AmazonFineFoodReviews_SVC_Assignment

July 22, 2018

1 SVC Amazon Fine Food Reviews

In [1]:
```
%matplotlib inline

import sqlite3
import pandas as pd #for data frames
import numpy as np #numpy array operations
import nltk #natural lang processing, for processing text
import string
import matplotlib.pyplot as plt
import seaborn as sns #for plotting
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

from nltk.stem.porter import PorterStemmer
import pickle
import seaborn as sn

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
from sklearn.model_selection import train_test_split
from sklearn.metrics import average_precision_score,f1_score,precision_score,recall_sc
from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.svm import SVC
from sklearn.manifold import TSNE

from scipy.stats import expon
import random
```

```
C:\Users\Dell\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: T
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\Dell\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: DeprecationWarning: This m
  DeprecationWarning)
```

```
In [19]: pickle_in=open("cleanedData.pickle","rb")
         final = pickle.load(pickle_in)

         '''pickle_in = open("BOW_tfidf_avgW2V_TfidfW2V.pickle","rb")
         count_vect = pickle.load(pickle_in) #BOW
         final_counts = pickle.load(pickle_in) #BOW

         tf_idf_vect = pickle.load(pickle_in) #TFIDF
         final_tf_idf = pickle.load(pickle_in) #TFIDF
         features = pickle.load(pickle_in) #TFIDF

         w2v_model = pickle.load(pickle_in) #w2v
         words = pickle.load(pickle_in) #w2v

         sent_vectors = pickle.load(pickle_in) #avg W2V'''

         import pickle
         pickle_in = open("BOW_tfidf_avgW2V_Train_test_data.pickle","rb")

         count_vect = pickle.load(pickle_in) #BOW
         final_counts_train = pickle.load(pickle_in) #BOW
         final_counts_test = pickle.load(pickle_in) #BOW
         tf_idf_vect = pickle.load(pickle_in) #tfidf
         final_tf_idf_train = pickle.load(pickle_in) #tfidf
         final_tf_idf_test = pickle.load(pickle_in) #tfidf
         features = pickle.load(pickle_in)
         sent_vectors_train = pickle.load(pickle_in) #avgW2v Vectors
         sent_vectors_test = pickle.load(pickle_in) #avgW2v Vectors
```

```
In [7]: train_data = final.head(int(0.80*final.shape[0]))
        test_data = final.head(int(0.20*final.shape[0])+1)

        scores = final['Score'].get_values()
        len(scores)
```

```
Out[7]: 291336
```

```
In [11]: def convScores(scores):
             li = lambda x: 1 if x=='positive'  else 0
             final_scores = []
             for i in range(0,len(scores)):
                 final_scores.append(li(scores[i]))
             return final_scores
```

```
In [9]: def convToNpArray(arr):
            if(type(arr) == list):
                arr = np.array(arr)
                return arr
            else:
                return arr;
```

```
In [10]: def confusionMatrix(y_test,pred):
             tn, fp, fn, tp = confusion_matrix(y_test, pred).ravel()
             tpr = tp/(fn+tp)
             tnr = tn/(tn+fp)
             fnr = fn/(fn+tp)
             fpr = fp/(tn+fp)

             print("\n######### Confusion Matrix #########")
             print("TPR :%f \t TNR : %f\nFPR : %f \t FNR: %f"%(tpr,tnr,fpr,fnr))
```

1.1 BOW

```
In [13]: # Total data frame

         x_1 = final_counts_train[0:10000]


         # this is only Score/rating  of data

         y_1 = convScores(train_data['Score'].get_values())[0:10000]

         x_test = final_counts_test[0:3000]
         y_test = convScores(test_data['Score'].get_values())[0:3000]

         #x_1, x_test, y_1, y_test = train_test_split(x,y, test_size=0.3, random_state=0)

         x_1 = convToNpArray(x_1)
         x_test = convToNpArray(x_test)
         y_1 = convToNpArray(y_1)
         y_test = convToNpArray(y_test)
```

1.1.1 GridSearch CV

```
In [14]: tuned_parameters = {'C':[10**-2,10**0,10,10**2,10**4],
                             'gamma':[0.0001,0.001,0.01,0.1,1]}


         svc_model = SVC(kernel='rbf',class_weight='balanced')
         model = GridSearchCV(svc_model,tuned_parameters,
                              scoring='f1',cv=5,n_jobs=4)
```

```
        model.fit(x_1,y_1)

        print(model.best_estimator_)
        print("Score: ",model.score(x_test,y_test))
```

```
SVC(C=10, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Score:  0.9973674313651749
```

```
In [15]: best_svc_model = model.best_estimator_
         best_svc_model.fit(x_1,y_1)
         pred = best_svc_model.predict(x_test)
         confusionMatrix(y_test,pred)
```

```
######### Confusion Matrix #########
TPR :0.994749          TNR : 1.000000
FPR : 0.000000          FNR: 0.005251
```

1.1.2 RandomSearchCV

```
In [16]: from scipy.stats import expon
         import random
         g_val=[]
         for i in range(10):
             g_val.append(random.random())
         tuned_parameters = {'C':expon(scale=10),'gamma':g_val}

         model = RandomizedSearchCV(SVC(kernel='rbf',class_weight='balanced'),tuned_parameters
                                    scoring='f1',cv=5,n_jobs=4)


         model.fit(x_1,y_1)
         print(model.best_estimator_)
         print("Score: ",model.score(x_test,y_test))
```

```
SVC(C=1.6677202991576625, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.15473857217161935,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
Score:  1.0
```

```
In [17]: svc_Model = model.best_estimator_
         svc_Model.fit(x_1,y_1)
```

```
pred = svc_Model.predict(x_test)
confusionMatrix(y_test,pred)
```

```
######### Confusion Matrix #########
TPR :1.000000          TNR : 1.000000
FPR : 0.000000          FNR: 0.000000
```

Observation: here it took gamma as large val, we can see that its Overfit case as TPR is 1, from this we can uderstand that if gamma increases model will be overfitted and will not give appropriate results

1.2 avg W2v SVC

1.2.1 GridSearchCV

In [26]:
```
x_1 = sent_vectors_train[0:10000]
```

```
# this is only Score/rating  of data

y_1 = convScores(train_data['Score'].get_values())[0:10000]

x_test = sent_vectors_test[0:3000]
y_test = convScores(test_data['Score'].get_values())[0:3000]


x_1 = convToNpArray(x_1)
x_test = convToNpArray(x_test)
y_1 = convToNpArray(y_1)
y_test = convToNpArray(y_test)
```

In [27]:
```
tuned_parameters = {'C':[1,10,10**2,10**4],
                    'gamma':[0.1,0.3,0.5,0.7]}


svc_model = SVC(kernel='rbf',class_weight='balanced')
model = GridSearchCV(svc_model,tuned_parameters,
                     scoring='f1',cv=5,n_jobs=4)

model.fit(x_1,y_1)

print(model.best_estimator_)
print("Score: ",model.score(x_test,y_test))
```

```
SVC(C=100, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
```

```
  tol=0.001, verbose=False)
Score:  0.9299709724238027
```

```
In [28]: best_svc_model = model.best_estimator_
         best_svc_model.fit(x_1,y_1)
         pred = best_svc_model.predict(x_test)
         confusionMatrix(y_test,pred)
```

```
######### Confusion Matrix #########
TPR :0.961365          TNR : 0.152695
FPR : 0.847305          FNR: 0.038635
```

1.2.2 RandomSearchCV

```
In [29]: g_val=[]
         for i in range(10):
             g_val.append(random.random())
         tuned_parameters = {'C':expon(scale=10),'gamma':g_val}


         model = RandomizedSearchCV(SVC(kernel='rbf',class_weight='balanced'),tuned_parameters
                                    scoring='f1',cv=5,n_jobs=4)

         model.fit(x_1,y_1)

         print(model.best_estimator_)
         print("Score: ",model.score(x_test,y_test))
         best_svc_model = model.best_estimator_
         best_svc_model.fit(x_1,y_1)
         pred = best_svc_model.predict(x_test)
         confusionMatrix(y_test,pred)
```

```
SVC(C=13.121730339609785, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.14483763756915313,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
Score:  0.9385216773276475

######### Confusion Matrix #########
TPR :0.990623          TNR : 0.038922
FPR : 0.961078          FNR: 0.009377
```

1.3 TfIDF SVC with GridSearchCv

```
In [30]: x_1 = final_tf_idf_train[0:10000]
```

```
        y_1 = convScores(train_data['Score'].get_values())[0:10000]

        x_test = final_tf_idf_test[0:3000]
        y_test = convScores(test_data['Score'].get_values())[0:3000]

        x_1 = convToNpArray(x_1)
        x_test = convToNpArray(x_test)
        y_1 = convToNpArray(y_1)
        y_test = convToNpArray(y_test)
```

```
In [31]: tuned_parameters = {'C':[1,10,10**2,10**4],
                             'gamma':[0.1,0.3,0.5,0.7]}


         svc_model = SVC(kernel='rbf',class_weight='balanced')
         model = GridSearchCV(svc_model,tuned_parameters,
                              scoring='f1',cv=5,n_jobs=4)

         model.fit(x_1,y_1)

         print(model.best_estimator_)
         print("Score: ",model.score(x_test,y_test))
```

```
SVC(C=1, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.3, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Score:  0.9975559315660838
```

```
In [32]: best_svc_model = model.best_estimator_
         best_svc_model.fit(x_1,y_1)
         pred = best_svc_model.predict(x_test)
         confusionMatrix(y_test,pred)
```

```
######### Confusion Matrix #########
TPR :0.995124           TNR : 1.000000
FPR : 0.000000           FNR: 0.004876
```

1.3.1 RandomSearchCV

```
In [33]: g_val=[]
         for i in range(10):
             g_val.append(random.random())
         tuned_parameters = {'C':expon(scale=10),'gamma':g_val}
```

```
        model = RandomizedSearchCV(SVC(kernel='rbf',class_weight='balanced'),tuned_parameters
                                   scoring='f1',cv=5,n_jobs=4)

        model.fit(x_1,y_1)

        print(model.best_estimator_)
        print("Score: ",model.score(x_test,y_test))
        best_svc_model = model.best_estimator_
        best_svc_model.fit(x_1,y_1)
        pred = best_svc_model.predict(x_test)
        confusionMatrix(y_test,pred)
```

```
SVC(C=5.415547648717219, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.14175540339881265,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
Score:  1.0

######### Confusion Matrix #########
TPR :1.000000          TNR : 1.000000
FPR : 0.000000          FNR: 0.000000
```

2 Summary

Here is the Summary of all Featurizations with Grid and Random Search CV.

Observation: As Gamma val Increase the data is getting overfitted TPR is becoming 1 as Gamma val decreases its Underfitting FPR is tending towards 1.

```
<tr>
    <th>Featurization</th>
    <th>CV</th>
    <th>C</th>
    <th>Gamma</th>
    <th>F1 Score</th>
</tr>
<tr>
    <td>BOW</td>
    <td>GridSearch</td>
    <td>10</td>
    <td>0.01</td>
    <td>99.7</td>
</tr>
<tr>
    <td>BOW</td>
    <td>RandomSearch</td>
    <td>1.66</td>
    <td>0.15</td>
```

```
    <td>100</td>
</tr>
<tr>
    <td>Tf-Idf</td>
    <td>GridSearch</td>
    <td>1</td>
    <td>0.3</td>
    <td>99</td>
</tr>
<tr>
    <td>Tf-Idf</td>
    <td>RandomSearch</td>
    <td>5.4</td>
    <td>0.14</td>
    <td>100</td>
</tr>
<tr>
    <td>Avg W2V</td>
    <td>GridSearch</td>
    <td>100</td>
    <td>0.1</td>
    <td>93</td>
</tr>
<tr>
    <td>Avg W2V</td>
    <td>RandomSearch</td>
    <td>13.12</td>
    <td>0.14</td>
    <td>93.85</td>
</tr>
<tr>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
</tr>
```

Note: Its Response Time is high when compared to other models. ###### Considered 10000 data points for train and 3000 data points as test ###### Result may vary if we consider more data points.