Part 10 - Truncated SVD

By Aziz Presswala

In [1]:
```
# importing libraries

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

import sqlite3
import pandas as pd
import numpy as np
from numpy import dot
from numpy.linalg import norm
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from prettytable import PrettyTable
from wordcloud import WordCloud, STOPWORDS
from nltk.stem.snowball import SnowballStemmer

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
```

[1.0] Importing Data

In [2]:
```
# Using the CleanedText column saved in final.sqlite db
con = sqlite3.connect('final.sqlite')
filtered_data = pd.read_sql_query("SELECT * FROM Reviews", con)
filtered_data.shape
```

Out[2]: (364171, 12)

In [3]:
```
x = filtered_data['CleanedText']
```

In [4]:
```
x.shape
```

Out[4]: (364171,)

[2.0] Applying TFIDF

In [5]:
```
#applying transform on datasset
tf_idf_vect = TfidfVectorizer(min_df=10)
x_tfidf = tf_idf_vect.fit_transform(x.values)
x_tfidf.shape
```

Out[5]: (364171, 14767)

In [6]:
```
print("Total number of unique features(words) present in the review cor
pus: ",len(tf_idf_vect.get_feature_names()))
```

```
Total number of unique features(words) present in the review corpus:  1
4767
```

[3.0] Taking top features from TFIDF

In [7]:
```
feat_names = tf_idf_vect.get_feature_names()
feat_index = np.argsort(tf_idf_vect.idf_)
top_features = np.array([(feat_names[i], tf_idf_vect.idf_[i]) for i in
feat_index[:2000]])
df_topFeatures = pd.DataFrame(data=top_features, columns = ['Top Words'
,'TF-IDF Value'])
```

In [8]:
```
df_topFeatures.head(10)
```

Out[8]:

	Top Words	TF-IDF Value
0	like	2.146305907859448
1	tast	2.1662728867720227
2	good	2.308016535633283
3	love	2.35241843868791
4	great	2.4052553230108167
5	flavor	2.442407274278731
6	one	2.4500108981260507
7	product	2.473930013658044
8	use	2.50522861440522
9	tri	2.5156996538297705

[4.0] Calulation of Co-occurrence matrix

In [9]:

```
#Generate the Co-Occurence Matrix
def get_coOccuranceMatrix(X_train, top_features, window):
    print("Generating the Co Occurence Matrix....")
    # dimensions of the square matrix
    dim=top_features.shape[0]
    square_matrix = np.zeros((dim,dim),int)

    values = [i for i in range(0,top_features.shape[0])]  # Contains al
l the top TF-IDF Scores as values.
    keys = [str(i) for i in df_topFeatures['Top Words']]  # Contains al
l the corresponding features names as keys.
    lookup_dict = dict(zip(keys,values))                  # We will use
 this dictionary as a look up table

    top_words= keys

    #Processing each reviews to build the co-occurence Matrix
    for reviews in tqdm(X_train):
```

```
        #Split each review into words
        words = reviews.split()
        lnt = len(words)
        for i in range(lnt):
            idx_of_neigbors= []
            if((i-window >= 0) and (i+window < lnt)):
                idx_of_neigbors = np.arange(i-window,i+window+1)
            elif((i-window < 0) and (i+window < lnt)):
                idx_of_neigbors = np.arange(0, i+window+1)
            elif((i-window >= 0) and (i+window >= lnt)):
                idx_of_neigbors = np.arange(i-window, lnt)

            for j in idx_of_neigbors:
                if((words[j] in top_words) and (words[i] in top_words
)):
                    row_idx = lookup_dict[words[i]]     # Get the index
 of the ith word from the lookup table
                    col_idx = lookup_dict[words[j]]     # Get the index
 of the jth word from the lookup table
                    square_matrix[row_idx,col_idx] += 1 # increment the
 corresponding row & column by 1

    #Fill all the diagonal elements of the co-occurence matrix with 0

    np.fill_diagonal(square_matrix, 0)
    print("Co Occurence Matrix is generated....")

    #Create a co-occurence dataframe.
    co_occur_df=pd.DataFrame(data=square_matrix, index=keys, columns=ke
ys)
    return co_occur_df

co_occur_matrix = get_coOccuranceMatrix(x, top_features, window=2)
```

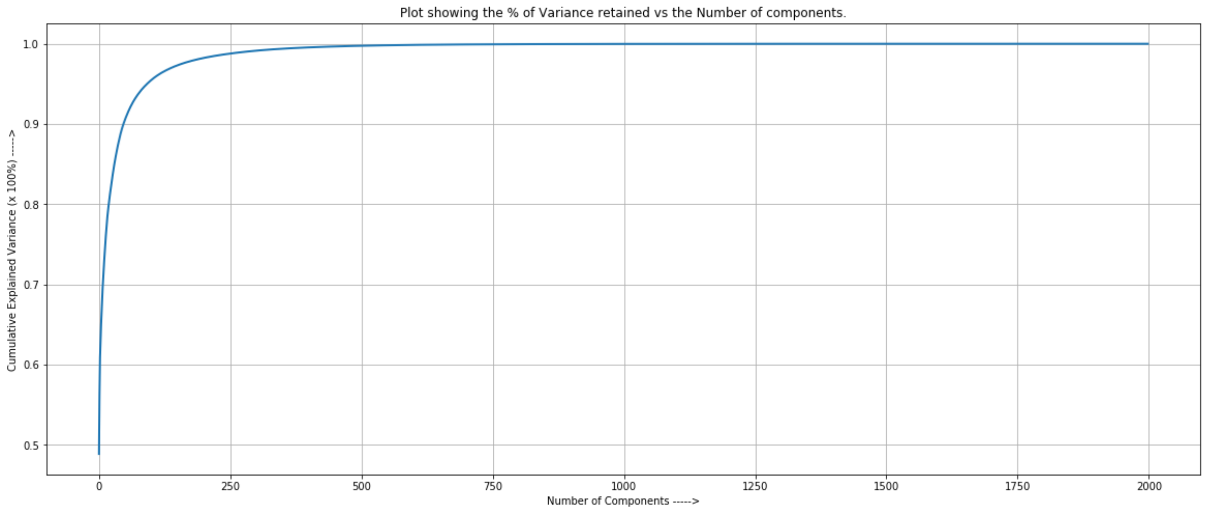
```
Generating the Co Occurence Matrix....
```

```
100%|██████████████████████████████████████████████████████████████████
███████| 364171/364171 [27:02<00:00, 224.49it/s]
```

```
Co Occurence Matrix is generated....
```

[5.0] Finding optimal value for number of components (n) to be retained.

In [10]:
```
n = co_occur_matrix.shape[0]-1

#Inititalize the truncated SVD object.
svd = TruncatedSVD(n_components=n,
                   algorithm='randomized',
                   n_iter=10,
                   random_state=0)
data=svd.fit_transform(co_occur_matrix)

cum_var_explained = np.cumsum(svd.explained_variance_ratio_)

# Plot the SVD spectrum
plt.figure(1, figsize=(20, 8))
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.title('Plot showing the % of Variance retained vs the Number of com
ponents.')
plt.xlabel('Number of Components ----->')
plt.ylabel('Cumulative Explained Variance (x 100%) ----->')
plt.show()
```



Observation : From the above plot the variance explained after 500 components is same, therefore 500 is taken as the optimal number of components

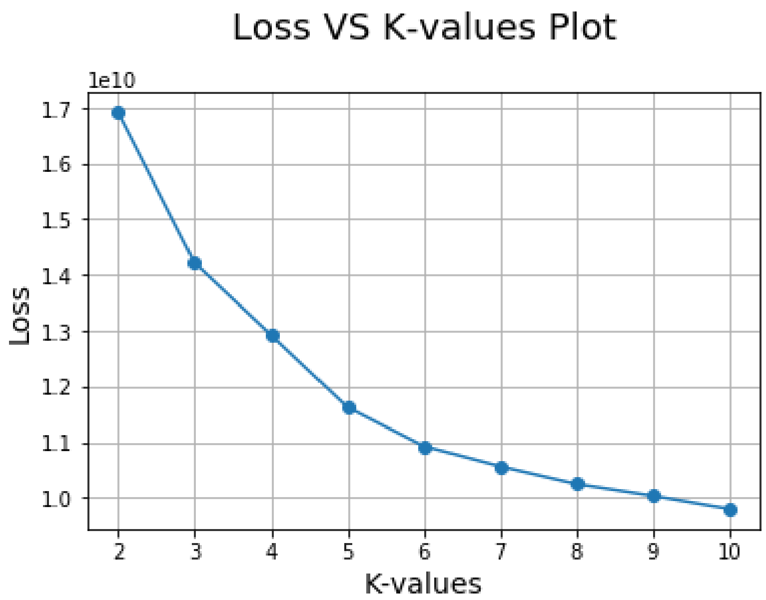
In [11]:
```
n_components = 500

svd = TruncatedSVD(n_components=n_components,
                   algorithm='randomized',
                   random_state=0,
                   n_iter=10)
U = svd.fit_transform(co_occur_matrix)
VT = svd.components_
Sigma = np.zeros((n_components,n_components),int)
np.fill_diagonal(Sigma, svd.singular_values_)

#Display the shapes of the matrix factors after SVD
print("U = {}, Sigma = {}, V.T = {}".format(U.shape,Sigma.shape,VT.shap
e))
```

```
U = (2000, 500), Sigma = (500, 500), V.T = (500, 2000)
```

[6.0] Applying k-means clustering

In [12]:
```
# k values we need to try on the model
k_values = list(range(2,11))
loss = []
for i in k_values:
    # initializing kmeans clustering model
    kmeans = KMeans(n_clusters=i, n_jobs=-1)
    kmeans.fit(U)
    loss.append(kmeans.inertia_)
```

In [13]:
```
# Plotting loss VS k_values
plt.plot(k_values, loss, '-o')
plt.xlabel('K-values',size=14)
plt.ylabel('Loss',size=14)
plt.title('Loss VS K-values Plot\n',size=18)
plt.grid()
plt.show()
```



Observation : In the above plot, loss reduces steeply until 6 but after that it reduces slowly hence optimal numbers of clusters = 6

In [14]:
```
# optimal k value
optimal_k = 6

# training the model using optimal k
kmeans=KMeans(n_clusters=optimal_k, n_jobs=-1)
kmeans.fit(U)
```

Out[14]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=6, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

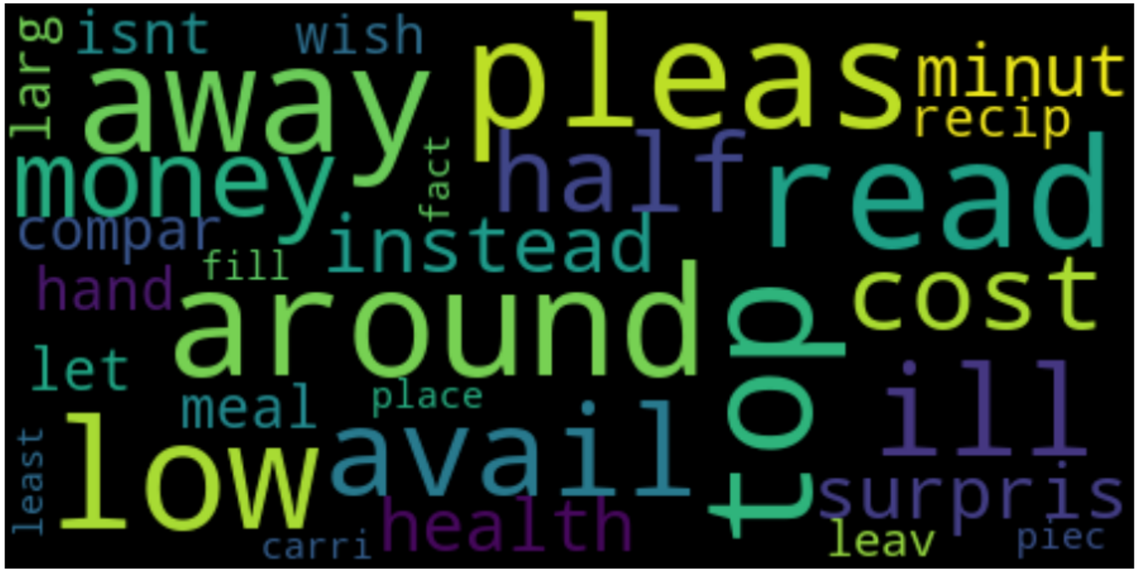
In [15]:
```
#Get number of words occuring in each cluster.
top_feats = [str(i) for i in df_topFeatures['Top Words']]
labels = list(set(kmeans.labels_))
clusters_list = [] #clusters_list will contain all the clusters, i.e. i
t contains words in all the clusters.
for i in labels:
    temp = []
    for word_idx in range(kmeans.labels_.shape[0]):
        if (kmeans.labels_[word_idx] == i):
            temp.append(top_feats[word_idx])
    clusters_list.append(temp)
```

[7.0] Wordclouds of clusters obtained in the above section

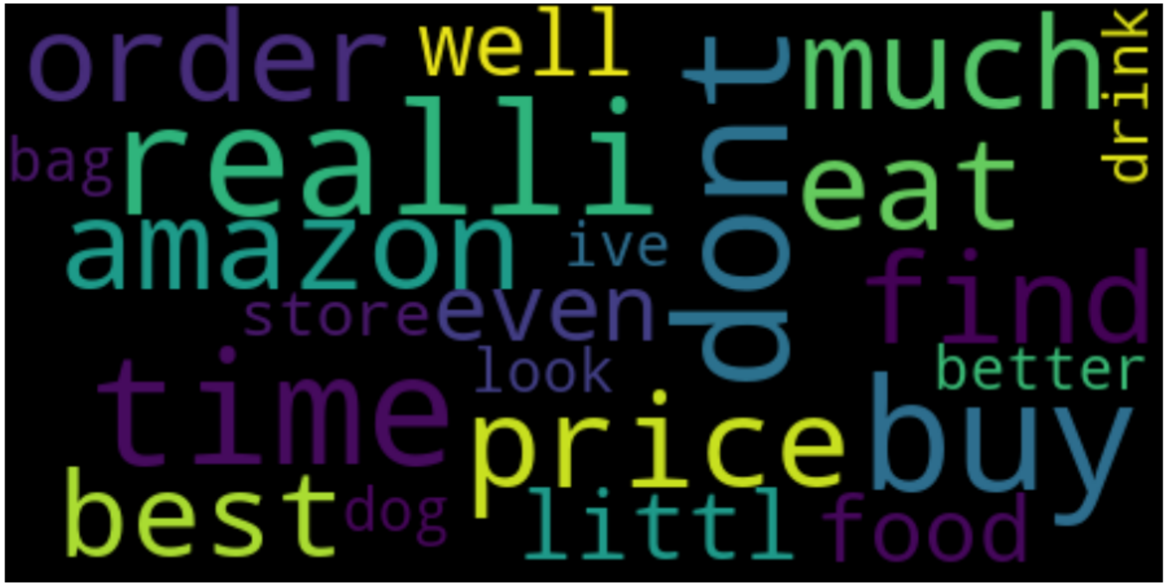
In [16]:
```
#Function to draw word clouds for each clusters.
def word_clouds(clusters_list):
    cluster_count = 1
    for cluster in clusters_list:
        word_corpus = ""
        for word in cluster:
            word_corpus += " " + word
```

```
        #we will select the maximum number of words from each cluster t
o be 20 to see the cluster at one go and understand briefly on what it
 means and how it has clustered data.
        wordcloud = WordCloud(width=450, height=225, max_words=30, rand
om_state=5, background_color='black',
                                      max_font_size=60, font_step=1, mode='RGB'
, repeat=False)
        wordcloud.generate(word_corpus)
        plt.figure(figsize=(16,9))
        plt.title("Word Cloud for Cluster {}, Number of words in cluste
r: {}".format(cluster_count,len(cluster)))
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        plt.show()
        cluster_count+=1

#Draw word cloud for each clusters
word_clouds(clusters_list)
```

Word Cloud for Cluster 1, Number of words in cluster: 1760

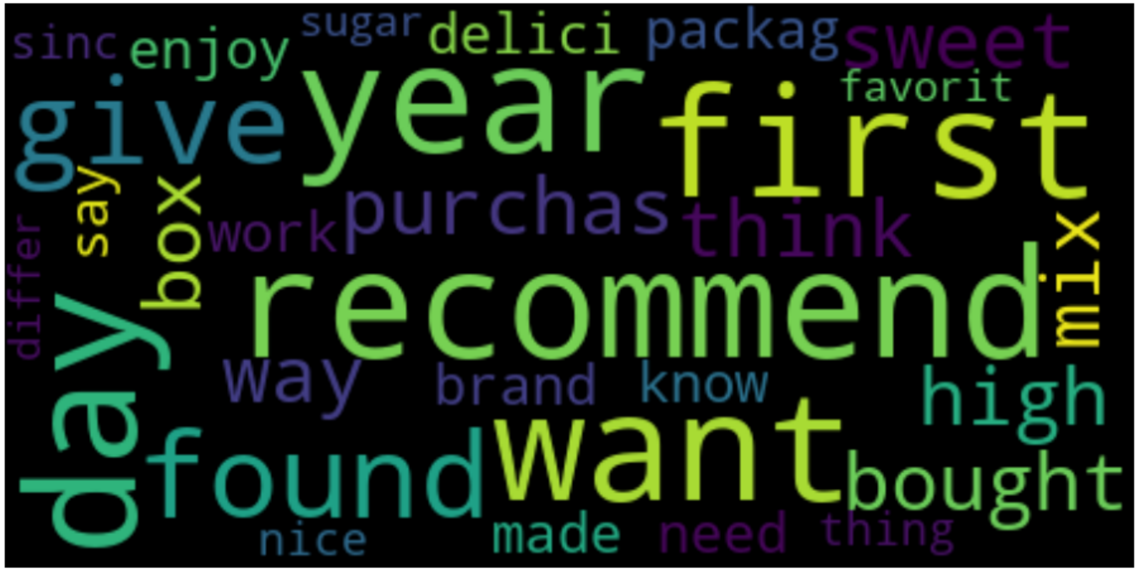


Word Cloud for Cluster 2, Number of words in cluster: 25

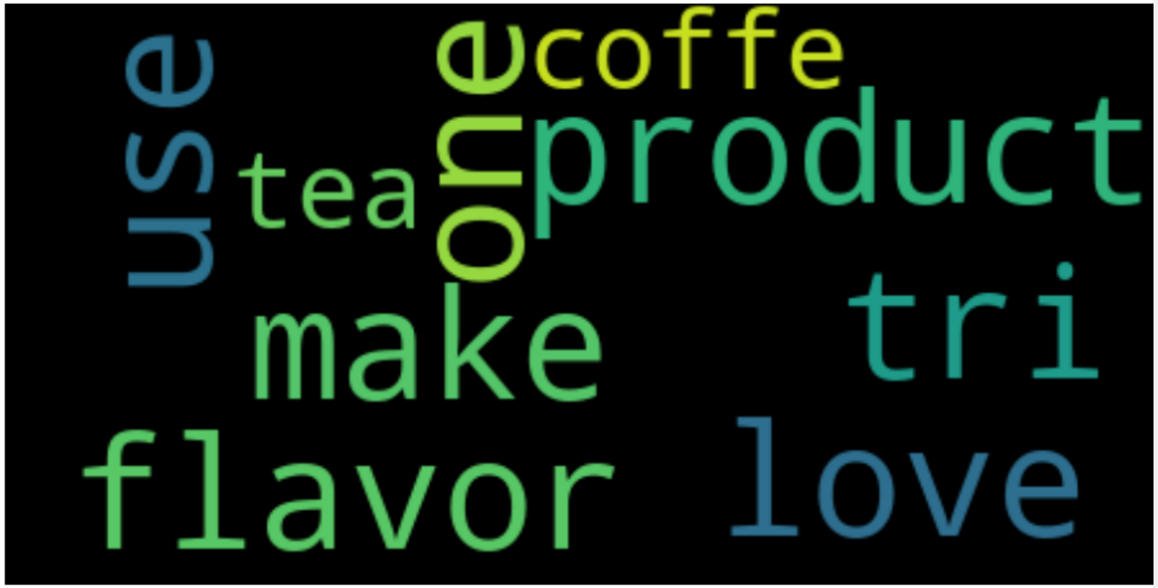


Word Cloud for Cluster 3, Number of words in cluster: 3

Word Cloud for Cluster 4, Number of words in cluster: 202



Word Cloud for Cluster 5, Number of words in cluster: 9



Word Cloud for Cluster 6, Number of words in cluster: 1

Inference

- Cluster 1 contains all the words realted to money, cost, top products, low price etc.
- Cluster 2 contains words related to amazon, buy, order, price etc.
- Cluster 3 contains only the words good & great.
- Cluster 4 contains words related to purchase and related information such as year, day, package, bought, recommended products etc.
- Cluster 5 contains all the words about the product, tea, coffee and their flavor.
- Cluster 6 only contains the word taste.

[8.0] Function that returns most similar words for a given word.

In [19]:
```
#Function to obtain cosine similarity
def cosine_sim(pt1, pt2):
    cos_dis = dot(pt1, pt2)/(norm(pt1)*norm(pt2))
    return (1-cos_dis) #cosine similarity = 1 - cosine distance


# Function that takes a word and returns the most similar words using c
osine similarity between the vectors.
def get_nearest_words(U,top_features,input_word):
    print("Words related to '{}':".format(input_word))

    #Stemming and stopwords removal
    sno = SnowballStemmer(language='english')
    input_word=(sno.stem(input_word.lower()))
    top_words=list(top_features['Top Words'])
    if input_word in top_words:
        for i in range(len(top_words)):
            if input_word == top_words[i]:
                index = i
```

```
        similarity_values = []
        for i in range(U.shape[0]): #U contains word vectors correspond
ing to all words.
            similarity_values.append(cosine_sim(U[i], U[index]))

        sorted_indexes = np.array(similarity_values).argsort()

        #Display top 10 nearest words to the input words in a PrettyTab
le format.
        sim_words = []
        sim_scores = []
        for i in range(1, 11):
            sim_words.append(top_words[sorted_indexes[i]])
            sim_scores.append(1-similarity_values[sorted_indexes[i]])

        # displaying similar words using PrettyTable
        table = PrettyTable()
        table.add_column("Similar Words", sim_words)
        table.add_column("Similarity Scores", sim_scores)
        print(table)

    else:
        print("This word is not present in the vocabulary of top word
s.")
```

In [20]:

```
word = input('Enter a word')
get_nearest_words(U, df_topFeatures, word)
```

```
 Enter a wordamazon
Words related to 'amazon':
+---------------+--------------------+
| Similar Words | Similarity Scores  |
+---------------+--------------------+
|     onlin     | 0.800842480157968  |
|      item     | 0.7367959279192321 |
|     vendor    | 0.7345467640502149 |
|    internet   | 0.7271696735284536 |
|      sale     | 0.723432155197395  |
```

```
|    supplier   | 0.7149858177405637 |
|     seller    | 0.7071241804714099 |
|      case     | 0.7008556341116979 |
|     costco    | 0.6997404624411739 |
|    walmart    | 0.6935515255301391 |
+---------------+--------------------+
```

In [21]:
```
word = input('Enter a word')
get_nearest_words(U, df_topFeatures, word)
```

```
Enter a wordproduct
Words related to 'product':
+---------------+--------------------+
| Similar Words | Similarity Scores  |
+---------------+--------------------+
|      also     | 0.8694553843184923 |
|     anyway    | 0.8436874060398843 |
|     stuff     | 0.8259774125606616 |
|     cours     | 0.8173438684019001 |
|     howev     | 0.8151546727924123 |
|      item     | 0.8042092131842262 |
|     still     | 0.8020331265029602 |
|      idea     | 0.801232392546523  |
|    starter    | 0.7981431711944302 |
|      lol      | 0.7953183792732914 |
+---------------+--------------------+
```

[9.0] Conclusions

Hyperparameters:-

- n_components = 500
- no. of clusters = 6

Steps Performed:-

- Perform TFIDF Vectorization on the reviews
- Select the top 2000 features based on tfidf scores
- Calculate the co-occurence matrix
- Plot % of variance retained vs number of components
- Select the optimal number of components
- Apply Truncated SVD on the co-occurence matrix with n_components=otpimal number of components
- Obtain the U matrix & apply kmeans on it by finding the optimal no. of clusters
- Plot Wordclouds for each cluster
- Define a function that returns similar words to the input word based on cosine similarity