Truncated SVD Kmeans on Amazon Fine FOod Reviews

In [1]:
```
#importing necessary packages
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from sklearn.feature_extraction.text import CountVectorizer,TfidfVector
izer
import pickle
import sklearn.cross_validation
from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
```

```
C:\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: Deprecat
ionWarning: This module was deprecated in version 0.18 in favor of the
model_selection module into which all the refactored classes and functi
ons are moved. Also note that the interface of the new CV iterators are
different from that of this module. This module will be removed in 0.2
0.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Reading already Cleaned, Preprocessed data from database

After removing stopwords, punctuations, meaningless characters, HTML tags from Text and done stemming. Using it directly as it was alredy done in prevoius assignment

In [2]:
```
#Reading
conn= sqlite3.connect('cleanedTextData.sqlite')
data= pd.read_sql_query('''
SELECT * FROM Reviews
''',conn)
data=data.drop('index',axis=1)
data.shape
```

Out[2]:
```
(364171, 11)
```

In [3]:
```
data.columns
```

Out[3]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerato
r',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',
       'CleanedText'],
      dtype='object')
```

In [4]:
```
data['CleanedText'].head(3)
```

Out[4]:
```
0    witti littl book make son laugh loud recit car...
1    rememb see show air televis year ago child sis...
2    beetlejuic well written movi everyth act speci...
Name: CleanedText, dtype: object
```

Sorting on the basis of 'Time' and taking top 100k pts

This data has time attribute so it will be reasonable to do time based splitting instead of random splitting.

So, before splitting we have to sort our data according to time and here we are taking 100k points from our dataset(population)

In [5]:
```
data["Time"] = pd.to_datetime(data["Time"], unit = "ms")
data = data.sort_values(by = "Time")
```

In [6]:
```
#latest 100k points according to time
data= data[:100000]
len(data)
```

Out[6]: 100000

In [7]:
```
Xdata= data['CleanedText']
```

SelfDefined Functions

In [27]:
```
from sklearn.cluster import KMeans
# Using same kmeans function from previous assignment
def optimalKmeans(Xdata):
    '''
    Returns the optimal k by plotting curve on inertia and k with minim
al inertia
    '''
    param_K = [2,4,6,8,10,12]
    inertia={}
    for K in param_K:
        model= KMeans(n_clusters=K, init= 'k-means++', precompute_dista
nces= True, n_jobs= -1)
        model.fit(Xdata)
        inertia[K]= model.inertia_
    plt.plot(list(inertia.keys()), list(inertia.values()))
    plt.xlabel("No. of cluster")
    plt.ylabel("Inertia")
    plt.show()
    bestK= min(inertia, key=inertia.get)
    print('The best K according to min inertia is ',bestK)
    return bestK
```

In [9]:
```
from wordcloud import WordCloud
```

```
    #prints wordclouds of all clusters seperatly
    def wordCloud(clusterPerReview,reviewText,k):
        '''
        Prints  wordclouds of all the clusters given cluster number per rev
    iew and review text
        '''
        clusterGroup={}
        i=0
        for c in clusterPerReview:
            if c in clusterGroup.keys():
                clusterGroup[c]+= reviewText[i]
            else:
                clusterGroup[c]=reviewText[i]
            i+=1
        print('So we have',k,'clusters here representing in wordcloud:')
        for i in list(set(cluster)):
            print('Cluster Number',i+1,':')
            plt.figure()
            plt.imshow(WordCloud().generate(clusterGroup[i]))
            plt.axis("off")
```

In [67]:

```
#https://towardsdatascience.com/overview-of-text-similarity-metrics-339
7c4601f50
#prints top 20 cosine similar words close to passed word
from sklearn.metrics.pairwise import cosine_similarity
def cosineSimilarWords(word):
    similarity = cosine_similarity(Xsvd)
    word_vect = similarity[FeatureWords.index(word)]
    df= pd.DataFrame({'x':FeatureWords, 'y':word_vect})
    df=df.sort_values(by='y',ascending=0).set_index(np.arange(3000))
    print("Similar Words for",word,':-')
    print(df['x'][1:21])
```

TFIDF Vectorization

In [11]:

```
# generating vetor out of text using tfidf only 3000 uses [[idf score]]
tfidfModel=TfidfVectorizer(use_idf=True, max_features=3000 )
```

```
XtfidfV= tfidfModel.fit_transform(Xdata)
```

Building Co-occurrence Matrix

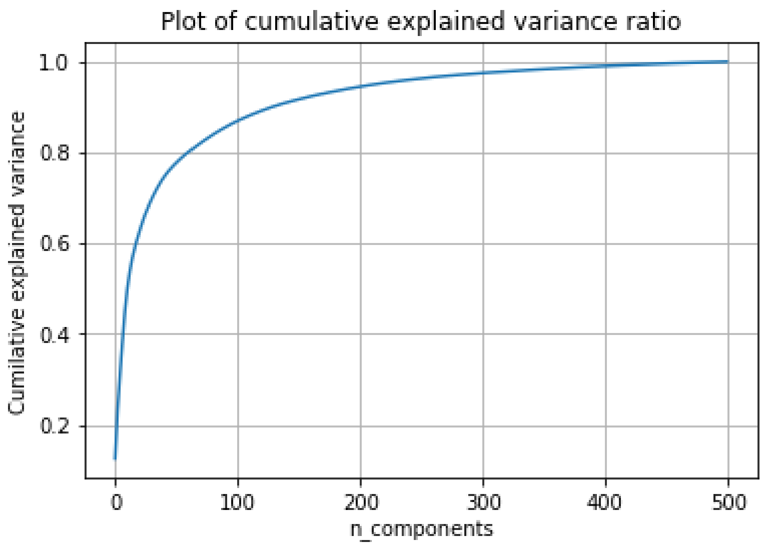
In [12]:

```
#getting feature names
#taking context window of size five 2 left and 2 right of the word
FeatureWords= tfidfModel.get_feature_names()
#half of context window
NearNeigh = 2
coocMatrix = np.zeros((3000,3000))
for row in list(Xdata.values):
    Sentance = row.split()
    for index,word in enumerate(Sentance):
        if word in FeatureWords:
            #setting left limit not preceed 0
            #settng right limit not exceed end of array
            lvalue= max(index-NearNeigh,0)
            rvalue= min(index+NearNeigh,len(Sentance)-1)
            contextWindow= range(lvalue, rvalue + 1)
            for i in contextWindow:
                if Sentance[i] in FeatureWords:
                    #one is added if word in column found in context wi
ndow of querry word
                    coocMatrix[FeatureWords.index(word),FeatureWords.in
dex(Sentance[i])] += 1
                else:
                    pass
        else:
            pass
```

In [13]:

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 500)
Xsvd = svd.fit_transform(coocMatrix)
expVar= svd.explained_variance_
cumulativeVariance = np.cumsum(expVar / np.sum(expVar))
plt.plot(cumulativeVariance)
plt.title('Plot of cumulative explained variance ratio')
```

```
plt.xlabel('n_components')
plt.ylabel('Cumilative explained variance')
plt.grid()
plt.show()
```



Max variance explained around 100 - 150, taking 120 as n_component

In [14]:
```
#applying SVD on our coocurance matrix
svd = TruncatedSVD(n_components = 120)
Xsvd = svd.fit_transform(coocMatrix)
```

In [15]:
```
Xsvd.shape
```

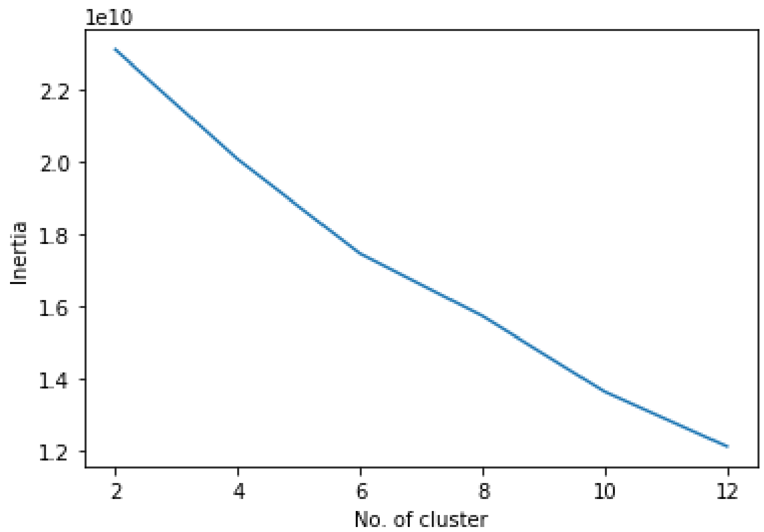
Out[15]:
```
(3000, 120)
```

So our Tfidf Vector is truncated to length of 120, Now applying Kmeans

KMeans

Applying Kmeans on the resultant vectors after Truncated SVD

In [29]:
```
k = optimalKmeans(Xsvd)
```




```
The best K according to min inertia is  12
```

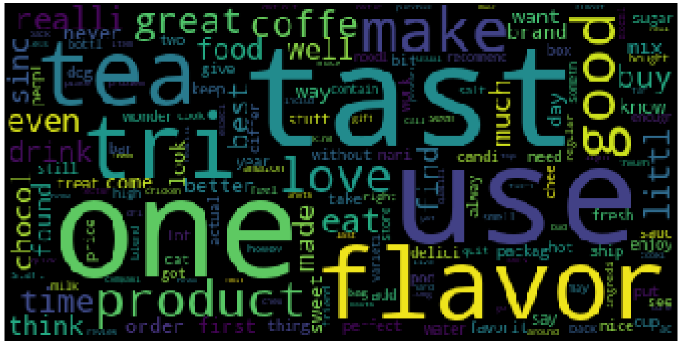
WordCloud of Kmeans's clusters

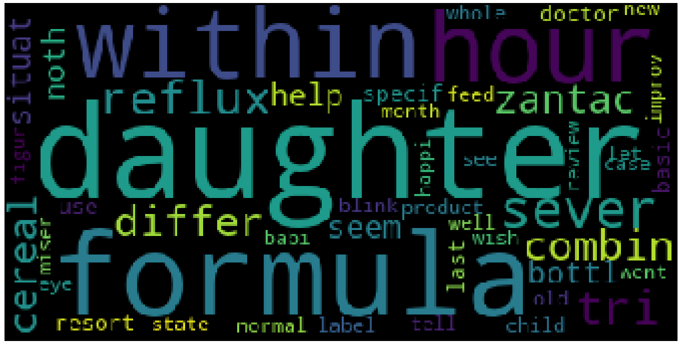
In [30]:
```
Kmeans= KMeans(n_clusters=k, init= 'k-means++', precompute_distances= True, n_jobs= -1)
cluster = Kmeans.fit_predict(Xsvd)
```

In [31]:
```
wordCloud(cluster,Xdata,k)
```

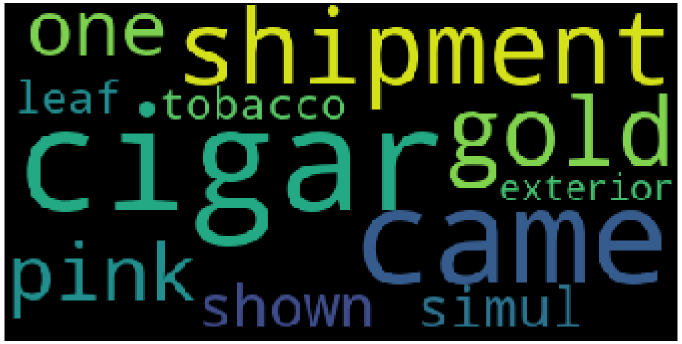
```
So we have 12 clusters here representing in wordcloud:
Cluster Number 1 :
Cluster Number 2 :
Cluster Number 3 :
Cluster Number 4 :
```

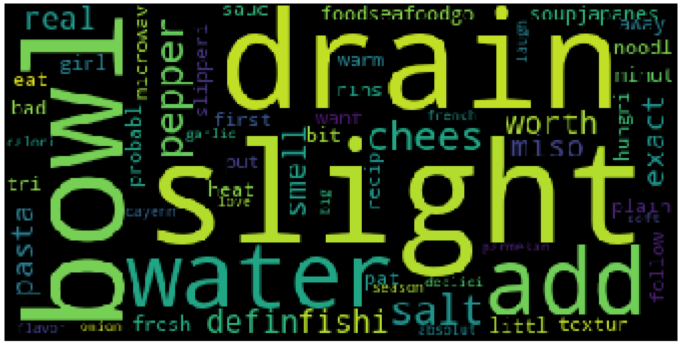
```
Cluster Number 5 :
Cluster Number 6 :
Cluster Number 7 :
Cluster Number 8 :
Cluster Number 9 :
Cluster Number 10 :
Cluster Number 11 :
Cluster Number 12 :
```

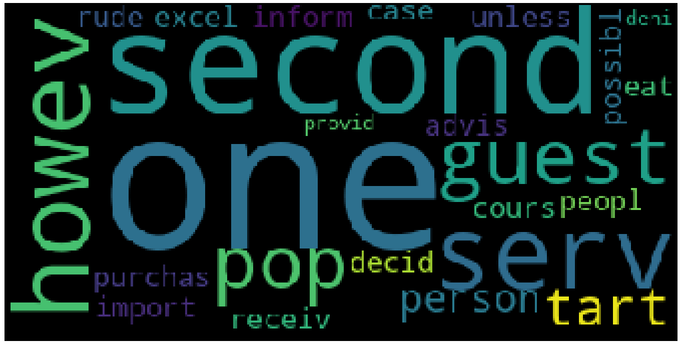





quit
cereal
found
littl
wheat
allergi
absolut
one
eat
love
best
frequent
crazi
mil


one
shipment
leaf
tobacco
cigar
gold
exterior
came
pink
shown
simul

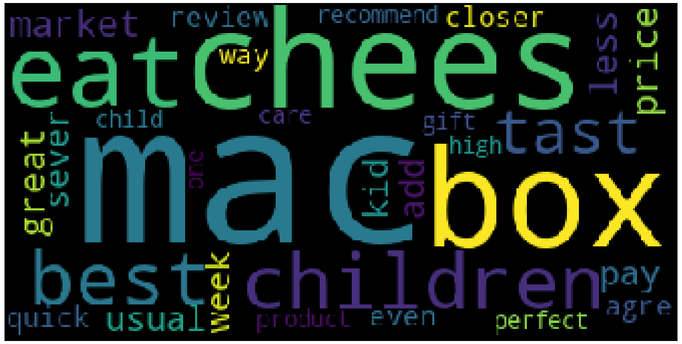

real
drain
foodseafoodgo
soupjapanes
girl
warm
noodl
eat
bad
pepper
first
worth
bit
chees
miso
exact
tri
heat
smell
bowl
slight
probabl
pasta
water
add
salt
definfishi
littl


milk
white
coffe
blend
chocol
regular
retail
morn
delight
high
mistak
joe
libat
worth
espresso
maker
need
tablespoon
quantiti
bitter
prepari
purchas
sweet
local
tri
pleas
strong
smaller
palat
astonish
brew
warm
cup
perk
togeth
mixtur
otherwis

rude
excel
inform
case
unless
second
howev
possibl
eat
provid
advis
one
guest
cours
peopl
pop
decid
serv
purchas
import
receiv
person
tart

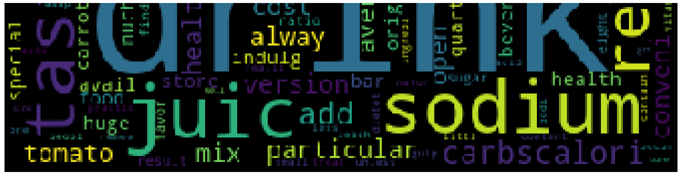
cup
thought
coffepod
wrong
foil
help
rather
pay
wast
came
make
keep
hint
holder
maker
fact
finish
holderit
put
practic
order
fresher
even
individu
flavor
delici
seal

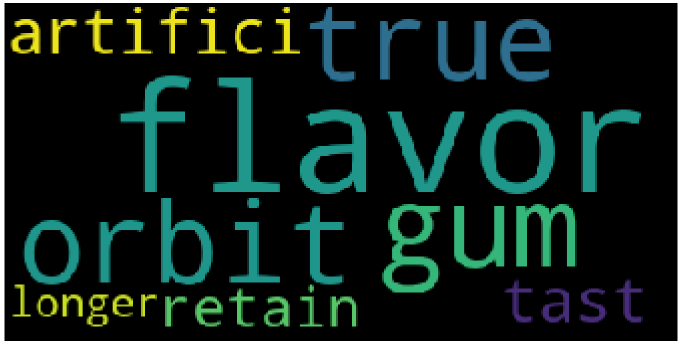
probabl
rank
best


recomend
cofe
one
near
mani
milk
perfect
opinion
product
drink
ucc
high
well
blend
coffe
recent


market
review
recommend
closer
eat
chees
way
less
price
child
care
gift
tast
high
great
sever
mac
kid
add
box
best
children
week
pay
quick
usual
product
even
perfect
agre


product
veget
low
per
serv
lycopen
amount
first
becom
type
ason





- Cluster Number 1 : This Cluster is about how did they feel :- tasty, like, delicious, flavour
- Cluster Number 2 : This Cluster is about baby food and health:- daughter, child, doctor
- Cluster Number 3 : This Cluster is about grains and cereals
- Cluster Number 4 : This Cluster is about tobaco, cigar
- Cluster Number 5 : This Cluster is about notthing can be said
- Cluster Number 6 : This Cluster is about Chocolate coffe milk
- Cluster Number 7 : This Cluster is about delivery
- Cluster Number 8 & Cluster Number 9 : This Cluster is about hot drinks like tea coffe milk
- Cluster Number 10 : This Cluster is about cheese
- Cluster Number 11 : This Cluster is about juice, energy and nutrients health
- Cluster Number 12 : This Cluster is about Chewing gum

CosineSimilar Words

In [68]:
```
cosineSimilarWords('jerki')
```

```
Similar Words for jerki :-
1         bear
2        gummi
3      popcorn
4      pretzel
5        salsa
6      licoric
7       sardin
8       wasabi
9         beef
10     ketchup
11       pickl
12        okay
13         gum
14       ramen
15       frank
16    marmalad
17       seawe
18      muesli
19        noth
20        tuna
Name: x, dtype: object
```

In [69]:
```
cosineSimilarWords('milk')
```

```
Similar Words for milk :-
1         skim
2         goat
3      condens
4          cow
5          soy
6       breast
7       powder
8       splash
9        muscl
10        whey
11        malt
12     soymilk
```

```
13        hemp
14        dunk
15        silk
16      lactos
17       cream
18      suppli
19      almond
20    lecithin
Name: x, dtype: object
```

In [70]:

```python
cosineSimilarWords('love')
```

```
Similar Words for love :-
1               fell
2           grandson
3      grandchildren
4            retriev
5            absolut
6              yorki
7          chihuahua
8           daughter
9               dear
10               kid
11               son
12           husband
13           everyon
14             poodl
15           toddler
16           terrier
17               dad
18        girlfriend
19         boyfriend
20               law
Name: x, dtype: object
```

Summary

- We can see Truncated SVD is very Effective but we have to pass in matrix like one we made called Co-occurance matrix
- Truncating Tfidf vectors requires a hyperparameter n_components for which tuning was done by best explained Variance ratio, and in this case n_components was 120
- Results from Kmeans was good and well seperated clusters were observed
- Cosine similarity of words were relatable like 'jerki' belong to nonveg family and what we got in top 10 similar words were beef, buffalo, turkey etc