

---

# IMPLEMENTATION OF WORD2VEC MODEL ON WIKIPEDIA ARTICLES

---

FATHIMA SYEDA – 2790024

## Introduction

We shall implement our project on text analysis of webpages from the Wikipedia website. We will be generating Word2Vec for each word in Wikipedia articles. We employ document vectorization in which we convert the long text documents into concise vector representations of the words that can be searched over efficiently. Then later we visualise the word2vec with t-SNE visualisation. We also perform K\_Means clustering on the word2vec model vocabulary built on our Wikipedia data.

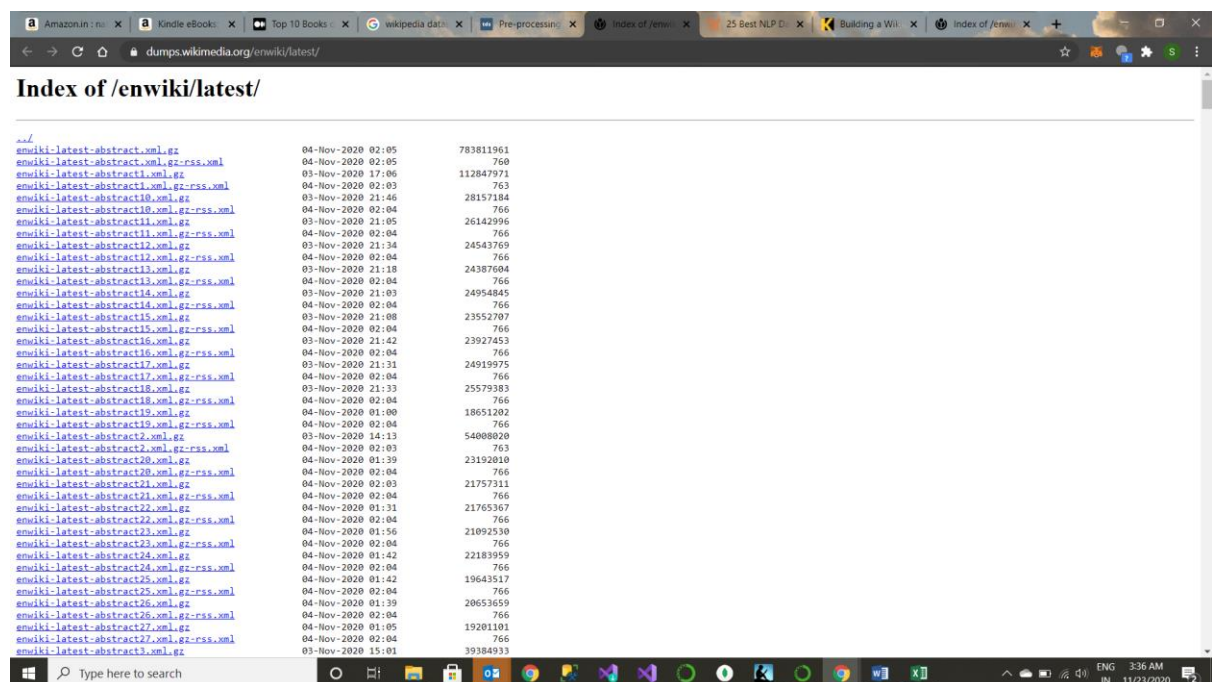
**Data Source:** webpages on the Wikipedia site

<http://www.wikipedia.com/>

## Data Collection

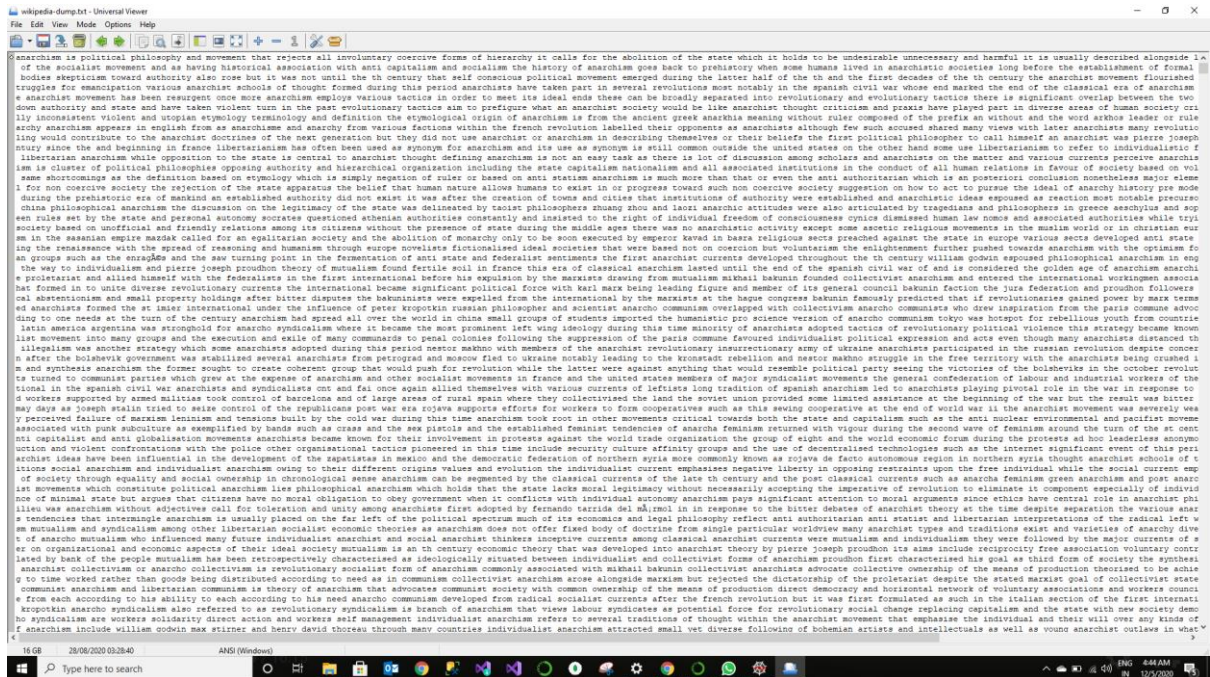
We download the Wikipedia dataset collection for our text mining task from the Wikipedia dump.

The wikipedia dataset is 16Gb in size and is very big. We only use a subset of this dataset.



The Wikipedia dump is huge in size but we shall only use a small section of it, we extract the downloaded files and clean them.

After cleaning the Wikipedia dump, we pre-process it by sentence-tokenizing the articles, as well as writing them one-sentence-per-line to a single text file. This is done using the BLingFireTokenizer.



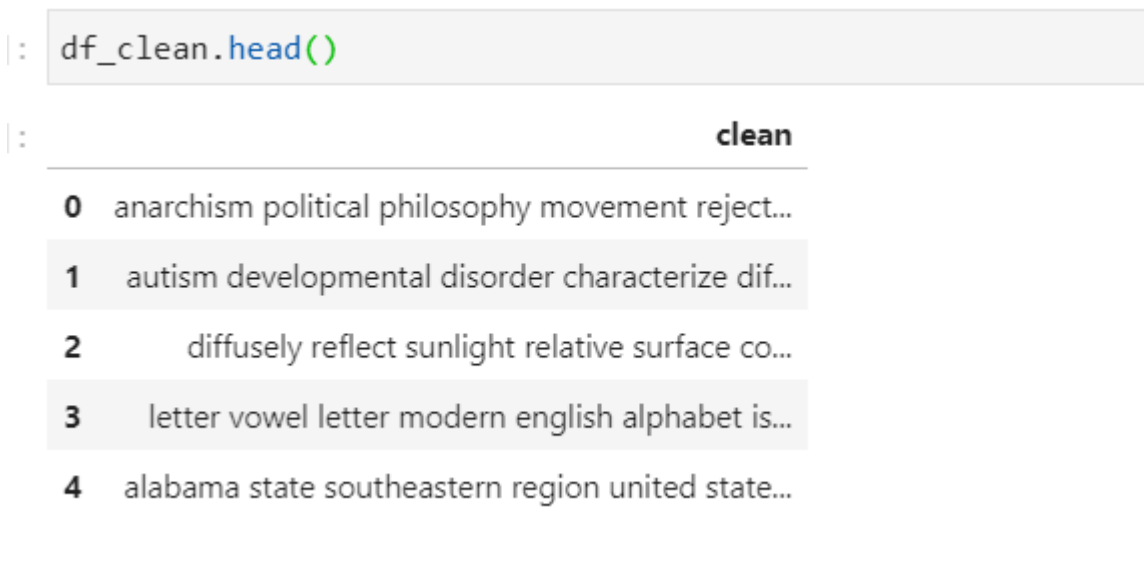
# Pre-processing

After cleaning the Wikipedia dump, we pre-process the text file of one-line per article. We only use 2000 lines from this dump as our training dataset.

We perform the following on our training dataset. We perform lemmatizing on the words and removing the stop words and non-alphabetic characters.

```
nlp = spacy.load('en', disable=['ner', 'parser'])  
brief_cleaning = (re.sub("[^A-Za-z']+ ", '', str(row)).lower() for row in Lines)
```

Some of our articles have non-alphabetic characters from formulae or other characters these need to be removed.



We also by using Gensim Phrases package to detect common phrases (bigrams) like new york, donald trump etc from a list of sentences as we want our vocabulary to not just rely on single words.

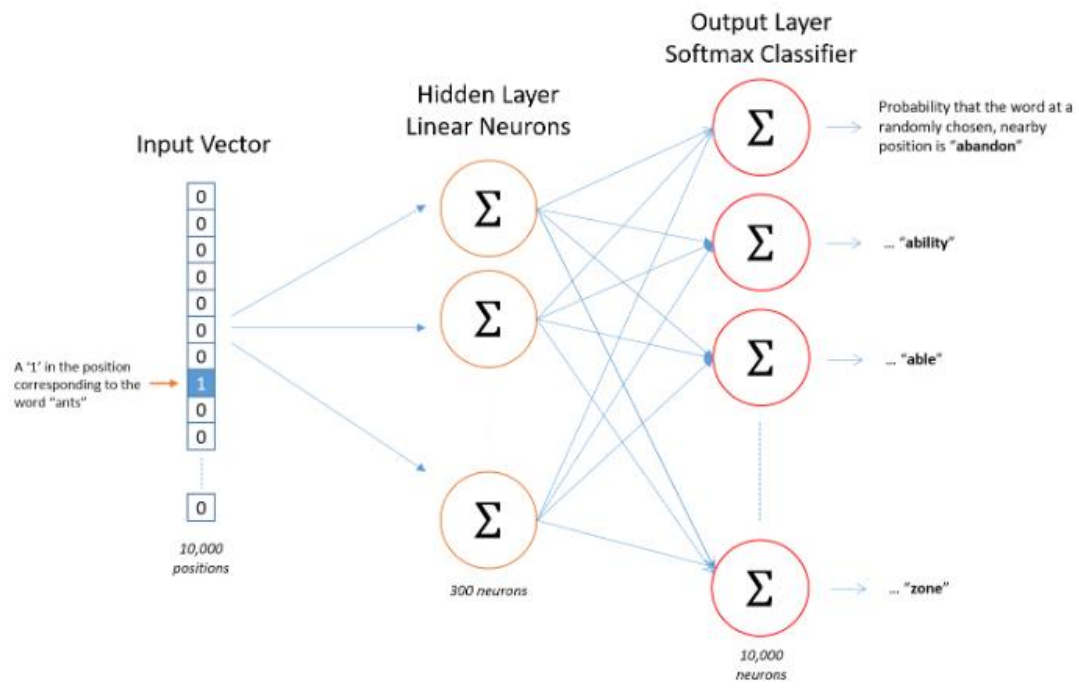
We get a list of most frequent words 104286

```
: sorted(word_freq, key=word_freq.get, reverse=True)[:10]
```

```
: ['include',  
  'time',  
  'american',  
  'state',  
  'year',  
  'work',  
  'new',  
  'form',  
  'use',  
  'city']
```

## Gensim Word2Vec Implementation

We use Gensim implementation of word2vec (CBOW).



Word2Vec works as when two words sharing similar contexts also share a similar meaning and consequently a similar vector representation from the model. For instance: "dog", "puppy" and "pup" are often used in similar situations, with similar surrounding words like "good", "fluffy" or "cute", and according to Word2Vec they will therefore share a similar vector representation.

## • Training the model:

In this first step, we set up the parameters of the model one-by-one. We are using the CBOW Word2Vec model implementation. The parameters are set as follows :

- `min_count = int` - Ignores all words with total absolute frequency lower than this - (2, 100)
- `window = int` - The maximum distance between the current and predicted word within a sentence. E.g. window words on the left and window words on the right of our target - (2, 10)
- `size = int` - Dimensionality of the feature vectors. - (50, 300)
- `negative = int` - If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drawn. If set to 0, no negative sampling is used. - (5, 20)
- `workers = int` - Use these many worker threads to train the model (=faster training with multicore machines)

Ignores all words with total absolute frequency lower than this - (2, 100)

words on the left and window words on the right of our target - (2, 10)

Dimensionality of the feature vectors. - (50, 300)

```
w2v_model = Word2Vec(min_count=20,
                      window=2,
                      size=300,
                      sample=6e-5,
```

```
alpha=0.03,  
min_alpha=0.0007,  
negative=20,  
workers=cores-1)
```

- Building the Vocabulary Table

We build the vocabulary table (taking all the words and filtering out the unique words, and doing some basic counts on them)

- Training of the model

- Train the model on w2v corpus for 30 epochs

The input size is the number of sentences

## Exploring the model

Once our model is trained and our word2vec vocabulary is built for the Wikipedia articles we do some exploring of the data model.

We find the most similar words to the word “American”



the most similar words to the word “trump”, then we find the similarity score between two given words –trump and Donald, America and Nixon. We also find the most dissimilar words from the given set of words[trump, usa, India]

## t-SNE VISUALIZATIONS:

T—SNE(t-Distributed Stochastic Neighbor Embedding) is a dimensionality reduction algorithm to represent high-dimensional data and the underlying relationships between vectors in lower dimensions.

It is an [unsupervised, non-linear technique](#) primarily used for data exploration and visualizing high-dimensional data.

In our tsne we set the pca =19 features.

- 10 Most similar words vs. 8 Random words

In the following visualization we give the word Trump and 8 random words and project them against the 10 most similar words to Trump

'trump', ['businessman', 'actor', 'screenwriter', 'korea', 'india', 'canadian', 'tower', 'lunch']

We notice that the top 10 most similar words to trump are names of other presidents and are closer in to it while the random words are marked further.



## 10 Most similar words vs. 10 Most dissimilar words

In the following visualization we give the word Trump and project the 10 most similar words to Trump against the 10 most dissimilar words to Trump

We notice that the top 10 most similar words to trump are names of other presidents and are closer in to it while the most dissimilar words are geographical locations.

In the following visualization we give the word Hollywood and project the 10 most similar words to Hollywood against the 10 most dissimilar words to Hollywood

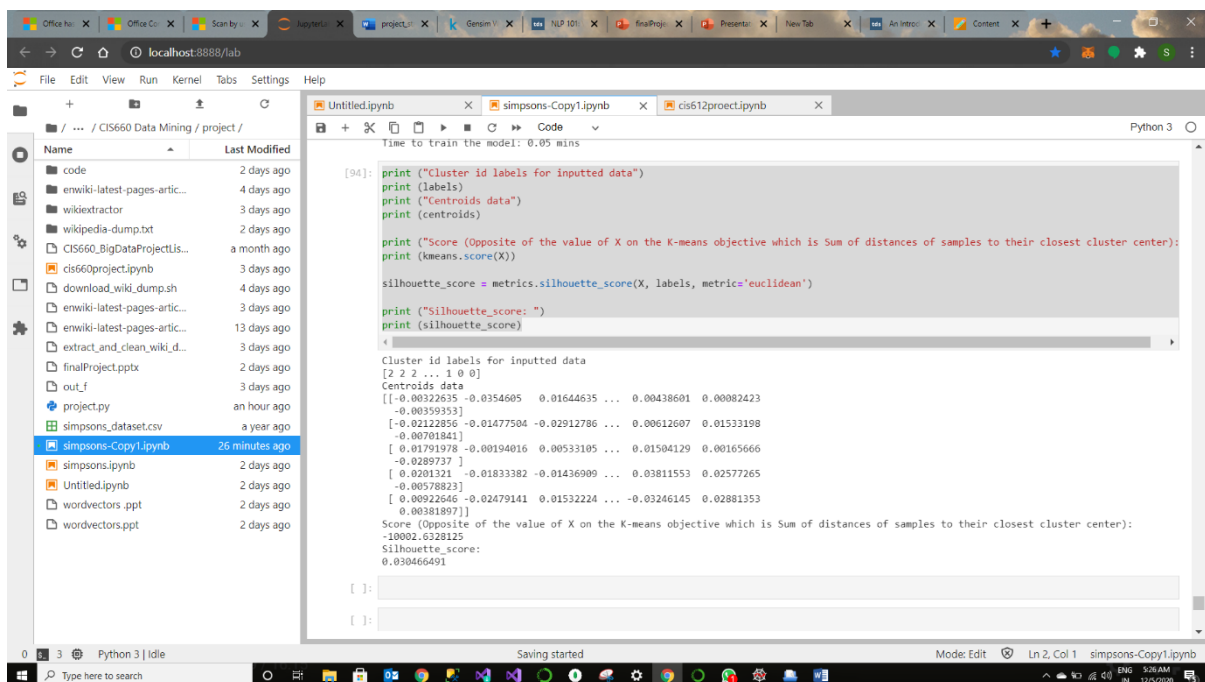
We notice that the top 10 most similar words to Hollywood are names of producers, screenwriters and directors and are closer in to it while the most dissimilar words are geographical locations.

## 10 Most similar words vs. 11th to 20th Most similar words

In the following visualization we give the word Pandemic and project the 10 most similar words to Pandemic against the 11th to 20th Most similar words .

# K-means Clustering on Word2Vec

We consider the different values of k from 1 to 10 to train the kmeans algorithm to find the best value of k. and we found that the elbow is at k=5 with the silhouette score is 0.030466.



```
[94]: print("Cluster id labels for inputted data")
      print(labels)
      print("Centroids data")
      print(centroids)

      print("Score (Opposite of the value of X on the K-means objective which is Sum of distances of samples to their closest cluster center):")
      print(kmeans.score(X))

      silhouette_score = metrics.silhouette_score(X, labels, metric='euclidean')
      print("Silhouette_score:")
      print(silhouette_score)

Cluster id labels for inputted data
[2 2 2 ... 1 0 0]
Centroids data
[[-0.00322635 -0.0354605  0.01644635 ... 0.00438601  0.00082423
  -0.00359353]
 [-0.02122856 -0.01477504 -0.02912786 ... 0.00612607  0.01533198
  -0.00701841]
 [ 0.01791978 -0.00194016  0.00533105 ... 0.01504129  0.00165666
  -0.0289737 ]
 [ 0.0201321  -0.01833382 -0.01436909 ... 0.03811553  0.02577265
  -0.00578823]
 [ 0.00922646 -0.02479141  0.01532224 ... -0.03246145  0.02881353
  0.00381897]]
Score (Opposite of the value of X on the K-means objective which is Sum of distances of samples to their closest cluster center):
-10002.6228125
Silhouette_score:
0.030466491
```

# Conclusion

We build the word2vec model on 2000 wikipedia articles. Once the model was built we did some data exploration by finding similar words to given words and finding the similarity between 2 words. We also found the dissimilar word among a given list of words. We also did some t-SNE visualisations on the data for 10 Most similar words vs. 8 Random words, 10 Most similar words vs. 10 Most dissimilar words and 10 Most similar words vs. 11th to 20th Most similar words. After that we also did kmeans clustering on them. We found the kmeans with different values of k from 1 to 10 to find the best k value. And found that the model performs best at k=5 as shown in the graph that the elbow is at 5.

Our word2vec model is a CBOW (Continuous bag of words model) which considers the history and context of the focus word. CBOW is faster and has better representations and accuracy for more frequent words.

## **Source Code:**

```
#!/usr/bin/env python
#!/bin/sh

set -e
```

```
WIKI_DUMP_FILE_IN=$1
```

```
WIKI_DUMP_FILE_OUT=${WIKI_DUMP_FILE_IN%%.*}.txt
```

```
# clone the WikiExtractor repository
```

```
git clone https://github.com/attardi/wikiextractor.git
```

```
# extract and clean the chosen Wikipedia dump
```

```
echo "Extracting and cleaning $WIKI_DUMP_FILE_IN to $WIKI_DUMP_FILE_OUT..."
```

```
python3 wikiextractor/WikiExtractor.py $WIKI_DUMP_FILE_IN --processes 8 -q -o - \
```

```
| sed "/^\s*$/d" \
```

```
| grep -v "^<doc id=" \
```

```
| grep -v "</doc>$" \
```

```
> $WIKI_DUMP_FILE_OUT
```

```
echo "Successfully extracted and cleaned $WIKI_DUMP_FILE_IN to $WIKI_DUMP_FILE_OUT"
```



```

import re

import pandas as pd

from time import time

from collections import defaultdict

import spacy

import logging

logging.basicConfig(format="%(levelname)s - %(asctime)s: %(message)s", datefmt= '%H:%M:%S',
                    level=logging.INFO)

file=open("wikipedia-dump.txt\\wikipedia-dump.txt",errors='ignore')


Lines=[]

for i in range(1000):

    line=file.readline()

    Lines.append(line)


import en_core_web_sm

nlp = en_core_web_sm.load(disable=['ner', 'parser'])


def cleaning(doc):

    # Lemmatizes and removes stopwords

    # doc needs to be a spacy Doc object

    txt = [token.lemma_ for token in doc if not token.is_stop]

    # Word2Vec uses context words to learn the vector representation of a target word,

    # if a sentence is only one or two words long,

    # the benefit for the training is very small

    if len(txt) > 2:

        return ' '.join(txt)


brief_cleaning = (re.sub("[^A-Za-z']+", ' ', str(row)).lower() for row in Lines)


t = time()

```

```
txt = [cleaning(doc) for doc in nlp.pipe(brief_cleaning, batch_size=5000, n_threads=-1)]
```

```
print('Time to clean up everything: {} mins'.format(round((time() - t) / 60, 2)))
```

```
df_clean = pd.DataFrame({'clean': txt})
```

```
df_clean = df_clean.dropna().drop_duplicates()
```

```
from gensim.models.phrases import Phrases, Phraser
```

```
sent = [row.split() for row in df_clean['clean']]
```

```
phrases = Phrases(sent, min_count=30, progress_per=100)
```

```
bigram = Phraser(phrases)
```

```
sentences = bigram[sent]
```

```
word_freq = defaultdict(int)
```

```
for sent in sentences:
```

```
    for i in sent:
```

```
        word_freq[i] += 1
```

```
len(word_freq)
```

```
print("top10 frequent words in our dataset are:")
```

```
print(sorted(word_freq, key=word_freq.get, reverse=True)[:10])
```

```
#training the model
```

```
import multiprocessing
```

```
from gensim.models import Word2Vec
```

```
cores = multiprocessing.cpu_count()
```

```
w2v_model = Word2Vec(min_count=20,
```

```
    window=2,
```

```
    size=300,
```

```

        sample=6e-5,
        alpha=0.03,
        min_alpha=0.0007,
        negative=20,
        workers=cores-1)

#build vocab table

t = time()

w2v_model.build_vocab(sentences, progress_per=100)

print('Time to build vocab: {} mins'.format(round((time() - t) / 60, 2)))

#training the model

t = time()

w2v_model.train(sentences, total_examples=w2v_model.corpus_count, epochs=30,
report_delay=1)

print('Time to train the model: {} mins'.format(round((time() - t) / 60, 2)))

w2v_model.init_sims(replace=True)

#most similar to

print("most similar words to american are:")
print(w2v_model.wv.most_similar(positive=["american"]))
print("most similar words to pandemic are:")
print(w2v_model.wv.most_similar(positive=["pandemic"]))
print("most similar words to election are:")
print(w2v_model.wv.most_similar(positive=["election"]))
print("most similar words to trump are:")
print(w2v_model.wv.most_similar(positive=["trump"]))

```

```

print("most similarity between words trump and president:")
print(w2v_model.wv.similarity("trump", 'president'))
print("most similarity between words america and nixon:")
print(w2v_model.wv.similarity('america', 'nixon'))
print("the most dissimilar of the words trump, donald, india is:")
print(w2v_model.wv.doesnt_match(['trump', 'donald', 'india']))

```

```

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

```

```

import seaborn as sns
sns.set_style("darkgrid")

```

```

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

```

```

def tsnescatterplot(model, word, list_names):

```

```

    """ Plot in seaborn the results from the t-SNE dimensionality reduction algorithm of the vectors of
    a query word,

```

```

    its list of most similar words, and a list of words.

```

```

    """

```

```

    arrays = np.empty((0, 300), dtype='f')

```

```

    word_labels = [word]

```

```

    color_list = ['red']

```

```

    # adds the vector of the query word

```

```

    arrays = np.append(arrays, model.wv.__getitem__([word]), axis=0)

```

```

    # gets list of most similar words

```

```

    close_words = model.wv.most_similar([word])

```

```

# adds the vector for each of the closest words to the array
for wrd_score in close_words:
    wrd_vector = model.wv.__getitem__([wrd_score[0]])
    word_labels.append(wrd_score[0])
    color_list.append('blue')
    arrays = np.append(arrays, wrd_vector, axis=0)

# adds the vector for each of the words from list_names to the array
for wrd in list_names:
    wrd_vector = model.wv.__getitem__([wrd])
    word_labels.append(wrd)
    color_list.append('green')
    arrays = np.append(arrays, wrd_vector, axis=0)

# Reduces the dimensionality from 300 to 50 dimensions with PCA
reduc = PCA(n_components=19).fit_transform(arrays)

# Finds t-SNE coordinates for 2 dimensions
np.set_printoptions(suppress=True)

Y = TSNE(n_components=2, random_state=0, perplexity=15).fit_transform(reduc)

# Sets everything up to plot
df = pd.DataFrame({'x': [x for x in Y[:, 0]],
                  'y': [y for y in Y[:, 1]],
                  'words': word_labels,
                  'color': color_list})

fig, _ = plt.subplots()
fig.set_size_inches(9, 9)

```

```
# Basic plot
```

```
p1 = sns.regplot(data=df,
                  x="x",
                  y="y",
                  fit_reg=False,
                  marker="o",
                  scatter_kws={'s': 40,
                              'facecolors': df['color']}
                  )
```

```
# Adds annotations one by one with a loop
```

```
for line in range(0, df.shape[0]):  
    p1.text(df["x"][line],  
            df['y'][line],  
            ' ' + df["words"][line].title(),  
            horizontalalignment='left',  
            verticalalignment='bottom', size='medium',  
            color=df['color'][line],  
            weight='normal'  
    ).set_size(15)
```

```
plt.xlim(Y[:, 0].min()-50, Y[:, 0].max()+50)
```

```
plt.ylim(Y[:, 1].min()-50, Y[:, 1].max()+50)
```

```
plt.title('t-SNE visualization for {}'.format(word.title()))
```

```
tsnecscatterplot(w2v_model, 'trump', ['businessman', 'actor', 'screenwriter', 'korea', 'india',  
'canadian', 'tower', 'lunch'])
```

```
tsnecatterplot(w2v_model, 'hollywood', [i[0] for i in
w2v_model.wv.most_similar(negative=["trump"])]))

tsnecatterplot(w2v_model, "pandemic", [t[0] for t in
w2v_model.wv.most_similar(positive=["pandemic"], topn=20)][10:])

print ("The vocabulary of our wiki documents is {}".format(len(w2v_model.wv.vocab)))
```

```
X=w2v_model[w2v_model.wv.vocab]
```

```
from nltk.cluster import KMeansClusterer
import nltk
import numpy as np
```

```
from sklearn import cluster
from sklearn import metrics
```

```
t=time()
from sklearn.cluster import KMeans
#kmeans for 10 values of k
kmeans_per_k = [KMeans(n_clusters=k, random_state=42).fit(X)
                 for k in range(1, 10)]
```

```
# reduc = PCA(n_components=19).fit_transform(arrays)
```

```
def Kmeans_Inertia_Score(XD,title,kmeans_per_k):
```

```
    inertias = [model.inertia_ for model in kmeans_per_k]
```

```
plt.figure(figsize=(8, 3.5))
plt.plot(range(1, 10), inertias, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Inertia", fontsize=14)
plt.annotate('Elbow',
```



```

        xy=(5, inertias[3]),
        xytext=(0.55, 0.55),
        textcoords='figure fraction',
        fontsize=16,
        arrowprops=dict(facecolor='black', shrink=0.1)
    )

plt.title("inertia score against k "+title)
plt.show()

#inertia score
print("KMEANS Inertia- Mean square error score")
Kmeans_Inertia_Score(X,"kmeans_per_k)

#Kmeans_Inertia_Score(X2D,' for nPCA=2',kmeans_per_k_2DPCA)
print('Time to train the model: {} mins'.format(round((time() - t) / 60, 2)))

t=time()
kmeans = cluster.KMeans(n_clusters=5)
kmeans.fit(X)

labels = kmeans.labels_
centroids = kmeans.cluster_centers_

print('Time to train the model: {} mins'.format(round((time() - t) / 60, 2)))
print ("Cluster id labels for inputted data")
print (labels)
print ("Centroids data")
print (centroids)

print ("Score (Opposite of the value of X on the K-means objective which is Sum of distances of
samples to their closest cluster center):")
print (kmeans.score(X))

```

```
silhouette_score = metrics.silhouette_score(X, labels, metric='euclidean')
```

```
print ("Silhouette_score: ")
```

```
print (silhouette_score)
```

