OBJECTIVE :

1. APPLYING LOGISTIC REGRESSION WITH TFIDF WORD2VEC VECTORIZATION
   - PERFORMING PERTUBATION TEST TO CHECK WHETHER OUR DATA FEATURES ARE COLLINER OR NOT AND PLOTTING THE RESULT
   - FINDING THE BEST HYPERPARAMETER USING GRIDSEARCHCV WITH TRAIN DATA AND CROSS-VALIDATION DATA BY PLOTTING THE RESLUTS OF VAROIUS TRAIN DATA AND CROSS VALIDATION DATA
   - USING THE APROPRIATE VALUE OF HYPERPARAMETER ,TESTING ACCURACY ON TEST DATA USING F1-SCORE
   - PLOTTING THE CONFUSION MATRIX TO GET THE PRECISOIN ,RECALL VALUE WITH HELP OF HEATMAP

In [0]:
```
from sklearn.model_selection import train_test_split         #importin
g the necessary libraries
from sklearn.model_selection import RandomizedSearchCV
from sklearn.datasets import *
from sklearn import naive_bayes
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import pandas as pd
from sklearn import *
from gensim.models import Word2Vec
import warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm
```

In [0]:
```
from google.colab import drive
drive.mount('/content/gdrive')#geeting the content from the google driv
e
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remoun
```

t, call drive.mount("/content/gdrive", force_remount=True).

In [0]:
```
final_processed_data=pd.read_csv("gdrive/My Drive/final_new_data.csv")#
loading the preprocessed data  with 100k points into dataframe
```

In [0]:
```
# getting the counts of 0 and 1 in "SCORE" column to know whether it is
 unbalanced data or not
count_of_1=0
count_of_0=0
for i in final_processed_data['Score']:
   if i==1:
    count_of_1+=1
   else:
    count_of_0+=1
print(count_of_1)
print(count_of_0)
#it is an imbalanced dataset
```

```
88521
11479
```

In [0]:
```
#spliiting the data into train and test data
x_train,x_test,y_train,y_test=model_selection.train_test_split(final_pr
ocessed_data['CleanedText'].values,final_processed_data['Score'].values
,test_size=0.2,shuffle=False)
```

In [0]:
```
# Training my own Word2Vec model using your own text corpus
list_of_sent=[]
for sent in x_train:
 list_of_sent.append(sent.split())#splitting of sentences into words AN
D appending them to list
print(x_train[0])
print("*****************************************************************
*")
print(list_of_sent[0])
word_to_vector=Word2Vec(list_of_sent,min_count=5,size=100,workers=2)#co
nstructing my our word to vector
w_t_c_words=list(word_to_vector.wv.vocab)
```

```
print("**********************************************************************")
print("sample words ", w_t_c_words[0:20])
```

```
witti littl book make son laugh loud recit car drive along alway sing r
efrain hes learn whale india droop love new word book introduc silli cl
assic book will bet son still abl recit memori colleg
*****************************************************************
['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'ca
r', 'drive', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn', 'whal
e', 'india', 'droop', 'love', 'new', 'word', 'book', 'introduc', 'sill
i', 'classic', 'book', 'will', 'bet', 'son', 'still', 'abl', 'recit',
'memori', 'colleg']
**********************************************************************
sample words  ['witti', 'littl', 'book', 'make', 'son', 'laugh', 'lou
d', 'car', 'drive', 'along', 'alway', 'sing', 'refrain', 'hes', 'lear
n', 'india', 'droop', 'love', 'new', 'word']
```

In [0]:

```
###### NOW STARTING TFIDF WORD TO VEC FOR TRAIN DATA##################
################################################
#NOW STARTING TF-IDF WEIGHTED WORD-TO-VEC
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(x_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
train_tfidf_sent_vectors =[]# the tfidf-w2v for each sentence/review is
 stored in this list

for sent in tqdm(list_of_sent): # for each review/sentence
  sent_vec = np.zeros(100) # as word vectors are of zero length
  weight_sum =0; # num of words with a valid vector in the sentence/rev
iew
  for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     tf_idf = dictionary[word]*(sent.count(word)/len(sent))# dictionary
[word] = idf value of word in whole courpus
     sent_vec += (vec * tf_idf)# sent.count(word) = tf valeus of word i
```

```
n this review
        weight_sum += tf_idf
    if weight_sum != 0:
     sent_vec /= weight_sum
     train_tfidf_sent_vectors.append(sent_vec)
```

```
100%|██████████| 80000/80000 [06:33<00:00, 203.33it/s]
```

In [0]:

```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_train_data=StandardScaler( with_mean=False).fit_transform(train_tfidf
_sent_vectors)
print(x_train_data.shape)
```

```
(80000, 100)
```

In [0]:

```
list_of_sent=[]
for sent in x_test:
 list_of_sent.append(sent.split())#splitting of sentences into words AN
D appending them to list
print(x_test[0])
print("****************************************************************
*")
print(list_of_sent[0])
print('****************************************************************
***')
```

```
hard find item dont buy mani either came stale got way quick classic no
netheless
*****************************************************************
['hard', 'find', 'item', 'dont', 'buy', 'mani', 'either', 'came', 'stal
e', 'got', 'way', 'quick', 'classic', 'nonetheless']
*******************************************************************
```

In [0]:

```
###### NOW STARTING TFIDF WORD TO VEC FOR TEST DATA###################
#############################################
#NOW STARTING TF-IDF WEIGHTED WORD-TO-VEC
model = TfidfVectorizer()
model.fit_transform(x_train)
```

```
model.transform(x_test)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
test_tfidf_sent_vectors =[]# the tfidf-w2v for each sentence/review is
 stored in this list

for sent in tqdm(list_of_sent): # for each review/sentence
  sent_vec = np.zeros(100) # as word vectors are of zero length
  weight_sum =0; # num of words with a valid vector in the sentence/rev
iew
  for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     tf_idf = dictionary[word]*(sent.count(word)/len(sent))# dictionary
[word] = idf value of word in whole courpus
     sent_vec += (vec * tf_idf)# sent.count(word) = tf valeus of word i
n this review
     weight_sum += tf_idf
  if weight_sum != 0:
   sent_vec /= weight_sum
   test_tfidf_sent_vectors.append(sent_vec)
```

```
100%|██████████| 20000/20000 [01:51<00:00, 179.80it/s]
```

In [0]:

```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_test_data=StandardScaler( with_mean=False).fit_transform(test_tfidf_s
ent_vectors)
print(x_test_data.shape)
```

```
(20000, 100)
```

In [0]:

```
#using time series split method for cross-validation score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
from sklearn.linear_model import LogisticRegression
from scipy.stats import uniform
```

```
parameter_range=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10
**4]#range of hyperparameter
```

In [0]:

```
lr=LogisticRegression(penalty='l2',class_weight={1:.5,0:.5},n_jobs=-1)#
building logistic regression model
tuned_parameters=[{'C':parameter_range}]
```

In [0]:

```
#applying the model of logistic regression and using gridsearchcv to fi
nd the best hyper parameter
%time
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(lr, tuned_parameters, scoring = 'f1', cv=tscv,n_jo
bs=-1)#building the gridsearchcv model
model.fit(x_train_data, y_train)#fiitting the training data

print(model.best_estimator_)#printing the best_estimator
print(model.score(x_test_data, y_test))#predicting  f1 score on test da
ta
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 9.06 µs
LogisticRegression(C=0.1, class_weight={1: 0.5, 0: 0.5}, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=Non
e,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
0.9398098682227883
```

In [0]:

```
lr=LogisticRegression(C=0.1,penalty='l2',class_weight={1:.5,0:.5},n_job
s=-1)#building model for getting wieght vector
lr.fit(x_train_data,y_train)#fitting the training data
z=lr.decision_function(x_train_data)#checking the signed distance of a
 point from hyperplane
print(z)#printing the signed distance
```

```
[3.3817815  3.72982275 4.43153667 ... 2.21317907 2.13869536 3.92686093]
```

In [0]:

```
wieght_vector=lr.coef_#getting the weight vector
```

```
print(wieght_vector.shape)#wieght vector shape
print(wieght_vector[:10])
```

```
(1, 100)
[[-0.16501958 -0.21457623  0.13722374 -0.11166733  0.22102247 -0.012869
58
  -0.06977602  0.23810098  0.1583446  -0.01580348 -0.56143422  0.044593
77
  -0.27458457 -0.02144127  0.23681264  0.3060031   0.13795873 -0.069243
6
   0.05188021  0.02388875 -0.09548089  0.32181222 -0.12674304  0.051333
34
   0.33665299  0.01795062 -0.21469577  0.36660306 -0.02321149  0.405117
09
  -0.35227165  0.26881642 -0.57090952  0.22466264  0.36935362  0.121718
55
  -0.20692414 -0.22261148 -0.23196746  0.11794993  0.0297931  -0.045191
93
  -0.62683822  0.30744293  0.70980847 -0.26242057  0.25476866  0.133513
17
  -0.11249602  0.59678395  0.07131134 -0.07589831  0.00088051 -0.056469
99
  -0.11040431 -0.05918636 -0.25525033 -0.20574963 -0.21631142  0.335051
51
   0.10419019 -0.35587076 -0.0598645  -0.20009404  0.19458033  0.117281
63
   0.17198177 -0.60256641  0.1729033  -0.10077599  0.17313286  0.090180
91
  -0.31024814 -0.25297867 -0.1085571  -0.02121957 -0.31238821 -0.139704
66
   0.06517518  0.03902697  0.05164876  0.2659251   0.2649679   0.067296
49
  -0.24576033 -0.10450755  0.06208229  0.01022223 -0.03161103 -0.051281
43
  -0.04471847  0.00125878 -0.10737445  0.20960195 -0.11948915  0.015933
72
   0.05709511 -0.0287014   0.11602911 -0.2704184 ]]
```

PERTUBATION TEST :

AIM: TO CHECK FOR MULTI COLLINEARITY OF FEATURES STEPS

1. GETTING THE WIEGHT VECTOR FROM MODEL AND SAVING IT</br>
2. ADDING NOISE TO THE TRAINING DATA TO GET NEW TRAINING DATA</br>
3. FITTING THE MODEL AGAIN ON NEW DATA</br>
4. GETTING THE WIEGHT VECTOR FROM THIS MODEL</br> 5.ADDING SMALL VALUE TO WEIGHT VECTOR OF BOTH TRAINNG DATA TO REMOVE ANY ERROR
5. FINDING THE PERCENTAGE CHANGE VECTOR
6. GEETING HOW MANY GEATURE HAS CHANGED USING SOME THRESHOLD VALUE(HERE TAKING IT AS 100)
7. PLOTTING THE QUANTILES WITH THIER PERCENTAGE WIGHT VALUE TO CHECK IF COLLINEARITY EXITS OR NOT</i></br>

RESULT : TO KNOW WHETHER FEATURES ARE MULTICOLLINEAR OR NOT AND TO KNOW WHETHER MODEL IS RELIABLE OR NOT

In [0]:
```
#here,we are adding noise to the data
from scipy.stats import norm
noise=norm.rvs(size=1)#noise
x_train_data.data+=noise#adding noise
```

In [0]:
```
print('shape of our new train data  after adding noise is : ',x_train_d
ata.shape)#printing shape of new training data
```

```
shape of our new train data  after adding noise is :  (80000, 100)
```

In [0]:
```
#uilding the model using timeSeriesSplit
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10) # 10 spilts cross validation
from sklearn.linear_model import LogisticRegression
from scipy.stats import uniform
data=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]#value
 range of hyper parameter for grid searchcv
lr=LogisticRegression(penalty='l2',class_weight={1:.5,0:.5},n_jobs=-1)#
```

```
building the model
tuned_parameters=[{'C':data}]
```

In [0]:
```
%time
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(lr, tuned_parameters, scoring = 'f1', cv=tscv,n_jo
bs=-1)#building the gridsearchcv model
model.fit(x_train_data, y_train)#fiitting the training data

print('best estimator of our new data is: ',model.best_estimator_)#prin
ting the best_estimator
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 9.54 µs
best estimator of our new data is:  LogisticRegression(C=0.1, class_wei
ght={1: 0.5, 0: 0.5}, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=Non
e,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [0]:
```
# again building the model for finding the wieght vector of the words f
rom model
lr=LogisticRegression(C=10,penalty='l2',class_weight={1:.5,0:.5},n_jobs
=-1)#building the logistic regression model
lr.fit(x_train_data,y_train)#fiting the training model
new_wieght_vector=lr.coef_
print(new_wieght_vector.shape)#printing shape of wieght vector
```

```
(1, 100)
```

In [0]:
```
percent_change_vec=np.ones((1,100))#generating the percent_change_vetor
 to store the percentage change  values for each word
```

In [0]:
```
wieght_vector=wieght_vector+10**-6 #adding some values to wieght vector
 to avoid error while division

new_wieght_vector=new_wieght_vector+10**-6 #adding some values to wiegh
```

```
t vector to avoid error while division

percent_change_vec=abs((wieght_vector-new_wieght_vector)/wieght_vector)
*100#calculating the percentage change in the vector
```

In [0]:
```
x=abs((wieght_vector[0][2]-new_wieght_vector[0][2])/wieght_vector[0][2
])#just checking randomly that every value is positve in percent_change
_vector
print(x)
```

```
0.00283117933505747
```

In [0]:
```
print('shape of percent change wieght vector is', percent_change_vec.sh
ape)#printing shape of percent_change_vector
```

```
shape of percent change wieght vector is (1, 100)
```

In [0]:
```
per_change_df=pd.DataFrame(percent_change_vec.T,columns=['CHANGE'])#bui
lding a dataframe from wight vector
```

In [0]:
```
per_change_df.head()#getting first 5 values
```

Out[0]:

	CHANGE
0	8.252026
1	5.562601
2	0.283118
3	6.209997
4	1.448420

In [0]:
```
sorted_Df=per_change_df.sort_values('CHANGE',ascending=True,axis=0)#sor
ting the dataframe to calculate the quantiles values
sorted_Df.describe()#describe function
```

Out[0]:

	CHANGE
count	100.000000
mean	21.766990
std	96.913322
min	0.041925
25%	2.558892
50%	5.629969
75%	8.782768
max	947.544153

In [0]:

```
quantiles=list( i/100 for i in range(0,101,5))#building the list of quantiles value
for i in quantiles:
  print('sorted_Data {:.2f}th quantiles is {:7.3f}'.format(i,sorted_Df['CHANGE'].quantile(i)))#printing the quantiles and thier coreesponding values
```

```
sorted_Data 0.00th quantiles is   0.042
sorted_Data 0.05th quantiles is   0.446
sorted_Data 0.10th quantiles is   1.014
sorted_Data 0.15th quantiles is   1.443
sorted_Data 0.20th quantiles is   1.793
sorted_Data 0.25th quantiles is   2.559
sorted_Data 0.30th quantiles is   3.411
sorted_Data 0.35th quantiles is   3.770
sorted_Data 0.40th quantiles is   4.985
sorted_Data 0.45th quantiles is   5.261
sorted_Data 0.50th quantiles is   5.630
```

```
sorted_Data 0.55th quantiles is   6.221
sorted_Data 0.60th quantiles is   6.526
sorted_Data 0.65th quantiles is   7.206
sorted_Data 0.70th quantiles is   8.266
sorted_Data 0.75th quantiles is   8.783
sorted_Data 0.80th quantiles is   9.965
sorted_Data 0.85th quantiles is  14.053
sorted_Data 0.90th quantiles is  27.903
sorted_Data 0.95th quantiles is  57.288
sorted_Data 1.00th quantiles is 947.544
```

In [0]:

```python
quantiles=list( i/100 for i in range(95,101,1))#printing the last percentiles values because this region is showing abrupt change
percent_change_list=[]#empty percent_change
for i in quantiles:
  print('sorted_Data {:.2f}th quantiles is {:7.3f}'.format(i,sorted_Df['CHANGE'].quantile(i)))
  percent_change_list.append(sorted_Df['CHANGE'].quantile(i))#building the list
```

```
sorted_Data 0.95th quantiles is  57.288
sorted_Data 0.96th quantiles is  64.478
sorted_Data 0.97th quantiles is  79.859
sorted_Data 0.98th quantiles is 159.585
sorted_Data 0.99th quantiles is 174.877
sorted_Data 1.00th quantiles is 947.544
```

In [0]:

```python
print(percent_change_list)
my_formatted_list = [ '%.2f' % elem for elem in percent_change_list ]#formatted list with string values in it
my_formatted_list=[float(i) for i in my_formatted_list]#formatted list with flaot values in it
print(my_formatted_list)#printing formatted list
print(quantiles)#printing quantiles
```

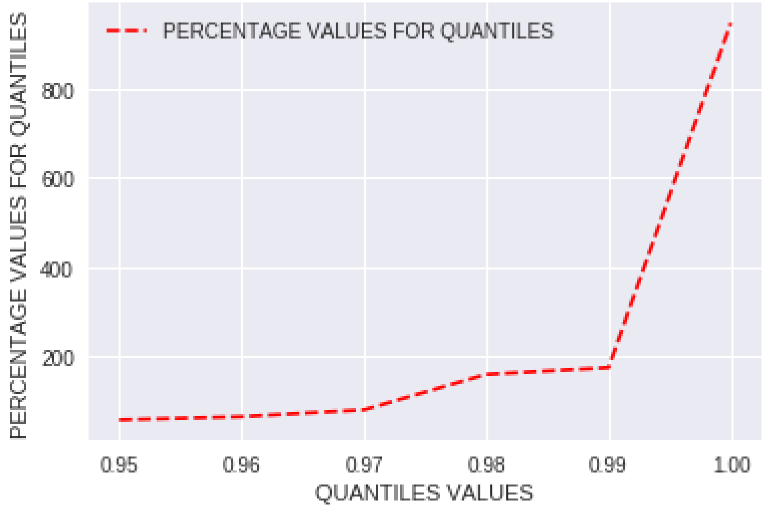
```
[57.28808961407178, 64.47832543785151, 79.85944211906845, 159.58493481949412, 174.8767261023266, 947.5441530541704]
[57.29, 64.48, 79.86, 159.58, 174.88, 947.54]
[0.95, 0.96, 0.97, 0.98, 0.99, 1.0]
```

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.show()
plt.xlabel('QUANTILES VALUES')
plt.ylabel('PERCENTAGE VALUES FOR QUANTILES')
plt.plot(quantiles,my_formatted_list,'r--' ,label='PERCENTAGE VALUES FO
R QUANTILES')
plt.legend(loc='best')
```

Out[0]:

```
<matplotlib.legend.Legend at 0x7fa1a6a6d978>
```



FROM THE ABOVE VISUALIZATION , MAIN POINTS ARE:.

1. THAT ONLY 1% OF FEATURES GOT AFFECTED AFTER ADDING NOISE TO THE DATA.
2. VERY LESS COLLINEARITY OF DATA IS PRESENT ,BECAUSE MOST OF THE WEIGHT VECTORS VALUES REMAINS SAME 3.THERFORE, OUR MODEL IS RELIABLE AND WE CAN PROCEED FURTHER TO CHECK ACCURACY ON TEST DATA

CALCULATING THE BEST HYPERPARAMETER ON TRAIN DATA AND CALCULATING THE ACCURACY USING F1-SCORE AND PLOTTING IT

In [0]:
```
#using time series split method for cross-validation score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
from sklearn.linear_model import LogisticRegression
from scipy.stats import uniform
data=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]#range
 of hyperparameter
```

In [0]:
```
lr=LogisticRegression(penalty='l2',class_weight={1:.5,0:.5},n_jobs=-1)#
building logistic regression model
tuned_parameters=[{'C':data}]
```

In [0]:
```
#applying the model of logistic regression and using gridsearchcv to fi
nd the best hyper parameter
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(lr, tuned_parameters, scoring = 'f1', cv=tscv,n_jo
bs=-1)#building the gridsearchcv model
model.fit(x_train_data, y_train)#fiitting the training data
```

Out[0]:
```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight={1: 0.5, 0: 0.
5}, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=Non
e,
          solver='warn', tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid=[{'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 1
0000]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [0]:
```
print(model.best_estimator_)
```

```
LogisticRegression(C=0.1, class_weight={1: 0.5, 0: 0.5}, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=-1, penalty='l2', random_state=Non
e,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [0]:
```
results=pd.DataFrame(model.cv_results_)# getting varoius cv_scores and
 train_scores various values of alpha given as parameter and storing it
 in a dataframe
results#printing the dataframe
```

Out[0]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parai
0	1.996376	0.014547	0.941283	0.941648	0.0001	{'C': 0.000
1	3.171960	0.014771	0.943710	0.945046	0.001	{'C': 0.001
2	4.698934	0.015680	0.946472	0.950018	0.01	{'C': 0.01}
3	7.046733	0.015911	0.946884	0.950443	0.1	{'C': 0.1}
4	7.538299	0.014789	0.946434	0.950578	1	{'C':
5	7.361695	0.012899	0.946418	0.950613	10	{'C': 10}
6	7.598320	0.013064	0.946426	0.950609	100	{'C': 100}
7	7.546490	0.016040	0.946426	0.950609	1000	{'C': 1000

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parai
8	7.403473	0.012514	0.946418	0.950605	10000	{'C': 1000(

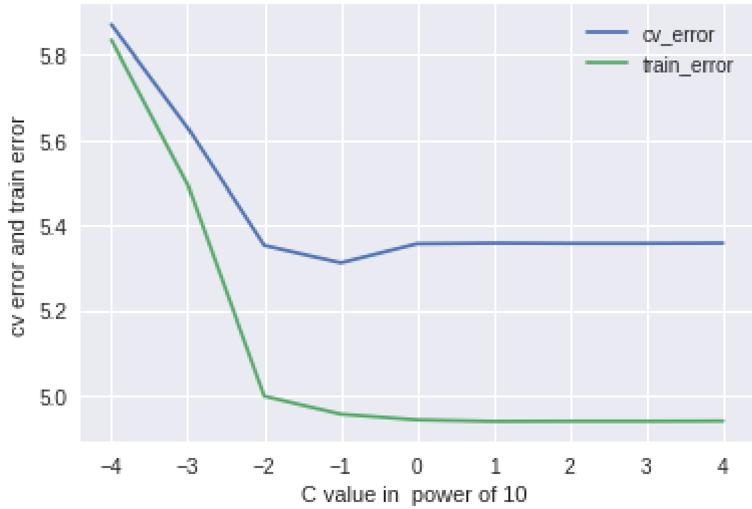
9 rows × 31 columns

In [0]:
```
%matplotlib inline
import matplotlib.pyplot as plt


mean_test_score=list(results['mean_test_score'])#taking mean_test_score
 values of various alpha into a list
mean_train_score=list(results['mean_train_score'])#taking mean_train_sc
ore values of varoius alpha into a list
cv_error_list=[]
train_error_list=[]
for i  in mean_test_score:
   i=1-i
   i=i*100
   cv_error_list.append(i)#appending the list with cv_error
for i  in mean_train_score:
   i=1-i
   i=i*100
   train_error_list.append(i)#appending  the list with train_error

print(cv_error_list)
C_values_in_10_power=[-4,-3,-2,-1,0,1,2,3,4]#list of alpha values in po
wer of 10
plt.plot(C_values_in_10_power,cv_error_list,label='cv_error')#plotting
 alpha with cv_error
plt.plot(C_values_in_10_power,train_error_list,label='train_error')#plo
tting aplhawith train_error
plt.xlabel('C value in  power of 10 ')
plt.ylabel('cv error and train error')
plt.legend(loc='best')
```

```
[5.8716940632852825, 5.629002786920556, 5.352804983977344, 5.3116281652
```

```
48572, 5.35657351955895, 5.358155325677227, 5.3574406314689575, 5.35744
06314689575, 5.358235102558151]
```

Out[0]:
```
<matplotlib.legend.Legend at 0x7fa1a630ee48>
```



From here,the best hyperparameter value is c=10 or alpha=0.1

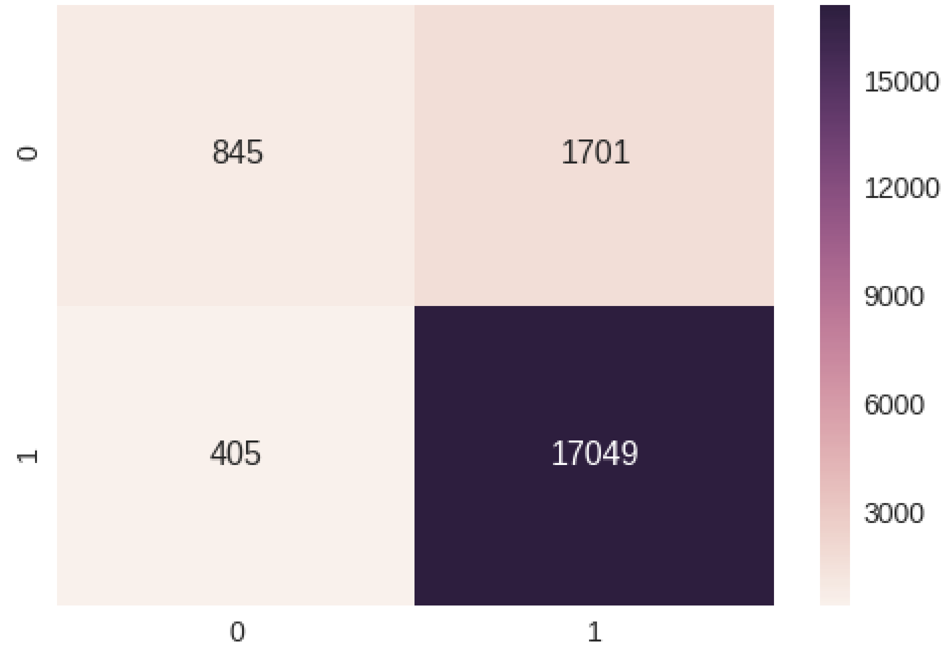
USING BEST HYPERPARAMETER VALUE ON TEST DATA AND PLOTTING THE CONFUSION MATRIX WITH HEATMAP

In [0]:
```
#Testing Accuracy on Test data
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoius metrics from sklearn
lr=LogisticRegression(C=10,penalty='l2',class_weight={1:.5,0:.5},n_jobs
=-1)#building logistic regression model#building the model
lr.fit(x_test_data,y_test)
y_pred = lr.predict(x_test_data)
```

```
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*1
00))#printing accuracy
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
#printing precision score
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred))) #prin
ting recall
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2
)) #generating the heatmap for confusion matrix
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 89.470%
Precision on test set: 0.909
Recall on test set: 0.977
F1-Score on test set: 0.942
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa1a69a4208>



FROM THE ABOVE OBSERVATIONS ,IT IS FOUND THAT THE BEST HYPERPARAMETER IS FOUND AS APLHA=0.1 AND IT IS ALSO HAVING HIGH PRECISION,RECALL VALUE ON TEST DATA

In [0]:
```
#TFIDF WIEGHTED WORD2VEC VECTORIZATION IS COMPLETED FOR LOGISTIC REGRES
SION
```

SUMMARIZING THE VAROIUS VECTORIZATIONS FOR F1-

SCORE,RECALL,PRECISION ,ACCURACY ON TEST DATA WITH HELP OF TABLE

In [3]:
```
from tabulate import tabulate
```

In [4]:
```
table = [["BOW",1000,90,99,95],["TF-IDF",1000,90,99,94], ["AVG_WORD_2_VEC",0.1,91,97,94],["TFIDF_AVG_WORD_2_VEC",0.1,91,98,94]]
print (tabulate(table))
```

```
--------------------  ------  --  --  --
BOW                   1000    90  99  95
TF-IDF                1000    90  99  94
AVG_WORD_2_VEC           0.1  91  97  94
TFIDF_AVG_WORD_2_VEC     0.1  91  98  94
--------------------  ------  --  --  --
```

In [5]:
```
headers=['VECTORIZATION','ALPHA','PRECISION','RECALL','F1_SCORE']
print (tabulate(table, headers, tablefmt="fancy_grid"))
```

VECTORIZATION	ALPHA	PRECISION	RECALL	F1_SCORE
BOW	1000	90	99	95
TF-IDF	1000	90	99	94
AVG_WORD_2_VEC	0.1	91	97	94

TFIDF_AVG_WORD_2_VEC	0.1	91	98	94