TSNE_W2V-TFIDF

July 8, 2018

1 TSNE Visualization of TF-IDF Word to Vector For Amazon Fine Food Reviews

This Dataset conists of reviews of fine foods from amazon. which includes: - Reviews from Oct 1999 - Oct 2012 - Total of 568,454 reviews - Given by 256,059 users - For 74,258 products

2 Data Cleaning and Loading

The same data is cleaned, by removing the duplicates and the reviews for which HelpfulnessNumerator is greater than HelpfulnessDenominator. So it is reduced to 364171 reviews with same 10 columns. This data with 364171 reviews is stored in a SQLite Database named 'final_sqlite' and the table for these reviews is 'Reviews'.

We load the data using SQLite in to pandas dataframe

```
In [1]: import sqlite3
        import pandas as pd
        import numpy as np

        conn = sqlite3.connect('final_sqlite')
        data = pd.read_sql_query('''select * from Reviews''', conn)
        print(data.shape)
        print(data.columns)
```

```
(364171, 12)
Index(['index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time',
       'Summary', 'Text', 'ClearedText'],
      dtype='object')
```

```
In [2]: def convert(x):
            '''To convert the reviews to positive or negative'''
            return 'Negative' if x<3 else 'Positive'
        score = data['Score'].map(convert)
        print(score.shape)
```

```
(364171,)
```

Here we determine a review as Positive or Negative by using the score. If score is more than 3 then it is considered as a positive and negative if it is less than 3 and will ignore if score is 3, as we can't decide whether it will fall into positive or negative category. The data which is in the Reviews table is queried/ saved without the reviews with score 3.

```
In [3]: import re
        def cleanhtml(sentence):
            '''To clean html-tags in the sentense'''
            cleanr = re.compile('<.*?>')
            cleantext = re.sub(cleanr, ' ', sentence)
            return cleantext
        def cleanpunc(sentence):
            '''To clean punctuation or special characters in the sentense'''
            cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
            cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
            return  cleaned
```

```
In [4]: list_of_sent=[]
        for sent in data['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 [5]: print(data['Text'].values[6])
        print("******************************************************************")
        print(list_of_sent[6])
```

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

3 TF-IDF Word to Vector

```
In [6]: import gensim
        w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

```
D:\Users\KiranPS\Anaconda\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
In [7]: words = list(w2v_model.wv.vocab)
        print(len(words))
```

```
33783
```

```
In [8]: # average Word2Vec
        # compute average word2vec for each review.
        sent_vectors = []
        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
            sent_vectors.append(sent_vec)
        print(len(sent_vectors))
        print(len(sent_vectors[0]))
```

```
D:\Users\KiranPS\Anaconda\lib\site-packages\ipykernel_launcher.py:14: RuntimeWarning: invalid



364171
50
```

```
In [9]: from sklearn.feature_extraction.text import TfidfVectorizer
        tf_idf_vect = TfidfVectorizer(ngram_range=(1,1))
        final_tf_idf = tf_idf_vect.fit_transform(data['Text'].values)
```

```
In [10]: print(final_tf_idf.shape)
```

```
(364171, 115281)
```

```
In [11]: # tfidf words/col-names
         tfidf_feat = tf_idf_vect.get_feature_names()
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

         tfidf_sent_vectors = []
         row=0
         for sent in list_of_sent[0:1000]:
             sent_vec = np.zeros(50)
             weight_sum =0
             for word in sent:
                 try:
                     vec = w2v_model.wv[word]
                     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

         print(tfidf_sent_vectors[0])
```

```
[ 0.72931442 -0.7407358  -0.5814222   0.51012776  0.29833264  0.41930299
  0.12800603  0.38393745 -0.30654065  0.10246312 -0.20858932 -0.73474739
  0.34302154 -0.02032635  0.08581883 -0.20262358  1.41278797 -0.5208605
 -0.34276316 -0.83262897  0.73443143  0.62860928 -0.78894525 -0.38847708
 -0.23678871  0.23029024  0.41237991 -0.35449512 -0.54021236  0.01595898
  0.03077845  0.40795082 -0.11024418  0.39467711 -0.27946289 -0.20622789
 -0.63418511 -0.43833407  0.26929678  0.0046896  -0.02967787 -1.0297157
 -0.16937571  0.21224442 -0.6771783  -0.1995969  -0.65302797 -0.75506137
  0.30510556 -0.02970457]
```

```
In [13]: dataframe = pd.DataFrame(tfidf_sent_vectors)
         print(dataframe.shape)
```

```
(1000, 50)
```

```
In [28]: from sklearn.preprocessing import StandardScaler
         dataframe = StandardScaler().fit_transform(dataframe)
         print(dataframe[0])
```

```
[ 0.7611664   0.03408173 -1.48504794  0.9008536  -0.0725867   0.01698244
 -0.07507977  1.26367505  0.1093421   0.34703378 -0.5350075  -0.51033332
  0.38723115  0.27285744 -0.42530542 -0.03306897  1.37489359 -0.42017789
  0.6418095  -1.48220927  0.79803986  0.18284128  0.24036663 -1.7404892
  0.37479865 -0.01881148  0.07069949 -0.54674261 -1.19417431  0.83096648
 -0.03991596  0.83868641 -0.61355462  0.02425933 -0.58162496  0.11344912
 -1.39452484 -0.06112462 -0.780642   -0.72580137 -0.42240513 -0.14870743
 -0.09184316  0.45350226 -0.82842624  0.38135924 -0.87591826 -1.16903055
 -0.4004034   0.28817864]
```

4 TSNE Visualization of TF-IDF W2V

```
In [30]: from sklearn.manifold import TSNE
         model = TSNE(n_components =2, random_state = 0)
         tsne_data = model.fit_transform(f_10k)
         print(tsne_data.shape)
```

```
(10000, 2)
```

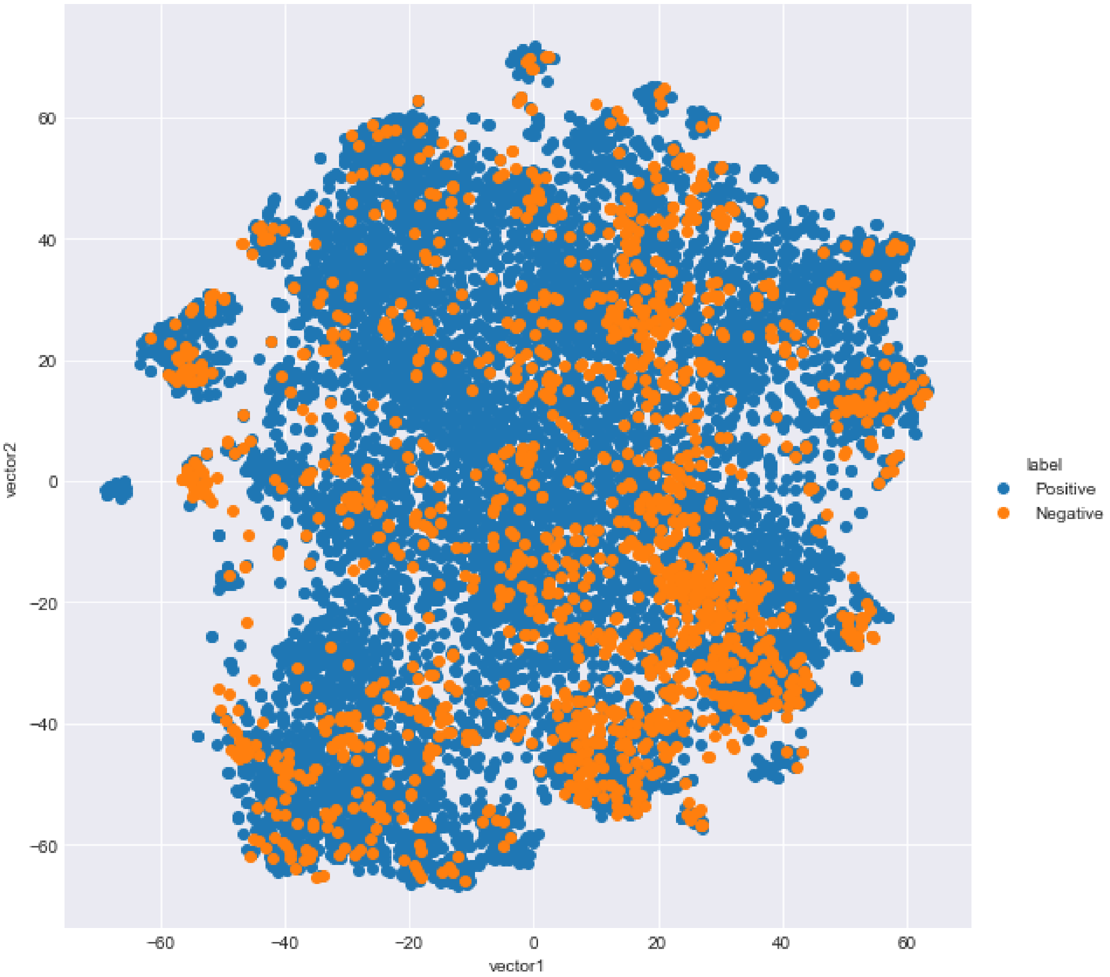
```
In [32]: dataframe = pd.DataFrame(tfidf_sent_vectors)
         f_10k = dataframe
         s_10k = score[0:10000]
         print(f_10k.shape)
         print(s_10k.shape)
```

```
(10000, 50)
(10000,)
```

```
In [34]: c_data = np.vstack((tsne_data.T, s_10k)).T
         frame = pd.DataFrame(c_data, columns =( 'vector1', 'vector2', 'label'))
         print(frame.shape)
```

```
(10000, 3)
```

```
In [35]: sns.set_style('darkgrid')
         sns.FacetGrid(frame, hue = 'label', size =8).map(plt.scatter,'vector1', 'vector2').add
         plt.show()
```



5 Observations:

Even these doesnot give much difference and we cannot bifercate or differentiate whether a review is positive or negative.But this plot is better when compared with the previous models.Though Both the reviews are spread accross the graph it is better when compared with other models. In order to get better results Machine learning algorithms would be handy and can predict more accurately than these models/visualizations.