1. PreProcessing_K-Means_Hier_DBScan

October 31, 2018

1 K-Means, Hierarchical & DBScan on Amazon Reviews (Part I)

1.1 Amazon Fine Food Review 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

1.2 Objective:

The code below would **clean the review text from html tags and punctuations and write it as a new column in the database and write it to disk**. This is further taken up in Part 2 to find accuracy of SVM on vectorized input data, for each of the 4 featurizations, namely BoW, tf-IDF, W2V, tf-IDF weighted W2V.

1.3 Data Loading & Cleaning

```
In [1]: 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


# using the SQLite Table to read data.
con = sqlite3.connect('./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 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
```

In [2]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId',
   axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')

#Deduplication of entries
final=sorted_data.drop_duplicates(subset={
    "UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[2]:
```
(364173, 10)
```

In [8]:
```
# value of HelpfulnessNumerator greater than HelpfulnessDenominator is not practically
# possible hence these two rows too are removed from calculations
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
final.shape
```

Out[8]:
```
(364171, 11)
```

2 Data Pre-Processing

In [4]:
```
import 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
#function to clean the word of any punctuation or special characters
def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [5]:
```
#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':
                #list of all words used to describe positive reviews
                        all_positive_words.append(s)
                    if(final['Score'].values)[i] == 'negative':
                #list of all words used to describe negative reviews reviews
                        all_negative_words.append(s)
                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 [6]: #adding a column of CleanedText which displays
        # the data after pre-processing of the review
        final['CleanedText']=final_string
        print(final['CleanedText'].head(3))
```

```
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...
Name: CleanedText, dtype: object
```

3 Save Cleaned Data

```
In [7]: # store final table into an SQlLite table for future use.
        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=True, index_label=None, chunksize=None, dtype=None)
```

4 Significant Points

1. **Duplication of reviews** are found with same userid and timestamp (Cleaned).
2. Found discrepancy issues with HelpfulnessDenominator (Cleaned).
3. final.sqlite db is **to be used for further processing** such as Text to Vector operations.
4. The preprocessing step is one time effort but the training & visualization steps require multiple runs. Hence, it is prudent to make preprocessing step independant, to avoid multiple runs.