Truncated SVD

In [1]:
```
#to ignore warnings
import warnings
warnings.filterwarnings("ignore")
#to use sqlite3 database
import sqlite3
import numpy as np
import pandas as pd
import string
import nltk
import matplotlib.pyplot as plt
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\cross_validation.py:
41: DeprecationWarning: This module was deprecated in version 0.18 in f
avor of the model_selection module into which all the refactored classe
s and functions are moved. Also note that the interface of the new CV i
terators are different from that of this module. This module will be re
moved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
con = sqlite3.connect('database.sqlite')
```

```python
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'
```

In [3]:

```python
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
```

In [4]:

```python
#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 filtered_data['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 (filtered_data['Score'].values)[i] == 'positive'
:
                        all_positive_words.append(s) #list of all words
 used to describe positive reviews
                    if(filtered_data['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
```

In [5]:
```
filtered_data['CleanedText']=final_string #adding a column of CleanedTe
xt which displays the data after pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("u
tf-8")
```

In [6]:
```
sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(100000)
```

In [7]:
```
sampledata['CleanedText'].head()
```

Out[7]:
```
138706    witti littl book make son laugh loud recit car...
138683    rememb see show air televis year ago child sis...
417839    beetlejuic well written movi everyth excel act...
```

```
417859    twist rumplestiskin captur film star michael k...
212472    twist rumplestiskin captur film star michael k...
Name: CleanedText, dtype: object
```

In [8]:
```
comment_words = []
for val in sampledata['CleanedText'].values:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()
    for i in range(len(tokens)):
        comment_words.append(tokens[i].lower())
```

TF-IDF

In [10]:
```
tf_idf_vect = TfidfVectorizer(1,min_df=20)
final_tf_idf1 = tf_idf_vect.fit(sampledata['CleanedText'].values)
final_tf_idf =final_tf_idf1.transform(sampledata['CleanedText'].values)
```

In [11]:
```
dictionary = sorted(zip(tf_idf_vect.idf_,tf_idf_vect.get_feature_names
()),reverse=True)
```

In [12]:
```
f=[]
count=0
for a,b in dictionary:
    if count<=2000:
        f.append(b)
        count+=1
    else:break


r= np.asarray(f)
length = len(f)
print(len(f))
```

```
2001
```

Function which takes the top 2000 features and find the co-occurance matrix based on reviews

In [13]:

```
window = 5
m = np.zeros([length,length]) # n is the count of all words
def cal_occ(f,m,comment_words):
    for i,word in enumerate(f):
        for j, e in enumerate(comment_words):
            if e == word:
                for k in range(max(j-window,0),min(j+window,len(comment
_words))):
                    if comment_words[k] in f:
                        l=f.index(comment_words[k])
                        if i == l:
                            continue
                        else:
                            m[i,l]+=1
```

In [14]:

```
cal_occ(f, m,comment_words)
```

Count of all the non zeros in co-occurance matrix and applying function on small data so that we can clarify if our matrix is correct

In [16]:

```
nonzeros = []
c=0
for i in range(length):
    for j in range(length):
        if m[i,j]!=0:
```

```
            c+=1
c
```

Out[16]: 9880

In [17]:
```
a = 'Im reading Sapiens right now a history of early mankind published
 last year by historian Yuval Noah Harari I havent gotten very far into
 it so I dont know if his idiosyncratic theories will end up being pers
uasive Still its the kind of learned but big think book I tend to like
 regardless of how well it holds up I wish more deeply accomplished peo
ple were willing to write stuff like this'
t = ['right','history','early']
comment_words1 = []

    # typecaste each val to string
a = str(a)

    # split the value
tokens = a.split()
for i in range(len(tokens)):
    comment_words1.append(tokens[i].lower())
r = np.zeros([len(t),len(t)])
cal_occ(t,r,comment_words1)
r
```

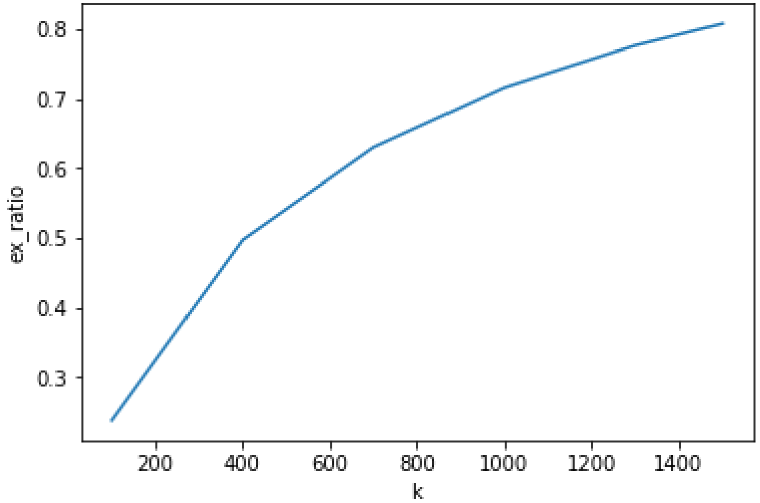
Out[17]:
```
array([[0., 1., 0.],
       [1., 0., 1.],
       [1., 1., 0.]])
```

Finding best components in Truncated SVD

In [25]:
```
from sklearn.decomposition import TruncatedSVD

k=[100,400,700,1000,1300,1500]
ex_ratio =[]
for i in k:
    svd = TruncatedSVD(n_components=i, n_iter=7, random_state=999)
```

```
    final_tf_idf2=svd.fit(final_tf_idf)
    ex_ratio.append(svd.explained_variance_ratio_.sum())
```

In [26]:
```
plt.plot(k,ex_ratio)
#plt.legend()
plt.xlabel('k')
plt.ylabel('ex_ratio')
plt.show()
```



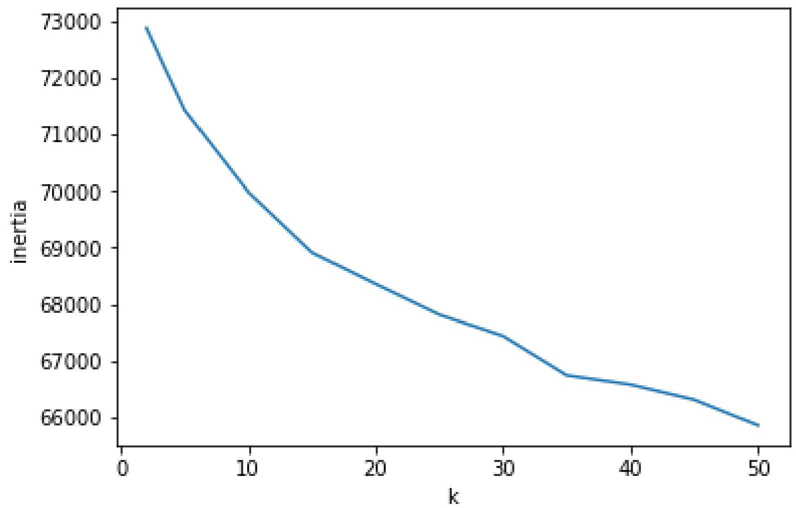
In [16]:
```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=1200, n_iter=7, random_state=999)
final_tf_idf2=svd.fit_transform(final_tf_idf)
```

Finding Best K in K-Means

In [40]:
```
from sklearn.cluster import KMeans
K = [2,5,10,15,20,25,30,35,40,45,50]
inertia = []
for k in K:
```

```
    kmeans = KMeans(n_clusters=k, random_state=0).fit(final_tf_idf2)
    inertia.append(kmeans.inertia_)
```

In [41]:
```
plt.plot(K,inertia)
#plt.legend()
plt.xlabel('k')
plt.ylabel('inertia')
plt.show()
```



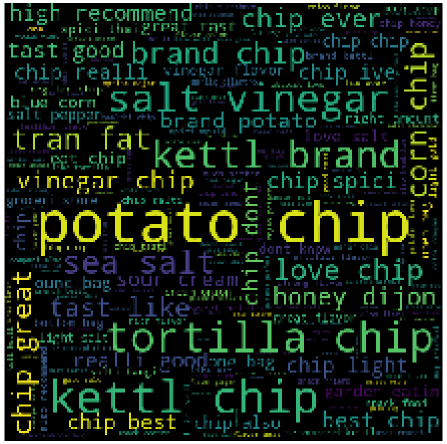
In [42]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=15, random_state=0).fit(final_tf_idf2)
```

In [43]:
```
def againcleaning(X):
    comment_words=' '
    for words in X:
        comment_words = comment_words + words + ' '
    return comment_words
count=0
review=sampledata['CleanedText'].values
topn_class1 = sorted(zip(kmeans.labels_, review))
feature =[]
for coef,feat in topn_class1:
```

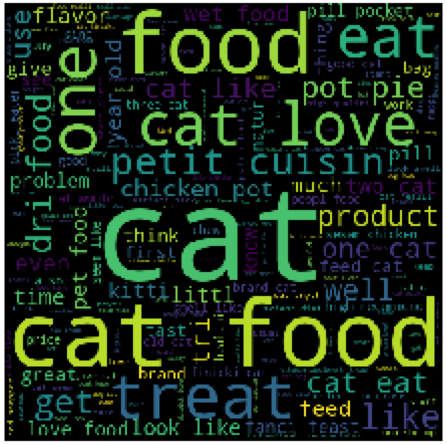
```
        if coef == count:
            feature.append(feat)
        else:
            a=againcleaning(feature)
            print(" cluster =", count)
            from wordcloud import WordCloud, STOPWORDS
            import matplotlib.pyplot as plt
            word_cloud=WordCloud(background_color='black',stopwords=sto
p,width=500,height=500).generate(a)
            plt.imshow(word_cloud)
            plt.axis("off")
            plt.show()
            count = count + 1
            feature =[]
            feature.append(feat)
#for label = 19
a=againcleaning(feature)
print(" cluster =", count)
#print(a, " " , count)
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
word_cloud=WordCloud(background_color='black',stopwords=stop,width=500,
height=500).generate(a)
plt.imshow(word_cloud)
plt.axis("off")
plt.show()
```

```
cluster = 0
```

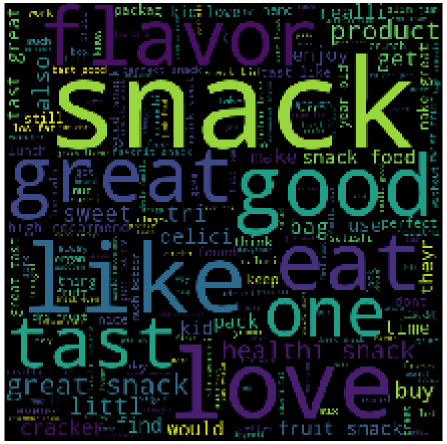
cluster = 1



cluster = 2

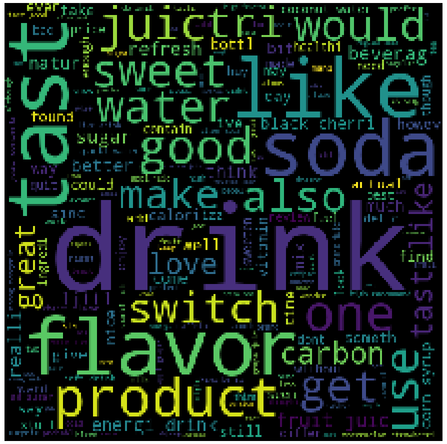


cluster = 3

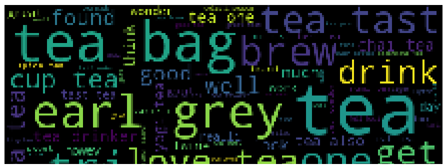


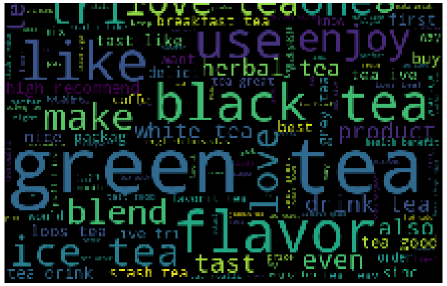
cluster = 4

cluster = 5



cluster = 6

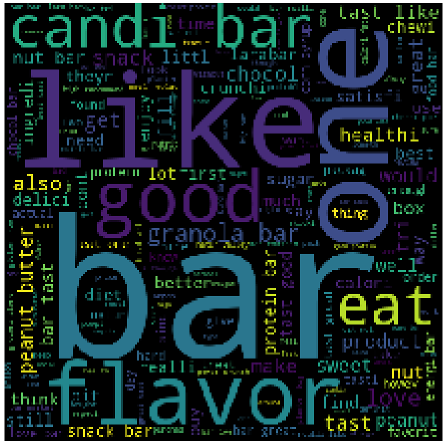




cluster = 7

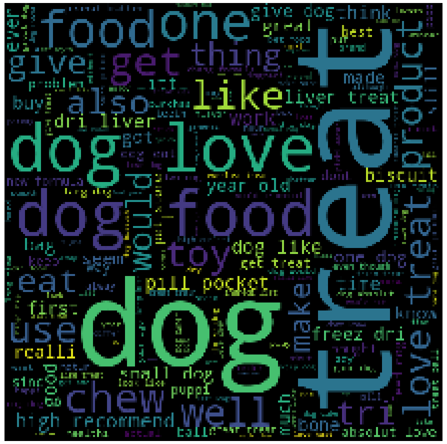
cluster = 8

cluster = 9



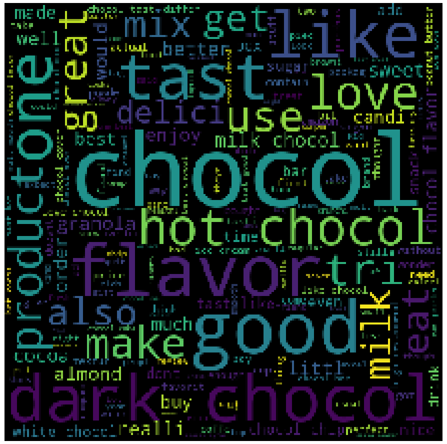
cluster = 10

cluster = 11

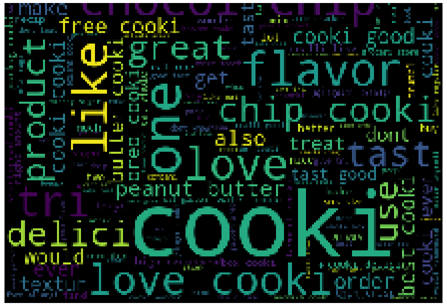


cluster = 12


```
cluster = 13
```




```
cluster = 14
```



This is a function which takes input as a word and find similar words using cosine similarity and using the reduces matrix in truncated svd only ..

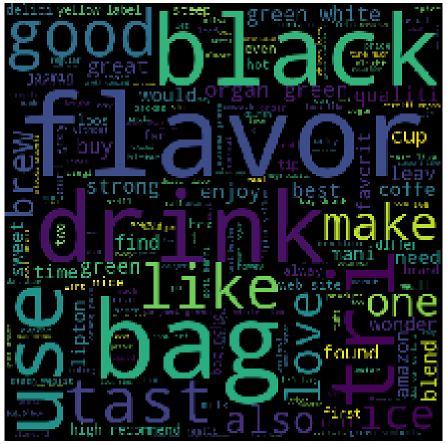
In [102]:

```
def cossim(word):

    wor = [word]
    wor1 = final_tf_idf1.transform(wor)
    wor2=svd.transform(wor1)
    from sklearn.metrics.pairwise import cosine_similarity
    cosine_similarities = cosine_similarity(wor2, final_tf_idf2).flatte
n()
    cosine_similarities
    zipp = sorted(zip(cosine_similarities,sampledata['CleanedText'].val
ues),reverse = True)[0:300]
    impwor = []
    for a,b in zipp:
        b=b.replace(word ,"")
        impwor.append(b)
    mw = againcleaning(impwor)


    #print(a, " " , count)
    from wordcloud import WordCloud, STOPWORDS
```

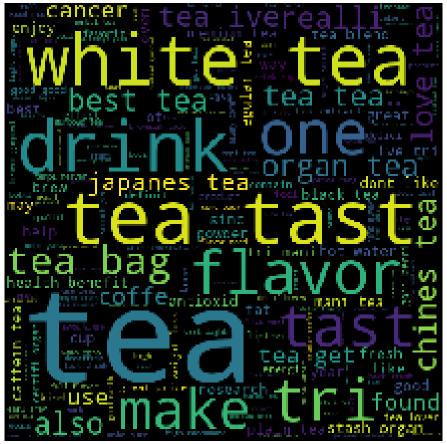
```
    import matplotlib.pyplot as plt
    word_cloud=WordCloud(background_color='black',stopwords=stop,width=
500,height=500).generate(mw)
    plt.imshow(word_cloud)
    plt.axis("off")
    plt.show()
```

In [103]:
```
cossim('like')
```

In [112]:
```
cossim('tea')
```

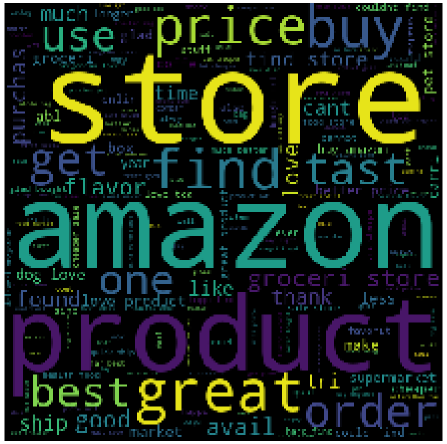


In [113]:
```
cossim('green')
```



In [115]:
```
cossim('coffee')
```

In [116]:
```
cossim('local')
```



Observation

Yes, Truncated SVD will reduce the features... and here there is no more change in clustering even after truncated svd because it will take features based on explained variance ratio.. The last one to calculate cosine similarity... my function takes a word .. applies tfidf and transform it andusing truncated svd it eill reduce it to same 1200 dimentions and find cossine similarity and sort them and print imp words in top cossine similarity score and print other similar words other than the word which we gave