In [37]:
```
NAME = "Yinglu Deng"
```

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/ (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 [38]:
```
!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-neg

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

```
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-pa
ckages (3.6.0)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/
dist-packages (from gensim) (1.19.5)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dis
t-packages (from gensim) (1.15.0)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python
3.7/dist-packages (from gensim) (5.2.1)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/
dist-packages (from gensim) (1.4.1)
fatal: destination path 'd3-scatterplot' already exists and is not an emp
ty directory.
File ‘GoogleNews-vectors-negative300.bin.gz’ already there; not retrievin
g.

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[38]:
```
True
```

Q1 (0.25 points)

Download your text corpus. (A good place to start is the nltk corpus (http://www.nltk.org/nltk_data/) or the gutenburg project (https://www.gutenberg.org/))

In [39]:
```
#your code here
url = "https://www.gutenberg.org/cache/epub/66627/pg66627.txt" ## Your raw
```

In [40]:
```
#Save the raw text that you just downloaded in this variable
raw = requests.get(url).content.decode('utf8')
```

In [41]:
```
#This is an autograded cell, do not edit/delete
print(raw[:1000])
```

```
The Project Gutenberg eBook of The Cat's Paw, by Natalie Sumner
Lincoln

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world 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. If you are not located in the United States, you
will have to check the laws of the country where you are located before
using this eBook.

Title: The Cat's Paw

Author: Natalie Sumner Lincoln

Release Date: October 29, 2021 [eBook #66627]

Language: English

Produced by: D A Alexander and the Online Distributed Proofreading Team
             at https://www.pgdp.net (https://www.pgdp.net) (This file wa
s produced from images
             generously made available by The Internet Archive)

*** START OF THE PROJECT GUTENBERG EBOOK THE CAT'S PAW ***




  THE CAT'S PAW

  BY

  NATALIE SUMNER LINCOLN

  AUTH
```

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', 'toda
y'],
 ['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 [42]:
```
#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)

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

In [43]:
```
#Save the tokenized sentences as a list of list in this variable
tok_corp = [nltk.word_tokenize(sent) for sent in nopunct]
# tok_corp[:3]
```

In [44]:
```
#This is an autograded cell, do not edit/delete
for sent in tok_corp[:3]:
  print(sent)
  print("\n")
```

```
['The', 'Project', 'Gutenberg', 'eBook', 'of', 'The', 'Cat', "'s", 'Paw',
'by', 'Natalie', 'Sumner', 'Lincoln', 'This', 'eBook', 'is', 'for', 'th
e', 'use', 'of', 'anyone', 'anywhere', 'in', 'the', 'United', 'States',
'and', 'most', 'other', 'parts', 'of', 'the', 'world', 'at', 'no', 'cos
t', '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']
```

Q3 (0.25 points)

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

```
In [45]: #code here
         #The most important hyper parameters of skip-gram/CBOW are vector size and
         model = gensim.models.Word2Vec(tok_corp, min_count=1, size=16, window=5)
```

```
In [46]: #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: 64703
Corpus Count: 5430
Training time: 0.3813554789985574
Sample words: ['The', 'Project', 'Gutenberg', 'eBook', 'of', 'Cat', "'s",
'Paw', 'by', 'Natalie']
```

Q4 (0.25 points)

Q4a

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

Type *Markdown* and LaTeX: α^2

```
In [47]: #code here
         unique_words = list(set([item for sublist in tok_corp for item in sublist]))
```

```
In [48]: #This is an autograded cell, do not edit/delete
         print(unique_words[:10])
```

```
['Rock', 'paid', 'injured', 'stylishly', 'teas', 'Stop', 'smiled', 'seat
s', 'BEFORE', 'provided']
```

Q4b

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

```
In [49]: #code here
         vector_list = model[unique_words]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: Deprecati
onWarning: Call to deprecated `__getitem__` (Method will be removed in 4.
0.0, use self.wv.__getitem__() instead).
```

In [50]:
```
#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: (7061, 16)
Rock [ 0.16139488 -0.15973805 -0.11409998 -0.10724256 -0.01362273  0.0958
1959
 -0.494564   -0.1170098  -0.22369134  0.02046887 -0.04864922  0.24512705
 -0.03558644  0.14628929 -0.02810979  0.27081192]
paid [ 0.17050995 -0.14856452 -0.11536963 -0.133644   -0.01472265  0.0969
1042
 -0.5295     -0.11868399 -0.23586832  0.05519501 -0.06764337  0.2682868
 -0.07513689  0.14319512 -0.00899039  0.29930264]
injured [ 0.04746434 -0.06378891 -0.00318533 -0.06773484 -0.02403311  0.0
0722292
 -0.15119536 -0.0344775  -0.09370983  0.03828652 -0.04688099  0.05477981
 -0.03511737  0.02334767  0.0137507   0.08604767]
stylishly [-0.0116773  -0.00406978  0.02124605 -0.01844885 -0.02446531 -
0.01551978
 -0.06808238 -0.01459019 -0.0083939   0.00864854  0.01034458  0.00288554
 -0.01135399 -0.00687733 -0.02779859  0.04286049]
teas [ 0.04344679  0.005319   -0.01229751 -0.01811429 -0.02387852  0.0169
5198
 -0.07770319 -0.00177251 -0.0272573   0.02585335 -0.03253726  0.01691595
 -0.03221815  0.03446426  0.01682974  0.00995977]
```

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.

---Your hypotheses here---

- Hypotheses 1: "train" is (cosine) more similar to "jump" than to "eat". Because we always "train" the cat to "jump" over obstacle and jumpping is a very important skill in training and we don't need to train the cat how to "eat".
- Hypotheses 2: "happy" is (cosine) more similar to "play" than to "eat". Because cat is more "happy" when we "play" and interact with them. They are still "happy" when they "eat", but not as happy as "play" time.

- Hypotheses 3: "die" is (cosine) more similar to "old" than to "disease". Because most of the cats "die" becasue they are "old" instead of suffering from "disease".

In [51]:
```
#your code here for validating hypotheses 1
hy1 = model.similarity('train', 'jump')
hy1_0 = model.similarity('train', 'eat')
print(hy1, hy1_0)
```

```
0.9699781 0.6138697

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: Deprecati
onWarning: Call to deprecated `similarity` (Method will be removed in 4.
0.0, use self.wv.similarity() instead).

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: Deprecati
onWarning: Call to deprecated `similarity` (Method will be removed in 4.
0.0, use self.wv.similarity() instead).
  This is separate from the ipykernel package so we can avoid doing impor
ts until
```

In [52]:
```
#your code here for validating hypotheses 2
hy2 = model.similarity('happy', 'play')
hy2_0 = model.similarity('happy', 'eat')
print(hy2, hy2_0)
```

```
0.8911781 0.6496581

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: Deprecati
onWarning: Call to deprecated `similarity` (Method will be removed in 4.
0.0, use self.wv.similarity() instead).

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: Deprecati
onWarning: Call to deprecated `similarity` (Method will be removed in 4.
0.0, use self.wv.similarity() instead).
  This is separate from the ipykernel package so we can avoid doing impor
ts until
```

In [53]:
```
#your code here for validating hypotheses 3
hy3 = model.similarity('die', 'old')
hy3_0 = model.similarity('die', 'disease')
print(hy3, hy3_0)
```

```
0.9559558 0.46240282

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: Deprecati
onWarning: Call to deprecated `similarity` (Method will be removed in 4.
0.0, use self.wv.similarity() instead).

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: Deprecati
onWarning: Call to deprecated `similarity` (Method will be removed in 4.
0.0, use self.wv.similarity() instead).
  This is separate from the ipykernel package so we can avoid doing impor
ts until
```

- Hypotheses 1 is true, "train" is (cosine) more similar to "jump" than to "eat". The similarity of "train" and "jump"(0.9699781) is higher than the similarity of "train" and "eat"(0.6138697).
- Hypotheses 2 is true, "happy" is (cosine) more similar to "play" than to "eat". The similarity of "happy" and "play"(0.8911781) is higher than the similarity of "happy" and "eat"(0.6496581).
- Hypotheses 3 is true, "die" is (cosine) more similar to "old" than to "disease". The similarity of "die" and "old"(0.9559558) is higher than the similarity of "die" and "disease"(0.46240282).

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 [73]:
```
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[73]:
```
array([15, 15,  7, ..., 17, 14,  5], dtype=int32)
```

Q6b

Reduce the dimensionality of your word vectors using TSNE

In [74]:
```
#your code here
from sklearn.manifold import TSNE
# Lets dim reduce the 16 dimension vectors to 2 dimensions to vizualise the
data_embed=TSNE(n_components=2, perplexity=50, verbose=2, method='barnes_hu

## Parameters
## n_components = number of dimensions you want your data to be reduced
## preplexity =  Number of neighboours to fit the gaussian , normally 30
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7061 samples in 0.015s...
[t-SNE] Computed neighbors for 7061 samples in 1.288s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7061
[t-SNE] Computed conditional probabilities for sample 2000 / 7061
[t-SNE] Computed conditional probabilities for sample 3000 / 7061
[t-SNE] Computed conditional probabilities for sample 4000 / 7061
[t-SNE] Computed conditional probabilities for sample 5000 / 7061
[t-SNE] Computed conditional probabilities for sample 6000 / 7061
[t-SNE] Computed conditional probabilities for sample 7000 / 7061
[t-SNE] Computed conditional probabilities for sample 7061 / 7061
[t-SNE] Mean sigma: 0.028807
[t-SNE] Computed conditional probabilities in 0.874s
[t-SNE] Iteration 50: error = 85.4329910, gradient norm = 0.0194241 (50 i
terations in 5.105s)
[t-SNE] Iteration 100: error = 78.7243423, gradient norm = 0.0019867 (50
iterations in 3.856s)
[t-SNE] Iteration 150: error = 78.4143066, gradient norm = 0.0015123 (50
iterations in 3.026s)
[t-SNE] Iteration 200: error = 78.2994461, gradient norm = 0.0011356 (50
iterations in 2.883s)
[t-SNE] Iteration 250: error = 78.2423935, gradient norm = 0.0009434 (50
iterations in 2.907s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 78.24
2393
[t-SNE] Iteration 300: error = 2.8801711, gradient norm = 0.0010287 (50 i
terations in 3.452s)
[t-SNE] Iteration 350: error = 2.6453605, gradient norm = 0.0004152 (50 i
terations in 3.763s)
[t-SNE] Iteration 400: error = 2.5410547, gradient norm = 0.0002451 (50 i
terations in 3.797s)
[t-SNE] Iteration 450: error = 2.4799180, gradient norm = 0.0001720 (50 i
terations in 3.725s)
[t-SNE] Iteration 500: error = 2.4402244, gradient norm = 0.0001205 (50 i
terations in 3.753s)
[t-SNE] Iteration 550: error = 2.4119456, gradient norm = 0.0000955 (50 i
terations in 3.717s)
[t-SNE] Iteration 600: error = 2.3911748, gradient norm = 0.0000839 (50 i
terations in 3.750s)
[t-SNE] Iteration 650: error = 2.3758135, gradient norm = 0.0000719 (50 i
terations in 3.761s)
[t-SNE] Iteration 700: error = 2.3648424, gradient norm = 0.0000635 (50 i
terations in 3.656s)
[t-SNE] Iteration 750: error = 2.3568597, gradient norm = 0.0000591 (50 i
terations in 3.679s)
[t-SNE] Iteration 800: error = 2.3507657, gradient norm = 0.0000553 (50 i
terations in 3.706s)
[t-SNE] Iteration 850: error = 2.3464160, gradient norm = 0.0000523 (50 i
terations in 3.673s)
[t-SNE] Iteration 900: error = 2.3430285, gradient norm = 0.0000508 (50 i
terations in 3.625s)
[t-SNE] Iteration 950: error = 2.3402326, gradient norm = 0.0000481 (50 i
terations in 3.690s)
[t-SNE] Iteration 1000: error = 2.3376927, gradient norm = 0.0000462 (50
iterations in 3.628s)
[t-SNE] KL divergence after 1000 iterations: 2.337693
```

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 [78]:
```
#your code here
df = pd.DataFrame(data_embed[:,:2],columns=["x","y"])
df['Feature 1'] = unique_words
df['Feature 2'] = kmeans.labels_
df
```

Out[78]:

	x	y	Feature 1	Feature 2
0	25.634808	-32.443207	Rock	15
1	22.389730	-33.978619	paid	15
2	32.615982	26.025373	injured	7
3	-50.333862	9.747921	stylishly	17
4	2.418999	-0.389328	teas	0
...	...	...	...	...
7056	-4.495298	10.029402	speculative	0
7057	36.399742	-23.941256	carefully	10
7058	-24.275473	16.506107	Stepping	17
7059	5.082191	16.665243	guest	14
7060	42.434135	-24.690569	wall	5

7061 rows × 4 columns

Q6d: Visualization

In this question, you are required to visualize and explore the reduced dataset you created in Q6c using the d3-scatterplot (https://github.com/CAHLR/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 [76]:
```
#your code here
%matplotlib inline
from matplotlib import pyplot as plt
df.to_csv('d3-scatterplot/mytext.tsv', sep='\t', index=False)
```

Visualize the reduced vectors by running the following code

```
In [77]: 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
             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 &'
         show_port(port, data_file, height)
```

```
<IPython.core.display.Javascript object>
```

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:

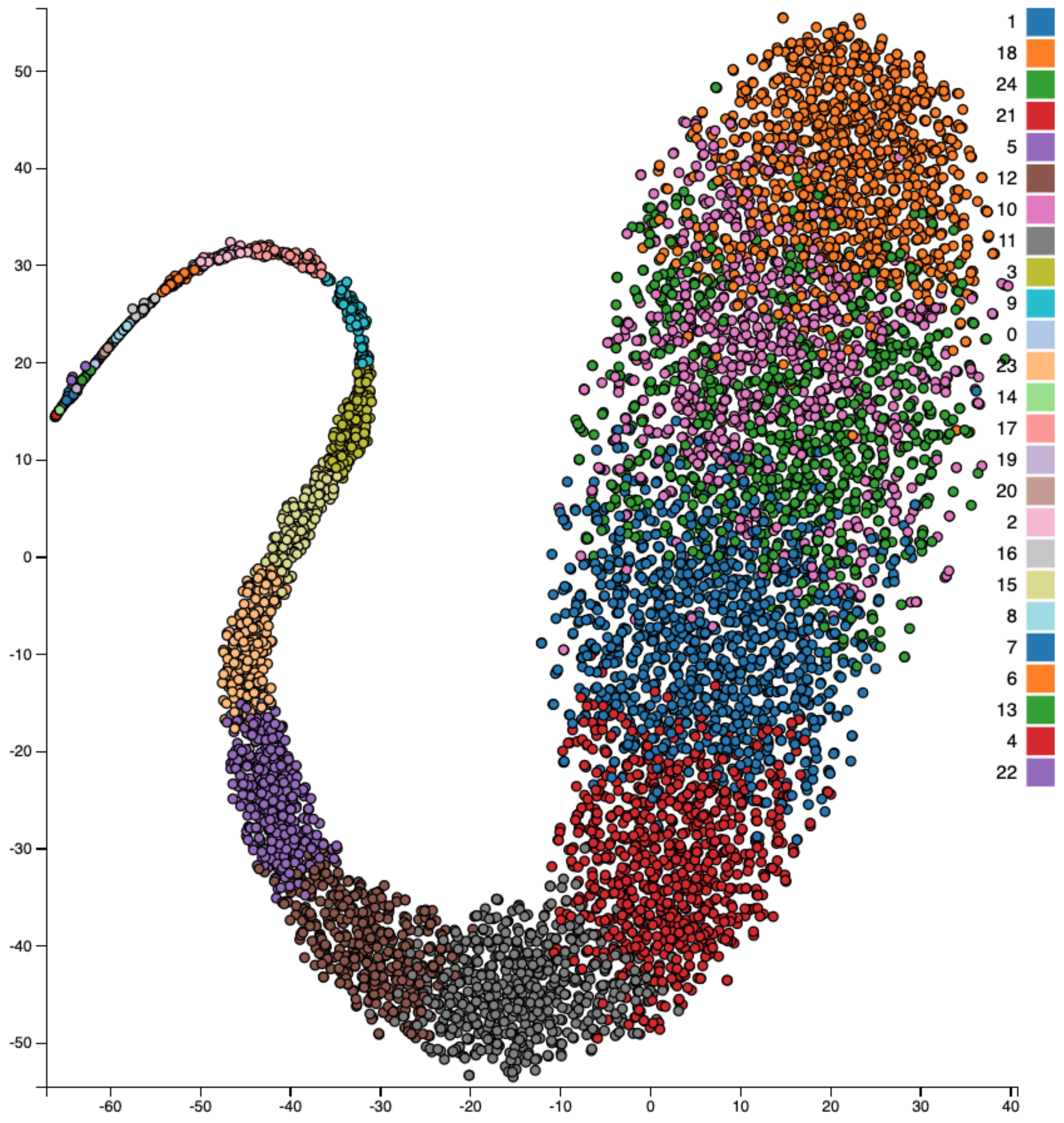
1) Take a sanpshot of the visualization and save it on your computer with the filename `trained_scatter.png`

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 [59]:
```
#This is an autograded cell, do not edit/delete
Image('trained_scatter.png')
```

Out[59]:



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 [60]:
```
#your code here
google_model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-
```

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

In [62]:
```
#your code here
import random
g_word = list(google_model.wv.vocab)

random.seed(42)
num = random.randint(10000, 25000)
word_selected = g_word[:num]  #selected 20k words
# word_selected
g_df = pd.DataFrame([google_model[w] for w in word_selected])
g_df['word'] = word_selected
g_df.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: Deprecati
onWarning: Call to deprecated `wv` (Attribute will be removed in 4.0.0, u
se self instead).
  This is separate from the ipykernel package so we can avoid doing impor
ts until
```

Out[62]:

	0	1	2	3	4	5	6	7	8
0	0.001129	-0.000896	0.000319	0.001534	0.001106	-0.001404	-0.000031	-0.000420	-0.000576
1	0.070312	0.086914	0.087891	0.062500	0.069336	-0.108887	-0.081543	-0.154297	0.020752
2	-0.011780	-0.047363	0.044678	0.063477	-0.018188	-0.063965	-0.001312	-0.072266	0.064453
3	-0.015747	-0.028320	0.083496	0.050293	-0.110352	0.031738	-0.014221	-0.089844	0.117676
4	0.007050	-0.073242	0.171875	0.022583	-0.132812	0.198242	0.112793	-0.107910	0.071777

5 rows × 301 columns

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

In [63]:
```
#your code here
g_unique_words = list(set(g_df['word']))
g_unique_words
```

```
 'Space',
 'Smart',
 'unified',
 'fabrication',
 'Arizona',
 'Pryor',
 'schemes',
 'facilitated',
 'behalf',
 'City',
 'dancers',
 'lieutenant',
 'architectural',
 'Bolton',
 'features',
 'currencies',
 'await',
 'sentence',
 'mediator',
 'reflection'.
```

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

In [64]:
```
#your code here
g_vector_list = google_model[g_unique_words]
g_vector_list
```

Out[64]:
```
array([[-0.22070312, -0.10693359, -0.21582031, ..., -0.26171875,
        -0.1015625 , -0.02331543],
       [ 0.03881836,  0.10498047,  0.01098633, ..., -0.12597656,
         0.10205078, -0.07177734],
       [ 0.0009079 ,  0.12011719,  0.02197266, ..., -0.00817871,
         0.00860596,  0.04760742],
       ...,
       [-0.08496094,  0.17480469,  0.01940918, ...,  0.20019531,
        -0.41992188,  0.09960938],
       [-0.20214844,  0.07324219,  0.01300049, ..., -0.04370117,
         0.09228516, -0.05737305],
       [ 0.29492188,  0.12792969,  0.18066406, ..., -0.11767578,
        -0.09326172,  0.14160156]], dtype=float32)
```

Q7b

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

In [65]:
```
#your code here
kmeans = KMeans(n_clusters=25, random_state=42)
X = np.array(g_vector_list)
kmeans.fit(X)
# silhouette_score(google_X, google_kmeans.labels_)
kmeans.labels_
```

Out[65]:
```
array([24, 20, 23, ...,  2, 14, 23], dtype=int32)
```

Q7c

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

In [66]:
```
#your code here
from sklearn.manifold import TSNE
# Lets dim reduce the 16 dimension vectors to 2 dimensions to vizualise the
g_data_embed=TSNE(n_components=2, perplexity=50, verbose=2, method='barnes_

## Parameters
## n_components = number of dimensions you want your data to be reduced
## preplexity =  Number of neighboourS to fit the gaussian , normally 30
```

```
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 20476 samples in 0.851s...
[t-SNE] Computed neighbors for 20476 samples in 295.716s...
[t-SNE] Computed conditional probabilities for sample 1000 / 20476
[t-SNE] Computed conditional probabilities for sample 2000 / 20476
[t-SNE] Computed conditional probabilities for sample 3000 / 20476
[t-SNE] Computed conditional probabilities for sample 4000 / 20476
[t-SNE] Computed conditional probabilities for sample 5000 / 20476
[t-SNE] Computed conditional probabilities for sample 6000 / 20476
[t-SNE] Computed conditional probabilities for sample 7000 / 20476
[t-SNE] Computed conditional probabilities for sample 8000 / 20476
[t-SNE] Computed conditional probabilities for sample 9000 / 20476
[t-SNE] Computed conditional probabilities for sample 10000 / 20476
[t-SNE] Computed conditional probabilities for sample 11000 / 20476
[t-SNE] Computed conditional probabilities for sample 12000 / 20476
[t-SNE] Computed conditional probabilities for sample 13000 / 20476
[t-SNE] Computed conditional probabilities for sample 14000 / 20476
[t-SNE] Computed conditional probabilities for sample 15000 / 20476
[t-SNE] Computed conditional probabilities for sample 16000 / 20476
[t-SNE] Computed conditional probabilities for sample 17000 / 20476
[t-SNE] Computed conditional probabilities for sample 18000 / 20476
[t-SNE] Computed conditional probabilities for sample 19000 / 20476
[t-SNE] Computed conditional probabilities for sample 20000 / 20476
[t-SNE] Computed conditional probabilities for sample 20476 / 20476
[t-SNE] Mean sigma: 0.994471
[t-SNE] Computed conditional probabilities in 1.807s
[t-SNE] Iteration 50: error = 102.7468262, gradient norm = 0.0826530 (50
iterations in 22.111s)
[t-SNE] Iteration 100: error = 103.1958618, gradient norm = 0.0766099 (50
iterations in 20.631s)
[t-SNE] Iteration 150: error = 103.6613617, gradient norm = 0.1109039 (50
iterations in 20.865s)
[t-SNE] Iteration 200: error = 104.1931000, gradient norm = 0.0816865 (50
iterations in 19.448s)
[t-SNE] Iteration 250: error = 103.7055206, gradient norm = 0.1001602 (50
iterations in 17.521s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 103.7
05521
[t-SNE] Iteration 300: error = 4.6408768, gradient norm = 0.0025113 (50 i
terations in 14.742s)
[t-SNE] Iteration 350: error = 4.0131359, gradient norm = 0.0016457 (50 i
terations in 14.951s)
[t-SNE] Iteration 400: error = 3.6668735, gradient norm = 0.0010666 (50 i
terations in 14.535s)
[t-SNE] Iteration 450: error = 3.5248189, gradient norm = 0.0002635 (50 i
terations in 13.929s)
[t-SNE] Iteration 500: error = 3.4199507, gradient norm = 0.0001994 (50 i
terations in 13.693s)
```

```
[t-SNE] Iteration 550: error = 3.3390403, gradient norm = 0.0001610 (50 i
terations in 13.440s)
[t-SNE] Iteration 600: error = 3.2755666, gradient norm = 0.0001341 (50 i
terations in 13.280s)
[t-SNE] Iteration 650: error = 3.2232065, gradient norm = 0.0001170 (50 i
terations in 13.383s)
[t-SNE] Iteration 700: error = 3.1793354, gradient norm = 0.0000993 (50 i
terations in 13.599s)
[t-SNE] Iteration 750: error = 3.1420145, gradient norm = 0.0000863 (50 i
terations in 13.679s)
[t-SNE] Iteration 800: error = 3.1098919, gradient norm = 0.0000762 (50 i
terations in 13.544s)
[t-SNE] Iteration 850: error = 3.0818801, gradient norm = 0.0000684 (50 i
terations in 13.471s)
[t-SNE] Iteration 900: error = 3.0572627, gradient norm = 0.0000622 (50 i
terations in 13.498s)
[t-SNE] Iteration 950: error = 3.0356407, gradient norm = 0.0000565 (50 i
terations in 13.628s)
[t-SNE] Iteration 1000: error = 3.0166764, gradient norm = 0.0000523 (50
iterations in 13.526s)
[t-SNE] KL divergence after 1000 iterations: 3.016676
```

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 [67]:
```
#your code here
google_df = pd.DataFrame(g_data_embed[:,:2],columns=["x","y"])
google_df['Feature 1'] = g_unique_words
google_df['Feature 2'] = kmeans.labels_
google_df.head()
```

Out[67]:

	x	y	Feature 1	Feature 2
0	11.618406	-38.218857	Rock	24
1	17.327204	29.614342	paid	20
2	-52.774097	25.333660	leg	23
3	-9.787192	58.046227	Stop	24
4	-28.509695	16.813747	seats	17

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 (https://github.com/CAHLR/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', inde
x=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 [68]:
```
#your code here
google_df.to_csv('d3-scatterplot/google_mytext.tsv', sep='\t', index=False)
```

Visualize the reduced vectors by running the following code

In [69]:
```
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
    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 &'
show_port(port, data_file, height)
```

```
<IPython.core.display.Javascript object>
```

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:

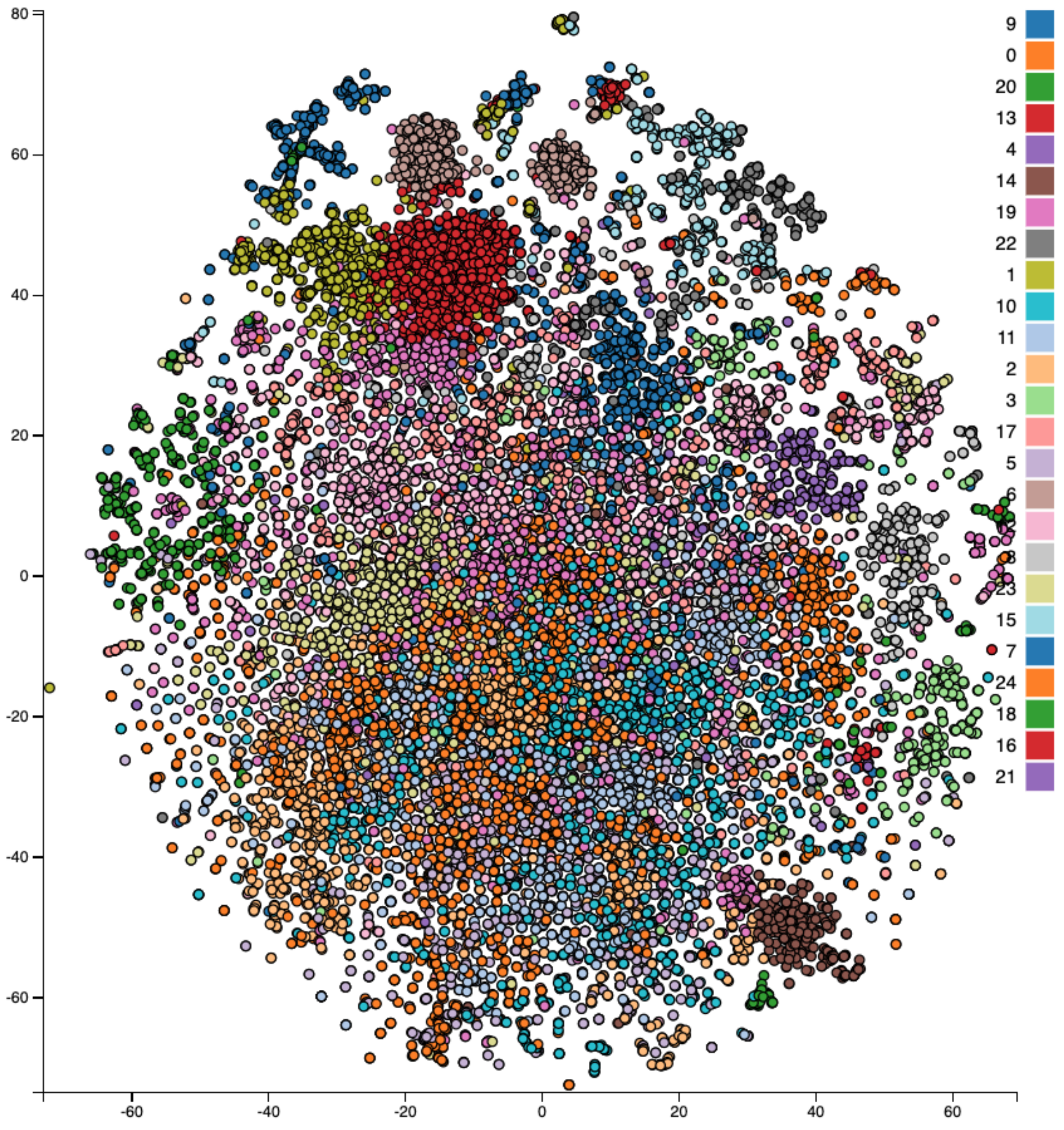
1) Take a sanpshot of the visualization and save it on your computer with the filename `google_trained_scatter.png`

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 [70]:
```
#This is an autograded cell, do not edit/delete
Image('google_trained_scatter.png')
```

Out[70]:



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?

--your answer here--

- In the vectors trained on your corpus, I selected label 13 and there are 484 words in label 13. Even though they have verb, norm, adverb, adjective or preposition in these words, they are all about cat training like "trust", "accept", "handshake", "waved", "tied", "repeat" and so on. For example, when training your cat, you need to build the "trust" between you and the cat. And using gesture like "waved" to train cats and "repeat" the process until they "accept" it and become muscle memory.
- In the pre-trained vectors, I selected label 3 and label4, because these two clusterings are close to each other. There are 908 words in label 3 and 529 words totally in label 4. I found the words are all about people name in label 3 and words are all location name in label 4. They are all names and have semantic similarity.

```
In [ ]:
```