Assignment:-

Applying Logistic Regression on Amazon fine Food Reviews analysis

Amazon Fine Food Reviews Analysis

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

1. Objective:

Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2). Use BoW, TF-IDF, Avg-Word2Vec,TF-IDF-Word2Vec to vectorise the reviews. Apply Logistic Regression Algorithm for Amazon fine food Reviews find right alpha(α) using cross validation Get feature importance for positive class and Negative class

In [1]:

```
# loading required libraries

import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib
import sqlite3
import string
import gensim
import scipy
import nltk
import time
import seaborn as sns
from scipy import stats
from matplotlib import pyplot as plt

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support as prf1

from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

1.1 Connecting SQL file

In [2]:

```
#Loading the data
con = sqlite3.connect('./final.sqlite')

data = pd.read_sql_query("""
SELECT *
FROM Reviews
""", con)
```

In [3]:

```
print(data.shape)
data.head()
```

```
(364171, 12)
```

Out[3]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

1.2 Data Preprocessing

In [4]:

```
data.Score.value_counts()
#i had done data preprocessing i had stored in final.sqlite now loaded this file no need tc
```

Out[4]:

```
positive    307061
negative     57110
Name: Score, dtype: int64
```

1.3 Sorting the data

In [5]:

```
# Sorting the data according to the time-stamp
sorted_data = data.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksc
sorted_data.head()
```

Out[5]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
330	346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
423	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	

1.4 Mapping

In [6]:

```
def partition(x):
    if x == 'positive':
        return 1
    return 0

#Preparing the filtered data
actualScore = sorted_data['Score']
positiveNegative = actualScore.map(partition)
sorted_data['Score'] = positiveNegative
sorted_data.head()
```

Out[6]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
330	346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
423	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	

1.5 Taking First 150K Rows

In [7]:

```
# We will collect different 150000 rows without repetition from time_sorted_data dataframe
my_final = sorted_data[:150000]
print(my_final.shape)
my_final.head()
```

(150000, 12)

Out[7]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
330	346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
423	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	

1.6 Spliting data into train and test based on time (70:30)

In [8]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate


x=my_final['CleanedText'].values
y=my_final['Score']

#Splitting data into train test and cross validation
x_train,x_test,y_train,y_test =train_test_split(x,y,test_size =0.3,random_state = 42)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(105000,)
(45000,)
(105000,)
(45000,)
```

2. Techniques For Vectorization

Why we have to convert text to vector

By converting text to vector we can use whole power of linear algebra.we can find a plane to seperate

Incase of logistic regression to find important features, firstly we have to check multi collinearity between features, if

features are collinear then we should find important features using forward or backward feature selection.

If features are not correlated then we should use optimal vector, in which consist of weight for each feature.

Multi collinearity: which means very high inter correlation among the independent variables.

2.1 BOW

In [9]:

```
#Bow

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
final_counts_Bow_tr= count_vect.fit_transform(x_train)# computing Bow
print("the type of count vectorizer ",type(final_counts_Bow_tr))
print("the shape of out text BOW vectorizer ",final_counts_Bow_tr.get_shape())
print("the number of unique words ", final_counts_Bow_tr.get_shape()[1])
final_counts_Bow_test= count_vect.transform(x_test)# computing Bow
print("the type of count vectorizer ",type(final_counts_Bow_test))
print("the shape of out text BOW vectorizer ",final_counts_Bow_test.get_shape())
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (105000, 38300)
the number of unique words  38300
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (45000, 38300)
```

2.1.1 Standardizing Data

In [10]:

```
# Data-preprocessing: Standardizing the data

from sklearn import preprocessing
standardized_data_train = preprocessing.normalize(final_counts_Bow_tr)
print(standardized_data_train.shape)
standardized_data_test = preprocessing.normalize(final_counts_Bow_test)
print(standardized_data_test.shape)
```

```
(105000, 38300)
(45000, 38300)
```

2.2 Applying Logistic Regression Algorithm

2.2.1 Gridsearch Cross Validation

2.2.1.1 Using L1 Regularization

In [15]:

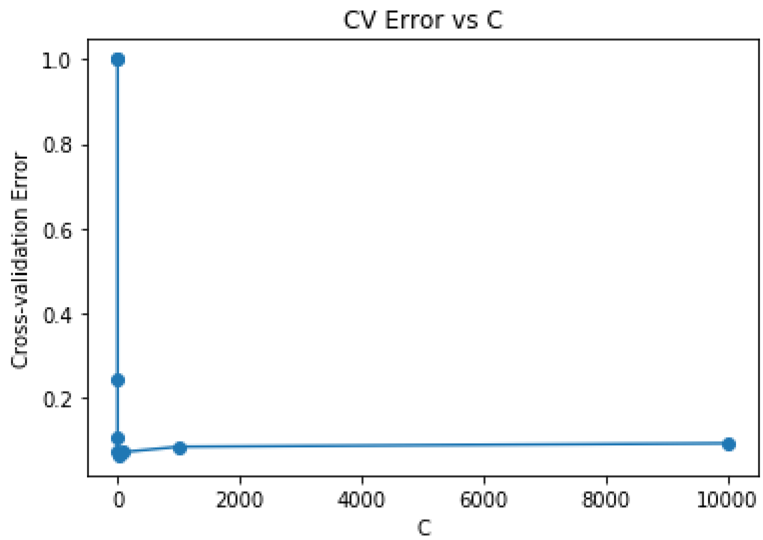
```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("Accuracy of the test model : ",model.score(standardized_data_test, y_test))
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=10, class_weight
='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
Accuracy of the test model :  0.9360583243695472
```

Plotting a graph between C vs CV Error

In [21]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [22]:

```
model.best_params_
```

Out[22]:

```
{'C': 10}
```

In [23]:

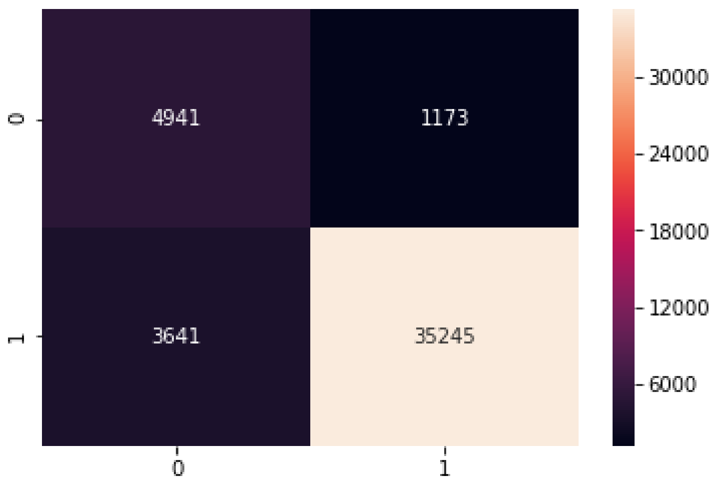
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l1',class_weight='balanced', C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_bow = lr.predict(standardized_data_test)
```

2.3 Confusion Matrix

In [24]:

```
cm_bow=confusion_matrix(y_test,pred_bow)
print("Confusion Matrix:")
sns.heatmap(cm_bow, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [25]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_bow.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4941
 false positives are 1173
 false negatives are 3641
 true positives are 35245
```

2.4 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [26]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_bow) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_bow = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_bow))

# evaluating precision
precision_score = precision_score(y_test, pred_bow)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_bow)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_bow)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 10.000 is 9
3.607245%

Test Error  Logistic Regression classifier is  6.392755%

The Test Precision of the Logistic Regression classifier for C = 10.000 is
0.967791

The Test Recall of the Logistic Regression classifier for C = 10.000 is 0.90
6367

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.58      0.81      0.67      6114
           1       0.97      0.91      0.94     38886

   micro avg       0.89      0.89      0.89     45000
   macro avg       0.77      0.86      0.80     45000
weighted avg       0.91      0.89      0.90     45000
```

2.5 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [27]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [28]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[-0.1873746   0.00267988  0.00070299 ...  0.07092126  0.00300
243
   0.02447227]]
```

Out[28]:

```
array([80.43086568, 68.34253473, 63.44736738, 62.67995684, 61.84514268,
       56.74291376, 48.17445131, 47.03001171, 47.00663241, 45.7840874 ,
       45.65869711, 45.54055911, 43.44398893, 42.32506811, 42.15079776,
       41.85722319, 40.20008194, 40.06860416, 39.52365714, 39.39666727])
```

In [29]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = count_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
   mozzerela	===> 	-81.221340
    sonewher	===> 	-69.028203
          ov	===> 	-64.114077
     devault	===> 	-63.686052
yadayadayada	===> 	-63.632296
       dcide	===> 	-57.967970
       gould	===> 	-50.026035
     disastr	===> 	-49.416620
     conceal	===> 	48.887017
     cucazza	===> 	-47.661094
     goodwil	===> 	-46.443136
    distrust	===> 	-45.964719
  robitussin	===> 	-45.468910
        coil	===> 	-44.417557
     differr	===> 	-43.397257
        lvoe	===> 	-42.912444
         yap	===> 	-42.056988
       sheat	===> 	-40.935271
   valentina	===> 	-40.833585
    allrecip	===> 	-40.803531
```

2.7 Using L2 Regularization

In [31]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L2 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=100, class_weigh
t='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L2 Regularization 0.9360205922938117
```

In [32]:

```
model.best_params_
```

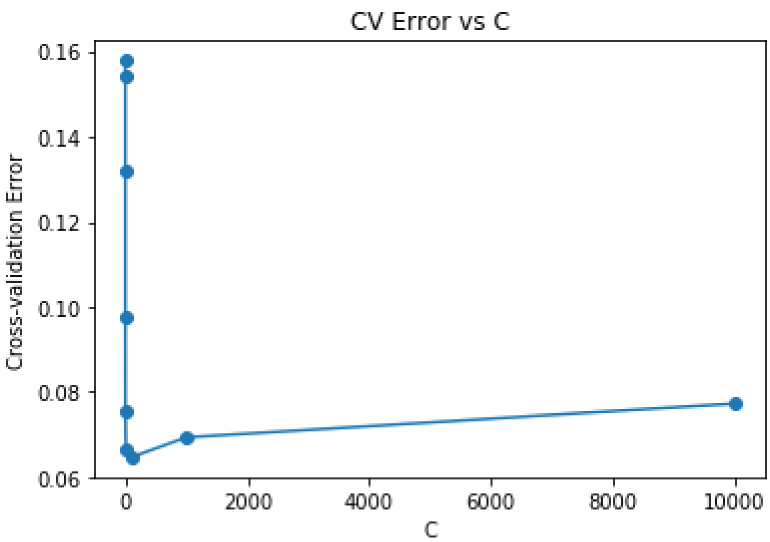
Out[32]:

```
{'C': 100}
```

Plotting a graph between C vs CV Error

In [33]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [34]:

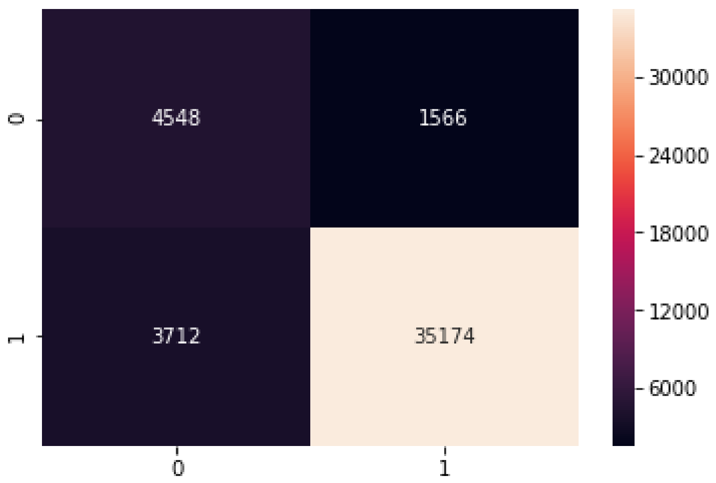
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l1', C=optimal_C, class_weight='balanced',n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_bow = lr.predict(standardized_data_test)
```

2.8 Confusion Matrix

In [35]:

```
cm_bow=confusion_matrix(y_test,pred_bow)
print("Confusion Matrix:")
sns.heatmap(cm_bow, annot=True, fmt='d')
plt.show()
```

```
Confusion Matrix:
```



In [36]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_bow.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4548
 false positives are 1566
 false negatives are 3712
 true positives are 35174
```

2.9 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [37]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_bow) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_bow = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_bow))

# evaluating precision
precision_score = precision_score(y_test, pred_bow)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_bow)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_bow)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 100.000 is 9
3.020919%

Test Error  Logistic Regression classifier is  6.979081%

The Test Precision of the Logistic Regression classifier for C = 100.000 is
0.957376

The Test Recall of the Logistic Regression classifier for C = 100.000 is 0.9
04541

The Test classification report of the Logistic regression classifier for C

              precision    recall  f1-score   support

           0       0.55      0.74      0.63      6114
           1       0.96      0.90      0.93     38886

   micro avg       0.88      0.88      0.88     45000
   macro avg       0.75      0.82      0.78     45000
weighted avg       0.90      0.88      0.89     45000
```

3.Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [38]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [39]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[-0.3867784   0.013296    0.00259353 ...  0.52455324  0.01590
12
   0.20404927]]
```

Out[39]:

```
array([279.88916404, 244.91940143, 243.18414983, 224.58891442,
       213.15402917, 184.8338223 , 182.25287213, 174.99382919,
       171.76343844, 169.23931672, 162.98911142, 162.7482134 ,
       161.09247776, 158.70466949, 157.13688115, 156.48302254,
       152.58759764, 151.56174503, 150.27188927, 149.85086208])
```

In [40]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = count_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
     fishier    ===>    -285.189896
      corect    ===>    -251.244529
      reclin    ===>    243.727351
     uninari    ===>    -236.927149
    advantix    ===>    -218.522241
      conlus    ===>    -188.172908
        poem    ===>    185.440551
     jivalim    ===>    -179.081780
     goodwil    ===>    -178.036782
    sonewher    ===>    -176.035821
   mozzerela    ===>    -170.565604
        ridx    ===>    -168.175897
     hermosa    ===>    -165.468684
       anywh    ===>    -164.294790
      glimps    ===>    -161.119542
    crosswis    ===>    -160.959288
 opportunist    ===>    -156.879202
   grainiest    ===>    -156.172455
  settlement    ===>    155.095231
     colicki    ===>    154.755744
```

3.1 Randomized Search Cross Validation

3.1.1 Using L1 Regularization

In [43]:

```
# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=100, class_weigh
t='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.9302091873165313
```

In [44]:

```
model.best_params_
```

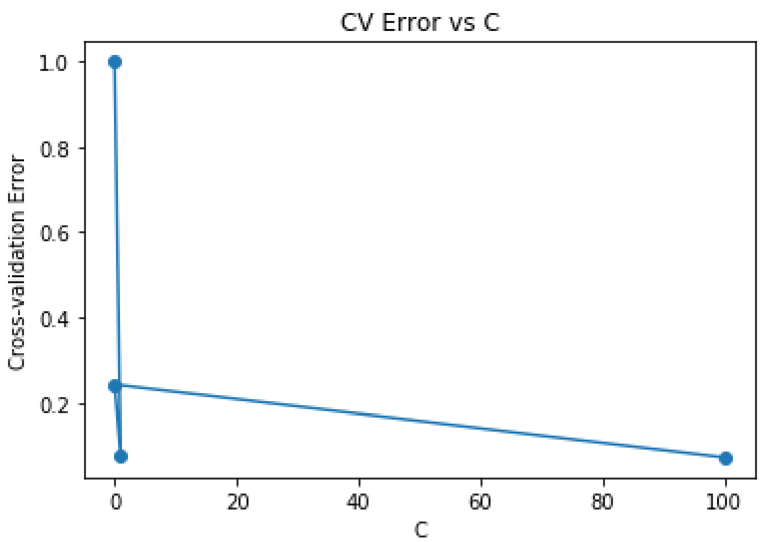
Out[44]:

```
{'C': 100}
```

Plotting a graph between C vs CV Error

In [45]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [46]:

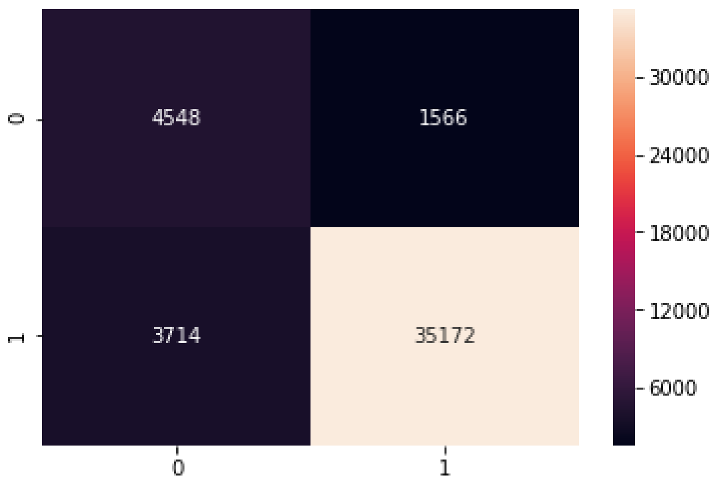
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l1', C=optimal_C,class_weight='balanced', n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_bow = lr.predict(standardized_data_test)
```

3.2 Confusion Matrix

In [47]:

```
cm_bow=confusion_matrix(y_test,pred_bow)
print("Confusion Matrix:")
sns.heatmap(cm_bow, annot=True, fmt='d')
plt.show()
```

```
Confusion Matrix:
```



In [48]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_bow.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4548
 false positives are 1566
 false negatives are 3714
 true positives are 35172
```

3.3 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [49]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_bow) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_bow = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_bow))

# evaluating precision
precision_score = precision_score(y_test, pred_bow)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_bow)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_bow)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 100.000 is 9
3.018089%

Test Error  Logistic Regression classifier is  6.981911%

The Test Precision of the Logistic Regression classifier for C = 100.000 is
0.957374

The Test Recall of the Logistic Regression classifier for C = 100.000 is 0.9
04490

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.55      0.74      0.63      6114
           1       0.96      0.90      0.93     38886

   micro avg       0.88      0.88      0.88     45000
   macro avg       0.75      0.82      0.78     45000
weighted avg       0.90      0.88      0.89     45000
```

3.4 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [50]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [51]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[-0.38614634  0.01331118  0.00259891 ...  0.5272905   0.01586
398
   0.20437205]]
```

Out[51]:

```
array([279.89783694, 244.92999676, 243.273134  , 224.13152239,
       213.19758887, 184.79413525, 182.30110335, 174.95294991,
       171.72373889, 169.2394331 , 162.7680524 , 161.05276989,
       158.38835047, 157.22959459, 157.04489649, 156.51135248,
       152.63940584, 151.55381123, 150.24652117, 149.86025713])
```

In [52]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = count_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
     fishier    ===>    -285.206418
      corect    ===>    -251.268055
      reclin    ===>    243.816896
     uninari    ===>    -236.471964
    advantix    ===>    -218.572686
      conlus    ===>    -188.138885
        poem    ===>    185.488846
     jivalim    ===>    -179.050896
     goodwil    ===>    -178.002684
    sonewher    ===>    -176.042775
   mozzerela    ===>    -170.591002
        ridx    ===>    -168.138783
       anywh    ===>    -162.639048
      glimps    ===>    -161.218308
    crosswis    ===>    -160.997334
     hermosa    ===>    -160.879583
 opportunist    ===>    -156.866545
   grainiest    ===>    -156.173835
  settlement    ===>    155.111289
     colicki    ===>    154.809600
```

3.5 Using L2 Regularization

In [53]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L2 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=100, class_weigh
t='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L2 Regularization 0.9360205922938117
```

In [54]:

```
model.best_params_
```

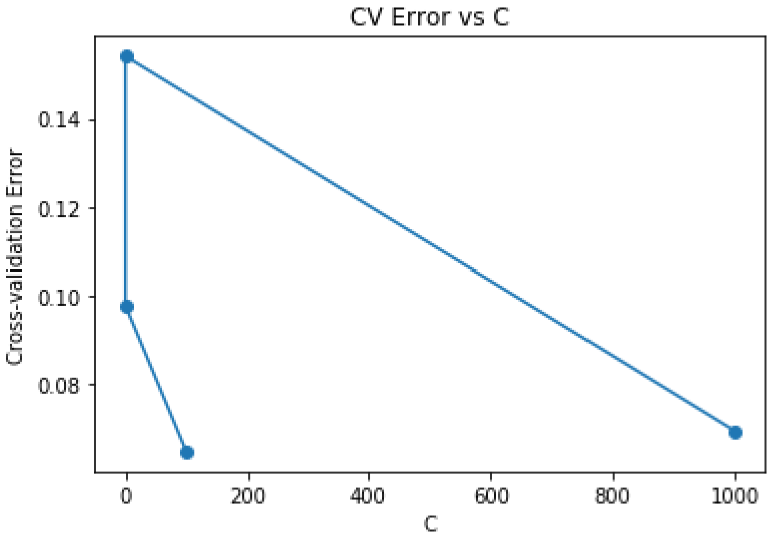
Out[54]:

```
{'C': 100}
```

Plotting a graph between C vs CV Error

In [55]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [56]:

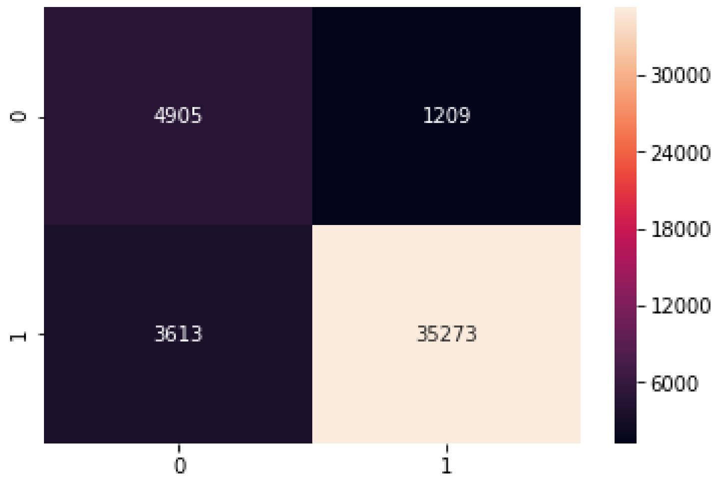
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2',class_weight='balanced', C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_bow = lr.predict(standardized_data_test)
```

3.6 Confusion Matrix

In [57]:

```
cm_bow=confusion_matrix(y_test,pred_bow)
print("Confusion Matrix:")
sns.heatmap(cm_bow, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [58]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_bow.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4905
 false positives are 1209
 false negatives are 3613
 true positives are 35273
```

3.7 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [59]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_bow) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (o

# Error on test data
test_error_bow = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_bow))

# evaluating precision
precision_score = precision_score(y_test, pred_bow)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_bow)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_bow)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 100.000 is 9
3.602059%

Test Error  Logistic Regression classifier is  6.397941%

The Test Precision of the Logistic Regression classifier for C = 100.000 is
0.966860

The Test Recall of the Logistic Regression classifier for C = 100.000 is 0.9
07087

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.58      0.80      0.67      6114
           1       0.97      0.91      0.94     38886

   micro avg       0.89      0.89      0.89     45000
   macro avg       0.77      0.85      0.80     45000
weighted avg       0.91      0.89      0.90     45000
```

3.8 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [60]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [61]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[ 0.08501586 -0.01889445 -0.00221554 ... -0.08913331 -0.02491
728
  -0.1553935 ]]
```

Out[61]:

```
array([15.3546002 , 14.81342372, 14.24930477, 14.0817224 , 13.89300998,
       13.78803927, 12.40404504, 12.1563743 , 12.07690168, 11.91967391,
       11.40036526, 11.34016018, 11.33498141, 10.77895681, 10.76065832,
       10.63107242, 10.57098597, 10.36789833, 10.27637153,  9.95360835])
```

In [62]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = count_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
     finnish	===> 	-30.804405
yadayadayada	===> 	-28.732250
     compass	===> 	-27.390156
   sleepless	===> 	-26.316562
    skinnier	===> 	-26.128913
       worst	===> 	-26.083704
      cosmos	===> 	-25.693283
        coil	===> 	-24.637455
     nicknam	===> 	-24.509341
       avert	===> 	-24.287710
      abomin	===> 	-23.418000
       gould	===> 	-22.707792
      innard	===> 	-22.500457
     puberti	===> 	-22.419660
     dorothi	===> 	-22.364858
     riducul	===> 	-22.072504
     disastr	===> 	-21.993514
    downtown	===> 	-21.979630
     conceal	===> 	21.906507
   wunderbar	===> 	-21.874626
```

4. TF-IDF

In [63]:

```
#tf-idf
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer()

final_counts_tfidf_tr= tf_idf_vect.fit_transform(x_train)
print("the type of count vectorizer ",type(final_counts_tfidf_tr))
print("the shape of out text tfidf vectorizer ",final_counts_tfidf_tr.get_shape())
print("the number of unique words ", final_counts_tfidf_tr.get_shape()[1])
final_counts_tfidf_test= tf_idf_vect.transform(x_test)
print("the type of count vectorizer ",type(final_counts_tfidf_test))
print("the shape of out text tfidf vectorizer ",final_counts_tfidf_test.get_shape())
print("the number of unique words ", final_counts_tfidf_test.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text tfidf vectorizer  (105000, 38300)
the number of unique words  38300
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text tfidf vectorizer  (45000, 38300)
the number of unique words  38300
```

4.1 Standardizing Data

In [64]:

```
# Data-preprocessing: Standardizing the data
from sklearn import preprocessing
standardized_data_train = preprocessing.normalize(final_counts_tfidf_tr)
print(standardized_data_train.shape)
standardized_data_test = preprocessing.normalize(final_counts_tfidf_test)
print(standardized_data_test.shape)
```

```
(105000, 38300)
(45000, 38300)
```

4.2 Applying Logistic Regression Algorithm

4.2.1 Gridsearch Cross Validation

4.2.1.1 Using L1 Regularization

In [66]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=10, class_weight
='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.9351884979791957
```

In [67]:

```
model.best_params_
```

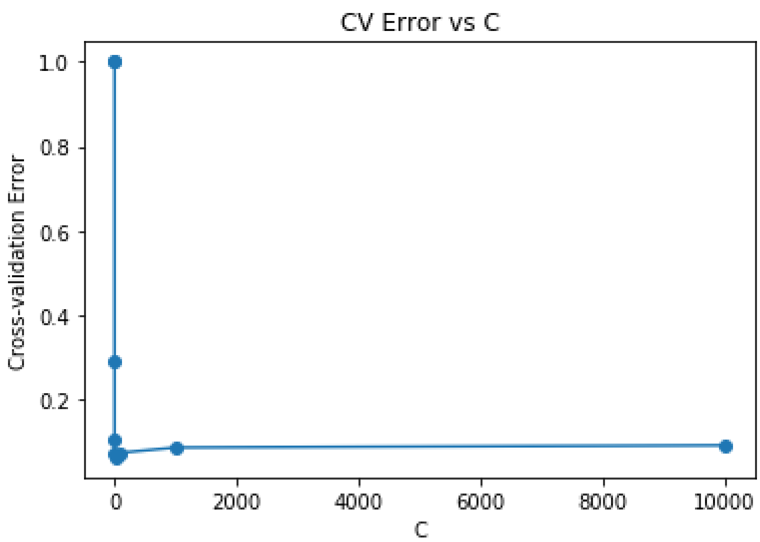
Out[67]:

```
{'C': 10}
```

Plotting a graph between C vs CV Error

In [68]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [69]:

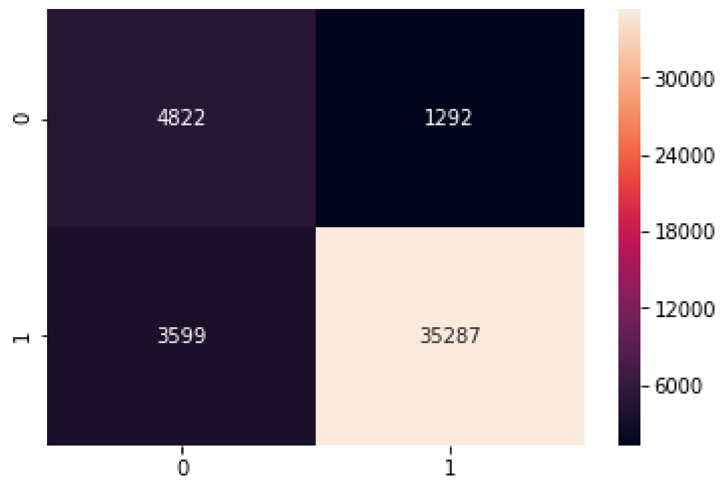
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l1', class_weight='balanced',C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidf = lr.predict(standardized_data_test)
```

4.3 Confusion Matrix

In [70]:

```
cm_tfidf=confusion_matrix(y_test,pred_tfidf)
print("Confusion Matrix:")
sns.heatmap(cm_tfidf, annot=True, fmt='d')
plt.show()
```

```
Confusion Matrix:
```



In [71]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidf.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4822
 false positives are 1292
 false negatives are 3599
 true positives are 35287
```

4.4 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [72]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidf) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_tfidf = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidf))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidf)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidf)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optin

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidf)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 10.000 is 9
3.518850%

Test Error  Logistic Regression classifier is  6.481150%

The Test Precision of the Logistic Regression classifier for C = 10.000 is
0.964679

The Test Recall of the Logistic Regression classifier for C = 10.000 is 0.90
7447

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.57      0.79      0.66      6114
           1       0.96      0.91      0.94     38886

   micro avg       0.89      0.89      0.89     45000
   macro avg       0.77      0.85      0.80     45000
weighted avg       0.91      0.89      0.90     45000
```

4.5 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [73]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [74]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[-0.27710367  0.00680627  0.00171921 ...  0.17550179  0.00778
236
   0.07935638]]
```

Out[74]:

```
array([47.28096945, 42.82660647, 39.69945422, 38.34484204, 36.99785103,
       35.19604317, 35.06539774, 33.58262556, 32.48656314, 32.03343388,
       31.50108337, 30.92770186, 29.19235448, 28.99190036, 28.76153169,
       28.68851382, 28.64328311, 28.5461175 , 28.50626186, 28.0785017 ])
```

In [75]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = tf_idf_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
      corect	===> 	-48.872809
   mozzerela	===> 	-44.468918
    sonewher	===> 	-41.609026
     fishier	===> 	-39.331064
     goodwil	===> 	-38.455216
          ov	===> 	-37.042570
     jivalim	===> 	-36.051134
     devault	===> 	-35.923801
yadayadayada	===> 	-35.870142
        ridx	===> 	-33.457121
   grainiest	===> 	-32.592764
    advantix	===> 	-32.038224
   insuffici	===> 	-31.829199
   voluntari	===> 	-31.356050
     merritt	===> 	-31.162582
disstributor	===> 	30.593466
  opportunist	===> 	-30.300442
   pessimist	===> 	-30.116889
       dcide	===> 	-29.952377
    insignia	===> 	-29.554033
```

4.6 Using L2 Regularization

In [76]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L2 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=10, class_weight
='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L2 Regularization 0.9363146336594129
```

In [77]:

```
model.best_params_
```

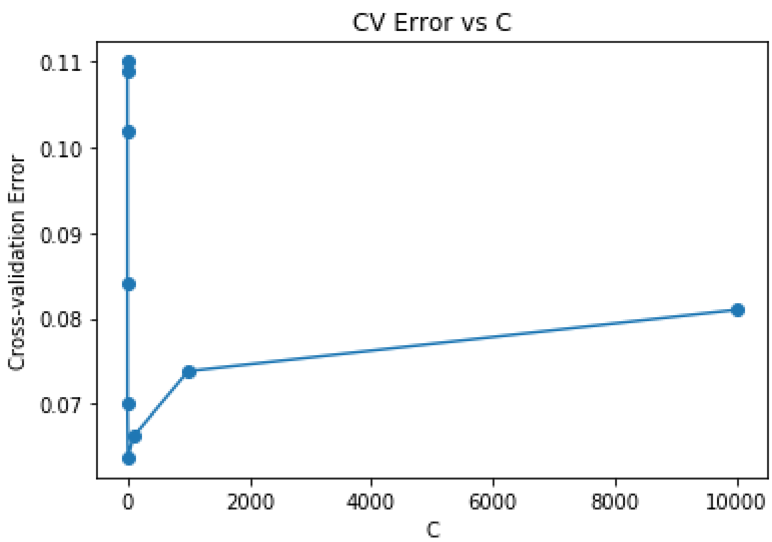
Out[77]:

```
{'C': 10}
```

Plotting a graph between C vs CV Error

In [78]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [79]:

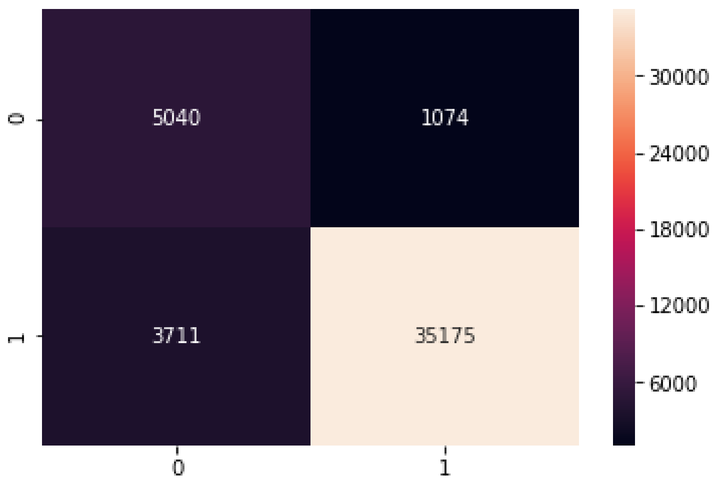
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2',class_weight='balanced', C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidf = lr.predict(standardized_data_test)
```

4.7 Confusion Matrix

In [80]:

```
cm_tfidf=confusion_matrix(y_test,pred_tfidf)
print("Confusion Matrix:")
sns.heatmap(cm_tfidf, annot=True, fmt='d')
plt.show()
```

```
Confusion Matrix:
```



In [81]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidf.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 5040
 false positives are 1074
 false negatives are 3711
 true positives are 35175
```

4.8 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [82]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidf) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_tfidf = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidf))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidf)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidf)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidf)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 10.000 is 9
3.631463%

Test Error  Logistic Regression classifier is  6.368537%

The Test Precision of the Logistic Regression classifier for C = 10.000 is
0.970372

The Test Recall of the Logistic Regression classifier for C = 10.000 is 0.90
4567

The Test classification report of the Logistic regression classifier for C

               precision    recall  f1-score   support

           0       0.58      0.82      0.68      6114
           1       0.97      0.90      0.94     38886

   micro avg       0.89      0.89      0.89     45000
   macro avg       0.77      0.86      0.81     45000
weighted avg       0.92      0.89      0.90     45000
```

4.9 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [83]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [84]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[ 0.01708526 -0.00745136 -0.0015592  ... -0.04162235 -0.01324
496
  -0.08478265]]
```

Out[84]:

```
array([5.70187706, 4.9461636 , 4.81924635, 4.69331995, 4.59300109,
       4.39143394, 4.38165593, 4.31749051, 4.30694245, 4.21006546,
       4.16142254, 4.01823138, 3.98269906, 3.96067984, 3.77184892,
       3.70826911, 3.69791019, 3.54576098, 3.54221151, 3.53803479])
```

In [85]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = tf_idf_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
       worst	===> 	-16.506950
       great	===> 	13.935663
      delici	===> 	13.480462
        best	===> 	12.510159
     perfect	===> 	12.193799
        amaz	===> 	11.647077
     terribl	===> 	-11.615033
        love	===> 	11.590019
  disappoint	===> 	-11.466183
     skeptic	===> 	10.841531
        hook	===> 	10.564131
        beat	===> 	10.195225
     horribl	===> 	-10.162381
      addict	===> 	9.747666
       excel	===> 	9.613231
     finnish	===> 	-9.521818
   sleepless	===> 	-9.335980
      awesom	===> 	9.327845
     mediocr	===> 	-9.293814
     concept	===> 	-9.088242
```

5. Randomized Search Cross Validation

5.1 Using L1 Regularization

In [86]:

```
# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=10000, class_wei
ght='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.9192951007657937
```

In [87]:

```
model.best_params_
```

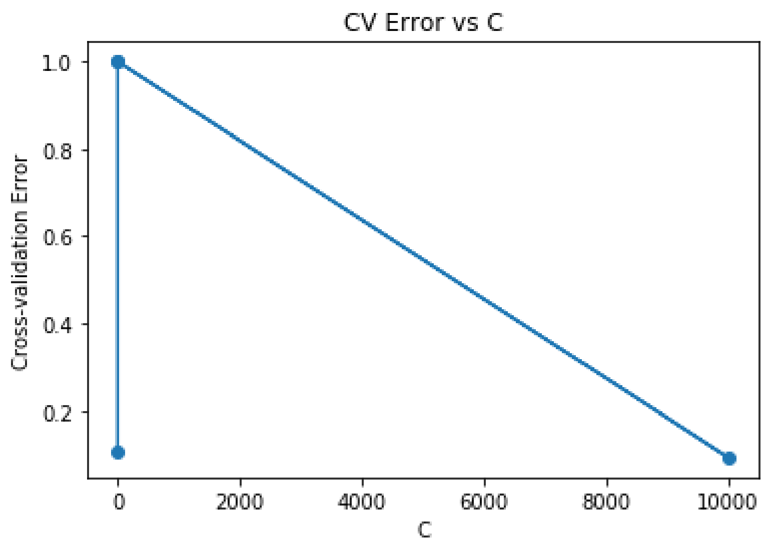
Out[87]:

```
{'C': 10000}
```

Plotting a graph between C vs CV Error

In [88]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [89]:

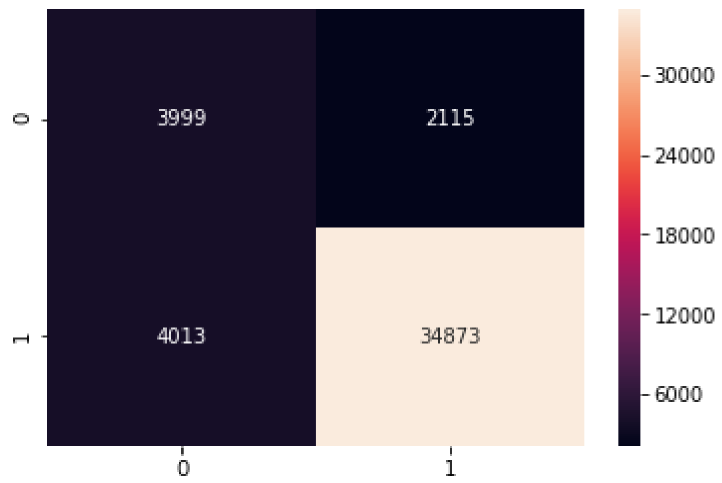
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l1', C=optimal_C,class_weight='balanced', n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidf = lr.predict(standardized_data_test)
```

5.2 Confusion Matrix

In [90]:

```
cm_tfidf=confusion_matrix(y_test,pred_tfidf)
print("Confusion Matrix:")
sns.heatmap(cm_tfidf, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [91]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidf.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 3999
 false positives are 2115
 false negatives are 4013
 true positives are 34873
```

5.3 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [92]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidf) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_tfidf = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidf))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidf)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidf)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidf)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 10000.000 is
91.923452%

Test Error  Logistic Regression classifier is  8.076548%

The Test Precision of the Logistic Regression classifier for C = 10000.000 i
s 0.942819

The Test Recall of the Logistic Regression classifier for C = 10000.000 is
0.896801

The Test classification report of the Logistic regression classifier for C

               precision    recall  f1-score   support

           0       0.50      0.65      0.57      6114
           1       0.94      0.90      0.92     38886

   micro avg       0.86      0.86      0.86     45000
   macro avg       0.72      0.78      0.74     45000
weighted avg       0.88      0.86      0.87     45000
```

5.4 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [93]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [94]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[5.72958555 0.15190945 0.01363539 ... 1.68188796 0.31407761
2.41157368]]
```

Out[94]:

```
array([705.54878152, 545.72212982, 536.37754024, 470.86663565,
       419.71316906, 413.09228173, 397.85290011, 397.22695379,
       393.19836303, 365.0708472 , 360.41327023, 347.88514845,
       345.41022466, 342.87193146, 337.86331378, 334.88698489,
       333.11382585, 332.15545253, 329.02264881, 328.56727238])
```

In [95]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = tf_idf_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
 hypothyroid	===> 	718.963735
       crohn	===> 	573.121601
antihistamin	===> 	549.200725
 lingonberri	===> 	481.851774
       scald	===> 	442.063725
    kalocsai	===> 	-423.824538
        wfgf	===> 	420.262767
      piazza	===> 	-419.366668
      deglaz	===> 	415.791976
    dismantl	===> 	-385.842543
    misrepres	===> 	-378.602006
      aachen	===> 	-378.119805
      drakar	===> 	-378.024980
      reclin	===> 	374.885015
    advantix	===> 	-373.630969
      treacl	===> 	372.488836
  talleyrand	===> 	365.259786
     preffer	===> 	360.788589
    extravag	===> 	-350.868300
    spinachi	===> 	340.726160
```

5.5 Using L2 Regularization

In [96]:

```
# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L2 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=100, class_weigh
t='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L2 Regularization 0.933857913073428
```

In [97]:

```
model.best_params_
```

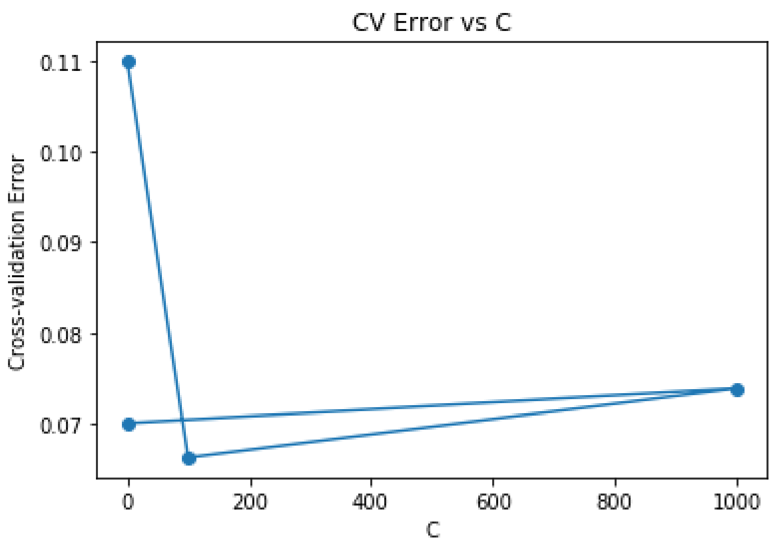
Out[97]:

```
{'C': 100}
```

Plotting a graph between C vs CV Error

In [98]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [99]:

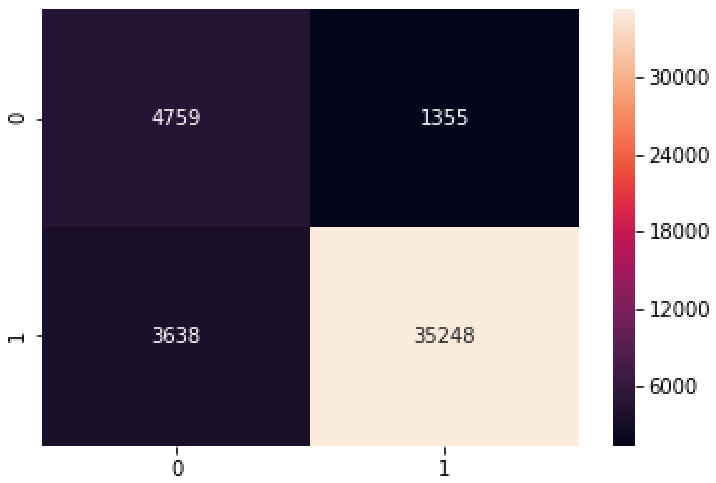
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2', class_weight='balanced',C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidf = lr.predict(standardized_data_test)
```

5.6 Confusion Matrix

In [100]:

```
cm_tfidf=confusion_matrix(y_test,pred_tfidf)
print("Confusion Matrix:")
sns.heatmap(cm_tfidf, annot=True, fmt='d')
plt.show()
```

```
Confusion Matrix:
```



In [101]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidf.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4759
 false positives are 1355
 false negatives are 3638
 true positives are 35248
```

5.7 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [102]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidf) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_tfidf = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidf))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidf)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidf)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidf)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 100.000 is 9
3.385791%

Test Error  Logistic Regression classifier is  6.614209%

The Test Precision of the Logistic Regression classifier for C = 100.000 is
0.962981

The Test Recall of the Logistic Regression classifier for C = 100.000 is 0.9
06444

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.57      0.78      0.66      6114
           1       0.96      0.91      0.93     38886

   micro avg       0.89      0.89      0.89     45000
   macro avg       0.76      0.84      0.79     45000
weighted avg       0.91      0.89      0.90     45000
```

5.8 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [103]:

```
import scipy as sp
epsilon = sp.stats.distributions.norm.rvs(loc=0,scale=0.0001)
# Vector before the addition of epsilon
W_before_epsilon = lr.coef_

# Number of non zero elements in Standardized_train sparse matrix
no_of_non_zero = standardized_data_train.count_nonzero()

# Importing library to create a sparse matrix of epsilon
from scipy.sparse import csr_matrix

# Creating new sparse matrix with epsilon at same position of non-zero elements of standard
x_train_indices = standardized_data_train.indices
x_train_indptr = standardized_data_train.indptr #CSR format index pointer array of the matr

# Creating a list of same element with repetition
data = [epsilon] * no_of_non_zero
Shape = standardized_data_train.shape

# Creating sparse matrix
sparse_epsilon = csr_matrix((data,x_train_indices,x_train_indptr),shape=Shape,dtype=float)

# Add sparse_epsilon and X-standardized_data_train to get a new sparse matrix with epsilon
# non-zero element of standardized_data_train
epsilon_train = standardized_data_train + sparse_epsilon

print(standardized_data_train.shape)
print(epsilon_train.shape)
```

```
(105000, 38300)
(105000, 38300)
```

In [104]:

```
# training Logistic Regression Classifier with epsilon_train
epsilon_lr = LogisticRegression(penalty='l2', C=optimal_C, n_jobs=-1)
epsilon_lr.fit(epsilon_train,y_train)

# Vector after the addition of epsilon
W_after_epsilon = epsilon_lr.coef_

# Change in vectors after adding epsilon
change_vector = W_after_epsilon - W_before_epsilon
print("change_vector",change_vector)


# Sort this change_vector array after making all the elements positive in ascending order t
sorted_change_vector = np.sort(np.absolute(change_vector))[:,::-1]

sorted_change_vector[0,0:20]
```

```
change_vector [[ 0.08771318 -0.03604998  0.00046193 ...  0.02653029 -0.03382
972
  -0.2034842 ]]
```

Out[104]:

```
array([11.8980787 , 10.71434974,  9.88321025,  9.71325821,  9.26748008,
        9.17898238,  9.10531123,  8.88205754,  8.27490006,  8.18450277,
        8.18372848,  8.07494542,  7.94326188,  7.91036549,  7.55704613,
        7.4815967 ,  7.325322  ,  7.32500814,  7.15969046,  7.0403881 ])
```

In [105]:

```
absolute_weights = np.absolute(W_before_epsilon)
sorted_absolute_index = np.argsort(absolute_weights)[:,::-1]
top_index = sorted_absolute_index[0,0:20]

all_features = tf_idf_vect.get_feature_names()
weight_values = lr.coef_

# Top 20 features are
print("Top 20 features with their weight values :")

for j in top_index:
    print("%12s\t===> \t%f"%(all_features[j],weight_values[0,j]))
```

```
Top 20 features with their weight values :
yadayadayada    ===>    -27.923918
          ov    ===>    -26.527925
   mozzerela    ===>    -24.951461
       worst    ===>    -23.527706
        coil    ===>    -20.754045
    sonewher    ===>    -20.435857
     finnish    ===>    -20.304019
    distrust    ===>    -20.243382
      cystic    ===>    -19.635566
     conceal    ===>    19.449148
     compass    ===>    -19.395615
     skeptic    ===>    19.341727
     uninari    ===>    -18.887920
    statesid    ===>    -18.872892
       gould    ===>    -18.790618
     fishier    ===>    -18.545537
    allrecip    ===>    -18.328219
         eng    ===>    -18.290134
     disastr    ===>    -18.207121
      ransid    ===>    -18.204572
```

6. WORD2VEC

In [106]:

```
from gensim.models import Word2Vec
# List of sentence in X_train text
sent_of_train=[]
for sent in x_train:
    sent_of_train.append(sent.split())

# List of sentence in X_est text
sent_of_test=[]
for sent in x_test:
    sent_of_test.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  12829
```

7. Avg Word2Vec

In [107]:

```
# compute average word2vec for each review for X_train .
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)

# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)
```

7.1 Standardizing Data

In [108]:

```
# Data-preprocessing: Standardizing the data
from sklearn import preprocessing
standardized_data_train = preprocessing.normalize(train_vectors)
print(standardized_data_train.shape)
standardized_data_test = preprocessing.normalize(test_vectors)
print(standardized_data_test.shape)
```

```
(105000, 50)
(45000, 50)
```

7.2 Applying Logistic Regression Algorithm

7.2.1 Gridsearch Cross Validation

7.2.1.1 Using L1 Regularization

In [109]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=1000, class_weig
ht='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.888227108266073
```

In [110]:

```
model.best_params_
```

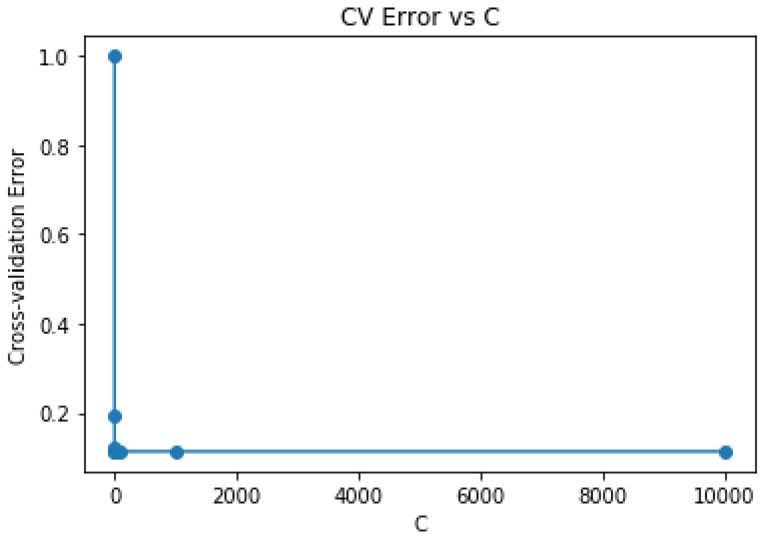
Out[110]:

```
{'C': 1000}
```

Plotting a graph between C vs CV Error

In [111]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [112]:

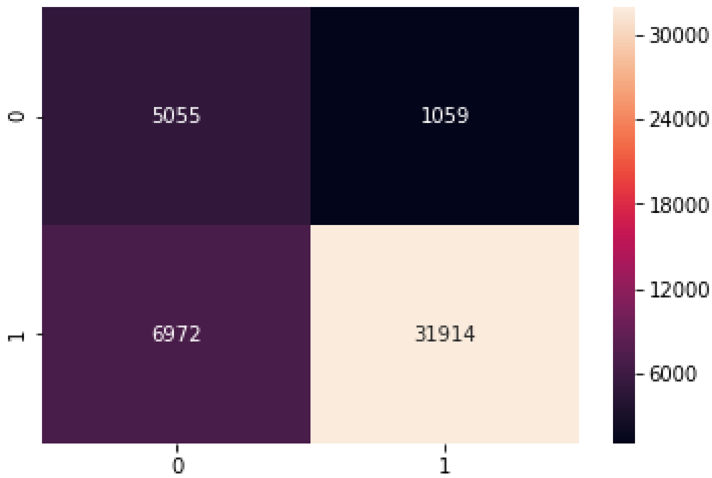
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l1', class_weight='balanced',C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_avgw2v = lr.predict(standardized_data_test)
```

7.3 Confusion Matrix

In [113]:

```
cm_avgw2v=confusion_matrix(y_test,pred_avgw2v)
print("Confusion Matrix:")
sns.heatmap(cm_avgw2v, annot=True, fmt='d')
plt.show()
```

```
Confusion Matrix:
```



In [114]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_avgw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 5055
 false positives are 1059
 false negatives are 6972
 true positives are 31914
```

7.4 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [115]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_avgw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_avgw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_avgw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_avgw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_avgw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_avgw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 1000.000 is
88.823947%

Test Error  Logistic Regression classifier is  11.176053%

The Test Precision of the Logistic Regression classifier for C = 1000.000 is
0.967883

The Test Recall of the Logistic Regression classifier for C = 1000.000 is 0.
820707

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.42      0.83      0.56      6114
           1       0.97      0.82      0.89     38886

   micro avg       0.82      0.82      0.82     45000
   macro avg       0.69      0.82      0.72     45000
weighted avg       0.89      0.82      0.84     45000
```

7.5 Checking sparsity with increasing value of lambda(decreasing C)¶

In [116]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=47
lambda=100.0 ; non-zeros=22
lambda=1000.0 ; non-zeros=2
lambda=10000.0 ; non-zeros=0
```

In [117]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[117]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [118]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= 0.0029464113914069215
```

In [119]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train, y_train)
```

Out[119]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [120]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 0
```

Hence Number of Features whose coefficients changed by more than 40% is less tese are less collineare hence we cannot calculate feature importance

As weight vector values before and after pertubation changes significantly, then we can't use |w| as feature importance measure.

7.6 Using L2 Regularization

In [122]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-4,10**-3,10**-2,10**-1, 1, 10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L2 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=100, class_weigh
t='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L2 Regularization 0.8900891394296188
```

In [123]:

```
model.best_params_
```

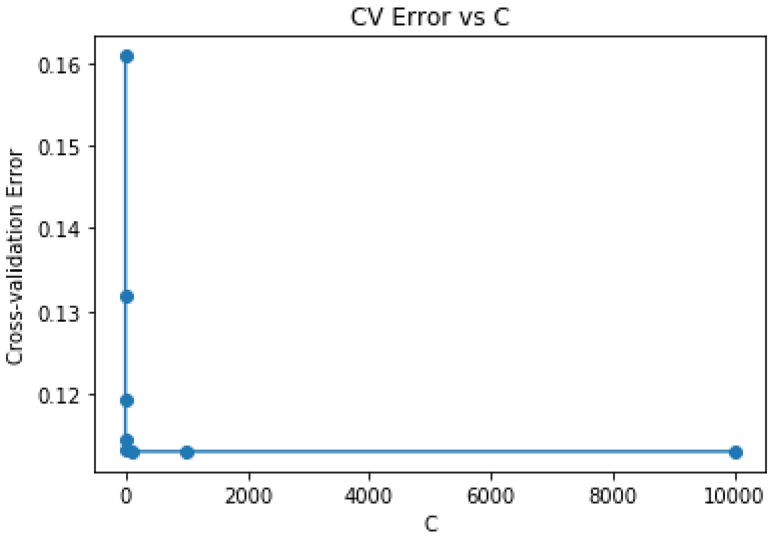
Out[123]:

```
{'C': 100}
```

Plotting a graph between C vs CV Error

In [124]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [125]:

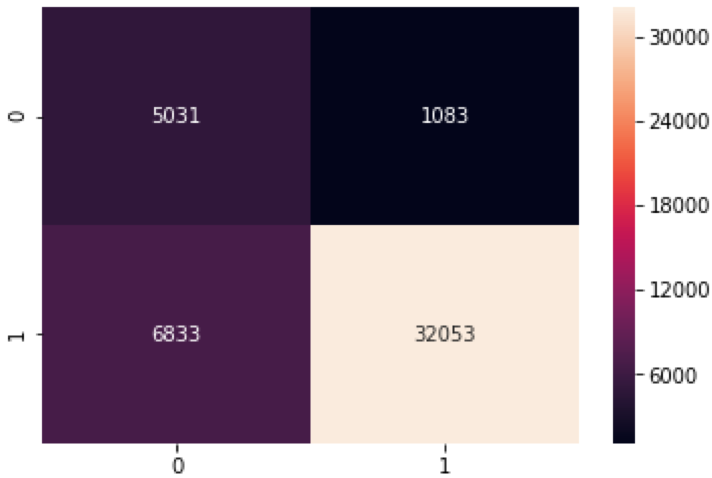
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2', class_weight='balanced',C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_avgw2v = lr.predict(standardized_data_test)
```

7.7 Confusion Matrix

In [126]:

```
cm_avgw2v=confusion_matrix(y_test,pred_avgw2v)
print("Confusion Matrix:")
sns.heatmap(cm_avgw2v, annot=True, fmt='d')
plt.show()
```

```
Confusion Matrix:
```



In [127]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_avgw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 5031
 false positives are 1083
 false negatives are 6833
 true positives are 32053
```

7.8 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [128]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_avgw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_avgw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_avgw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_avgw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_avgw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_avgw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 100.000 is 8
9.008914%

Test Error  Logistic Regression classifier is  10.991086%

The Test Precision of the Logistic Regression classifier for C = 100.000 is
0.967317

The Test Recall of the Logistic Regression classifier for C = 100.000 is 0.8
24281

The Test classification report of the Logistic regression classifier for C

              precision    recall  f1-score   support

           0       0.42      0.82      0.56      6114
           1       0.97      0.82      0.89     38886

   micro avg       0.82      0.82      0.82     45000
   macro avg       0.70      0.82      0.72     45000
weighted avg       0.89      0.82      0.85     45000
```

7.9 Checking sparsity with increasing value of lambda(decreasing C)¶

In [129]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=47
lambda=100.0 ; non-zeros=22
lambda=1000.0 ; non-zeros=2
lambda=10000.0 ; non-zeros=0
```

In [130]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[130]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [131]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= 0.08343477240791726
```

In [132]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train, y_train)
```

Out[132]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [133]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 10
```

Hence Number of Features whose coefficients changed by more than 40% is less tese are less collineare hence we cannot calculate feature importance

7.10 Randomized Search Cross Validation

7.10.1 Using L1 Regularization

In [134]:

```
# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=1000, class_weig
ht='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.9300732217573222
```

In [135]:

```
model.best_params_
```

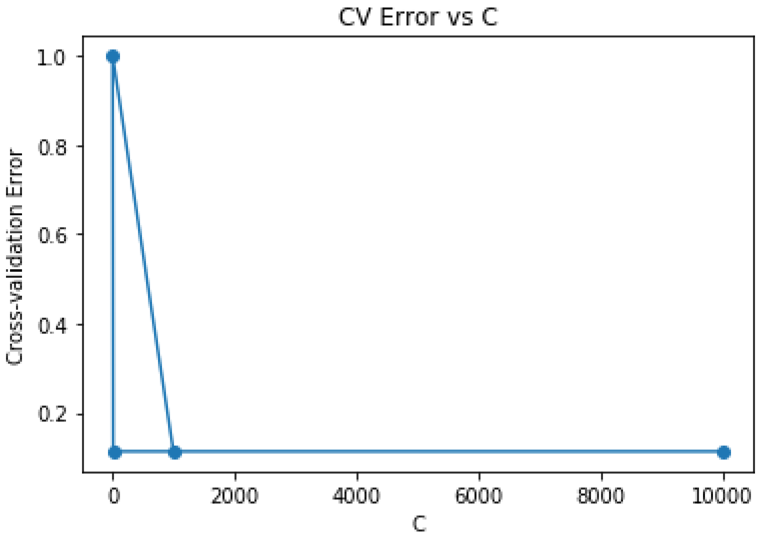
Out[135]:

```
{'C': 1000}
```

Plotting a graph between C vs CV Error

In [136]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [137]:

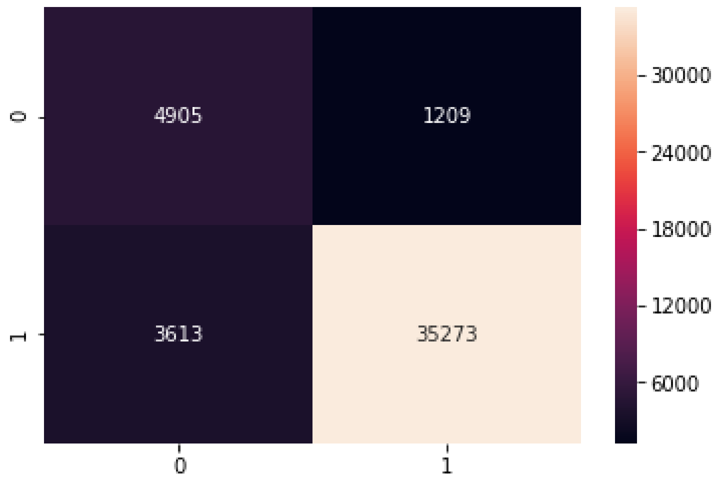
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l1',class_weight='balanced', C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_avgw2v = lr.predict(standardized_data_test)
```

7.11 Confusion Matrix

In [138]:

```
cm_avgw2v=confusion_matrix(y_test,pred_avgw2v)
print("Confusion Matrix:")
sns.heatmap(cm_bow, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [139]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_avgw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4092
 false positives are 2022
 false negatives are 3325
 true positives are 35561
```

7.12 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [140]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_avgw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_avgw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_avgw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_avgw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_avgw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_avgw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 1000.000 is
93.007624%

Test Error  Logistic Regression classifier is  6.992376%

The Test Precision of the Logistic Regression classifier for C = 1000.000 is
0.946199

The Test Recall of the Logistic Regression classifier for C = 1000.000 is 0.
914494

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.55      0.67      0.60      6114
           1       0.95      0.91      0.93     38886

   micro avg       0.88      0.88      0.88     45000
   macro avg       0.75      0.79      0.77     45000
weighted avg       0.89      0.88      0.89     45000
```

7.13 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [141]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=46
lambda=100.0 ; non-zeros=22
lambda=1000.0 ; non-zeros=2
lambda=10000.0 ; non-zeros=0
```

In [142]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[142]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [143]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= 0.03855709270957694
```

In [144]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train.data, y_train)
```

Out[144]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [145]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 4
```

```
Hence Number of Features whose coefficients changed by more than 40% is less these are
less collineare hence we cannot calculate feature importance
```

7.14 Using L2 Regularization

In [146]:

```
# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=10000, class_wei
ght='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.9384175502929749
```

In [147]:

```
model.best_params_
```

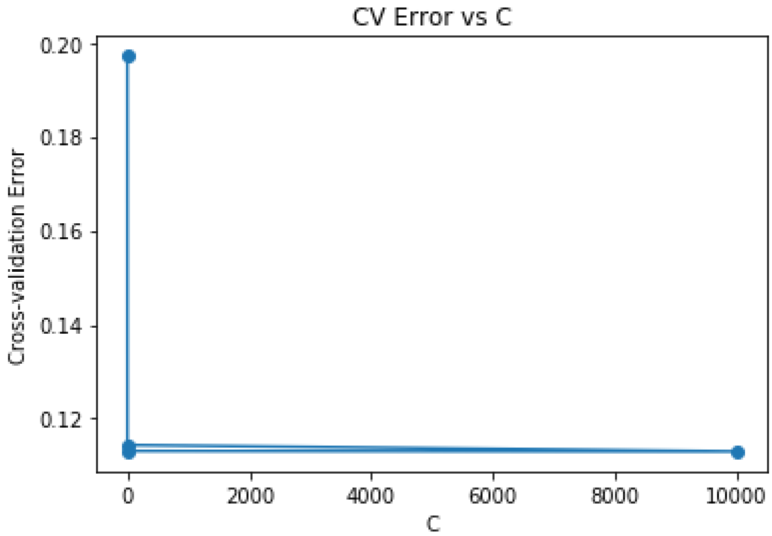
Out[147]:

```
{'C': 10000}
```

Plotting a graph between C vs CV Error

In [148]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [149]:

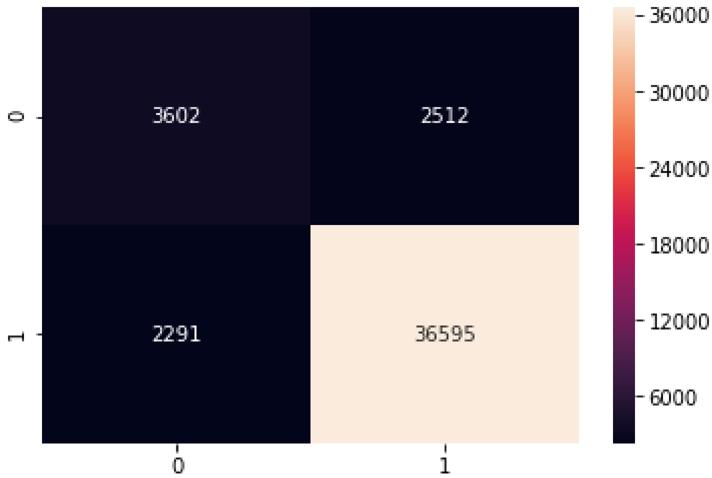
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2', class_weight='balanced',C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_avgw2v = lr.predict(standardized_data_test)
```

7.15 Confusion Matrix

In [150]:

```
cm_avgw2v=confusion_matrix(y_test,pred_avgw2v)
print("Confusion Matrix:")
sns.heatmap(cm_avgw2v, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [151]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_avgw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 3602
 false positives are 2512
 false negatives are 2291
 true positives are 36595
```

7.16 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [152]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_avgw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_avgw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_avgw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_avgw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_avgw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_avgw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 10000.000 is
93.841755%

Test Error  Logistic Regression classifier is  6.158245%

The Test Precision of the Logistic Regression classifier for C = 10000.000 i
s 0.935766

The Test Recall of the Logistic Regression classifier for C = 10000.000 is
0.941084

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.61      0.59      0.60      6114
           1       0.94      0.94      0.94     38886

   micro avg       0.89      0.89      0.89     45000
   macro avg       0.77      0.77      0.77     45000
weighted avg       0.89      0.89      0.89     45000
```

7.17 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [153]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=47
lambda=100.0 ; non-zeros=23
lambda=1000.0 ; non-zeros=2
lambda=10000.0 ; non-zeros=0
```

In [154]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[154]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [155]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= -0.05255630041709112
```

In [156]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train, y_train)
```

Out[156]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [157]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 6
```

Hence Number of Features whose coefficients changed by more than 40% is less tese are less collineare hence we cannot calculate feature importance

8.TFIDF-Word2Vec

In [158]:

```
#tf-idf weighted w2v

from sklearn.feature_extraction.text import TfidfVectorizer

tfidfw2v_vect = TfidfVectorizer()
final_counts_tfidfw2v_train= tfidfw2v_vect.fit_transform(x_train)
print(type(final_counts_tfidfw2v_train))
print(final_counts_tfidfw2v_train.shape)

final_counts_tfidfw2v_test= tfidfw2v_vect.transform(x_test)
print(type(final_counts_tfidfw2v_test))
print(final_counts_tfidfw2v_test.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(105000, 38300)
<class 'scipy.sparse.csr.csr_matrix'>
(45000, 38300)
```

In [159]:

```
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidfw2v_vect.get_feature_names(), list(tfidfw2v_vect.idf_)))


# TF-IDF weighted Word2Vec
tfidf_feat = tfidfw2v_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_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1

#Test case

tfidf_sent_vectors1 = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors1.append(sent_vec)
    row += 1
print(len(tfidf_sent_vectors))
print(len(tfidf_sent_vectors1))
```

```
105000
45000
```

8.1 Standardizing Data

In [160]:

```
# Data-preprocessing: Standardizing the data
from sklearn import preprocessing
standardized_data_train = preprocessing.normalize(tfidf_sent_vectors)
print(standardized_data_train.shape)
standardized_data_test = preprocessing.normalize(tfidf_sent_vectors1)
print(standardized_data_test.shape)
```

```
(105000, 50)
(45000, 50)
```

8.2 Applying Logistic Regression Algorithm

8.2.1 Gridsearch Cross Validation

8.2.1.1 Using L1 Regularization

In [161]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=1, class_weight
='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.868918918918919
```

In [162]:

```
model.best_params_
```

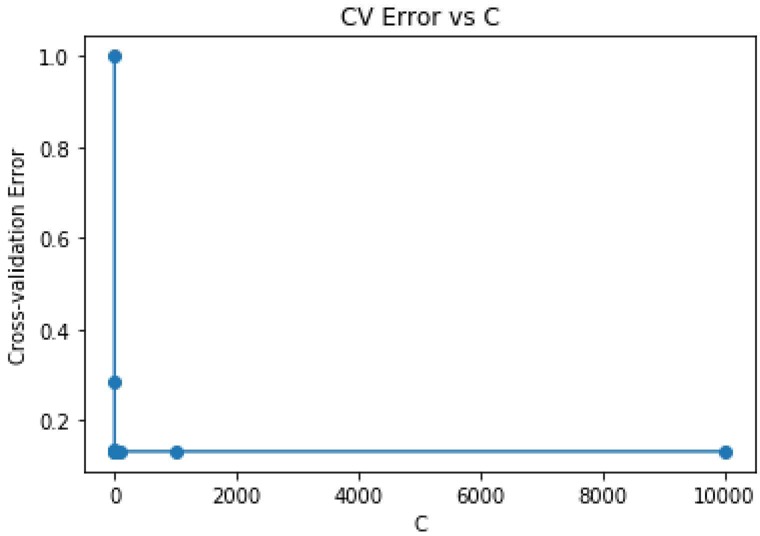
Out[162]:

```
{'C': 1}
```

Plotting a graph between C vs CV Error

In [163]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [164]:

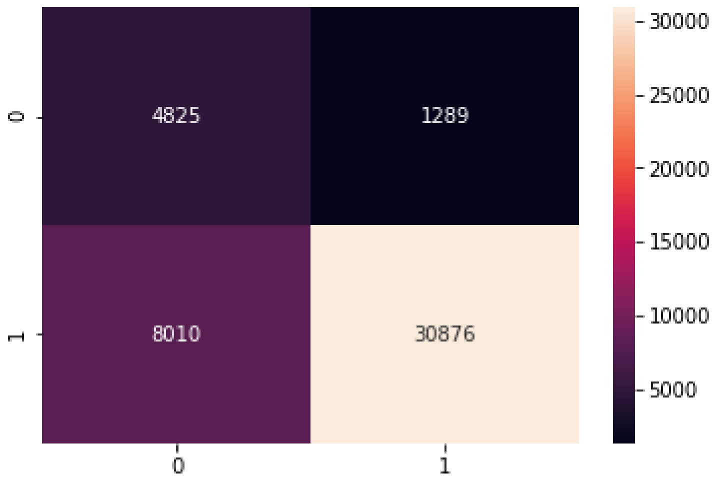
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2',class_weight='balanced', C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidfw2v = lr.predict(standardized_data_test)
```

8.3 Confusion Matrix

In [165]:

```
cm_tfidfw2v=confusion_matrix(y_test,pred_tfidfw2v)
print("Confusion Matrix:")
sns.heatmap(cm_tfidfw2v, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [166]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidfw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 4825
 false positives are 1289
 false negatives are 8010
 true positives are 30876
```

8.4 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [167]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidfw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_tfidfw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidfw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidfw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidfw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidfw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 1.000 is 86.
912218%

Test Error  Logistic Regression classifier is  13.087782%

The Test Precision of the Logistic Regression classifier for C = 1.000 is 0.
959925

The Test Recall of the Logistic Regression classifier for C = 1.000 is 0.794
013

The Test classification report of the Logistic regression classifier for C

                precision    recall  f1-score   support

           0       0.38      0.79      0.51      6114
           1       0.96      0.79      0.87     38886

   micro avg       0.79      0.79      0.79     45000
   macro avg       0.67      0.79      0.69     45000
weighted avg       0.88      0.79      0.82     45000
```

8.5 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [168]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=47
lambda=100.0 ; non-zeros=23
lambda=1000.0 ; non-zeros=1
lambda=10000.0 ; non-zeros=0
```

In [169]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[169]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [170]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= 0.12618696701812981
```

In [171]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train, y_train)
```

Out[171]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [172]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 15
```

Hence Number of Features whose coefficients changed by more than 40% is less tese are less collineare hence we cannot calculate feature importance

8.6 Using L2 Regularization

In [174]:

```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logisti

from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'C': [10**-5,10**-4,10**-3,10**-2,10**-1, 1, 10**2,10**3, 10**4]}]
model = GridSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), tuned_para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L2 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=1, class_weight
='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L2 Regularization 0.9264148747721073
```

In [175]:

```
model.best_params_
```

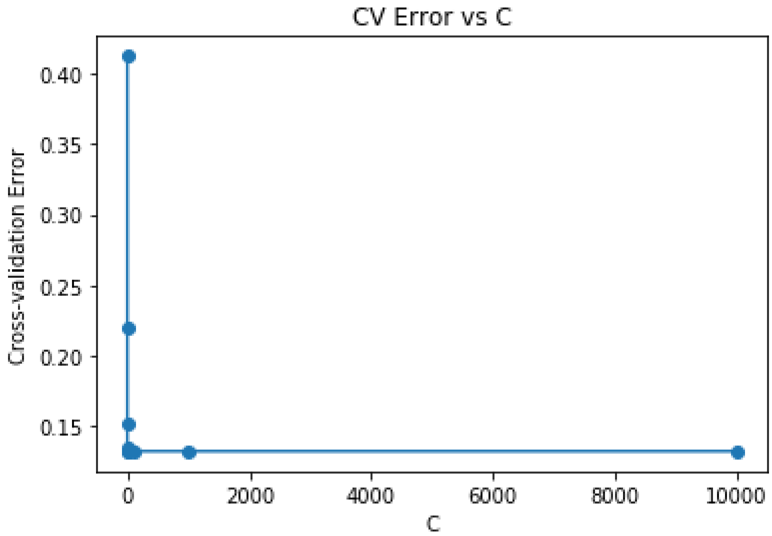
Out[175]:

```
{'C': 1}
```

Plotting a graph between C vs CV Error

In [176]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [177]:

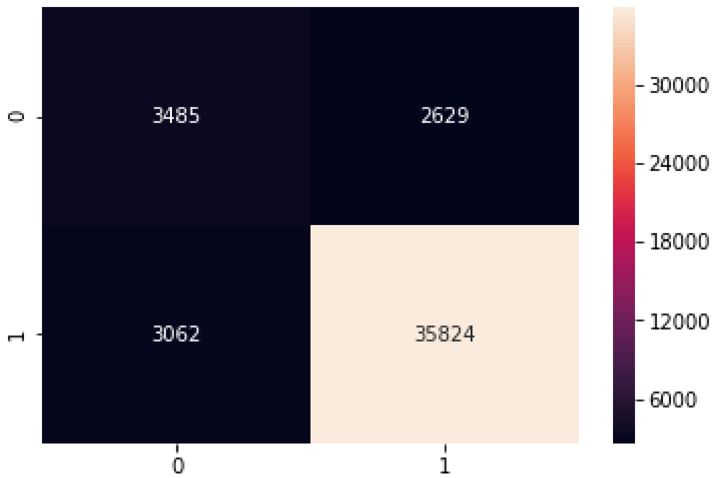
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2',class_weight='balanced', C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidfw2v = lr.predict(standardized_data_test)
```

8.7 Confusion Matrix

In [178]:

```
cm_tfidfw2v=confusion_matrix(y_test,pred_tfidfw2v)
print("Confusion Matrix:")
sns.heatmap(cm_tfidfw2v, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [179]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidfw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 3485
 false positives are 2629
 false negatives are 3062
 true positives are 35824
```

8.8 Accuracy,Error on test data,Precision,Recall,Classification Report

In [180]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidfw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (o

# Error on test data
test_error_tfidfw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidfw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidfw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidfw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidfw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 1.000 is 92.
641487%

Test Error  Logistic Regression classifier is  7.358513%

The Test Precision of the Logistic Regression classifier for C = 1.000 is 0.
931631

The Test Recall of the Logistic Regression classifier for C = 1.000 is 0.921
257

The Test classification report of the Logistic regression classifier for C

              precision    recall  f1-score   support

           0       0.53      0.57      0.55      6114
           1       0.93      0.92      0.93     38886

   micro avg       0.87      0.87      0.87     45000
   macro avg       0.73      0.75      0.74     45000
weighted avg       0.88      0.87      0.88     45000
```

8.9 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [181]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=47
lambda=100.0 ; non-zeros=24
lambda=1000.0 ; non-zeros=2
lambda=10000.0 ; non-zeros=0
```

In [182]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[182]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [183]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= 0.0615243967943100335
```

In [184]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train, y_train)
```

Out[184]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [185]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 6
```

Hence Number of Features whose coefficients changed by more than 40% is less tese are less collineare hence we cannot calculate feature importance

9. Randomized Search Cross Validation

9.1 Using L1 Regularization

In [186]:

```
# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l1',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=0.001, class_wei
ght='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.8358122910181023
```

In [187]:

```
model.best_params_
```

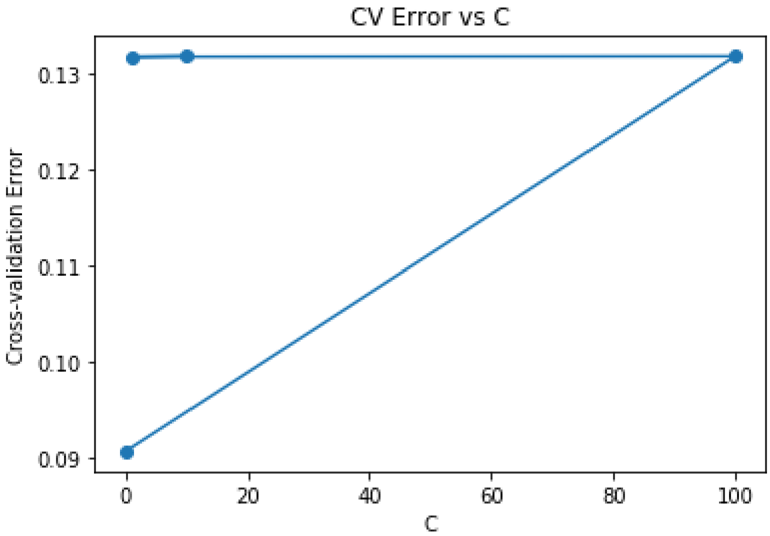
Out[187]:

```
{'C': 0.001}
```

Plotting a graph between C vs CV Error

In [188]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [189]:

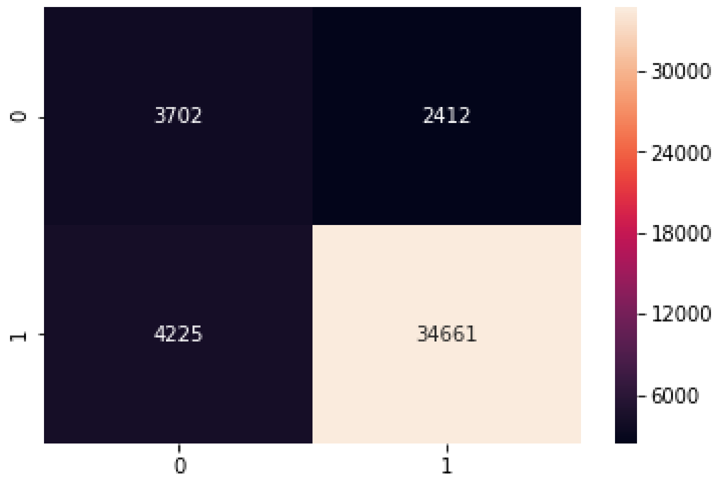
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2', class_weight='balanced',C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidfw2v = lr.predict(standardized_data_test)
```

9.2 Confusion Matrix

In [190]:

```
cm_tfidfw2v=confusion_matrix(y_test,pred_tfidfw2v)
print("Confusion Matrix:")
sns.heatmap(cm_tfidfw2v, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [191]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidfw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 3702
 false positives are 2412
 false negatives are 4225
 true positives are 34661
```

9.3 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [192]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidfw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_tfidfw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidfw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidfw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidfw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidfw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 0.001 is 91.
262392%

Test Error  Logistic Regression classifier is  8.737608%

The Test Precision of the Logistic Regression classifier for C = 0.001 is 0.
934939

The Test Recall of the Logistic Regression classifier for C = 0.001 is 0.891
349

The Test classification report of the Logistic regression classifier for C

              precision    recall  f1-score   support

           0       0.47      0.61      0.53      6114
           1       0.93      0.89      0.91     38886

   micro avg       0.85      0.85      0.85     45000
   macro avg       0.70      0.75      0.72     45000
weighted avg       0.87      0.85      0.86     45000
```

9.4 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [193]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=48
lambda=100.0 ; non-zeros=25
lambda=1000.0 ; non-zeros=2
lambda=10000.0 ; non-zeros=0
```

In [194]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[194]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [195]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= 0.2971801627396628
```

In [196]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train, y_train)
```

Out[196]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [197]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 4
```

Hence Number of Features whose coefficients changed by more than 40% is less tese are less collineare hence we cannot calculate feature importance

9.5 Using L2 Regularization

In [198]:

```
# Finding the best parameters using Random Seach CV using 10-fold Cross-Validation in Logis

from sklearn.model_selection import RandomizedSearchCV
param_distributions = {'C': [10**-4,10**-3, 10**-2,10**-1, 1, 10**1,10**2,10**3, 10**4]}
model = RandomizedSearchCV(LogisticRegression(penalty = 'l2',class_weight='balanced'), para
model.fit(standardized_data_train, y_train)
print("The optimal value of C(1/lambda) is : ",model.best_estimator_)
optimal_C = model.best_estimator_.C
print("\n Accuracy of model using L1 Regularization",model.score(standardized_data_test, y_
```

```
The optimal value of C(1/lambda) is :  LogisticRegression(C=10, class_weight
='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)

 Accuracy of model using L1 Regularization 0.9317876021143681
```

In [199]:

```
model.best_params_
```

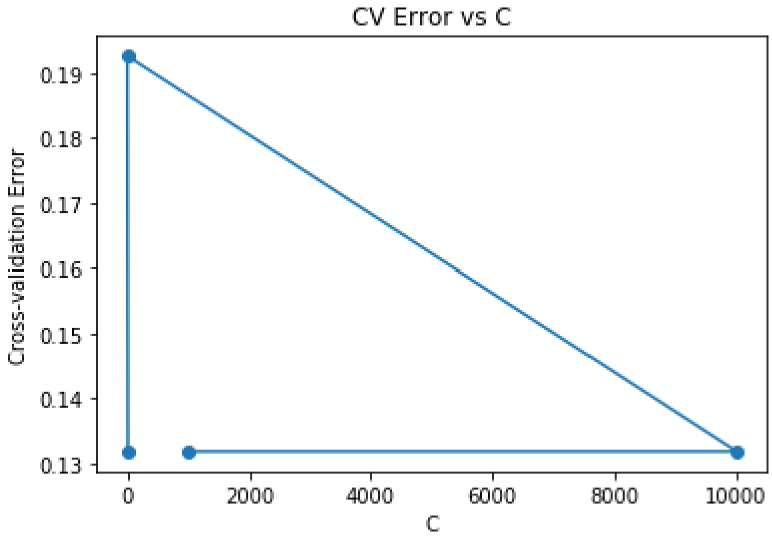
Out[199]:

```
{'C': 10}
```

Plotting a graph between C vs CV Error

In [200]:

```
score = model.cv_results_
score
plot_df = pd.DataFrame(score)
plt.plot(plot_df["param_C"], 1- plot_df["mean_test_score"], "-o")
plt.title("CV Error vs C")
plt.xlabel("C")
plt.ylabel("Cross-validation Error")
plt.show()
```



In [201]:

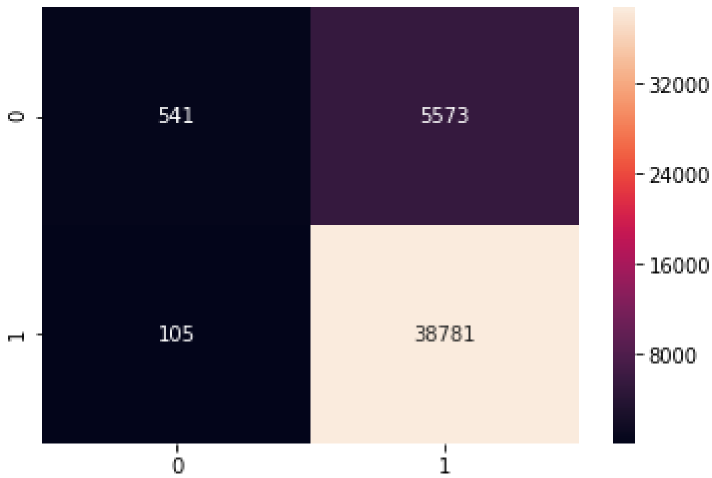
```
# Logistic Regression with Optimal value of C(1/lambda)
lr = LogisticRegression(penalty='l2',class_weight='balanced', C=optimal_C, n_jobs=-1)
lr.fit(standardized_data_train,y_train)
pred_tfidfw2v = lr.predict(standardized_data_test)
```

9.6 Confusion Matrix

In [202]:

```
cm_tfidfw2v=confusion_matrix(y_test,pred_tfidfw2v)
print("Confusion Matrix:")
sns.heatmap(cm_tfidfw2v, annot=True, fmt='d')
plt.show()
```

Confusion Matrix:



In [203]:

```
#finding out  true negative , false positive , false negative and true positve
tn, fp, fn, tp = cm_tfidfw2v.ravel()
( tp, fp, fn, tp)
print(" true negitves are {} \n false positives are {} \n false negatives are {}\n true pos
```

```
 true negitves are 541
 false positives are 5573
 false negatives are 105
 true positives are 38781
```

9.7 Calculating Accuracy,Error on test data,Precision,Recall,Classification Report

In [204]:

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report


# evaluating accuracy
f1score = f1_score(y_test, pred_tfidfw2v) * 100
print('\nThe Test Accuracy of the Logistic Regression classifier for C = %.3f is %f%%' % (c

# Error on test data
test_error_tfidfw2v = 100-f1score
print("\nTest Error  Logistic Regression classifier is  %f%%" % (test_error_tfidfw2v))

# evaluating precision
precision_score = precision_score(y_test, pred_tfidfw2v)
print('\nThe Test Precision of the Logistic Regression classifier for C = %.3f is %f' % (op

# evaluating recall
recall_score = recall_score(y_test, pred_tfidfw2v)
print('\nThe Test Recall of the Logistic Regression classifier for C = %.3f is %f' % (optim

# evaluating Classification report
classification_report = classification_report(y_test, pred_tfidfw2v)
print('\nThe Test classification report of the Logistic regression classifier for C \n\n ',
```

```
The Test Accuracy of the Logistic Regression classifier for C = 10.000 is 9
3.178760%

Test Error  Logistic Regression classifier is  6.821240%

The Test Precision of the Logistic Regression classifier for C = 10.000 is
0.874352

The Test Recall of the Logistic Regression classifier for C = 10.000 is 0.99
7300

The Test classification report of the Logistic regression classifier for C

              precision    recall  f1-score   support

           0       0.84      0.09      0.16      6114
           1       0.87      1.00      0.93     38886

   micro avg       0.87      0.87      0.87     45000
   macro avg       0.86      0.54      0.55     45000
weighted avg       0.87      0.87      0.83     45000
```

9.8 Perturbation Test

Pertubation test means adding noise to one of the data point and comparing the difference b/w change in previous weights and New

weights to find of collinearity.

In [205]:

```
lambd = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
for i in lambd[::-1]:
    lrr = LogisticRegression(penalty = 'l1', C = i)
    lrr.fit(standardized_data_train, y_train)
    print("lambda="+str(1/i)+" ; non-zeros="+str(np.count_nonzero(lrr.coef_)))
```

```
lambda=0.0001 ; non-zeros=50
lambda=0.001 ; non-zeros=50
lambda=0.01 ; non-zeros=50
lambda=0.1 ; non-zeros=50
lambda=1.0 ; non-zeros=50
lambda=10.0 ; non-zeros=45
lambda=100.0 ; non-zeros=25
lambda=1000.0 ; non-zeros=3
lambda=10000.0 ; non-zeros=0
```

In [206]:

```
lr = LogisticRegression(penalty='l2', C=0.01)
lr.fit(standardized_data_train, y_train)
```

Out[206]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [207]:

```
# Applying perturbation and checking if the coefficients differ too much
# Will not add noise to zeros

noise = np.random.normal(0 , 0.1 , 1)
print("Noise= "+str(noise[0]))
standardized_data_train.data = standardized_data_train.data + noise[0]
```

```
Noise= -0.14592227151454049
```

In [208]:

```
# Fitting the new model on the transformed data

lr2 = LogisticRegression(penalty='l2', C=0.01)
lr2.fit(standardized_data_train, y_train)
```

Out[208]:

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [209]:

```
# Calculating the percentage change between the old and new coefficients
count=0
for i in range(standardized_data_train.shape[1]):
    delta = abs((((lr.coef_[0][i] - lr2.coef_[0][i]) * 100))/ lr.coef_[0][i])
    if delta>40:
        count+=1

print("Number of features whose coefficients changed by more than 40% =",count)
```

```
Number of features whose coefficients changed by more than 40% = 0
```

Hence Number of Features whose coefficients changed by more than 40% is less tese are less collineare hence we cannot calculate feature importance

10.Conclusion

Model performance table

Model	Hyper parameter(c) with Random search	Regularizer	Test Error	Accuracy
Logistic Regression with Bow	100	L2	6.3979	93.602
Logistic Regression with Tfidf	100	L2	6.614209	93.385791
Logistic Regression with Avgw2v	1000	L2	6.1558245	93.841755
Logistic Regression with Tfidfw2v	10	L2	6.821240	93.17876

Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or

more nominal, ordinal, interval or ratio-level independent variables.

Logistic regression is used to obtain odds ratio in the presence of more than one explanatory variable.

Logistic regression does not assume a linear relationship between the dependent variable and the independent variables, but it

does assume linear relationship between the logic of the explanatory variables and the response.

Independent variables can be even the power terms or some other nonlinear transformations of the original independent variables

The dependent variable does NOT need to be normally distributed, but it typically assumes a distribution from an exponential

family (e.g. binomial, Poisson, multinomial, normal,...); binary logistic regression assume binomial distribution of the

response

The goal of logistic regression is to correctly predict the category of outcome for individual cases using the most parsimonious model. To accomplish this goal, a model is created that includes all predictor variables that are useful in

predicting the response variable.

Assumptions of Logistic Regression

1)logistic regression does not require a linear relationship between the dependent and independent variables.

2)Second, the error terms (residuals) do not need to be normally distributed.

3)Third, homoscedasticity is not required.

4)Finally, the dependent variable in logistic regression is not measured on an interval or ratio scale.

Steps Involved:-

1)Connecting SQL file

2)Data Preprocessing(Already i had done preprocessing no need to do again)

3)Sorting the data based on time

4)Mapping the data (i had changed my partition positive=1 and Negative=0)

5)Taking 1st 150K Rows (Due to low Ram)

6)Spliting data into train and test based on time (70:30)

7)Techniques For Vectorization Bow,TF-IDF,Avgword2vec,Tfidfword2vec

8)Standardizing Data and Applying Logistic Regression Algorithm

9)I calculated Accuracy,Error on Test Data, Confusion Matrix, Classification Report,Precision Score,Recall Score, F1-Score,Feature Importance,Log-Probabilities.

10)Performing perturbation test

11)Gettting Important Features

12)I Designed Model Performance Table

13)Conclusion

In [ ]:

```

```