Natural Language Processing Assignment 1 Report_Revised

20191138 Hyungyu Lee 이현규

Problem 0. Setup

Environment

Running environment for this assignment was Google Colab.

Choosing Word2Vec Model

```
your_model_code = 'glove-twitter-200'
```

I chose `glove-twitter-200` model. I worked with models that were trained with twitter dataset because I wanted to use words from lyrics of a specific artist, which are more colloquial, for problem 5. I used model with biggest dimension since I had time long enough to compute the model for this assignment. My initial motive was 'the bigger the better', but I believe throughout the course, I will be able to try models of different sizes and find out if it's true or not. Downloading the model definitely took some time, but we'll see if it's worth it. Running `model['cat']` returned a array with size of (200,), just like the dimension of the model, 200.

Problem 1. Simple Mathematics with Word2Vec

I had to complete those subtasks for this question:

- Write `word_analogy_with_vector(model, x_1, x_2, y_1)` function.
  - Input: `model, x_1, x_2, y_1` whereas `model` is a Word2Vec model, and `x_1, x_2, y_1` is string of word in `model`.
  - Output: a vector in np.ndarray, which can be used to find proper `y_2` ,like `most_similar()` function.
- Write `get_cosine_similarity(model, x, y)` function.
  - Input: `model, x, y` whereas `model` is a Word2Vec model, and `x` and `y` are string of words in `model`.
  - Output: float of similarity of x and y, value in range [-1, 1]

And as mentioned in question, I had to normalize each vector with L2 norm. These are the answers to my questions, and I commented on the code to facilitate understanding. I deleted code annotation provided, so that the report does not get unnecessarily lengthy.

1. word_analogy function

Code

```
def word_analogy_with_vector(model, x_1, x_2, y_1):
  # Write your code from here
  # so basically, we have to find a vector y_2, which is similar to
  # y_1 - x_2 + x_1

  #we declare three model array to do arithmetic operations
  x_1_model = model[x_1]
  x_2_model = model[x_2]
  y_1_model = model[y_1]

  # before doing this arithmetic, we have to L2 normalize vectors that correspond to
x_1, x_2, y_1
  x_1_model = x_1_model / np.linalg.norm(x_1_model)
  x_2_model = x_2_model / np.linalg.norm(x_2_model)
  y_1_model = y_1_model / np.linalg.norm(y_1_model)

  #then, we can simply return the code after arithmetic operation.
  resVec = y_1_model - x_1_model + x_2_model
  return resVec

# test whether the function works well
result_vector = word_analogy_with_vector(model, 'man', 'king', 'woman')
print(model.most_similar(positive=['woman', 'king'], negative=['man'], topn=10))
print('result vector is ', result_vector[:5])
assert isinstance(result_vector, np.ndarray), "Output of the function has to be
np.ndarray"
# model.most_similar(result_vector, topn=13)
model.most_similar(result_vector)
```

L2 Normalization - Errors, Solutions and Improvements

I tried to do L2 normalization on each words' vector with this formula like this:

```
  # before doing this arithmetic, we have to L2 normalize vectors that correspond to
x_1, x_2, y_1
  x_1_model /= np.sqrt(np.sum(x_1_model**2))
  x_2_model /= np.sqrt(np.sum(x_2_model**2))
  y_1_model /= np.sqrt(np.sum(y_1_model**2))
```

But this returned `ValueError: output array is read-only` because `numpy` arrays are immutable. So, x /= y doesn't work, while x = x / y works. I edited the code and it worked:

```
  # before doing this arithmetic, we have to L2 normalize vectors that correspond to
x_1, x_2, y_1
  x_1_model = x_1_model / np.sqrt(np.sum(x_1_model**2))
  x_2_model = x_2_model / np.sqrt(np.sum(x_2_model**2))
  y_1_model = y_1_model / np.sqrt(np.sum(y_1_model**2))
```

Also, I figured out that `numpy` provides L2 normalization function, `np.linalg.norm()`. So I substituted the mathematical equation with that function, making the code like the final submission form.

Results

With `glove-twitter-200` dataset the model answered that relation of man to king is similar to woman to queen, just like expected. Also, probability values that are provided by my `word_analogy_with_vector` function and `model.most_similar()` function could be declared as identical. There were difference whether query words are included or not, but the words and their probability were pretty much the same.

`result_vector` with `word_analogy_with_vector` function is as follows:

```
[('queen', 0.6820898056030273), ('prince', 0.5875527262687683), ('princess',
0.5620489120483398), ('royal', 0.5522865056991577), ('mother',
0.5362966656684875), ('elizabeth', 0.5142496228218079), ('lady',
0.5010437369346619), ('lion', 0.4998807907104492), ('women',
0.4985955059528351), ("'s", 0.4935073256492615)]
```

result with provided `model.most_similar()` function:

I used `topn` flag to see more words (I unbroke lines):

```
[('king', 0.739342212677002),  ('queen', 0.6820898056030273),  ('woman',
0.626846432685852),  ('prince', 0.5875527262687683),  ('princess',
0.5620488524436951),  ('royal', 0.5522865056991577),  ('mother',
0.5362966060638428),  ('elizabeth', 0.5142496228218079),  ('lady',
0.5010437369346619),  ('lion', 0.49988076090812683),
```

Queen was the most similar word to woman, just as expected with probability value of `0.6820898056030273.` Then words that are not query words, such as `prince, princess, royal, mother, elizabeth, lady, lion, women, 's`, appeared in both result vectors with almost identical values. Thus, it could be said that the `word_analogy_with_vector` was well written.

2. get_cosine_similarity function

Code

```
def get_cosine_similarity(model, x, y):
  # Write your codes from here
  #cosine similarity is dot product of two vectors divided by l2 norm of each vector
multiplied.
  x_vector = model[x]
  y_vector = model[y]

  #  return np.dot(x_vector, y_vector)/
(np.sqrt(np.sum(x_vector**2))*np.sqrt(np.sum(y_vector**2)))
  # this calculation at denominator can be simplified
  return np.dot(x_vector, y_vector)/
(np.linalg.norm(x_vector)*np.linalg.norm(y_vector))

# test the output with your own choice
word_a = 'good'
word_b = 'bad'

# these two should be identical
similarity = get_cosine_similarity(model, word_a, word_b)
print(similarity)
assert -1 <= similarity <= 1, "Similarity has to be between -1 and 1"

print('gensim library result:', model.similarity(word_a, word_b))
```

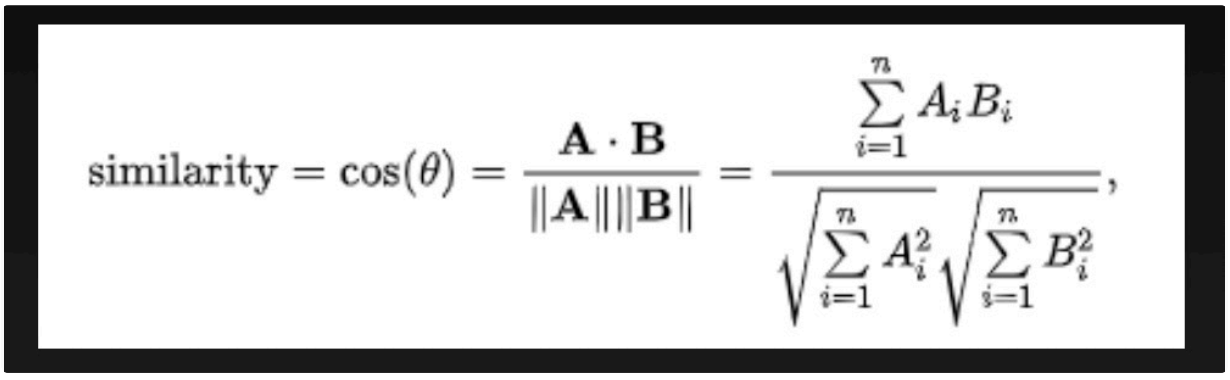
Result was as expected, with two output values being identical.

`0.7983508`

`gensim library result: 0.7983508`

Explanation

To calculate the cosine similarity of two vectors, we have to divide dot of two vectors by l2 norm of each vectors. Just like provided in the lecture slides:



Lecture note file nlp_24_2_word2vec.pdf, page 28

I calculated numerator, dot product of two vectors using `np.dot(A, B)`, and denominator, multiplied value of l2 sum of each vectors using command below:

```
np.sqrt(np.sum(a**2))*np.sqrt(np.sum(b**2))
```

But just like the first function, I figured out that this command could be simplified as:

```
(np.linalg.norm(x_vector)*np.linalg.norm(y_vector))
```

And voila! This function provides a same value as the `similarity` function from `gensim` library.

Problem 2. Find Most Similar Words

Altered Code

Instead of using one cell, or replicating 5 cells, I used array of words, and altered the provided code as follows:

```
target_words = ['diana','gaga', 'netflix', 'virus', 'cheonan'] # Enter your word
strings here

for target_word in target_words:
  # check the word is in the vocabulary of the model
  assert model.has_index_for(target_word), f"The selected word, {target_word}, is
not included in the model's vocabulary"
  pp.pprint(model.most_similar(target_word))
```

Word Choices

I chose diana, gaga, netflix, virus and cheonan to understand context of when, where and how Word2Vec model was trained, and to differentiate the result from my expectations.

Diana

Although a person's name is highly likely to be closely associated with other names, I thought that Princess Diana is such an well-known figure that could actually stand out from those relations. So in hopes of words like `royal, charles, elizabeth, british, wales` appearing, I used `model.most_similar('diana')`.

The results were not what I expected (I unbroke lines):

`[('laura', 0.6634964942932129), ('nadia', 0.6587172150611877), ('sara', 0.6520454287528992), ('lina', 0.6286117434501648), ('sarah', 0.6259469985961914), ('maria', 0.6209515333175659), ('daniela', 0.6194291710853577), ('tania', 0.6157795190811157), ('sabrina', 0.6114152073860168), ('andrea', 0.6097177267074585)]`

Other female names like Laura, Nadia, Sara, Lina, etc. appeared. I could infer that one person cannot be that influential to the name. I checked elizabeth as well to see if the Queen can be the exception and the result was `[('lynn', 0.6177338361740112), ... ('queen', 0.5807889103889465)]` having other female names, and the word queen at the end. Charles and Philip did not have any royal related words as similar words. But famous figure name that are not that usual, like barack and donald returned the words that connected to themselves (having president, trump in their list, respectively).

I could conclude that to privatize their first name, one has to be as popular and iconic as the Queen or have a very unique name.

Gaga

Being a baby monster(Lady Gaga's fan) since teenage, I was just curious if Lady will appear as reference to Gaga. The result was `[('lady', 0.7784555554389954), ('katy', 0.76102215051651), ('madonna', 0.7485153079032898), ('rihanna', 0.7212374806404114), ('miley', 0.7206928133964539), ('swift', 0.7128058671951294), ('selena', 0.7001330256462097), ('taylor', 0.6920899152755737), ('perry', 0.6823055148124695), ('britney', 0.6755322217941284)]` , having Lady as most similar words, and names of other pop starts as others.

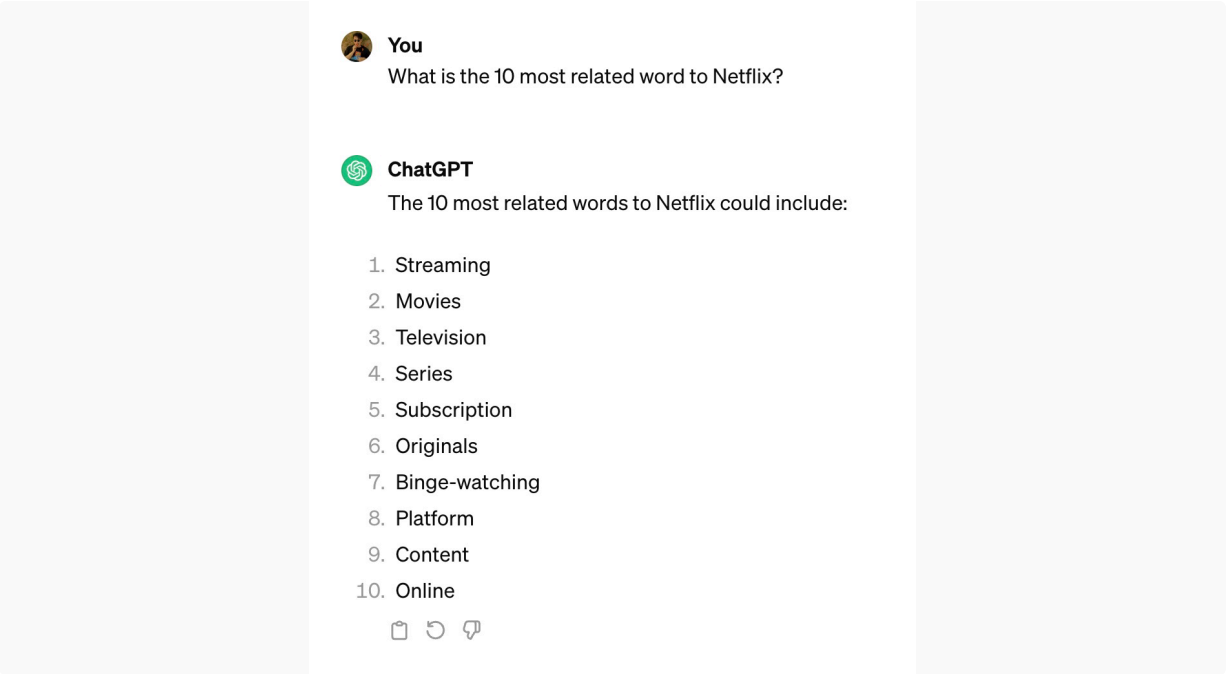
Lady had gaga as most related words, having p-value of 0.7784554958343506, too. I thought it can be caused from the fact that I used twitter dataset, which is packed with pop listeners. So I gave `glove-wiki-gigaword-50` a shot. I searched lady in `glove-wiki-gigaword-50` model, and the result was `('queen', 0.8050609827041626), ('mary', 0.7916390299797058), ('elizabeth', 0.7746644616127014), ('daughter', 0.761814534664154), ('mother', 0.7314392328262329), ('her', 0.7281726598739624), ('aunt', 0.7224655151367188), ('grandmother', 0.7196080088615417), ('wife', 0.7174438238143921), ('sister', 0.7173954248428345)]` which is not related to Lady Gaga at all.

Also, I tried searching diana in wiki dataset too, and it had `princess, elizabeth, camilla` in the result. I think by using these two words in two different datasets, I could see that how texts from twitter is more trendy, celebrity-focused, and how wikipedia texts are more historic or academic.

Netflix

Motive to choosing this word is trivial, but I think it might be a fun refreshment in this wordy report, and also the good analysis on scope of the model. I was playing netflix drama while waiting for the `model` to load. Then I randomly opened ChatGPT and asked:

- What are the 10 most related words to Netflix?



ChatGPT provided streaming, movies, television, series, subscription, originals, binge-watching, platform, content and online as related word, and figures that are actually in the movie/original series as the related entity.

And what the `glove-twitter-200` gave me was as follows:

```
[('movies', 0.6666310429573059), ('hbo', 0.6364792585372925), ('episodes', 0.6205011606216431), ('movie', 0.5996598601341248), ('watching', 0.5983713865280151), ('watch', 0.5762320160865784), ('channel', 0.5635140538215637), ('watched', 0.5465399622917175), ('dexter', 0.545410692691803), ('thrones', 0.5439996123313904)]
```

Some words overlapped, and other words, mainly the words from title/character from Netflix series, and competitors(HBO Max) appeared. I was interested that neither of them included more competitors, like Disney+ or movie theaters. If I ask a person to think of 10 words related to Netflix, I think verbs will appear less, and name of competitors will appear more. I think this shows a feature of model, as a natural language processor, that unlike humans, it can think of words without regards to part of speech.

Virus

I thought that checking if virus is related to COVID, COVID-19, Quarantine, etc. could help me find out if the model has data after 2020 or not. Also, I was interested in whether this model has racial bias or not, maybe connecting Asian words to virus or covid. I thought finding out words related to virus, or covid could help me figure this out.

The result for virus was as follows:

`[('malware', 0.642665445804596), ('hepatitis', 0.5642246603965759), ('flu', 0.5599080920219421), ('hacker', 0.5506444573402405), ('aids', 0.544348418712616), ('anti', 0.5252096056938171), ('hiv', 0.5232646465301514), ('influenza', 0.5220769643783569), ('bug', 0.49769023060798645), ('penyakit', 0.49434715509414673)]`

And I tried looking for covid, but it invoked `AssertionError: The selected word, covid, is not included in the model's vocabulary`. I could thus refer that this model was built before 2020 Spring. And I checked and confirmed that glove-twitter-200 was built in 2015 from their project website (https://nlp.stanford.edu/projects/glove/). If this model was built nowadays, I expect that word related to computer virus will be less related, and what happened following COVID-19, like vaccine, quarantine, test, etc. would appear more.

Cheonan

I live alone in Seoul, but my family lives in Cheonan. Cheonan is a city in Chungcheongnam-do(Chungnam), which is near megacity Daejeon. Also, it is well known for its Sundae Gukbap boiling soup, Independence Hall, and walnut cake for Koreans. I was wondering if this dataset had these knowledges as well, or just would say that random Korean region names as most similar.

This is what the `glove-twitter-200` gave me:

`[('gwangju', 0.45354315638542175), ('ilsan', 0.41581812500953674), ('gyeongju', 0.4149320721626282), ('s.m.art', 0.4062999188899994), ('busan', 0.4000849425792694), ('daegu', 0.3900299072265625), ('geondae', 0.38499194383621216), ('gs&c', 0.38424697518348694), ('nanjing', 0.38299983739852905), ('yri', 0.3812805116176605)]`

Gwangju, Ilsan, Gyeongju, Busan, Daegu is a Korean city name, Geondae is a town in Seoul, s.m.art is previous slogan of Asan, a city next to Cheonan, and gs&c, nanjing, yri is rather unrelated. I thought that these lacked actual relations of Korean region name.

Thus, I could conclude that since `glove-twitter-200` is based on English speaker's point-of-view, and that we need to consider linguistic/cultural barrier when it comes to NLP. This fact, that public AI models or datasets are often in English-centered point-of-view, is something I strive to overcome.

Problem 3. Word Analogy

Word Choices

I chose 5 words that represented my current status and interests. I am a senior student who has to prepare for post-graduation life, and start looking for jobs. I like going to clubs and travel abroad, so chose words that reflect those.

My 5 target words are: alumni, employed, dance, airplane, passport

For each words, x_1 and x_2 are the words that have certain relationship, which can be related to y_1 and y_2, which will be determined by the model. I altered code to run it in `for` loop.

```
def analogy(model, x1, x2, y1):
  pp.pprint(model.most_similar([x2, y1], negative=[x1]))
  print()

# Try with your own word choice
# analogy(model, 'man', 'king', 'woman')

my_word_analogy_pairs = [
    ['student', 'school', 'alumni'],
    ['unemployed', 'recession', 'employed'],
    ['music', 'concert', 'dance'],
    ['train', 'station', 'airplane'],
    ['id', 'domestic', 'passport']
]

for x1, x2, y1 in my_word_analogy_pairs:
  analogy(model, x1, x2, y1)
```

Word 1 - alumni as in relation of student and school

Expectations - jobs, Homecoming, Company

Student goes to school. So I expected that alumni will go to jobs, grad school, or commute to their jobs. Maybe they could come back for homecoming.

Result

```
[('sd', 0.5498698353767395),
 ('osis', 0.5004357695579529),
 ('highschool', 0.49871546030044556),
 ('reunian', 0.4949413537979126),
 ('smp', 0.4894418716430664),
 ('tomorrow', 0.48640692234039307),
 ('reuni', 0.48418116569519043),
 ('boys', 0.48372092843055725),
 ('graduation', 0.48320072889328003),
 ('shs', 0.48291248083114624)]
```

Explanation

I think of this analogy as a failure. sd seems to be a word for retaken exams, osis is an online student information system in university. Some alumni-related words like reunian, reuni appears in the list, but the overall result is not what I expected. I think this is because aluimni is more considered as part of school administration, rather than a status opposed to student. Maybe the word alumni is used only in resumes, so they are not usually in a very natural, informal writing.

Word 2 - employed as in unemployed and recession

Expectations - boom, expending, prosperity

If there is a recession, unemployment increases. Seemingly, I imagined that employment will be related to boom, expending, prosperity or such economic terms that mean positive trend.

Result

```
[('downturn', 0.4404403865337372),
 ('triple-dip', 0.4340003728866577),
 ('economic', 0.4171305298805237),
 ('explained', 0.4051300287246704),
 ('inflation', 0.4030877351760864),
 ('gdp', 0.40111812949180603),
 ('economy', 0.40024855732917786),
 ('slump', 0.3988068699836731),
 ('caused', 0.3962199091911316),
 ('slowdown', 0.3859215974807739)]
```

Explanation

downturn, triple-dip is all terms that are related to recession. triple-dip means a situation which an economy goes to recession, then begins to get better but goes back into recession again. I think the words in this analogy is rather negative or neutral because employed can be connected to employment, which can easily be related to negative statistics. Maybe looking at boom in terms of recession-unemployment could have been more convincing.

Word 3 - dance as in music and concert

Expectations - club, stage

concert is an event in which many people listen to music, so I expected that putting dance in this relationship will provide a word where many people dance.

Result

```
[('stage', 0.6779358386993408),
 ('tour', 0.5775321125984192),
 ('perform', 0.5644304156303406),
 ('flashmob', 0.5615183711051941),
 ('dancing', 0.5590847730636597),
 ('performing', 0.5416744351387024),
 ('tickets', 0.5249868631362915),
 ('concerts', 0.5237776041030884),
 ('performance', 0.5086872577667236),
 ('rehearsal', 0.5066430568695068)]
```

Explanation

Result was as expected. stage, tour, perform, flashmob, etc is all related to dancing and performance. I think using simpler words, or words that have rather one meaning yields more intuitive results, which will be the reason why we have to study NLP further.

Word 4 - airplane as in train and station

Expectations - airport, concourse, terminal

station is a place where we take trains. Same wise, I expect the place related to airplanes to appear.

Result

```
[('remote', 0.47134241461753845),
 ('plane', 0.45174673199653625),
 ('docking', 0.43280714750289917),
 ('helicopter', 0.4240418076515198),
 ('stations', 0.42059046030044556),
 ('floating', 0.41908758878707886),
 ('airline', 0.41419467329978943),
 ('space', 0.41104838252067566),
 ('landing', 0.4090425372123718),
 ('flight', 0.404619425535202)]
```

Explanation

Although the result was not exactly the same, some words that I expected appeared. existence of remote is questionable, though.

Word 5 - passport as in id and domestic

Expectations - international, worldwide

id (identity card) works in a country, but a passport is worldwide. I expected synonyms to international to appear.

Result

```
[('diplomatic', 0.44091305136680603),
 ('aircraft', 0.423111230134964),
 ('migrant', 0.4226188659667969),
 ('aboriginal', 0.4128878712654114),
 ('wildlife', 0.4117456078529358),
 ('passports', 0.4108889102935791),
 ('tourism', 0.410648912191391),
 ('transporting', 0.4051721394062042),
 ('airline', 0.4012957811355591),
 ('deportation', 0.3999738097190857)]
```

Explanation

The result did not directly convey the meaning of international or wide. I think this is because the term id can be interpreted too broadly. If I use this relationship in Korean, which has a singular word for domestic ID (신분증, 민증, 주민등록증, 면허증), and look 여권(passport) as in (신분증, 민증, 주민등록증, 면허증 - 국내) (domestic ID - domestic), maybe some words that are related to 국제, 해외 (international, abroad) might appear. Instead, this result is in general about migration. I think this shows how different language structure for a word can affect NLP result, and why we want to further develop our technology to make a NLP model that can consider several words at once, which will take much more computing power.

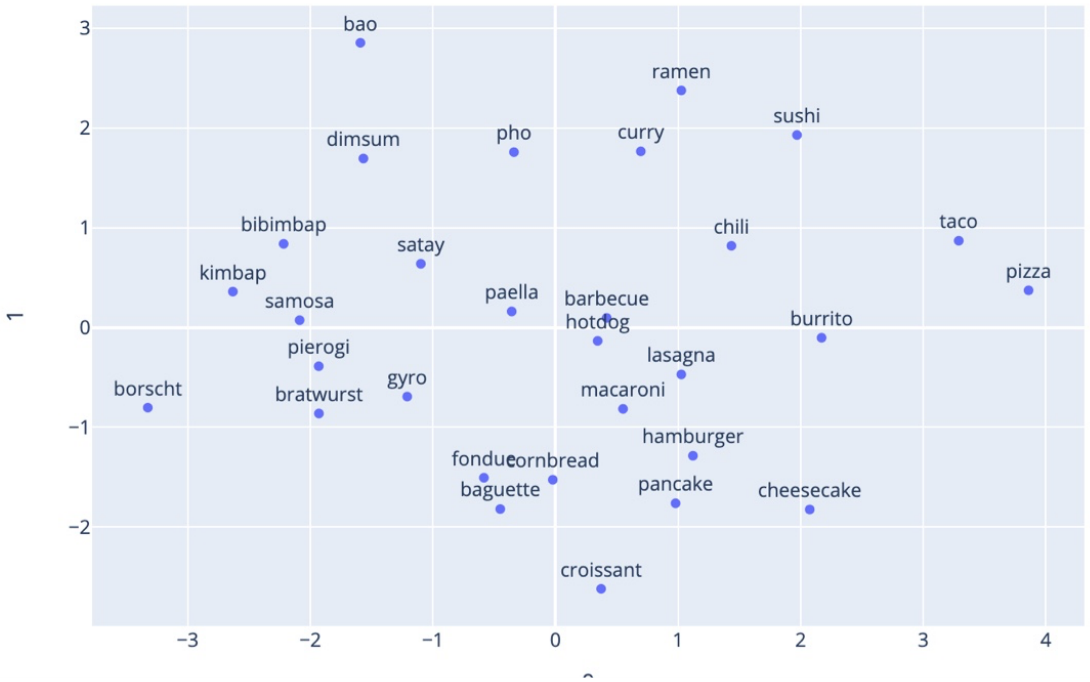
Problem 4. Visualize Word Vectors

Word List

I used the name of the food I tried in 2023. I wondered if food will be plotted according to the country, or the ingredients.

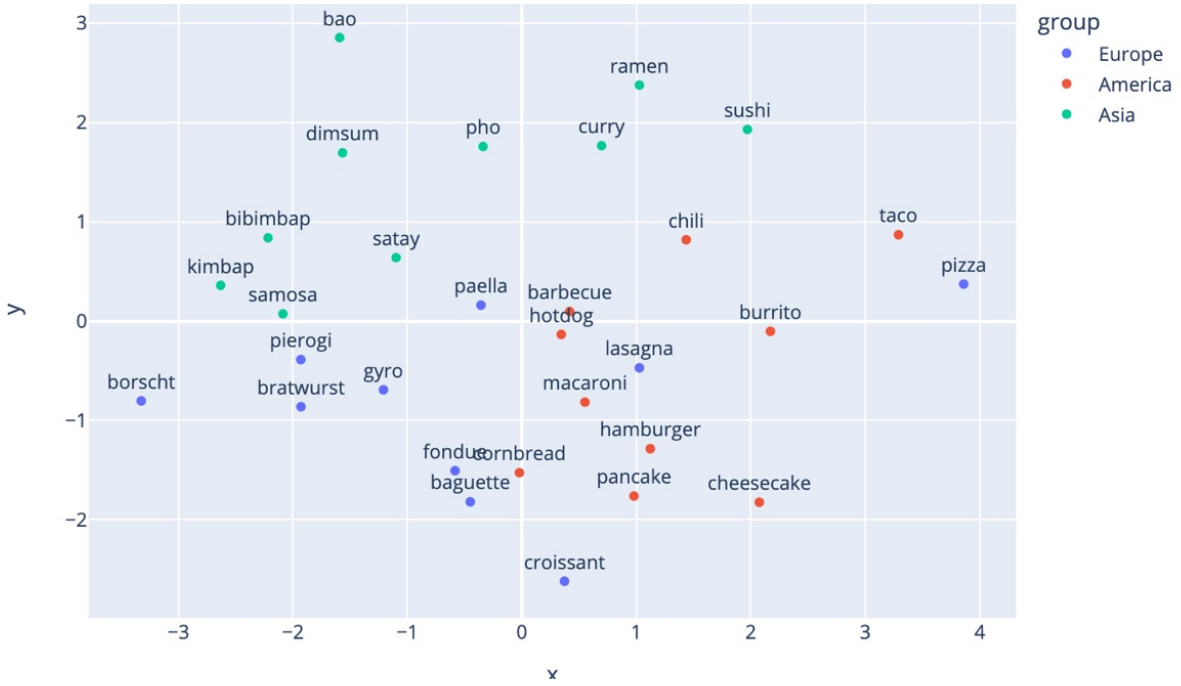
```
# Select word list of your own interests
word_list = [
            "baguette", "pizza", "pierogi", "croissant", "paella",
             "borscht", "gyro", "lasagna", "bratwurst", "fondue",
             "hamburger", "hotdog", "taco", "burrito", "barbecue",
             "chili", "macaroni", "pancake", "cornbread", "cheesecake",
             "sushi", "ramen", "curry", "bibimbap", "pho",
             "bao", "satay", "samosa", "dimsum", "kimbap"
]
display_pca_scatterplot(model, word_list)
```

Visualization



I modified the `display_pca_scatterplot` code so that food from different regions are colored distinctly, then added annotations to understand it.

```python
# Run this cell to
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import plotly.express as px

#altered code
def display_pca_scatterplot(model, words=None, sample=0):
  if len(words) < 30:
    print("WARNING: For your report, please select more than 30 word samples for the visualization")
    print(f"Current length of input word list: {len(words)}")

  word_vectors = np.array([model[w] for w in words])
  twodim = PCA().fit_transform(word_vectors)[:,:2]

  # Separate words into groups based on their source (Europe, America, Asia)
  european_words = words[:10]
  american_words = words[10:20]
  asian_words = words[20:]

  # Create a DataFrame for visualization with Plotly
  import pandas as pd
  df = pd.DataFrame(twodim, columns=['x', 'y'])
  df['word'] = words
  df['group'] = ['Europe'] * 10 + ['America'] * 10 + ['Asia'] * 10 #assign groups

  # Plot the scatterplot with different colors for each group
  fig = px.scatter(df, x='x', y='y', text='word', color='group')
  fig.update_traces(textposition='top center')
  fig.show()
```



Explanation - Clusterization , Evaluation and Surprises

Asian foods are located slightly apart from European and American foods. Then, European foods are mostly located in lower-left quadrant, and American foods are located in right quadrants. The exceptionally overlapping foods are lasagna and pizza. I think although I put those as European foods, they can also be considered as American foods, too. So that being said, more authentic(not in a sense of quality, but in terms of being more original, indigenous) foods are more well clustered according to the region.

Also, most words that share similar ingredients are rather close to each other. [taco, pizza] are basically made of same ingredients(flour, veggies, sauce, toppings) in a different way. [cornbread, baguette, croissant] are all breads, [hamburger, pancake, cheesecake] are all stacked form of buns/breads. [samosa and pierogi] are made of similar ingredients, too. But I was surprised that bao and dimsum is not that close, even further than pierogi and bratwurst(a german sausage). Two Korean foods are rather close to each other, but I expected they would be close to paella, since they share main ingredient, rice.

Thus, I could conclude that the relationships of food name words are rather more influenced from region, more than ingredients.

Problem 5. Train New Word2Vec

I used Lyrics from Taylor Swift's album data, which can be acquired from Kaggle (https://www.kaggle.com/datasets/ishikajohari/taylor-swift-all-lyrics-30-albums). I created a model with lyric text data only, and did analogy of some words that are considered significant for Taylor Swift. I compared it with `glove-twitter-200` model to see what kind of messages Taylor make in such issues.

Importing Files to Train the Model

In this file, each album had its own folder, and each song were saved in separate files. A file looks like this:

```
99 ContributorsTranslationsEspañolFrançaisPortuguês中文
СрпскиMagyarItalianoУкраїнськаTürkçeNederlandsSvenskaDeutschMagyarΕλληνικάLavender
Haze Lyrics[Intro]
Meet me at midnight

[Verse 1]
Staring at the ceiling with you
Oh, you don't ever say too much
And you don't really read into
My melancholia

[Pre-Chorus]
I been under scrutiny (Yeah, oh, yeah)
You handle it beautifully (Yeah, oh, yeah)

...

[Outro]
Get it off your chest
Get it off my desk
That lavender haze
I just wanna stay
I just wanna stay in that lavender haze35Embed
```

I mounted Google Drive rather than uploading those files, so that I do not have to reupload the files whenever `colab` runtime is over.

```
#multiple file -> use google drive, then browse to the correct directory

from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/ColabData/TaylorSwiftAlbums
```

Then, I imported the files with modified version of the code provided.

```
#text importer with modifications - different txt file for different names

#Modification that should be made
#0. use Google Drive - cell above
#1. multiple file structure: Albums/{album_name}/{song.txt} - for loop
#2. first line of the file should be ignored - in append function
#3. numbers, word that start with [ should be eliminated
#4. delete nnembed. first check if the numbers are all two digits.
#   then do appropriate tasks

strings=[]

from glob import glob
from tqdm import tqdm

albums = glob("/content/drive/MyDrive/ColabData/TaylorSwiftAlbums/*")
for album in tqdm(albums):
  songs = glob(str(album)+"/*")
  for song in songs:
    with open(song, 'r') as f:
      song_lyric = f.readlines()[1:] #ignore first line
      strings.extend([x.lower() for x in song_lyric if x[0] !="["])
      #delete dividers like [chorus], make it lowercase
    #print(strings[-1][-10:]) #last nnembed
```

I checked the `nnembed` forms using print in line 24. I figured out that there were all different, with having 1 to 3 digits, mainly two, followed by string embed. But there were cases in which `nnembed` did not exist. so I modified the code again using `re` module, which helps me use regular expressions.

```
...

for album in tqdm(albums):
  songs = glob(str(album)+"/*")
  for song in songs:
    with open(song, 'r') as f:
      song_lyric = f.readlines()[1:] #ignore first line
      if song_lyric:
        strings.extend([x.lower() for x in song_lyric[:-1] if x[0] !="["])#ignore
last line
        #delete dividers like [chorus]then add lowercased version
        #dividers are not in the last line, so we can use the expression below

        #taylor rarely puts numbers in numeric form, so deletion of number followed
by string in only last line
        #does not negatively affect

        #this line deletes ##embed
        last_line_modified = re.sub(r'\d+\s*\w+', '', song_lyric[-1].lower())
        #this line deletes embed, without numbers beforehand
        last_line_modified = last_line_modified.replace('embed','')
        strings.append(last_line_modified)
    # break #used to check
    # print(strings[-1]) #used to check
  # break
pp.pprint(strings)
```

I had to use `append` instead of `extend` for the last line, because it yielded error of each letter being saved apart. Through print at line 35, I could see that the issue, and `nnembed` removal is both well resolved. Then, I used text formatting function provided, after removing quotation marks. Then I checked 100, instead of 5 sentences to see check multiple songs. They turned out well, with 35786 sentences (there will be duplicates, because different editions of albums or lyrics are counted multiple times).

```
...
Sentence 96: but darling, now you have to
Sentence 97: if you're anything like me, there's a justice system in your head for
names you'll never speak again, and you make your ruthless rulings
Sentence 98: each new enemy turns to steel they become the bars that confine you, in
your own little golden prison cell..
Sentence 99: but darling, there is where you meet yourself
Sentence 100:  if you're anything like me you've grown to hate your pride to love
your thighs and no amount of friends at 25 will fill the empty seats at the lunch
tables of your past the teams that picked you last..
```

Training the model

```
taylor_model = Word2Vec(sentences=corpus, vector_size=80, window=3, min_count=6,
sg=1)
taylor_model = taylor_model.wv # To match with previous codes, we use wv
(KeyedVector) of the Word2Vec class
# Try the function above with the newely trained model
```

Arguments

I decreased `vector_size` because this dataset was rather small, having only 11629 unique words. I used code below to calculate number of unique words.

```
# words = set(corpus)
unique_word = set()
for x in corpus:
  for y in x:
    unique_word.add(y)
print(len(unique_word)) #num of unique words
```

I used `min_count` of 6 because a significant word will appear at least 6 times (an important word will appear at least 4~5 times in a song, since there are two verse and chorus, so words that appear more than 6 times are significant). `Window_size` is set to 3 because each lyric lines are rather short compared to modern-day writing. I used skip-gram because my dataset is rather small, and there can be some rare words. Also, lyrics are more semantic than syntactic. So I trained the model with Skip-gram. I made `taylor_model` variable so that we can compare the results from glove-twitter model and `taylor_model` easily.

Trying Functions

Similar Words

I looked for these three words using this code. Upper 10 are results from `taylor_model`, and lower 10 are from `model`.

```
target_words = ['woman', 'love', 'friend']
for target_word in target_words:
  print(target_word)
  pp.pprint(taylor_model.most_similar(target_word))
  pp.pprint(model.most_similar(target_word))
```

woman

```
[('likes', 0.8789421319961548),
 ('fellow', 0.8760899305343628),
 ('safe', 0.8748647570610046),
 ('marrying', 0.8726508617401123),
 ('sinks', 0.8682652115821838),
 ('masterpiece', 0.8681350350379944),
 ('second', 0.8645243048667908),
 ('ere', 0.8632925152778625),
 ('realizing', 0.8592833876609802),
 ('means', 0.8580912351608276)]

[('girl', 0.7817050218582153),
 ('women', 0.7705847024917603),
 ('guy', 0.7154314517974854),
 ('she', 0.710436224937439),
 ('person', 0.703464686870575),
 ('wife', 0.7029582262039185),
 ('female', 0.7000528573989868),
 ('mother', 0.6994999647140503),
 ('lady', 0.6945760846138),
 ('who', 0.6705518960952759)]
```

I consider Taylor's music to be more empowering and supporting woman. So I expected words that represent power, ability, etc. could be more prevalent than in ordinary twitter model. And words like safe, fellow, sisters, masterpiece, safe, won appears, supporting my expectation. Comparing with the twitter model, shows how the word 'woman' is related to positive adjectives or describing words, instead of other nouns. I think this is a phenomenon that happened because I am comparing lyrics to ordinary writing.

love

```
[('spiral', 0.7282082438468933),
 ('uhoh', 0.6988125443458557),
 ('fallin', 0.6968759298324585),
 ('affair', 0.6765228509902954),
 ('tragic', 0.6732059717178345),
 ('nobodys', 0.6559011936187744),
 ('desperately', 0.6525399684906006),
 ('insane', 0.6440119743347168),
 ('worship', 0.643915057182312),
 ('true', 0.6437399387359619)]

[('you', 0.8460860252380371),
 ('much', 0.7890045642852783),
 ('always', 0.7601684331893921),
 ('know', 0.7598055005073547),
 ('my', 0.7519949674606323),
 ('and', 0.7513089776039124),
 ('loves', 0.7512385249137878),
 ('life', 0.7443934082984924),
 ('it', 0.7426839470863342),
 ("n't", 0.7408116459846497)]
```

I'm actually surprised to see n't in glove-twitter `model`. `Taylor_model` returned word that are well connected to love, but they are not all positive. Fallin, tragic, spiral, insane, nobodys, despeartely can be used to portray both positive and negative meanings. This might prove that this model has acquired enough data to talk about the word love.

friend

```
[('criminal', 0.8674241900444031),
 ('hadnt', 0.8670597076416016),
 ('twice', 0.854934811592102),
 ('picking', 0.8546170592308044),
 ('conversation', 0.8532916903495789),
 ('such', 0.8524346351623535),
 ('lifetime', 0.8450250625610352),
 ('kept', 0.8415279388427734),
 ('risk', 0.8404600620269775),
 ('missed', 0.8388441205024719)]

[('friends', 0.8503412008285522),
 ('sister', 0.8243557214736938),
 ('bestfriend', 0.8121258020401001),
 ('brother', 0.7812201380729675),
 ('girl', 0.7702170014381409),
 ('you', 0.7522699236869812),
 ('dad', 0.751276433467865),
 ('bff', 0.7511802911758423),
 ('guy', 0.7429671883583069),
 ('mom', 0.7414224743843079)]
```

I expected words like friendship, bracelet (friendship bracelet is a huge thing for Taylor Swift fans), or name of friends to appear. But more of negative words, like criminal, risk appeared. This can be connected to some of her songs regarding broken friendship, like *Bad Blood*.

So far, the results are showing that choosing a personal record or lyrics for model can be a hard choice. Also, colloquial, musical abbreviations like you've, 'is appear in some of the queries, which are not included in this model. I think I should find out ways to practice on processing them as well.

Analogy to a relation

woman - man, he

```
analogy(taylor_model, 'man', 'he', 'woman')
analogy(model, 'man', 'he', 'woman')
```

The pronouns would have appeared a lot of times in the dataset. So I wondered if this model will be able to pass simple analogy test, which is similar to man, king, woman test we did before. But I thought that the dataset would not have much occurrence of king and queen, so I used he instead.

```
[('chasing', 0.7183302640914917),
 ('fame', 0.708710253238678),
 ('hasnt', 0.6730809211730957),
 ('midnight', 0.6669391393661499),
 ('teatime', 0.6615288853645325),
 ('changed', 0.6504514217376709),
 ('agrees', 0.6459696888923645),
 ('stayed', 0.6376455426216125),
 ('trimalchio', 0.6363664269447327),
 ('calls', 0.6356037855148315)]

[('she', 0.7365854382514954),
 ('has', 0.6785745620727539),
 ('thinks', 0.6429623365402222),
 ('knows', 0.6414571404457092),
 ('says', 0.6331944465637207),
 ('told', 0.6292889714241028),
 ('tells', 0.6237863898277283),
 ('does', 0.623668909072876),
 ('said', 0.6230323314666748),
 ('once', 0.6061710119247437)]
```

`taylor_model` did not return she as the most related words. I think this shows the limitation of the model. If only Taylor Swift had more songs, I think she will naturally be connected to woman in terms of man and he.

beautiful - good, bad

```
analogy(taylor_model, 'good', 'bad', 'beautiful')
analogy(model, 'good', 'bad', 'beautiful')
```

I expected negative words, such as ugly, miserable, desperate to appear.

```
[('tragic', 0.6374070644378662),
 ('rush', 0.6065912842750549),
 ('fuckin', 0.5918627381324768),
 ('perfect', 0.5720840692520142),
 ('peaks', 0.5548309087753296),
 ('pretty', 0.55328369140625),
 ('million', 0.5497406721115112),
 ('weird', 0.549546480178833),
 ('whos', 0.5354764461517334),
 ('motion', 0.5340595841407776)]

[('girl', 0.7060506939888),
 ('gorgeous', 0.6973251700401306),
 ('pretty', 0.6896135807037354),
 ('ugly', 0.6653832793235779),
 ('perfect', 0.6541532874107361),
 ('crazy', 0.6417362093925476),
 ('such', 0.6366500854492188),
 ('look', 0.6262451410293579),
 ('cute', 0.6249862909317017),
 ('stunning', 0.6183186173439026)]
```

The results were as expected, with negative feeling words like tragic, fuckin, weird appearing. I also find the existence of the word pretty interesting.

Overall, I could see why using lyrics as a base for a NLP model can be a difficult choice. Especially, lack of certain part of speech, or variety in some different kinds of words can be an obstacle.

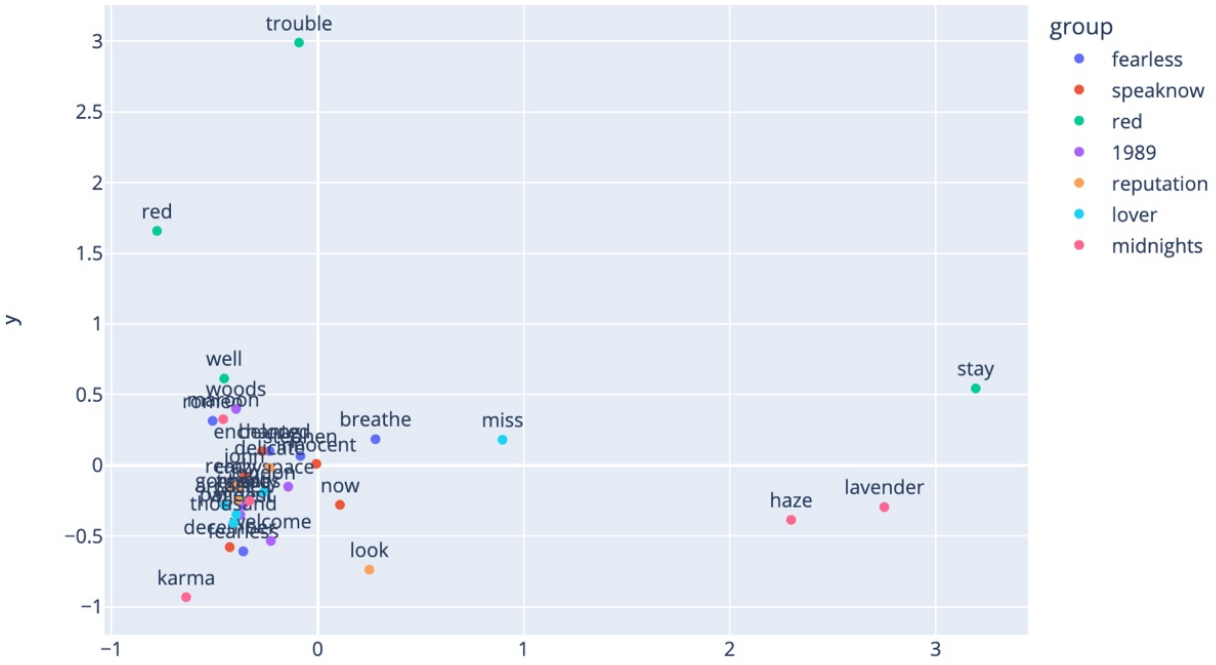
Visualization

I chose some rather unique words from lyrics. I wondered if specific words that are used in specific era(album) are related in this model, because these words will be related otherwise in other models, especially twitter model. So chose 5 words from one album, that can be seen unrelated in terms of normal language.

these are my word choices:

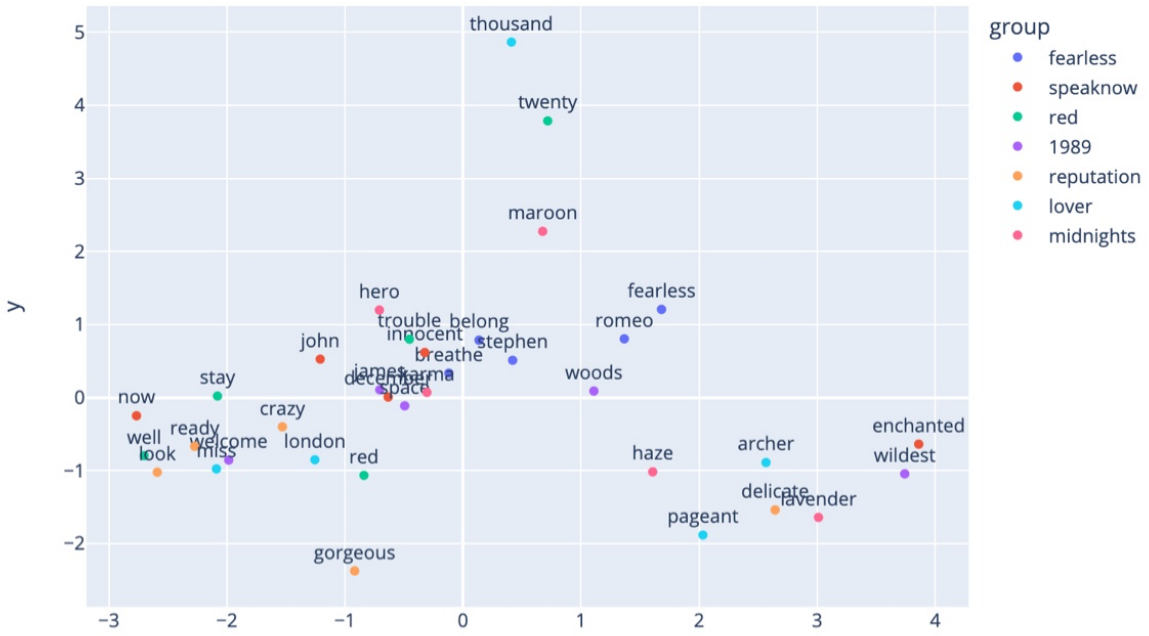
```
word_list = [
  # fearless album
  'breathe', 'fearless', 'stephen', 'romeo', 'belong',
  # speak now album
  'enchanted', 'december', 'john', 'innocent', 'now',
  # red album
  'red', 'well', 'twenty', 'trouble', 'stay',
  # 1989 album
  'welcome', 'space', 'woods', 'james', 'wildest',
  # reputation album
  'delicate', 'crazy', 'look', 'ready', 'gorgeous',
  # lover album
  'thousand', 'miss', 'pageant', 'london', 'archer',
  # midnights album
  'lavender', 'karma', 'maroon', 'haze', 'hero'
]
```

And the result was rather disappointing:



I could not retrieve good insights from this data. One thing to note was that outliers like trouble, red, stay, haze, lavender, was the title of songs that were single-cut from the albums. Maybe that track was standing out from other tracks for popularity.

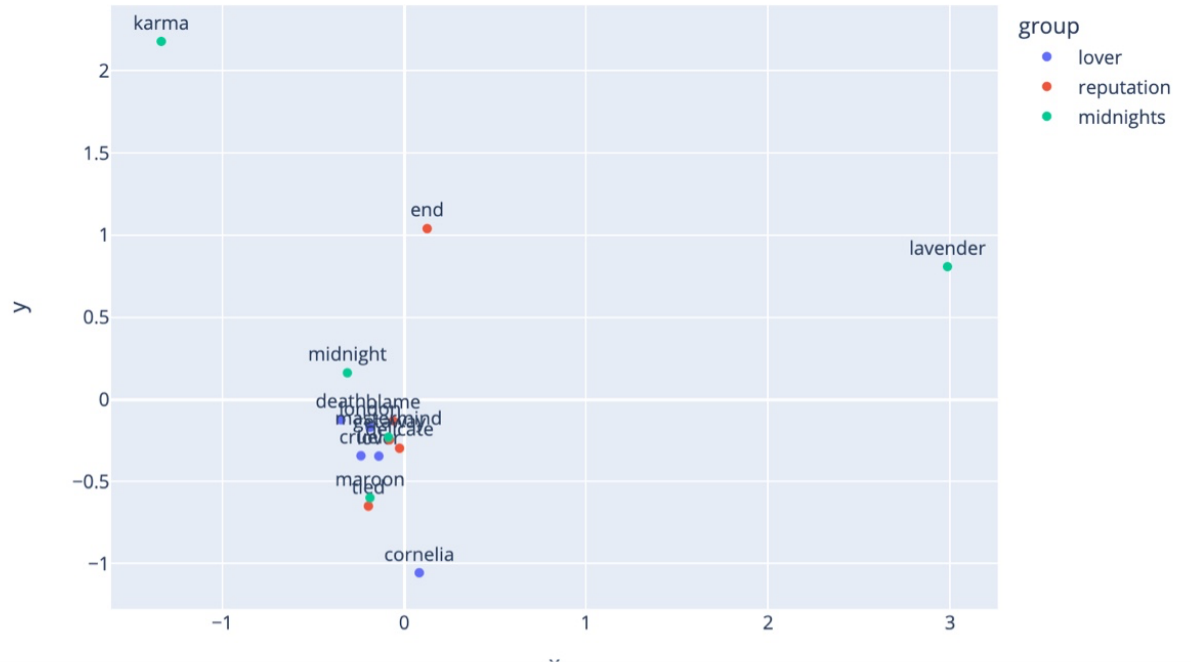
I put these words in `model`, and wanted to see how the result changes.



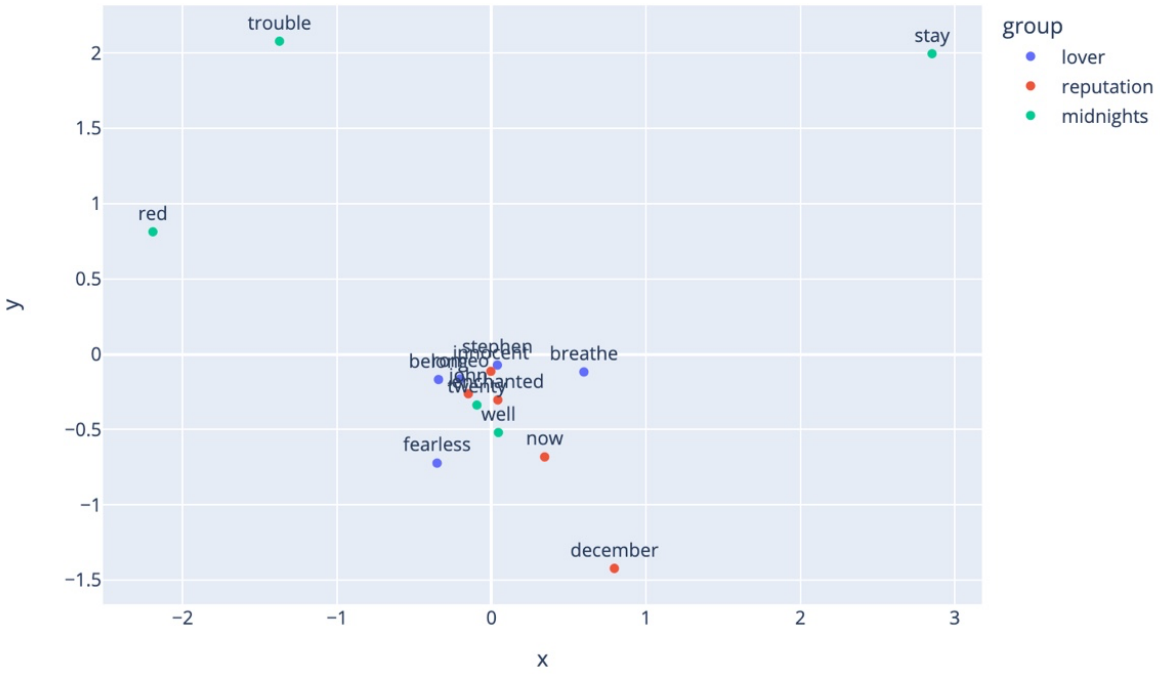
Words from *fearless* , *speak now*, *and* *reputation* album were rather clustered. So this meant that my trial to visualize words in such manner was wrong, since the `taylor_model` was not optimized even for the title of the songs, even worse than twitter-based model. I could thus conclude that having a small dataset, solely written by a person can be difficult to handle.

To utilize the data, I decreased size of `word_list`, and chose words from less album, and ran `taylor_model` to visualize again.

```
word_list = [
  #lover era(5)
  'lover', 'cornelia', 'cruel', 'london', 'death',
  #reputation era
  'delicate', 'end', 'blame', 'getaway', 'tied',
  #midnights era
  'karma', 'lavender', 'maroon', 'mastermind', 'midnight'
]
display_pca_scatterplot_groups_of_five(taylor_model, word_list)
```




```
word_list = [
  # fearless album
  'breathe', 'fearless', 'stephen', 'romeo', 'belong',
  # speak now album
  'enchanted', 'december', 'john', 'innocent', 'now',
  # red album
  'red', 'well', 'twenty', 'trouble', 'stay',
]
display_pca_scatterplot_groups_of_five_smaller(taylor_model, word_list)
```



I think the whole limits of the model originate from the dataset itself. I tried to remove the outliers, but then whole relationship changed, creating another outlier. I also tried removing duplicate albums (different editions) before building corpus. But since all songs in regular version are present in different editions, it did not influence much. I think this is the limitation of choosing music lyric of one artist, because although some keywords may vary, the variety of vocabulary can't help being small compared to batch data.

I found the `taylor_model` quite disappointing. So I tried to revise the model in different methods, such as changing the method in which txt data is imported and using negative sampling.

Importing whole Song as One Sentence

I looked at `corpus` again, and found out that lyrics, that make one sentence, are not considered in this model as one sentence, because they are given in different lines. That is, no matter how I adjust window_size, the model will not look at adjacent words, since they are in different lines ('night' and 'to' in this sense.) 'It feels like a perfect night to dress up like hipsters, and make fun of our exes' is one sentence, but in this data, there is no method to connect them.

```
#print(corpus[:100])
[['it', 'feels', 'like', 'a', 'perfect', 'night'], ['to', 'dress', 'up', 'like',
'hipsters'],... ]
```

I tried to import whole song as one sentence and see if it improves or worsens situation. Adding whole song at once yields a problem that words that are not in same sentence may be considered together, but I wanted to try it and see if the pros of sentence disjoint resolved or cons of different sentence being joined is bigger.

I altered the code like this:

```
strings=[]

from glob import glob
from tqdm import tqdm
import re

albums = glob("/content/drive/MyDrive/ColabData/TaylorSwiftAlbums/*")

for album in tqdm(albums):
  songs = glob(str(album)+"/*")
  for song in songs:
    with open(song, 'r') as f:
      song_lyric = f.readlines()[1:] #ignore first line
      if song_lyric:
        this_song_lyric = ""
        for line in song_lyric[:-1]:
          line = line.lower()
          if "[" not in line:
            line = line.replace('\n',' ')
            this_song_lyric+=" "+line
        last_line_modified = re.sub(r'\d+\s*\w+', '', song_lyric[-1].lower())
        last_line_modified = last_line_modified.replace('embed','')
        this_song_lyric+=" "+(last_line_modified)
        strings.append(this_song_lyric)

pp.pprint(strings[0])
```

For reference, the old `taylor_model` `strings` maker looked like this (only altered parts):

```
for album in tqdm(albums):
  songs = glob(str(album)+"/*")
  for song in songs:
    with open(song, 'r') as f:
      song_lyric = f.readlines()[1:] #ignore first line
      if song_lyric:
        strings.extend([x.lower() for x in song_lyric[:-1] if x[0] !="["])
        last_line_modified = re.sub(r'\d+\s*\w+', '', song_lyric[-1].lower())
        last_line_modified = last_line_modified.replace('embed','')
        strings.append(last_line_modified)
```

Instead of adding each line to strings via extend, I added each line to an temporary string variable `this_song_lyric`. Each lines are divided with a space, instead of being saved individually. Methodology for last_line_modified is virtually identical, and this_song_lyric is added to strings via append, not extend. While writing this code, I could discern the difference between append and extend, and why I had to use each of them to the previous sections that I wrote them.

new corpus looked like this:

```
print(corpus[:100])
[['it', 'feels', 'like', 'a', 'perfect', 'night', 'to', 'dress', 'up', 'like',
'hipsters', 'and', ... ],....]
```

Then, with this 544-sentence(actually song)-long strings, instead of old one with 35786, I built and made new model.

```
taylor_model = Word2Vec(sentences=corpus, vector_size=80, window=3, min_count=6,
sg=1, negative=10)
taylor_model = taylor_model.wv
```

I used same vector, window size and min_count, but added some negative sampling in hopes of processing meaningless rhyme words like ‘hmm’, ‘oh’, etc. well. I used similar questions to assess this model, and the result was as follows:

Word Analogy

```
woman
[('trace', 0.8545616269111633),
 ('whom', 0.8512718081474304),
 ('likes', 0.8409059643745422),
 ('wears', 0.8380570411682129),
 ('lovely', 0.8344222903251648),
 ('ascyltos', 0.8294989466667175),
 ('himself', 0.8281412720680237),
 ('countryman', 0.8275983929634094),
 ('whose', 0.8264104127883911),
 ('tis', 0.8249528408050537)]

love
[('fallin', 0.6349379420280457),
 ('affair', 0.6345676779747009),
 ('shining', 0.6283682584762573),
 ('play', 0.6232140064239502),
 ('ruthless', 0.6126230955123901),
 ('worship', 0.6116556525230408),
 ('players', 0.6061928272247314),
 ('tragic', 0.593811571598053),
 ('ours', 0.5881608128547668),
 ('heels', 0.5881523489952087)]

friend
[('share', 0.8309302926063538),
 ('mythical', 0.8218633532524109),
 ('trophy', 0.8200570344924927),
 ('heartbreak', 0.8128861784934998),
 ('shot', 0.8001842498779297),
 ('tho', 0.7999062538146973),
 ('masterpiece', 0.7982810139656067),
 ('bought', 0.7973718047142029),
 ('shall', 0.7965989112854004),
 ('never', 0.7961673736572266)]
```

For all three words, I could see that meaningless words, such as 'ere', 'uhoh', 'me-' disappeared. One thing to note is that associated words to woman has completely changed. Words like sisters, escape, boar disappeared. New words are lovely, knelt, saint, hercules and empire. I think woman-empowering/praising characters are more well reflected. words to love and friend has not changed in terms of meaning.

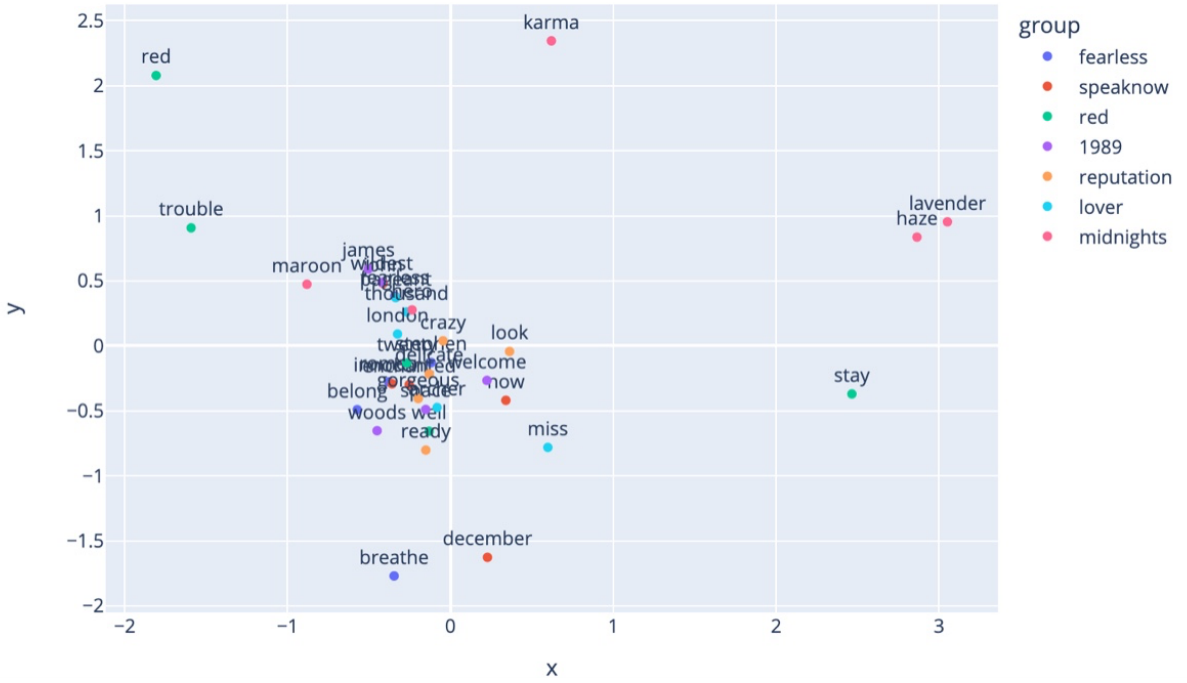
Relations

```
analogy(taylor_model, 'man', 'he', 'woman')

[('thinks', 0.6637744903564453),
 ('says', 0.6459203958511353),
 ('midnight', 0.6441094875335693),
 ('stayed', 0.613298773765564),
 ('calls', 0.6122281551361084),
 ('comfortable', 0.6117876768112183),
 ('sunshine', 0.5995890498161316),
 ('ascyltos', 0.5914212465286255),
 ('knelt', 0.5906340479850769),
 ('exactly', 0.5832395553588867)]
```

```
analogy(taylor_model, 'good', 'bad', 'beautiful')

[('peaks', 0.6466321349143982),
 ('tragic', 0.6325075030326843),
 ('rush', 0.6206655502319336),
 ('gold', 0.6052859425544739),
 ('flying', 0.604031503200531),
 ('weird', 0.5992770195007324),
 ('fuckin', 0.5983691811561584),
 ('windermere', 0.588883638381958),
 ('angel', 0.5804174542427063),
 ('sad', 0.5757679343223572)]
```

Relation-wise, the analogy for beautiful has been changed, but it does not look significant. More descriptive words appeared. However, performance for the analogy of woman has gotten worse, with she missing from the query. Maybe the word she does not appear in the lyrics dataset (since Taylor is a woman herself) or using whole lyrics at once, which makes the over-connections between words in other sentences (in terms of context) affected on this negatively.

Visualization



Mapping using same word list seems to have improved, with the music from same album being close to each other in either x_axis or y_axis. Still words from *lover* album is really close to each other, and this can be related to the fact that all songs are related to the theme of love, concisely.

Albums like *1989*, *midnights* are not as concise as *lover* in terms of theme and tone of songs. I think adding title of album as label to each words, or processing each lyrics manually by adding period(.) to sentence could be essential for NLP of music lyrics.

Summary

I used lyrics from Taylor Swift's discography from Kaggle to make my own `taylor_model`. I altered and wrote codes provided to preprocess my files, which was multiple txt with some unwanted informations at the beginning and the end. I could practice `re` module to practice using regular expressions. Then with my `taylor_model`, I analyzed relational words and analogy to a relationship, then tried to plot some words. I saw some weak relations, then tried to improve it by altering arguments or using new ways to import words, but the trial was not successful.

I could conclude that using lyrics can be more challenging, because audio factors have to be included as well to fully understand the true meaning (e.g. saying 'happy' with minor melody can be actually meant to convey the meaning of sad.) Now that I think of it, using dataset of news articles or SNS articles regarding the celebrity could actually have been helpful to find related words and do analogy.