Pradip Dharam

2. Exercise - t-SNE Visualization Amazon Dataset

last run 3 days ago · IPython Notebook HTML
using data from multiple data sources · Private Make Public

2. Mandatory Exercise 17.17 **Exercise: t-SNE visualization of Amazon reviews with polarity based color-coding** Get the Amaxon Food Reviews Data. Get the Bow, TF-IDF, Avg word2vec, TF-IDF weighted word2vec vector representations. Then do t-SNE visualization for each vector representation. Polarities for reviews are positive and negative, same should be visualized using t-SNE

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

# using the SQLite Table to read data.
#con = sqlite3.connect('./amazon-fine-food-reviews/database.sqlite')
#con = sqlite3.connect('../input/database.sqlite')
con = sqlite3.connect('../input/amazon-fine-food-reviews/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

display= pd.read_sql_query("""
#### SELECT *
#### FROM Reviews
#### WHERE Score != 3 AND UserId="AR5J8UI46CURR"
#### ORDER BY ProductID
""", con)
display

#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='q
uicksort', na_position='last')

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

#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100

display= pd.read_sql_query("""
#### SELECT *
#### FROM Reviews
#### WHERE Score != 3 AND Id=44737 OR Id=64422
#### ORDER BY ProductID
""", con)
display

final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]

"""
```

Out[1]:

```
', con)\ndisplay\n\nfinal=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]\n\n'
```

Text Preprocessing: Stemming, stop-word removal and Lemmatization.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [2]:
```
"""

# find sentences containing HTML tags
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/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
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned


"""
```

Out[2]:

```
'\n\n# find sentences containing HTML tags\nimport re\ni=0;\nfor sent in final[\'Text\'].value
s:\n    if (len(re.findall(\'<.*?>\', sent))):\n        print(i)\n        print(sent)\n
break;\n    i += 1;    \n\nimport re\n# Tutorial about Python regular expressions: https://pym
otw.com/2/re/\nimport string\nfrom nltk.corpus import stopwords\nfrom nltk.stem import PorterS
temmer\nfrom nltk.stem.wordnet import WordNetLemmatizer\n\nstop = set(stopwords.words(\'englis
h\')) #set of stopwords\nsno = nltk.stem.SnowballStemmer(\'english\') #initialising the snowba
ll stemmer\n\ndef cleanhtml(sentence): #function to clean the word of any html-tags\n    clean
r = re.compile(\'<.*?>\')\n    cleantext = re.sub(cleanr, \' \', sentence)\n    return cleante
xt\ndef cleanpunc(sentence): #function to clean the word of any punctuation or special charact
ers\n    cleaned = re.sub(r\'[?|!|\'|"|#]\',r\'\',sentence)\n    cleaned = re.sub(r\'[.|,|)|(|
\\|/]\',r\' \',cleaned)\n    return  cleaned\n\n\n'
```

In [3]:

```
"""

#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':
                        all_positive_words.append(s) #list of all words used to describe positive
 reviews
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative
 reviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
```

```
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("*************************************************************")

    final_string.append(str1)
    i+=1

    """
```

Out[3]:

```
'\n\n#Code for implementing step-by-step the checks mentioned in the pre-processing phase\n# t
his code takes a while to run as it needs to run on 500k sentences.\ni=0\nstr1=\' \'\nfinal_st
ring=[]\nall_positive_words=[] # store words from +ve reviews here\nall_negative_words=[] # st
ore words from -ve reviews here.\ns=\'\'\nfor sent in final[\'Text\'].values:\n    filtered_se
ntence=[]\n    #print(sent);\n    sent=cleanhtml(sent) # remove HTMl tags\n    for w in sent.s
plit():\n        for cleaned_words in cleanpunc(w).split():\n            if((cleaned_words.isa
lpha()) & (len(cleaned_words)>2)):    \n                if(cleaned_words.lower() not in sto
p):\n                    s=(sno.stem(cleaned_words.lower())).encode(\'utf8\')\n
     filtered_sentence.append(s)\n                    if (final[\'Score\'].values)[i] == \'pos
itive\': \n                        all_positive_words.append(s) #list of all words used to des
cribe positive reviews\n                    if(final[\'Score\'].values)[i] == \'negative\':\n
                    all_negative_words.append(s) #list of all words used to describe negati
ve reviews reviews\n                else:\n                    continue\n            else:\n
              continue \n    #print(filtered_sentence)\n    str1 = b" ".join(filtered_sentenc
e) #final string of cleaned words\n    #print("*************************************************
************************")\n    \n    final_string.append(str1)\n    i+=1\n    \n    '
```

In [4]:

```
"""

final['CleanedText']=final_string #adding a column of CleanedText which displays the data after p
re-processing of the review
cleanedText=final['CleanedText']
finalScore = final['Score']
print(finalScore.head(2))
print(cleanedText.head(2))
print("Shape cleanedText: ",cleanedText.shape)
print("Shape: finalScore",finalScore.shape)

final1=pd.DataFrame()
final1['Text'] = final['Text']
final1['CleanedText']  = final['CleanedText']
final1['Score'] = finalScore

final1.to_csv('final1.csv')

"""
```

Out[4]:

```
'\n\nfinal[\'CleanedText\']=final_string #adding a column of CleanedText which displays the da
ta after pre-processing of the review \ncleanedText=final[\'CleanedText\']\nfinalScore = final
[\'Score\']\nprint(finalScore.head(2))\nprint(cleanedText.head(2))\nprint("Shape cleanedText:
",cleanedText.shape)\nprint("Shape: finalScore",finalScore.shape)\n\nfinal1=pd.DataFrame()\nfi
nal1[\'Text\'] = final[\'Text\']\nfinal1[\'CleanedText\']  = final[\'CleanedText\']\nfinal1
[\'Score\'] = finalScore\n\nfinal1.to_csv(\'final1.csv\')\n\n'
```

In [5]:

```
%matplotlib inline

from sklearn.manifold import TSNE
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

final1=pd.read_csv("../input/final1/final1.csv")
Text = final1['Text']
cleanedText  = final1['CleanedText']
finalScore = final1['Score']

def tsne_visualize(data, labels):
  #model = TSNE(n_components=2, random_state=0)
  model = TSNE(n_components=2, random_state=0, perplexity = 50, n_iter=5000)
  # configuring the parameteres,  # the number of components = 2,  # default perplexity = 30,  #
 default learning rate = 200
  # default Maximum number of iterations for the optimization = 1000
  tsne_data = model.fit_transform(data)  # creating a new data frame which help us in ploting th
e result data
  tsne_data = np.vstack((tsne_data.T, labels)).T
  tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))
  # Ploting the result of tsne
  sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
```

```
plt.show()
```

#

BAG OF WORDS

In [6]:
```
count_vect = CountVectorizer() #in scikit-learn
final_counts_bow = count_vect.fit_transform(cleanedText.values)
final_counts_bow.shape
```

Out[6]:
```
(364171, 71632)
```

In [7]:
```
type(final_counts_bow)
```

Out[7]:
```
scipy.sparse.csr.csr_matrix
```

In [8]:
```
final_counts_bow
```

Out[8]:
```
<364171x71632 sparse matrix of type '<class 'numpy.int64'>'
        with 11460605 stored elements in Compressed Sparse Row format>
```

Processing only 5000 records since entire dataset processing takes too much RAM

In [9]:
```
ndatapoints=5000
data = final_counts_bow[0:ndatapoints,:]
labels = finalScore[0:ndatapoints]
```

- Converting sparce to dense metrics to standardize the data
- Standardizing the data

In [10]:
```
# Converting sparce to dense metrics to standardize the data
data_dense=data.todense()
```

```
data_dense=data.todense()

# Standardizing the data
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler().fit_transform(data_dense)
standardized_data.shape
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

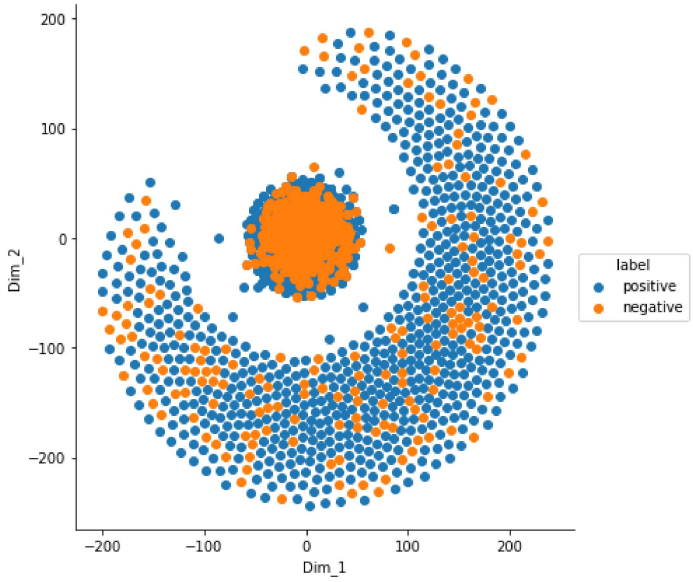
Out[10]:

```
(5000, 71632)
```

t-SNE visualization for BAG OF WORDS

In [11]:

```
# TSNE
tsne_visualize(standardized_data,labels)
```



#

TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY : TF-IDF

In [12]:
```
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(cleanedText.values)
final_tf_idf.shape
```

Out[12]:
```
(364171, 71632)
```

Processing only 5000 records since entire dataset processing takes too much RAM

In [13]:
```
ndatapoints=5000
data = final_tf_idf[0:ndatapoints,:]
labels = finalScore[0:ndatapoints]
```

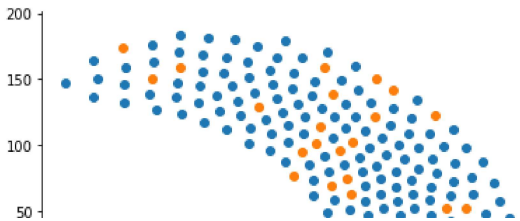
- Converting sparce to dense metrics to standardize the data
- Standardizing the data

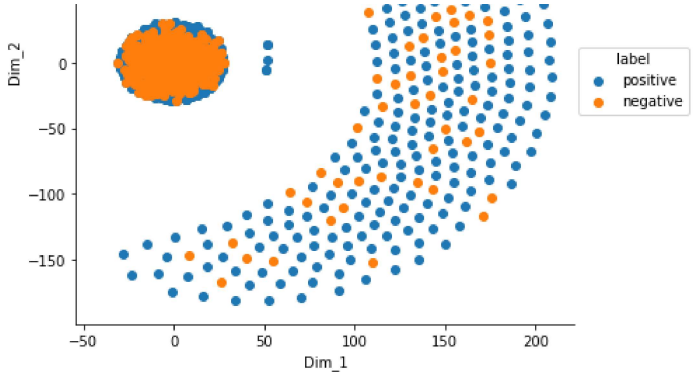
In [14]:
```
# Converting sparce to dense metrics to standardize the data
data_dense=data.todense()

#Standardizing the data
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler().fit_transform(data_dense)
```

t-SNE visualization for TF-IDF

In [15]:
```
tsne_visualize(standardized_data,labels)
```





#

OBSERVATIONS:

- **t-SNE visualization looks almost similar for both BoW and TF-IDF**

#

I'm using google word to vector to measure semantic similarities

Word2Vec

In [16]:
```
#"""
# Using Google News Word2Vectors
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
model = KeyedVectors.load_word2vec_format('../input/googlenews-vectors-negative300.bin/GoogleNe
ws-vectors-negative300.bin', binary=True)
#model.wv['computer']; #print(model.wv.similarity('woman', 'man')); #print(model.wv.similarity('q
ueen', 'queen'))
#print(model.wv.most_similar('tasty') ) # "tasti" is the stemmed word for tasty, tastful
#"""
```

#

Average Word2Vec

Processing only 5000 records since entire dataset processing takes too much RAM

In [17]:
```
ndatapoints=5000
data = cleanedText[0:ndatapoints]
labels = finalScore[0:ndatapoints]
```

Getting vector for each word in review and creating new scentense vector such that its an average of word2vector of all the words in review text

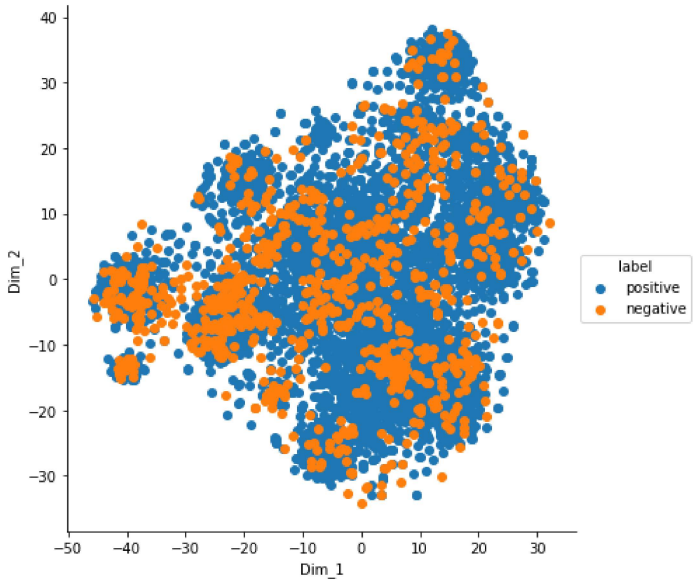
In [18]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in data: # for each review/sentence
    #print("Sentense : ", sent, "Type: ",type(sent))
    sent_vec = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    #for word in str(sent).split():
        #print("word: ", str(word))
    for word in str(sent).split(): # for each word in a review/sentence
        try:
            #print("word : ", word)
            vec = model.wv[word]
            #print("vec : ", vec.shape)
            sent_vec += vec
            cnt_words += 1
        except:
            #print("Exception")
            pass
    sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
#print("Shape: ",sent_vectors.shape)
#"""
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:13: DeprecationWarning: Call to d
eprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  del sys.path[0]


5000
300
```

t-SNE visualization for average word2vector

In [19]:
```
tsne_visualize(sent_vectors, labels)
```



#

TF-IDF weighted word2vector

Processing only 5000 records since entire dataset processing takes too much RAM

In [20]:
```
ndatapoints=5000
data = cleanedText[0:ndatapoints]
labels = finalScore[0:ndatapoints]
```

Getting vector for each word in review and creating new scentense vector such that word2vector is been weighted with respective TF-IDF of the review

I have commented the print statements which I did added to undestand the execution throughly for few data points

In [21]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in data: # for each review/sentence
    sent_vec = np.zeros(300) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    #sent='littl book make son laugh loud recit car drive along alway sing refrain hes learn whal
e india droop love new word book introduc silli classic book will bet son still abl recit memori
 colleg'
    #print("Sentense : ", sent)
    for word in sent.split(): # for each word in a review/sentence
        try:
            #print("Word: ", word)
            vec = model.wv[word]
            #print("Vector: ", vec, " Size: ", vec.shape)
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            #print("tfidf: ", tf_idf)
            sent_vec += (vec * tf_idf)
            #print("sent_vec Calculated")
            weight_sum += tf_idf
        except:
           # print("Exception")
            #e = sys.exc_info()[0]
            #print("Exception: ",e)
            ##write_to_page( "<p>Error: %s</p>" % e )
            pass
    zero_weight_sum_count=0
    if(weight_sum != 0):
```

```
    if(weight_sum != 0):
        sent_vec /= weight_sum
    else        :
        zero_weight_sum_count += 1
    tfidf_sent_vectors.append(sent_vec)
    row += 1

print(len(tfidf_sent_vectors))
print(len(tfidf_sent_vectors[0]))
#print("zero_weight_sum_count: ", zero_weight_sum_count)

#print("tfidf_sent_vectors : ")
#print(tfidf_sent_vectors)
#for vec in tfidf_sent_vectors:
 #   for i in vec:
 #       if(i!=0):
 #           print("i=",i)
```

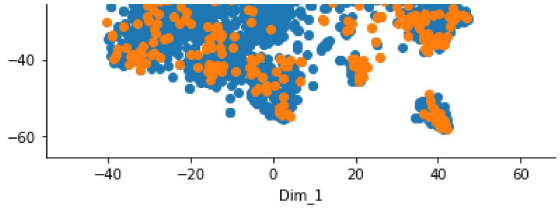
```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:15: DeprecationWarning: Call to d
eprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  from ipykernel import kernelapp as app


5000
300
```

t-SNE visualization for TF-IDF weighted word2vector

In [22]:

```
# TSNE
tsne_visualize(tfidf_sent_vectors, labels)
```



#

OBSERVATIONS:

- **t-SNE visualization looks almost similar for both Avg word2vec and TF-IDF weighted word2vec**
- **t-SNE visualization for both BoW and TF-IDF looks different than both Avg word2vec and TF-IDF weighted word2vec**

#