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 L1 regularization on different vectors like BOW, Tf-idf, Avg-W2vec & Tf-idf_W2vec.

```python
%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']
```

```
#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.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
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
```

Applying L1 regularization

Gridsearch CV using L2

```
gamma_range = [0.000001,0.00001,0.0001,0.001,0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,5,10,15,20,30,
T= TimeSeriesSplit(n_splits=5)
weight=[None,'balanced']

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

# instantiate and fit the grid
grid = GridSearchCV(LogisticRegression(penalty='l1'), param_grid, cv=T, scoring='f1', return_train_scor
```

```
{'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, 5, 10, 15, 20, 30, 40, 50], 'cl
ass_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)
```

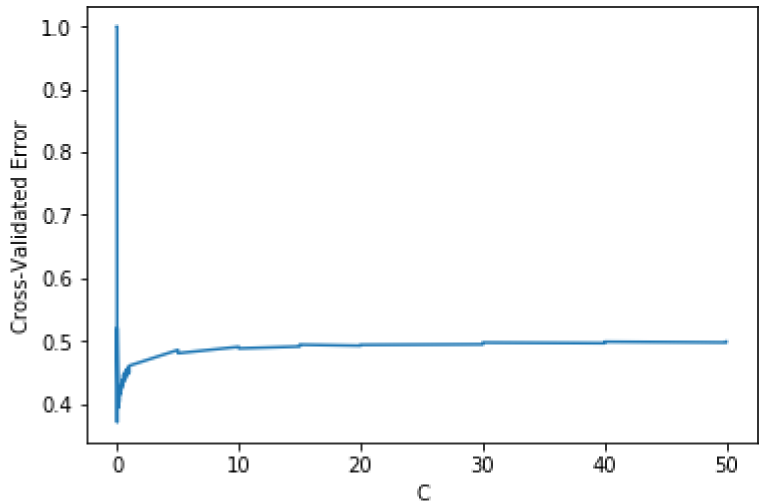
Fitting Grid search cv on BOW

```
grid.fit(Bow_x_train, y_train)

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

```
0.6296566961732983
{'C': 0.01, 'class_weight': 'balanced'}
```

```
#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')
```



```python

#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.01,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 :  0.7461583187972424
The overall f1_score for the Test Data is :  0.6732643118148599
```

Pertubation test

```python
# 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
```

```python
Noise_Bow_x_train.shape
```

```
(80000, 33433)
```

```python

#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal_noise=LogisticRegression(penalty='l1',C=0.01,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 :  0.7461893092993311
The overall f1_score for the Test Data is :  0.6737707413609377
```

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

```
(33433,)
```

```python
#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	0.000000	0.000000	NaN
aaaaaaaaagghh	0.000000	0.000000	NaN
aaaaah	0.000000	0.000000	NaN
aaaaahhhhhhhhhhhhhhh	0.000000	0.000000	NaN
aaaah	0.000000	0.000000	NaN
aaah	0.000000	0.000000	NaN
aachen	0.000000	0.000000	NaN
aad	0.000000	0.000000	NaN
aadp	0.000000	0.000000	NaN
aafco	0.000000	0.000000	NaN
aagh	-0.000872	-0.000104	-88.105923
aah	0.000000	0.000000	NaN
aahh	0.000000	0.000000	NaN
aand	0.000000	0.000000	NaN
aardvark	-0.006501	-0.001462	-77.508067
aarrgh	0.009030	0.009029	-0.002622
ab	0.000000	0.000000	NaN
aback	-0.000381	-0.000386	1.460521
abandon	0.000000	0.000000	NaN
abaolut	0.000000	0.000000	NaN
abattoir	0.000000	0.000000	NaN
abba	0.000000	0.000000	NaN
abbey	0.000000	0.000000	NaN
abbi	0.000000	0.000000	NaN

	0	1	2
abbott	0.000000	0.000000	NaN
abbrevi	0.000000	0.000000	NaN
abc	0.000000	0.000000	NaN
abcstor	0.000000	0.000000	NaN
abd	0.000000	0.000000	NaN
abdomen	0.000000	0.000000	NaN
...	...	...	...
zot	-0.011494	-0.011496	0.014257
zotz	0.000000	0.000000	NaN
zour	0.000000	0.000000	NaN
zout	0.000000	0.000000	NaN
zowi	0.000000	0.000000	NaN
zreport	0.000000	0.000000	NaN
zsweet	0.000000	0.000000	NaN
zuc	0.000000	0.000000	NaN
zucchini	0.000000	0.000000	NaN
zuccini	0.000000	0.000000	NaN
zuccnini	0.000000	0.000000	NaN
zuchinni	0.000000	0.000000	NaN
zuke	0.000000	0.000000	NaN
zulu	0.000000	0.000000	NaN
zum	0.000000	0.000000	NaN
zummi	0.000000	0.000000	NaN
zune	0.000000	0.000000	NaN
zupreem	0.000000	0.000000	NaN
zurich	0.000000	0.000000	NaN
zwar	0.000000	0.000000	NaN
zwieback	0.000646	0.000641	-0.704890
zwiebeck	0.000000	0.000000	NaN

	0	1	2
zydeco	0.000000	0.000000	NaN
zzzzzs	0.000000	0.000000	NaN
zzzzzz	0.000000	0.000000	NaN
zzzzzzz	0.000759	0.000756	-0.283037
zzzzzzzz	0.000000	0.000000	NaN
zzzzzzzzzz	0.000000	0.000000	NaN
zzzzzzzzzzz	0.000000	0.000000	NaN
çay	0.000000	0.000000	NaN

33433 rows × 3 columns

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

```
(253, 3)
```

253 features out of 33433 shows percentage change > 30 post pertubation test i.e 0.75%

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)
```

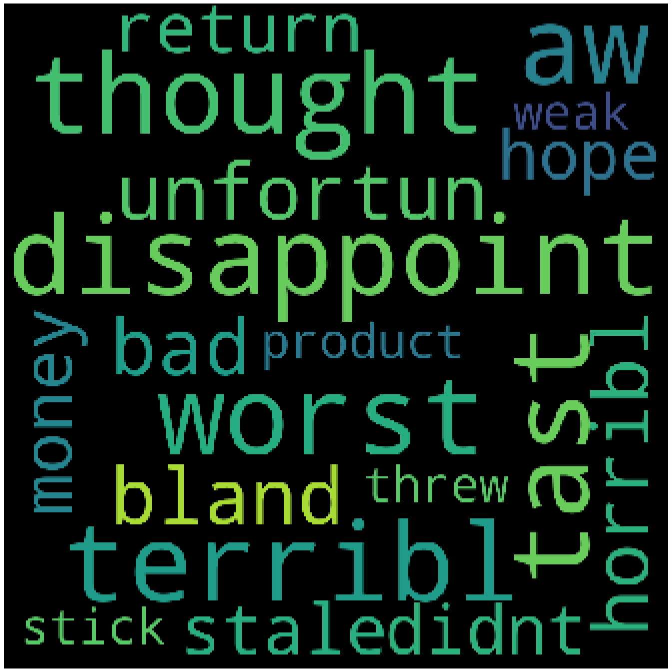
```
great,best,love,perfect,delici,excel,good,favorit,amaz,nice,wonder,addict,tasti,find,awesom,thank,keep,happi,smooth,year,
```

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

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

```
disappoint,worst,terribl,thought,tast,aw,unfortun,horribl,bad,bland,would,return,stale,money,didnt,hope,product,stick,thre
w,weak,
```

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

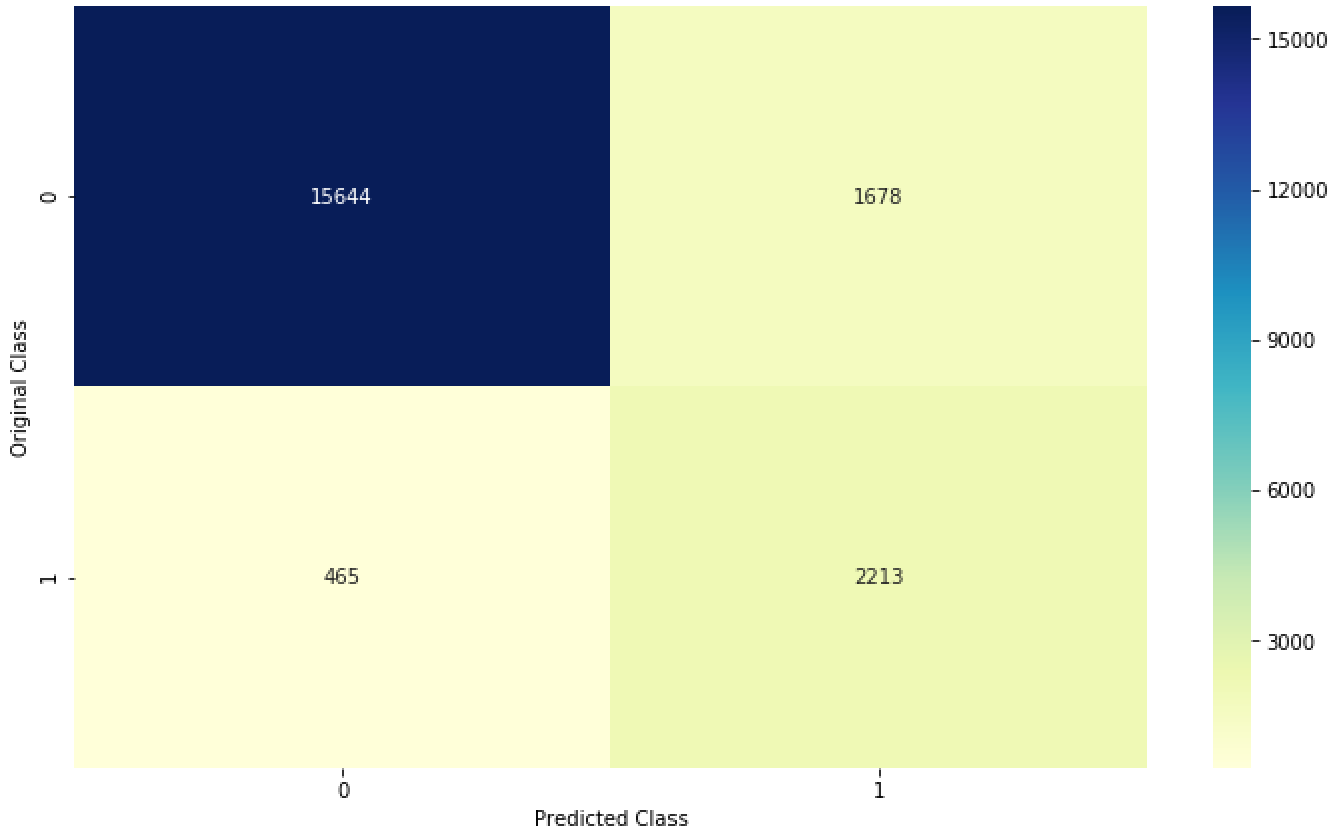



```
************ Top 20 Positive words ******************
```

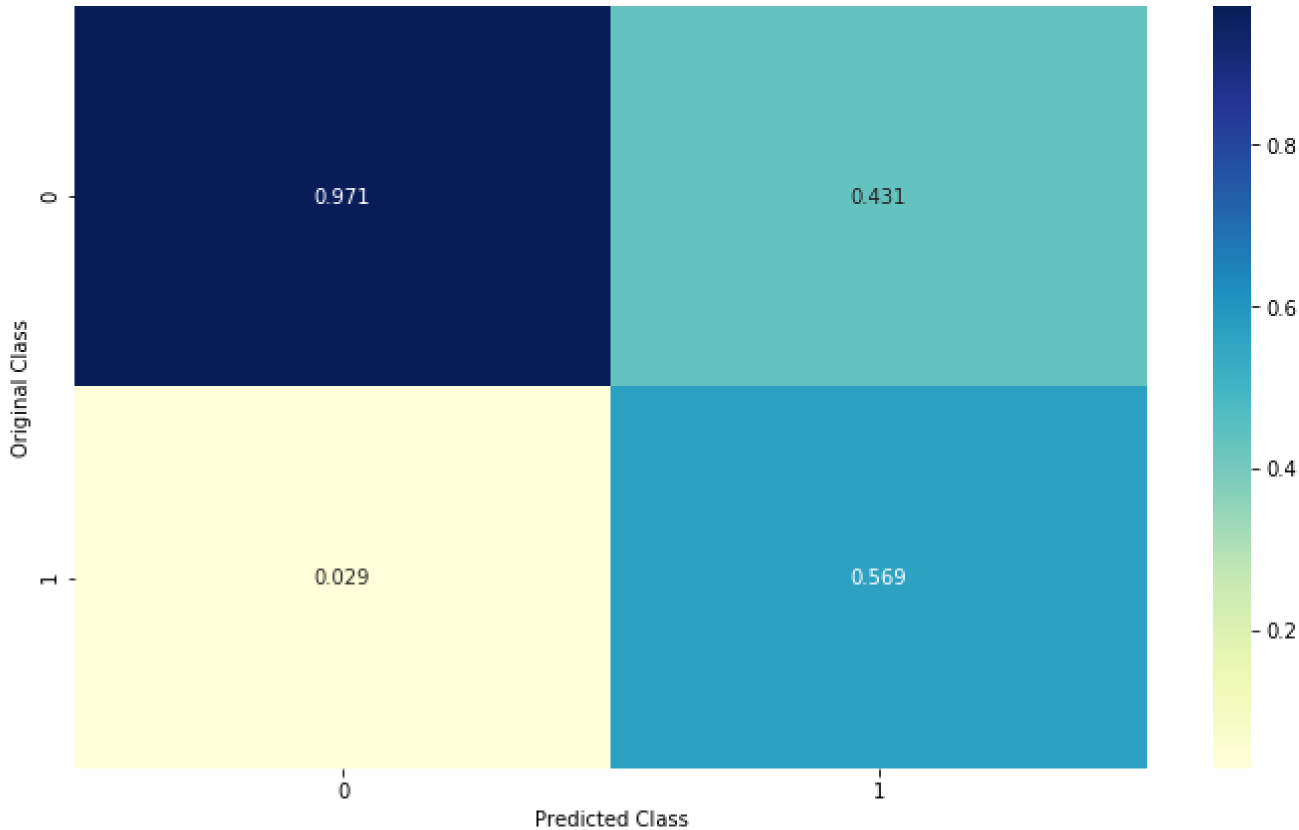

```
#Confusion matrix
C = confusion_matrix(y_test, pred_bow)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

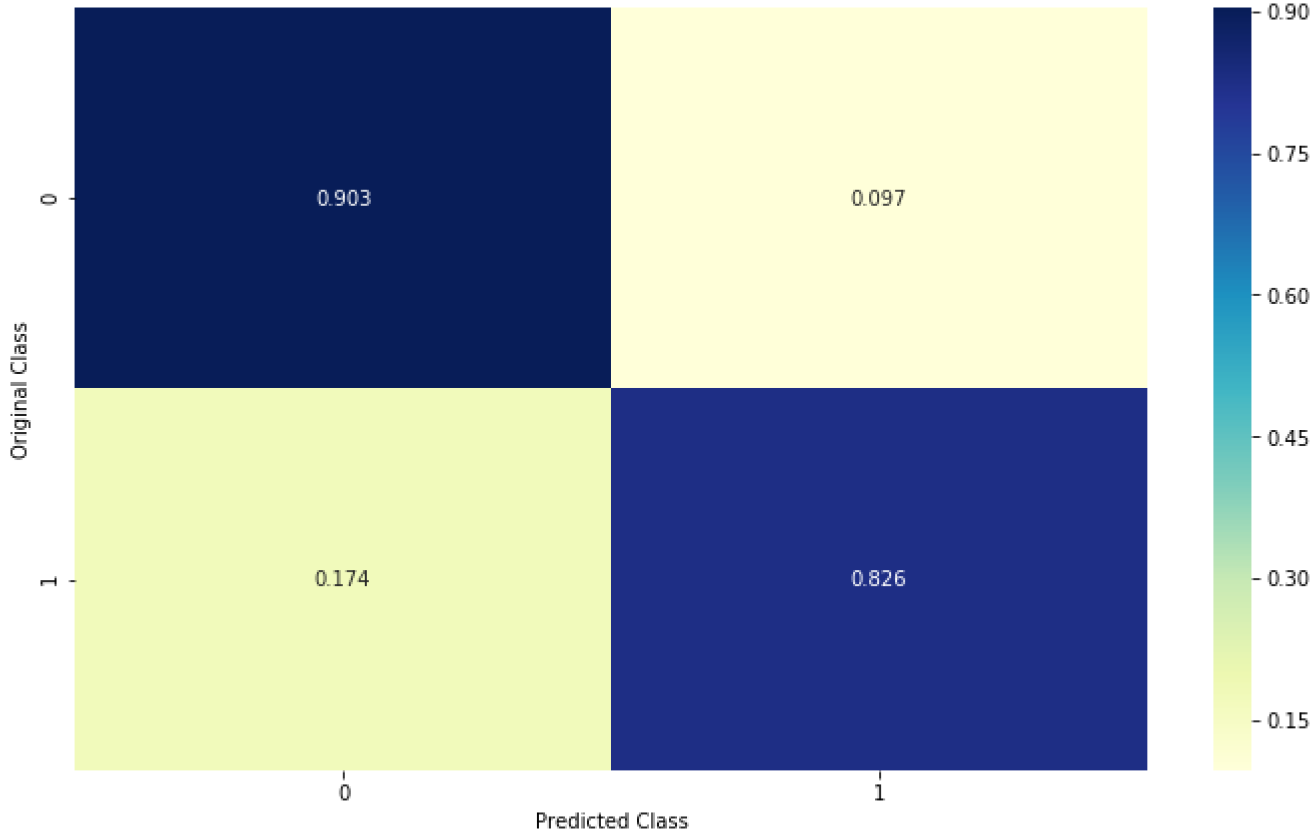
```
-------------------- Confusion matrix --------------------
```




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




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



What if we decrease the value of C? How does it effect Sparsity and Error?

C=1/ λ

```
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
5293
```

```python
#Decreasing the value of C
#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.001,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)
```

```python
#Checking  non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
245
```

```python
#Decreasing the value of C
#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',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)
```

```
#Checking  non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
1
```

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)
```

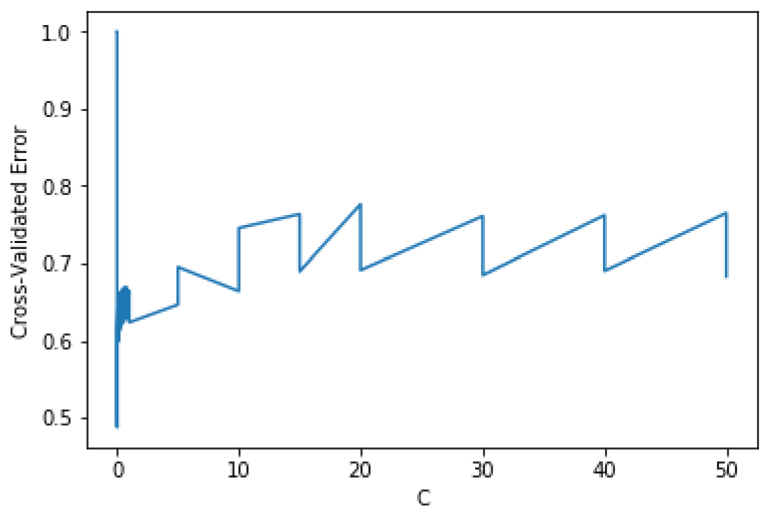
Fitting Grid Search on Tf-Idf

```
grid.fit(Tfidf_x_train, y_train)

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

```
0.5125440355342478
{'C': 0.01, 'class_weight': 'balanced'}
```

```python
#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')
```



```python

#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.01,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 :  0.954717356687898
The overall f1_score for the Test Data is :  0.679509632224168
```

Pertubation test

```python
# 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
from sklearn.metrics import f1_score
#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal_noise=LogisticRegression(penalty='l1',C=0.01,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.predict(Nois
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 :  0.9545273631840796
The overall f1_score for the Test Data is :  0.6779925795743019
```

```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	0.0	0.000000	NaN
aaa condit	0.0	0.000000	NaN
aaa perfect	0.0	0.000000	NaN
aaaaaaaaagghh	0.0	0.000000	NaN
aaaaah	0.0	0.000000	NaN
aaaaah awak	0.0	0.000000	NaN
aaaaah satisfi	0.0	0.000000	NaN
aaaaahhhhhhhhhhhhhhh	0.0	0.000000	NaN
aaaaahhhhhhhhhhhhhhh angel	0.0	0.000000	NaN
aaaah	0.0	0.000000	NaN
aaaah snob	0.0	0.000000	NaN
aaah	0.0	0.000000	NaN
aaah inhal	0.0	0.000000	NaN
aaah miss	0.0	0.000000	NaN
aaah sip	0.0	0.000000	NaN
aachen	0.0	0.000000	NaN
aachen munich	0.0	0.000000	NaN
aad	0.0	0.000000	NaN
aad sausag	0.0	0.000000	NaN
aadp	0.0	0.000000	NaN
aafco	0.0	0.000000	NaN
aafco also	0.0	0.000000	NaN
aafco certifi	0.0	0.000168	inf
aafco countri	0.0	0.000000	NaN

	0	1	2
aafco definit	0.0	0.000000	NaN
aafco dog	0.0	0.000000	NaN
aafco guidelin	0.0	0.000000	NaN
aafco requir	0.0	0.000000	NaN
aagh	0.0	0.000000	NaN
aagh yelp	0.0	0.000000	NaN
...	...	...	...
zum heal	0.0	0.000000	NaN
zummi	0.0	0.000000	NaN
zummi love	0.0	0.000000	NaN
zummi tast	0.0	0.000000	NaN
zummi tri	0.0	0.000000	NaN
zune	0.0	0.000000	NaN
zune video	0.0	0.000000	NaN
zupreem	0.0	0.000000	NaN
zupreem ferret	0.0	0.000000	NaN
zurich	0.0	0.000000	NaN
zurich schnatzlet	0.0	0.000000	NaN
zwar	0.0	0.000000	NaN
zwar billig	0.0	0.000000	NaN
zwieback	0.0	0.000000	NaN
zwieback toast	0.0	0.000000	NaN
zwiebeck	0.0	0.000000	NaN
zwiebeck toast	0.0	0.000000	NaN
zydeco	0.0	0.000000	NaN
zydeco saturday	0.0	0.000000	NaN
zzzzzs	0.0	0.000000	NaN
zzzzzs larg	0.0	0.000000	NaN
zzzzzz	0.0	0.000000	NaN

	0	1	2
zzzzzz say	0.0	0.000000	NaN
zzzzzzz	0.0	0.000000	NaN
zzzzzzz high	0.0	0.000000	NaN
zzzzzzzz	0.0	0.000000	NaN
zzzzzzzzzz	0.0	0.000000	NaN
zzzzzzzzzz final	0.0	0.000000	NaN
zzzzzzzzzzz	0.0	0.000000	NaN
çay	0.0	0.000000	NaN

1013943 rows × 3 columns

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

```
(12588, 3)
```

12588 features out of 1013943 shows percentage change > 30 post pertubation test i.e 1.24%

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)
```

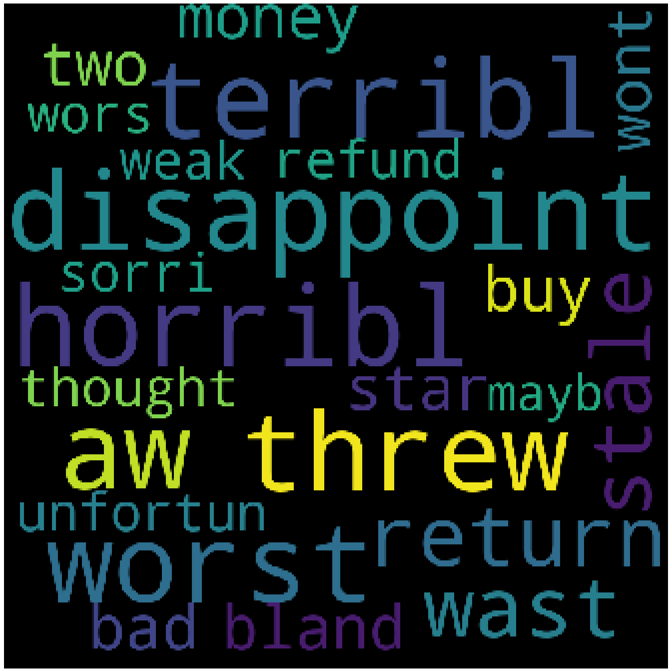
```
great,best,love,delici,good,perfect,excel,high recommend,favorit,wonder,nice,find,tasti,amaz,keep,alway,enjoy,addict,thank,
easi,
```

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

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

```
disappoint,worst,terribl,aw,threw,horribl,return,stale,wast money,two star,wont buy,bad,bland,unfortun,thought,refund,sorr
i,wors,weak,mayb,
```

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

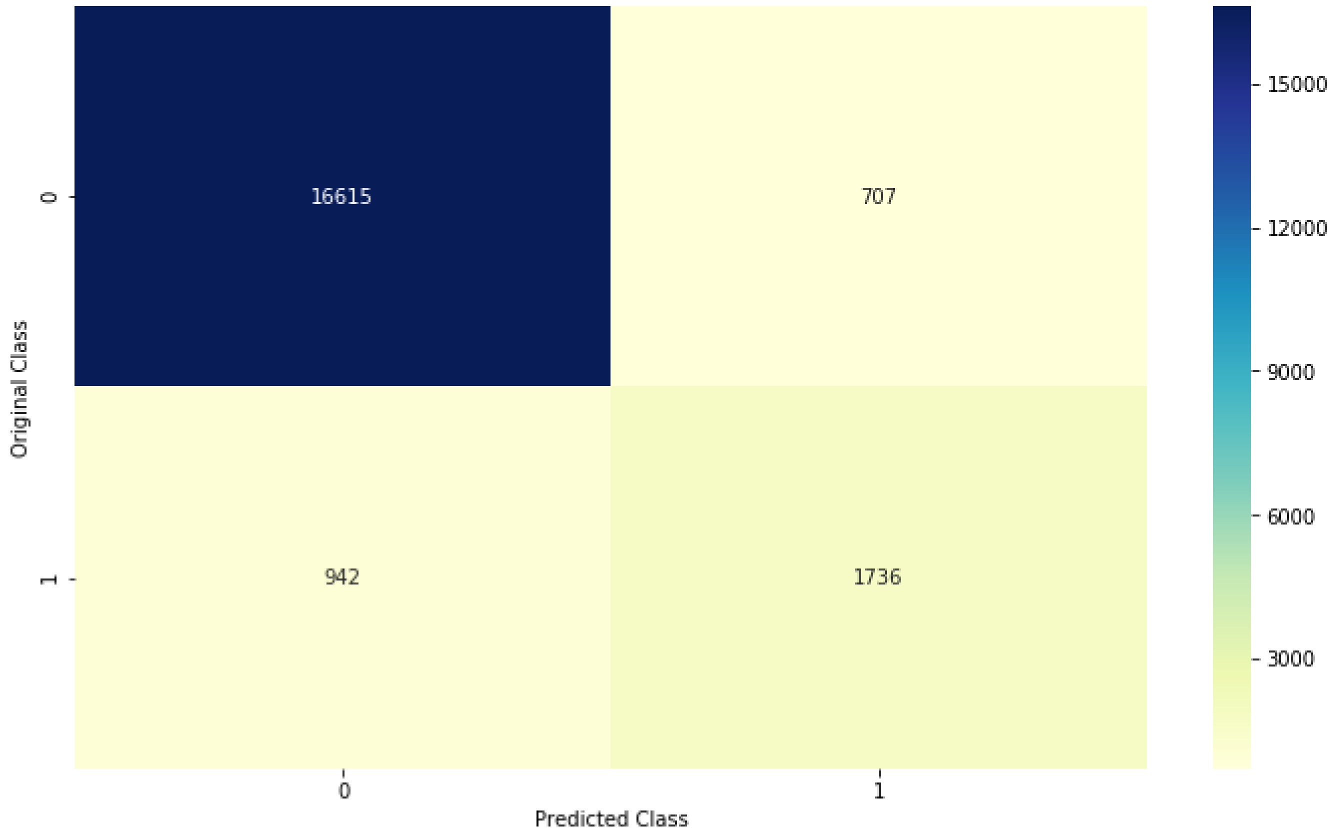



```
************ Top 20 Positive words ******************
```

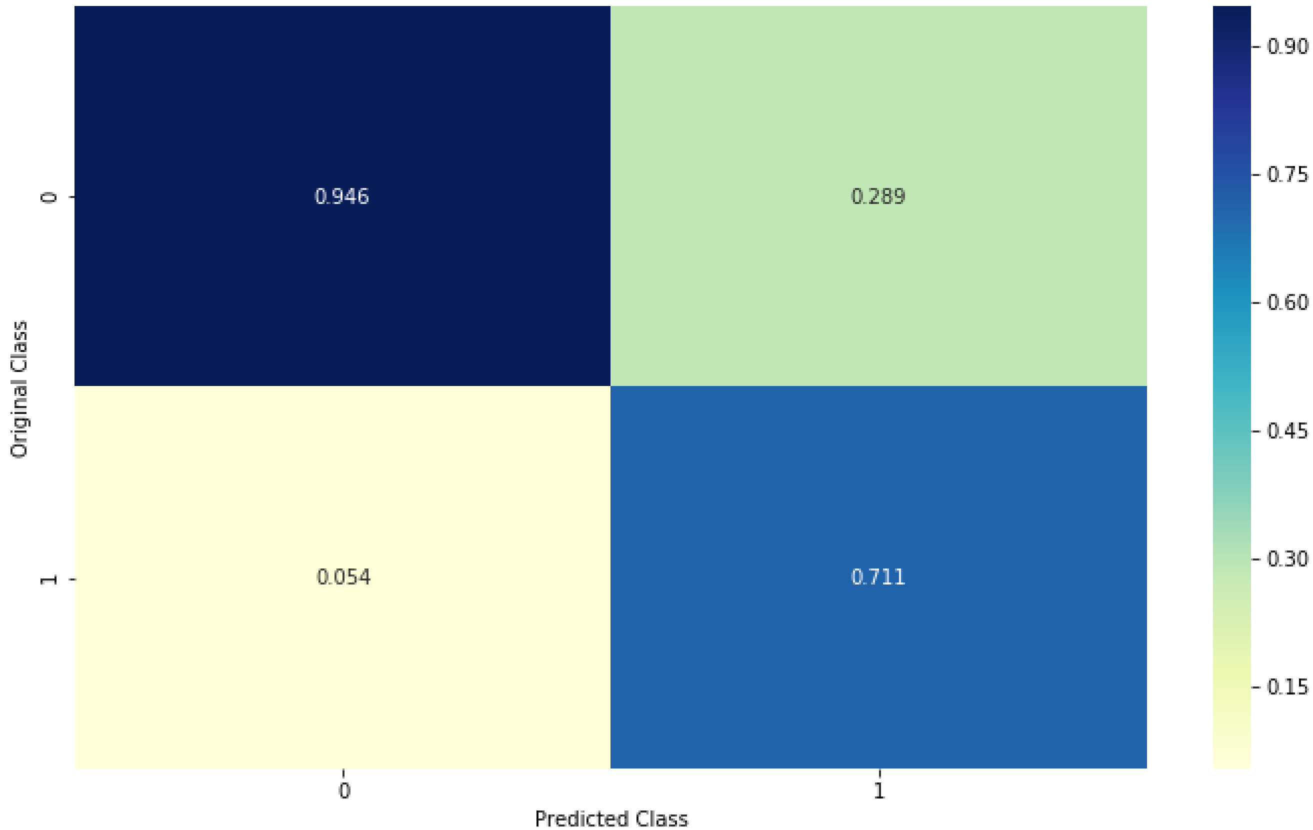

```
#Confusion matrix
C = confusion_matrix(y_test, pred_tfidf)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

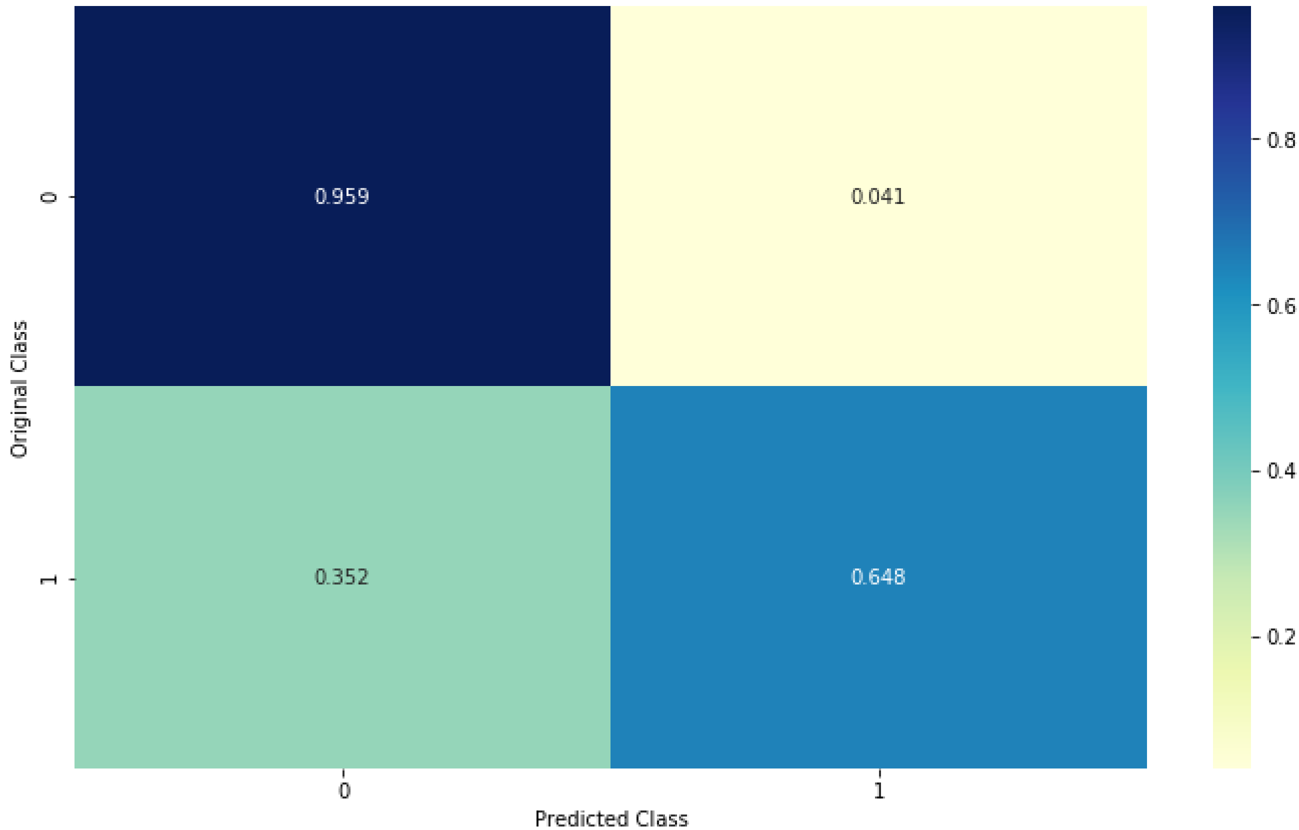
```
-------------------- Confusion matrix --------------------
```




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




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



What if we decrease the value of C? How does it effect Sparsity and Error?

C=1/ λ

```
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
31591
```

```python
#Decreasing value of c
#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.001,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)
```

```python
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
577
```

```python
#Decreasing value of c
#{'C': 0.01, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.0001,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)
```

```
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
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)
```

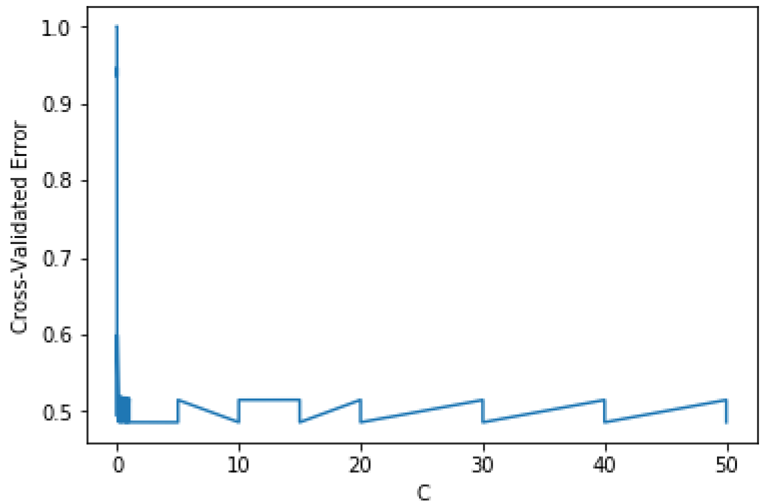
Fitting grid search on Avg-W2V

```
grid.fit(sent_vectors_train_avgw2v, y_train)

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

```
0.5141808414956328
{'C': 5, 'class_weight': 'balanced'}
```

```
#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')
```



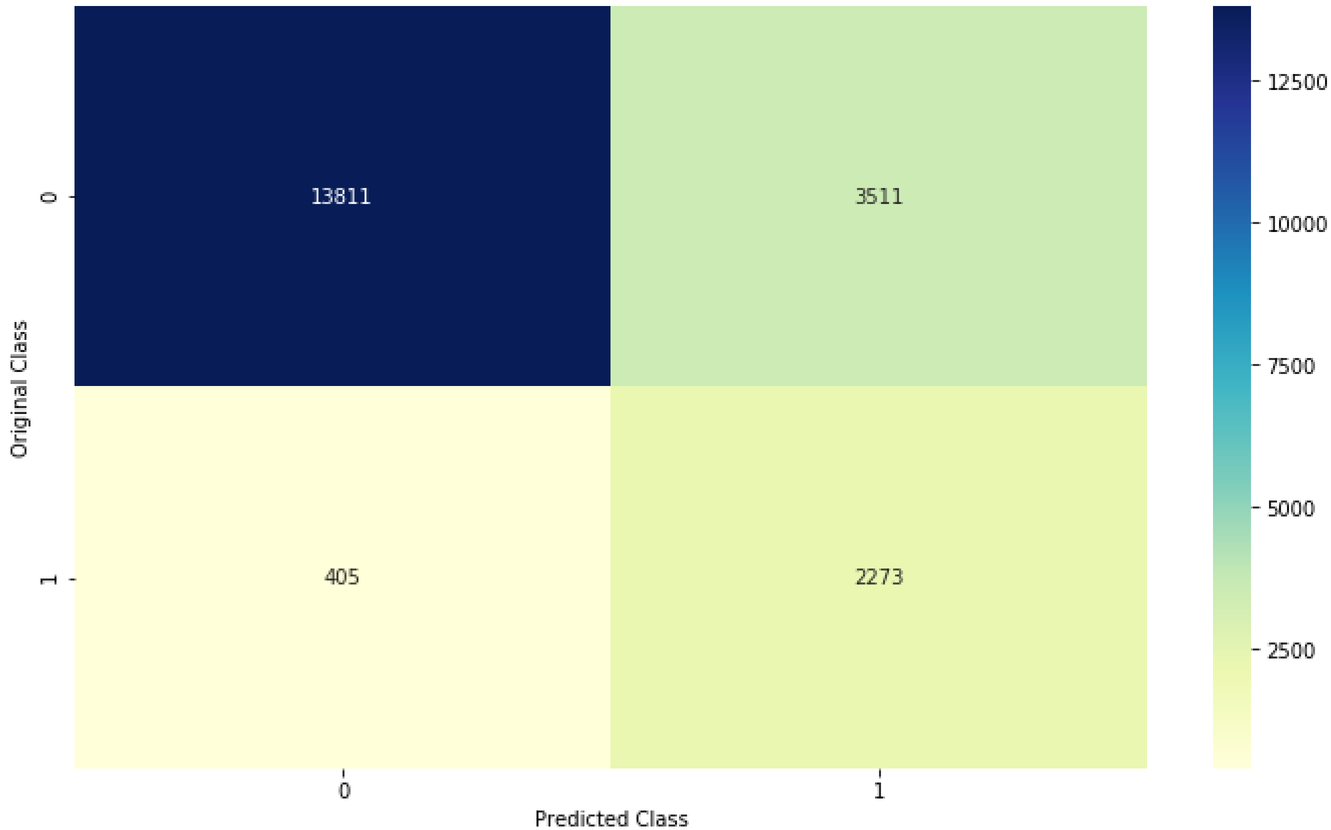
```python

#{'C': 5, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=5,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 :  0.5156973751930005
The overall f1_score for the Test Data is :  0.5372252422595132
```

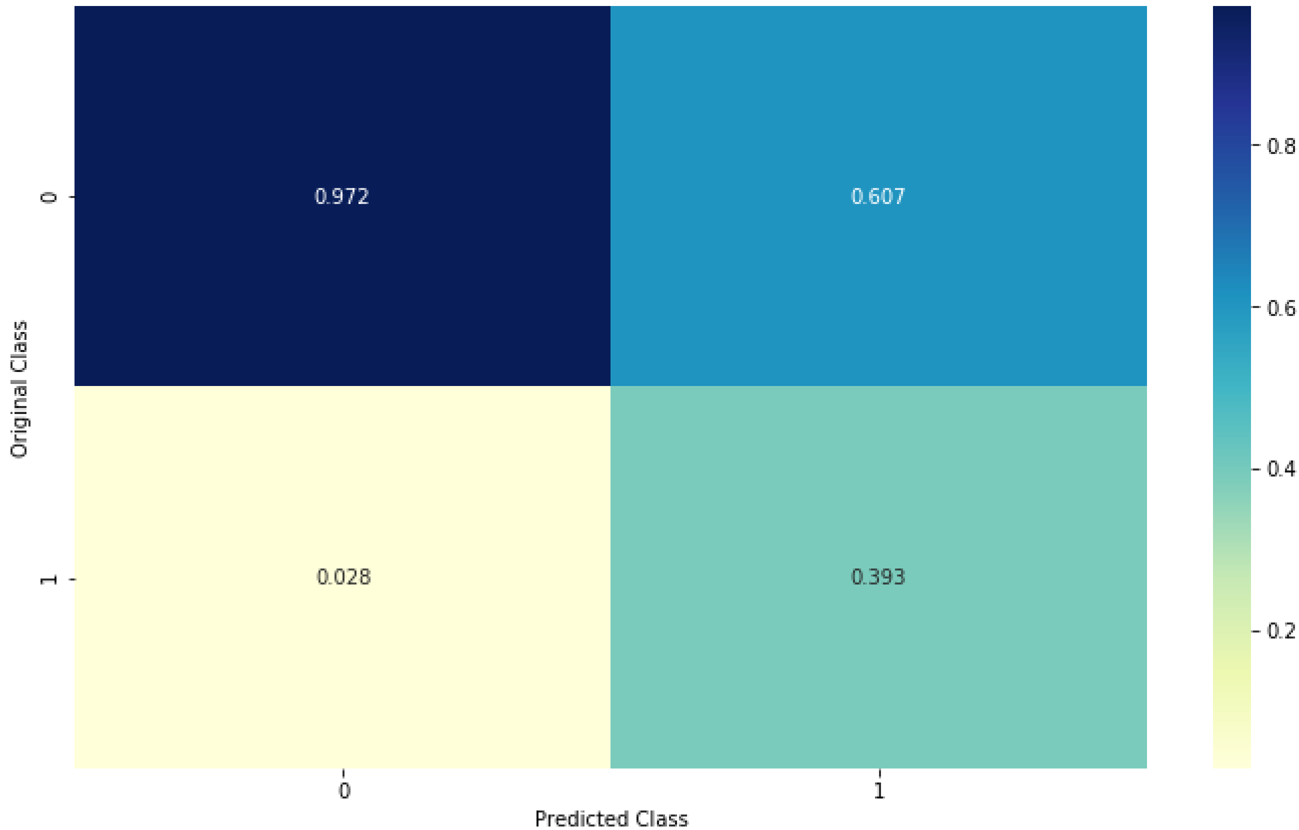
```python
#Confusion matrix
C = confusion_matrix(y_test, pred_avg_w2v)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

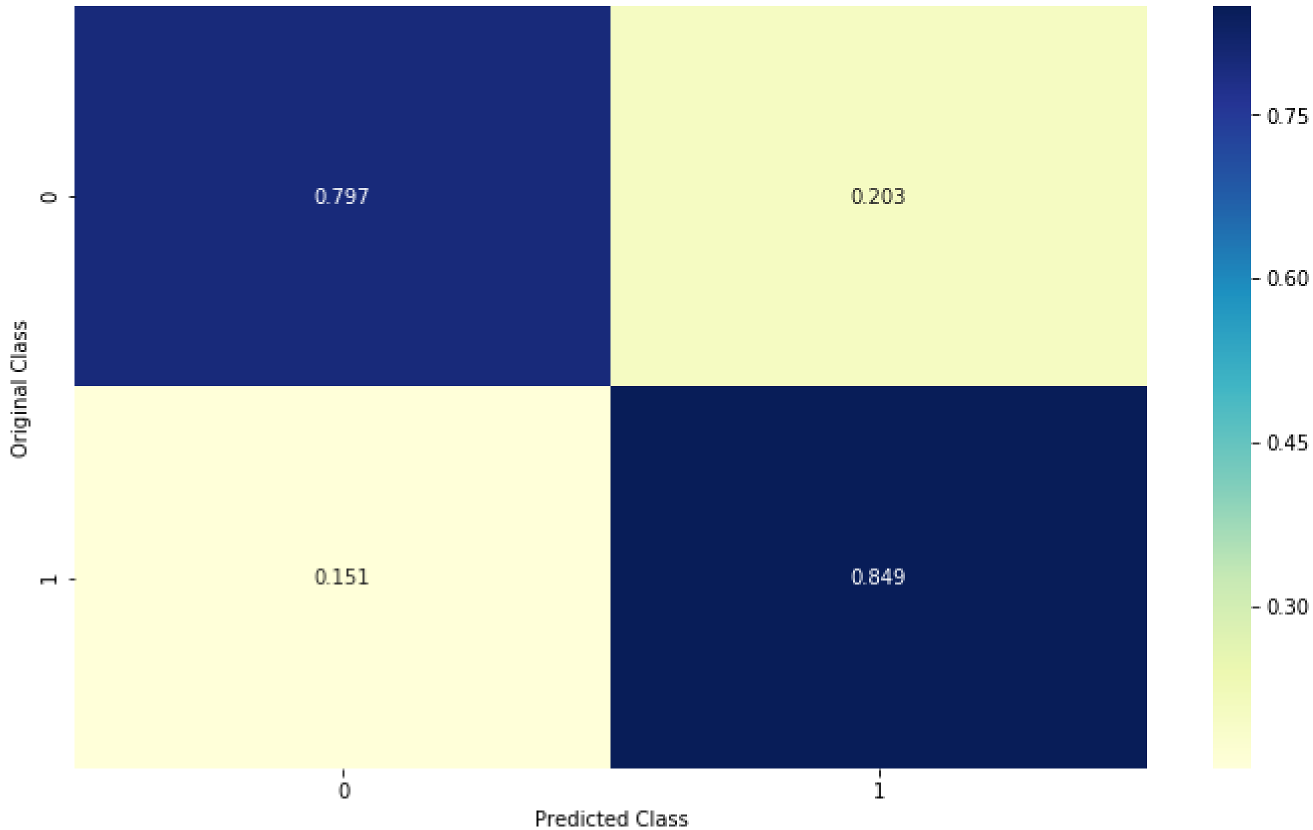
```
-------------------- Confusion matrix --------------------
```




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




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



What if we decrease the value of C? How does it effect Sparsity and Error?

C=1/ λ

```
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
50
```

```
#Decrese the value of C
#{'C': 5, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.1,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)
```

```
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
50
```

```
#Decrese the value of C
#{'C': 5, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.001,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)
```

```
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
33
```

```
#Decrese the value of C
#{'C': 5, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.0001,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)
```

```
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
3
```

Tf-idf W2V

```
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, 1112.80it/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, 1098.16it/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)
```

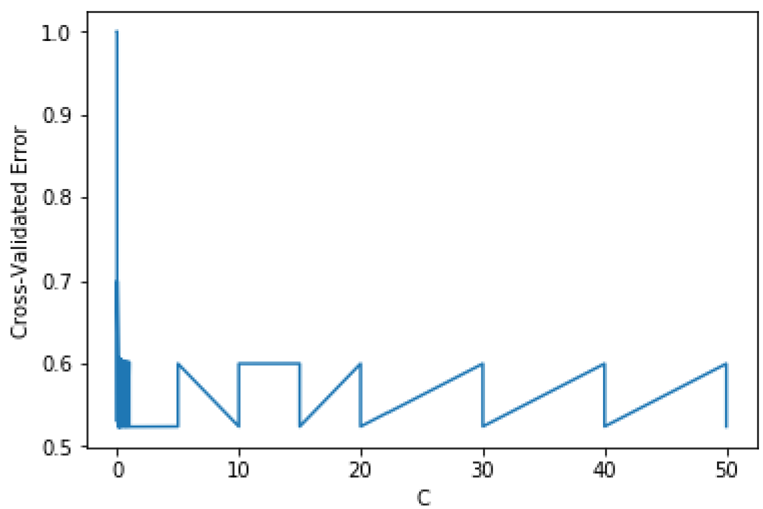
Fitting grid search cv on tfidf-w2vec

```
grid.fit(tfidf_w2v_sent_vectors_train, y_train)

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

```
0.47690283662194616
{'C': 0.1, 'class_weight': 'balanced'}
```

```
#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')
```



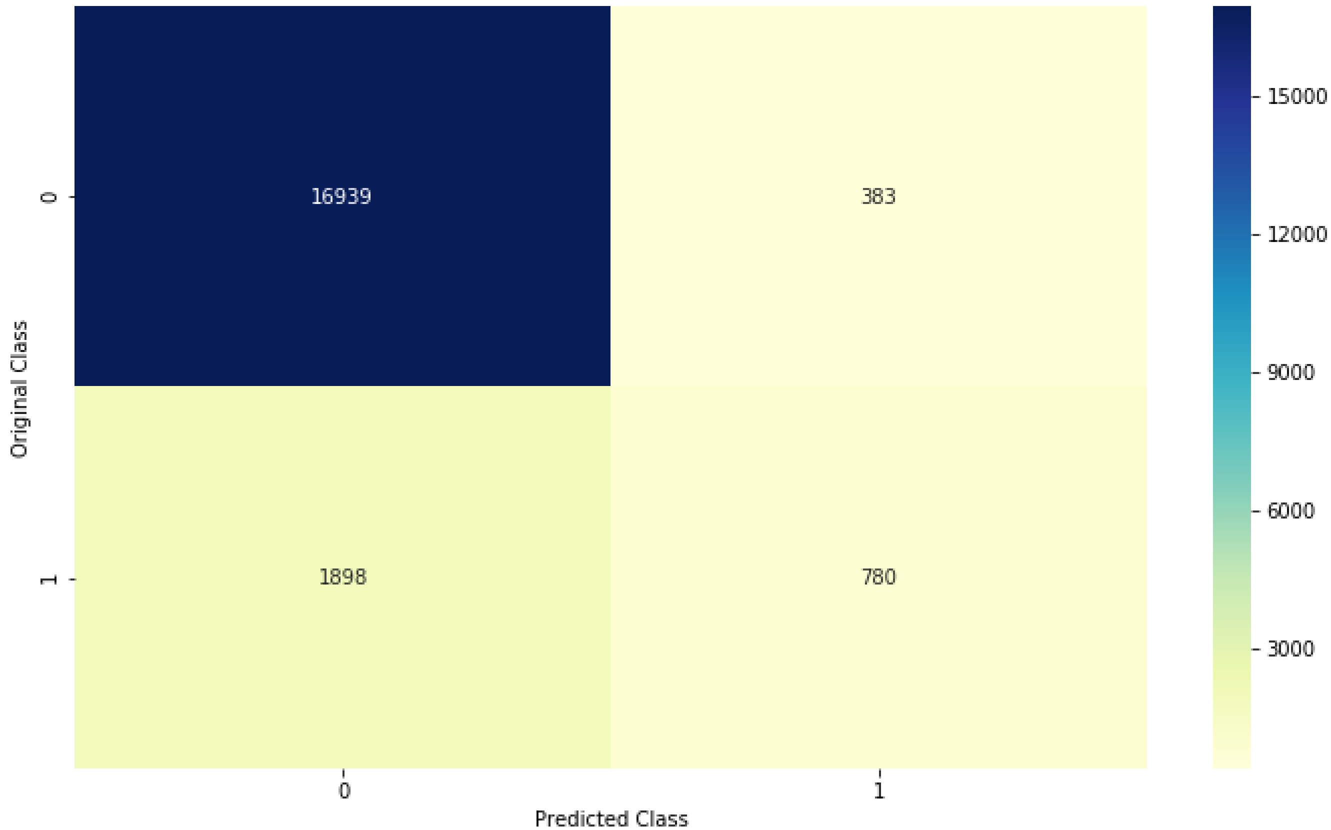
```
#{'C': 0.1, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.1,class_weight=None)

# 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 :  0.41298833079654995
The overall f1_score for the Test Data is :  0.40614423327258525
```

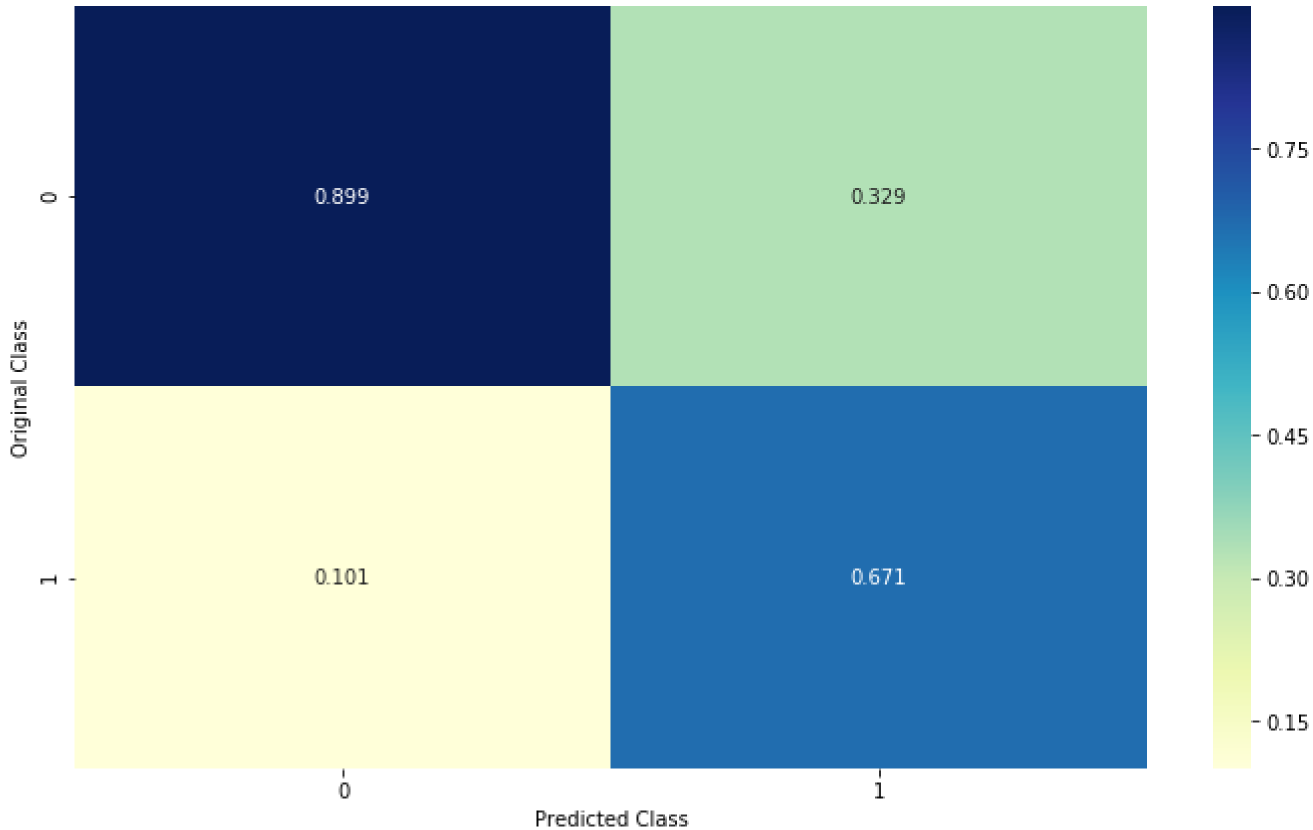
```
#Confusion matrix
C = confusion_matrix(y_test, pred_tfidf_w2v_sent_vectors_test)
A =(((C.T)/(C.sum(axis=1))).T)
B =(C/C.sum(axis=0))
labels = [0,1]
```

```
 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()
```

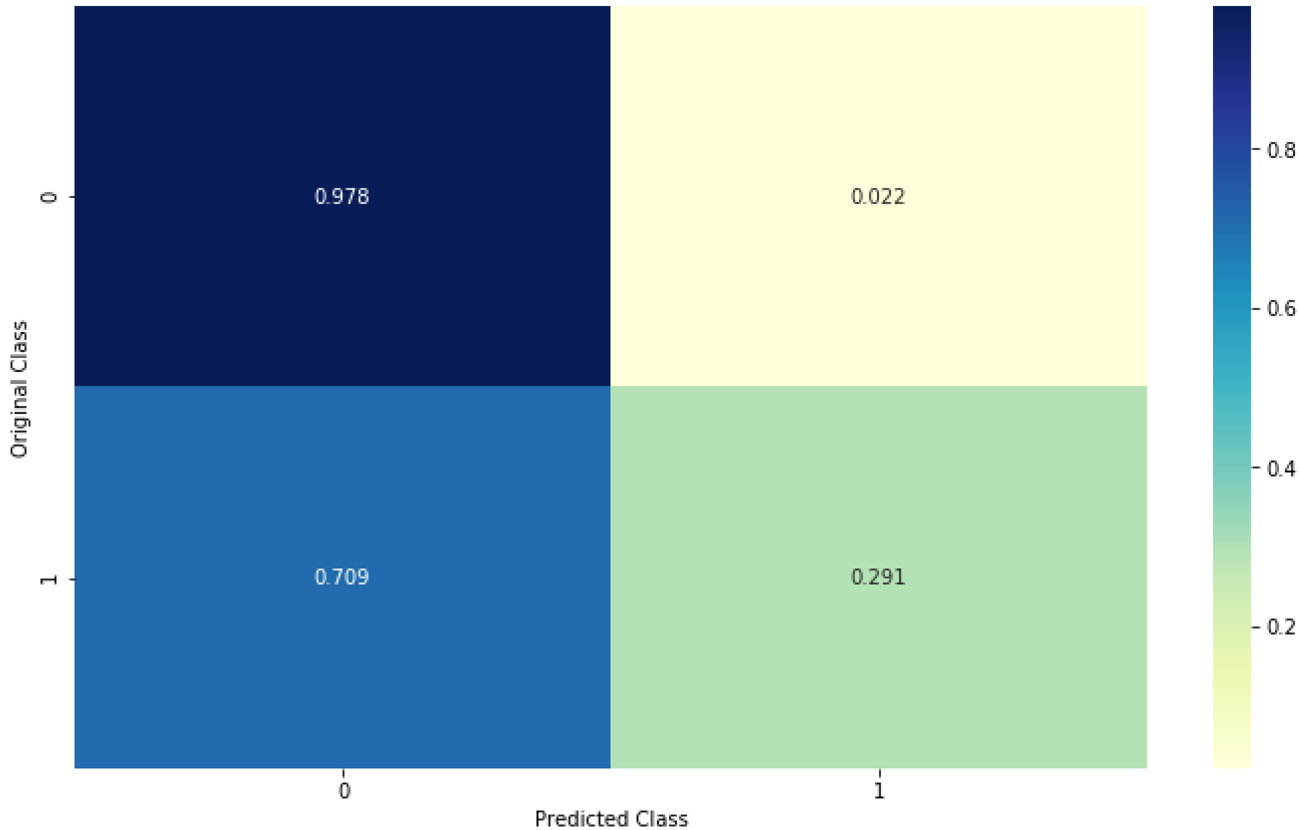
```
-------------------- Confusion matrix --------------------
```




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




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



What if we decrease the value of C? How does it effect Sparsity and Error?

C=1/ λ

```
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
50
```

```
#Decreasing value of C
#{'C': 0.1, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.001,class_weight=None)

# 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)

```

```
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
19
```

```
#Decreasing value of C
#{'C': 0.1, 'class_weight': 'balanced'}
LR_optimal=LogisticRegression(penalty='l1',C=0.0001,class_weight=None)

# 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)

```

```
#Checking current non-zero elements
a=np.array(LR_optimal.coef_)
print(np.count_nonzero(a))
```

```
0
```

Reporting f1_score for above featurization with L1 regularizer

```
from prettytable import PrettyTable
```

```
x=PrettyTable()
x.field_names = ["Model","Bow", "Tfidf", "Avg-W2V", "Tfidf-W2V"]
x.add_row(["C",0.01,0.01,5,0.1])
x.add_row(["Train f1_score",0.74,0.95,0.51,0.41])
x.add_row(["Test f1_score",0.67,0.67,0.53,0.40])

print(x)
```

```
+----------------+------+-------+---------+-----------+
|     Model      | Bow  | Tfidf | Avg-W2V | Tfidf-W2V |
+----------------+------+-------+---------+-----------+
|       C        | 0.01 |  0.01 |    5    |    0.1    |
| Train f1_score | 0.74 |  0.95 |   0.51  |    0.41   |
| Test f1_score  | 0.67 |  0.67 |   0.53  |    0.4    |
+----------------+------+-------+---------+-----------+
```