Assignment3_new

October 22, 2018

In [56]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pickle,pprint

import sqlite3
import pandas as pd
import numpy as np
import nltk
import prettytable

import re

import pickle
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec

from sklearn.neighbors import KNeighborsClassifier
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.metrics import confusion_matrix,f1_score,make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.decomposition import TruncatedSVD
```

In [2]:
```
pkl_file = open('data.pkl', 'rb')
filtered_data = pickle.load(pkl_file)
#pprint.pprint(filtered_data)
pkl_file.close()
```

In [3]:
```
filtered_data.columns
```

```
Out[3]: Index(['ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
               'HelpfulnessDenominator', 'Time', 'Summary', 'Text', 'Sentiment',
               'CleanedText'],
              dtype='object')
```

```
In [4]: filtered_data.sort_values('Time',inplace=True)
```

```
In [5]: filtered_10k=filtered_data.iloc[:100000,:]
```

```
In [6]: filtered_10k.Time.head(20)
```

```
Out[6]: 61682     948672000
        1130      961718400
        1129      962236800
        26503    1067040000
        26502    1067040000
        54376    1067472000
        35508    1067558400
        35655    1067644800
        35654    1067904000
        10388    1067990400
        82577    1068076800
        26501    1068076800
        81533    1068422400
        55130    1068940800
        98503    1069027200
        75316    1069113600
        36417    1069459200
        44110    1070668800
        22710    1071100800
        74752    1071705600
        Name: Time, dtype: int64
```

```
In [7]: filtered_10k.reset_index(inplace=True)
```

```
In [8]: label=filtered_10k.Sentiment
```

```
In [9]: filtered_10k.drop('Sentiment',axis=1,inplace=True)
```

```
In [10]: filtered_10k.head()
```

```
Out[10]:    index   ProductId          UserId    ProfileName  HelpfulnessNumerator  \
         0  61682  B00002N8SM  A32DW342WBJ6BX    Buttersugar                     0
         1   1130  B00002Z754  A29Z5PI9BW2PU3         Robbie                     7
         2   1129  B00002Z754  A3B8RCEI0FXFI6      B G Chase                    10
         3  26503  B00008RCMI  A284C7M23F0APC     A. Mendoza                     0
         4  26502  B00008RCMI  A19E94CF5O1LY7  Andrew Arnold                     0

            HelpfulnessDenominator        Time  \
```

```
0                     0   948672000
1                     7   961718400
2                    10   962236800
3                     0  1067040000
4                     0  1067040000

                                     Summary  \
0                      A sure death for flies
1                               Great Product
2                 WOW Make your own 'slickers' !
3                       Best sugarless gum ever!
4  I've chewed this gum many times, but used?

                                                Text  \
0  I bought a few of these after my apartment was...
1  This was a really good idea and the final prod...
2  I just received my shipment and could hardly w...
3  I love this stuff. It is sugar-free so it does...
4  Nothing against the product, but it does bothe...

                                         CleanedText
0  bought apart infest fruit fli hour trap mani f...
1  realli good idea final product outstand use de...
2  receiv shipment could hard wait tri product lo...
3        love stuff rot gum tast good go buy gum get
4  noth product bother link top page buy use chew...
```

In [11]:
```python
filtered_10k.drop('index',axis=1,inplace=True)
```

In [12]:
```python
filtered_10k.head()
```

Out[12]:
```
    ProductId          UserId     ProfileName  HelpfulnessNumerator  \
0  B00002N8SM  A32DW342WBJ6BX     Buttersugar                     0
1  B00002Z754  A29Z5PI9BW2PU3          Robbie                     7
2  B00002Z754  A3B8RCEI0FXFI6       B G Chase                    10
3  B00008RCMI  A284C7M23F0APC     A. Mendoza                      0
4  B00008RCMI  A19E94CF5O1LY7  Andrew Arnold                     0

   HelpfulnessDenominator        Time  \
0                       0   948672000
1                       7   961718400
2                      10   962236800
3                       0  1067040000
4                       0  1067040000

                                     Summary  \
0                      A sure death for flies
1                               Great Product
```

```
2              WOW Make your own 'slickers' !
3                    Best sugarless gum ever!
4  I've chewed this gum many times, but used?

                                               Text  \
0  I bought a few of these after my apartment was...
1  This was a really good idea and the final prod...
2  I just received my shipment and could hardly w...
3  I love this stuff. It is sugar-free so it does...
4  Nothing against the product, but it does bothe...

                                        CleanedText
0  bought apart infest fruit fli hour trap mani f...
1  realli good idea final product outstand use de...
2  receiv shipment could hard wait tri product lo...
3        love stuff rot gum tast good go buy gum get
4  noth product bother link top page buy use chew...
```

0.0.1 KNN

```
In [13]: X_1, X_test, y_1, y_test = train_test_split(filtered_10k, label, test_size=0.2, randor
         X_tr, X_cv, y_tr, y_cv = train_test_split(X_1,y_1, test_size=0.25, random_state=50)
```

```
In [15]: print(X_1.shape,len(y_1))
         print(X_tr.shape,len(y_tr))
         print(X_test.shape,len(y_tr))
```

```
(80000, 9) 80000
(60000, 9) 60000
(20000, 9) 60000
```

0.0.2 Bag Of Words

```
In [16]: count_vector=CountVectorizer(ngram_range=(1,2))
         vocab=count_vector.fit(X_tr.CleanedText)

         bow_train=count_vector.transform(X_tr.CleanedText)
         bow_cv=count_vector.transform(X_cv.CleanedText)
         bow_test=count_vector.transform(X_test.CleanedText)
```

```
In [17]: print(bow_train.get_shape())
         print(bow_cv.get_shape())
         print(bow_test.get_shape())
```

```
(60000, 837824)
(20000, 837824)
(20000, 837824)
```

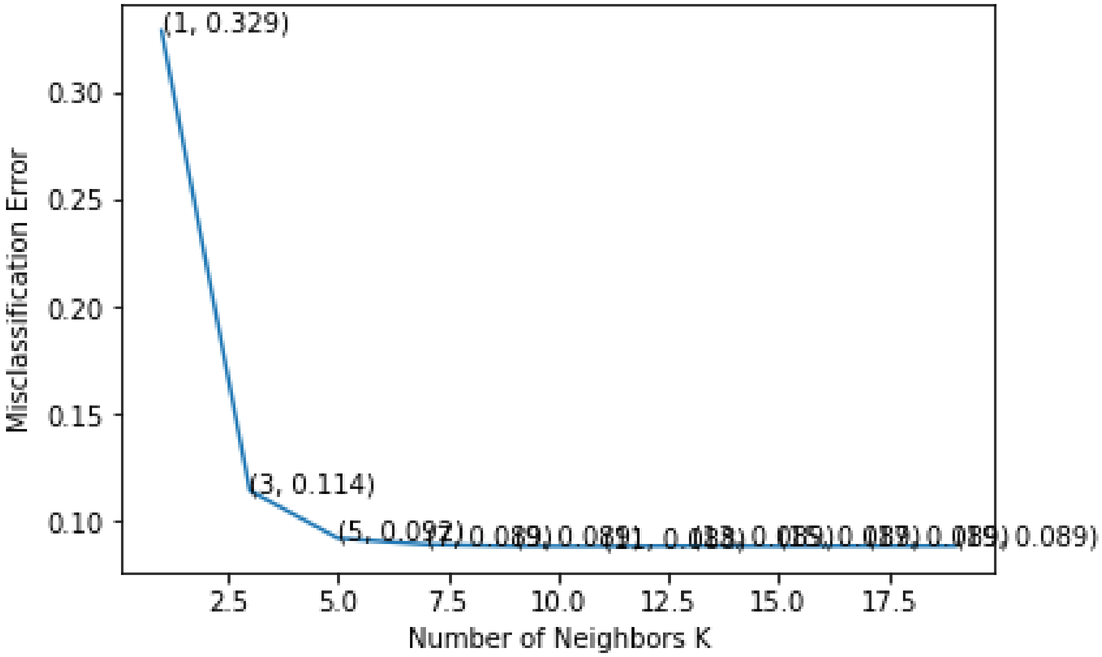
```
In [22]: summary_table = prettytable.PrettyTable(["Method", "Algorithm", "Optimam K", "F-1 Sco

         def fit_and_val(X_train,y_train,X_cv,y_cv,method,algo='brute',table=summary_table):

             nbrs_list=list(range(1,20,2))
             cv_scores=[]
             f1_scorer = make_scorer(f1_score, pos_label="Positive")
             if algo == 'kd_tree':
                     svd = TruncatedSVD()
                     svd.fit_transform(X_train)
                     X_train = svd.fit_transform(X_train)
                     X_cv = svd.fit_transform(X_cv)
             for k in nbrs_list:
                 model=KNeighborsClassifier(n_neighbors=k,algorithm=algo)
                 model.fit(X_train,y_train)
                 scores=cross_val_score(model, X_train, y_train, cv=5, scoring=f1_scorer)
                 print("\nFor n_neighbors =",str(k))
                 print ("\nmodel accuracy is {} %".format(scores.mean()*100))
                 cv_scores.append(scores.mean())
             MSE = [1 - x for x in cv_scores]
             optimal_k = nbrs_list[MSE.index(min(MSE))]
             print('The optimal number of neighbors is %d.' % optimal_k)


             plt.plot(nbrs_list, MSE)

             for xy in zip(nbrs_list, 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 k value is : ", np.round(MSE,3))

             knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

         # fitting the model
             knn_optimal.fit(X_train, y_train)

         # predict the response
             pred = knn_optimal.predict(X_cv)
             f1=f1_score(y_pred=pred,y_true=y_cv,pos_label='Positive')
             score=f1*100

             table.add_row([method,algo,optimal_k,score])
         # evaluate accuracy
```

```
        cnf_mtrx=confusion_matrix(y_cv, pred, labels=None, sample_weight=None)
        print(cnf_mtrx)
        sns.heatmap(cnf_mtrx,annot=True,cmap='Blues', fmt='g')
        print (classification_report(y_cv, pred))
        return optimal_k,score
```

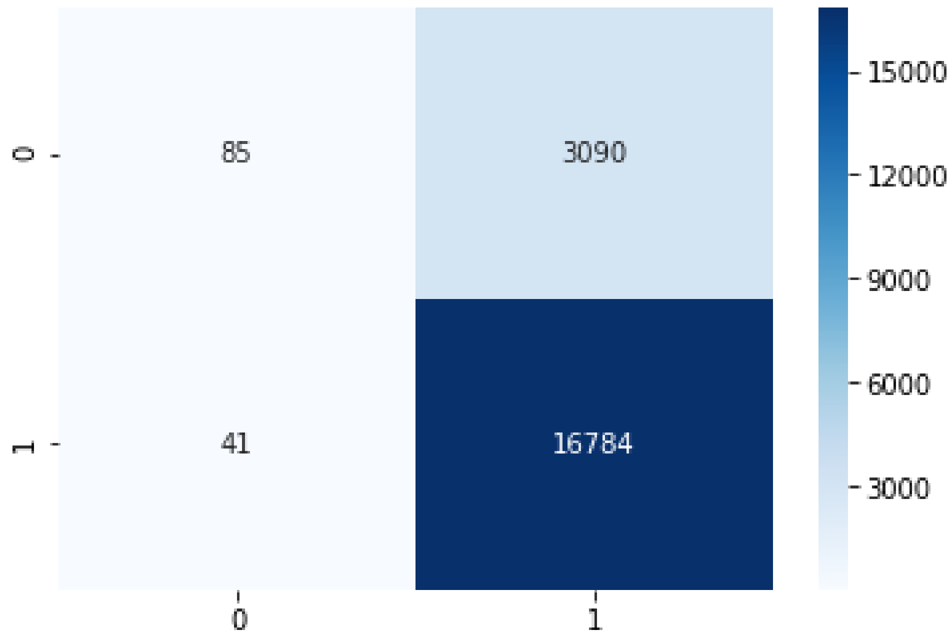
```
In [24]: bow_opt_brute,bow_tr_f1_brute=fit_and_val(bow_train,y_tr,bow_cv,y_cv,"Bag Of Word",'b
```

```
For n_neighbors = 1

model accuracy is 67.10193295649964 %

For n_neighbors = 3

model accuracy is 88.56004621350169 %

For n_neighbors = 5

model accuracy is 90.78190382458132 %

For n_neighbors = 7

model accuracy is 91.06938056694356 %

For n_neighbors = 9

model accuracy is 91.14377541413788 %

For n_neighbors = 11

model accuracy is 91.1530689964144 %

For n_neighbors = 13

model accuracy is 91.12989187316266 %

For n_neighbors = 15

model accuracy is 91.14513192026493 %

For n_neighbors = 17

model accuracy is 91.13217458711904 %

For n_neighbors = 19
```

```
model accuracy is 91.13525729332514 %
The optimal number of neighbors is 11.
```



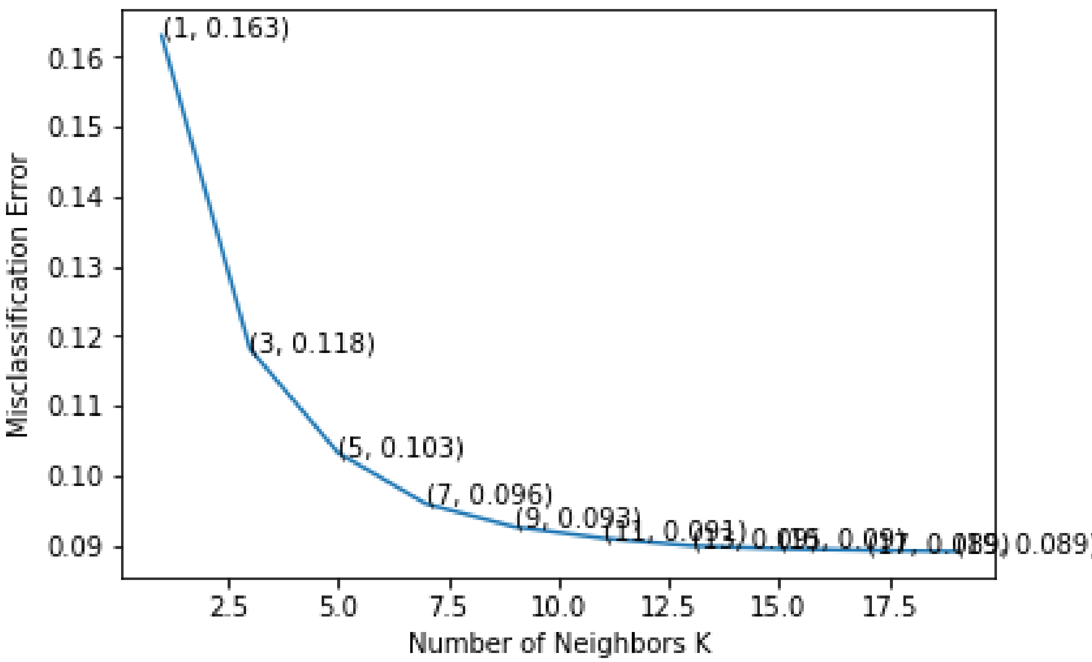

```
the misclassification error for each k value is :  [0.329 0.114 0.092 0.089 0.089 0.088 0.089 0
[[   85  3090]
 [   41 16784]]
             precision    recall  f1-score   support

   Negative       0.67      0.03      0.05      3175
   Positive       0.84      1.00      0.91     16825

avg / total       0.82      0.84      0.78     20000
```



In [27]: bow_opt_kd,bow_tr_f1_kd=fit_and_val(bow_train,y_tr,bow_cv,y_cv,"Bag Of Word",'kd_tree

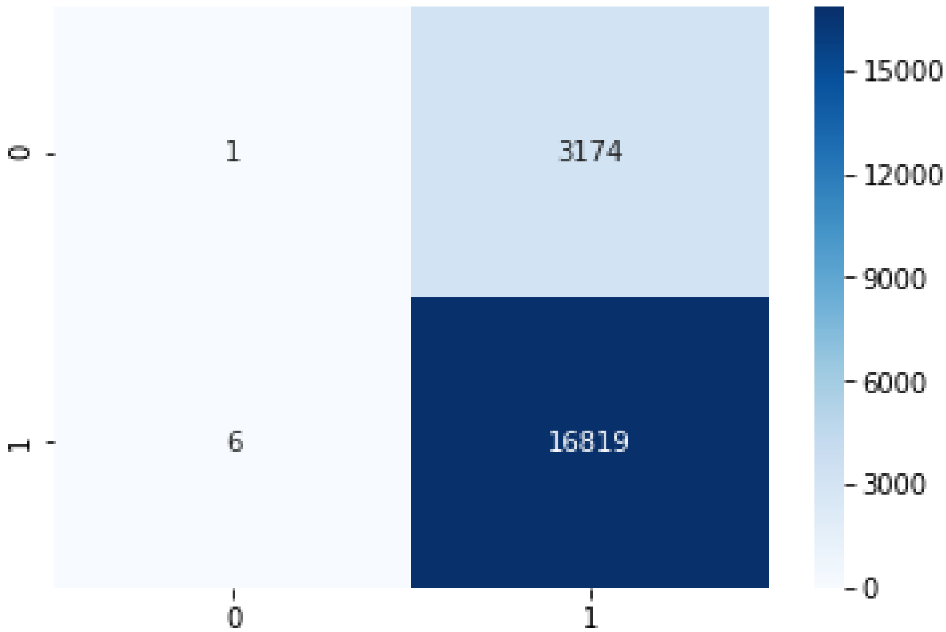
```
For n_neighbors = 1

model accuracy is 83.70394030281325 %

For n_neighbors = 3

model accuracy is 88.18528487838077 %

For n_neighbors = 5

model accuracy is 89.67437236543341 %

For n_neighbors = 7

model accuracy is 90.40687458705446 %

For n_neighbors = 9

model accuracy is 90.73377611288242 %

For n_neighbors = 11

model accuracy is 90.89618453051689 %
```

```
For n_neighbors = 13

model accuracy is 91.00361735319093 %

For n_neighbors = 15

model accuracy is 91.045818656938 %

For n_neighbors = 17

model accuracy is 91.06756840776045 %

For n_neighbors = 19

model accuracy is 91.08222347499047 %
The optimal number of neighbors is 19.
```




```
the misclassification error for each k value is :  [0.163 0.118 0.103 0.096 0.093 0.091 0.09
[[    1  3174]
 [    6 16819]]
             precision    recall  f1-score   support

   Negative       0.14      0.00      0.00      3175
```

```
   Positive       0.84      1.00      0.91     16825

avg / total       0.73      0.84      0.77     20000
```




```
In [24]: print(bow_opt_kd,bow_tr_err_kd,bow_cv_err_kd)
```

```
19 0.164 15.900000000000006
```

0.0.3 TF-IDF intialization and dimension creation

```
In [34]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
         filtered_tf_idf = tf_idf_vect.fit(X_tr.CleanedText)
         print("the type of count vectorizer ",type(filtered_tf_idf))
         #print("the shape of out TF-IDF vectorizer ",filtered_tf_idf.get_shape())
```

```
the type of count vectorizer  <class 'sklearn.feature_extraction.text.TfidfVectorizer'>
```

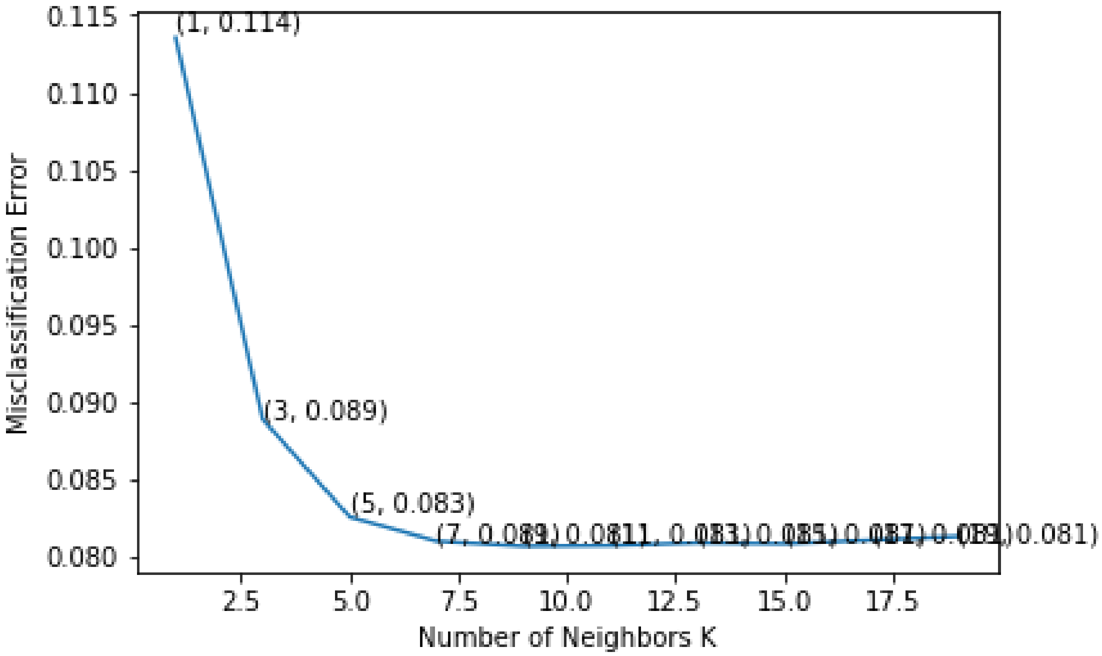
```
In [30]: tf_idf_train=tf_idf_vect.transform(X_tr.CleanedText)
         tf_idf_cv=tf_idf_vect.transform(X_cv.CleanedText)
         tf_idf_test=tf_idf_vect.transform(X_test.CleanedText)
```

```
In [31]: print(tf_idf_train.get_shape())
         print(tf_idf_cv.get_shape())
         print(tf_idf_test.get_shape())
```

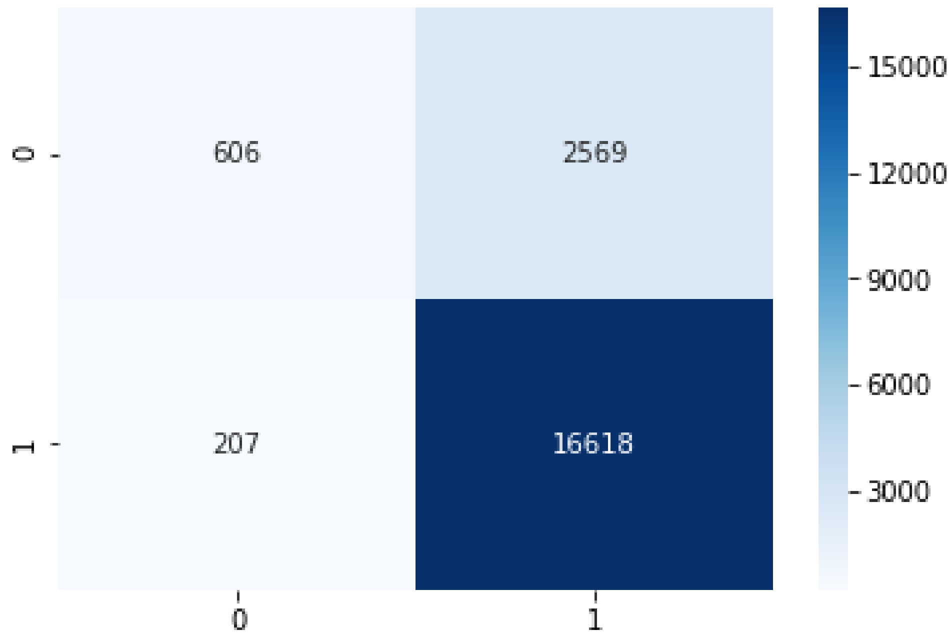
```
(60000, 837824)
(20000, 837824)
(20000, 837824)
```

```
In [32]: tf_idf_opt_brute,tf_idf_tr_f1_brute=fit_and_val(tf_idf_train,y_tr,tf_idf_cv,y_cv,"TF-I
```

```
For n_neighbors = 1

model accuracy is 88.64580437265674 %

For n_neighbors = 3

model accuracy is 91.10975537047678 %

For n_neighbors = 5

model accuracy is 91.74049631770733 %

For n_neighbors = 7

model accuracy is 91.89699174189914 %

For n_neighbors = 9

model accuracy is 91.93062365409513 %

For n_neighbors = 11

model accuracy is 91.92581733328937 %

For n_neighbors = 13

model accuracy is 91.90850507311943 %

For n_neighbors = 15

model accuracy is 91.9140797376684 %

For n_neighbors = 17

model accuracy is 91.89020476083523 %

For n_neighbors = 19

model accuracy is 91.8666565308003 %
The optimal number of neighbors is 9.
```



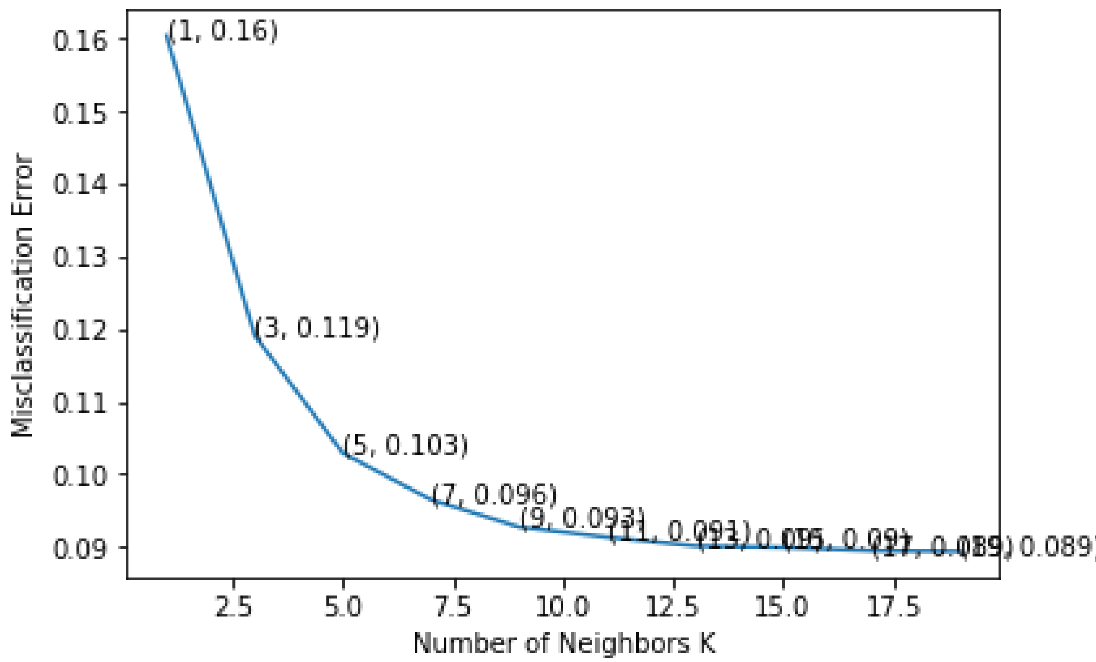

```
the misclassification error for each k value is :  [0.114 0.089 0.083 0.081 0.081 0.081 0.081 0
[[  606  2569]
 [  207 16618]]
             precision    recall  f1-score   support

   Negative       0.75      0.19      0.30      3175
   Positive       0.87      0.99      0.92     16825

avg / total       0.85      0.86      0.82     20000
```




```
In [35]: tf_idf_opt_kd,tf_idf_tr_f1_kd=fit_and_val(tf_idf_train,y_tr,tf_idf_cv,y_cv,"TF-IDF",'I
```

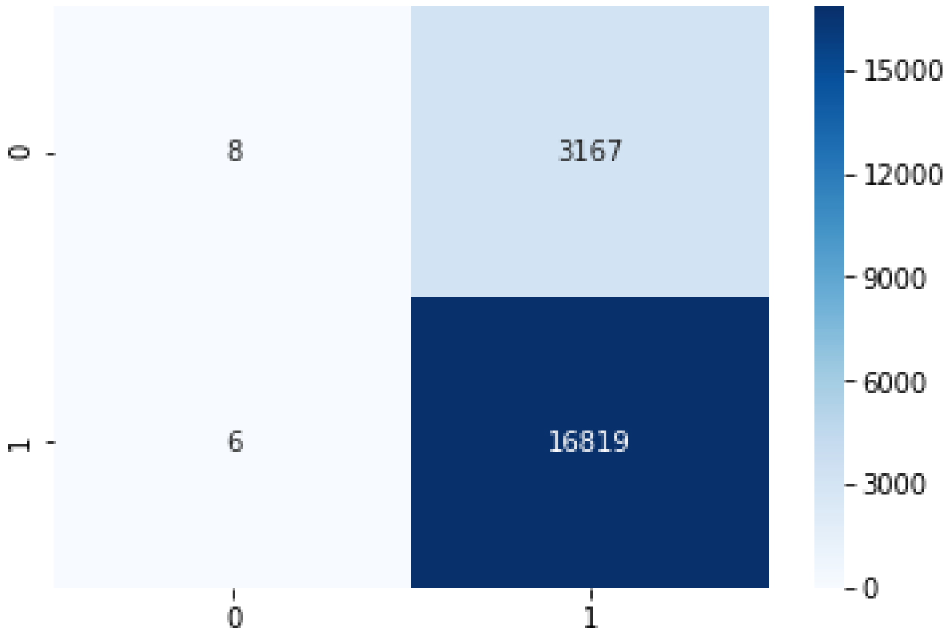
```
For n_neighbors = 1

model accuracy is 83.95793639129569 %

For n_neighbors = 3

model accuracy is 88.08999333436716 %

For n_neighbors = 5

model accuracy is 89.70939461124432 %

For n_neighbors = 7

model accuracy is 90.35278105799452 %

For n_neighbors = 9

model accuracy is 90.7289476729766 %

For n_neighbors = 11

model accuracy is 90.86964431711625 %
```

```
For n_neighbors = 13

model accuracy is 90.98896562054732 %

For n_neighbors = 15

model accuracy is 91.01056799696201 %

For n_neighbors = 17

model accuracy is 91.05398455482933 %

For n_neighbors = 19

model accuracy is 91.06219775615149 %
The optimal number of neighbors is 19.
```




```
the misclassification error for each k value is :  [0.16  0.119 0.103 0.096 0.093 0.091 0.09  0
[[    8  3167]
 [    6 16819]]
             precision    recall  f1-score   support

   Negative       0.57      0.00      0.01      3175
```

```
   Positive       0.84      1.00      0.91     16825

avg / total       0.80      0.84      0.77     20000
```



0.0.4 Wieghted Word2Vec

```
In [37]: list_of_sent_train=[]
         for sent in X_tr.CleanedText:
             list_of_sent_train.append(sent.split())

         list_of_sent_cv=[]
         for sent in X_cv.CleanedText:
             list_of_sent_cv.append(sent.split())


         list_of_sent_test=[]
         for sent in X_test.CleanedText:
             list_of_sent_test.append(sent.split())
```

```
In [38]: print(len(list_of_sent_train))
         print(list_of_sent_train[0])
```

```
60000
['must', 'south', 'east', 'product', 'never', 'saw', 'went', 'vacat', 'carolina', 'tri', 'look
```

In [39]:
```
w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

In [40]:
```
len(w2v_words)
```

Out[40]: 9501

In [41]:
```
def avg_w2v(X,word_vector,model):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in X: # 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/review
        for word in sent: # for each word in a review/sentence
            if word in word_vector:
                vec = model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    print(len(sent_vectors))
    print(len(sent_vectors[0]))
    return sent_vectors
```

In [42]:
```
sent_vectors_train=avg_w2v(list_of_sent_train,w2v_words,w2v_model)
sent_vectors_cv=avg_w2v(list_of_sent_cv,w2v_words,w2v_model)
sent_vectors_test=avg_w2v(list_of_sent_test,w2v_words,w2v_model)
```

```
60000
50
20000
50
20000
50
```

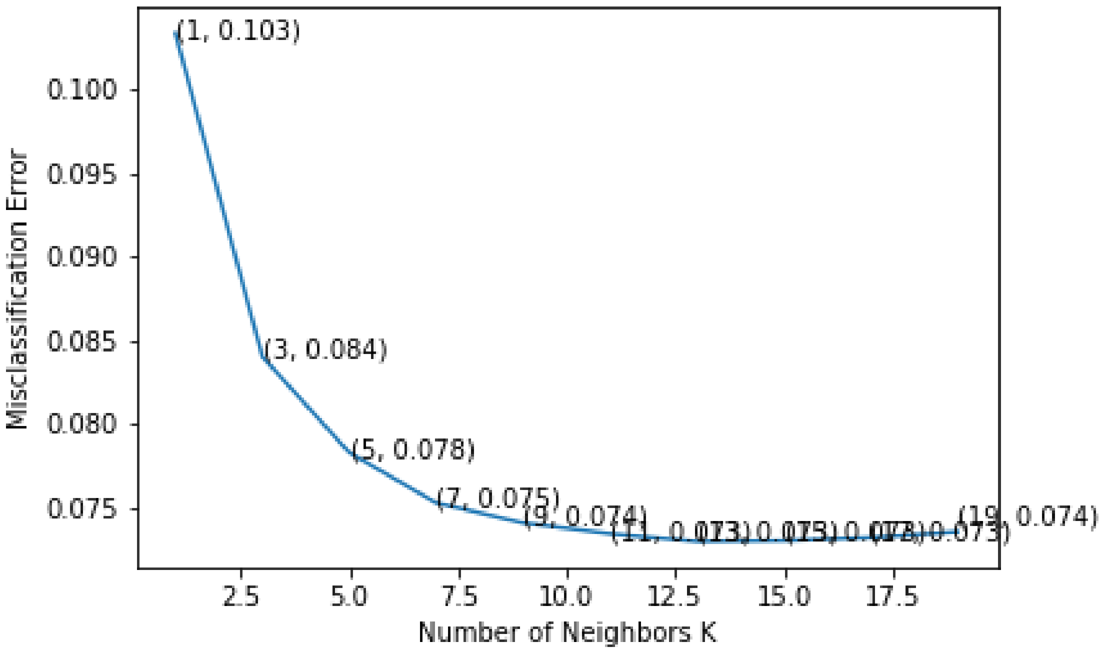
In [43]:
```
type(sent_vectors_train)
```

Out[43]: list

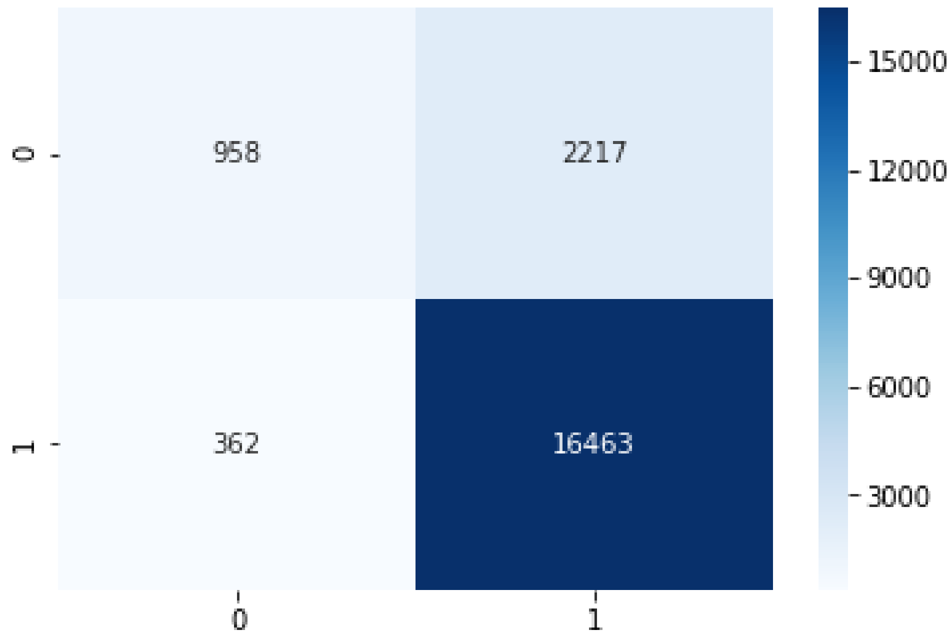
In [44]:
```
WW2v_opt_brute,WW2v_tr_f1_brute=fit_and_val(sent_vectors_train,y_tr,sent_vectors_cv,y_
```

```
For n_neighbors = 1

model accuracy is 89.66453416403442 %

For n_neighbors = 3
```

```
model accuracy is 91.60027019976759 %

For n_neighbors = 5

model accuracy is 92.17470757487752 %

For n_neighbors = 7

model accuracy is 92.47542600763859 %

For n_neighbors = 9

model accuracy is 92.59531619129714 %

For n_neighbors = 11

model accuracy is 92.65934313050191 %

For n_neighbors = 13

model accuracy is 92.70583589413184 %

For n_neighbors = 15

model accuracy is 92.70114031767194 %

For n_neighbors = 17

model accuracy is 92.68250131763003 %

For n_neighbors = 19

model accuracy is 92.64804382827018 %
The optimal number of neighbors is 13.
```



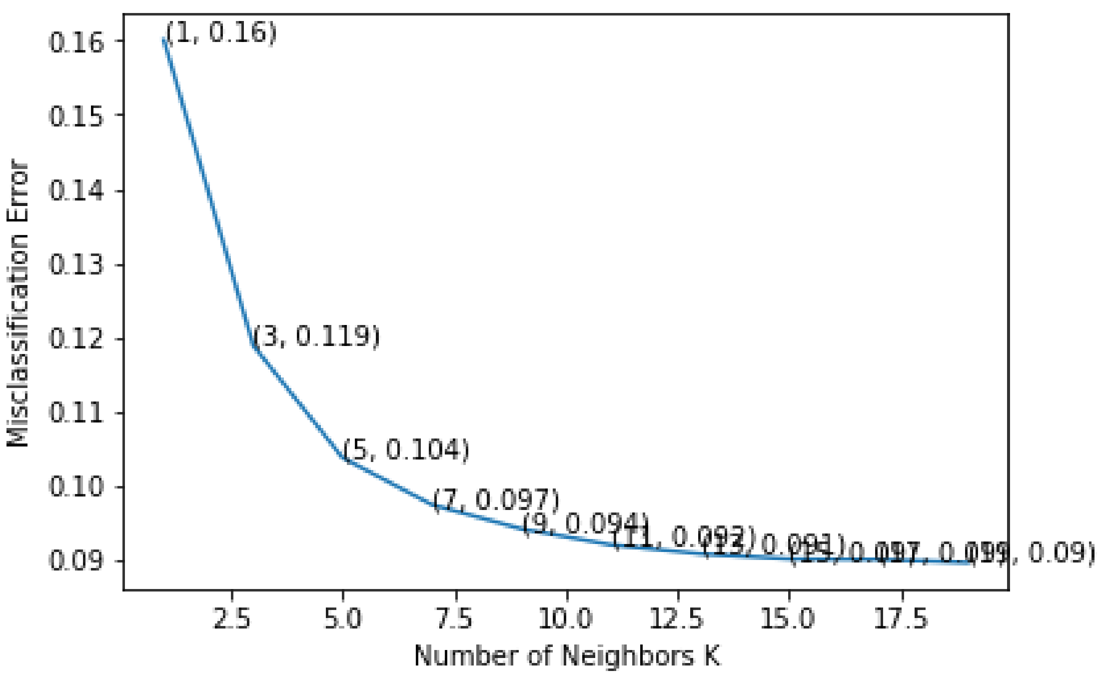

```
the misclassification error for each k value is :  [0.103 0.084 0.078 0.075 0.074 0.073 0.073 0
[[  958  2217]
 [  362 16463]]
             precision    recall  f1-score   support

   Negative       0.73      0.30      0.43      3175
   Positive       0.88      0.98      0.93     16825

avg / total       0.86      0.87      0.85     20000
```




```
In [46]: WW2v_opt_kd,WW2v_tr_f1_kd=fit_and_val(sent_vectors_train,y_tr,sent_vectors_cv,y_cv,"W
```

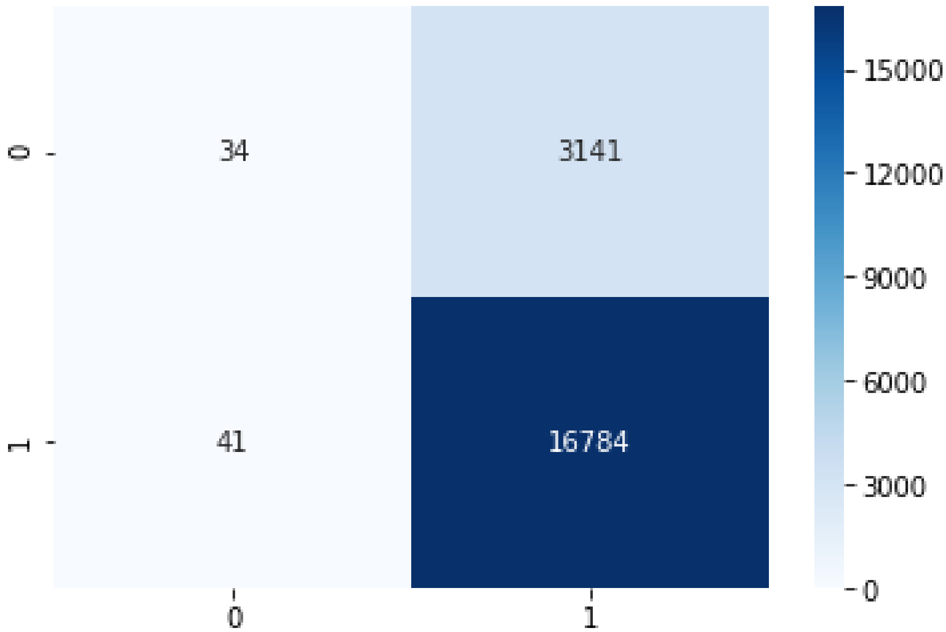
```
For n_neighbors = 1

model accuracy is 83.99531757835467 %

For n_neighbors = 3

model accuracy is 88.1064652626858 %

For n_neighbors = 5

model accuracy is 89.61779195947705 %

For n_neighbors = 7

model accuracy is 90.25009005986755 %

For n_neighbors = 9

model accuracy is 90.57504696193897 %

For n_neighbors = 11

model accuracy is 90.7892219395315 %
```

```
For n_neighbors = 13

model accuracy is 90.91189117542984 %

For n_neighbors = 15

model accuracy is 90.98025435041717 %

For n_neighbors = 17

model accuracy is 90.98716628249348 %

For n_neighbors = 19

model accuracy is 91.02812506288114 %
The optimal number of neighbors is 19.
```




```
the misclassification error for each k value is :  [0.16  0.119 0.104 0.097 0.094 0.092 0.091
[[   34  3141]
 [   41 16784]]
             precision    recall  f1-score   support

   Negative       0.45      0.01      0.02      3175
```

```
   Positive       0.84      1.00      0.91     16825

avg / total       0.78      0.84      0.77     20000
```



0.0.5 Wieghted TF-IDF

```
In [48]: """filtered_tf_idf=filtered_data.iloc[:20000,:]
         filtered_tf_idf.reset_index(inplace=True)

         label=filtered_tf_idf.Sentiment

         filtered_tf_idf.drop('Sentiment',axis=1,inplace=True)

         filtered_tf_idf.drop('index',axis=1,inplace=True)

         X_1, X_test, y_1, y_test = train_test_split(filtered_tf_idf, label, test_size=0.2, ra
         X_tr, X_cv, y_tr, y_cv = train_test_split(X_1,y_1, test_size=0.25, random_state=50)

         print(X_1.shape,len(y_1))
         print(X_tr.shape,len(y_tr))
         print(X_test.shape,len(y_tr))"""

         list_of_sent_train=[]
         for sent in X_tr.CleanedText:
```

```
        list_of_sent_train.append(sent.split())

    list_of_sent_cv=[]
    for sent in X_cv.CleanedText:
        list_of_sent_cv.append(sent.split())


    list_of_sent_test=[]
    for sent in X_test.CleanedText:
        list_of_sent_test.append(sent.split())

    w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=4)
    w2v_words = list(w2v_model.wv.vocab)

    len(w2v_words)
```

Out[48]: 9501

In [49]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
filtered_tf_idf = tf_idf_vect.fit(X_tr.CleanedText)
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))
print("the type of count vectorizer ",type(filtered_tf_idf))


tf_idf_train=tf_idf_vect.transform(X_tr.CleanedText)
tf_idf_cv=tf_idf_vect.transform(X_cv.CleanedText)
tf_idf_test=tf_idf_vect.transform(X_test.CleanedText)

print(tf_idf_train.get_shape())
print(tf_idf_cv.get_shape())
print(tf_idf_test.get_shape())
```

```
the type of count vectorizer  <class 'sklearn.feature_extraction.text.TfidfVectorizer'>
(60000, 837824)
(20000, 837824)
(20000, 837824)
```

In [50]:
```
# tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf


def w_tf_idf(X,word_vector,model,tfidf_feat,filtered_tf_idf):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
#    row=0;
    for sent in X: # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
```

```
                if word in word_vector:
                    vec = model.wv[word]
                # obtain the tf_idfidf of a word in a sentence/review
                    tf_idf = dictionary[word]*sent.count(word)
                    """tf_idf = filtered_tf_idf[row, tfidf_feat.index(word)]"""
                    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
        print(len(tfidf_sent_vectors))
        print(len(tfidf_sent_vectors[0]))
        return tfidf_sent_vectors
```

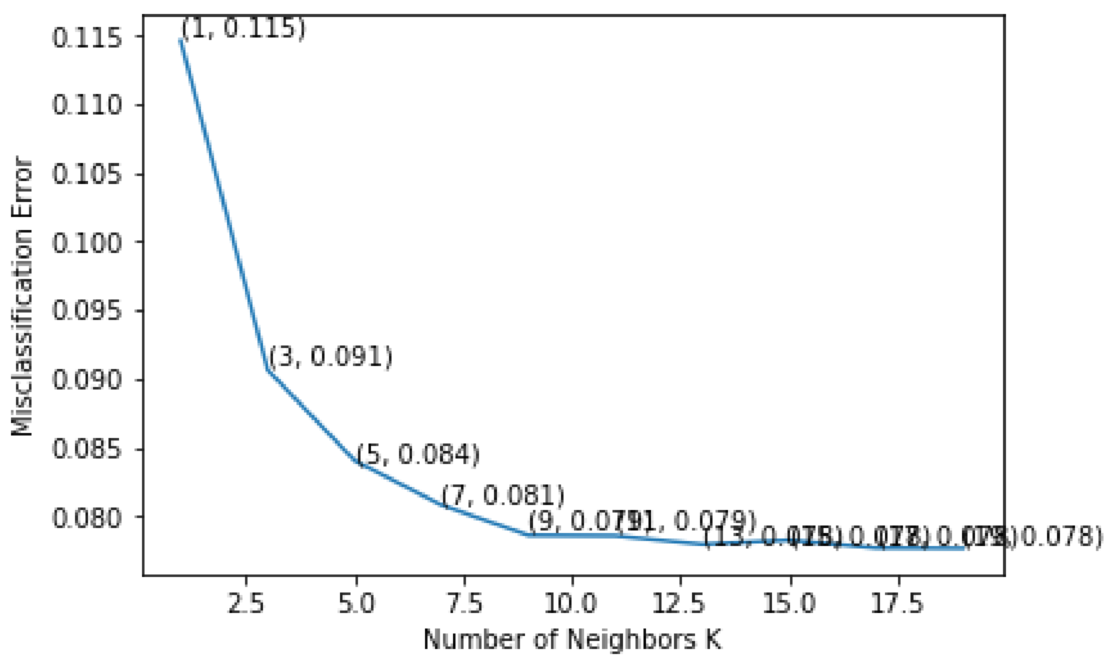
In [51]:
```
tfidf_feat = tf_idf_vect.get_feature_names()
tf_idf_sent_vectors_train=w_tf_idf(list_of_sent_train,w2v_words,w2v_model,tfidf_feat,t
tf_idf_sent_vectors_cv=w_tf_idf(list_of_sent_cv,w2v_words,w2v_model,tfidf_feat,tf_idf_
tf_idf_sent_vectors_test=w_tf_idf(list_of_sent_test,w2v_words,w2v_model,tfidf_feat,tf_
```

```
60000
50
20000
50
20000
50
```

In [52]:
```
tf_idf_opt_brute,tf_idf_tr_f1_brute=fit_and_val(tf_idf_sent_vectors_train,y_tr,tf_idf_
```

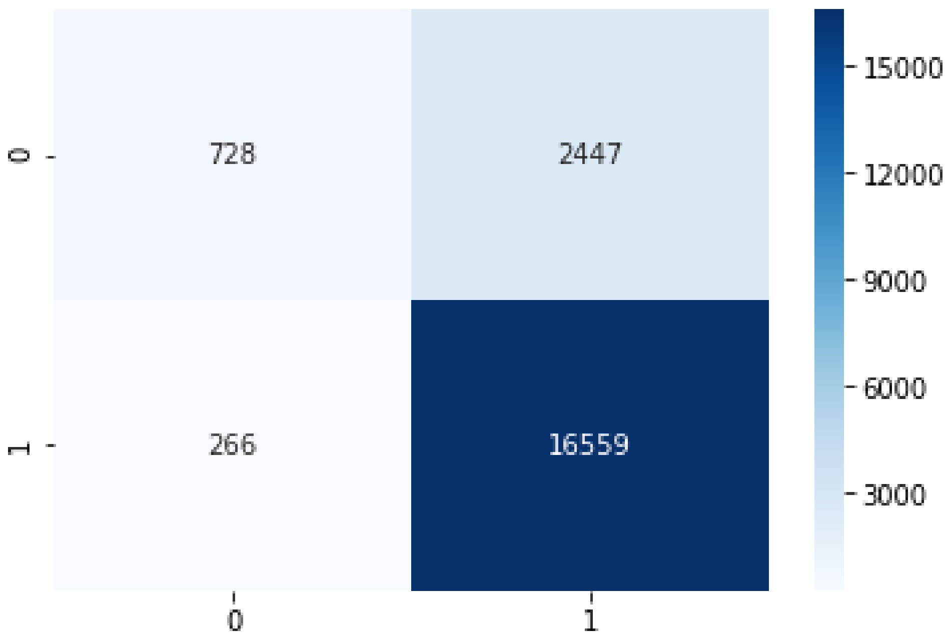
```
For n_neighbors = 1

model accuracy is 88.53501966901433 %

For n_neighbors = 3

model accuracy is 90.93583475610278 %

For n_neighbors = 5

model accuracy is 91.59748815903768 %

For n_neighbors = 7

model accuracy is 91.92138540401513 %

For n_neighbors = 9
```

```
model accuracy is 92.14241692613638 %

For n_neighbors = 11

model accuracy is 92.14588476373054 %

For n_neighbors = 13

model accuracy is 92.20808959827473 %

For n_neighbors = 15

model accuracy is 92.17695588818403 %

For n_neighbors = 17

model accuracy is 92.2355210477359 %

For n_neighbors = 19

model accuracy is 92.23819250804644 %
The optimal number of neighbors is 19.
```




```
the misclassification error for each k value is :  [0.115 0.091 0.084 0.081 0.079 0.079 0.078
[[  728  2447]
```

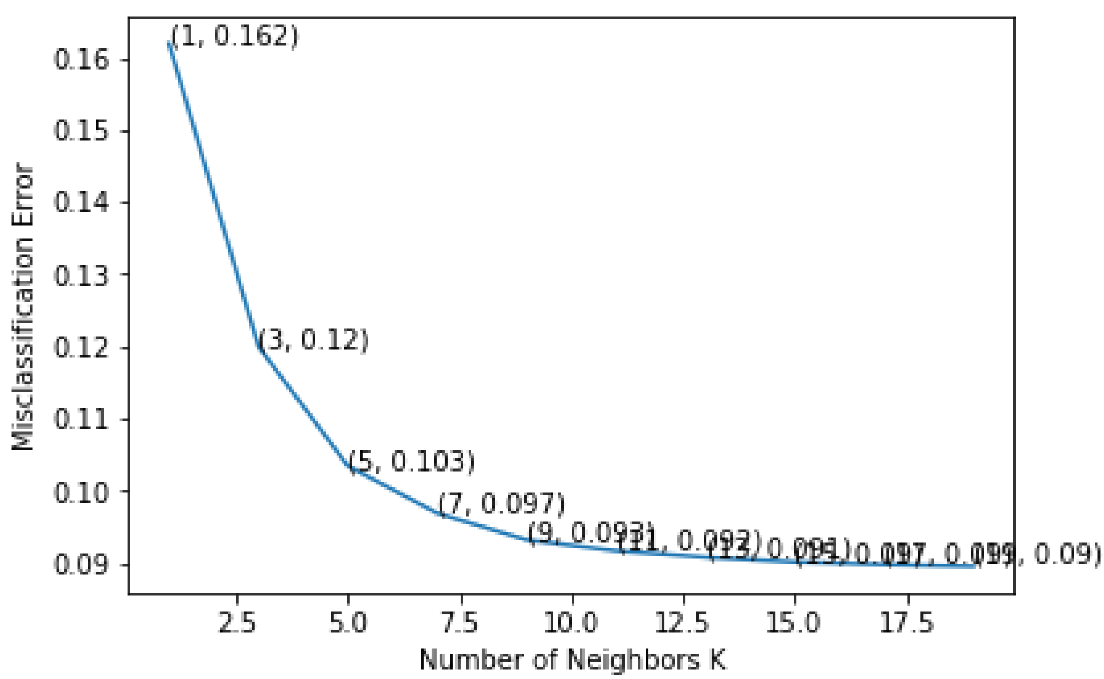
```
 [  266 16559]]
             precision    recall  f1-score   support

   Negative       0.73      0.23      0.35      3175
   Positive       0.87      0.98      0.92     16825

avg / total       0.85      0.86      0.83     20000
```



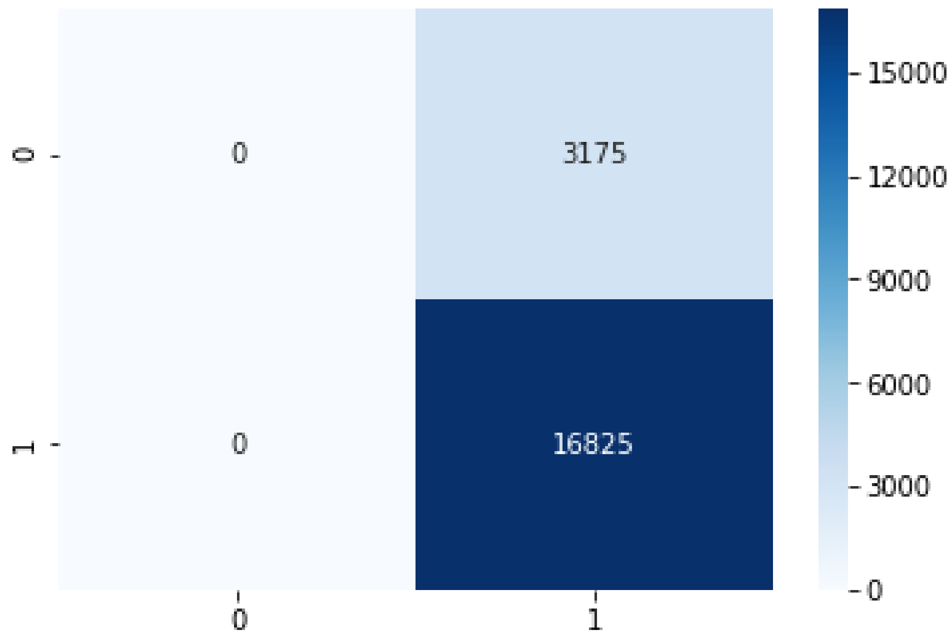
In [53]: `tf_idf_opt_kd,tf_idf_tr_f1_kd=fit_and_val(tf_idf_sent_vectors_train,y_tr,tf_idf_sent_`

```
For n_neighbors = 1

model accuracy is 83.80690191165189 %

For n_neighbors = 3

model accuracy is 88.00287016102259 %

For n_neighbors = 5

model accuracy is 89.65650638595486 %

For n_neighbors = 7
```

```
model accuracy is 90.31169160105674 %

For n_neighbors = 9

model accuracy is 90.68215167980478 %

For n_neighbors = 11

model accuracy is 90.83265196431822 %

For n_neighbors = 13

model accuracy is 90.92032967661534 %

For n_neighbors = 15

model accuracy is 90.98630600244103 %

For n_neighbors = 17

model accuracy is 91.02180736022805 %

For n_neighbors = 19

model accuracy is 91.04752313308599 %
The optimal number of neighbors is 19.
```



```
the misclassification error for each k value is :  [0.162 0.12  0.103 0.097 0.093 0.092 0.091 0
[[    0  3175]
 [    0 16825]]
             precision    recall  f1-score   support

   Negative       0.00      0.00      0.00      3175
   Positive       0.84      1.00      0.91     16825

avg / total       0.71      0.84      0.77     20000
```




```
In [58]: print(summary_table)
```

```
+------------------+-----------+-----------+-------------------+
|      Method      | Algorithm | Optimam K |     F-1 Score     |
+------------------+-----------+-----------+-------------------+
|   Bag Of Word    |   brute   |     11    | 91.46843238235374 |
|   Bag Of Word    |  kd_tree  |     19    | 91.36292031071758 |
|      TF-IDF      |   brute   |     9     | 92.29145840275463 |
|      TF-IDF      |  kd_tree  |     19    | 91.38029393387845 |
|  Wieghted W2Vec  |   brute   |     13    |  92.7362343331925 |
|  Wieghted W2Vec  |  kd_tree  |     19    | 91.34149659863945 |
```

```
| Wieghted TF-IDF |   brute   |     19    | 92.42834417124836 |
| Wieghted TF-IDF |  kd_tree  |     19    | 91.37813985064494 |
+-----------------+-----------+-----------+-------------------+
```

0.1 Conclusion : The best K is 13 ,The best method is wieghted word2vec , although Kd_tree is faster but 'brute_force' method has given highest accuracy (~93%).