1. Assignment_TSNE_PreProcessing

June 1, 2018

1 Assignment 2: TSNE Visualization (Part I)

1.1 Amazon Fine Food Review Dataset

1.1.1 Dataset High-Level Information

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.**

1. Number of reviews: **568,454**
2. Number of users: 256,059
3. Number of products: 74,258
4. Timespan: Oct 1999 - Oct 2012
5. Number of Attributes/Columns in data: 10

1.1.2 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:

Given a new 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 considered 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.

1.3 Overview of Assignment:

This assignment is **split in 5 parts** for ease of execution.

1) **Pre Processing: Removal of stop words, punctuation, special characters, HTML tags, stemming & lemmatization** along with Data Preparation and integrity check is done in this step. The output is written to a file which is read by each of the remaining parts.

2) **Bag of Words:** The output of 1st step is taken as input. The **sparse matrix** obtained from BoW text-to-vector method is fed to **Truncated SVD for dimensionality reduction**. t-SNE is done on TruncatedSVD dimension-reduced data & results are plotted.

3) **TF-IDF:** The output of 1st step is taken as input. The sparse matrix obtained from TF-IDF text-to-vector method is fed to Truncated SVD and then to t-SNE for plotting.

4) **Word2Vec:** The output of 1st step is taken as input. **Dense matrix is obtained from Word2Vec**. Hence, further dimensionality methods are not required. **Average-Word2Vec** is obtained for each review & Results are plotted after running t-SNE.

5) **TF-IDF weighted Word2Vec:** The output of 1st step is taken as input. Multiply the W2V value with TF-IDF weight and do the same steps as in Part 4.

1.4 Main Challenges Encountered:

1) **Heavy memory usage of t-SNE:** In my personal box with 4GB memory, t-SNE was giving frequent memory exceptions, which is obvious. Even after reducing the data points to a subset of < 5K, the program was throwing exception.

- **Solution Found:** Ran the code in "Google Colabs" Platform. The platform enabled me to use > 10GB memory and to run the code faster. But there are some hiccups which I faced such as frequent disconnections and kernel dying after 12-13GB memory usage. Also, packages need to be re-installed on every fresh run.

2) **Time Complexity of t-SNE:** t-SNE was taking **so many hours to execute. But we have to run t-SNE many times for different perplexity values and steps**, to check whether any shape or separation is coming out.

- **Solution Found:** Used the Multicore-TSNE implementation which does parallelization of t-SNE algorithm using multiple cores. The dataset is pretty huge having 3,64,000 data points. We have downsampled the data to 5-25K for faster execution. This significantly reduced execution time.

3) **Interfacing in Google Colabs:** The input file in google colabs cannot be loaded from local drives. It is capable to fetch the file from google cloud or gdrive. But google cloud is expensive and in gdrive connection is less stable.

- **Solution Found:** Installed a Drive FUSE wrapper & authorised tokens for colab to load the google drive as a fuse filesystem. After loading the file system, we can get all the input files from the 'drive/' directory as if in a local drive. **(code attached as 'Google_colab_dump.pdf')**

1.5 1. Data Loading & Cleaning

In [2]:
```
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 [3]:
```
#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[3]: (364173, 10)
```

```
In [4]: # 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[4]: (364171, 10)
```

2 2. Data Pre-Processing

```
In [5]: 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

        #function to clean the word of any html-tags
        def cleanhtml(sentence):
            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 [6]: #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)
                            #list of all words used to describe positive reviews
                            if (final['Score'].values)[i] == 'positive':
                                all_positive_words.append(s)
                            #list of all words used to describe negative reviews reviews
                            if(final['Score'].values)[i] == 'negative':
                                all_negative_words.append(s)
                        else:
                            continue
                    else:
                        continue
            #print(filtered_sentence)
            str1 = b" ".join(filtered_sentence) #final string of cleaned words

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

```
In [7]: #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 3. Save Cleaned Data

```
In [8]: # 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 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.