In [1]:

```
NAME = "Tyler Freund"
```

Lab 7: Dimensionality Reduction

Please read the following instructions very carefully

Working on the assignment / FAQs

- **Always use the seed/random_state as *42* wherever applicable (This is to ensure repeatability in answers, across students and coding environments)**
- **The type of question and the points they carry are indicated in each question cell**
- **To avoid any ambiguity, each question also specifies what *value* must be set. Note that these are dummy values and not the answers**
- **If an autograded question has multiple answers (due to differences in handling NaNs, zeros etc.), all answers will be considered.**
- **You can delete the `raise NotImplementedError()`**
- **Submitting the assignment : Download the '.ipynb' file and the PDF file from Colab and upload it to bcourses. Do not delete any outputs from cells before submitting.**
- **That's about it. Happy coding!**

Available software:

- **Python's Gensim module: https://radimrehurek.com/gensim/ (install using pip)**
- **Sklearn's TSNE module in case you use TSNE to reduce dimension (optional)**
- **Python's Matplotlib (optional)**

Note: The most important hyper parameters of skip-gram/CBOW are vector size and windows size

In [2]:

```
!pip install gensim

import pandas as pd
import numpy as np
import gensim
import requests
import string

from IPython.display import Image
from sklearn.manifold import TSNE


!git clone https://github.com/CAHLR/d3-scatterplot.git
from google.colab.output import eval_js
from IPython.display import Javascript
from gensim.models import KeyedVectors

!wget -nc https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz

import nltk
from nltk import word_tokenize, tokenize
nltk.download('punkt')
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (f
rom gensim) (1.4.1)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-package
```

```
s (from gensim) (5.2.1)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist-packages (f
rom gensim) (1.19.5)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from
gensim) (1.15.0)
Cloning into 'd3-scatterplot'...
remote: Enumerating objects: 1045, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 1045 (delta 16), reused 0 (delta 0), pack-reused 1016
Receiving objects: 100% (1045/1045), 1.95 MiB | 17.38 MiB/s, done.
Resolving deltas: 100% (605/605), done.
--2021-11-02 22:18:27--  https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-ne
gative300.bin.gz
Resolving s3.amazonaws.com (s3.amazonaws.com)... 54.231.130.184
Connecting to s3.amazonaws.com (s3.amazonaws.com)|54.231.130.184|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1647046227 (1.5G) [application/x-gzip]
Saving to: 'GoogleNews-vectors-negative300.bin.gz'

GoogleNews-vectors- 100%[===================>]   1.53G  31.0MB/s    in 65s

2021-11-02 22:19:32 (24.1 MB/s) - 'GoogleNews-vectors-negative300.bin.gz' saved [16470462
27/1647046227]

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[2]:

```
True
```

Q1 (0.25 points)

Download your text corpus. (A good place to start is the nltk corpus or the gutenburg project)

In [3]:

```
#your code here

url = "https://www.gutenberg.org/files/41/41-0.txt"
```

In [4]:

```
#Save the raw text that you just downloaded in this variable
raw = requests.get(url).content.decode('utf8')
```

In [5]:

```
#This is an autograded cell, do not edit/delete
print(raw[:1000])
```

```
Project Gutenberg's The Legend of Sleepy Hollow, by Washington Irving

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: The Legend of Sleepy Hollow

Author: Washington Irving

Posting Date: June 25, 2008 [EBook #41]
Release Date: October, 1992
Last Updated: September 14, 2016

Language: English

Character set encoding: UTF-8
```

```
*** START OF THIS PROJECT GUTENBERG EBOOK THE LEGEND OF SLEEPY HOLLOW ***



Produced by Ilana M. (Kingsley) Newby and Greg Newby




THE LEGEND OF SLEEPY HOLLOW


by Washington Irving




FOUND AMONG THE PAPERS OF THE LATE DIEDRICH KNICKERBOCKER.


        A pleasing land of drowsy head it was,
          Of dreams that wave before the half-shut eye;
        And of gay castles in the clouds that pass,
```

Q2 (0.25 points)

Tokenize your corpus. Make sure that that the result is a list of lists i.e. The top-level list (outer list) is a list of sentences, and the inner list is a list of words in a given sentence.

Consider the following text:

```
text = "I spent $15.35 on my lunch today. Food in Berkeley is very expensive!"
```

It could be tokenized as follows:

```
tok_corp = [['I', 'spent', '$', '15.35', 'on', 'my', 'lunch', 'today'],
 ['Food', 'in', 'Berkeley', 'is', 'very', 'expensive']]
```

Note: There are many different (and correct) ways of tokenizing. Your answer doesn't need to match exactly with this illustrative example.

In [6]:

```
#code here
firstlook = tokenize.sent_tokenize(raw)

pattern = r'''(?x)  # set flag to allow verbose regexps
(?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
|\w+(?:[-']\w+)*    # words with optional internal hyphens
|\$?\d+(?:\.\d+)?   # currency, e.g. $12.80
|\.\.\.             # elipses
|[.,;"'?()-_`]      # these are separate tokens
'''
tokenized_raw = " ".join(nltk.regexp_tokenize(raw, pattern))
tokenized_raw = tokenize.sent_tokenize(tokenized_raw)

nopunct = []
for sent in tokenized_raw:
    a = [w for w in sent.split() if w not in string.punctuation]
    nopunct.append(" ".join(a))
```

In [8]:

```
#Save the tokenized sentences as a list of list in this variable
```

```
tok_corp = [nltk.word_tokenize(sent) for sent in nopunct]
```

In [9]:

```
#This is an autograded cell, do not edit/delete
for sent in tok_corp[:3]:
  print(sent)
  print("\n")
```

```
['Project', 'Gutenberg', 's', 'The', 'Legend', 'of', 'Sleepy', 'Hollow', 'by', 'Washingto
n', 'Irving', 'This', 'eBook', 'is', 'for', 'the', 'use', 'of', 'anyone', 'anywhere', 'at
', 'no', 'cost', 'and', 'with', 'almost', 'no', 'restrictions', 'whatsoever']


['You', 'may', 'copy', 'it', 'give', 'it', 'away', 'or', 're-use', 'it', 'under', 'the',
'terms', 'of', 'the', 'Project', 'Gutenberg', 'License', 'included', 'with', 'this', 'eBo
ok', 'or', 'online', 'at', 'www']


['gutenberg']
```

Q3 (0.25 points)

Train gensim using your own dataset. Name the trained model variable as `model`.

In [10]:

```
#code here
model = gensim.models.Word2Vec(tok_corp, min_count=1, size=16, window=5)
```

In [11]:

```
#This is an autograded cell, do not edit/delete
print(f'Corpus Size: {model.corpus_total_words}')
print(f'Corpus Count: {model.corpus_count}')
print(f'Training time: {model.total_train_time}')
print(f'Sample words: {list(model.wv.vocab.keys())[:10]}')
```

```
Corpus Size: 15419
Corpus Count: 508
Training time: 0.09190132800131323
Sample words: ['Project', 'Gutenberg', 's', 'The', 'Legend', 'of', 'Sleepy', 'Hollow', 'b
y', 'Washington']
```

Q4 (0.25 points)

Q4a

Create a list of the unique set of words from your corpus. Name the list variable as `unique_words`.

In [15]:

```
#code here
unique_words = list(set([item for sublist in tok_corp for item in sublist]))
```

In [16]:

```
#This is an autograded cell, do not edit/delete
print(unique_words[:10])
```

```
['High', 'music', 'hillside', '2001', 'jockey', 'FULL', 'enthroned', 'leafy', 'motion', '
wives']
```

Q4b

Extract respective vectors corresponding to the words in your corpus and store the vectors in a variable called `vector_list`.

In [17]:

```
#code here
vector_list = model[unique_words]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: Call
to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() i
nstead).
```

In [18]:

```
#This is an autograded cell, do not edit/delete
print(f'Array Shape: {np.array(vector_list).shape}')
for i in range(5):
    print(unique_words[i], vector_list[i])
```

```
Array Shape: (3919, 16)
High [-0.00899667 -0.01898508 -0.00728832  0.01036037 -0.0009049   0.0011302
  0.0327841  -0.03731004  0.01353694  0.03345887  0.01417384  0.00854476
  0.03005959  0.00485835  0.03602692  0.06273617]
music [ 0.00263501  0.02899377  0.01917414  0.0006789  -0.01293606 -0.00991178
  0.01998896  0.01742947 -0.03560718  0.01387622  0.00531943 -0.02698849
  0.02277133  0.02533497  0.02695986  0.03072815]
hillside [ 0.02032239  0.02434933 -0.03384691  0.01301549  0.01834786  0.00360978
  0.0163553  -0.02443611  0.00580833  0.03184057 -0.01468459 -0.01823715
  0.02664859 -0.01227189  0.00368132  0.03014257]
2001 [-0.01291607  0.00129182 -0.03332432 -0.02256056  0.00970524  0.0277961
  0.02815603 -0.0268498  -0.01018582  0.00390703  0.01352851 -0.03375209
 -0.0174372   0.00092753 -0.02454752 -0.0079782 ]
jockey [-0.01594231  0.01081124 -0.00063181  0.02572731  0.0072372  -0.01307207
  0.00705491 -0.01750007 -0.00131171 -0.01050271  0.00691994  0.01678816
  0.00722145 -0.00308109  0.00641087 -0.00266955]
```

Q5 (3 points)

Based on your knowledge and understanding of the text corpus you have chosen, form 3 hypotheses of analogies or relationships (between words) that you expect will hold and give a reason why. Experimentally validate these hypotheses using similarity of the word vectors.

Example: If using Moby Dick as the corpus, one hypothesis might be that the whale, "Moby Dick" is (cosine) more similar to "fate" than to "evil" because Moby Dick is symbolic of the nature and the universe and isn't necessarily 'bad'. Or "Moby Dick" is more similar to "opposition" than to "surrender" because Moby Dick fights for its survival.

Note: Please do NOT use the same example as in the prompt.

Note 2: It's okay if the model disproves your hypotheses.

Hypotheses 1: "Ichabod" is more similar to "good" than "bad" because he is the protagonist.

Hypotheses 2: "Katrina" is more similar to "jealousy" than to "love" because the two men (Brom and Ichabod) fight over her love.

Hypotheses 3: "Headless" is more similar to "fear" than to "real" because the headless horseman is supernatural and rather scary.

In [40]:

```
#your code here for validating hypotheses 1
print('Hypotheses 1: Ichabod similarity score to good')
print(model.similarity("Ichabod", "good"))
print('Hypotheses 1: Ichabod similarity score to bad')
```

```
print(model.similarity("Ichabod", "bad"))
```

```
Hypotheses 1: Ichabod similarity score to good
0.97810763
Hypotheses 1: Ichabod similarity score to bad
0.61234426
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: Call
to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() ins
tead).
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call
to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() ins
tead).
  """
```

In [41]:

```
#your code here for validating hypotheses 2
print('Hypotheses 1: Katrina similarity score to jealousy')
print(model.similarity("Katrina", "jealousy"))
print('Hypotheses 1: Katrina similarity score to love')
print(model.similarity("Katrina", "love"))
```

```
Hypotheses 1: Katrina similarity score to jealousy
0.5329054
Hypotheses 1: Katrina similarity score to love
0.5397118
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: Call
to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() ins
tead).
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call
to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() ins
tead).
  """
```

In [42]:

```
#your code here for validating hypotheses 3
print('Hypotheses 1: Headless similarity score to fear')
print(model.similarity("Headless", "fear"))
print('Hypotheses 1: Headless similarity score to real')
print(model.similarity("Headless", "real"))
```

```
Hypotheses 1: Headless similarity score to fear
0.7870484
Hypotheses 1: Headless similarity score to real
0.749326
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: Call
to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() ins
tead).
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call
to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() ins
tead).
  """
```

Q6 Visualizing the trained vectors (1.5 points)

Q6a

Run Kmeans clustering on your word vectors (as you did in Q-6 of Lab-5). Use the word vectors from the model you trained in this lab.

In [43]:

```
#your code here
```

```
%matplotlib inline
from matplotlib import pyplot as plt

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

kmeans = KMeans(n_clusters=25, random_state=42)
X = np.array(vector_list)
kmeans.fit(X)
# silhouette_score(google_X, google_kmeans.labels_)
kmeans.labels_
```

Out[43]:

```
array([18, 20, 12, ..., 21, 18,  0], dtype=int32)
```

Q6b

Reduce the dimensionality of your word vectors using TSNE

In [44]:

```
#your code here
from sklearn.manifold import TSNE

data_embed=TSNE(n_components=2, perplexity=50, verbose=2, method='barnes_hut').fit_trans
form(vector_list)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 3919 samples in 0.019s...
[t-SNE] Computed neighbors for 3919 samples in 0.731s...
[t-SNE] Computed conditional probabilities for sample 1000 / 3919
[t-SNE] Computed conditional probabilities for sample 2000 / 3919
[t-SNE] Computed conditional probabilities for sample 3000 / 3919
[t-SNE] Computed conditional probabilities for sample 3919 / 3919
[t-SNE] Mean sigma: 0.027784
[t-SNE] Computed conditional probabilities in 0.536s
[t-SNE] Iteration 50: error = 79.3276062, gradient norm = 0.0274379 (50 iterations in 2.7
37s)
[t-SNE] Iteration 100: error = 77.5462494, gradient norm = 0.0003661 (50 iterations in 1.
828s)
[t-SNE] Iteration 150: error = 77.5206299, gradient norm = 0.0000302 (50 iterations in 1.
501s)
[t-SNE] Iteration 200: error = 77.5191193, gradient norm = 0.0000604 (50 iterations in 1.
477s)
[t-SNE] Iteration 250: error = 77.5188293, gradient norm = 0.0000035 (50 iterations in 1.
375s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 77.518829
[t-SNE] Iteration 300: error = 2.8607459, gradient norm = 0.0015602 (50 iterations in 1.6
37s)
[t-SNE] Iteration 350: error = 2.6792605, gradient norm = 0.0005171 (50 iterations in 2.1
31s)
[t-SNE] Iteration 400: error = 2.5966120, gradient norm = 0.0002585 (50 iterations in 2.1
16s)
[t-SNE] Iteration 450: error = 2.5504925, gradient norm = 0.0001595 (50 iterations in 2.0
10s)
[t-SNE] Iteration 500: error = 2.5242720, gradient norm = 0.0001168 (50 iterations in 1.8
83s)
[t-SNE] Iteration 550: error = 2.5080705, gradient norm = 0.0000878 (50 iterations in 1.9
05s)
[t-SNE] Iteration 600: error = 2.4969497, gradient norm = 0.0000687 (50 iterations in 1.9
33s)
[t-SNE] Iteration 650: error = 2.4887493, gradient norm = 0.0000583 (50 iterations in 1.9
82s)
[t-SNE] Iteration 700: error = 2.4831483, gradient norm = 0.0000524 (50 iterations in 2.0
03s)
[t-SNE] Iteration 750: error = 2.4791322, gradient norm = 0.0000468 (50 iterations in 1.9
80s)
[t-SNE] Iteration 800: error = 2.4762270, gradient norm = 0.0000472 (50 iterations in 1.9
83s)
[t-SNE] Iteration 850: error = 2.4740195, gradient norm = 0.0000413 (50 iterations in 1.9
```

```
88s)
[t-SNE] Iteration 900: error = 2.4722323, gradient norm = 0.0000352 (50 iterations in 1.9
96s)
[t-SNE] Iteration 950: error = 2.4705420, gradient norm = 0.0000369 (50 iterations in 2.0
24s)
[t-SNE] Iteration 1000: error = 2.4693112, gradient norm = 0.0000324 (50 iterations in 2.
013s)
[t-SNE] KL divergence after 1000 iterations: 2.469311
```

Q6c

Create a dataframe with the following columns:

Column	Description
x	the first dimension of result from TSNE
y	the second dimension of result from TSNE
Feature 1	the word corresponding to the vector
Feature 2	the kmeans cluster label

Below is a sample of what the dataframe could look like:

x	y	Feature 1	Feature 2
7.154159	9.251100	lips	8
-53.254147	-13.514915	them	9
34.049191	-13.402843	topic	0
-32.515320	28.699677	sofa	24
13.006302	-4.270626	half-past	21

In [45]:

```
#your code here
df = pd.DataFrame(data_embed[:,:2],columns=["x","y"])
df['Feature 1'] = unique_words
df['Feature 2'] = kmeans.labels_
df.head()
```

Out[45]:

	x	y	Feature 1	Feature 2
0	-14.984633	4.343380	High	18
1	-0.541475	-3.343648	music	20
2	0.359864	-23.660727	hillside	12
3	-43.162170	-4.012386	2001	6
4	-37.265121	-16.938587	jockey	6

Q6d: Visualization

In this question, you are required to visualize and explore the reduced dataset you created in Q6c using the [d3-scatterplot](#) library.

Note: The first code-chunk at the top in this notebook clones the libary from github. Make sure that it has been executed before you proceed.

Save your dataset as a tsv file 'd3-scatterplot/mytext.tsv'

Example:

```
df.to_csv('d3-scatterplot/mytext.tsv', sep='\t', index=False)
```

Note 1: The TSV file needs to be stored in the `d3-scatterplot` folder so that the d3-scatterplot library can access it.

Note 2: Make sure to save the TSV file WITHOUT the row index i.e. use `index=False`.

In [46]:

```
#your code here
!git clone https://github.com/CAHLR/d3-scatterplot.git
from google.colab.output import eval_js
from IPython.display import Javascript

df.to_csv('d3-scatterplot/mytext.tsv', sep='\t', index=False)
```

```
fatal: destination path 'd3-scatterplot' already exists and is not an empty directory.
```

Visualize the reduced vectors by running the following code

In [47]:

```
def show_port(port, data_file, width=600, height=800):
  display(Javascript("""
  (async ()=>{
    fm = document.createElement('iframe')
    fm.src = await google.colab.kernel.proxyPort(%d) + '/index.html?dataset=%s'
    fm.width = '90%%'
    fm.height = '%d'
    fm.frameBorder = 0
    document.body.append(fm)
  })();
  """ % (port, data_file, height)))

port = 8000
data_file = 'mytext.tsv'
height = 1400

get_ipython().system_raw('cd d3-scatterplot && python3 -m http.server %d &' % port)
show_port(port, data_file, height)
```

Color the points by their kmeans cluster. You can do this by choosing "Feature 2" as the variable in the "Color" drop-down in the visualization.

Pleae include a snapshot of your visualization in the notebook. Refer to the tutorial video, and/or follow the steps below:

1) Take a sanpshot of the visualization and save it on your computer with the filename `trained_scatter.png`

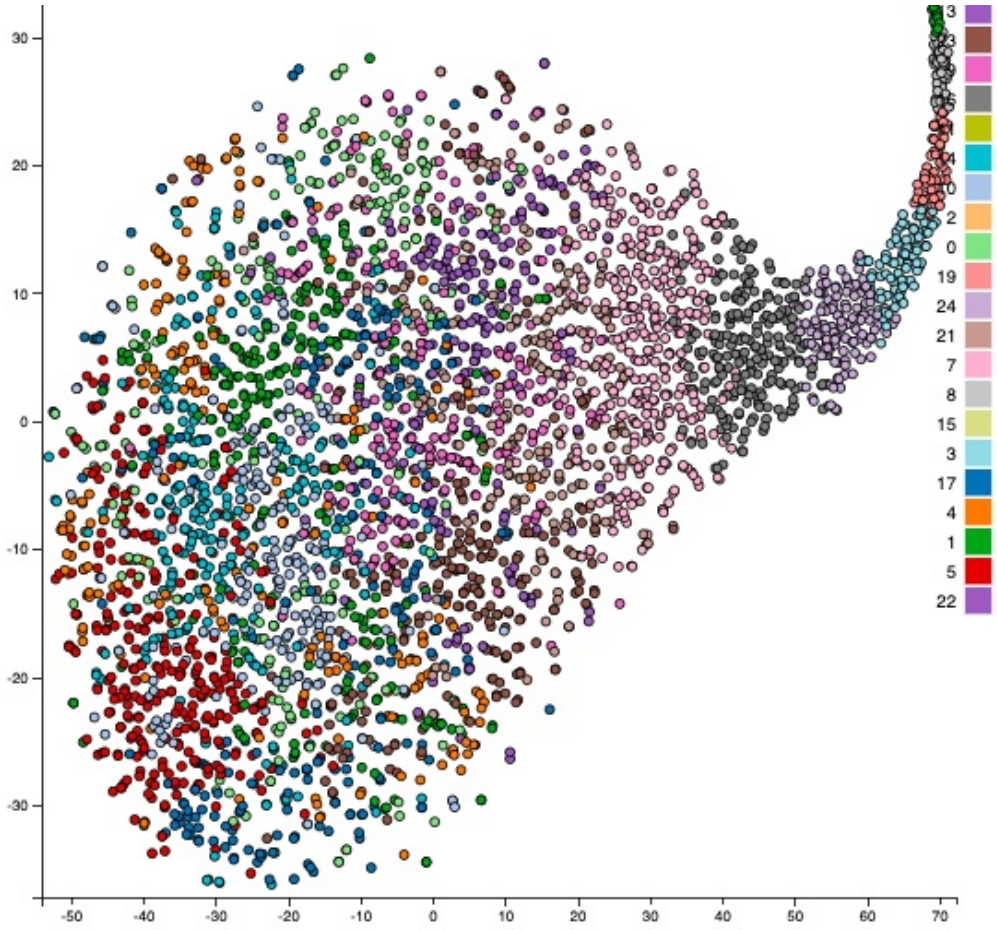
2) Upload the `trained_scatter.png` by clicking on the 'Files' icon on the left sidebar and clicking on the upload icon (the one with an upward arrow on top left)

3) Run the code below to display the image

In [48]:

```
#This is an autograded cell, do not edit/delete
Image('trained_scatter.png')
```

Out[48]:



Q7 Visualizing the PRE-TRAINED vectors (1.5 points)

In this question, you'll execute the same analysis as in Q6, but on PRE-TRAINED vectors.

Q7a

Load the google vector model

(It must be downloaded as `GoogleNews-vectors-negative300.bin.gz` for you if you ran the first code-chunk at the top of this notebook)

In [53]:

```
#your code here
google_model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negat
ive300.bin.gz', binary=True)
```

Downsample the pre-trained google model to anywhere between 10,000 to 25,000 words.

In [54]:

```
#your code here
import random

#Get random sample of words and extract corresponding vectors into array.
all_words_list = list(google_model.wv.vocab)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: Call
to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  """
```

Create a list of the unique set of words from this downsampled model

In [55]:

```
#your code here
random.seed(42)
unique_words = random.sample(all_words_list, 15000)
```

Extract respective vectors corresponding to the words in the down-sampled, pre-trained model

In [56]:

```
#your code here
vector_list = []
unique_words
for words in unique_words:
  vector_list.append(google_model[words])
```

Q7b

Run Kmeans clustering on the pre-trained word vectors. Make sure to use the word vectors from the pre-trained model.

In [57]:

```
#your code here
%matplotlib inline
from matplotlib import pyplot as plt

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

kmeans = KMeans(n_clusters=25, random_state=42)
X = np.array(vector_list)
kmeans.fit(X)
# silhouette_score(google_X, google_kmeans.labels_)
kmeans.labels_
```

Out[57]:

```
array([22, 16,  9, ..., 22, 19, 17], dtype=int32)
```

Q7c

Reduce the dimensionality of the word vectors from the pre-trained model using TSNE

In [58]:

```
#your code here
data_embed= TSNE(n_components=2, perplexity=50, verbose=2, method='barnes_hut').fit_tran
sform(vector_list)
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 15000 samples in 0.763s...
[t-SNE] Computed neighbors for 15000 samples in 198.629s...
[t-SNE] Computed conditional probabilities for sample 1000 / 15000
[t-SNE] Computed conditional probabilities for sample 2000 / 15000
[t-SNE] Computed conditional probabilities for sample 3000 / 15000
[t-SNE] Computed conditional probabilities for sample 4000 / 15000
[t-SNE] Computed conditional probabilities for sample 5000 / 15000
[t-SNE] Computed conditional probabilities for sample 6000 / 15000
[t-SNE] Computed conditional probabilities for sample 7000 / 15000
[t-SNE] Computed conditional probabilities for sample 8000 / 15000
[t-SNE] Computed conditional probabilities for sample 9000 / 15000
[t-SNE] Computed conditional probabilities for sample 10000 / 15000
[t-SNE] Computed conditional probabilities for sample 11000 / 15000
[t-SNE] Computed conditional probabilities for sample 12000 / 15000
[t-SNE] Computed conditional probabilities for sample 13000 / 15000
[t-SNE] Computed conditional probabilities for sample 14000 / 15000
[t-SNE] Computed conditional probabilities for sample 15000 / 15000
[t-SNE] Mean sigma: 0.259116
[t-SNE] Computed conditional probabilities in 1.556s
[t-SNE] Iteration 50: error = 93.4952316, gradient norm = 0.0965119 (50 iterations in 17.
```

```
010s)
[t-SNE] Iteration 100: error = 93.4972687, gradient norm = 0.0962155 (50 iterations in 17
.207s)
[t-SNE] Iteration 150: error = 94.1236191, gradient norm = 0.1152375 (50 iterations in 15
.465s)
[t-SNE] Iteration 200: error = 95.0899353, gradient norm = 0.0672304 (50 iterations in 14
.639s)
[t-SNE] Iteration 250: error = 94.7655563, gradient norm = 0.0924168 (50 iterations in 15
.905s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 94.765556
[t-SNE] Iteration 300: error = 4.2497177, gradient norm = 0.0039211 (50 iterations in 16.
853s)
[t-SNE] Iteration 350: error = 3.8171191, gradient norm = 0.0009142 (50 iterations in 11.
106s)
[t-SNE] Iteration 400: error = 3.6913888, gradient norm = 0.0003086 (50 iterations in 9.7
98s)
[t-SNE] Iteration 450: error = 3.6182642, gradient norm = 0.0002108 (50 iterations in 9.9
78s)
[t-SNE] Iteration 500: error = 3.5722270, gradient norm = 0.0001633 (50 iterations in 9.8
42s)
[t-SNE] Iteration 550: error = 3.5401177, gradient norm = 0.0001228 (50 iterations in 9.5
73s)
[t-SNE] Iteration 600: error = 3.5169125, gradient norm = 0.0000996 (50 iterations in 9.4
90s)
[t-SNE] Iteration 650: error = 3.4997029, gradient norm = 0.0000788 (50 iterations in 9.7
55s)
[t-SNE] Iteration 700: error = 3.4863548, gradient norm = 0.0000686 (50 iterations in 9.7
60s)
[t-SNE] Iteration 750: error = 3.4755781, gradient norm = 0.0000636 (50 iterations in 9.7
75s)
[t-SNE] Iteration 800: error = 3.4668152, gradient norm = 0.0000580 (50 iterations in 9.7
89s)
[t-SNE] Iteration 850: error = 3.4596264, gradient norm = 0.0000551 (50 iterations in 11.
245s)
[t-SNE] Iteration 900: error = 3.4533064, gradient norm = 0.0000521 (50 iterations in 9.6
00s)
[t-SNE] Iteration 950: error = 3.4481843, gradient norm = 0.0000474 (50 iterations in 9.6
03s)
[t-SNE] Iteration 1000: error = 3.4439068, gradient norm = 0.0000499 (50 iterations in 9.
440s)
[t-SNE] KL divergence after 1000 iterations: 3.443907
```

Q7d

Create a dataframe with the following columns using the pre-trained vectors and corpus:

Column	Description
x	the first dimension of result from TSNE
y	the second dimension of result from TSNE
Feature 1	the word corresponding to the vector
Feature 2	the kmeans cluster label

Below is a sample of what the dataframe could look like:

x	y	Feature 1	Feature 2
7.154159	9.251100	lips	8
-53.254147	-13.514915	them	9
34.049191	-13.402843	topic	0
-32.515320	28.699677	sofa	24
13.006302	-4.270626	half-past	21

In [59]:

```
#your code here
df = pd.DataFrame(data_embed[:,:2],columns=["x","y"])
df['Feature 1'] = unique_words
df['Feature 2'] = kmeans.labels_
df.head()
```

Out[59]:

	x	y	Feature 1	Feature 2
0	-0.039646	-15.943139	espresso_martinis	22
1	-33.333626	-16.178673	wherefore	16
2	-37.079254	-4.581667	HARD	9
3	4.519468	-17.936140	courtly_manners	16
4	-2.640751	-11.055662	Hawai'ian	22

Q7e: Visualization

In this question, you are required to visualize and explore the reduced dataset from the pretrained model you created in Q7d using the [d3-scatterplot] library.

Note: The first code-chunk at the top in this notebook clones the libary from github. Make sure that it has been executed before you proceed.

Save your dataset as a tsv file 'd3-scatterplot/google_mytext.tsv'

Example:

```
google_df.to_csv('d3-scatterplot/google_mytext.tsv', sep='\t', index=False)
```

Note 1: The TSV file needs to be stored in the `d3-scatterplot` folder so that the d3-scatterplot library can access it.

Note 2: Make sure to save the TSV file WITHOUT the row index i.e. use `index=False`.

In [62]:

```
#your code here
!git clone https://github.com/CAHLR/d3-scatterplot.git
from google.colab.output import eval_js
from IPython.display import Javascript

df.to_csv('d3-scatterplot/google_mytext.tsv', sep='\t', index=False)
```

```
fatal: destination path 'd3-scatterplot' already exists and is not an empty directory.
```

Visualize the reduced vectors by running the following code

In [63]:

```
def show_port(port, data_file, width=600, height=800):
  display(Javascript("""
  (async ()=>{
    fm = document.createElement('iframe')
    fm.src = await google.colab.kernel.proxyPort(%d) + '/index.html?dataset=%s'
    fm.width = '90%%'
    fm.height = '%d'
    fm.frameBorder = 0
    document.body.append(fm)
  })();
  """ % (port, data_file, height)))

port = 8001
```

```
data_file = 'google_mytext.tsv'
height = 1400

get_ipython().system_raw('cd d3-scatterplot && python3 -m http.server %d &' % port)
show_port(port, data_file, height)
```

Color the points by their kmeans cluster. You can do this by choosing "Feature 2" as the variable in the "Color" drop-down in the visualization.

Pleae include a snapshot of your visualization in the notebook. Refer to the tutorial video, and/or follow the steps below:

1) Take a sanpshot of the visualization and save it on your computer with the filename `google_trained_scatter.png`

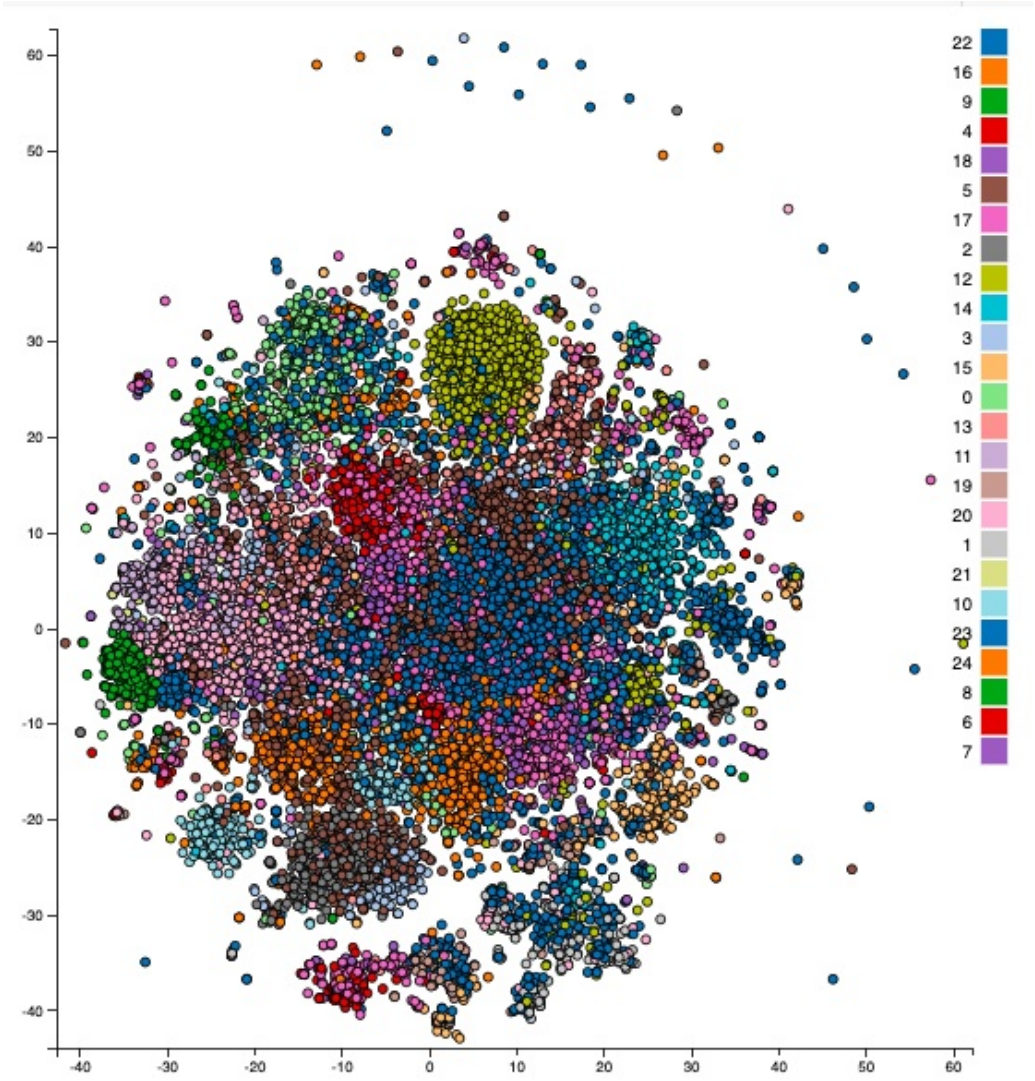
2) Upload the `google_trained_scatter.png` **by clicking on the 'Files' icon on the left sidebar and clicking on the upload icon (the one with an upward arrow on top left)**

3) Run the code below to display the image

In [64]:

```
#This is an autograded cell, do not edit/delete
Image('google_trained_scatter.png')
```

Out[64]:



Q8: Exploration (0.5 point)

This is an open-ended question.

On the visualizations in Q6 & Q7, lasso select a group of points with the left mouse button and look at summaries of the group on the right-side of the plot. (Refer to the tutorial video for a demo on the lasso selection). Also look at the words / features of the selected points.

Comment on any patterns / similarities you see in the selected words in the visualization for the pre-trained vectors and the vectors trained on your corpus. Are you able to find any group of points that are close to each other in the 2D space that also have semantic similarity?

I was able to find similarity in the group of points in cluster 24 from the first plot and cluster 23 from the second plot. The semantic similarity I noticed was that they both shared similar dark/spooky words such as ghost/swamp/darkness/etc. This made sense but I also wondered if the sleepy hollows text I chose to use as my corpus may have such an overwhleming number of dark and ominous words that I was bound to find some similarity with similar words in the google model.

In [ ]: