Logistic Regression 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 Logistic regression using L2 regularization 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
import os
```

```
#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']
```

```python
#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)
```

```python
from sklearn.linear_model import LogisticRegression
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
from tqdm import tqdm
```

Applying L2 regularization

Randomised CV using L2

```
gamma_range = [0.0000000001,0.000000001,0.00000001,0.0000001,0.000001,0.00001,0.0001,0.001,0.01,0.1,0.2,
T= TimeSeriesSplit(n_splits=5)
weight=[None,'balanced']

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

# instantiate and fit the grid
grid = RandomizedSearchCV(LogisticRegression(penalty='l2'), param_distributions, cv=T, scoring='f1', re
```

```
{'C': [1e-10, 1e-09, 1e-08, 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, 5,
10, 15, 20, 30, 40, 50], '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
Scaler=StandardScaler(with_mean=False)
Bow_x_train = Scaler.fit_transform(Bow_x_train)
Bow_x_test = Scaler.transform(Bow_x_test)

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

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

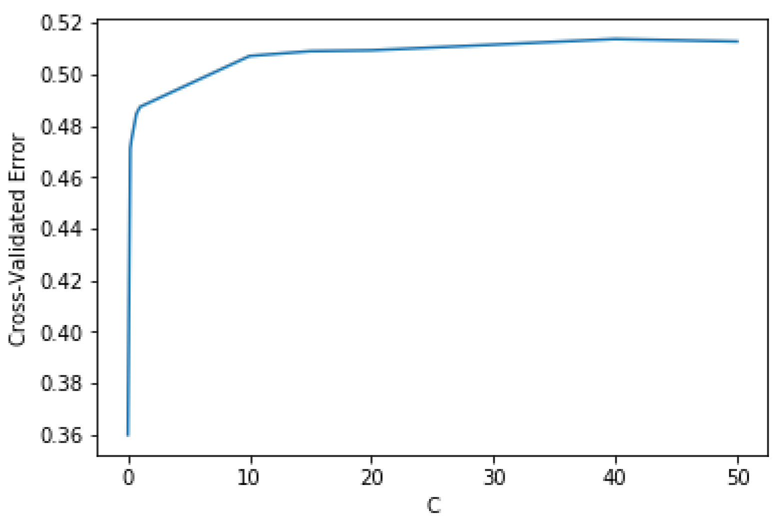
Fitting Grid Search CV on BOW

```
grid.fit(Bow_x_train, y_train)

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

```
0.6401387857438695
{'class_weight': 'balanced', 'C': 0.0001}
```

```
#Plotting C 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': 0.0001}
LR_optimal=LogisticRegression(penalty='l2',C=0.0001,class_weight='balanced')

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

# predict the response
pred_bow = LR_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,LR_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 :  0.8089229382604777
The overall f1_score for the Test Data is :  0.675998091299507
```

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)
```

```

#{'class_weight': 'balanced', 'C': 0.0001}
LR_optimal_noise=LogisticRegression(penalty='l2',C=0.0001,class_weight='balanced')

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

# predict the response
pred_bow = LR_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,LR_optimal.predict(Nois
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 :  0.8088692595430168
The overall f1_score for the Test Data is :  0.675998091299507
```

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

```
(33433,)
```

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

```
(1, 33433)
```

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

```
(1, 33433)
```

```
merge_arr = np.concatenate([LR_optimal.coef_, LR_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	-0.001722	-0.001722	0.002999
aaaaaaaaagghh	-0.002087	-0.002087	-0.012497
aaaaah	-0.002006	-0.002006	0.005672
aaaaahhhhhhhhhhhhhhh	-0.000765	-0.000765	0.000203
aaaah	-0.000964	-0.000964	-0.003298
aaah	-0.001224	-0.001224	-0.004986
aachen	0.005927	0.005927	-0.008612
aad	-0.000364	-0.000364	-0.071263
aadp	-0.001017	-0.001017	-0.024636
aafco	-0.000879	-0.000878	-0.090446
aagh	-0.003549	-0.003549	-0.004424
aah	-0.002004	-0.002004	0.000561
aahh	-0.001183	-0.001182	-0.024529
aand	-0.001929	-0.001929	-0.005073
aardvark	-0.003908	-0.003909	0.014199
aarrgh	0.009679	0.009679	0.000378
ab	-0.002795	-0.002795	0.003201
aback	-0.006500	-0.006501	0.003168
abandon	0.003590	0.003590	-0.012733
abaolut	-0.001443	-0.001442	-0.028476
abattoir	-0.000879	-0.000880	0.026223
abba	-0.002686	-0.002686	-0.004696
abbey	-0.002013	-0.002013	0.001276
abbi	-0.003064	-0.003064	-0.006595

	0	1	2
abbott	-0.000479	-0.000479	-0.044658
abbrevi	-0.000541	-0.000541	-0.002948
abc	0.000694	0.000695	0.252658
abcstor	-0.001687	-0.001687	0.020410
abd	-0.001774	-0.001774	-0.028242
abdomen	0.000109	0.000109	-0.001692
...	...	...	...
zot	-0.008788	-0.008788	0.005860
zotz	-0.004918	-0.004919	0.003821
zour	0.002518	0.002518	0.000398
zout	-0.001946	-0.001946	-0.022958
zowi	-0.000718	-0.000718	-0.045905
zreport	-0.004741	-0.004741	0.000733
zsweet	-0.002095	-0.002094	-0.028194
zuc	-0.002182	-0.002182	-0.009051
zucchini	0.004017	0.004017	0.002359
zuccini	-0.004626	-0.004625	-0.005549
zuccnini	-0.000321	-0.000321	-0.013392
zuchinni	-0.003164	-0.003164	0.001102
zuke	-0.001739	-0.001739	-0.015763
zulu	-0.001037	-0.001037	-0.007865
zum	-0.000321	-0.000320	-0.059621
zummi	-0.000321	-0.000320	-0.059621
zune	-0.003632	-0.003632	-0.002227
zupreem	-0.000610	-0.000610	-0.001370
zurich	-0.001767	-0.001767	-0.010648
zwar	-0.000111	-0.000111	0.007019
zwieback	0.003339	0.003340	0.018697
zwiebeck	-0.002276	-0.002275	-0.025352

	0	1	2
zydeco	-0.001672	-0.001672	0.003113
zzzzzs	-0.002309	-0.002309	0.006540
zzzzzz	-0.000076	-0.000076	-0.063760
zzzzzzz	0.007196	0.007195	-0.003373
zzzzzzzz	-0.000630	-0.000630	0.014911
zzzzzzzzzz	-0.000379	-0.000379	-0.081017
zzzzzzzzzzz	-0.002822	-0.002822	0.004119
çay	-0.001229	-0.001229	-0.016318

33433 rows × 3 columns

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

(3, 3)

3 features out of 33433 shows percentage change > 30 post pertubation test i.e 0.0089%

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

```
feature_names = np.array(vocabulary.get_feature_names())
sorted_coef_index = LR_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)
```

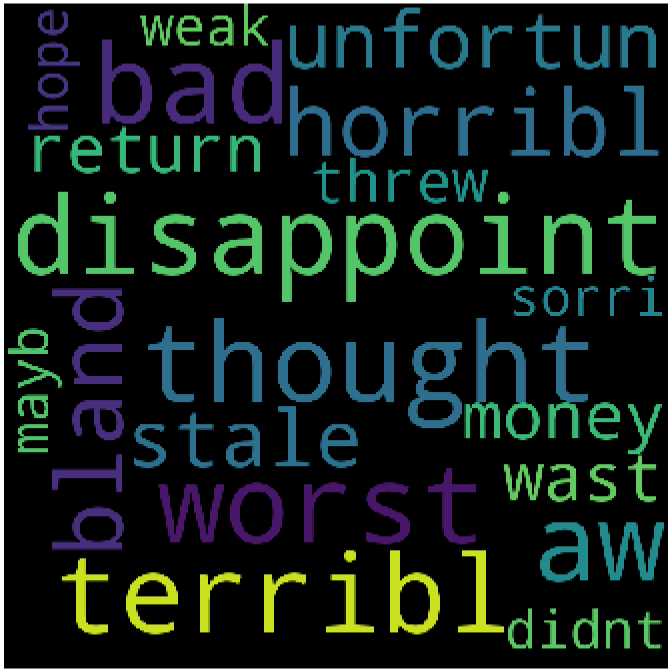
great,love,best,delici,perfect,excel,good,favorit,nice,wonder,find,tasti,easi,amaz,thank,addict,alway,keep,year,snack,

```
n=feature_names[sorted_coef_index[:-21:-1]]

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

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

```python
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 ******************
```

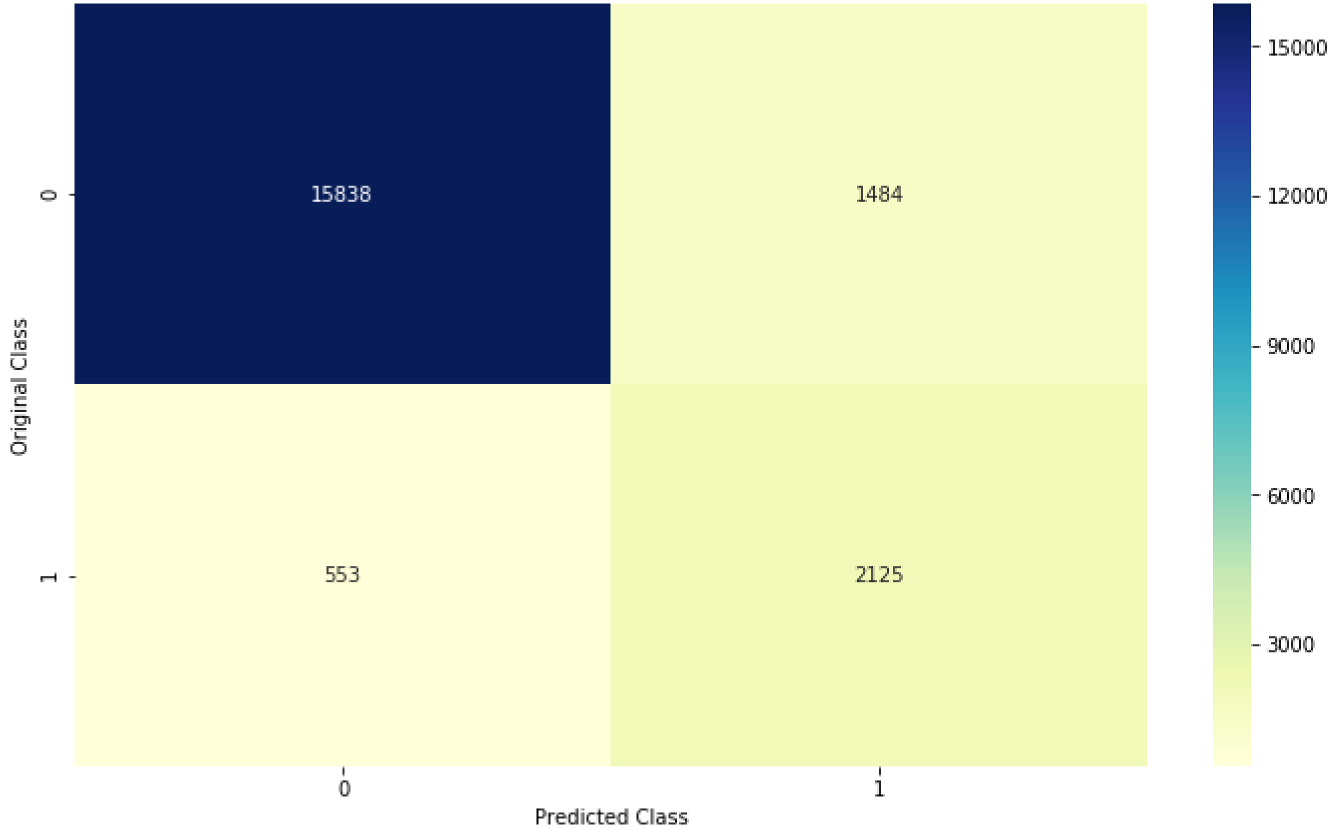



```
************ Top 20 Positive words ******************
```

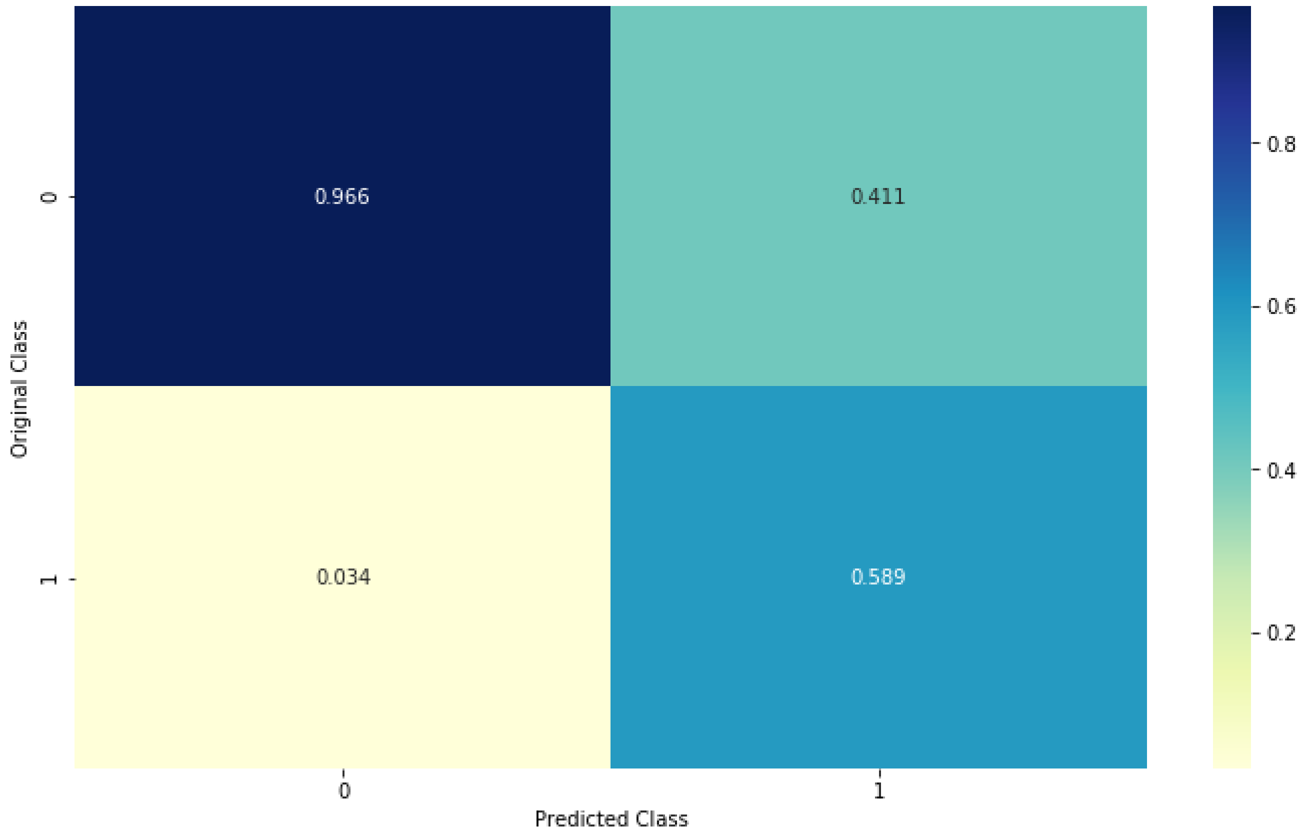

```
#Confusion matrix
C = confusion_matrix(y_test, pred_bow)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

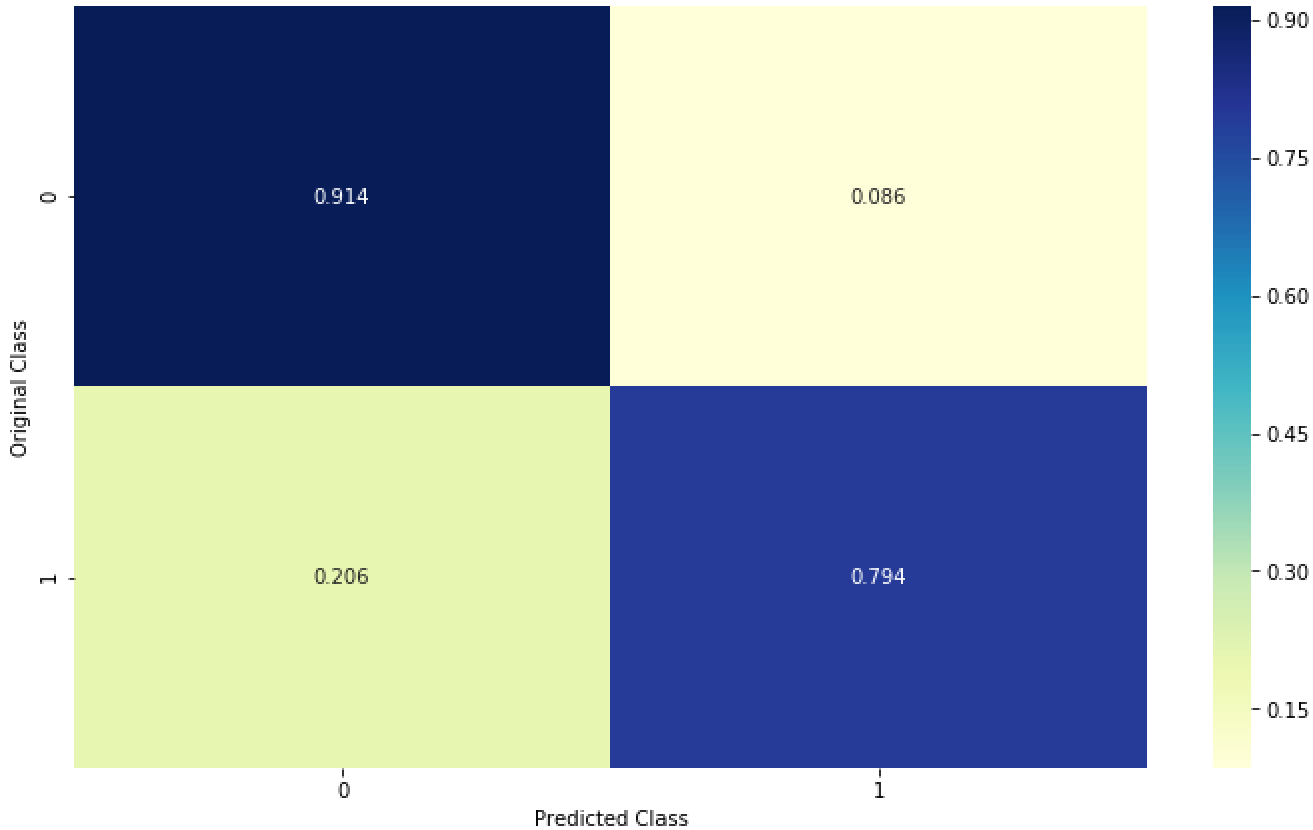
```
-------------------- Confusion matrix --------------------
```




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




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



0.914
0.086
0.206
0.794
Original Class
Predicted Class
0
1
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)
```

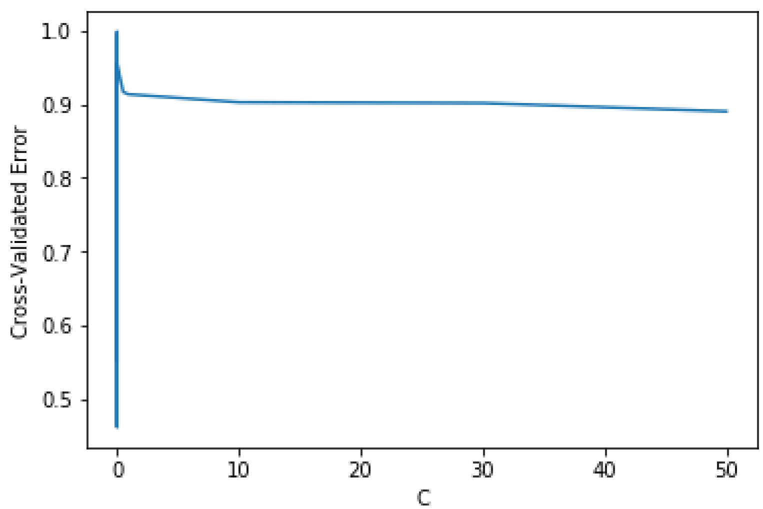
Fitting Randomsearch on Tf-Idf

```
grid.fit(Tfidf_x_train, y_train)

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

```
0.5390181835332085
{'class_weight': 'balanced', 'C': 1e-07}
```

```
#Plotting C 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': 1e-07}
LR_optimal=LogisticRegression(penalty='l2',C=0.0000001,class_weight='balanced')

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

# predict the response
pred_tfidf = LR_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,LR_optimal.predict(Tfid
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 :  0.9030708364732479
The overall f1_score for the Test Data is :  0.5339728217426059
```

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
```

```python

#{'class_weight': 'balanced', 'C': 1e-07}
LR_optimal_noise=LogisticRegression(penalty='l2',C=0.0000001,class_weight='balanced')

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

# predict the response
pred_tfidf = LR_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,LR_optimal_noise.predic
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 :  0.9030283049969386
The overall f1_score for the Test Data is :  0.5340948029697316
```

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

```
(1013943,)
```

```python
LR_optimal.coef_.shape
```

```
(1, 1013943)
```

```python
LR_optimal_noise.coef_.shape
```

```
(1, 1013943)
```

```
merge_arr = np.concatenate([LR_optimal.coef_, LR_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	-0.000010	-0.000010	0.003393
aaa condit	-0.000008	-0.000008	0.002404
aaa perfect	-0.000008	-0.000008	0.000782
aaaaaaaaagghh	-0.000008	-0.000008	0.000995
aaaaah	-0.000011	-0.000011	0.002724
aaaaah awak	-0.000008	-0.000008	0.000999
aaaaah satisfi	-0.000008	-0.000008	0.002002
aaaaahhhhhhhhhhhhhhh	-0.000008	-0.000008	0.001810
aaaaahhhhhhhhhhhhhhh angel	-0.000008	-0.000008	0.001810
aaaah	-0.000007	-0.000007	-0.000594
aaaah snob	-0.000007	-0.000007	-0.000594
aaah	-0.000013	-0.000013	0.002666
aaah inhal	-0.000008	-0.000008	0.000134
aaah miss	-0.000008	-0.000008	0.000481
aaah sip	-0.000008	-0.000008	0.000763
aachen	0.000045	0.000045	0.000334
aachen munich	0.000045	0.000045	0.000334
aad	-0.000008	-0.000008	-0.001176
aad sausag	-0.000008	-0.000008	-0.001176
aadp	-0.000008	-0.000008	0.000288
aafco	0.000002	0.000002	0.041650
aafco also	-0.000007	-0.000007	-0.002049
aafco certifi	0.000018	0.000018	0.001439
aafco countri	-0.000007	-0.000007	-0.002049

	0	1	2
aafco definit	0.000042	0.000042	0.001049
aafco dog	-0.000008	-0.000008	0.000260
aafco guidelin	-0.000008	-0.000008	-0.000410
aafco requir	-0.000008	-0.000008	0.001271
aagh	-0.000008	-0.000008	0.002790
aagh yelp	-0.000008	-0.000008	0.002790
...	...	...	...
zum heal	-0.000008	-0.000008	0.000441
zummi	-0.000008	-0.000008	0.000441
zummi love	-0.000008	-0.000008	0.000441
zummi tast	-0.000008	-0.000008	0.000441
zummi tri	-0.000008	-0.000008	0.000441
zune	-0.000008	-0.000008	0.002882
zune video	-0.000008	-0.000008	0.002882
zupreem	-0.000008	-0.000008	0.000569
zupreem ferret	-0.000008	-0.000008	0.000569
zurich	-0.000008	-0.000008	0.001963
zurich schnatzlet	-0.000008	-0.000008	0.001963
zwar	-0.000007	-0.000007	0.002107
zwar billig	-0.000007	-0.000007	0.002107
zwieback	0.000022	0.000022	0.004092
zwieback toast	0.000022	0.000022	0.004092
zwiebeck	-0.000008	-0.000008	0.001889
zwiebeck toast	-0.000008	-0.000008	0.001889
zydeco	-0.000008	-0.000008	0.001634
zydeco saturday	-0.000008	-0.000008	0.001634
zzzzzs	-0.000011	-0.000011	0.002403
zzzzzs larg	-0.000008	-0.000008	0.002258
zzzzzz	-0.000007	-0.000007	-0.001191

	0	1	2
zzzzzz say	-0.000007	-0.000007	-0.001191
zzzzzzz	0.000053	0.000053	0.001704
zzzzzzz high	0.000053	0.000053	0.001704
zzzzzzzz	-0.000007	-0.000007	0.002698
zzzzzzzzzz	-0.000008	-0.000008	-0.000450
zzzzzzzzzz final	-0.000008	-0.000008	-0.000450
zzzzzzzzzzz	-0.000008	-0.000008	0.001947
çay	-0.000008	-0.000008	0.000097

1013943 rows × 3 columns

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

```
(1, 3)
```

1 features out of 1013943 shows percentage change > 30 post pertubation test i.e 0%

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

```
feature_names = np.array(vocabulary.get_feature_names())
sorted_coef_index = LR_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)
```

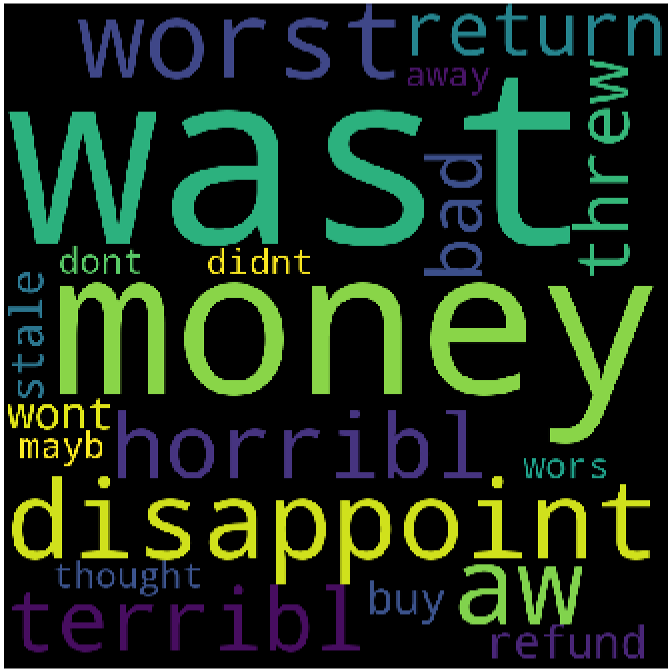
```
great,love,best,delici,perfect,favorit,good,find,make,high recommend,easi,excel,wonder,nice,use,snack,alway,keep,add,tasti,
```

```
n=feature_names[sorted_coef_index[:-21:-1]]

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

```
disappoint,wast money,worst,wast,aw,horribl,terribl,return,threw,money,bad,refund,stale,wont buy,thought,mayb,didnt,dont wa
st,away,wors,
```

```
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 ******************
```

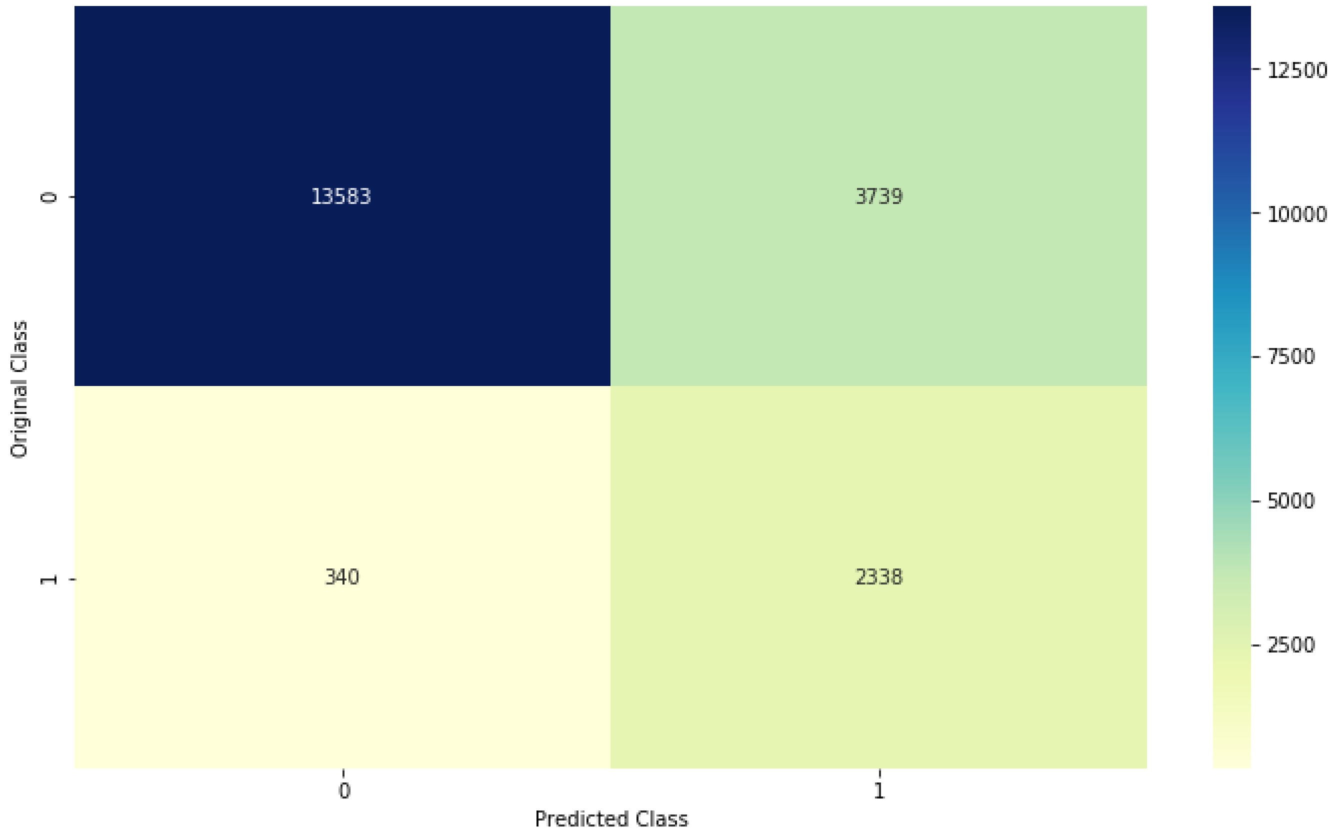



```
************ Top 20 Positive words *******************
```

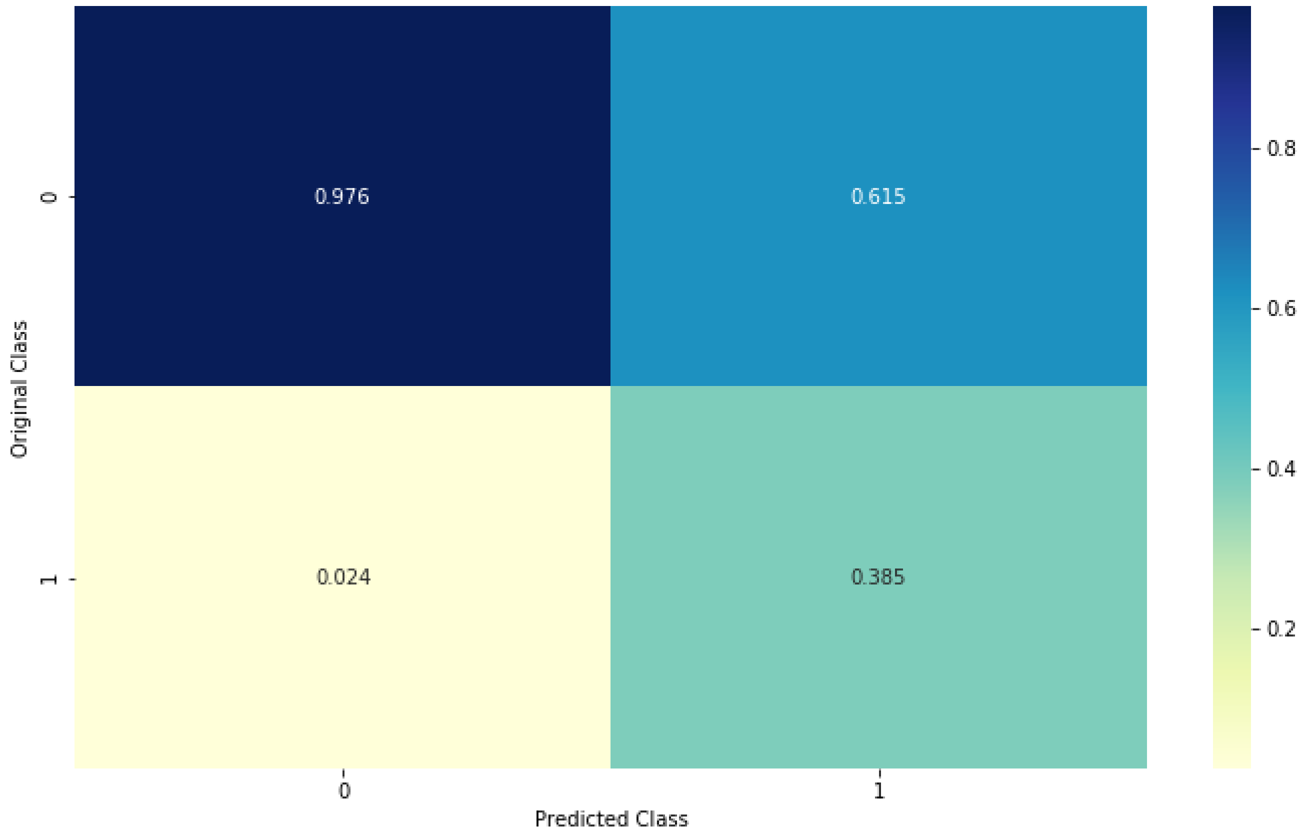

```
#Confusion matrix
C = confusion_matrix(y_test, pred_tfidf)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

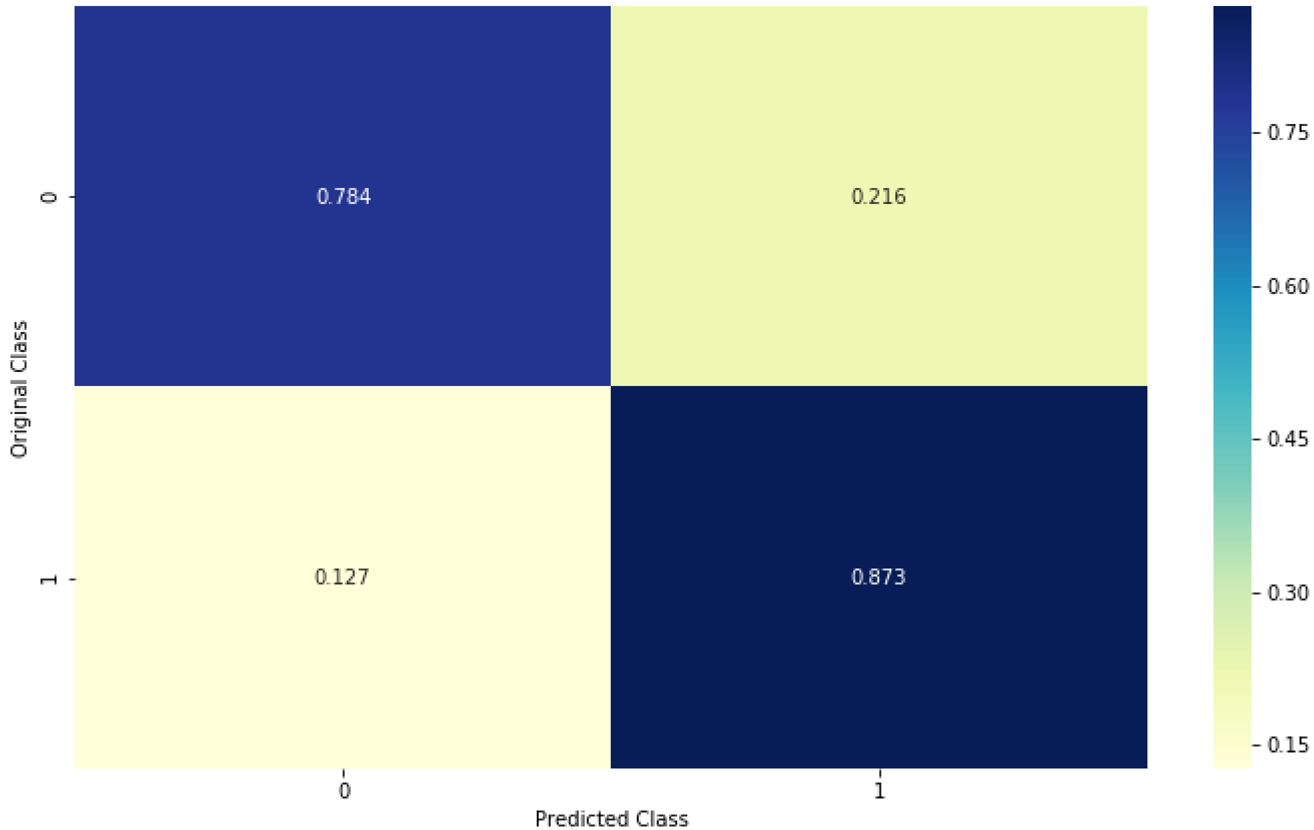
```
-------------------- Confusion matrix --------------------
```




```
-------------------- 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)
```

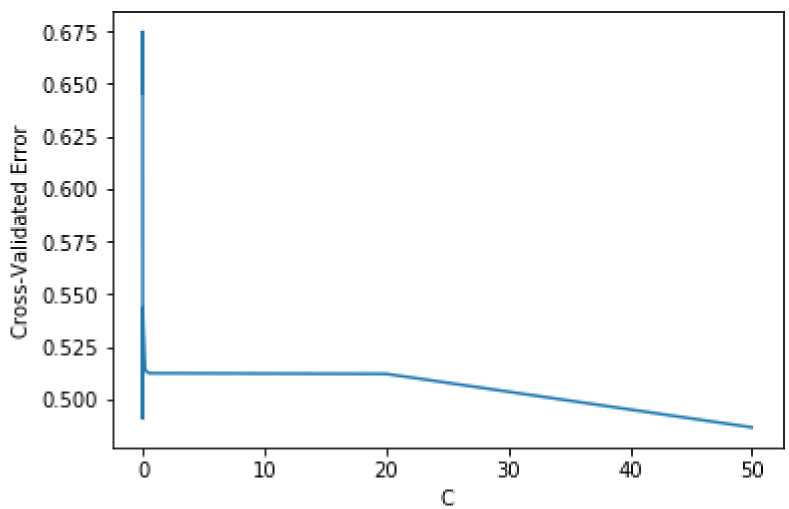
Fitting grid search on Avg-W2V

```
grid.fit(sent_vectors_train_avgw2v, y_train)

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

```
0.5134126417862019
{'class_weight': 'balanced', 'C': 50}
```

```
#Plotting C 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')
```



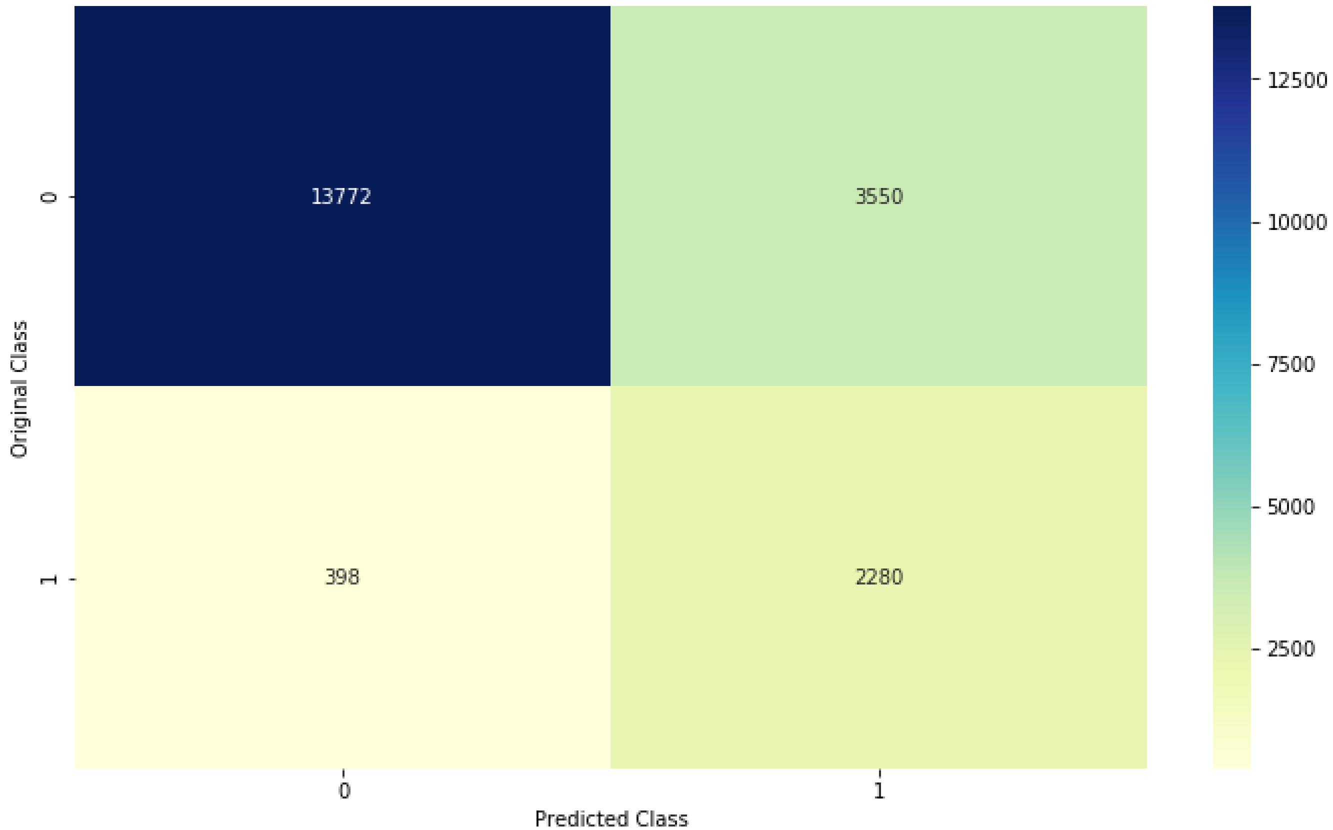
```
#{'class_weight': 'balanced', 'C': 50}
LR_optimal=LogisticRegression(penalty='l2',C=50,class_weight='balanced')

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

# predict the response
pred_avg_w2v = LR_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,LR_optimal.predict(sent_
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 :  0.5152863591253961
The overall f1_score for the Test Data is :  0.535966149506347
```

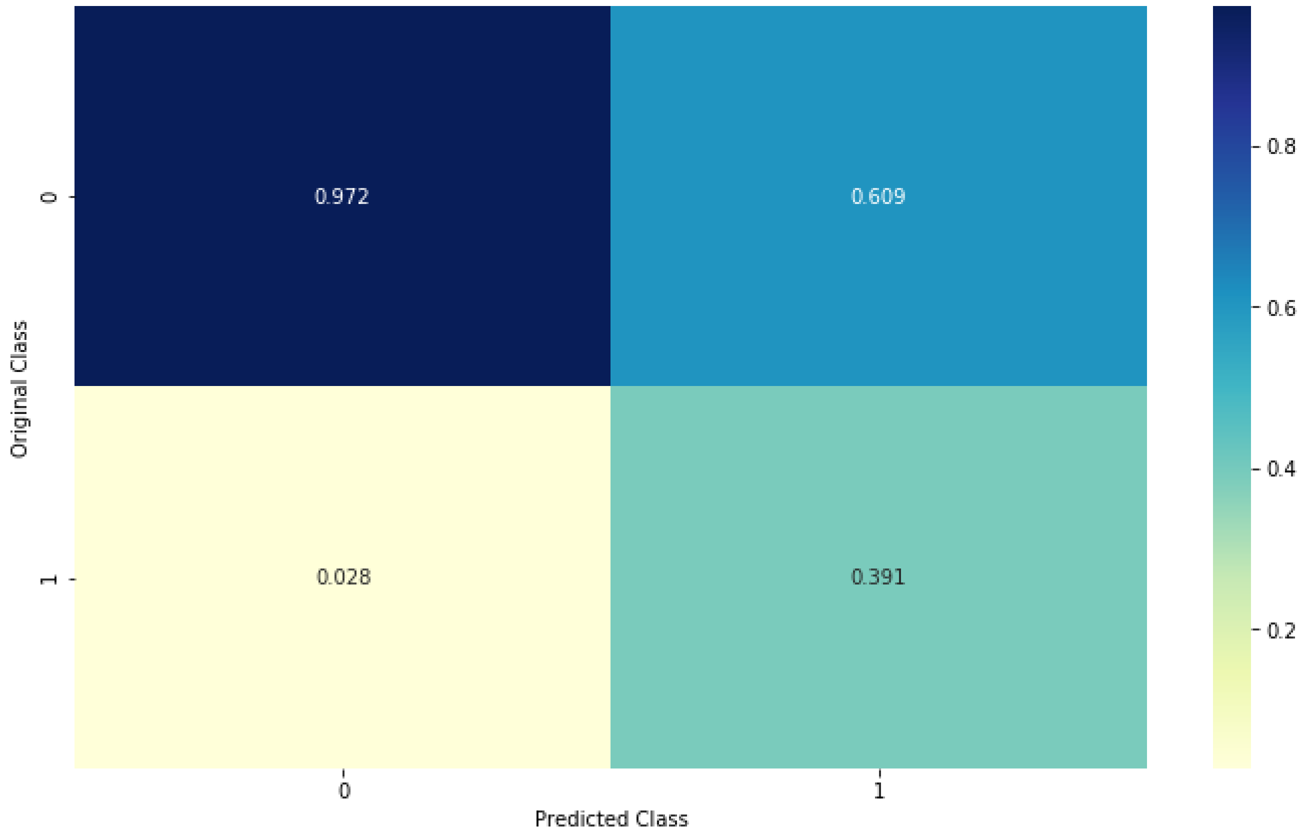
```
#Confusion matrix
C = confusion_matrix(y_test, pred_avg_w2v)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

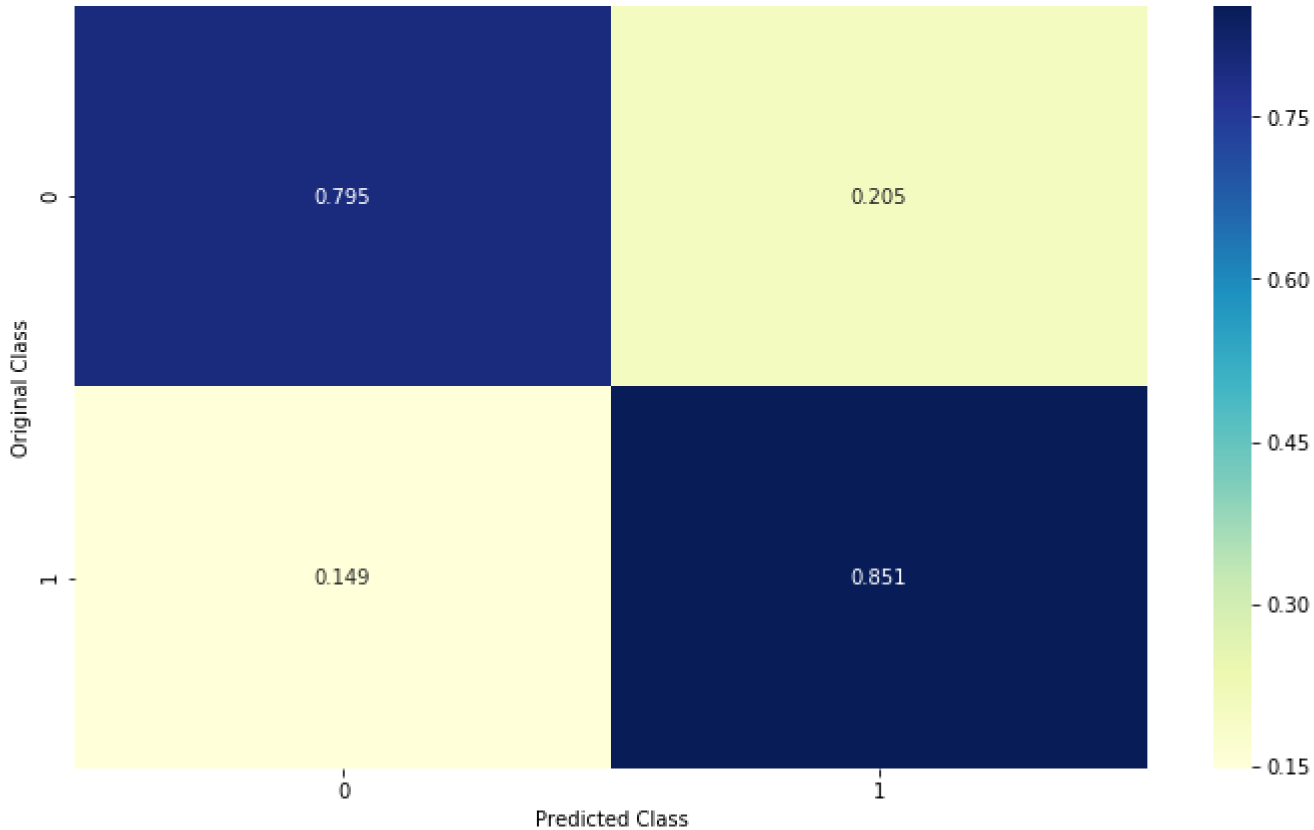
```
-------------------- Confusion matrix --------------------
```




```
-------------------- 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 [01:11<00:00, 1113.33it/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 [00:18<00:00, 1080.47it/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)
```

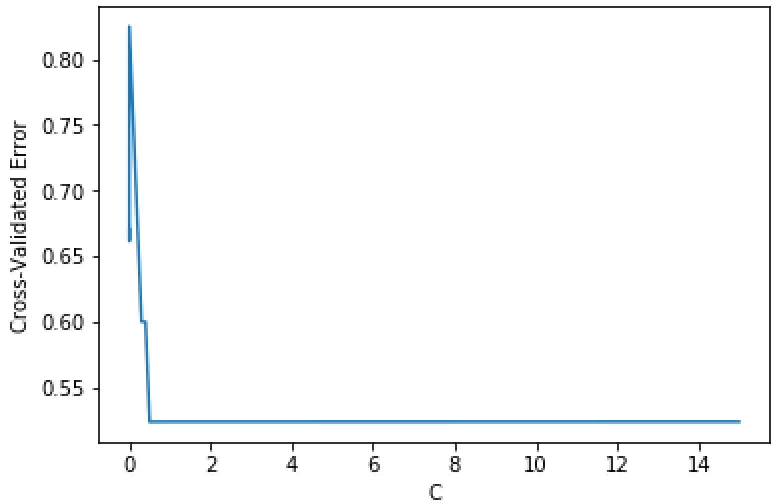
Fitting grid search cv on tfidf-w2vec

```
grid.fit(tfidf_w2v_sent_vectors_train, y_train)

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

```
0.47618209651423954
{'class_weight': 'balanced', 'C': 0.5}
```

```
#Plotting C 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')
```



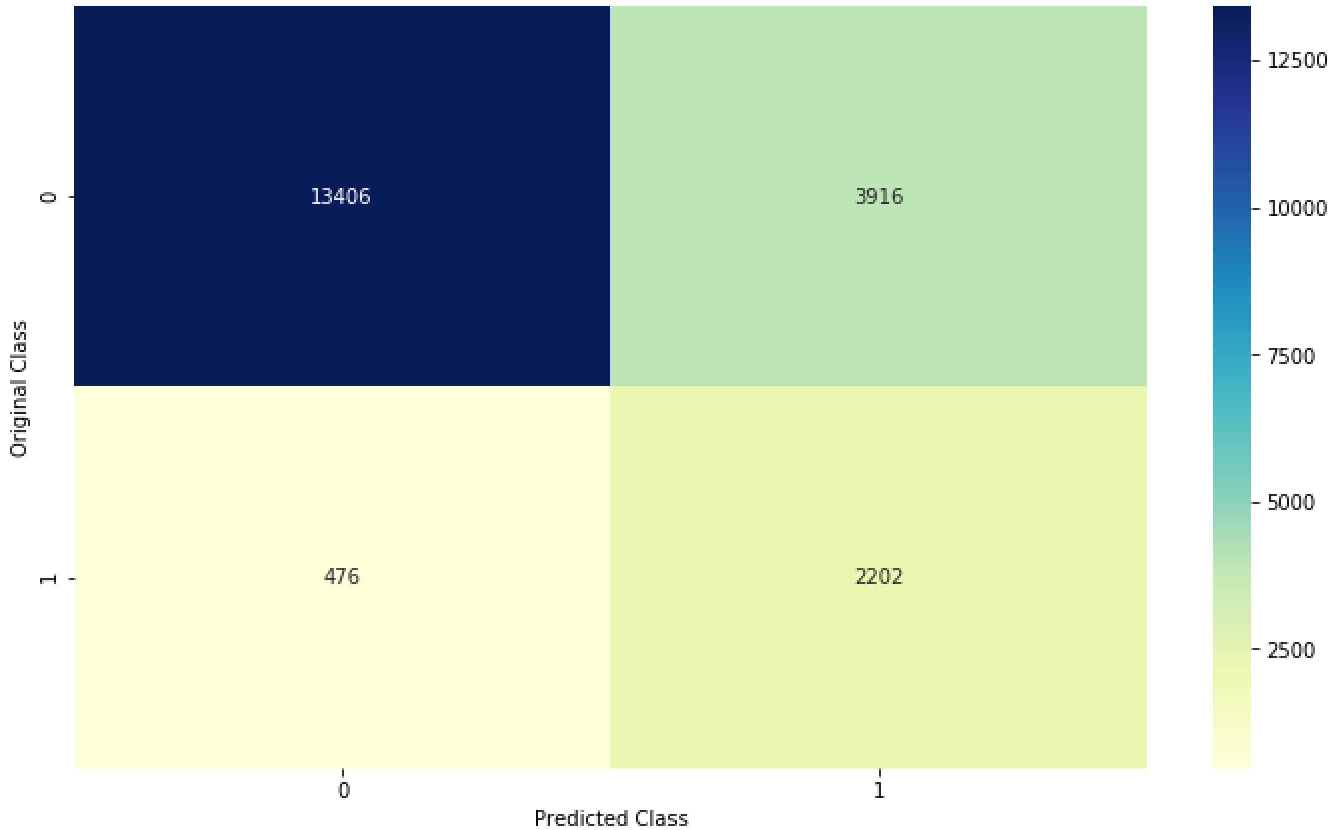
```
#{'class_weight': 'balanced', 'C': 0.5}
LR_optimal=LogisticRegression(penalty='l2',C=0.5,class_weight='balanced')

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

# predict the response
pred_tfidf_w2v_sent_vectors_test = LR_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,LR_optimal.predict(tfid
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 :  0.47774873135475937
The overall f1_score for the Test Data is :  0.5006821282401092
```

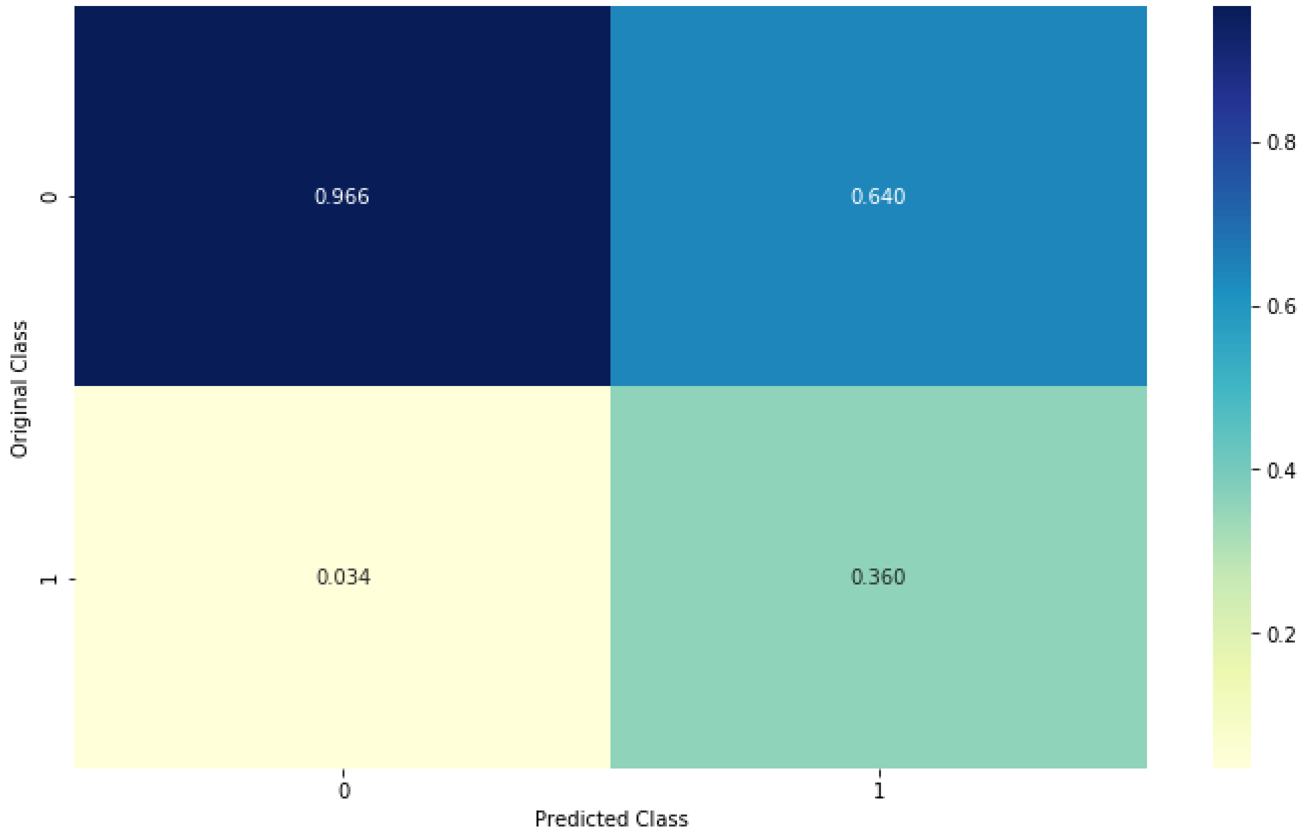
```
#Confusion matrix
C = confusion_matrix(y_test, pred_tfidf_w2v_sent_vectors_test)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

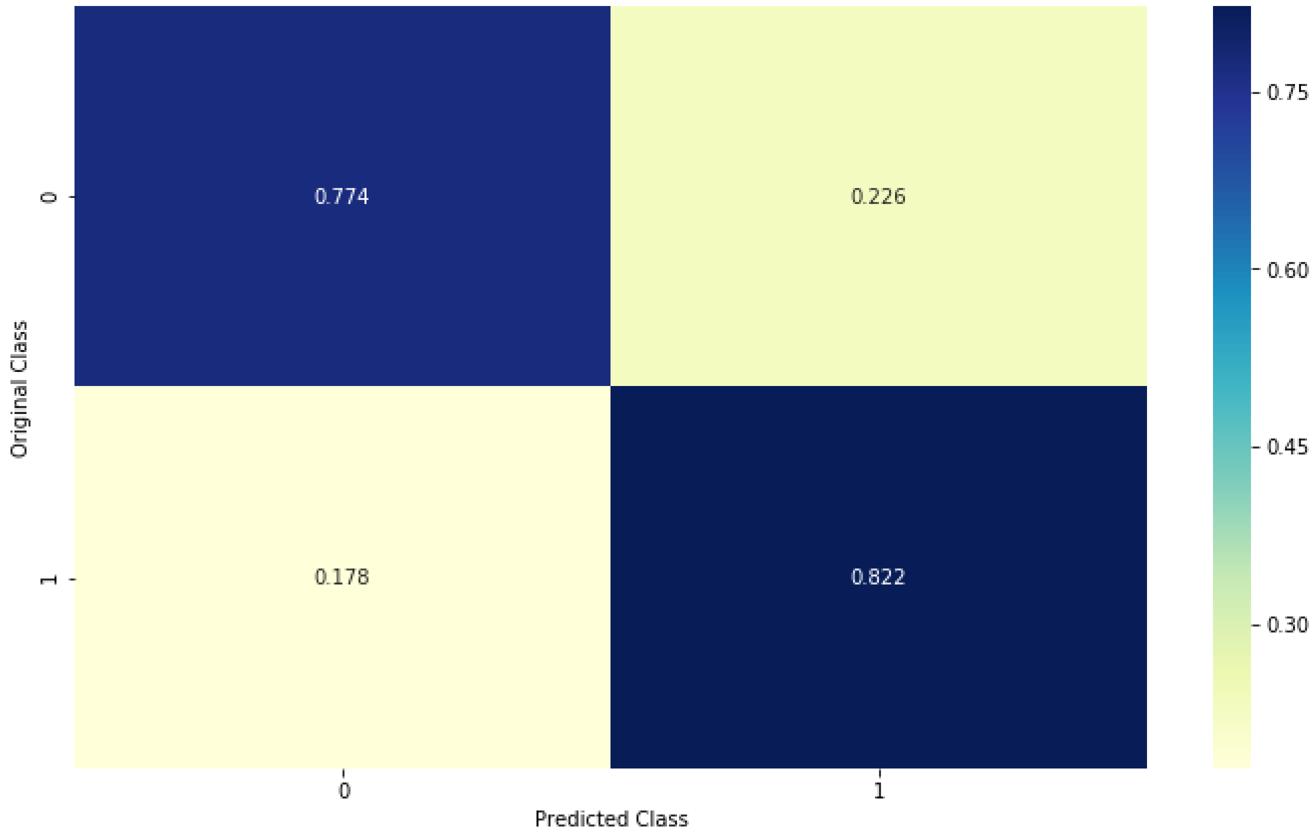
```
-------------------- Confusion matrix --------------------
```




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




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



Reporting f1_score for above featurization with L2 regularizer

```
from prettytable import PrettyTable
```

```
x=PrettyTable()
x.field_names = ["Model","Bow", "Tfidf", "Avg-W2V", "Tfidf-W2V"]
x.add_row(["C",0.01,1e-07,50,0.5])
x.add_row(["Train f1_score",0.80,0.90,0.51,0.47])
x.add_row(["Test f1_score",0.67,0.53,0.53,0.50])

print(x)
```

```
+----------------+------+-------+---------+-----------+
|     Model      | Bow  | Tfidf | Avg-W2V | Tfidf-W2V |
+----------------+------+-------+---------+-----------+
|       C        | 0.01 | 1e-07 |    50   |    0.5    |
| Train f1_score | 0.8  |  0.9  |   0.51  |    0.47   |
| Test f1_score  | 0.67 |  0.53 |   0.53  |    0.5    |
+----------------+------+-------+---------+-----------+
```