39.15.Amazon_food_review_truncated_SVD

July 14, 2018

1 Amazon food review dataset apply truncated SVD

Data set from https://www.kaggle.com/snap/amazon-fine-food-reviews

2 Objective

1. Take 2000 words by TFIDF importance
2. Calculate cooccurance matrix with neighbourhood of size 5 and count how many times wi occur in context of wj
3. Then do truncated SVD
4. try multiple value of k(find optimal k by amount of variance explained)[use singular value]
5. cluster(kmeans k=50) word vector for top 2000
6. word cluster together should be related

3 Import data and libraries

```
In [97]: 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
         from sklearn.cross_validation import train_test_split,KFold
         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.grid_search import GridSearchCV
         from sklearn.linear_model import LogisticRegression

         con = sqlite3.connect('database.sqlite')

         #get only +ve and -ve review
         raw_data = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3""", con)
```

4 Data preprocessing

```
In [98]: filtered_data=raw_data
         # Score>3 a positive rating, and 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

         filtered_data.sample(5)
         filtered_data['Score'].value_counts()

         #Sorting data according to ProductId in ascending order
         sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fal

         #Deduplication of entries for same profilename,userid, time, text and take first elem
         sorted_data=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}
```

```
In [206]: #take only 50000 data
          print('total data \n',sorted_data['Score'].value_counts())
          #clean_data=sorted_data.sample(frac=1).groupby('Score').head(10000)
          #take stratified sampling i.e. positive and negative reviews are proportionate to ra
          #testing
          _ , clean_data = train_test_split(sorted_data, test_size = 50000, random_state=1,stra
          clean_data['Score'].value_counts()
```

```
total data
 positive    307063
negative     57110
Name: Score, dtype: int64
```

```
Out[206]: positive    42159
```

```
negative     7841
Name: Score, dtype: int64
```

```
In [207]: # Clean html tag and punctuation
          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

          #substitute html tag and punctuation
          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 c
              cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
              cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
              return  cleaned
          print(sno.stem('tasty'))

          i=0
          str1=' '
          mystop={'of','four','one','would'}
          final_string=[]
          all_positive_words=[] # store words from +ve reviews here
          all_negative_words=[] # store words from -ve reviews here.
          s=''
          #Create new catagory as Cleanedtext after removing htmltag and punctuation and upper
          for sent in clean_data['Text'].values:
              filtered_sentence=[]
              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) & (cleaned_words.lower() not i
                              s=(sno.stem(cleaned_words.lower())).encode('utf8')
                              filtered_sentence.append(s)
                              if (clean_data['Score'].values)[i] == 'positive':
                                  all_positive_words.append(s) #list of all words used to desc
                              if(clean_data['Score'].values)[i] == 'negative':
                                  all_negative_words.append(s) #list of all words used to desc
                          else:
                              continue
                      else:
```

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

    final_string.append(str1)
    i+=1

clean_data['CleanedText']=final_string
print(clean_data.shape)
#Sort data on timestamp
clean_data=clean_data.sort_values(by=['Time'],ascending=False)
#clean_data
clean_data['CleanedText'].sample(2)
clean_data['CleanedText'].iloc[0]
```

```
tasti
(50000, 11)


C:\Users\suman\Anaconda3\lib\site-packages\ipykernel_launcher.py:52: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
Out[207]: b'use lot coconut late granola cake cup cake etc particular brand serv purpos well p
```

5 Get top 2000 words by TFIDF score and create co-occurence matrix by window 5

```
In [209]: x=clean_data['CleanedText'].values
          y = clean_data['Score']
          tf_idf_vect = TfidfVectorizer()
          final_counts = tf_idf_vect.fit_transform(x)
          #use the same vectors to convert test data
          indices = np.argsort(tf_idf_vect.idf_)[::-1]
          features = tf_idf_vect.get_feature_names()
          #testing
          top_n = 2000
          top_features = [features[i] for i in indices[:top_n]]
          print (top_features[0:20])
          print('len of top feature',len(top_features))

          #remove other words from review
          final_string=[]
          all_string=[]

          i=0
```

```
for sent in clean_data['CleanedText'].values:

    filtered_sentence=[]
    for w in sent.decode('utf8').split():
                if(w in top_features):
                    filtered_sentence.append(w.encode('utf8'))

                else:
                    continue
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    if ((i<5) & (str1!=b'')):
        print('sentence copy',str1)
    if (str1.decode('utf8') !=''):
      i=i+1
      final_string.append(str1)
      all_string.append(str1)

#clean_data['CleanedText']=final_string
#Now final_string is ready to work with
#print(clean_data['CleanedText'].shape)
all_string[0:20]
```

```
['çaykur', 'jail', 'jaegermeist', 'jagar', 'jager', 'jagermeist', 'jagger', 'jaguar', 'jaim',
len of top feature 2000
sentence copy b'howevert'
sentence copy b'jeesh'
sentence copy b'kadoda'
sentence copy b'hula'
sentence copy b'gough galantin'
```

```
Out[209]: [b'howevert',
           b'jeesh',
           b'kadoda',
           b'hula',
           b'gough galantin',
           b'gami',
           b'likelt',
           b'gallopin',
           b'krapelien holm',
           b'lakritz',
           b'gayelord',
           b'fuggedaboud',
           b'jeweltim jeweltim',
           b'happitud geen',
           b'hummingbyrd',
           b'gnash',
           b'jagermeist',
           b'gravey',
           b'intersess keiki',
           b'ingrededi']
```

```
In [210]: #Convert to cooccurance mat
          #type(final_string)
          print(len(top_features))
          window=5
          len1=len(top_features)
          #print(len)
          m=np.zeros([len1,len1])
          columns=top_features
          rows=top_features
          df=pd.DataFrame(m,columns=columns,index=rows)
          #print(df)

          def cal_occ(sentence,df):
              sen=sentence.split()
              l=len(sen)-1
              for i,word in enumerate(sen):
                  #loop through every sentence in a window and get neigherest words and keep a
                  for j in range(max(i-window,0),min(i+window+1,l+1)):
                      if word!=sen[j]:
                          #print('printing',word,sen[j])
                          df[word][sen[j]]+=1

          for sentence in final_string:
              #print('call',sentence)
              cal_occ(sentence.decode('utf8'),df)

          print(df.shape)
```

```
2000
(2000, 2000)
```

```
In [211]: from sklearn.preprocessing import StandardScaler
          #Get cooccuring words for a given word
          print('The most similar word like ',df.index[4])
          aa=df.iloc[4]
          bb=aa.sort_values(ascending=False)
          print(type(bb))
```

```
The most similar word like  jager
<class 'pandas.core.series.Series'>
```

```
In [212]: #df
```

6 Create countvectorizer using cooccurence matrix

```
In [213]: #count_vect = CountVectorizer(vocabulary=top_features) #in scikit-learn
          #X = count_vect.fit_transform(final_string)

          #print(X.shape)

          #Cooccurance matrix
          #X = (X.T * X) # this is co-occurrence matrix in sparse csr format
          #X.setdiag(0) # sometimes you want to fill same word cooccurence to 0
          #print(X.todense())

          #print(count_vect.vocabulary_)

          #Create truncated SVD
          from sklearn.decomposition import TruncatedSVD

          #Try different component
          l=[20,50,100,150,200,250]
          for i in l:
            svd = TruncatedSVD(n_components=i, n_iter=7, random_state=0)
            svd.fit(df.values)
            #print(svd.explained_variance_ratio_)
            l1=svd.explained_variance_ratio_
            print('% variance explained with component ',i,svd.explained_variance_ratio_.sum()
            #print('singular values',svd.singular_values_)

          #So looks like with 25 component 96% variance is explained
```

```
% variance explained with component  20 0.614688953311
% variance explained with component  50 0.775927938321
% variance explained with component  100 0.862135844503
% variance explained with component  150 0.889968288638
% variance explained with component  200 0.91780506569
% variance explained with component  250 0.945634269034
```

SO by 250 component 95% variance is explained # Use SVD

```
In [229]: #VT = svd.components_
          #TruncatedSVD is basically a wrapper around sklearn.utils.extmath.randomized_svd; yo

          from sklearn.utils.extmath import randomized_svd

          U, Sigma, VT = randomized_svd(df.values,
                                        n_components=250,
                                        n_iter=50,
                                        random_state=0)
```

```
        print('U value\n')
        #print(U)
        print('sigma value\n')
        #print(Sigma)
        print('VT value\n')
        #print(VT)
        print(U.shape,Sigma.shape,VT.shape)
        print('1st word vector representation',df.index[0])
```

```
U value

sigma value

VT value

(2000, 250) (250,) (250, 2000)
1st word vector representation çaykur
```

7 Form cluster of 10 using those important words SVD value

8 Get top few words similar to a random word

```
In [230]: from random import randint
          j=randint(0, 2000)
          print('1st word vector representation',df.index[j],' for j',j)#print(U.shape[0])
          #Calculate distance of this word with all words and sort in descending order
          #take log transform
          U=np.log(U+1)
          l=[]
          for i in range(U.shape[1]):
            a=np.linalg.norm(U[j]-U[i])
            l.append(a)
          l1=sorted(range(len(l)), key=lambda k: l[k])
          #print(l)
          print('top 10 words similar to ',df.index[j],' are ',df.index[l1[0:10]])
```

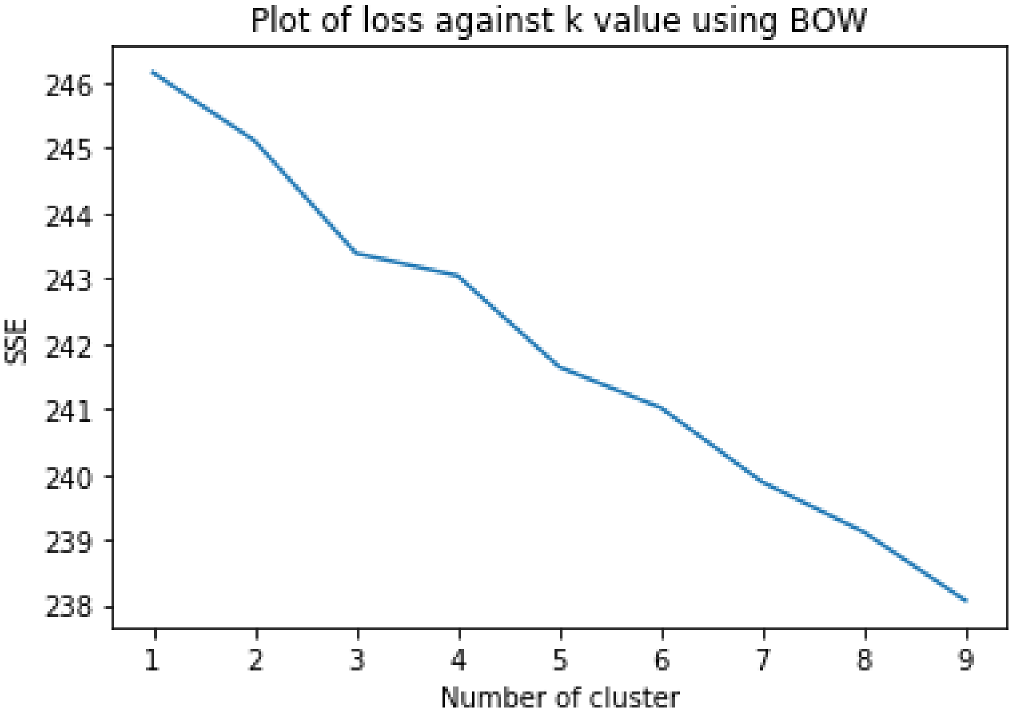
```
1st word vector representation granmder  for j 1331
top 10 words similar to  granmder  are  Index(['jitterbean', 'jot', 'josephus', 'jedi', 'jeera
       'jitterbug', 'jem', 'jolen'],
      dtype='object')
```

```
In [233]: from sklearn.cluster import KMeans
          # Now U is vec presentation of words
          n_clusters=10
          kmeans=KMeans(n_clusters=10, random_state=0).fit(U)
          kmeans.cluster_centers_
```

```
sse = {}
for k in range(1, 10):
    kmeans = KMeans(init='k-means++',n_clusters=k, max_iter=100).fit(U)
    sse[k] = kmeans.inertia_ # Inertia: Sum of distances of samples to their closest
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.title("Plot of loss against k value using BOW")
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()

#a=np.where(kmeans.labels_ == 1)[0]
#b=np.where(kmeans.labels_ == 0)[0]
#check 5 text for cluster 1

kmeans = KMeans(init='k-means++',n_clusters=50, max_iter=100).fit(U)
n_clusters=50
print(a.shape)
for i in range(n_clusters):
  a=np.where(kmeans.labels_ == i)[0]
  print('in cluster \n',i)
  print(a[0:10])
  k=0
  for j in a:
      k=k+1
      if (k<10):
        print(top_features[j][:10])
```



```
(1,)
in cluster
 0
[0 1 2 3 4 5 6 7 8 9]
çaykur
jail
jaegermeis
jagar
jager
jagermeist
jagger
jaguar
jaim
in cluster
 1
[882]
kittredg
in cluster
 2
[1289]
gradul
in cluster
 3
[968]
```

```
kelsey
in cluster
 4
[22]
jab
in cluster
 5
[1845]
hushpuppi
in cluster
 6
[1892]
hotown
in cluster
 7
[632]
lanolin
in cluster
 8
[1570]
gingerspic
in cluster
 9
[1966]
hairstyl
in cluster
 10
[1063]
knowlton
in cluster
 11
[1213]
gleam
in cluster
 12
[1280]
grenad
in cluster
 13
[1573]
giovanni
in cluster
 14
[984]
kwazulu
in cluster
 15
[678]
```

```
laurenc
in cluster
 16
[123]
istelf
in cluster
 17
[1017 1806]
laevulos
higer
in cluster
 18
[1661]
genui
in cluster
 19
[1654]
genovo
in cluster
 20
[1636]
genom
in cluster
 21
[1733]
homag
in cluster
 22
[987]
kushka
in cluster
 23
[827]
lindo
in cluster
 24
[ 654 1665]
lamma
genteel
in cluster
 25
[1863]
hydron
in cluster
 26
[1440]
gaul
in cluster
```

```
 27
[992]
kuechenmei
in cluster
 28
[281]
implant
in cluster
 29
[776]
lisey
in cluster
 30
[1589]
gim
in cluster
 31
[1385]
gravita
in cluster
 32
[1078]
komissbrot
in cluster
 33
[160]
juarez
in cluster
 34
[1692]
hobbits
in cluster
 35
[1341]
grandaroma
in cluster
 36
[387]
immacul
in cluster
 37
[373]
iceberg
in cluster
 38
[1487]
fue
in cluster
```

```
 39
[1400]
ganoderma
in cluster
 40
[1797]
hijiki
in cluster
 41
[448]
insalata
in cluster
 42
[798]
licken
in cluster
 43
[1796]
hijo
in cluster
 44
[101]
ipodo
in cluster
 45
[86]
ironwork
in cluster
 46
[1430]
gastroente
in cluster
 47
[685 963]
latitud
kensington
in cluster
 48
[537]
ingredien
in cluster
 49
[841]
kiefer
```

9 Observation

Most of the cluster contains 1-2 words and most of the words in one cluster To find the similar word its not giving proper words, maybe non engligh words and text cleanning is required much

```
In [ ]:
```

Ignore the above 2 plots those plots are plotted below again