SVM_AMAZON

July 21, 2018

1 SVM ON AMAZON REVIEWS DATASET

1.1 about dataset:

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

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
```

```
        con = sqlite3.connect('./amazon-fine-food-reviews/database.sqlite')
        filtered_data = pd.read_sql_query("""
        SELECT *
```

```
        FROM Reviews
        WHERE Score != 3
        """, con)
```

In [2]:
```
s1= filtered_data.loc[filtered_data["Score"]>=4].sample(n=12000,random_state=24)
print(s1.shape)

s2= filtered_data.loc[filtered_data["Score"]<=2].sample(n=9500,random_state=127)
print(s2.shape)
data=s1
data=data.append(s2)
print(data.shape)
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
data['Score'] = positiveNegative
sorted_data=data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind=
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

```
(12000, 10)
(9500, 10)
(21500, 10)
```

Out[2]:
```
(20282, 10)
```

In [3]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
final=final.drop_duplicates(subset={"UserId","ProfileName","Time"},keep='first',inplac
final.shape
final['Score'].value_counts()
```

Out[3]:
```
positive    11579
negative     8522
Name: Score, dtype: int64
```

1.2 Text preprocessing

In [4]:
```
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
5
okay, it's freeze-dried liver cube-lets.  my common sense tells me that for a lot less than th
```

```
In [5]: import string
        import nltk
        nltk.download('stopwords')
        from nltk.corpus import stopwords
        from nltk.stem import PorterStemmer
        from nltk.stem.wordnet import WordNetLemmatizer

        stop = set(stopwords.words('english'))
        sno = nltk.stem.SnowballStemmer('english')

        def cleanhtml(sentence):
            cleanr = re.compile('<.*?>')
            cleantext = re.sub(cleanr, ' ', sentence)
            return cleantext
        def cleanpunc(sentence):
            cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
            cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
            return  cleaned
```

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

```
In [6]: i=0
        str1=' '
        final_string=[]
        all_positive_words=[]
        all_negative_words=[]
        s=''
        for sent in final['Text'].values:
            filtered_sentence=[]

            sent=cleanhtml(sent)
            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)
                            if(final['Score'].values)[i] == 'negative':
                                all_negative_words.append(s)
```

```
                    else:
                        continue
                else:
                    continue

            str1 = b" ".join(filtered_sentence)

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

```
In [7]: final['CleanedText']=final_string
        final.head(3)


        conn = sqlite3.connect('final.sqlite')
        c=conn.cursor()
        conn.text_factory = str
        final.to_sql('Reviews', conn, flavor=None, schema=None, if_exists='replace', index=Tru
                     dtype=None)
        final.shape
```

```
Out[7]: (20101, 11)
```

```
In [8]: final=final.sort_values('Time')
```

```
In [9]: x= np.array(final.iloc[:, 0:10])
        y= np.array(final['Score'])
```

1.3 BoW SVM

```
In [10]: count_vect = CountVectorizer()
         final_bow = count_vect.fit_transform(x[:,9])
         final_bow.get_shape()
```

```
Out[10]: (20101, 29330)
```

```
In [11]: from sklearn.cross_validation import train_test_split
         from sklearn.cross_validation import cross_val_score
         from sklearn import cross_validation
```

```
C:\Users\santosh\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [12]: X_1, X_test, y_1, y_test = cross_validation.train_test_split(final_bow, y, test_size=0

         X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)
```

```
In [13]: from sklearn.model_selection import GridSearchCV
         from sklearn import svm
```

```
In [14]: from sklearn.metrics import accuracy_score
         svc=svm.SVC()
         parameters = {'kernel':('linear','rbf'), 'C':[10] ,'gamma':[1.0]}
         clf=GridSearchCV(svc,parameters)
         clf.fit(X_tr,y_tr)
         pred = clf.predict(X_cv)
         acc1 = accuracy_score(y_cv, pred, normalize=True) * float(100)
         print('\nCV accuracy for gamma=1.0 and C=10 is %d%%' %acc1)
```

```
CV accuracy for gamma=1.0 and C=10 is 84%
```

```
In [15]: pred = clf.predict(X_test)
         acc1 = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\ntest accuracy is %d%%' %acc1)
```

```
test accuracy is 82%
```

```
In [16]: from sklearn.model_selection import RandomizedSearchCV
         clf=RandomizedSearchCV(svc,parameters,n_iter=2)
         clf.fit(X_tr,y_tr)
         pred = clf.predict(X_cv)
         acc1 = accuracy_score(y_cv, pred, normalize=True) * float(100)
         print('\nCV accuracy for gamma=1.0 and C=10 is %d%%' %acc1)
```

```
CV accuracy for gamma=1.0 and C=10 is 84%
```

```
In [20]: pred = clf.predict(X_test)
         acc1 = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\ntest accuracy is %d%%' %acc1)
```

```
test accuracy is 83%
```

1.4 Tfidf SVM

```
In [31]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
         final_tf_idf = tf_idf_vect.fit_transform(final['Text'].values)
         final_tf_idf.get_shape()
```

```
Out[31]: (20101, 446865)
```

```
In [22]: X_1, X_test, y_1, y_test = cross_validation.train_test_split(final_tf_idf, y, test_si

         X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

In [23]: clf=GridSearchCV(svc,parameters)
         clf.fit(X_tr,y_tr)
         pred = clf.predict(X_cv)
         acc1 = accuracy_score(y_cv, pred, normalize=True) * float(100)
         print('\nCV accuracy for gamma=1.0 and C=10 is %d%%' %acc1)


CV accuracy for gamma=1.0 and C=10 is 89%


In [24]: pred = clf.predict(X_test)
         acc1 = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\ntest accuracy is %d%%' %acc1)


test accuracy is 89%


In [25]: clf=RandomizedSearchCV(svc,parameters,n_iter=2)
         clf.fit(X_tr,y_tr)
         pred = clf.predict(X_cv)
         acc1 = accuracy_score(y_cv, pred, normalize=True) * float(100)
         print('\nCV accuracy is %d%%' %acc1)


CV accuracy is 89%


In [26]: pred = clf.predict(X_test)
         acc1 = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\ntest accuracy is %d%%' %acc1)


test accuracy is 89%
```

1.5 Word2Vec SVM

```
In [17]: X_1, X_test, y_1, y_test = cross_validation.train_test_split(final['Text'], y, test_s

In [29]: from gensim.models import Word2Vec
         from gensim.models import KeyedVectors
```

```
import pickle
import gensim
i=0
list_of_sent=[]
for sent in X_1.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)
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
words = list(w2v_model.wv.vocab)
```

In [19]:
```
avg_w2v = [];
for sent in list_of_sent:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    avg_w2v.append(sent_vec)
print(len(avg_w2v))
print(len(avg_w2v[0]))
```

```
14070
50
```

In [22]:
```
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(avg_w2v, y_1, test_size=0.3
```

In [23]:
```
clf=GridSearchCV(svc,parameters)
clf.fit(X_tr,y_tr)
pred = clf.predict(X_cv)
acc1 = accuracy_score(y_cv, pred, normalize=True) * float(100)
print('\nCV accuracy for gamma=1.0 and C=10 is %d%%' %acc1)
```

```
CV accuracy for gamma=1.0 and C=10 is 79%
```

In [24]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
import gensim
i=0
list_of_sent=[]
for sent in X_test.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)
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
words = list(w2v_model.wv.vocab)
avg_w2v = [];
for sent in list_of_sent:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    avg_w2v.append(sent_vec)
print(len(avg_w2v))
print(len(avg_w2v[0]))
```

```
6031
50
```

In [25]:
```
pred = clf.predict(avg_w2v)
acc1 = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\ntest accuracy is %d%%' %acc1)
```

```
test accuracy is 57%
```

In [26]:
```
clf=RandomizedSearchCV(svc,parameters,n_iter=2)
clf.fit(X_tr,y_tr)
```

```
        pred = clf.predict(X_cv)
        acc1 = accuracy_score(y_cv, pred, normalize=True) * float(100)
        print('\nCV accuracy is %d%%' %acc1)
```

```
CV accuracy is 79%
```

```
In [28]: pred = clf.predict(avg_w2v)
         acc1 = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\ntest accuracy is %d%%' %acc1)
```

```
test accuracy is 57%
```

2 CONCLUSIONS:

- using the Support vector machine rbf algorithm the results are :
- from GridsearchCV :
- for BoW from CV data=84% and test=82% it is from gridsearchCV however the accuracy for randomizedsearchcv the test accuracy is 83%.
- from Tfidf Cv data=89% and test=89% it is the same for both GridSearch and randomized-Search.
- for Word2vec the accuracy results are from CV=79% and for test=57%