OBJECTIVE :

1. APPLYING LOGISTIC REGRESSION WITH TFIDF 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
   - PRINTING THE TOP 20 FEATURES FOR BOTH POSITIVE AND NEGATIVE WORDS #

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 *
import warnings
warnings.filterwarnings("ignore")
```

In [43]:
```
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 [45]:

```
# 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 [47]:

```
vectorizer=TfidfVectorizer(min_df=2)#building the vertorizer with word
 counts equal and more then 2
train_bow=vectorizer.fit_transform(x_train)#fitting the model on traini
ng data
print(train_bow.shape)
```

```
(80000, 17204)
```

In [48]:

```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
```

```
x_train_data=StandardScaler( with_mean=False).fit_transform(train_bow)
print(x_train_data.shape)
```

```
(80000, 17204)
```

In [49]:
```
test_bow=vectorizer.transform(x_test)#fitting the bow model on test dat
a
print("shape of x_test after bow vectorization ",test_bow.shape)
x_test_data=StandardScaler( with_mean=False).fit_transform(test_bow)#st
andarizing the test data
print("shape of x_test after standardization ",x_test_data.shape)
```

```
shape of x_test after bow vectorization  (20000, 17204)
shape of x_test after standardization  (20000, 17204)
```

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 [52]:
```
#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 4 µs, sys: 1 µs, total: 5 µs
Wall time: 9.54 µs
LogisticRegression(C=0.001, 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)
0.9499237910489122
```

In [53]:

```
lr=LogisticRegression(C=0.001,penalty='l2',class_weight={1:.5,0:.5},n_j
obs=-1)#again building the model to find best hyperparameter
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
```

```
[ 3.22430529  3.7502316   1.46200477 ...  6.44421549 -2.94619339
  5.07137844]
```

In [54]:

```
wieght_vector=lr.coef_#getting the weight vector
print(wieght_vector.shape)#wieght vector shape
print(wieght_vector[:20])
```

```
(1, 17204)
[[ 0.00156031 -0.01348685 -0.00636204 ...  0.00090725  0.00090725
   0.00790802]]
```

PERTUBATION TEST :

```
AIM:    TO CHECK FOR MULTI COLLINEARITY  OF FEATURES
```

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</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 [56]:
```
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, 17204)
```

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 [58]:
```
%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 6 µs, sys: 0 ns, total: 6 µs
Wall time: 11.9 µs
best estimator of our new data is:  LogisticRegression(C=0.001, class_w
eight={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 [59]:

```
# again building the model for finding the wieght vector of the words f
rom model
lr=LogisticRegression(C=0.001,penalty='l2',class_weight={1:.5,0:.5},n_j
obs=-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, 17204)
```

In [0]:

```
percent_change_vec=np.ones((1,17204))#generating the percent_change_vet
or 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 [62]:

```
x=(wieght_vector[0][2]-new_wieght_vector[0][2])/wieght_vector[0][2]#jus
t checking randomly that every value is positve in percent_change_vecto
r
print(x)
```

```
0.5041951123949374
```

In [63]:
```
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, 17204)
```

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

In [65]:
```
per_change_df.head()#getting first 5 values
```

Out[65]:

	CHANGE
0	42.443140
1	5.888841
2	50.419511
3	26.623331
4	0.641874

In [66]:
```
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[66]:

	CHANGE
count	17204.000000
mean	105.345025
std	4572.583468
min	0.001482
25%	7.841999
50%	18.893531

	CHANGE
75%	42.919648
max	589144.069804

In [67]:
```
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   0.001
sorted_Data 0.05th quantiles is   1.482
sorted_Data 0.10th quantiles is   3.008
sorted_Data 0.15th quantiles is   4.629
sorted_Data 0.20th quantiles is   6.233
sorted_Data 0.25th quantiles is   7.842
sorted_Data 0.30th quantiles is   9.637
sorted_Data 0.35th quantiles is  11.568
sorted_Data 0.40th quantiles is  13.699
sorted_Data 0.45th quantiles is  16.154
sorted_Data 0.50th quantiles is  18.894
sorted_Data 0.55th quantiles is  22.261
sorted_Data 0.60th quantiles is  25.857
sorted_Data 0.65th quantiles is  30.450
sorted_Data 0.70th quantiles is  35.828
sorted_Data 0.75th quantiles is  42.920
sorted_Data 0.80th quantiles is  53.868
sorted_Data 0.85th quantiles is  70.981
sorted_Data 0.90th quantiles is 102.337
sorted_Data 0.95th quantiles is 183.193
sorted_Data 1.00th quantiles is 589144.070
```

```
In [68]: 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
```

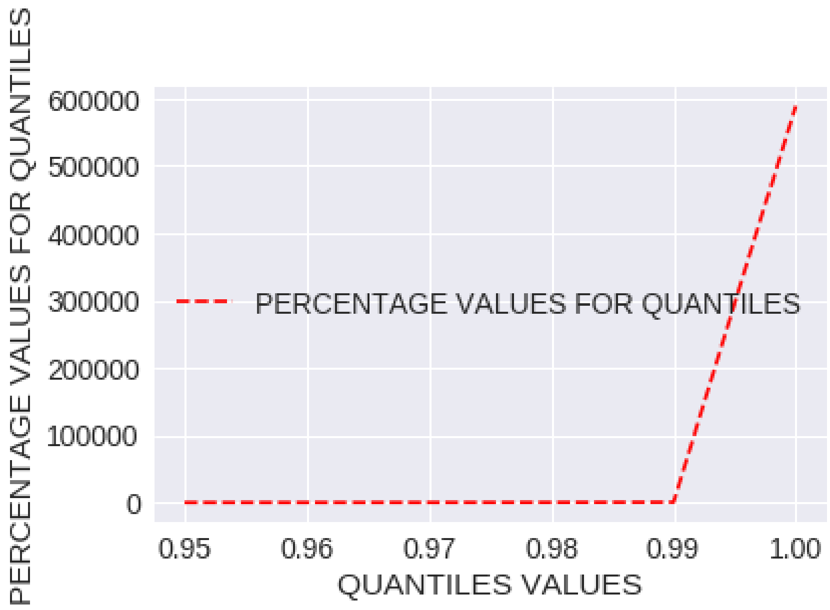
```
sorted_Data 0.95th quantiles is 183.193
sorted_Data 0.96th quantiles is 220.656
sorted_Data 0.97th quantiles is 270.647
sorted_Data 0.98th quantiles is 383.633
sorted_Data 0.99th quantiles is 679.049
sorted_Data 1.00th quantiles is 589144.070
```

```
In [69]: 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
```

```
[183.1929593634676, 220.65596077380062, 270.64692175469014, 383.6329012667263, 679.0491786417696, 589144.0698039597]
[183.19, 220.66, 270.65, 383.63, 679.05, 589144.07]
[0.95, 0.96, 0.97, 0.98, 0.99, 1.0]
```

```
In [70]: %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 FOR QUANTILES')
plt.legend(loc='best')
```

```
Out[70]: <matplotlib.legend.Legend at 0x7f9864386ef0>
```



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 [73]:
```
#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[73]:
```
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 [74]:
```
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[74]:

Out[74]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parar
0	0.717606	0.003946	0.944873	0.964170	0.0001	{'C': 0.000
1	1.459562	0.003723	0.949598	0.985778	0.001	{'C': 0.001
2	2.893135	0.003624	0.945169	0.995277	0.01	{'C': 0.01}
3	5.871643	0.003663	0.936517	0.998380	0.1	{'C': 0.1}
4	10.949172	0.003649	0.929837	0.999340	1	{'C': 1
5	21.886775	0.003672	0.923833	0.999674	10	{'C': 1
6	30.515149	0.003657	0.920352	0.999783	100	{'C': 100}
7	30.891412	0.003730	0.916224	0.999775	1000	{'C': 1000}
8	33.434185	0.003593	0.915188	0.999757	10000	{'C': 1000

9 rows × 31 columns

In [75]:

```
%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=[]
```

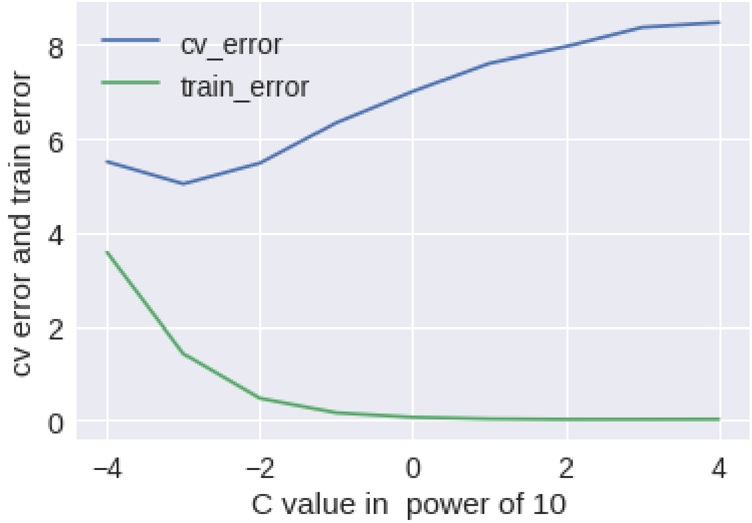
```
for i  in mean_test_score:
   i=1-i
   i=i*100
   cv_error_list.append(i)#appending the list with cv_error
for i  in mean_train_score:
   i=1-i
   i=i*100
   train_error_list.append(i)#appending  the list with train_error

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')
```

```
[5.512688041304193, 5.0401969174698085, 5.483117272006677, 6.3482623609
5829, 7.016290512105949, 7.616710721853415, 7.964755829772341, 8.377557
84182751, 8.481160225314222]
```

Out[75]:

```
<matplotlib.legend.Legend at 0x7f98640e44e0>
```



From here,the best hyperparameter value is c=0.001 or alpha=1000

NOW GETTING THE TOP 30 FEATURES WORDS FOR POSITIVE AND NEGATIVE WORDS

In [76]:

```
#building 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
lr=LogisticRegression(C=0.001,penalty='l2',class_weight={1:.5,0:.5},n_j
obs=-1)#building logistic regression model
lr.fit(x_train_data,y_train)
```

Out[76]:

```
LogisticRegression(C=0.001, 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 [77]:
```
z=lr.coef_[0]#getting the wieght of the vector
print(z)#printing the wieght of the vector
```

```
[ 0.00222298 -0.01428101 -0.00315483 ...  0.00183116  0.00183116
  0.00728819]
```

In [78]:
```
a=z.argsort()
print('shape of wieght vector is:',a.shape)
top_30_positive=np.take(vectorizer.get_feature_names(),a[17174:])
top_30_negative=np.take(vectorizer.get_feature_names(),a[:30])
```

```
shape of wieght vector is: (17204,)
```

In [79]:
```
print("POSITVE WORDS\t|\tNEGATIVE WORDS")
for i,j in zip(top_30_positive,top_30_negative):
    print( '{}\t\t|\t\t{}'.format(i,j) )#printing the postive and negat
ive words
```

```
POSITVE WORDS   |       NEGATIVE WORDS
alway           |               disappoint
friend          |               worst
happi           |               terribl
ever            |               aw
smooth          |               unfortun
ive             |               return
awesom          |               stale
yummi           |               threw
amaz            |               horribl
fast            |               bland
high            |               mayb
keep            |               bad
addict          |               weak
quick           |               thought
right           |               didnt
snack           |               wast
tasti           |               money
```

```
easi                |              wors
thank               |              vomit
find                |              hope
wonder              |              receiv
nice                |              gross
excel               |              tasteless
favorit             |              unpleas
perfect             |              disgust
delici              |              lack
good                |              sorri
best                |              decept
love                |              refund
great               |              expedit
```

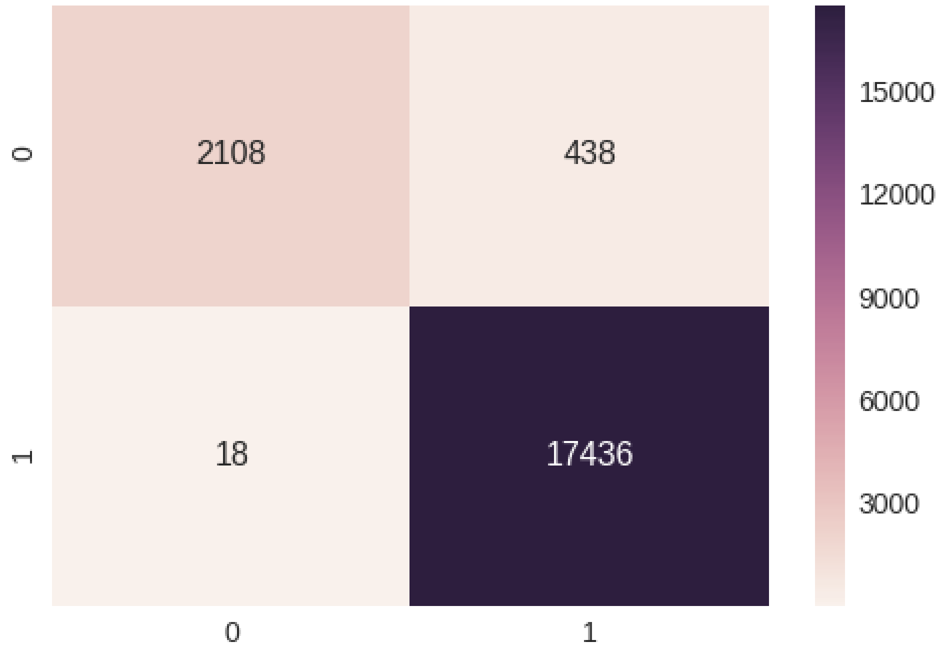
USING BEST HYPERPARAMETER VALUE ON TEST DATA AND PLOTTING THE CONFUSION MATRIX WITH HEATMAP

In [80]:

```
#Testing Accuracy on Test data
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoius metrics from sklearn
#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: 97.720%
Precision on test set: 0.975
Recall on test set: 0.999
F1-Score on test set: 0.987
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[80]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f986404fb70>`



FROM THE ABOVE OBSERVATIONS ,IT IS FOUND THAT THE BEST HYPERPARAMETER IS FOUND AS APLHA=1000 AND IT IS ALSO HAVING HIGH PRECISION,RECALL VALUE ON TEST DATA

In [0]:

```
#BOW VECTORIZATION IS COMPLETED FOR LOGISTIC REGRESSION
```