In [14]:
```
#to ignore warnings
import warnings
warnings.filterwarnings("ignore")
#to use sqlite3 database
import sqlite3
import numpy as np
import pandas as pd
import string
import nltk
import matplotlib.pyplot as plt

from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.grid_search import GridSearchCV
from sklearn.grid_search import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [15]:
```
con = sqlite3.connect('database.sqlite')

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 """, con)


# Give reviews with Score>3 a positive rating, and reviews with a score
<3 a negative rating.
def partition(x):
    if x < 3:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        return 'negative'
    return 'positive'

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
```

Text Preprocessing on all data points

In [3]:
```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball s
temmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation
 or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [4]:
```
#Code for implementing step-by-step the checks mentioned in the pre-pro
cessing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in filtered_data['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTml tags
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (filtered_data['Score'].values)[i] == 'positive'
:
                        all_positive_words.append(s) #list of all words
 used to describe positive reviews
                    if(filtered_data['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words
 used to describe negative reviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("**************************************************
************")

    final_string.append(str1)
    i+=1
```

In [17]:
```
filtered_data['CleanedText']=final_string #adding a column of CleanedTe
xt which displays the data after pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("u
tf-8")
```

Sort the datapoints according to time and take first 50000 points

In [18]:
```
sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(50000)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
S = sorted_data['Score']
Score = S.head(50000)
```

Splitting

In [19]:
```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(sa
mpledata, Score, test_size=0.2, random_state=0)
```

BAG OF WORDS

In [19]:
```
count_vect = CountVectorizer() #in scikit-learn
vec = count_vect.fit(X_train['CleanedText'].values)
```

In [9]:
```
X_trvec = vec.transform(X_train['CleanedText'].values)
X_testvec = vec.transform(X_test['CleanedText'].values)
```

Finding best lamda here c= 1/lamda using both grid and randomised search cv

In [10]:
```
from sklearn import preprocessing
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
lb = preprocessing.LabelBinarizer()
hyperparameter = dict(C=[10**-4, 10**-2, 10**0, 10**2, 10**4],penalty=[
'l1','l2'])
y_train1 = np.array([number[0] for number in lb.fit_transform(y_train
)])
#Using GridSearchCV
model = GridSearchCV(LogisticRegression(class_weight='balanced',multi_c
lass='ovr'), hyperparameter, scoring = 'f1', cv=5)
model.fit(X_trvec, y_train1)
y_test1 = np.array([number[0] for number in lb.fit_transform(y_test)])
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print(model.best_estimator_)
print(model.score(X_testvec, y_test1))
```

```
LogisticRegression(C=100, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.957808778496087
```

In [97]:
```
print("error",float(1-model.score(X_testvec, y_test1)))
```

```
0.04219122150391297
```

In [11]:
```
from sklearn import preprocessing
import scipy.stats as stats
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=
5)}
lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modelr = RandomizedSearchCV(LogisticRegression(class_weight='balanced'
),tuned_parameters,n_iter=5, scoring = 'f1', cv=5)
modelr.fit(X_trvec, y_train1)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelr.best_estimator_)
print(modelr.score(X_testvec, y_test1))
```

```
LogisticRegression(C=310.40783685296634, class_weight='balanced', dual=
False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.9576251982778156
```

Using both l1 and l2 penality

In [12]:
```
modell2 = LogisticRegression(penalty='l2' , C=100,class_weight='balance
d',multi_class='ovr')
modell2.fit(X_trvec, y_train)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print(modell2.score(X_testvec, y_test))
modell1 = LogisticRegression(penalty = 'l1' , C=100,class_weight='balan
ced')
modell1.fit(X_trvec, y_train)
#print(modell1.class_weight)

print(modell1.score(X_testvec, y_test))
```

```
0.9256
0.9265
```

As c increses, error and sparsity

AS C DECREASES non zero elements decreaces sparsity increases

In [13]:
```
tuned_parameters = [10**-4, 10**-2, 10**0, 10**2, 10**4]
ERROR=[]
Nonzero = []
for c in tuned_parameters:
    print("for c= ", c)
    modell1 = LogisticRegression(penalty = 'l1' , C=c)
    modell1.fit(X_trvec, y_train1)
    f1_score=modell1.score(X_testvec, y_test1)
    error = float(1-f1_score)
    print("error=",error)
    ERROR.append(error)
    w=modell1.coef_
    print("nonzeros = ",np.count_nonzero(w))
    Nonzero.append(np.count_nonzero(w))
```

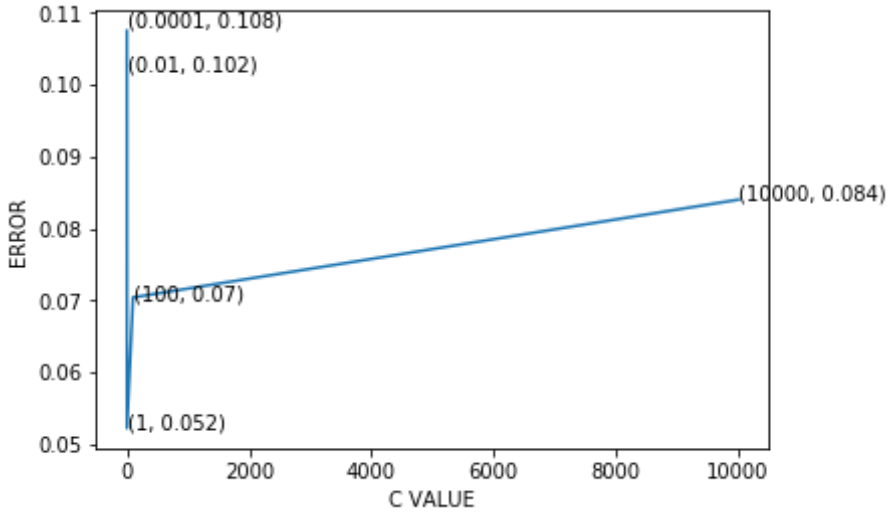
```
for c=  0.0001
error= 0.10750000000000004
nonzeros =  0
for c=  0.01
error= 0.10229999999999995
nonzeros =  65
for c=  1
error= 0.052200000000000024
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
nonzeros =  2327
for c=  100
error= 0.07040000000000002
nonzeros =  4861
for c=  10000
error= 0.08399999999999996
nonzeros =  9163
```

In [14]:
```
tuned_parameters1=tuned_parameters
```

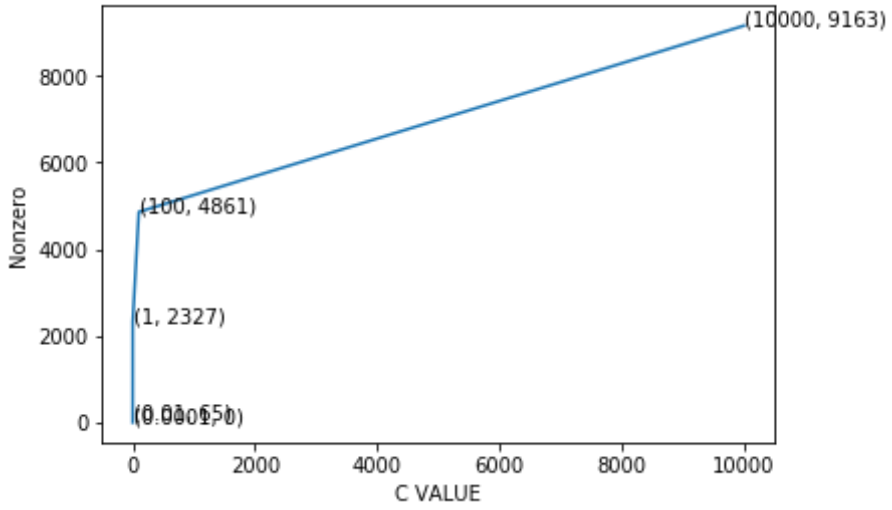
In [15]:
```
plt.plot(tuned_parameters1,ERROR)
for xy in zip(tuned_parameters1, np.round(ERROR,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel('C VALUE')
plt.ylabel('ERROR')
plt.show()
```



In [16]:
```
plt.plot(tuned_parameters,Nonzero)
for xy in zip(tuned_parameters, np.round(Nonzero,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel('C VALUE')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
plt.ylabel('Nonzero')
plt.show()
```



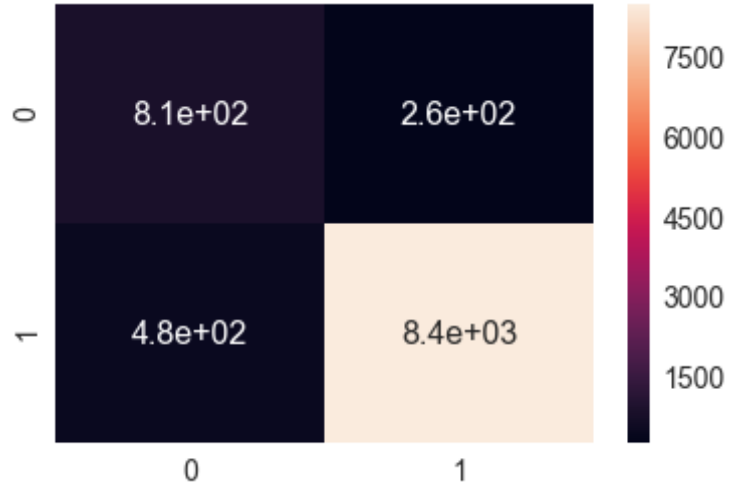
Confusion matrix

In [17]:
```
pred = model.predict(X_testvec)
from sklearn.metrics import confusion_matrix
import seaborn as sn
print(confusion_matrix(y_test1, pred))
CFM = confusion_matrix(y_test1, pred)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
[[ 811  264]
 [ 480 8445]]
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x129afd48780>

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [18]:
```
TP = CFM[0][0]
FP = CFM[1][0]
FN = CFM[0][1]
TN = CFM[1][1]
P = TP+FP
N = TN+FN
print('TP Value is',TP )
print('FP Value is',FP )
print('TN Value is',TN )
print('FN Value is',FN )

TPR = float(TP/P)
FPR = float(FP/P)
TNR = float(TN/N)
FNR = float(FN/N)
print('TPR Value is',TPR )
print('FPR Value is',FPR )
print('TNR Value is',TNR )
print('FNR Value is',FNR )
```

```
TP Value is 811
FP Value is 480
TN Value is 8445
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
FN Value is 264
TPR Value is 0.6281951975213013
FPR Value is 0.3718048024786987
TNR Value is 0.9696865311746469
FNR Value is 0.030313468825353084
```

Different metrics

In [19]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test1, pred))
print(f1_score(y_test1, pred, average="macro"))
print(float(1-f1_score(y_test1, pred, average="macro")))
print(precision_score(y_test1, pred, average="macro"))
print(recall_score(y_test1, pred, average="macro"))
```

```
0.9256
0.8216770012514247
0.17832299874857527
0.7989408643479741
0.8503185460230605
```

Important words in word colab

In [64]:
```
comment_words = ' '
for val in X_train['CleanedText'].values:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '
```

In [65]:

```
from wordcloud import WordCloud
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Feature importance for bow vectorizer

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [20]:
```
modeln = LogisticRegression(C=10000, class_weight='balanced', dual=Fals
e,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
modeln.fit(X_trvec, y_train1)
pred = modeln.predict(X_trvec)
def important_features(vectorizer,classifier,n=10):
    class_labels = classifier.classes_
    print(class_labels)
    feature_names =vectorizer.get_feature_names()
    classifier.coef_
    topn_class1 = sorted(zip(pred,classifier.coef_[0], feature_names),r
everse=True)
    coef1 = classifier.coef_
    #print(coef1.shape)
    #topn_class2 = sorted(zip(classifier.coef_[1], feature_names),rever
se=True)[:n]
    #coef2 = classifier.coef_[1]
    #print(topn_class1)

    count=0
    count1=0
    print("Important words in positive reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 1 and count<=n:
            print( class1,coef, feat)
            count=count+1
        if count == 20:
            break
    print("Important words in negative reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 0 and count1<=n:
            print( class1,coef, feat)
            count1=count1+1
        if count1 == 20:
            break
    #return coef1
    #for coef, feat in topn_class2:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    #   print( coef, feat)
   #return coef2
important_features(vec,modeln,n=20)
```

```
[0 1]
Important words in positive reviews
1 73.14032739311038 disstributor
1 70.62960656993062 overjoy
1 63.948492840639446 honeyd
1 62.06836165179028 encas
1 61.13115979849053 coffeepot
1 58.493077359740624 ucc
1 58.36057712100579 nutro
1 57.6230797870639 wherev
1 57.0435335303401 emirel
1 53.47956428182208 outgo
1 53.35055713079648 scissor
1 50.990941823312426 segment
1 47.844826721044264 choclat
1 46.6376674613895 candida
1 46.5831820035843 crackl
1 46.1614359580231 wager
1 45.87374085992653 oct
1 45.67272028229311 grit
1 45.491059121884554 deliious
1 45.47251011415741 latino
Important words in negative reviews
0 54.53304211388773 honeyvillgrain
0 50.70242261789873 tater
0 50.261323176136585 demolish
0 48.147534329126174 zreport
0 47.83746278268702 mintu
0 41.69267583921716 cottonse
0 38.064131300017465 beneficiari
0 37.11451290349055 karla
0 36.87799583120173 downsid
0 36.322052978642276 puppet
0 34.90445096583091 dishwash
0 33.34273051291668 livestock
0 32.03882568584797 ild
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
0 30.672968694360396 prevent
0 29.844320587595252 twang
0 29.623733632441738 haystack
0 29.46140965343905 angostura
0 29.31788662300061 craft
0 29.221470319162272 bookstor
0 29.211011271614687 saut
```

TFIDF

In [20]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit(X_train['CleanedText'].values)
```

In [21]:
```
X_tr_tf_idf = final_tf_idf.transform(X_train['CleanedText'].values)
X_test_tf_idf = final_tf_idf.transform(X_test['CleanedText'].values)
```

In [23]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn import preprocessing
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
lb = preprocessing.LabelBinarizer()
hyperparameter = dict(C=[10**-4, 10**-2, 10**0, 10**2, 10**4],penalty=[
'l1','l2'])
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using GridSearchCV
modeltf_idf = GridSearchCV(LogisticRegression(class_weight='balanced',m
ulti_class='ovr'), hyperparameter, scoring = 'f1', cv=5)
modeltf_idf.fit(X_tr_tf_idf, y_train1)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modeltf_idf.best_estimator_)
print(modeltf_idf.score(X_test_tf_idf, y_test1))
```

```
LogisticRegression(C=100, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.977556945064761
```

In [98]:
```
print("error=",float(1-modeltf_idf.score(X_test_tf_idf, y_test1)))
```

```
error= 0.022443054935239015
```

In [25]:
```
from sklearn import preprocessing
import scipy.stats as stats
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=
5)}
lb = preprocessing.LabelBinarizer()
y_train1 = np.array([number[0] for number in lb.fit_transform(y_train
)])
#Using RandomizedSearchCV
modeltf_idfr = RandomizedSearchCV(LogisticRegression(),tuned_parameters
,n_iter=5, scoring = 'f1', cv=5)
modeltf_idfr.fit(X_tr_tf_idf, y_train1)
y_test1 = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modeltf_idfr.best_estimator_)
print(modeltf_idfr.score(X_test_tf_idf, y_test1))
```

```
LogisticRegression(C=7958.206871637053, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.9785702419846061
```

In [22]:
```
modell2 = LogisticRegression(penalty = 'l2' , C=100,class_weight='balan
ced')
modell2.fit(X_tr_tf_idf, y_train)
print(modell2.score(X_test_tf_idf, y_test))
modell1 = LogisticRegression(penalty = 'l1' , C=100,class_weight='balan
ced')
modell1.fit(X_tr_tf_idf, y_train)
print(modell1.score(X_test_tf_idf, y_test))
```

```
0.9598
0.9552
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

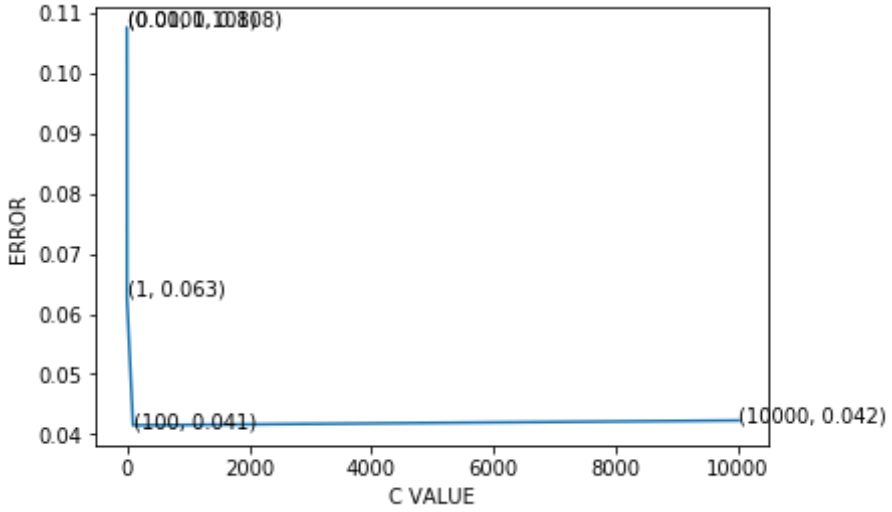
In [26]:
```
tuned_parameters = [10**-4, 10**-2, 10**0, 10**2, 10**4]
ERROR=[]
Nonzero = []
for c in tuned_parameters:
    print("for c= ", c)
    modell1 = LogisticRegression(penalty = 'l1' , C=c)
    modell1.fit(X_tr_tf_idf, y_train1)
    f1_score=modell1.score(X_test_tf_idf, y_test1)
    error = float(1-f1_score)
    print("error=",error)
    ERROR.append(error)
    w=modell1.coef_
    print("sparsity = ",np.count_nonzero(w))
    Nonzero.append(np.count_nonzero(w))
```

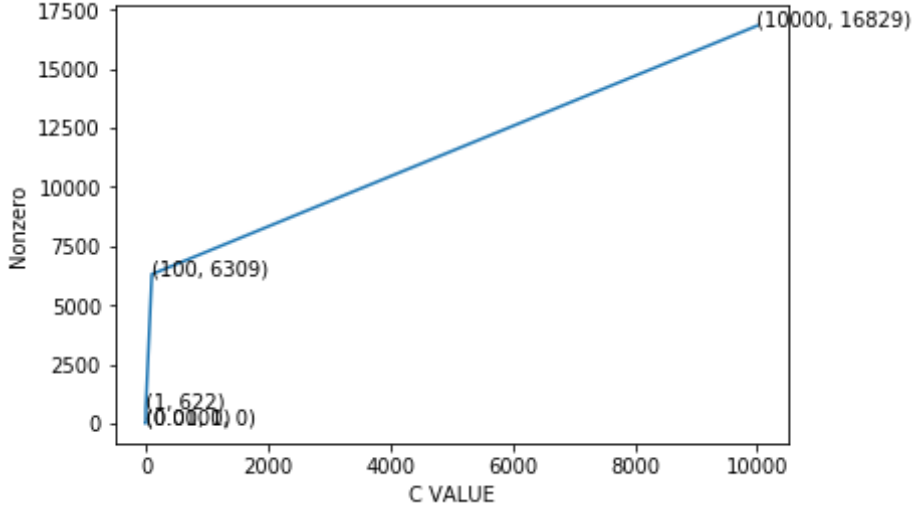
```
for c=  0.0001
error= 0.10750000000000004
sparsity =  0
for c=  0.01
error= 0.10750000000000004
sparsity =  0
for c=  1
error= 0.06259999999999999
sparsity =  622
for c=  100
error= 0.04149999999999998
sparsity =  6309
for c=  10000
error= 0.042300000000000004
sparsity =  16829
```

In [27]:
```
tuned_parameters1=tuned_parameters
plt.plot(tuned_parameters1,ERROR)
for xy in zip(tuned_parameters1, np.round(ERROR,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel('C VALUE')
plt.ylabel('ERROR')
plt.show()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
plt.plot(tuned_parameters,Nonzero)
for xy in zip(tuned_parameters, np.round(Nonzero,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel('C VALUE')
plt.ylabel('Nonzero')
plt.show()
```



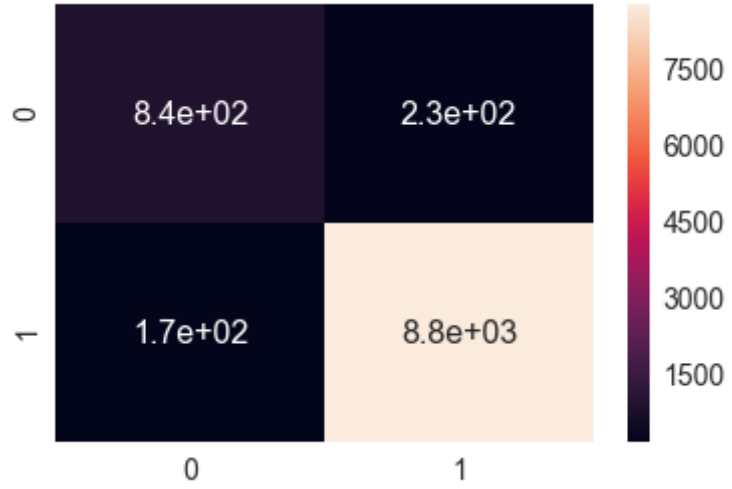


Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [28]:
```
pred = modeltf_idf.predict(X_test_tf_idf)
from sklearn.metrics import confusion_matrix
import seaborn as sn
print(confusion_matrix(y_test1, pred))
CFM = confusion_matrix(y_test1, pred)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
[[ 843  232]
 [ 170 8755]]
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x12a06668e48>



In [29]:
```
TP = CFM[0][0]
FP = CFM[1][0]
FN = CFM[0][1]
TN = CFM[1][1]
P = TP+FP
N = TN+FN
print('TP Value is',TP )
print('FP Value is',FP )
print('TN Value is',TN )
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print('FN Value is',FN )

TPR = float(TP/P)
FPR = float(FP/P)
TNR = float(TN/N)
FNR = float(FN/N)
print('TPR Value is',TPR )
print('FPR Value is',FPR )
print('TNR Value is',TNR )
print('FNR Value is',FNR )
```

```
TP Value is 843
FP Value is 170
TN Value is 8755
FN Value is 232
TPR Value is 0.8321816386969397
FPR Value is 0.16781836130306022
TNR Value is 0.9741849337932569
FNR Value is 0.02581506620674307
```

In [30]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test1, pred))
print(f1_score(y_test1, pred, average="macro"))
print(float(1-f1_score(y_test1, pred, average="macro")))
print(precision_score(y_test1, pred, average="macro"))
print(recall_score(y_test1, pred, average="macro"))
```

```
0.9598
0.8925141047162886
0.10748589528371144
0.9031832862450984
0.8825692137320045
```

Feature importance for TFIDF vectorizer

In [56]:
```
modeltf_idfn = LogisticRegression(C=100, class_weight='balanced', dual=
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
modeltf_idfn.fit(X_tr_tf_idf, y_train1)
pred = modeltf_idfn.predict(X_tr_tf_idf)
def important_features(vectorizer,classifier,n=10):
    class_labels = classifier.classes_
    print(class_labels)
    feature_names =vectorizer.get_feature_names()
    classifier.coef_
    topn_class1 = sorted(zip(pred,classifier.coef_[0], feature_names),r
everse=True)
    coef1 = classifier.coef_
    print(coef1.shape)
    #topn_class2 = sorted(zip(classifier.coef_[1], feature_names),rever
se=True)[:n]
    #coef2 = classifier.coef_[1]
    #print(topn_class1)

    count=0
    count1=0
    print("Important words in positive reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 1 and count<=20:
            print( class1,coef, feat)
            count=count+1
        if count == 20:
            break
    print("Important words in negative reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 0 and count1<=20:
            print( class1,coef, feat)
            count1=count1+1
        if count1 == 20:
            break
    return coef1[0]

normal=important_features(final_tf_idf,modeltf_idfn,n=20)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[0 1]
(1, 485312)
Important words in positive reviews
1 25.545720238949258 best
1 8.512751837102169 awesom
1 8.385784215009762 alway
1 8.2777696280068 beat
1 7.683489255034327 beauti
1 6.1438481006770855 along
1 5.225690896337486 add
1 4.835804167495692 bargain
1 4.786002674913451 also like
1 4.706959628082526 authent
1 4.68011649278405 bad thing
1 4.650964975559128 best tast
1 4.539657125612854 adult
1 4.463284329198772 balanc
1 4.441126279807594 allergi
1 4.342642564153123 actual tast
1 4.310667446096405 amazon price
1 3.9027172715394727 almond
1 3.783188286773656 best cant
1 3.5787289215238594 better price
Important words in negative reviews
0 13.01194700657768 amaz
0 11.642472607269777 addict
0 2.9976969468035466 best price
0 2.9898557915731465 almost good
0 2.8738598747733546 arriv fresh
0 2.8090363217525818 banana
0 2.7877747144787035 anywher
0 2.5640538139856317 avail
0 2.357029392613495 ahead
0 2.3399179549393674 aunt
0 2.2985060900942305 better amazon
0 2.160139161253434 auto
0 1.8926735564787154 aid
0 1.884130833390187 best one
0 1.8797842303473924 amazon problem
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
0 1.8638585792588342 aisl
0 1.844321357764051 better plain
0 1.8421231304129893 bean tortilla
0 1.8375564761688785 add anyth
0 1.7912853911349305 abl order
```

Feature importance

Adding noise and Pertubation test

Applying pertubation test to tfidf as it is more accurate

In [68]:
```
from scipy.sparse import csr_matrix
data=csr_matrix(X_tr_tf_idf)
Noice=0.00001
data.data=data.data+Noice
modeltf_idfn = LogisticRegression(C=100, class_weight='balanced', dual=
False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
modeltf_idfn.fit(data, y_train1)
```

Out[68]:
```
LogisticRegression(C=100, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [69]:
```
noice=important_features(final_tf_idf,modeltf_idfn,n=20)
```

```
[0 1]
(1, 485312)
Important words in positive reviews
1 25.544019892069866 best
1 8.512465810489692 awesom
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
1 8.385105073137431 alway
1 8.277537480925751 beat
1 7.683140635507165 beauti
1 6.143490995659745 along
1 5.225134699633982 add
1 4.835611592911452 bargain
1 4.78589097131044 also like
1 4.706894156300907 authent
1 4.68002933135486 bad thing
1 4.650582128840519 best tast
1 4.539350076912054 adult
1 4.463058179945184 balanc
1 4.440926680570245 allergi
1 4.342530466122892 actual tast
1 4.31043516232886 amazon price
1 3.9024929493350426 almond
1 3.7830032119537016 best cant
1 3.5785721866598426 better price
Important words in negative reviews
0 13.01141785484945 amaz
0 11.641959985391477 addict
0 2.9974478898095236 best price
0 2.9897819269331825 almost good
0 2.873742625233141 arriv fresh
0 2.808807909633348 banana
0 2.787585899862552 anywher
0 2.563765194710402 avail
0 2.3569771975470553 ahead
0 2.339796313693008 aunt
0 2.298474857648644 better amazon
0 2.160028936885527 auto
0 1.8925840013461055 aid
0 1.884003371598605 best one
0 1.8797638439922708 amazon problem
0 1.8637384611778436 aisl
0 1.8443157061170476 better plain
0 1.8421157702856246 bean tortilla
0 1.837518244280568 add anyth
0 1.7911839379679584 abl order
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [70]:
```
avg = (abs(normal - noice)/normal)*100
feature_names = final_tf_idf.get_feature_names()
print(sorted(avg,reverse=True)[0:20])
```

```
[100.63020194855979, 67.37613679880691, 49.78855842209875, 49.788558422
09875, 32.92641236778007, 20.690436418943985, 14.35349231810507, 14.353
49231810507, 14.35349231810507, 14.35349231810507, 14.35349231810507, 1
4.35349231810507, 14.35349231810507, 14.35349231810507, 14.353492318105
07, 14.35349231810507, 14.35349231810507, 14.35349231810507, 14.3534923
1810507, 14.35349231810507]
```

In [71]:
```
sorted_diff=sorted(zip(avg, feature_names),reverse=True)[0:20]
for av,feat in sorted_diff :
    if av>30:
        print(av,feat)
#five of them are more than 30% , I think my data has multicollinearity
 but as nothing of them is an important word   it will not effect my mo
del
```

```
100.63020194855979 reliv
67.37613679880691 tummi troubl
49.78855842209875 write home
49.78855842209875 noth write
32.92641236778007 jerki unless
```

WORD TO VEC

In [33]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
i=0
list_of_sent=[]
for sent in X_train['CleanedText'].values:
    list_of_sent.append(sent.split())
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [34]:
```
from tqdm import tqdm
import os
sent_vectorstr = []; # the avg-w2v for each sentence/review is stored i
n this list
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorstr.append(sent_vec)
```

```
100%|████████████████████████████████████████████████████████████████
████████| 40000/40000 [00:36<00:00, 1087.76it/s]
```

In [35]:
```
list_of_sent1 = []
for sent in X_test['CleanedText'].values:
    list_of_sent1.append(sent.split())
sent_vectorstest = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sent1): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorstest.append(sent_vec)
```

```
100%|████████████████████████████████████████████████████████████████
████████| 10000/10000 [00:10<00:00, 976.93it/s]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [20]:
```
from sklearn import preprocessing
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
lb = preprocessing.LabelBinarizer()
y_train1 = np.array([number[0] for number in lb.fit_transform(y_train
)])
#Using GridSearchCV
hyperparameter = dict(C=[10**-4, 10**-2, 10**0, 10**2, 10**4],penalty=[
'l1','l2'])
modelw2v = GridSearchCV(LogisticRegression(class_weight='balanced',mult
i_class='ovr'), hyperparameter, scoring = 'f1', cv=5)
modelw2v.fit(sent_vectorstr, y_train1)
y_test1 = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelw2v.best_estimator_)
print(modelw2v.score(sent_vectorstest, y_test1))
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.8660936721474838
```

In [21]:
```
print("error=",float(1-modelw2v.score(sent_vectorstest, y_test1)))
```

```
error= 0.13390632785251622
```

In [22]:
```
from sklearn import preprocessing
import scipy.stats as stats
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=
5)}
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modelr = RandomizedSearchCV(LogisticRegression(),tuned_parameters,n_ite
r=5, scoring = 'f1', cv=5)
modelr.fit(sent_vectorstr, y_train1)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelr.best_estimator_)
print(modelr.score(sent_vectorstest, y_test1))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
LogisticRegression(C=7960.542830421929, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.950235555314886
```

In [38]:
```
modell2 = LogisticRegression(penalty = 'l2' , C=1 ,class_weight='balanced')
modell2.fit(sent_vectorstr, y_train)
print(modell2.score(sent_vectorstest, y_test))
modell1 = LogisticRegression(penalty = 'l1' , C=1 ,class_weight='balanced')
modell1.fit(sent_vectorstr, y_train)
print(modell1.score(sent_vectorstest, y_test))
```

```
0.7882
0.788
```

In [39]:
```
tuned_parameters = [10**-4, 10**-2, 10**0, 10**2, 10**4]
ERROR=[]
Nonzero = []
for c in tuned_parameters:
    print("for c= ", c)
    modell1 = LogisticRegression(penalty = 'l1' , C=c)
    modell1.fit(sent_vectorstr, y_train)
    f1_score=modell1.score(sent_vectorstest, y_test)
    error = float(1-f1_score)
    print("error=",error)
    ERROR.append(error)
    w=modell1.coef_
    print("sparsity = ",np.count_nonzero(w))
    Nonzero.append(np.count_nonzero(w))
```

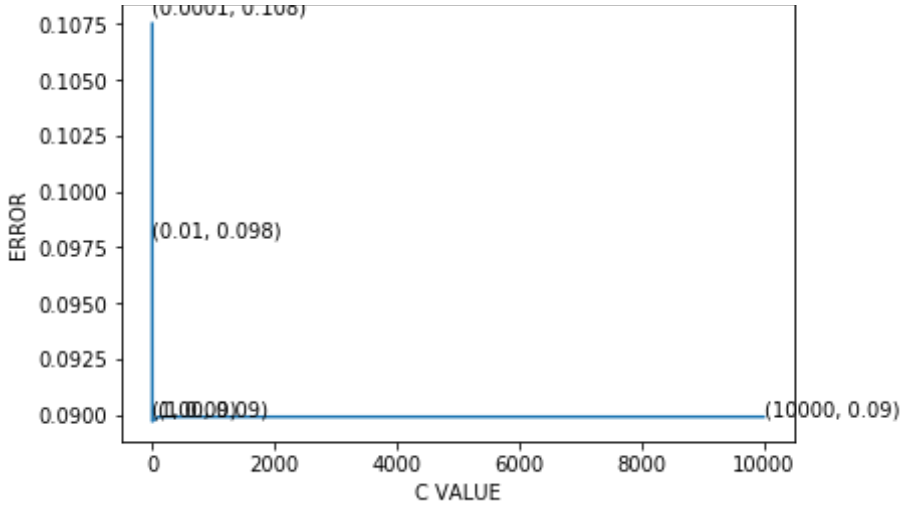
```
for c=  0.0001
error= 0.10750000000000004
sparsity =  0
for c=  0.01
error= 0.09819999999999995
sparsity =  26
```

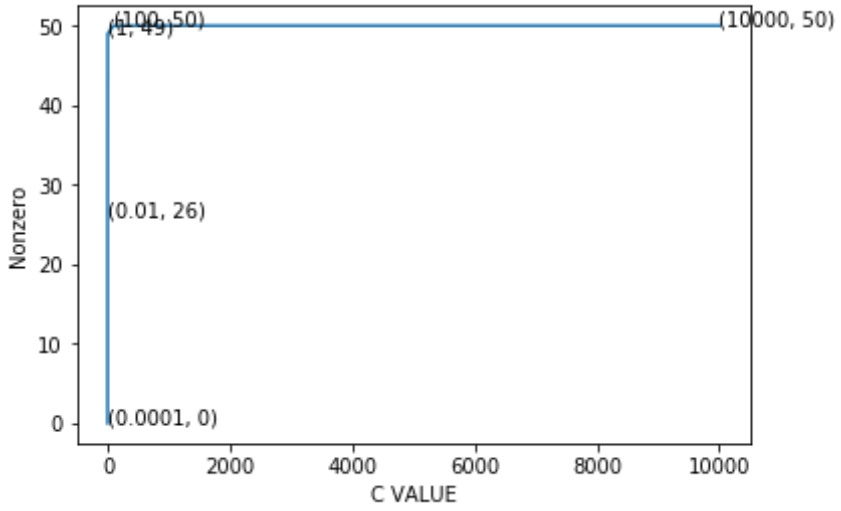
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
for c=  1
error= 0.0897
sparsity =  49
for c=  100
error= 0.08989999999999998
sparsity =  50
for c=  10000
error= 0.08989999999999998
sparsity =  50
```

In [40]:
```
tuned_parameters1=tuned_parameters
plt.plot(tuned_parameters1,ERROR)
for xy in zip(tuned_parameters1, np.round(ERROR,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel('C VALUE')
plt.ylabel('ERROR')
plt.show()
plt.plot(tuned_parameters,Nonzero)
for xy in zip(tuned_parameters, np.round(Nonzero,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel('C VALUE')
plt.ylabel('Nonzero')
plt.show()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD





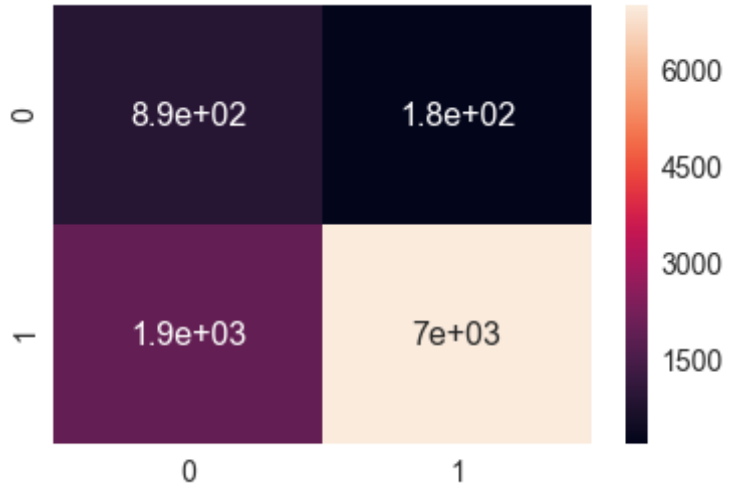
In [41]:
```
pred = modell2.predict(sent_vectorstest)
from sklearn.metrics import confusion_matrix
import seaborn as sn
print(confusion_matrix(y_test, pred))
CFM = confusion_matrix(y_test, pred)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
[[ 893  182]
 [1936 6989]]
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x22c0a6b24a8>



In [42]:
```
TP = CFM[0][0]
FP = CFM[1][0]
FN = CFM[0][1]
TN = CFM[1][1]
P = TP+FP
N = TN+FN
print('TP Value is',TP )
print('FP Value is',FP )
print('TN Value is',TN )
print('FN Value is',FN )

TPR = float(TP/P)
FPR = float(FP/P)
TNR = float(TN/N)
FNR = float(FN/N)
print('TPR Value is',TPR )
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print('FPR Value is',FPR )
print('TNR Value is',TNR )
print('FNR Value is',FNR )
```

```
TP Value is 893
FP Value is 1936
TN Value is 6989
FN Value is 182
TPR Value is 0.31565924354895725
FPR Value is 0.6843407564510428
TNR Value is 0.9746199972109887
FNR Value is 0.025380002789011296
```

In [43]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(float(1-f1_score(y_test, pred, average="macro")))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.7882
0.6629470105595933
0.33705298944040674
0.645139620379973
0.8068894534558009
```

TF-IDF WORD TO VEC

In [44]:
```
from tqdm import tqdm
import os
# TF-IDF weighted Word2Vec

tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect
.idf_)))# tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sent): # 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/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 40000/40000 [00:43<00:00, 923.57it/s]
```

In [45]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sent1): # 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/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
                # obtain the tf_idfidf of a word in a sentence/review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors_test.append(sent_vec)
        row += 1
```

```
100%|██████████████████████████████████████████████████████████████████
█████████| 10000/10000 [00:11<00:00, 885.63it/s]
```

In [34]:

```
from sklearn import preprocessing
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using GridSearchCV
hyperparameter = dict(C=[10**-4, 10**-2, 10**0, 10**2, 10**4],penalty=[
'l1','l2'])

modeltfw2v = GridSearchCV(LogisticRegression(class_weight='balanced',mu
lti_class='ovr'), hyperparameter, scoring = 'f1', cv=5)
modeltfw2v.fit(tfidf_sent_vectors, y_train1)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modeltfw2v.best_estimator_)
print(modeltfw2v.score(tfidf_sent_vectors_test, y_test1))
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1135: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1135: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1135: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\metrics\classificati
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
on.py:1135: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1135: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
LogisticRegression(C=10000, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.8473720739478832
```

In [35]:
```
print("error=",float(1-modeltfw2v.score(tfidf_sent_vectors_test, y_test
1)))
```

```
error= 0.15262792605211684
```

In [36]:
```
from sklearn import preprocessing
import scipy.stats as stats
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=
5)}
hyperparameter = dict(C=[10**-4, 10**-2, 10**0, 10**2, 10**4],penalty=[
'l1','l2'])
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modelr = RandomizedSearchCV(LogisticRegression(),hyperparameter,n_iter=
5, scoring = 'f1', cv=5)
modelr.fit(tfidf_sent_vectors, y_train1)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelr.best_estimator_)
print(modelr.score(tfidf_sent_vectors_test, y_test1))
```

```
LogisticRegression(C=10000, class_weight=None, dual=False, fit_intercep
t=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.00
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
01,
          verbose=0, warm_start=False)
0.9457564973579209
```

In [47]:
```
modell2 = LogisticRegression(penalty = 'l2' , C=10000 ,class_weight='ba
lanced')
modell2.fit(tfidf_sent_vectors, y_train)
print(modell2.score(tfidf_sent_vectors_test, y_test))
modell1 = LogisticRegression(penalty = 'l1' , C=10000 ,class_weight='ba
lanced')
modell1.fit(tfidf_sent_vectors, y_train)
print(modell1.score(tfidf_sent_vectors_test, y_test))
```

```
0.7616
0.7616
```

In [48]:
```
tuned_parameters = [10**-4, 10**-2, 10**0, 10**2, 10**4]
ERROR=[]
Nonzero = []
for c in tuned_parameters:
    print("for c= ", c)
    modell1 = LogisticRegression(penalty = 'l1' , C=c)
    modell1.fit(tfidf_sent_vectors, y_train)
    f1_score=modell1.score(tfidf_sent_vectors_test, y_test)
    error = float(1-f1_score)
    print("error=",error)
    ERROR.append(error)
    w=modell1.coef_
    print("sparsity = ",np.count_nonzero(w))
    Nonzero.append(np.count_nonzero(w))
```

```
for c=  0.0001
error= 0.10750000000000004
sparsity =  0
for c=  0.01
error= 0.1049
sparsity =  30
for c=  1
error= 0.09860000000000002
```

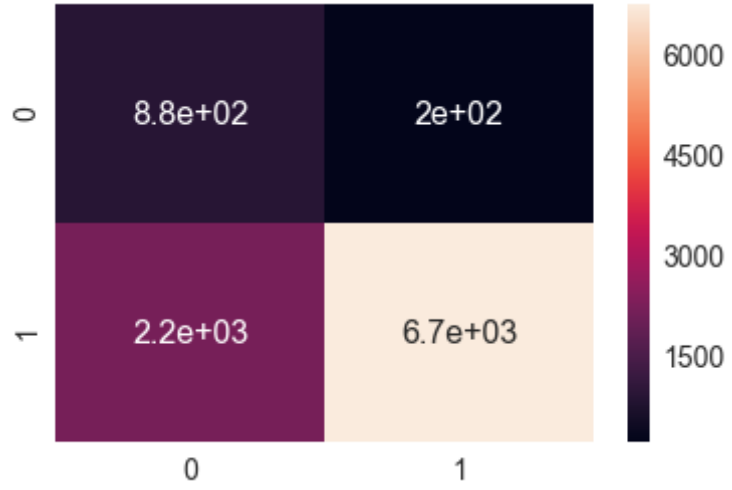
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
sparsity =  50
for c=  100
error= 0.09830000000000005
sparsity =  50
for c=  10000
error= 0.09830000000000005
sparsity =  50
```

In [49]:
```
pred = modell2.predict(tfidf_sent_vectors_test)
from sklearn.metrics import confusion_matrix
import seaborn as sn
print(confusion_matrix(y_test, pred))
CFM = confusion_matrix(y_test, pred)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
[[ 877  198]
 [2186 6739]]
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x22c0792ea90>



In [50]:
```
TP = CFM[0][0]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
FP = CFM[1][0]
FN = CFM[0][1]
TN = CFM[1][1]
P = TP+FP
N = TN+FN
print('TP Value is',TP )
print('FP Value is',FP )
print('TN Value is',TN )
print('FN Value is',FN )

TPR = float(TP/P)
FPR = float(FP/P)
TNR = float(TN/N)
FNR = float(FN/N)
print('TPR Value is',TPR )
print('FPR Value is',FPR )
print('TNR Value is',TNR )
print('FNR Value is',FNR )
```

```
TP Value is 877
FP Value is 2186
TN Value is 6739
FN Value is 198
TPR Value is 0.2863206007182501
FPR Value is 0.71367939928175
TNR Value is 0.9714574023353034
FNR Value is 0.028542597664696554
```

In [51]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(float(1-f1_score(y_test, pred, average="macro")))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.7616
0.6367899815463716
0.36321001845362844
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
0.6288890015267767
0.7854419907497883
```

In [52]:

```
from prettytable import PrettyTable
x = PrettyTable(["Table", "BOW", "TF-IDF","W2V","TFIDF W2V"])
while True:
    #- Get value
    prompt = input("Please add a head to the list\n")

    try:
         #- Type Casting.
        prompt1 = float(input("Please add a BOW to the list\n"))
        prompt2 = float(input("Please enter a TF-IDF for the service\n"
))
        prompt3 = float(input("Please enter a W2V for the service\n"))
        prompt4 = float(input("Please enter a TFIDF W2V for the service
\n"))
    except ValueError:
        print("Please enter valid type")
        continue
    #- Add row
    x.add_row([ prompt,prompt1, prompt2,prompt3,prompt4])
    #- Ask user to Continue or not.
    choice = input("Continue yes/ no:").lower()
    if not(choice=="yes" or choice=="y"):
        break
```

```
Please add a head to the list
C
Please add a BOW to the list
100
Please enter a TF-IDF for the service
100
Please enter a W2V for the service
1
Please enter a TFIDF W2V for the service
10000
Continue yes/ no:Y
Please add a head to the list
c-error
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Please add a BOW to the list
0.042
Please enter a TF-IDF for the service
0.022
Please enter a W2V for the service
0.133
Please enter a TFIDF W2V for the service
0.152
Continue yes/ no:y
Please add a head to the list
Test error
Please add a BOW to the list
0.17
Please enter a TF-IDF for the service
0.107
Please enter a W2V for the service
0.337
Please enter a TFIDF W2V for the service
0.363
Continue yes/ no:n
```

In [53]:
```
print(x)
```

```
+------------+-------+--------+-------+-----------+
|   Table    |  BOW  | TF-IDF |  W2V  | TFIDF W2V |
+------------+-------+--------+-------+-----------+
|     C      | 100.0 | 100.0  |  1.0  |  10000.0  |
|  c-error   | 0.042 | 0.022  | 0.133 |   0.152   |
| Test error |  0.17 | 0.107  | 0.337 |   0.363   |
+------------+-------+--------+-------+-----------+
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD