Truncated SVD- Amazon fine food reviews

This Kernel explores the TruncatedSVD and how it can be used on Amazon Food reviews to gather meaningful Clusters

[49]:

```
import os
print(os.listdir("../input"))
```

```
['database.sqlite', 'hashes.txt', 'Reviews.csv']
```

[ ]:

```

```

[ ]:

```
# Import all the required libraries

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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 CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV

from collections import Counter



from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.metrics import average_precision_score

# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import re
import string
import nltk.corpus
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.porter import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
```

[ ]:

```
# using the SQLite Table to read data.
import sqlite3
show_tables = "select tbl_name from sqlite_master where type = 'table'"
conn = sqlite3.connect('../input/database.sqlite')
pd.read_sql(show_tables,conn)
```

[ ]:

```
# Data cleaning steps
#filtering only positive and negative reviews i.e. SKIPPING reviews with Score=

filtered_data = pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3"""

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a neg
```

```
def partition(x):
    if x < 3:
        return '0'
    return '1'

#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 attributes
filtered_data.head()
```

[ ]:

```
#Data Cleaning: Deduplication:
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", conn)
display.head()

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

#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"
final.shape

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

# Cleadning the data for Helpfulness duplication

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

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

#Before starting the next phase of preprocessing lets see the number of entrie
print(final.shape)

#How many positive and negative reviews are present in our dataset?
```

[ ]:

```
# Get 10k  Pts for this analysis

from random import sample
final_dataset = final.ix[np.random.choice(final.index, 10000)]
```

[ ]:

```
#Sorting data according to Time in ascending order
KMEANS_DATASET=final_dataset.sort_values('Time', axis=0, ascending=True, inpla
```

[ ]:

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

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

# STEMMING
import nltk.corpus
from nltk.corpus import stopwords
```

```
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 spe
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
#print(stop)
#print('************************************')
#print(sno.stem('tasty'))
```

[ ]:

```
# LEMMATIZATION
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 KMEANS_DATASET['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 (KMEANS_DATASET['Score'].values)[i] == '1':
                        all_positive_words.append(s) #list of all words used t
                    if(KMEANS_DATASET['Score'].values)[i] == '0':
                        all_negative_words.append(s) #list of all words used t
                else:
                    continue
            else:
                continue
```

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

    final_string.append(str1)
```

[ ]:

```
KMEANS_DATASET['CleanedText']=final_string #adding a column of CleanedText whi
KMEANS_DATASET['CleanedText']=KMEANS_DATASET['CleanedText'].str.decode("utf-8"
```

[ ]:

```
# define column names
names = ['CleanedText']


# create design matrix X and target vector y
X =  KMEANS_DATASET[names]
Y = KMEANS_DATASET['Score']

X_train, X_test = model_selection.train_test_split(X,test_size=0.1, random_sta
```

Getting the Top 2000 TFIDF features

[ ]:

```
tf_idf_vect = TfidfVectorizer()

tfidf = tf_idf_vect.fit(X['CleanedText'].values)

tfidf_train = tfidf.transform(X['CleanedText'].values)

#tfidf_test  = tfidf.transform(X_test['CleanedText'].values)

features = tf_idf_vect.get_feature_names()
```

[ ]:

```
features_set= (np.asarray(features)[0:5,])
print(features_set)
```

MODULE 1: ANALYSIS OF TOP 2000 TFIDF FEATURES

Part 1: get the Top 2000 TFIDF Features

[ ]:
```
# source: https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=10000):
    ''' Get top n tfidf values in row and return them with their corresponding
    topn_ids = np.argsort(row)[::-1][:top_n]
    #top_feats = [(features[i], row[i]) for i in topn_ids]
    top_feats = [(features[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    #df.columns = ['feature', 'tfidf']
    df.columns = ['feature']
    return top_feats

top_tfidf = top_tfidf_feats(tfidf_train[1,:].toarray()[0],features,2000)
```

Part 2 : Created Word Co-Occurence Matrix with neighbourhood = 5

[ ]:
```
import numpy as np
import pandas as pd

ctxs = list(top_tfidf)

l_unique = list(set((' '.join(ctxs)).split(' ')))
mat = np.zeros((len(l_unique), len(l_unique)))

nei = []
nei_size = 5

for ctx in ctxs:
    words = ctx.split(' ')

    for i, _ in enumerate(words):
```

```
            nei.append(words[i])

            if len(nei) > (nei_size * 2) + 1:
                nei.pop(0)

            pos = int(len(nei) / 2)
            for j, _ in enumerate(nei):
                if nei[j]  in l_unique and words[i] in l_unique:
                    mat[l_unique.index(nei[j]), l_unique.index(words[i])] += 1

    mat = pd.DataFrame(mat)
    mat.index = l_unique
    mat.columns = l_unique
    print(mat)
```

Part 3: Word Co-Occurence Matrix Decomposition using SVD

[ ]:

```
import numpy as np

la= np.linalg

#u, s, vh = la.svd(mat, full_matrices=True)

U, d, Vt = la.svd( mat, full_matrices=False )

assert np.all( d[:-1] >= d[1:] )  # sorted

eigen = d**2/2000

sumvariance = np.cumsum(eigen)

sumvariance /= sumvariance[-1]

i= np.arange(1,2001)
```

Part 4: Get best value of 'k', based on explained variance

of matrix U

```
import matplotlib.pyplot as plt
plt.plot( i,sumvariance)
plt.rcParams["figure.figsize"]= [7,7]
```

Part 5: TruncatedSVD on U to reduce U to 'k' components

```
import numpy as np
from sklearn.decomposition import TruncatedSVD

component_matrix =[]
variance_matrix = []



model = TruncatedSVD(n_components=250).fit(U)
X_proj = model.transform(U)

explained_variances = round(np.mean(np.var(X_proj, axis=0) / np.var(U, axis=0)
```

```
from sklearn.decomposition import TruncatedSVD

from scipy.sparse import csr_matrix

standardized_data_sparse_train = csr_matrix(U)

tsvd = TruncatedSVD(n_components=250)

standardized_data_sparse_tsvd_train = tsvd.fit(standardized_data_sparse_train)
```

Part 6: # Aggregate the Features and visualize the Clusters (K Means)

[ ]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.spatial import distance
from scipy.spatial.distance import cdist

clusters=range(1,40)
meandist=[]

meandist = []
Inertia_matrix = []
Assignment_matrix = []

for k in clusters:
    kmeanModel = KMeans(init='k-means++',n_clusters=k).fit(standardized_data_sp
    kmeanModel.fit(standardized_data_sparse_tsvd_train)
    Assignment_matrix.append(kmeanModel.predict(standardized_data_sparse_tsvd_
    #meandist.append(sum(np.min(cdist(bow_train_vector, kmeanModel.cluster_cen
    Inertia_matrix.append( kmeanModel.inertia_)
```

Finding Optimal K from the elbow plot

[ ]:

```
# Finding Optimal K from the elbow plot

fig = plt.figure()
ax = fig.add_subplot(111)



# Set the figure width and heights
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 12
fig_size[1] = 12
plt.rcParams["figure.figsize"] = fig_size

plt.plot(clusters, Inertia_matrix)
```

```
ax.plot(clusters,Inertia_matrix, marker='o', markersize=4,markeredgewidth=5, ma

plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```

[ ]:

```
# Using Optimal K get the Centroids,Labels and Inertia details
```

[ ]:

```
from sklearn import cluster
centroids,labels,inertia = cluster.k_means(standardized_data_sparse_tsvd_train
```

Part 7: Take a word and gather 100 words closer to it based on Cosine Similarity

- Here the word taken is :

[77]:

```
top_tfidf[8]
```

```
'cafe'
```

[78]:

```
from scipy import linalg, mat, dot

a = standardized_data_sparse_tsvd_train[8]

cosine_similarity_values=[]
cosine_similarity_index = []

for i in range(0,2000):
    c = abs(dot(a,standardized_data_sparse_tsvd_train[i].T)/linalg.norm(a)/lin
    cosine_similarity_index.append(i)
```

```
    cosine_similarity_values.append(c)
```

[79]:

```
c = np.column_stack((cosine_similarity_index,cosine_similarity_values))
```

[81]:

```
datanew = c[c[:,1].argsort()[::-1]]
#print(datanew)
```

[82]:

```
top100_words = (np.asarray(datanew)[0:100,])
#print(top100_words)
```

[83]:

```
top100_index= [int(row[0]) for row in top100_words]
```

[ ]:

```
#np.asarray(top100_index).shape
```

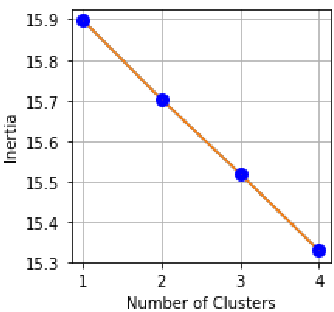
[57]:

```
top_closest_words= np.asarray(top_tfidf)[top100_index]
```

[84]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.spatial import distance
from scipy.spatial.distance import cdist

clusters=range(1,5)
meandist=[]

meandist = []
Inertia_matrix = []
Assignment_matrix = []

for k in clusters:
    kmeanModel = KMeans(init='k-means++',n_clusters=k).fit(standardized_data_s
    kmeanModel.fit(standardized_data_sparse_tsvd_train[top100_index])
    Assignment_matrix.append(kmeanModel.predict(standardized_data_sparse_tsvd_
    Inertia_matrix.append( kmeanModel.inertia_)
```

```python
# Finding Optimal K from the elbow plot

fig = plt.figure()
ax = fig.add_subplot(111)



# Set the figure width and heights
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 1
fig_size[1] = 1
plt.rcParams["figure.figsize"] = fig_size

plt.plot(clusters, Inertia_matrix)
ax.plot(clusters,Inertia_matrix, marker='o', markersize=4,markeredgewidth=5, ma

plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```



[87]:

```python
from sklearn import cluster
centroids,labels,inertia = cluster.k_means(standardized_data_sparse_tsvd_train

# All the Cluster details:
{i: np.where(kmeanModel.labels_ == i)[0] for i in range(kmeanModel.n_clusters)
```

```
{0: array([12, 69, 81, 86, 88]),
```

```
 1: array([ 3,  4,  5,  7, 16, 20, 21, 22, 23, 24, 32, 35, 36, 39, 42, 45, 5
2,
        60, 61, 62, 67, 68, 70, 72, 75, 77, 91, 92, 98]),
 2: array([ 0,  1,  6, 10, 11, 13, 15, 17, 18, 19, 25, 26, 27, 28, 29, 30, 3
1,
        34, 38, 40, 43, 46, 47, 49, 51, 53, 54, 55, 56, 57, 59, 63, 65, 66,
        71, 73, 74, 76, 78, 80, 82, 83, 85, 87, 90, 93, 94, 95, 96, 97, 99]),
 3: array([ 2,  8,  9, 14, 33, 37, 41, 44, 48, 50, 58, 64, 79, 84, 89])}
```

[88]:

```
# No of words in Cluster 0
sum(kmeanModel.labels_ == 0)
```

```
5
```

[89]:

```
# No of words in Cluster 1
sum(kmeanModel.labels_ == 1)
```

```
29
```

[90]:

```
# No of words in Cluster 2
sum(kmeanModel.labels_ == 2)
```

```
51
```

[91]:

```
# No of words in Cluster 3
sum(kmeanModel.labels_ == 3)
```

```
15
```

Part 8: Display the Clusters

[92]:

```
#from sklearn import cluster
```

```
#centroids,labels,inertia = cluster.k_means(standardized_data_sparse_tsvd_trai

# All the Cluster details:
#{i: np.where(kmeanModel.labels_ == i)[0] for i in range(kmeanModel.n_clusters

# Getting details of Cluster 5

index= np.where(kmeanModel.labels_ == 2)[0]
index_new = np.asarray(index)
cluster5 = str(np.asarray(X_train)[index_new,])

from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

print("The WordCloud for Cluster 1 is given Below:")

wordcloud = WordCloud().generate(str(cluster5))

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 20
fig_size[1] = 20
plt.rcParams["figure.figsize"] = fig_size

plt.imshow(wordcloud, interpolation='bilinear')
plt.show()
```

```
The WordCloud for Cluster 1 is given Below:
```

This cluster is themed around "Cafe"- we have words like coffe, tea, cup, breakfast, cereal,espresso,eat,chai agregated around this theme.

MODULE 2: ANALYSIS OF TOP 5000 TFIDF FEATURES

Part 1: get the Top 5k TFIDF Features

[93]:

```
# source: https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=5000):
    ''' Get top n tfidf values in row and return them with their corresponding
    topn_ids = np.argsort(row)[::-1][:top_n]
    #top_feats = [(features[i], row[i]) for i in topn_ids]
    top_feats = [(features[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    #df.columns = ['feature', 'tfidf']
    df.columns = ['feature']
    return top_feats
```

```
top_tfidf = top_tfidf_feats(tfidf_train[1,:].toarray()[0],features,5000)
```

Part 2 : Created Word Co-Occurence Matrix with neighbourhood = 5

[94]:

```
import numpy as np
import pandas as pd

ctxs = list(top_tfidf)

l_unique = list(set((' '.join(ctxs)).split(' ')))
mat = np.zeros((len(l_unique), len(l_unique)))

nei = []
nei_size = 5

for ctx in ctxs:
    words = ctx.split(' ')

    for i, _ in enumerate(words):
        nei.append(words[i])

        if len(nei) > (nei_size * 2) + 1:
            nei.pop(0)

        pos = int(len(nei) / 2)
        for j, _ in enumerate(nei):
          if nei[j]  in l_unique and words[i] in l_unique:
            mat[l_unique.index(nei[j]), l_unique.index(words[i])] += 1

mat = pd.DataFrame(mat)
mat.index = l_unique
mat.columns = l_unique
print(mat)
```

```
             drunk  hunger  evo  hondura  ...   boil  epa  bittersweet  smell
drunk          1.0     0.0  0.0      0.0  ...    0.0  0.0          0.0    0.0
hunger         0.0     1.0  0.0      0.0  ...    0.0  0.0          0.0    0.0
evo            0.0     0.0  1.0      0.0  ...    0.0  0.0          0.0    0.0
hondura        0.0     0.0  0.0      1.0  ...    0.0  0.0          0.0    0.0
box            0.0     0.0  0.0      0.0  ...    0.0  0.0          0.0    0.0
```

fret	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
ginseng	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
hcg	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
impuls	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
accomod	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
help	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
farley	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
hors	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
boscoli	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
hazan	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
heath	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
dougla	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
hectic	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
amazonprim	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
jake	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
domino	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
landmin	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
kenya	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
hlaf	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
boba	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0
genet	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
appletini	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
evidenc	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
harlan	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
gem	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
barney	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
guzzl	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
frere	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
konbu	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
havoc	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
johnson	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
crop	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
growl	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
heavier	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
elev	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
bald	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
equilibrium	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
famlili	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
afer	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
assess	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
bind	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

```
earthborn      0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
boquet         0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
crummi         0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
enterpris      0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
lamb           0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
henri          0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
joyva          0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
led            0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
headach        0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
krusteaz       0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    0.0
boil           0.0    0.0  0.0    0.0  ...    1.0  0.0       0.0    0.0
epa            0.0    0.0  0.0    0.0  ...    0.0  1.0       0.0    0.0
bittersweet    0.0    0.0  0.0    0.0  ...    0.0  0.0       1.0    0.0
smell          0.0    0.0  0.0    0.0  ...    0.0  0.0       0.0    1.0

[5000 rows x 5000 columns]
```

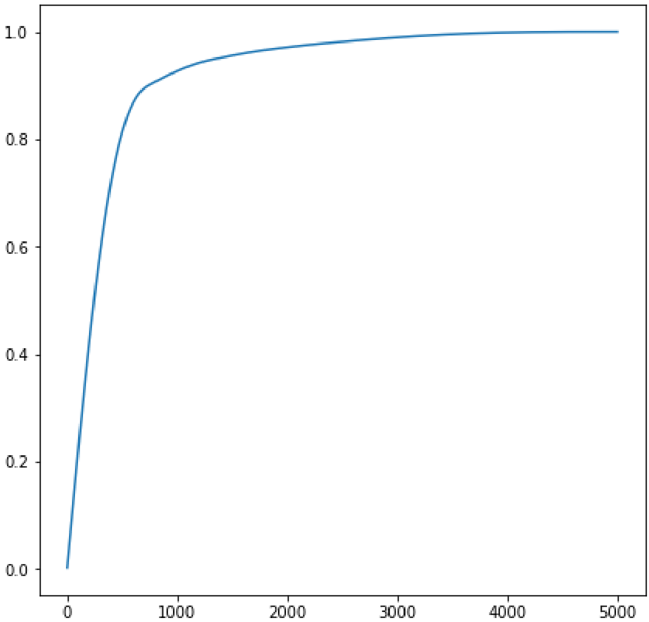
Part 3: Word Co-Occurence Matrix Decomposition using SVD

```
import numpy as np

la= np.linalg

#u, s, vh = la.svd(mat, full_matrices=True)

U, d, Vt = la.svd( mat, full_matrices=False )

assert np.all( d[:-1] >= d[1:] )  # sorted

eigen = d**2/5000

sumvariance = np.cumsum(eigen)

sumvariance /= sumvariance[-1]

i= np.arange(1,5001)
```

Part 4: Get best value of 'k', based on explained variance of matrix U

[97]:

```
import matplotlib.pyplot as plt
plt.plot( i,sumvariance)
plt.rcParams["figure.figsize"]= [2,2]
```



Part 5: TruncatedSVD on U to reduce U to 'k' components (from the elbow curve)

[98]:

```
import numpy as np
from sklearn.decomposition import TruncatedSVD

component_matrix =[]
variance_matrix = []



model = TruncatedSVD(n_components=500).fit(U)
X_proj = model.transform(U)

explained_variances = round(np.mean(np.var(X_proj, axis=0) / np.var(U, axis=0)
```

[103]:

```
from sklearn.decomposition import TruncatedSVD

from scipy.sparse import csr_matrix

standardized_data_sparse_train = csr_matrix(U)
```

https:/

```
tsvd = TruncatedSVD(n_components=500)

standardized_data_sparse_tsvd_train = tsvd.fit(standardized_data_sparse_train)
```

Part 6: # Aggregate the Features and visualize the Clusters (K Means)

[100]:

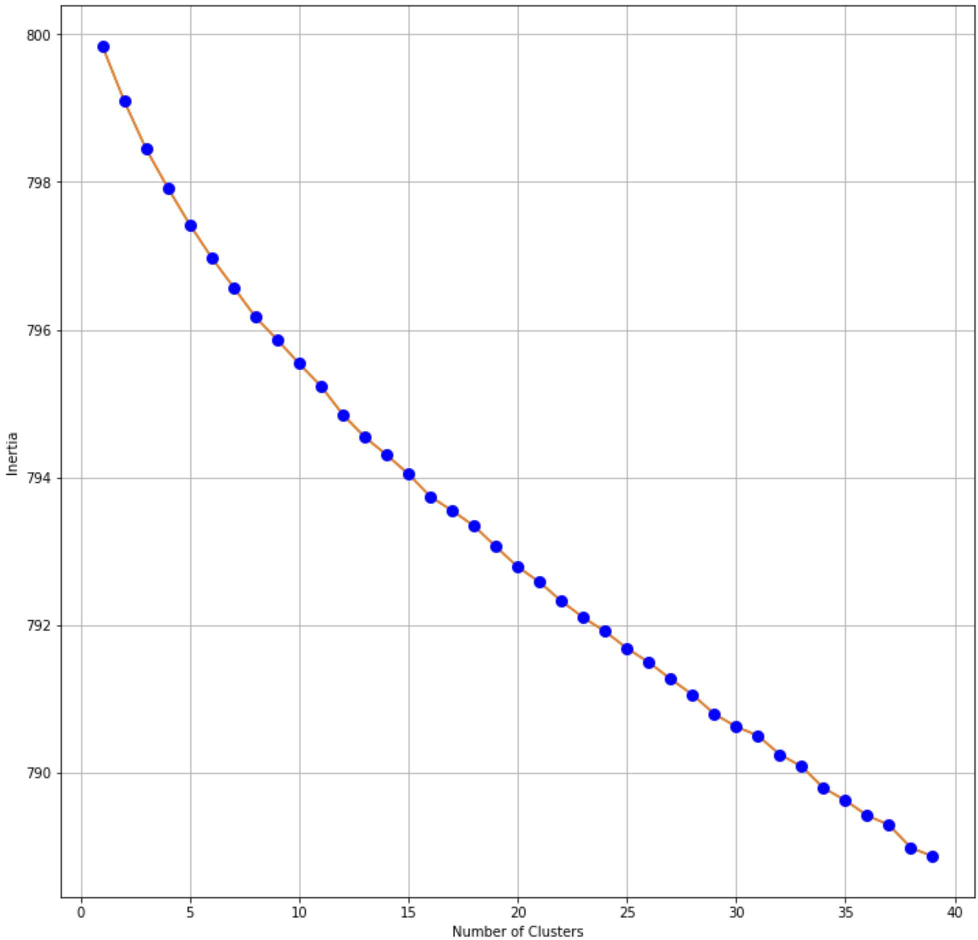
```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.spatial import distance
from scipy.spatial.distance import cdist

clusters=range(1,40)
meandist=[]

meandist = []
Inertia_matrix = []
Assignment_matrix = []

for k in clusters:
    kmeanModel = KMeans(init='k-means++',n_clusters=k).fit(standardized_data_s
    kmeanModel.fit(standardized_data_sparse_tsvd_train)
    Assignment_matrix.append(kmeanModel.predict(standardized_data_sparse_tsvd_
    #meandist.append(sum(np.min(cdist(bow_train_vector, kmeanModel.cluster_cen
    Inertia_matrix.append( kmeanModel.inertia_)
```

Finding Optimal K from the elbow plot

[105]:

```
# Finding Optimal K from the elbow plot

fig = plt.figure()
ax = fig.add_subplot(111)



# Set the figure width and heights
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 20
```

```
fig_size[1] = 20
plt.rcParams["figure.figsize"] = fig_size

plt.plot(clusters, Inertia_matrix)
ax.plot(clusters,Inertia_matrix, marker='o', markersize=4,markeredgewidth=5, ma

plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```



[106]:
```
# Using Optimal K get the Centroids,Labels and Inertia details
from sklearn import cluster
centroids,labels,inertia = cluster.k_means(standardized_data_sparse_tsvd_train
```

Part 7: Take a word and gather 100 words closer to it based on Cosine Similarity

Here the word taken is

[157]:
```
top_tfidf[1000]
```

```
'eighteen'
```

[158]:
```
from scipy import linalg, mat, dot

a = standardized_data_sparse_tsvd_train[1000]

cosine_similarity_values=[]
cosine_similarity_index = []

for i in range(0,5000):
    c = abs(dot(a,standardized_data_sparse_tsvd_train[i].T)/linalg.norm(a)/lina
    cosine_similarity_index.append(i)
```

```
        cosine_similarity_values.append(c)
```

[159]:

```
c = np.column_stack((cosine_similarity_index,cosine_similarity_values))

datanew = c[c[:,1].argsort()[::-1]]

top100_words = (np.asarray(datanew)[0:100,])

top100_index= [int(row[0]) for row in top100_words]

top_closest_words= np.asarray(top_tfidf)[top100_index]
```

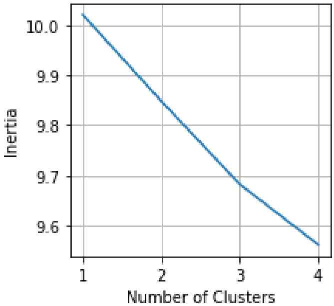
[160]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.spatial import distance
from scipy.spatial.distance import cdist

clusters=range(1,5)
meandist=[]

meandist = []
Inertia_matrix = []
Assignment_matrix = []

for k in clusters:
    kmeanModel = KMeans(init='k-means++',n_clusters=k).fit(standardized_data_s
    kmeanModel.fit(standardized_data_sparse_tsvd_train[top100_index])
    Assignment_matrix.append(kmeanModel.predict(standardized_data_sparse_tsvd_
    Inertia_matrix.append( kmeanModel.inertia_)
```

[162]:

```
# Finding Optimal K from the elbow plot

fig = plt.figure()
#ax = fig.add_subplot(111)


# Set the figure width and heights
```

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 3
fig_size[1] = 3
plt.rcParams["figure.figsize"] = fig_size

plt.plot(clusters, Inertia_matrix)
ax.plot(clusters,Inertia_matrix, marker='o', markersize=4,markeredgewidth=5, ma

plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
```



[163]:

```
from sklearn import cluster
centroids,labels,inertia = cluster.k_means(standardized_data_sparse_tsvd_train

# All the Cluster details:
{i: np.where(kmeanModel.labels_ == i)[0] for i in range(kmeanModel.n_clusters)
```

```
{0: array([ 3, 15, 48, 58, 74, 86, 90]),
 1: array([ 1,  6,  8, 14, 24, 28, 30, 31, 33, 46, 52, 53, 54, 55, 60, 62, 64,
        68, 69, 70, 71, 76, 77, 78, 80, 82, 89, 95, 96]),
 2: array([ 7, 16, 22, 25, 29, 34, 43, 50, 56, 75, 98]),
 3: array([ 0,  2,  4,  5,  9, 10, 11, 12, 13, 17, 18, 19, 20, 21, 23, 26, 27,
        32, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 47, 49, 51, 57, 59, 61,
        63, 65, 66, 67, 72, 73, 79, 81, 83, 84, 85, 87, 88, 91, 92, 93, 94,
        97, 99])}
```

[168]:

```
# No of words in Cluster 0
```

```
sum(kmeanModel.labels_ == 0)
```

```
7
```

[167]:

```
# No of words in Cluster 1
sum(kmeanModel.labels_ == 1)
```

```
29
```

[166]:

```
# No of words in Cluster 2
sum(kmeanModel.labels_ == 2)
```

```
11
```

[165]:

```
# No of words in Cluster 3
sum(kmeanModel.labels_ == 3)
```

```
53
```

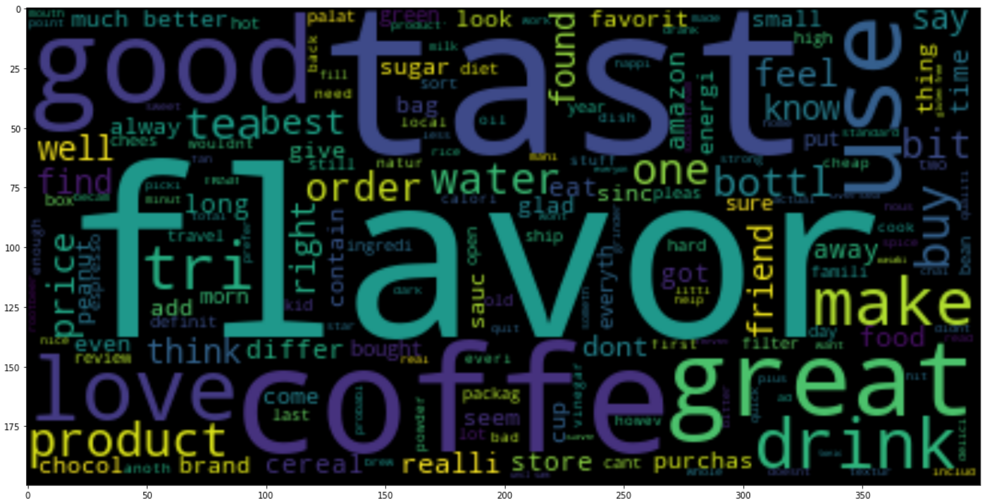
Part 8: Display the Clusters with max number of words

[169]:

```
index= np.where(kmeanModel.labels_ == 3)[0]
index_new = np.asarray(index)
cluster5 = str(np.asarray(X_train)[index_new,])

from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

print("The WordCloud for the Cluster is given Below:")
```

```
wordcloud = WordCloud().generate(str(cluster5))

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 20
fig_size[1] = 20
plt.rcParams["figure.figsize"] = fig_size

plt.imshow(wordcloud, interpolation='bilinear')
plt.show()
```

```
The WordCloud for the Cluster is given Below:
```



Cluster theme is "Diet" Indicated by words like cereal, energi,Green, diet etc.

Module 3: Analysis of top 10K TFIDF features

Part 1: get the Top 10k TFIDF Features

```
# source: https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=10000):
    ''' Get top n tfidf values in row and return them with their corresponding
    topn_ids = np.argsort(row)[::-1][:top_n]
    #top_feats = [(features[i], row[i]) for i in topn_ids]
    top_feats = [(features[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    #df.columns = ['feature', 'tfidf']
    df.columns = ['feature']
    return top_feats

top_tfidf = top_tfidf_feats(tfidf_train[1,:].toarray()[0],features,10000)
```

Part 2 : Created Word Co-Occurence Matrix with neighbourhood = 5

[189]:

```
import numpy as np
import pandas as pd

ctxs = list(top_tfidf)

l_unique = list(set((' '.join(ctxs)).split(' ')))
mat = np.zeros((len(l_unique), len(l_unique)))

nei = []
nei_size = 5

for ctx in ctxs:
    words = ctx.split(' ')

    for i, _ in enumerate(words):
        nei.append(words[i])

        if len(nei) > (nei_size * 2) + 1:
            nei.pop(0)

        pos = int(len(nei) / 2)
        for j, _ in enumerate(nei):
            if nei[j]  in l_unique and words[i] in l_unique:
              mat[l_unique.index(nei[j]), l_unique.index(words[i])] += 1
```

```
mat = pd.DataFrame(mat)
mat.index = l_unique
mat.columns = l_unique
print(mat)
```

```
            underneath  hondura  constitu  ...  lamb  headach  joyva
underneath         1.0      0.0       0.0  ...   0.0      0.0    0.0
hondura            0.0      1.0       0.0  ...   0.0      0.0    0.0
constitu           0.0      0.0       1.0  ...   0.0      0.0    0.0
copycat            0.0      0.0       0.0  ...   0.0      0.0    0.0
ginseng            0.0      0.0       0.0  ...   0.0      0.0    0.0
take               0.0      0.0       0.0  ...   0.0      0.0    0.0
impuls             0.0      0.0       0.0  ...   0.0      0.0    0.0
tastybit           0.0      0.0       0.0  ...   0.0      0.0    0.0
help               0.0      0.0       0.0  ...   0.0      0.0    0.0
hectic             0.0      0.0       0.0  ...   0.0      0.0    0.0
jake               0.0      0.0       0.0  ...   0.0      0.0    0.0
hlaf               0.0      0.0       0.0  ...   0.0      0.0    0.0
kenya              0.0      0.0       0.0  ...   0.0      0.0    0.0
theater            0.0      0.0       0.0  ...   0.0      0.0    0.0
snot               0.0      0.0       0.0  ...   0.0      0.0    0.0
splinter           0.0      0.0       0.0  ...   0.0      0.0    0.0
stubborn           0.0      0.0       0.0  ...   0.0      0.0    0.0
harlan             0.0      0.0       0.0  ...   0.0      0.0    0.0
scrape             0.0      0.0       0.0  ...   0.0      0.0    0.0
wast               0.0      0.0       0.0  ...   0.0      0.0    0.0
stage              0.0      0.0       0.0  ...   0.0      0.0    0.0
col                0.0      0.0       0.0  ...   0.0      0.0    0.0
falafel            0.0      0.0       0.0  ...   0.0      0.0    0.0
dramat             0.0      0.0       0.0  ...   0.0      0.0    0.0
art                0.0      0.0       0.0  ...   0.0      0.0    0.0
harbor             0.0      0.0       0.0  ...   0.0      0.0    0.0
spasm              0.0      0.0       0.0  ...   0.0      0.0    0.0
duke               0.0      0.0       0.0  ...   0.0      0.0    0.0
jazz               0.0      0.0       0.0  ...   0.0      0.0    0.0
thin               0.0      0.0       0.0  ...   0.0      0.0    0.0
...                ...      ...       ...  ...   ...      ...    ...
gourd              0.0      0.0       0.0  ...   0.0      0.0    0.0
swoon              0.0      0.0       0.0  ...   0.0      0.0    0.0
darwin             0.0      0.0       0.0  ...   0.0      0.0    0.0
dosent             0.0      0.0       0.0  ...   0.0      0.0    0.0
tealik             0.0      0.0       0.0  ...   0.0      0.0    0.0
```

```
funnel          0.0    0.0    0.0  ...    0.0    0.0    0.0
jewel           0.0    0.0    0.0  ...    0.0    0.0    0.0
nug             0.0    0.0    0.0  ...    0.0    0.0    0.0
airless         0.0    0.0    0.0  ...    0.0    0.0    0.0
trap            0.0    0.0    0.0  ...    0.0    0.0    0.0
doodl           0.0    0.0    0.0  ...    0.0    0.0    0.0
held            0.0    0.0    0.0  ...    0.0    0.0    0.0
gobstopp        0.0    0.0    0.0  ...    0.0    0.0    0.0
goop            0.0    0.0    0.0  ...    0.0    0.0    0.0
tidi            0.0    0.0    0.0  ...    0.0    0.0    0.0
nostril         0.0    0.0    0.0  ...    0.0    0.0    0.0
guzzl           0.0    0.0    0.0  ...    0.0    0.0    0.0
upright         0.0    0.0    0.0  ...    0.0    0.0    0.0
unison          0.0    0.0    0.0  ...    0.0    0.0    0.0
voxbox          0.0    0.0    0.0  ...    0.0    0.0    0.0
equilibrium     0.0    0.0    0.0  ...    0.0    0.0    0.0
shift           0.0    0.0    0.0  ...    0.0    0.0    0.0
unapp           0.0    0.0    0.0  ...    0.0    0.0    0.0
vanish          0.0    0.0    0.0  ...    0.0    0.0    0.0
chiavetta       0.0    0.0    0.0  ...    0.0    0.0    0.0
creek           0.0    0.0    0.0  ...    0.0    0.0    0.0
henri           0.0    0.0    0.0  ...    0.0    0.0    0.0
lamb            0.0    0.0    0.0  ...    1.0    0.0    0.0
headach         0.0    0.0    0.0  ...    0.0    1.0    0.0
joyva           0.0    0.0    0.0  ...    0.0    0.0    1.0

[10000 rows x 10000 columns]
```

Part 3: Word Co-Occurence Matrix Decomposition using SVD

[190]:

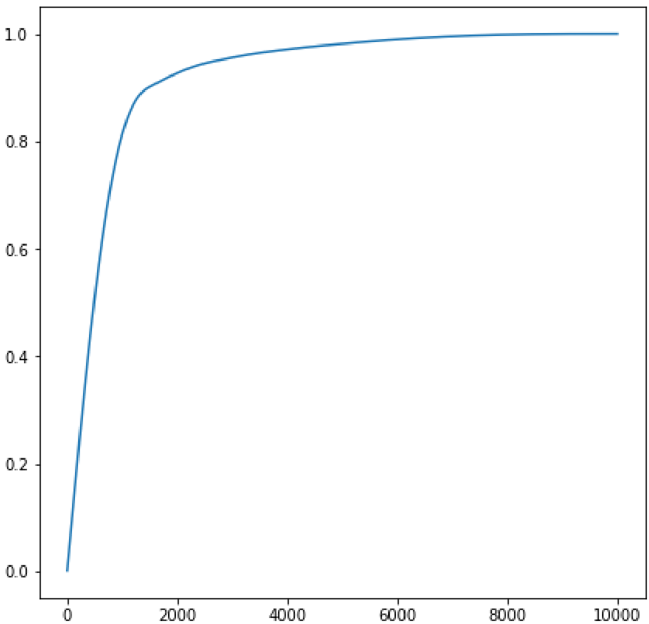
```
import numpy as np

la= np.linalg

#u, s, vh = la.svd(mat, full_matrices=True)

U, d, Vt = la.svd( mat, full_matrices=False )

assert np.all( d[:-1] >= d[1:] )  # sorted
```

```
eigen = d**2/10000

sumvariance = np.cumsum(eigen)

sumvariance /= sumvariance[-1]

i = np.arange(1,10001)
```

Part 4: Get best value of 'k', based on explained variance of matrix U

[192]:

```
import matplotlib.pyplot as plt
plt.plot( i,sumvariance)
plt.rcParams["figure.figsize"]= [2,2]
```



Part 5: TruncatedSVD on U to reduce U to 'k' components

[193]:

```
import numpy as np
```

```
from sklearn.decomposition import TruncatedSVD

component_matrix =[]
variance_matrix = []


model = TruncatedSVD(n_components=1200).fit(U)
X_proj = model.transform(U)

explained_variances = round(np.mean(np.var(X_proj, axis=0) / np.var(U, axis=0)
```

[194]:

```
from sklearn.decomposition import TruncatedSVD

from scipy.sparse import csr_matrix

standardized_data_sparse_train = csr_matrix(U)

tsvd = TruncatedSVD(n_components=1200)

standardized_data_sparse_tsvd_train = tsvd.fit(standardized_data_sparse_train)
```

Part 6: # Aggregate the Features and visualize the Clusters (K Means)

[195]:

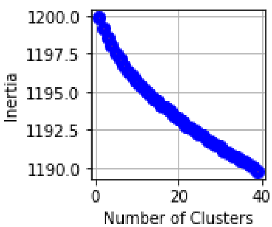
```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.spatial import distance
from scipy.spatial.distance import cdist

clusters=range(1,40)
meandist=[]

meandist = []
Inertia_matrix = []
Assignment_matrix = []

for k in clusters:
    kmeanModel = KMeans(init='k-means++',n_clusters=k).fit(standardized_data_s
    kmeanModel.fit(standardized_data_sparse_tsvd_train)
    Assignment_matrix.append(kmeanModel.predict(standardized_data_sparse_tsvd_
```

```
    #meandist.append(sum(np.min(cdist(bow_train_vector, kmeanModel.cluster_cen
```

Finding Optimal K from the elbow plot

[196]:

```
# Finding Optimal K from the elbow plot

fig = plt.figure()
ax = fig.add_subplot(111)


# Set the figure width and heights
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 12
fig_size[1] = 12
plt.rcParams["figure.figsize"] = fig_size

plt.plot(clusters, Inertia_matrix)
ax.plot(clusters,Inertia_matrix, marker='o', markersize=4,markeredgewidth=5, ma

plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```



[197]:

```
from sklearn import cluster
centroids,labels,inertia = cluster.k_means(standardized_data_sparse_tsvd_train
```

Part 7: Take a word and gather 100 words closer to it based on Cosine Similarity

Here the word taken is :

[219]:

```
top_tfidf[1234]
```

```
'enfamil'
```

[220]:

```
from scipy import linalg, mat, dot

a = standardized_data_sparse_tsvd_train[1234]

cosine_similarity_values=[]
cosine_similarity_index = []

for i in range(0,10000):
    c = abs(dot(a,standardized_data_sparse_tsvd_train[i].T)/linalg.norm(a)/lin
    cosine_similarity_index.append(i)
    cosine_similarity_values.append(c)
```

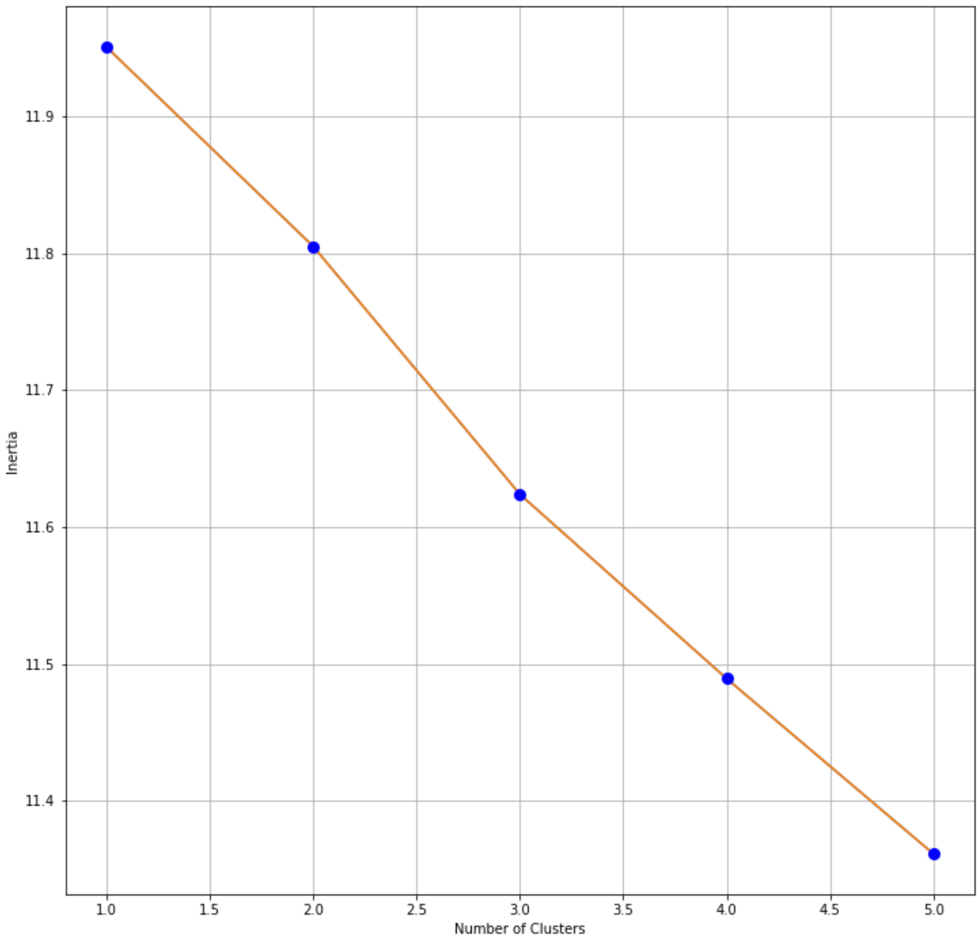
[221]:

```
c = np.column_stack((cosine_similarity_index,cosine_similarity_values))

datanew = c[c[:,1].argsort()[::-1]]


top100_words = (np.asarray(datanew)[0:100,])

top100_index= [int(row[0]) for row in top100_words]

top_closest_words= np.asarray(top_tfidf)[top100_index]
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
from scipy.spatial import distance
from scipy.spatial.distance import cdist

clusters=range(1,6)
meandist=[]

meandist = []
Inertia_matrix = []
Assignment_matrix = []

for k in clusters:
    kmeanModel = KMeans(init='k-means++',n_clusters=k).fit(standardized_data_sp
    kmeanModel.fit(standardized_data_sparse_tsvd_train[top100_index])
    Assignment_matrix.append(kmeanModel.predict(standardized_data_sparse_tsvd_t
    Inertia_matrix.append( kmeanModel.inertia_)
```

[223]:

```
# Finding Optimal K from the elbow plot

fig = plt.figure()
ax = fig.add_subplot(111)



# Set the figure width and heights
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 3
fig_size[1] = 3
plt.rcParams["figure.figsize"] = fig_size

plt.plot(clusters, Inertia_matrix)
ax.plot(clusters,Inertia_matrix, marker='o', markersize=4,markeredgewidth=5, ma

plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```


11.9
11.8
11.7
11.6
11.5
11.4
Inertia
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of Clusters

[224]:
```
from sklearn import cluster
centroids,labels,inertia = cluster.k_means(standardized_data_sparse_tsvd_train

# All the Cluster details:

{i: np.where(kmeanModel.labels_ == i)[0] for i in range(kmeanModel.n_clusters)
```

```
{0: array([ 3, 10, 21, 33, 40, 69, 74, 79, 92, 96]),
 1: array([ 4,  9, 11, 16, 20, 23, 24, 34, 36, 43, 44, 45, 46, 47, 50, 51, 55,
        56, 57, 58, 61, 65, 66, 77, 83, 84, 86, 95, 98, 99]),
 2: array([ 0,  5,  6,  7,  8, 12, 13, 14, 15, 18, 19, 26, 27, 28, 29, 31, 32,
        35, 39, 41, 49, 54, 59, 62, 67, 68, 70, 72, 73, 75, 78, 82, 87, 88,
        91, 93, 97]),
 3: array([17, 42]),
 4: array([ 1,  2, 22, 25, 30, 37, 38, 48, 52, 53, 60, 63, 64, 71, 76, 80, 81,
        85, 89, 90, 94])}
```

[225]:
```
# No of words in Cluster 1
sum(kmeanModel.labels_ == 0)
```

```
10
```

[226]:
```
# No of words in Cluster 2
sum(kmeanModel.labels_ == 1)
```

```
30
```

[227]:
```
# No of words in Cluster 3
sum(kmeanModel.labels_ == 2)
```

```
37
```

[228]:

```
# No of words in Cluster 4
sum(kmeanModel.labels_ == 3)
```

```
2
```

Part 8: Display the Cluster which has the maximum number of words

[230]:

```
index= np.where(kmeanModel.labels_ == 2)[0]
index_new = np.asarray(index)
cluster5 = str(np.asarray(X_train)[index_new,])

from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

print("The WordCloud for the cluster is given Below:")

wordcloud = WordCloud().generate(str(cluster5))

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 25
fig_size[1] = 25
plt.rcParams["figure.figsize"] = fig_size

plt.imshow(wordcloud, interpolation='bilinear')
plt.show()
```

```
The WordCloud for the cluster is given Below:
```

Theme of this cluster is Fitness: we have words like Breakfast, cereal,energi,Granola,Peanut Butter