In [20]:
```python
#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.cross_validation import cross_val_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [2]:
```python
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:
        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]:
```python
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

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]:
```python
#Code for implementing step-by-step the checks mentioned in the pre-processing 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
    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 re
views
                    if(filtered_data['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative re
views 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 [5]:
```python
filtered_data['CleanedText']=final_string #adding a column of CleanedText which displays the data af
ter pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("utf-8")
```

Sort the datapoints according to time and take first 100000 points

In [6]:
```python
sorted_data=filtered_data.sort_values(by=['Time'])


S = sorted_data['Score']
Score = S
```

Splitting

In [51]:
```python
# HERE WE ARE SPLITTING THE DATA POINTS IN TO 80% TRAIN AND 20% FOR TEST
X_1, X_test, y_1, y_test = cross_validation.train_test_split(sorted_data, Score, test_size=0.2, rand
om_state=0)
#HERE WE ARE AGAIN SPLITTING THE TRAIN DATA IN EARLIER LINE X_1 IN TO 75% TRAINING AND 25% CROSS VAL
IDATION DATA so we have 60% 20% 20% ratio
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.25)
```

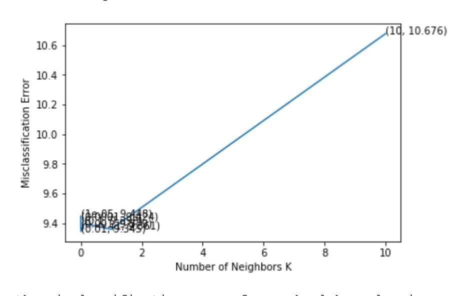
BAG OF WORDS

In [52]:
```python
count_vect = CountVectorizer() #in scikit-learn
vec = count_vect.fit(X_tr['CleanedText'].values)
```

In [53]:
```python
X_trvec = vec.transform(X_tr['CleanedText'].values)
X_cvvec = vec.transform(X_cv['CleanedText'].values)
X_testvec = vec.transform(X_test['CleanedText'].values)
```

In [10]:
```python
#[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
from sklearn.naive_bayes import MultinomialNB
listacc=[]

alpha = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
for i in alpha:

    b = MultinomialNB(alpha=float(i))

    # fitting the model on crossvalidation train
    b.fit(X_trvec, y_tr)

    # predict the response on the crossvalidation train
    pred = b.predict(X_cvvec)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for alpha = %f is %d%%' % (i, acc))
    listacc.append(acc)
MSE = [100 - x for x in listacc]


optimal_k = alpha[MSE.index(min(MSE))]
print('\nThe best alpha  is %f.' % optimal_k)

plt.plot(alpha, MSE)

for xy in zip(alpha, np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each alpha value is : ", np.round(MSE,3))
```

```
CV accuracy for alpha = 0.000010 is 90%

CV accuracy for alpha = 0.000100 is 90%

CV accuracy for alpha = 0.001000 is 90%

CV accuracy for alpha = 0.010000 is 90%

CV accuracy for alpha = 0.100000 is 90%

CV accuracy for alpha = 1.000000 is 90%

CV accuracy for alpha = 10.000000 is 89%

The best alpha  is 0.010000.
```




```
the misclassification error for each alpha value is :  [ 9.448  9.424  9.379  9.345  9.398  9.36
1 10.676]
```

In [11]:
```python
b = MultinomialNB(alpha=float(optimal_k))#Using Multinomial nb for best alpha
s= b.fit(X_trvec, y_tr)#fitting this for  training data
b.get_params(deep=True)#function for getting parameters
```

Out[11]: {'alpha': 0.01, 'class_prior': None, 'fit_prior': True}

In [12]:
```python
pred = b.predict(X_trvec)
acc = accuracy_score(y_tr, pred) * 100
print('\nThe accuracy of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, acc))
print('\nThe Train error of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, 100-a
cc))
```

```
The accuracy of the MultinomialNB classifier for alpha = 0.010000 is 91.935668%

The Train error of the MultinomialNB classifier for alpha = 0.010000 is 8.064332%
```

In [13]:
```python
# predict the response
pred = b.predict(X_testvec)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, acc))
print('\nThe Test error of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, 100-ac
c))
```

```
The accuracy of the MultinomialNB classifier for alpha = 0.010000 is 90.673526%

The Test error of the MultinomialNB classifier for alpha = 0.010000 is 9.326474%
```

Feature Importance

Usingfeature_count and feature_log_prob

In [360]:
```python
pos_sort=sorted(b.feature_count_[1],reverse=True)
neg_sort=sorted(b.feature_count_[0],reverse=True)
```

In [361]:
```python
print("pos imp features=",pos_sort[0:20])
print("neg imp features=",neg_sort[0:20])
```

```
pos imp features= [124542.0, 110398.0, 99238.0, 99167.0, 97297.0, 90758.0, 86462.0, 83395.0, 789
17.0, 76770.0, 75838.0, 73907.0, 69339.0, 64100.0, 61227.0, 48954.0, 48474.0, 48174.0, 47495.0,
46075.0]
neg imp features= [29267.0, 28790.0, 23962.0, 18308.0, 17614.0, 16123.0, 15799.0, 15416.0, 1392
5.0, 13357.0, 13059.0, 12263.0, 12226.0, 11041.0, 10661.0, 10652.0, 10619.0, 10575.0, 10144.0, 9
443.0]
```

In [362]:
```python
pos_sort=sorted(b.feature_log_prob_[1],reverse=True)
neg_sort=sorted(b.feature_log_prob_[0],reverse=True)
```

In [363]:
```python
print("pos imp features=",pos_sort[0:20])
print("neg imp features=",neg_sort[0:20])
```

```
pos imp features= [-4.398341457443124, -4.518891419412563, -4.625461413052712, -4.62617711363818
2, -4.645214061247401, -4.714784853462213, -4.7632759184796996, -4.7993921486597095, -4.85458315
65288236, -4.882165530566965, -4.894379823037694, -4.9201714187506145, -4.983970554986643, -5.06
25325308145595, -5.10838789401406, -5.3320909864374855, -5.341944294114805, -5.34815228094309, -
5.362346998275147, -5.39270028817926]
neg imp features= [-4.303485471263084, -4.319917401981268, -4.503469266194436, -4.77258749805440
1, -4.811229428491666, -4.899671395029948, -4.919970300251947, -4.944509443606666, -5.0462226437
64512, -5.0878647839814235, -5.110426116436583, -5.173312098715355, -5.176333619122815, -5.27827
4056423015, -5.3132942218842985, -5.314138697655807, -5.317241224471076, -5.3213929571755365, -
5.3629992670070905, -5.434600621061602]
```

In [30]:
```python
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.feature_log_prob_[0], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.feature_log_prob_[1], feature_names),reverse=True)[:n]
    print("Important words in negative reviews")
    for coef, feat in topn_class1:
        print(class_labels[0], coef, feat)
    print("-----------------------------------------")
    print("Important words in positive reviews")
    for coef, feat in topn_class2:
        print(class_labels[1], coef, feat)
```

In [31]:
```python
important_features(vec,b,n=20)
```

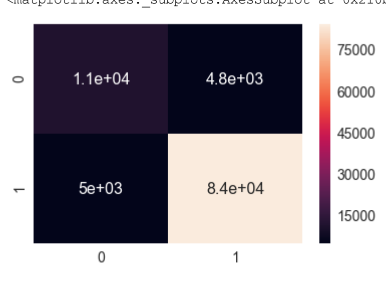
```
Important words in negative reviews
negative -4.27556296546887 tast
negative -4.29421328285664 like
negative -4.48669351695775 product
negative -4.746719339750831 one
negative -4.778228474312305 flavor
negative -4.879439650113969 would
negative -4.8940760402729335 tri
negative -4.918221422723155 food
negative -5.00124508943559 coffe
negative -5.065534715895948 good
negative -5.093566082788707 use
negative -5.140619595700686 buy
negative -5.156699592760521 get
negative -5.254291562316126 order
negative -5.270294815464419 dog
negative -5.276582975366738 dont
negative -5.286558343615864 eat
negative -5.304902259709062 tea
negative -5.343920645972659 even
negative -5.406568980595022 bag
-----------------------------------------
Important words in positive reviews
positive -4.39361154973788 like
positive -4.510939642865598 tast
positive -4.6181432996785325 love
positive -4.6197777813836485 flavor
positive -4.639747031653936 good
positive -4.710455799791102 great
positive -4.752500232887623 one
positive -4.7920503162302985 use
positive -4.851113668895646 coffe
positive -4.878136236364986 tri
positive -4.889298517753657 product
positive -4.918299272041327 tea
positive -4.9778832425218 food
positive -5.052962881411771 get
positive -5.106293281507648 make
positive -5.324349724336699 would
positive -5.339353244131697 dog
positive -5.339477668642575 eat
positive -5.3538276314724556 time
positive -5.38318216487963 buy
```

It consists 4 things • True Positive (TP) : Observation is positive, and is predicted to be positive.[0,0] • False Negative (FN) : Observation is positive, but is predicted negative.[0,1] • True Negative (TN) : Observation is negative, and is predicted to be negative.[1,1] • False Positive (FP) : Observation is negative, but is predicted positive.[1,0] indexes If the data is imbalanced , then we will use these things as accuracy is not a correct measure and we will also have other metrix to measure using confusion matrix like 1)precision = TP/TP+FP 2)Recall = TP/TP+FN 3)F1 score = 2*recall*precision/recall + precision These can also be calculated directly from sklearn...

In [32]:
```python
pred = b.predict(X_testvec)
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})
```

```
[[11345  4809]
 [ 4999 84010]]
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x210bd5a39b0>



In [33]:
```python
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 11345
FP Value is 4999
TN Value is 84010
FN Value is 4809
TPR Value is 0.6941385217816936
FPR Value is 0.3058614782183064
TNR Value is 0.9458561794210698
FNR Value is 0.054143820578930184
```

In [34]:
```python
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(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9067352585985565
0.8215211966174236
0.8199973506013817
0.8230699876377905
```

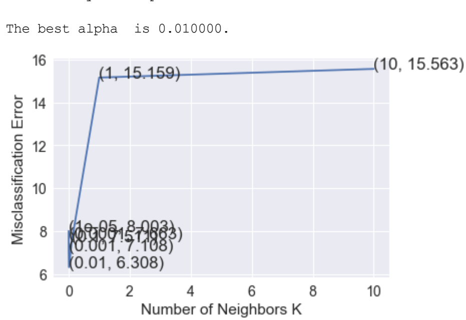
TF_IDF

In [35]:
```python
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit(X_tr['CleanedText'].values)
```

In [36]:
```python
X_tr_tf_idf = final_tf_idf.transform(X_tr['CleanedText'].values)
X_cv_tf_idf = final_tf_idf.transform(X_cv['CleanedText'].values)
X_test_tf_idf = final_tf_idf.transform(X_test['CleanedText'].values)
```

In [37]:
```python
#[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
from sklearn.naive_bayes import MultinomialNB
listacc=[]

alpha=   [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]
for i in alpha:
    b = MultinomialNB(alpha=float(i))

    # fitting the model on crossvalidation train
    b.fit(X_tr_tf_idf,y_tr)

    # predict the response on the crossvalidation train
    pred = b.predict(X_cv_tf_idf)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for alpha = %f is %d%%' % (i, acc))
    listacc.append(acc)
MSE = [100 - x for x in listacc]

# determining best k
optimal_k = alpha[MSE.index(min(MSE))]
print('\nThe best alpha  is %f.' % optimal_k)

plt.plot(alpha, MSE)

for xy in zip(alpha,np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each alpha is : ", np.round(MSE,3))
```

```
CV accuracy for alpha = 0.000010 is 91%

CV accuracy for alpha = 0.000100 is 92%

CV accuracy for alpha = 0.001000 is 92%

CV accuracy for alpha = 0.010000 is 93%

CV accuracy for alpha = 0.100000 is 92%

CV accuracy for alpha = 1.000000 is 84%

CV accuracy for alpha = 10.000000 is 84%

The best alpha  is 0.010000.
```




```
the misclassification error for each alpha is :  [ 8.003  7.663  7.108  6.308  7.511 15.159 15.5
63]
```

In [38]:
```python
b = MultinomialNB(alpha=float(optimal_k))
s= b.fit(X_tr_tf_idf, y_tr)
```

In [39]:
```python
pred = b.predict(X_tr_tf_idf)
acc = accuracy_score(y_tr, pred) * 100
print('\nThe accuracy of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, acc))
print('\nThe Train error of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, 100-a
cc))
```

```
The accuracy of the MultinomialNB classifier for alpha = 0.010000 is 99.668133%

The Train error of the MultinomialNB classifier for alpha = 0.010000 is 0.331867%
```

In [40]:
```python
# predict the response
pred = b.predict(X_test_tf_idf)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, acc))
print('\nThe Test error of the MultinomialNB classifier for alpha = %f is %f%%' % (optimal_k, 100-ac
c))
```

```
The accuracy of the MultinomialNB classifier for alpha = 0.010000 is 93.713568%

The Test error of the MultinomialNB classifier for alpha = 0.010000 is 6.286432%
```

In [41]:
```python
pos_sort=sorted(b.feature_count_[1],reverse=True)
neg_sort=sorted(b.feature_count_[0],reverse=True)
```

In [42]:
```python
print("pos imp features=",pos_sort[0:20])
print("neg imp features=",neg_sort[0:20])
```

```
pos imp features= [3899.5630239897587, 3778.4641511552327, 3698.1179378571205, 3619.589106082303
7, 3562.19168801116, 3508.4901957107027, 3461.7097065916832, 3455.7424292521887, 2873.6047717889
82, 2813.33194474528, 2798.7623799430457, 2595.9831458121234, 2497.6993578884644, 2341.364203695
1674, 2280.7247700876837, 2233.4620419798143, 2095.512529918485, 2076.205390650043, 2053.3290900
47975, 1960.8216508872604]
neg imp features= [914.8143085235572, 819.7419762281842, 765.5311024365781, 605.1826232990338, 5
74.3568868837139, 568.2095246099273, 544.597430009016, 490.4498739539403, 488.5440580878693, 47
4.78053989029814, 458.50031047692113, 458.40336751333814, 440.52947982657093, 432.6730241352027
5, 417.2315060876149, 412.9681753527209, 405.49315630613063, 396.89312075112724, 386.27978569586
526, 383.5575068171069]
```

In [43]:
```python
pos_sort=sorted(b.feature_log_prob_[1],reverse=True)
neg_sort=sorted(b.feature_log_prob_[0],reverse=True)
```

In [44]:
```python
print("pos imp features=",pos_sort[0:20])
print("neg imp features=",neg_sort[0:20])
```

```
pos imp features= [-6.176856987445463, -6.208403789137492, -6.229897324692791, -6.25136077803038
15, -6.267345248169278, -6.282535402067234, -6.295958562939207, -6.2976838402377595, -6.48215331
3930014, -6.503350972695824, -6.508543168253829, -6.583754895009042, -6.622349998549658, -6.6869
86030026815, -6.7132263949276885, -6.734166709409974, -6.797921113662427, -6.80717734091601, -6.
81825676109113, -6.864355358634456]
neg imp features= [-6.047199684144144, -6.156929891949123, -6.225348811202625, -6.46038492546520
3, -6.512663350550992, -6.523423887396735, -6.565866495236424, -6.670588249195231, -6.6744815916
92364, -6.703057980312967, -6.737948939587884, -6.738160392222976, -6.777931652048243, -6.795926
304391397, -6.832266512995674, -6.84253697105788, -6.860803061176245, -6.882239495867597, -6.909
343890495197, -6.916416084917896]
```

In [45]:
```python
#We can use the function which we made previously
important_features(final_tf_idf,b,n=20)
```

```
Important words in negative reviews
negative -6.047199684144144 tast
negative -6.156929891949123 like
negative -6.225348811202625 product
negative -6.460384925465203 coffe
negative -6.512663350550992 flavor
negative -6.523423887396735 would
negative -6.565866495236424 one
negative -6.670588249195231 buy
negative -6.674481591692364 tri
negative -6.703057980312967 order
negative -6.737948939587884 tea
negative -6.738160392222976 food
negative -6.777931652048243 box
negative -6.795926304391397 dog
negative -6.832266512995674 dont
negative -6.84253697105788 good
negative -6.860803061176245 disappoint
negative -6.882239495867597 get
negative -6.909343890495197 bag
negative -6.916416084917896 use
-----------------------------------------
Important words in positive reviews
positive -6.176856987445463 love
positive -6.208403789137492 great
positive -6.229897324692791 coffe
positive -6.2513607780303815 like
positive -6.267345248169278 tast
positive -6.282535402067234 good
positive -6.295958562939207 tea
positive -6.2976838402377595 flavor
positive -6.482153313930014 product
positive -6.503350972695824 use
positive -6.508543168253829 one
positive -6.583754895009042 tri
positive -6.622349998549658 food
positive -6.686986030026815 dog
positive -6.7132263949276885 get
positive -6.734166709409974 make
positive -6.797921113662427 price
positive -6.80717734091601 best
positive -6.81825676109113 buy
positive -6.864355358634456 realli
```

Different Parameters in MultinomialNB

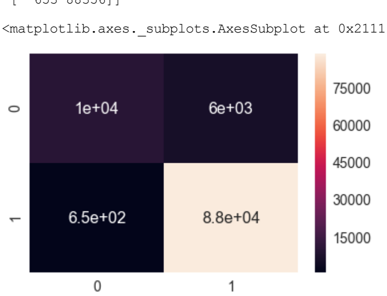
In [46]:
```python
#Smoothed empirical log probability for each class.
print(b.class_log_prior_)
#Mirrors class_log_prior_ for interpreting MultinomialNB as a linear model.
print(b.intercept_)
#Empirical log probability of features given a class, P(x_i|y).
print(b.feature_log_prob_)
#Mirrors feature_log_prob_ for interpreting MultinomialNB as a linear model
print(b.coef_)

print(b.class_count_)
print(b.classes_)
```

```
[-1.8518248  -0.17072961]
[-0.17072961]
[[-17.4711019  -17.4711019  -17.4711019  ... -17.4711019  -17.4711019
  -17.4711019 ]
 [-15.93113747 -16.03113027 -17.70201943 ... -15.22673249 -15.79780411
  -15.79780411]]
[[-15.93113747 -16.03113027 -17.70201943 ... -15.22673249 -15.79780411
  -15.79780411]]
[ 49516. 265972.]
['negative' 'positive']
```

In [47]:
```python
pred = b.predict(X_test_tf_idf)
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})
```

```
[[10196  5958]
 [  653 88356]]
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x2111f036d30>



In [48]:
```python
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 10196
FP Value is 653
TN Value is 88356
FN Value is 5958
TPR Value is 0.939810120748456
```