister', 'later', 'bough
t', 'day', 'thirti', 'someth', 'use', 'seri', 'song', 'student', 'teach', 'preschool']
```

```
#Train data
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train_avgw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent_train: # 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_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train_avgw2v.append(sent_vec)
print(len(sent_vectors_train_avgw2v))
print(len(sent_vectors_train_avgw2v[0]))
```

```
80000
50
```

```
#Test data
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test_avgw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent_test: # 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_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test_avgw2v.append(sent_vec)
print(len(sent_vectors_test_avgw2v))
print(len(sent_vectors_test_avgw2v[0]))
```

```
20000
50
```

```
#Standardizing Avg-W2v
from sklearn.preprocessing import StandardScaler
Standard=StandardScaler()
sent_vectors_train_avgw2v = Standard.fit_transform(sent_vectors_train_avgw2v)
sent_vectors_test_avgw2v = Standard.transform(sent_vectors_test_avgw2v)

print(sent_vectors_train_avgw2v.shape)
print(sent_vectors_test_avgw2v.shape)
```

```
(80000, 50)
(20000, 50)
```

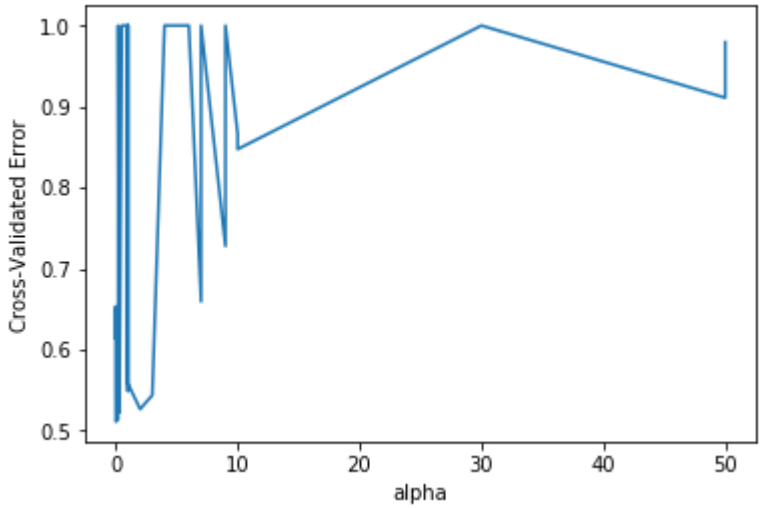
Fitting grid search on Avg-W2V

```
grid.fit(sent_vectors_train_avgw2v, y_train)

# examine the best model
print(grid.best_score_)
print(grid.best_params_)
```

```
0.48896063270159695
{'epsilon': 0.7, 'class_weight': 'balanced', 'alpha': 0.01}
```

```
#Plotting alpha v/s CV_error
a=pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
a['alpha'] = [d.get('alpha') for d in a['params']]
b=a.sort_values(['alpha'])
CV_Error=1-b['mean_test_score']
alpha =b['alpha']


plt.plot(alpha,CV_Error)
plt.xlabel('alpha')
plt.ylabel('Cross-Validated Error')
```

```
Text(0,0.5,'Cross-Validated Error')
```



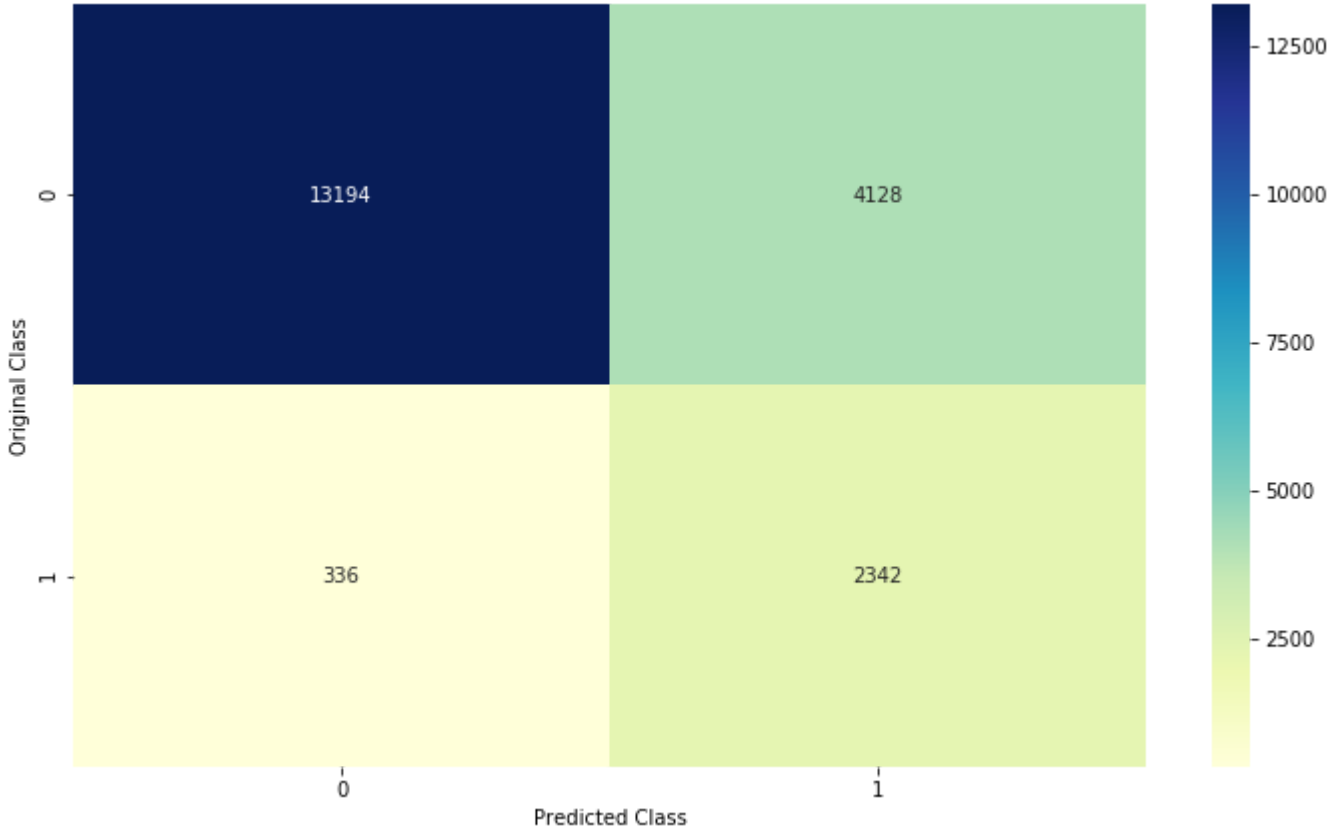
```
#{'epsilon': 0.7, 'class_weight': 'balanced', 'alpha': 0.01}
svm_optimal=SGDClassifier(alpha=0.01,epsilon=0.7,class_weight='balanced')

# fitting the model
svm_optimal.fit(sent_vectors_train_avgw2v, y_train)

# predict the response
pred_avg_w2v = svm_optimal.predict(sent_vectors_test_avgw2v)

# evaluate f1_score
f1_score = f1_score(y_test, pred_avg_w2v)

# Train & Test Error
print("The overall f1_score for the Train Data is : ", metrics.f1_score(y_train,svm_optimal.predict(sen
print("The overall f1_score for the Test Data is : ", metrics.f1_score(y_test,pred_avg_w2v))
```

```
The overall f1_score for the Train Data is :  0.4781548524054046
The overall f1_score for the Test Data is :  0.5021829411138323
```

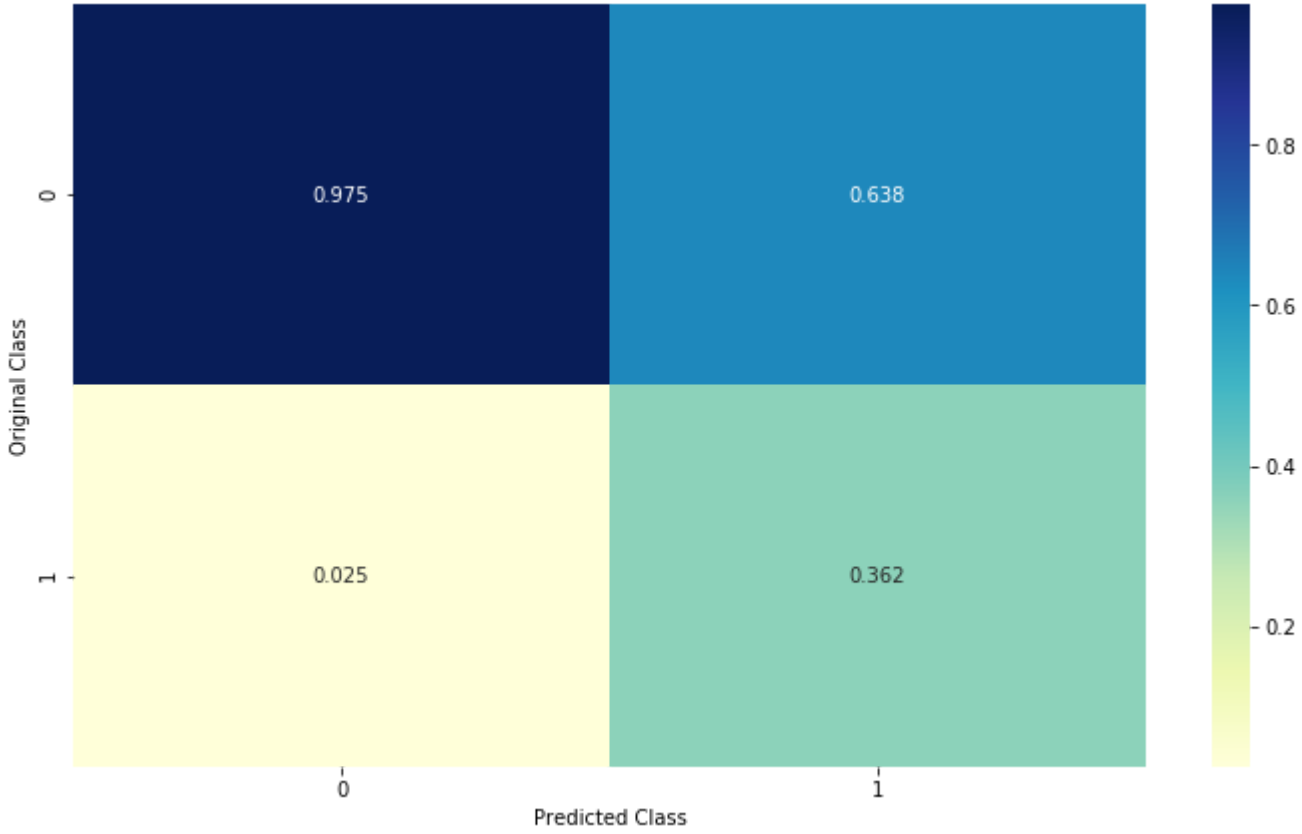
```
#Confusion matrix
C = confusion_matrix(y_test, pred_avg_w2v)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 print("-"*20, "Confusion matrix", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt="g", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
 plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

    # representing B in heatmap format
print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```

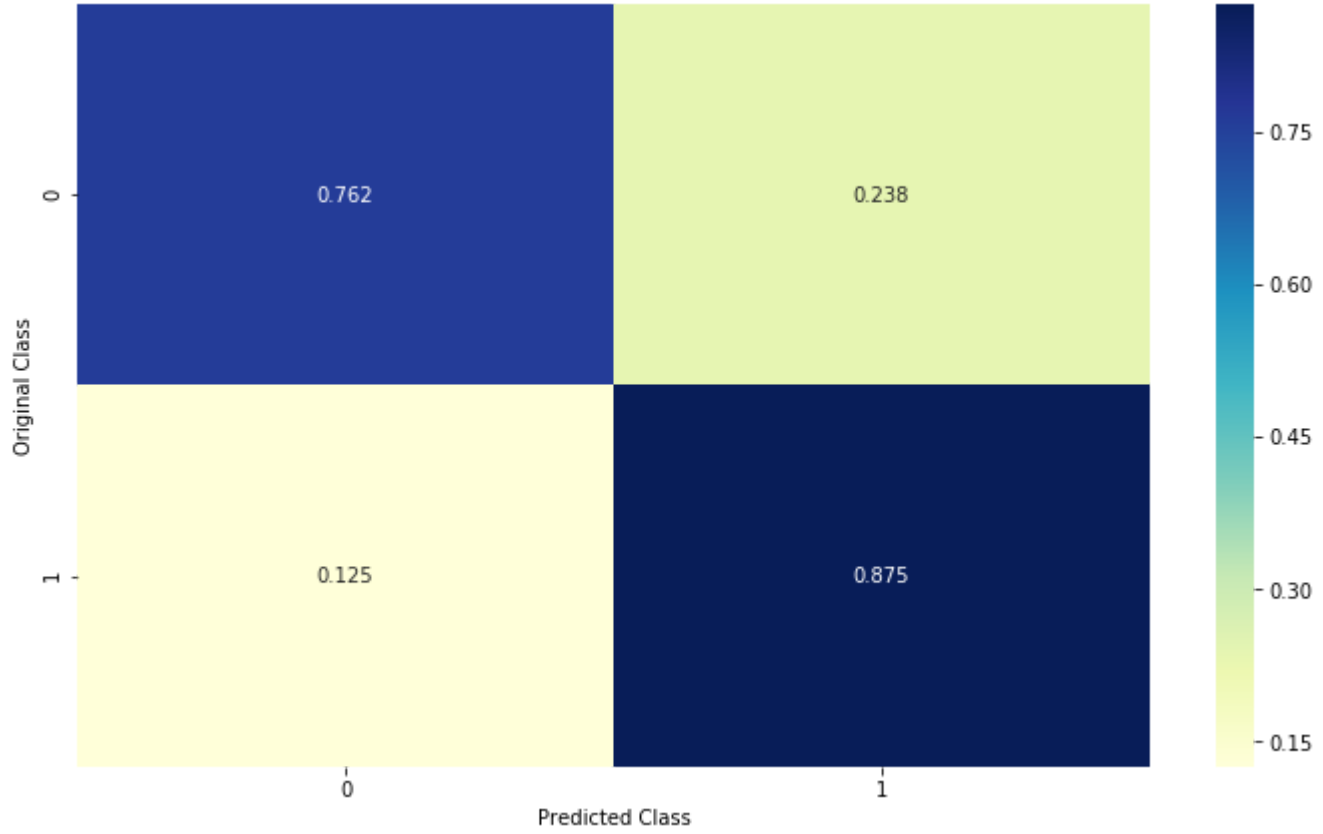
```
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



---------- Recall matrix (Row sum=1) ----------



TF-Idf W2Vec

```
tf_idf_vect = TfidfVectorizer()
vocabulary = tf_idf_vect.fit(train_data['CleanedText'])
final_tf_idf= tf_idf_vect.transform(train_data['CleanedText'])

# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(vocabulary.get_feature_names(), list(tf_idf_vect.idf_)))
```

```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_w2v_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_train): # 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_train:
            vec = w2v_model_train.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_w2v_sent_vectors_train.append(sent_vec)
    row += 1
```

```
100%|██████████| 80000/80000 [01:10<00:00, 1138.03it/s]
```

```
final_tf_idf= tf_idf_vect.transform(test_data['CleanedText'])

tfidf_w2v_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_test): # 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_train:
            vec = w2v_model_train.wv[word]
            # obtain the tf_idfidf of a word in a sentence/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_w2v_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 20000/20000 [00:17<00:00, 1123.51it/s]
```

```
#Standardizing
from sklearn.preprocessing import StandardScaler
Standard=StandardScaler()
tfidf_w2v_sent_vectors_train = Standard.fit_transform(tfidf_w2v_sent_vectors_train)
tfidf_w2v_sent_vectors_test = Standard.transform(tfidf_w2v_sent_vectors_test)

print(tfidf_w2v_sent_vectors_train.shape)
print(tfidf_w2v_sent_vectors_test.shape)
```

```
(80000, 50)
(20000, 50)
```

Fitting grid search cv on tfidf-w2vec

```
grid.fit(tfidf_w2v_sent_vectors_train, y_train)

# examine the best model
print(grid.best_score_)
print(grid.best_params_)
```

```
0.46358605223865923
{'epsilon': 0.1, 'class_weight': 'balanced', 'alpha': 0.01}
```

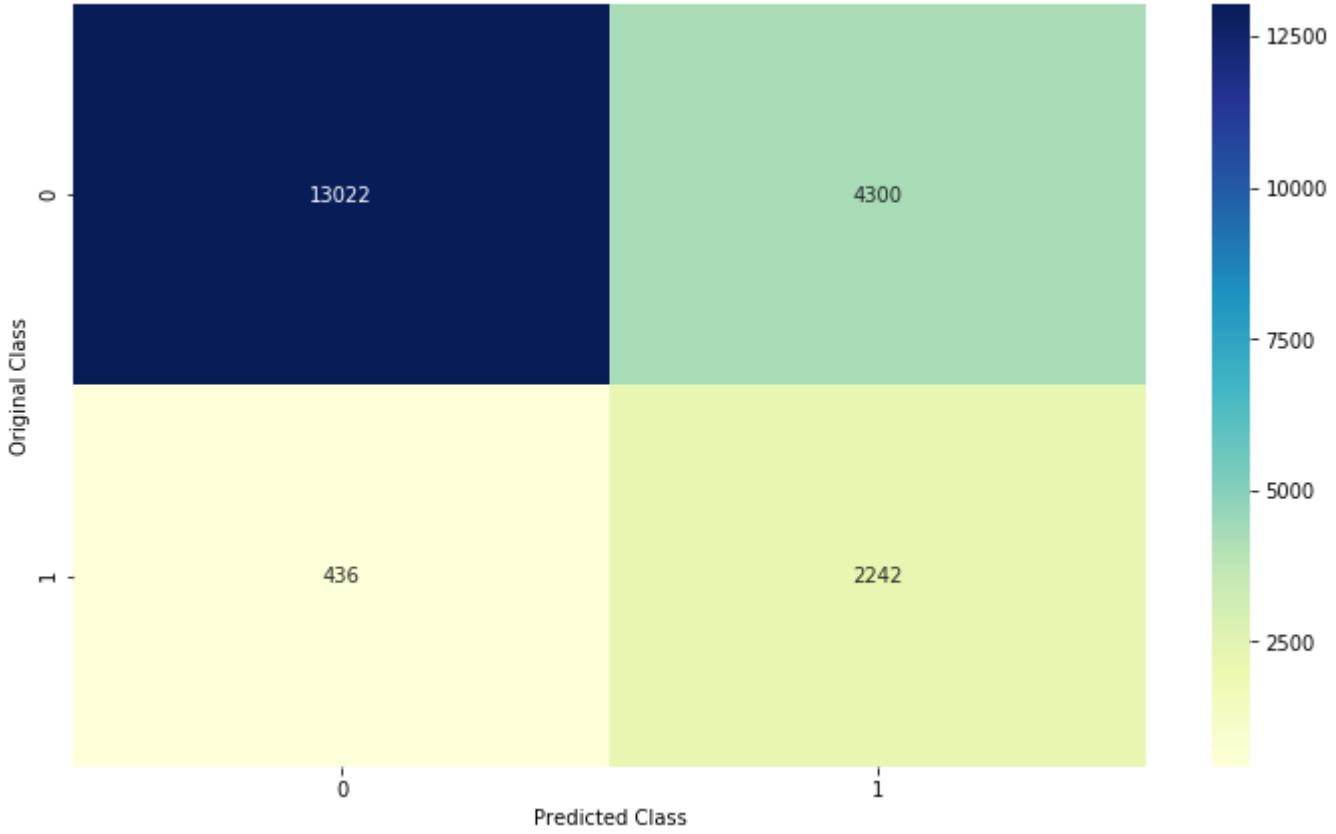
```
from sklearn.metrics import f1_score
#{'epsilon': 0.1, 'class_weight': 'balanced', 'alpha': 0.01}
svm_optimal=SGDClassifier(alpha=0.01,epsilon=0.1,class_weight='balanced')

# fitting the model
svm_optimal.fit(tfidf_w2v_sent_vectors_train, y_train)

# predict the response
pred_tfidf_w2v_sent_vectors_test = svm_optimal.predict(tfidf_w2v_sent_vectors_test)

# evaluate f1_score
f1_score = f1_score(y_test, pred_tfidf_w2v_sent_vectors_test)

# Train & Test Error
print("The overall f1_score for the Train Data is : ", metrics.f1_score(y_train,svm_optimal.predict(tfi
print("The overall f1_score for the Test Data is : ", metrics.f1_score(y_test,pred_tfidf_w2v_sent_vecto
```

```
The overall f1_score for the Train Data is :  0.458559030811346
The overall f1_score for the Test Data is :  0.4863340563991323
```

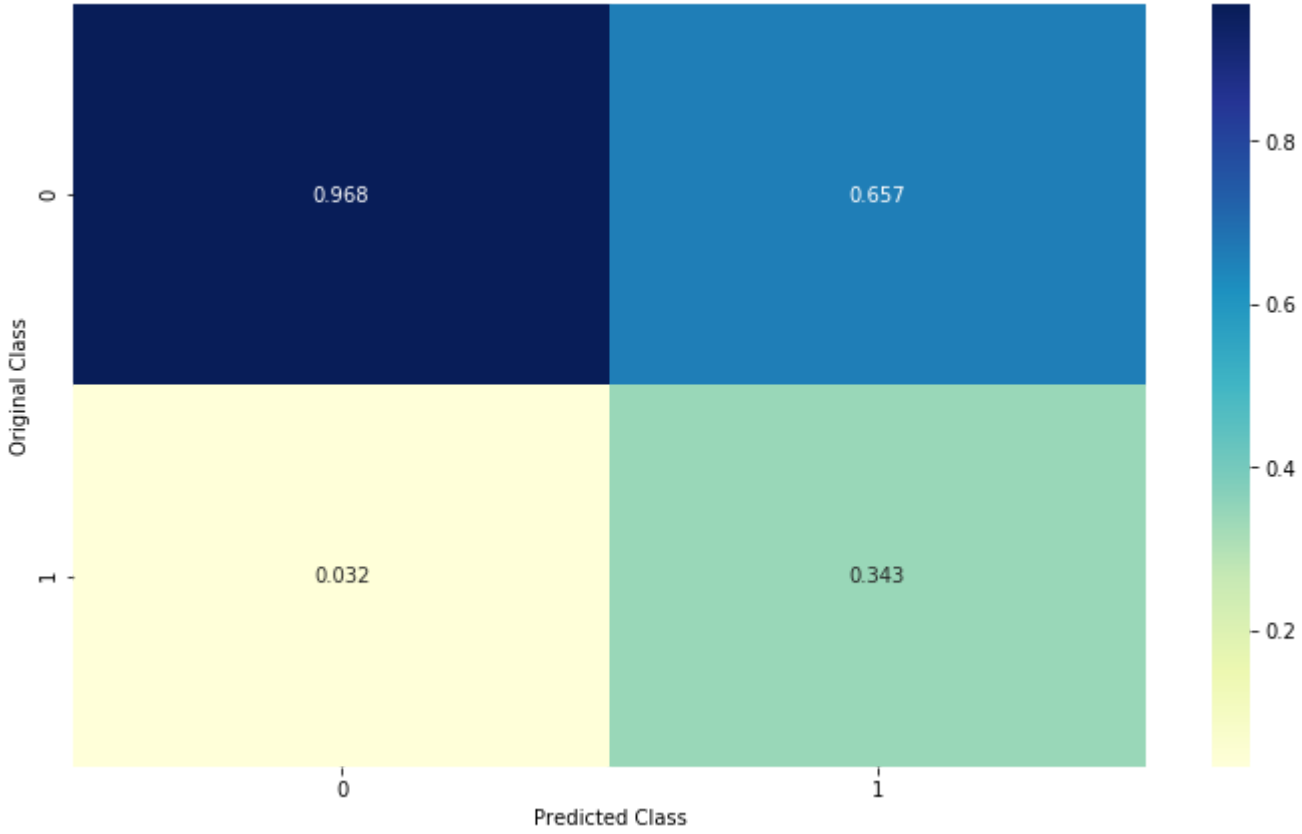
```
#Confusion matrix
C = confusion_matrix(y_test, pred_tfidf_w2v_sent_vectors_test)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 print("-"*20, "Confusion matrix", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt="g", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
 plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

    # representing B in heatmap format
print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```

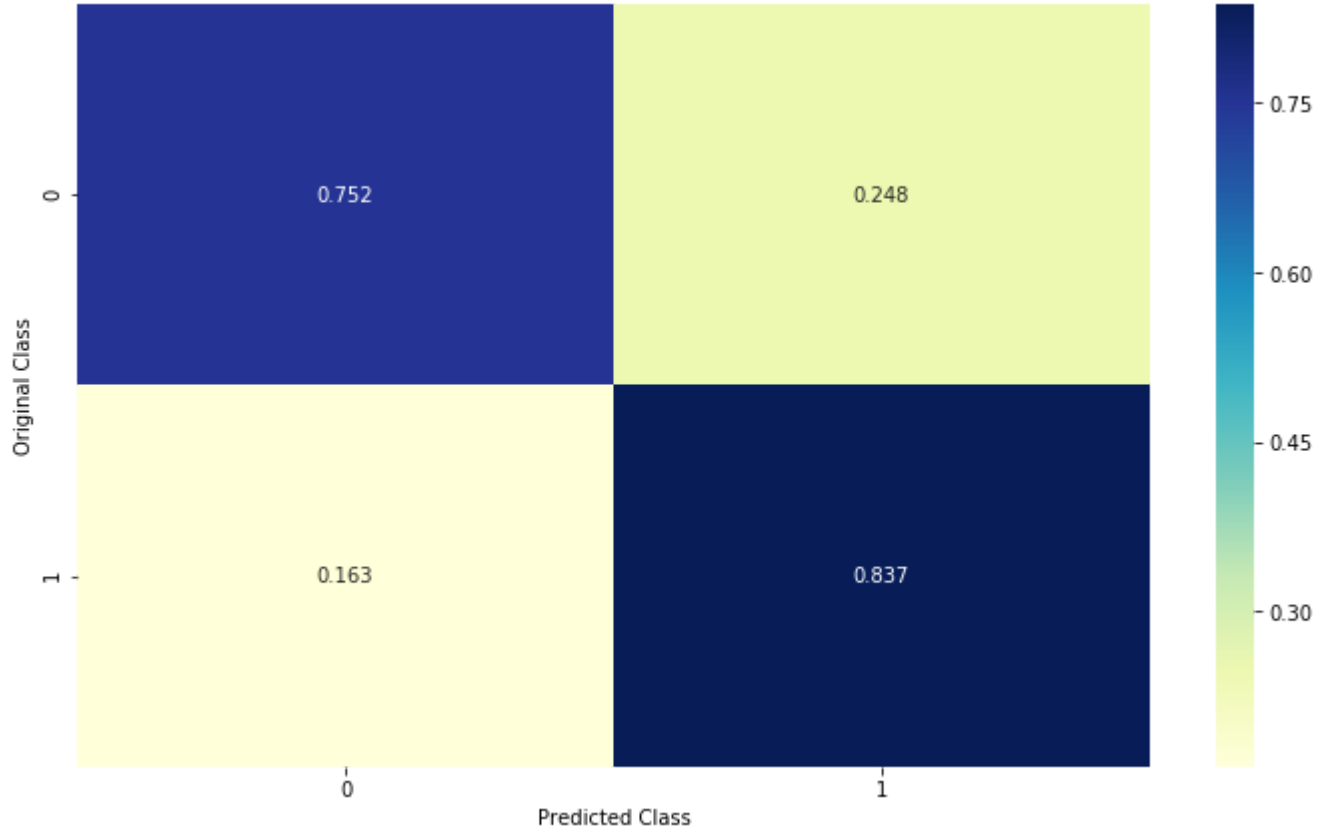
```
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



---------- Recall matrix (Row sum=1) ----------



Reporting f1_score after running SGD classifier on above featurization

```
1 from prettytable import PrettyTable
```

```
x=PrettyTable()
x.field_names = ["Model","Bow", "Tfidf", "Avg-W2V", "Tfidf-W2V"]
x.add_row(["Alpha",3,5,0.01,0.01])
x.add_row(["Train f1_score",0.748,0.998,0.478,0.458])
x.add_row(["Test f1_score",0.693,0.582,0.502,0.486])

print(x)
```

```
+----------------+-------+-------+---------+-----------+
|     Model      |  Bow  | Tfidf | Avg-W2V | Tfidf-W2V |
+----------------+-------+-------+---------+-----------+
|     Alpha      |   3   |   5   |   0.01  |    0.01   |
| Train f1_score | 0.748 | 0.998 |  0.478  |   0.458   |
| Test f1_score  | 0.693 | 0.582 |  0.502  |   0.486   |
+----------------+-------+-------+---------+-----------+
```

We can see that BOW has the best f1_score therefore we will run SVC classifier on it.

Randomised search cv

```
C = [0.000001,0.00001,0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1,2,3,4,5,6,7,8,9,10,15
T= TimeSeriesSplit(n_splits=3)
weight=[None,'balanced']

param_distributions = dict(C=C,class_weight=weight)
print(param_distributions)

# instantiate and fit the grid
grid = RandomizedSearchCV(SVC(), param_distributions, cv=T, scoring='f1', return_train_score=False,n_jol
```

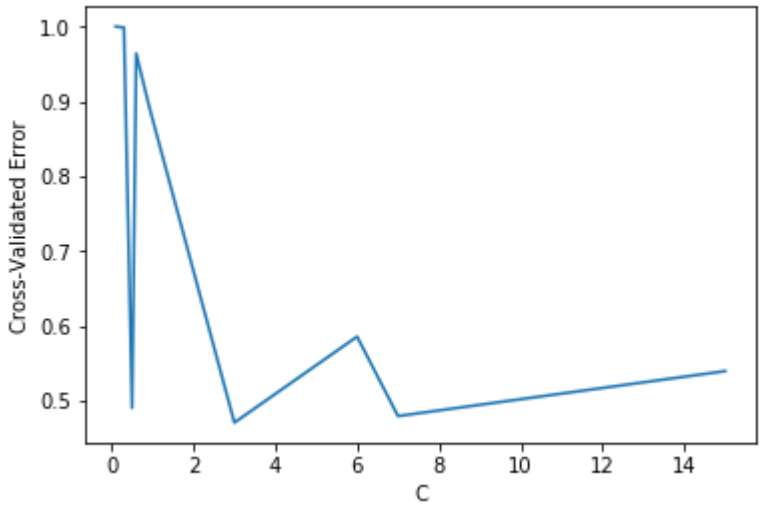
```
{'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
15, 20, 30, 40, 50], 'class_weight': [None, 'balanced']}
```

```
grid.fit(Bow_x_train, y_train)

# examine the best model
print(grid.best_score_)
print(grid.best_params_)
```

```
0.5289631785421901
{'class_weight': 'balanced', 'C': 3}
```

```
#Plotting alpha v/s CV_error
a=pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
a['C'] = [d.get('C') for d in a['params']]
b=a.sort_values(['C'])
CV_Error=1-b['mean_test_score']
C =b['C']


plt.plot(C,CV_Error)
plt.xlabel('C')
plt.ylabel('Cross-Validated Error')
```

```
Text(0,0.5,'Cross-Validated Error')
```



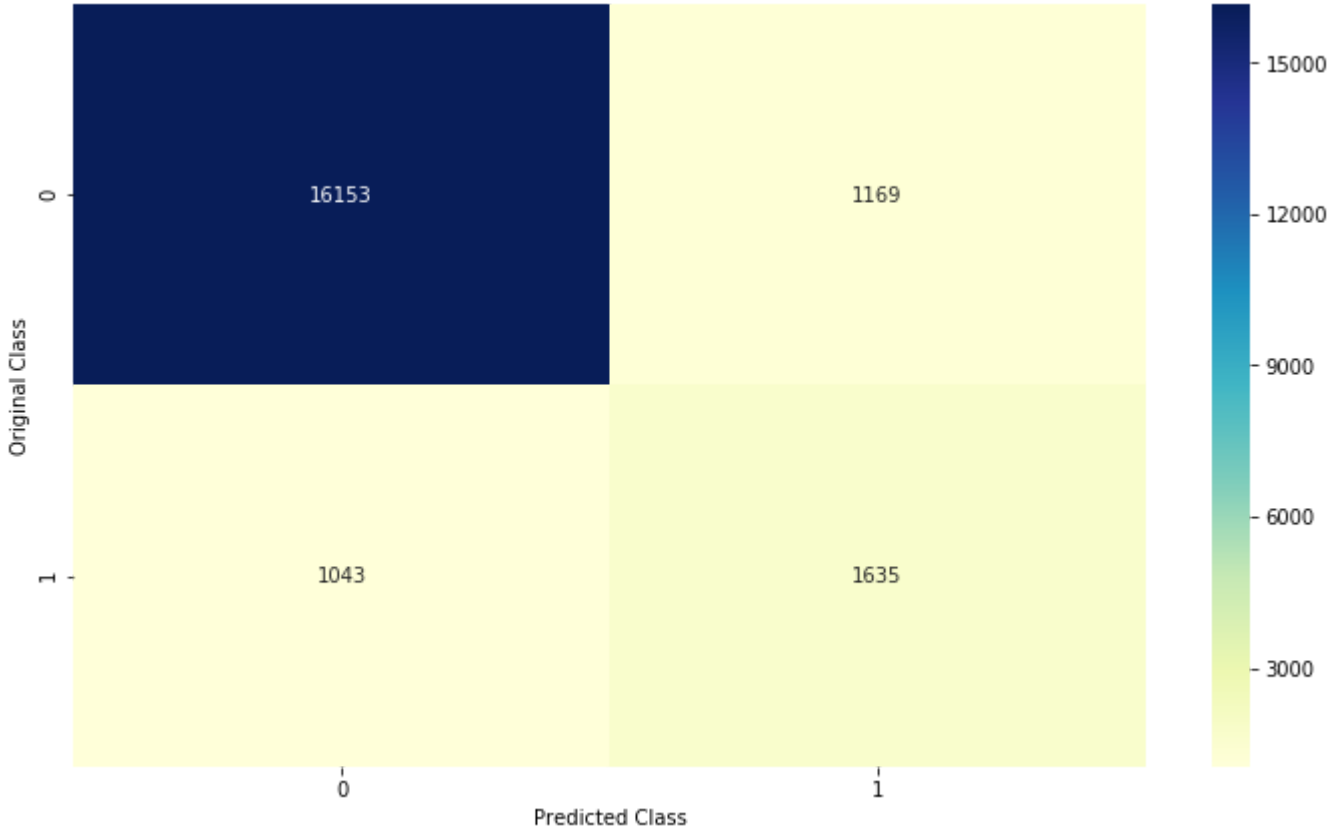
```
#{'class_weight': 'balanced', 'C': 3}
svm_optimal=SVC(C=3,class_weight='balanced')

# fitting the model
svm_optimal.fit(Bow_x_train, y_train)

# predict the response
pred_bow = svm_optimal.predict(Bow_x_test)

# evaluate f1_score
f1_score = f1_score(y_test, pred_bow)

# Train & Test Error
print("The overall f1_score for the Train Data is : ", metrics.f1_score(y_train,svm_optimal.predict(Bow_
print("The overall f1_score for the Test Data is : ", metrics.f1_score(y_test,pred_bow))
```

```
The overall f1_score for the Train Data is :  0.8612301957129542
The overall f1_score for the Test Data is :  0.5964976286026997
```

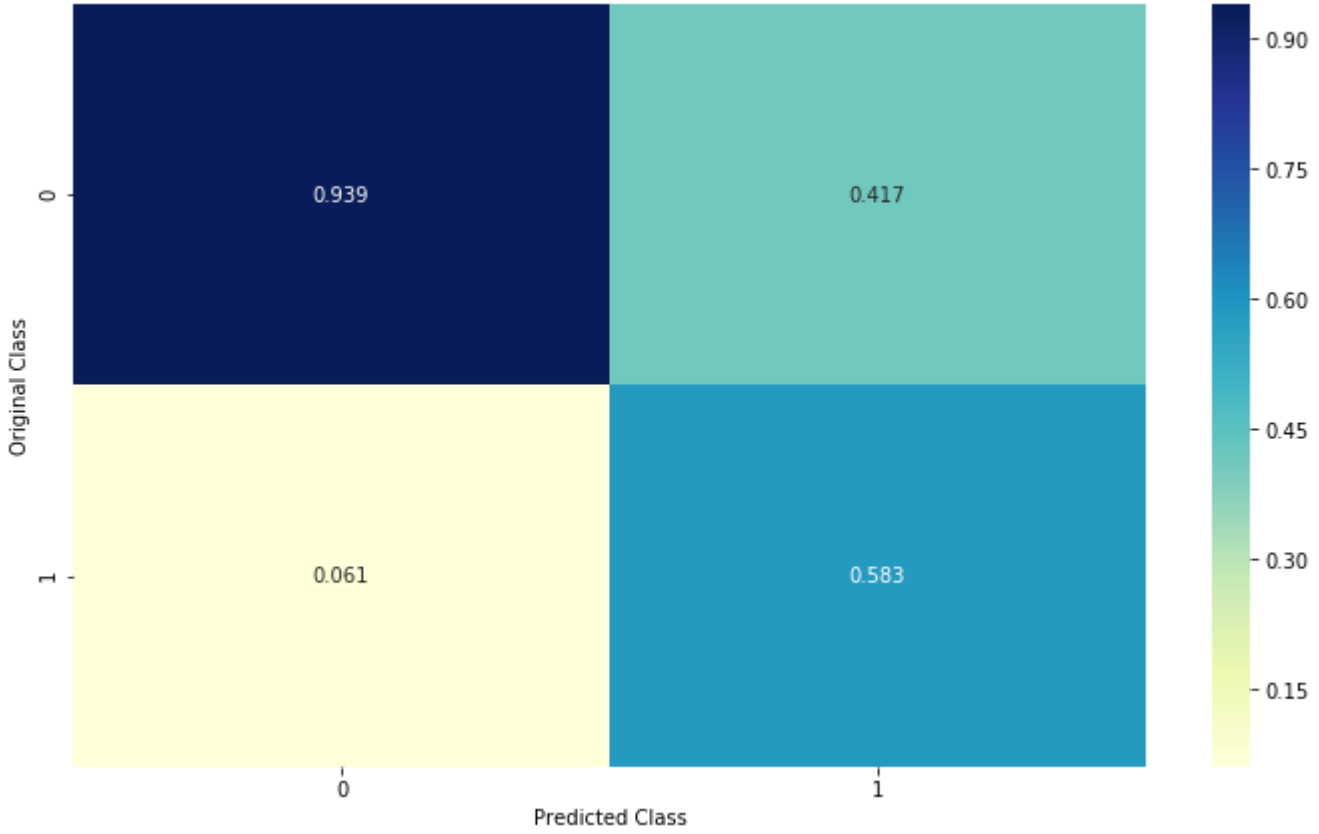
```
#Confusion matrix
C = confusion_matrix(y_test, pred_bow)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 print("-"*20, "Confusion matrix", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt="g", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
 plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

    # representing B in heatmap format
print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
plt.figure(figsize=(12,7))
sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```

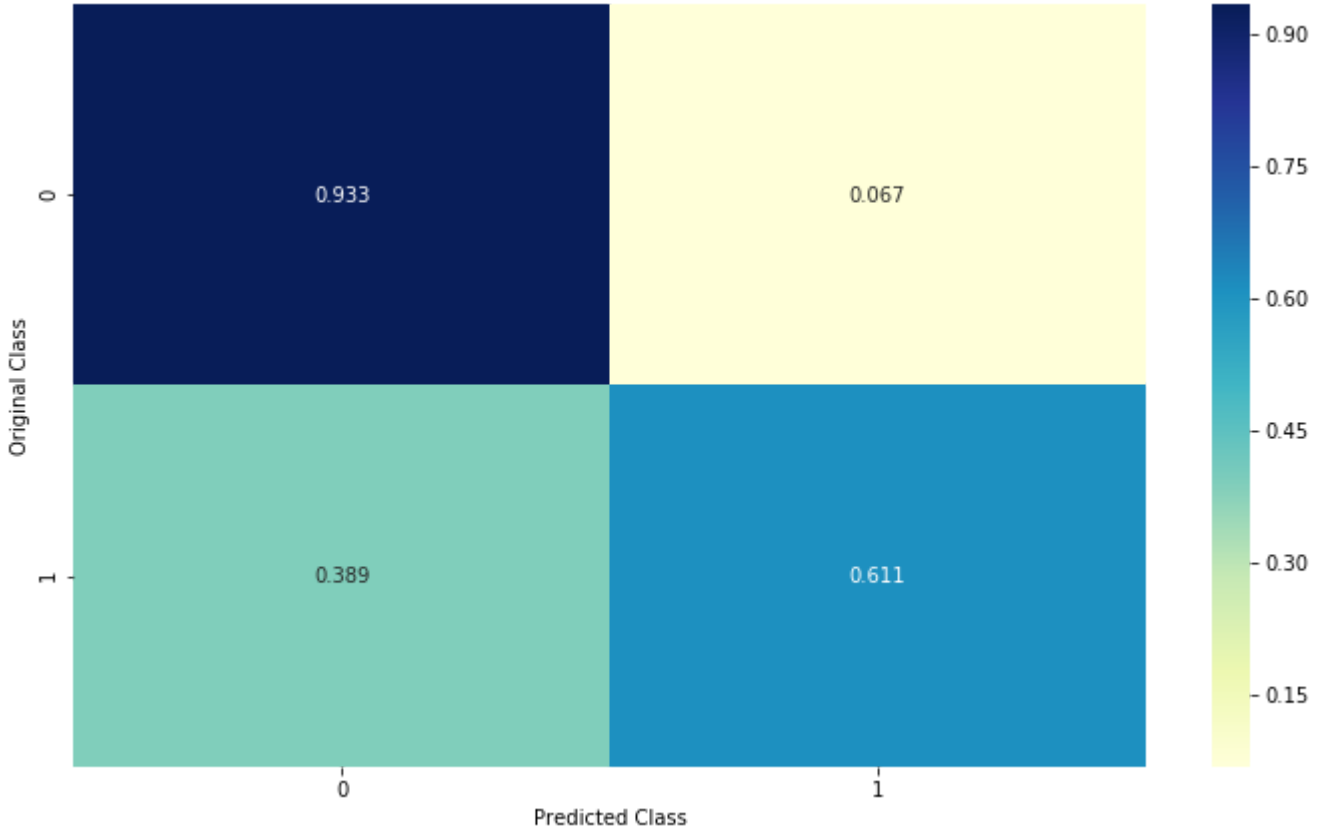
```
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



---------- Recall matrix (Row sum=1) ----------


0.933
0.067
0.389
0.611
Original Class
Predicted Class
0
1
0
1
0.90
0.75
0.60
0.45
0.30
0.15