Amazon Fine Food Reviews Analysis

October 21, 2018

1 [7] Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

1.1 [7.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
In [1]: %matplotlib inline

        import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import TfidfVectorizer

        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.metrics import confusion_matrix
        from sklearn import metrics
        from sklearn.metrics import roc_curve, auc
        from nltk.stem.porter import PorterStemmer
        import warnings
        warnings.filterwarnings("ignore")



        # using the SQLite Table to read data.
        con = sqlite3.connect('./amazon-fine-food-reviews/database.sqlite')



        #filtering only positive and negative reviews i.e.
        # not taking into consideration those reviews with Score=3
        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 r
        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
```

```
In [2]: filtered_data.shape #looking at the number of attributes and size of the data
        filtered_data.head()
```

```
Out[2]:    Id   ProductId          UserId                      ProfileName  \
        0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
        1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
        2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
        3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl
        4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

           HelpfulnessNumerator  HelpfulnessDenominator     Score        Time  \
        0                     1                       1  positive  1303862400
        1                     0                       0  negative  1346976000
        2                     1                       1  positive  1219017600
        3                     3                       3  negative  1307923200
        4                     0                       0  positive  1350777600

                         Summary                                               Text
        0  Good Quality Dog Food  I have bought several of the Vitality canned d...
        1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
        2  "Delight" says it all  This is a confection that has been around a fe...
        3         Cough Medicine  If you are looking for the secret ingredient i...
        4            Great taffy  Great taffy at a great price.  There was a wid...
```

2 Exploratory Data Analysis

2.1 [7.1.2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

```
In [3]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display
```

```
Out[3]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
        3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
        4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

           HelpfulnessDenominator  Score        Time  \
```

```
0                       2      5   1199577600
1                       2      5   1199577600
2                       2      5   1199577600
3                       2      5   1199577600
4                       2      5   1199577600

                               Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As can be seen above the same user has multiple reviews of the with the same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [4]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

In [5]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

Out[5]:
```
(364173, 10)
```

In [6]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[6]:
```
69.25890143662969
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

```
In [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND Id=44737 OR Id=64422
        ORDER BY ProductID
        """, con)
        display
```

```
Out[7]:       Id   ProductId          UserId              ProfileName  \
        0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
        1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

           HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        0                     3                       1      5  1224892800
        1                     3                       2      4  1212883200

                                              Summary  \
        0            Bought This for My Son at College
        1  Pure cocoa taste with crunchy almonds inside

                                                       Text
        0  My son loves spaghetti so I didn't hesitate or...
        1  It was almost a 'love at first bite' - the per...
```

```
In [8]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [9]: #Before starting the next phase of preprocessing lets see the number of entries left
        print(final.shape)

        #How many positive and negative reviews are present in our dataset?
        final['Score'].value_counts()
```

```
(364171, 10)
```

```
Out[9]: positive    307061
        negative     57110
        Name: Score, dtype: int64
```

2.2 7.2.3 Text Preprocessing: Stemming, stop-word removal and Lemmatization.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric

4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [10]:
```
# find sentences containing HTML tags
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
6
I set aside at least an hour each day to read to my son (3 y/o). At this point, I consider myse
```

In [11]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\risha\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[11]: True

In [12]:
```
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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 ch
```

```
        cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
        cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
        return  cleaned
    print(stop)
    print('************************************')
    print(sno.stem('tasty'))
```

```
{'and', 'theirs', "you're", 'hers', 'itself', "hadn't", 'those', 'our', "you'd", 'couldn', 'do
************************************
tasti
```

In [13]:
```
#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 final['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 (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to descr
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to descr
                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 [14]:
```
final['CleanedText']=final_string #adding a column of CleanedText which displays the 
final.head(3)
```

Out[14]:
```
             Id   ProductId          UserId          ProfileName  \
138706   150524  0006641040   ACITT7DI6IDDL       shari zychinski
```

```
138688  150506  0006641040   A2IW4PEEKO2R0U                  Tracy
138689  150507  0006641040  A1S4A3IQ2MU7V4  sally sue "sally sue"

        HelpfulnessNumerator  HelpfulnessDenominator     Score        Time  \
138706                     0                       0  positive   939340800
138688                     1                       1  positive  1194739200
138689                     1                       1  positive  1191456000

                                           Summary  \
138706                  EVERY book is educational
138688  Love the book, miss the hard cover version
138689                chicken soup with rice months

                                                      Text  \
138706  this witty little book makes my son laugh at l...
138688  I grew up reading these Sendak books, and watc...
138689  This is a fun way for children to learn their ...

                                               CleanedText
138706  b'witti littl book make son laugh loud recit c...
138688  b'grew read sendak book watch realli rosi movi...
138689  b'fun way children learn month year learn poem...
```

In [15]:
```python
# storing final table into an SQlLite table for future.
conn = sqlite3.connect('final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn, schema=None, if_exists='replace', index=True, index_labe
```

3 [7.2.2] Bag of Words (BoW)

In [16]:
```python
#BoW
#time based sorting
data_sorted=final[0:1000].sort_values('Time',kind='quicksort')
#data_sorted has datapoints sorted on basis of time
count_vect = CountVectorizer() #in scikit-learn

final_counts = count_vect.fit_transform(data_sorted['Text'].values) #Taking the top 1
```

In [17]:
```python
type(final_counts)
```

Out[17]:
```
scipy.sparse.csr.csr_matrix
```

In [18]:
```python
final_counts.get_shape()
```

Out[18]:
```
(1000, 7109)
```

3.1 [7.2.4] Bi-Grams and n-Grams.

Motivation

Now that we have our list of words describing positive and negative reviews lets analyse them. We begin analysis by getting the frequency distribution of the words as shown below

```
In [ ]: freq_dist_positive=nltk.FreqDist(all_positive_words)
        freq_dist_negative=nltk.FreqDist(all_negative_words)
        print("Most Common Positive Words : ",freq_dist_positive.most_common(20))
        print("Most Common Negative Words : ",freq_dist_negative.most_common(20))
```

Observation:- From the above it can be seen that the most common positive and the negative words overlap for eg. 'like' could be used as 'not like' etc. So, it is a good idea to consider pairs of consequent words (bi-grams) or q sequnce of n consecutive words (n-grams)

```
In [ ]: #bi-gram, tri-gram and n-gram

        #removing stop words like "not" should be avoided before building n-grams
        count_vect = CountVectorizer(ngram_range=(1,2) ) #in scikit-learn
        final_bigram_counts = count_vect.fit_transform(final['Text'].values)
```

```
In [ ]: final_bigram_counts.get_shape()
```

4 [7.2.5] TF-IDF

```
In [19]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))


         #Sampling 1000 datapoints randomly
         final_tf_idf = tf_idf_vect.fit_transform(data_sorted['Text'].values)#Taking the top 1
```

```
In [20]: print(final_tf_idf.get_shape())
         print(type(final_tf_idf))
```

```
(1000, 52885)
<class 'scipy.sparse.csr.csr_matrix'>
```

```
In [21]: features = tf_idf_vect.get_feature_names()
         len(features)
```

```
Out[21]: 52885
```

```
In [ ]: features[100000:100010]
```

```
In [ ]: # covnert a row in saprsematrix to a numpy array
        print(final_tf_idf[3,:].toarray()[0])
```

In [ ]:
```
# source: https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

top_tfidf = top_tfidf_feats(final_tf_idf[1,:].toarray()[0],features,25)
```

In [ ]:
```
top_tfidf
```

5 [7.2.6] Word2Vec

In [ ]:
```
# Using Google News Word2Vectors
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary
```

In [ ]:
```
model.wv['computer']
```

In [ ]:
```
model.wv.similarity('woman', 'man')
```

In [ ]:
```
model.wv.most_similar('woman')
```

In [ ]:
```
model.wv.most_similar('tasti')  # "tasti" is the stemmed word for tasty, tastful
```

In [ ]:
```
model.wv.most_similar('tasty')
```

In [ ]:
```
model.wv.similarity('tasty', 'tast')
```

In [22]:
```
# Train your own Word2Vec model using your own text corpus
import gensim
i=0
list_of_sent=[]
for sent in data_sorted['Text'].values:
```

```
            filtered_sentence=[]
            sent=cleanhtml(sent)
            for w in sent.split():
                for cleaned_words in cleanpunc(w).split():
                    if(cleaned_words.isalpha()):
                        filtered_sentence.append(cleaned_words.lower())
                    else:
                        continue
            list_of_sent.append(filtered_sentence)
```

```
In [23]: print(final['Text'].values[0])
         print("*****************************************************************")
         print(list_of_sent[0])
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
*****************************************************************
['this', 'witty', 'little', 'book', 'makes', 'my', 'son', 'laugh', 'at', 'loud', 'i', 'recite'
```

```
In [24]: w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

```
In [25]: #Fetching the trained word vectors from the model
         w2v_matrix=w2v_model.wv
         print(w2v_matrix)
```

```
<gensim.models.keyedvectors.Word2VecKeyedVectors object at 0x0000021FE6D74828>
```

```
In [ ]: #w2v_model.wv.most_similar('like')
```

```
In [ ]: count_vect_feat = count_vect.get_feature_names() # list of words in the BoW
        count_vect_feat.index('like')
        print(count_vect_feat[64055])
```

6 [7.2.7] Avg W2V, TFIDF-W2V

```
In [26]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in 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/review
             for word in sent: # for each word in a review/sentence
                 try:
                     vec = w2v_model.wv[word]
                     sent_vec += vec
                     cnt_words += 1
```

```
            except:
                pass
        sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    print(len(sent_vectors))
    print(len(sent_vectors[0]))
```

```
1000
50
```

```
In [27]: # 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 cell_val = tfidf

         tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
         row=0;
         for sent in 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/review
             for word in sent: # for each word in a review/sentence
                 try:
                     vec = w2v_model.wv[word]
                     # obtain the tf_idfidf of a word in a sentence/review
                     tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
                     sent_vec += (vec * tf_idf)
                     weight_sum += tf_idf
                 except:
                     pass
             sent_vec /= weight_sum
             tfidf_sent_vectors.append(sent_vec)
             row += 1
```

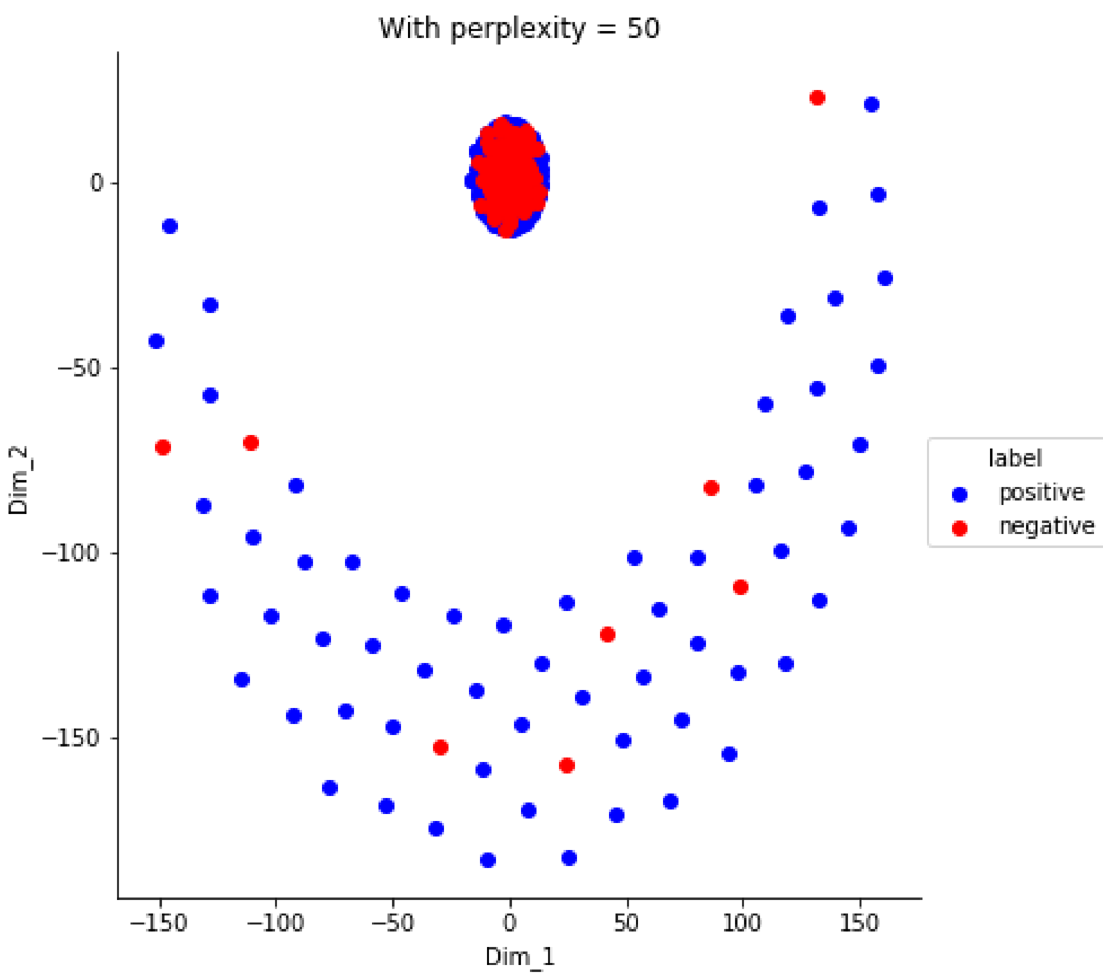
7 t-SNE Visualization of BoW

```
In [28]: #Standardizing the data
         from sklearn.preprocessing import StandardScaler
         standardized_data=StandardScaler(with_mean=False).fit_transform(final_counts)

         #Dimesion of the standardized_data
         print(standardized_data.shape)
```

```
(1000, 7109)
```

In [29]:
```
from sklearn.manifold import TSNE

#Using the first 1k datapoints


#Converting the sparse matrix to numpy array
data_array=standardized_data.toarray()
labels_1k=data_sorted['Score'].values
model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000)
tsne_data1=model.fit_transform(data_array)
tsne_data1 = np.vstack((tsne_data1.T, labels_1k)).T
```

In [30]:
```
# creating a new data fram which help us in ploting the result data
import seaborn as sns
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", palette=['blue','red'],size=6).map(plt.scatter, 'I
plt.title('With perplexity = 50')
plt.show()
```



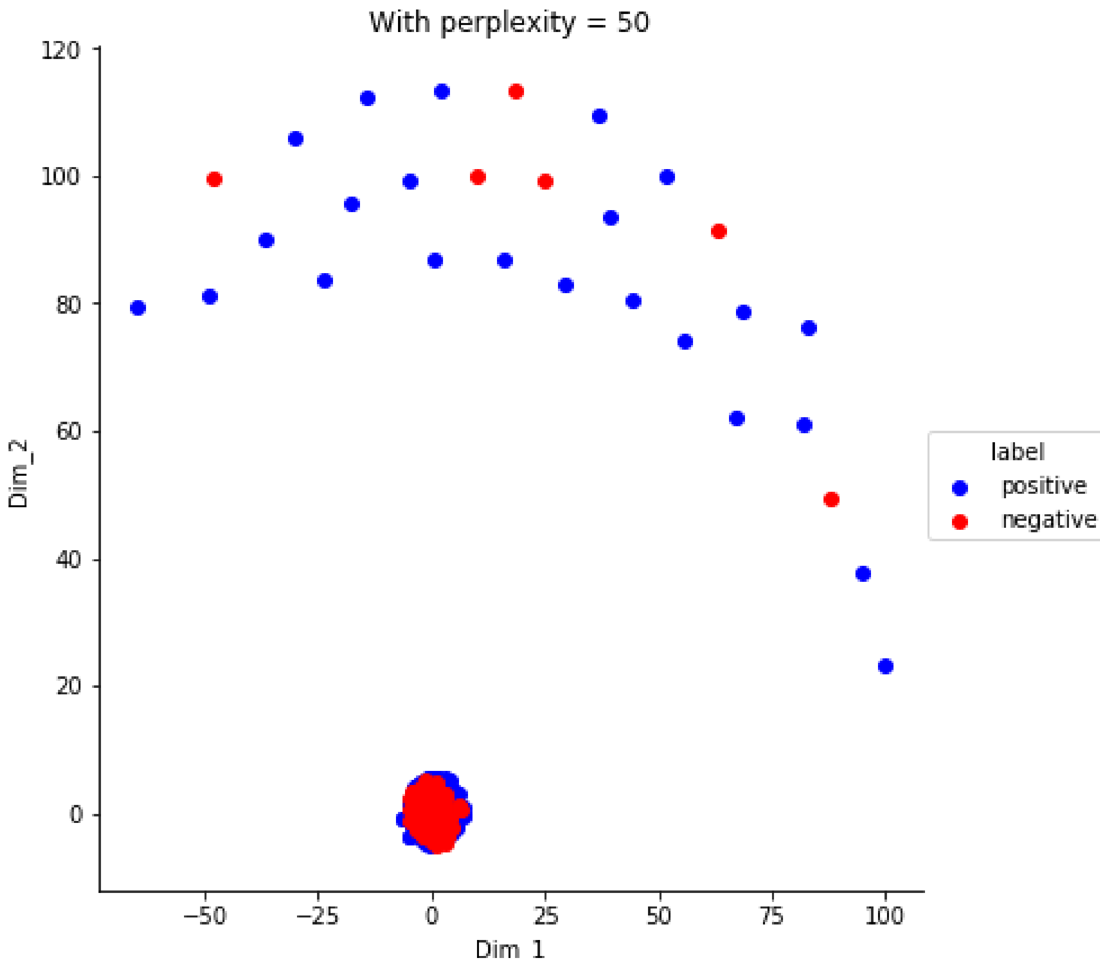
8 t-SNE visualization of tf-idf

In [31]:
```
#Standardizing the data
from sklearn.preprocessing import StandardScaler
standardized_data_tf_idf=StandardScaler(with_mean=False).fit_transform(final_tf_idf)

#Printing the shape
print(standardized_data_tf_idf.shape)
```

```
(1000, 52885)
```

In [32]:
```
from sklearn.manifold import TSNE


#Converting the sparse matrix to numpy array
tfidf_data_array=standardized_data_tf_idf.toarray()
labels_1k=data_sorted['Score'].values
model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000)
tsne_data2=model.fit_transform(tfidf_data_array)
tsne_data2 = np.vstack((tsne_data2.T, labels_1k)).T
```

In [33]:
```
# creating a new data fram which help us in ploting the result data
import seaborn as sns
tsne_df2 = pd.DataFrame(data=tsne_data2, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sns.FacetGrid(tsne_df2, hue="label", palette=['blue','red'],size=6).map(plt.scatter,
plt.title('With perplexity = 50')
plt.show()
```



9 tSNE visualization of Avg-W2V

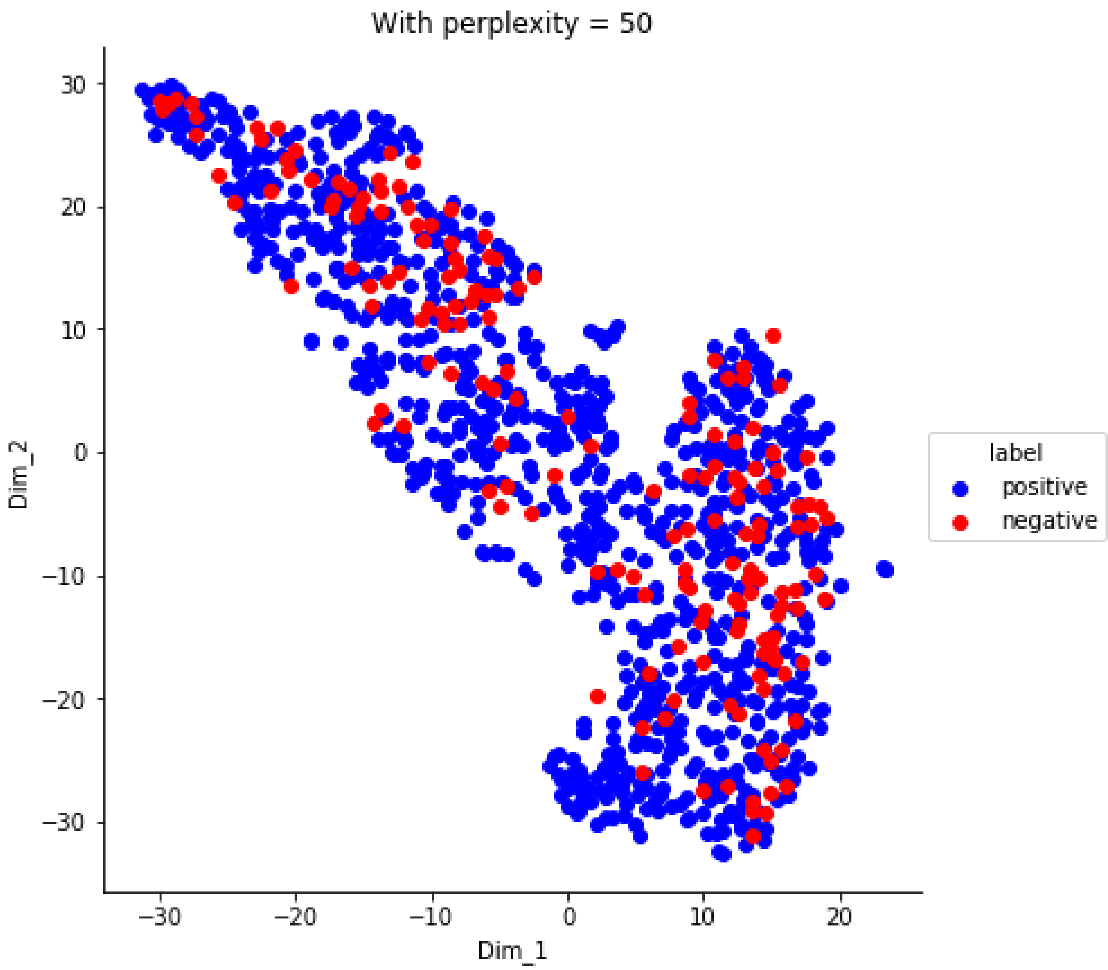
```
In [34]: #Standardizing the data
         from sklearn.preprocessing import StandardScaler
         standardized_data_w2v=StandardScaler(with_mean=False).fit_transform(sent_vectors)

         #Printing the shape
         print(standardized_data_w2v.shape)
         print(type(standardized_data_w2v))
```

```
(1000, 50)
<class 'numpy.ndarray'>
```

```
In [35]: from sklearn.manifold import TSNE
```

```
labels_1k=data_sorted['Score'].values
model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000)
tsne_data_w2v=model.fit_transform(standardized_data_w2v)
tsne_data_w2v = np.vstack((tsne_data_w2v.T, labels_1k)).T
```

In [36]:
```
# creating a new data fram which help us in ploting the result data
import seaborn as sns
tsne_df_w2v = pd.DataFrame(data=tsne_data_w2v, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sns.FacetGrid(tsne_df_w2v, hue="label", palette=['blue','red'],size=6).map(plt.scatte
plt.title('With perplexity = 50')
plt.show()
```



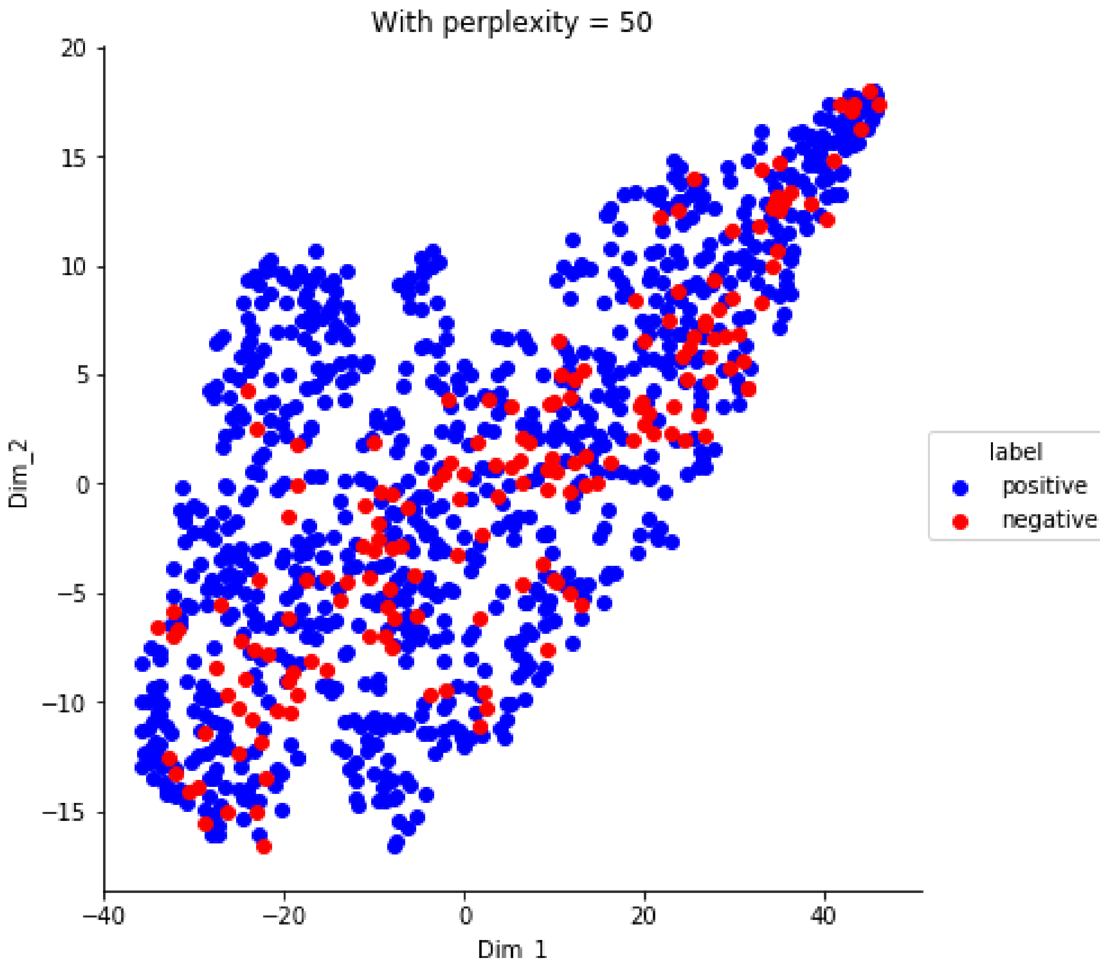
10 tSNE visualization of tfidf-w2v

In [37]:
```
#Standardizing the data
from sklearn.preprocessing import StandardScaler
standardized_data_tfidfw2v=StandardScaler().fit_transform(tfidf_sent_vectors)

#Printing the shape
print(standardized_data_tfidfw2v.shape)
print(type(standardized_data_tfidfw2v))
```

```
(1000, 50)
<class 'numpy.ndarray'>
```

In [38]:
```
from sklearn.manifold import TSNE

labels_1k=data_sorted['Score'].values
model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000)
tsne_data_tfidfw2v=model.fit_transform(standardized_data_tfidfw2v)
tsne_data_tfidfw2v = np.vstack((tsne_data_tfidfw2v.T, labels_1k)).T
```

In [39]:
```
# creating a new data fram which help us in ploting the result data
import seaborn as sns
tsne_df_tfidfw2v = pd.DataFrame(data=tsne_data_tfidfw2v, columns=("Dim_1", "Dim_2", "l

# Ploting the result of tsne
sns.FacetGrid(tsne_df_tfidfw2v, hue="label", palette=['blue','red'],size=6).map(plt.sc
plt.title('With perplexity = 50')
plt.show()
```



11 KNN on BoW using kfold Cross Validation

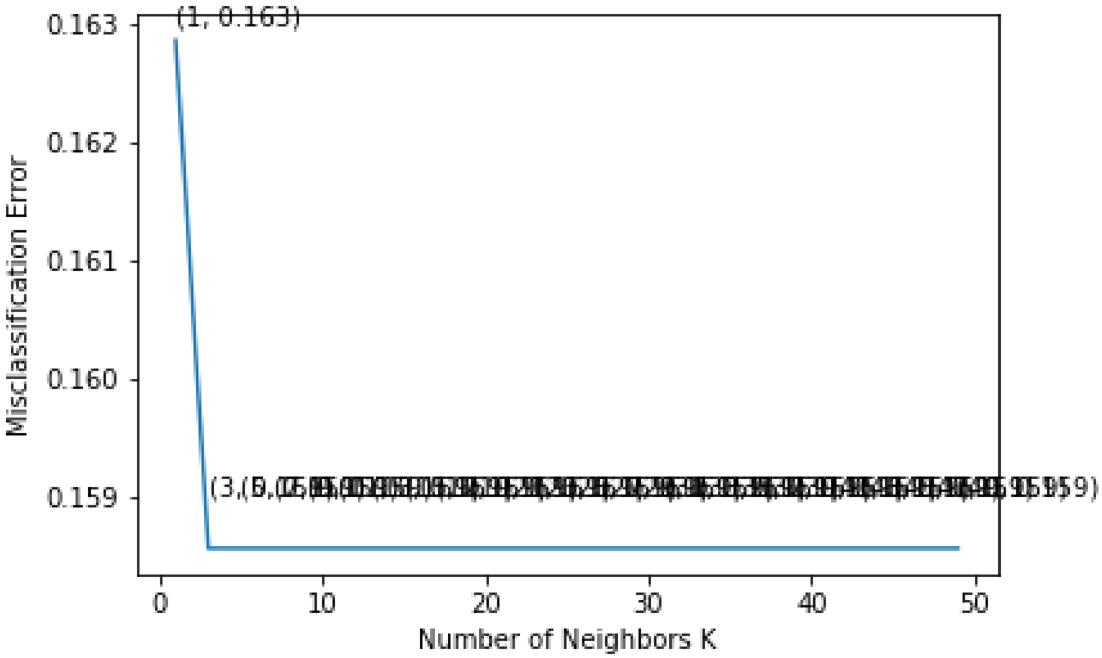
In [41]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
```

In [60]:
```
#Splitting in data in tarin and test
#The test data has the latest reviews as data is sorted on basis of time_stamp
#data_sorted is a pandas series sorted with respect to time_stamp
```

```
x_train=data_array[0:700] #taking the top 70% data ast the traing set
x_test=data_array[700:1000]#taking the latest 30% data as the test set
y_train=data_sorted['Score'].iloc[0:700].values
y_test=data_sorted['Score'].iloc[700:1000].values
```

In [61]:
```
k=list(range(0,50))
k_neighbours=list(filter(lambda x: x%2 !=0,k))
cv_scores=[]

# perform 10-fold cross validation
for i in k_neighbours:
    knn = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn, x_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = k_neighbours[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

# plot misclassification error vs k
plt.plot(k_neighbours, MSE)

for xy in zip(k_neighbours, 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))
```

```
The optimal number of neighbors is 3.
```




```
the misclassification error for each k value is :  [0.163 0.159 0.159 0.159 0.159 0.159 0.159 0
 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159
 0.159]
```

In [62]:
```
# ============================== KNN with k = optimal_k ===============================
# instantiate learning model k = optimal_k
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_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
```

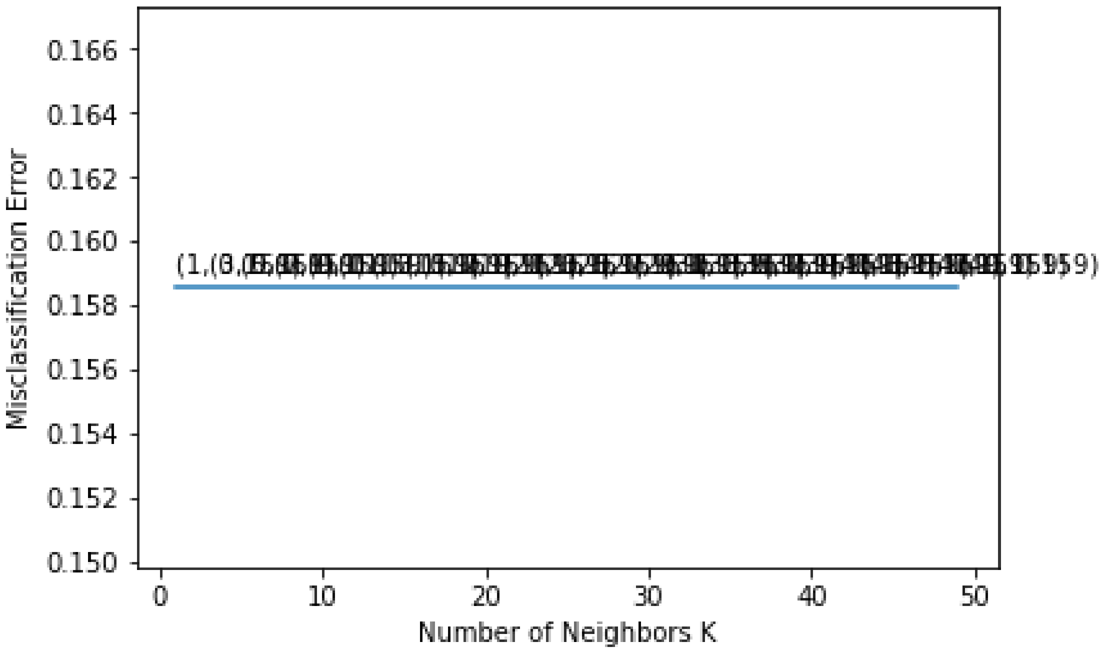
```
The accuracy of the knn classifier for k = 3 is 80.666667%
```

12 KNN on tfidf using kfold Cross Validation

In [65]:
```
#Splitting the tfidf vectors into train and test
#Splitting in data in tarin and test
#The test data has the latest reviews as data is sorted on basis of time_stamp
x_train_tfidf=tfidf_data_array[0:700]
x_test_tfidf=tfidf_data_array[700:1000]
y_train_tfidf=data_sorted['Score'].iloc[0:700].values
y_test_tfidf=data_sorted['Score'].iloc[700:1000].values
```

In [66]:
```
k=list(range(0,50))
k_neighbours=list(filter(lambda x: x%2 !=0,k))
cv_scores=[]
# perform 10-fold cross validation
for i in k_neighbours:
    knn1 = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn1, x_train_tfidf, y_train_tfidf, cv=10, scoring='accu
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE1 = [1 - x for x in cv_scores]

# determining best k
optimal_k = k_neighbours[MSE1.index(min(MSE1))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

# plot misclassification error vs k
plt.plot(k_neighbours, MSE1)

for xy in zip(k_neighbours, np.round(MSE1,3)):
    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(MSE1,3))
```

```
The optimal number of neighbors is 1.
```




```
the misclassification error for each k value is :  [0.159 0.159 0.159 0.159 0.159 0.159 0.159 0
 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159
 0.159]
```

```
In [67]: # ============================== KNN with k = optimal_k ==============================
         # instantiate learning model k = optimal_k
         knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

         # fitting the model
         knn_optimal.fit(x_train_tfidf, y_train_tfidf)

         # predict the response
         pred = knn_optimal.predict(x_test_tfidf)

         # evaluate accuracy
         acc = accuracy_score(y_test_tfidf, pred) * 100
         print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
```

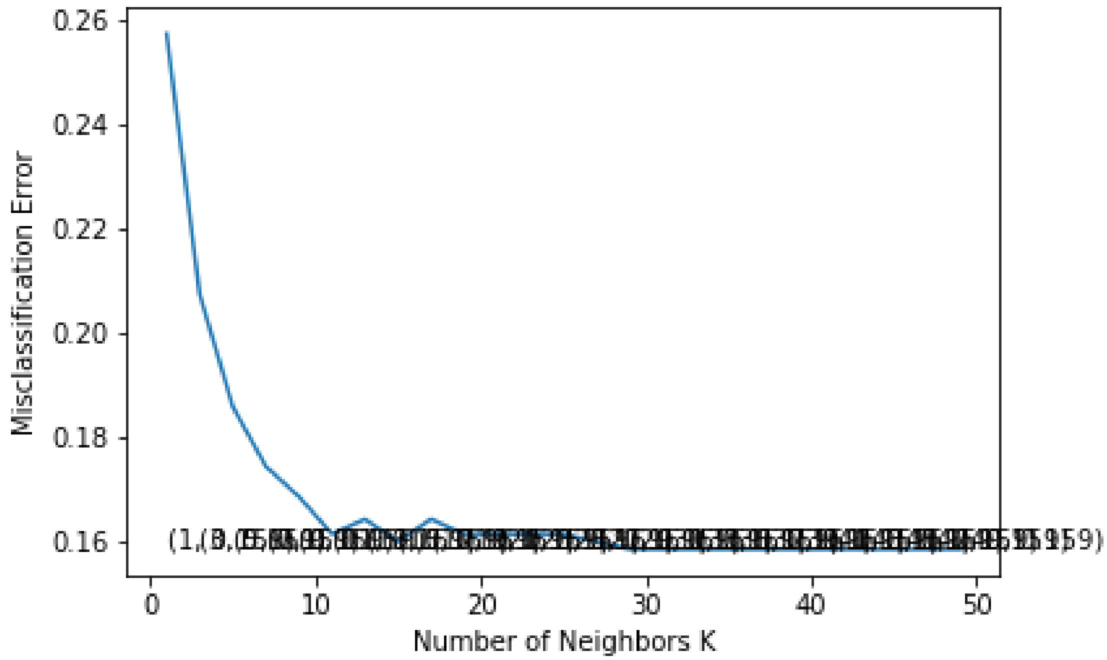
```
The accuracy of the knn classifier for k = 1 is 80.666667%
```

13 KNN on Avg-W2V using kfold Cross Validation

In [69]:
```
#Splitting the tfidf vectors into train and test
#Splitting in data in tarin and test
#The test data has the latest reviews as data is sorted on basis of time_stamp
x_train_avg=standardized_data_w2v[0:700]
x_test_avg=standardized_data_w2v[700:1000]
y_train_avg=data_sorted['Score'].iloc[0:700].values
y_test_avg=data_sorted['Score'].iloc[700:1000].values
```

In [70]:
```
k=list(range(0,50))
k_neighbours=list(filter(lambda x: x%2 !=0,k))
cv_scores=[]
# perform 10-fold cross validation
for i in k_neighbours:
    knn1 = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn1, x_train_avg, y_train_avg, cv=10, scoring='accuracy
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE2 = [1 - x for x in cv_scores]

# determining best k
optimal_k = k_neighbours[MSE2.index(min(MSE2))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

# plot misclassification error vs k
plt.plot(k_neighbours, MSE2)

for xy in zip(k_neighbours, np.round(MSE1,3)):
    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(MSE2,3))
```

```
The optimal number of neighbors is 29.
```




```
the misclassification error for each k value is :  [0.257 0.207 0.186 0.174 0.169 0.161 0.164 0
 0.161 0.16  0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159
 0.159]
```

In [73]:
```python
# ============================== KNN with k = optimal_k ===============================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(x_train_avg, y_train_avg)

# predict the response
pred = knn_optimal.predict(x_test_avg)

# evaluate accuracy
ac = accuracy_score(y_test_avg, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, ac))
```

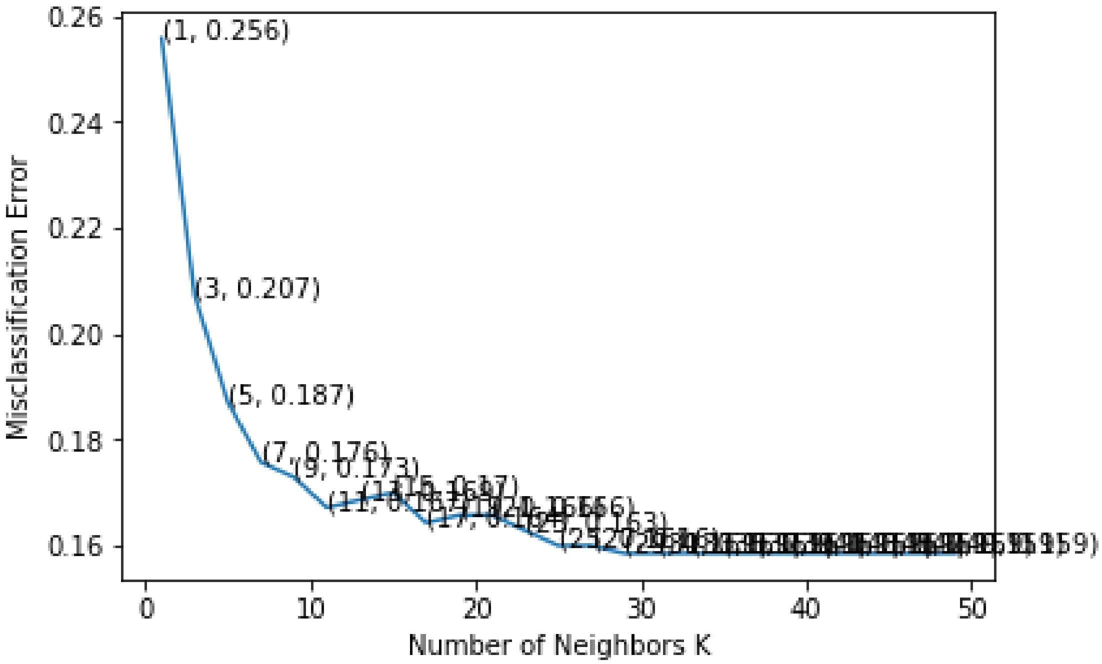
```
The accuracy of the knn classifier for k = 29 is 80.666667%
```

14 KNN on tfidf-w2v using kfold Cross Validation

```
In [76]: #Splitting the tfidf vectors into train and test
         #Splitting in data in tarin and test
         #The test data has the latest reviews as data is sorted on basis of time_stamp
         x_train_tfw2v=standardized_data_tfidfw2v[0:700]
         x_test_tfw2v=standardized_data_tfidfw2v[700:1000]
         y_train_tfw2v=data_sorted['Score'].iloc[0:700].values
         y_test_tfw2v=data_sorted['Score'].iloc[700:1000].values
```

```
In [77]: k=list(range(0,50))
         k_neighbours=list(filter(lambda x: x%2 !=0,k))
         cv_scores=[]
         # perform 10-fold cross validation
         for i in k_neighbours:
             knn1 = KNeighborsClassifier(n_neighbors=i)
             scores = cross_val_score(knn1, x_train_tfw2v, y_train_tfw2v, cv=10, scoring='accu
             cv_scores.append(scores.mean())

         # changing to misclassification error
         MSE4 = [1 - x for x in cv_scores]

         # determining best k
         optimal_k = k_neighbours[MSE4.index(min(MSE4))]
         print('\nThe optimal number of neighbors is %d.' % optimal_k)

         # plot misclassification error vs k
         plt.plot(k_neighbours, MSE4)

         for xy in zip(k_neighbours, np.round(MSE4,3)):
             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(MSE4,3))
```

```
The optimal number of neighbors is 29.
```




```
the misclassification error for each k value is :  [0.256 0.207 0.187 0.176 0.173 0.167 0.169 0
 0.16  0.16  0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159 0.159
 0.159]
```

```
In [79]: # ============================== KNN with k = optimal_k ==============================
         # instantiate learning model k = optimal_k
         knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

         # fitting the model
         knn_optimal.fit(x_train_tfw2v, y_train_tfw2v)

         # predict the response
         pred = knn_optimal.predict(x_test_tfw2v)

         # evaluate accuracy
         ac = accuracy_score(y_test_tfw2v, pred) * 100
         print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, ac))
```

```
The accuracy of the knn classifier for k = 29 is 80.666667%
```

15 Conclusion for KNN on the dataset

After using KNN with kfold cross validation on diferent NLP Algorthims the result are as follows:-

1) KNN on Bag of Words Optimal k=3 Accuracy on test set=80.67%
2) KNN on tfidf Optimal k=1 Accuracy on test set=80.67%
3) KNN on Average Word2Vec Optimal k=29 Accuracy on test set=80.67%
4) KNN on tfidf-Word2Vec Optimal k=29 Accuracy on test set=80.67%