Word2Vec for Fragrance Mapping

Dohoon Kim(김도훈)

1. Visualize Word Vectors

#At first, I selected words indicating perfume ingredients since perfume is my personal interest.

#To do so, I gained information about ingredients from article:

https://www.fragrancex.com/blog/perfume-ingredients/

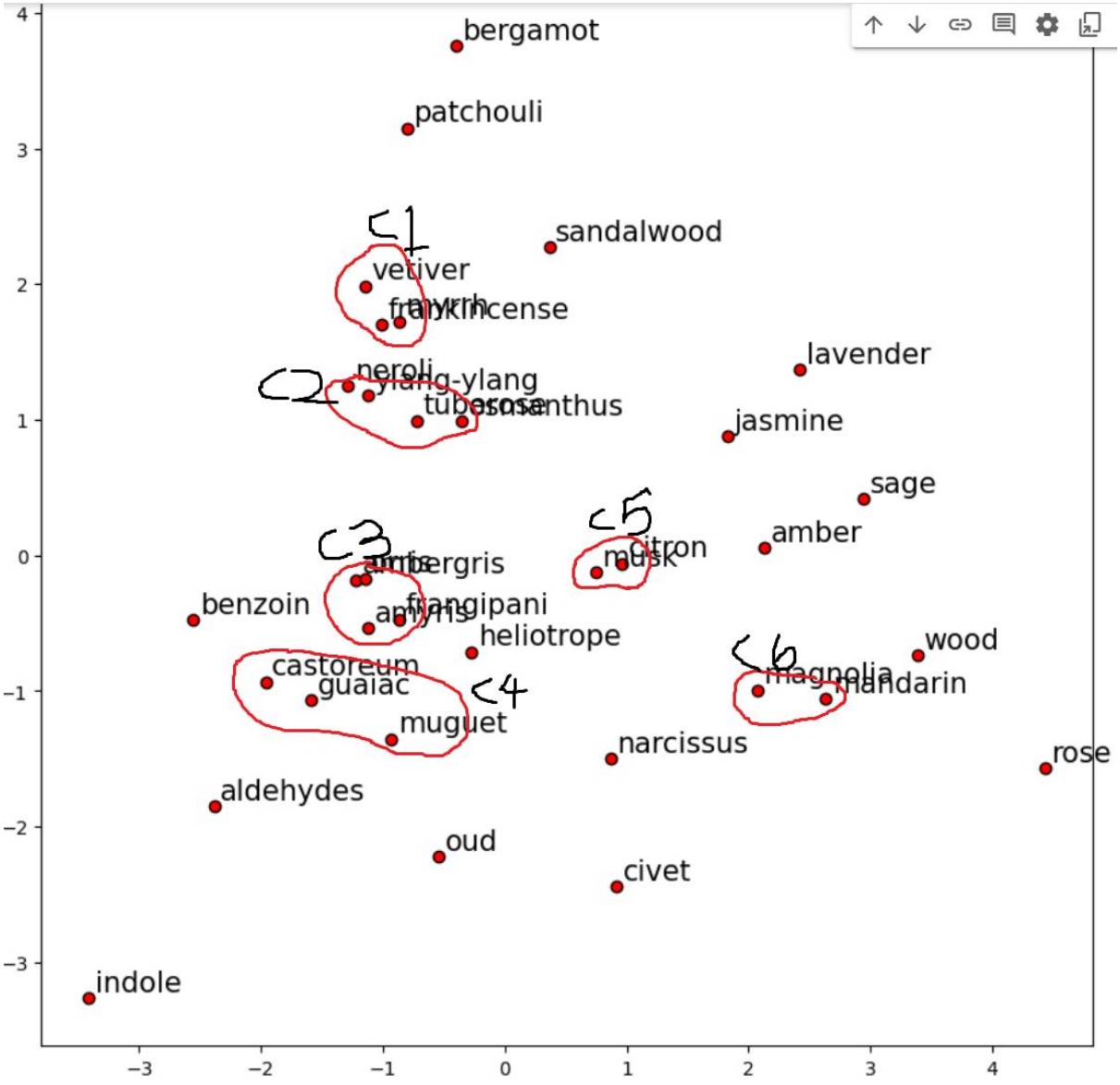
and made list of names of each ingredient.

#However, there are some rarely used ingredients that the model doesn't include

(e.g. agrumen, ambrette, labdanum, etc.). Therefore, I modified list by eliminating them and 34 left.

```
# Select word list of your own interests
word_list = ['aldehydes','amber','ambergris', 'amyris',
             'benzoin','bergamot','castoreum','citron',
             'civet','sage','frangipani','frankincense',
             'guaiac','wood','heliotrope','indole','jasmine',
             'lavender','magnolia','mandarin','muguet','musk',
             'myrrh','narcissus','neroli','orris','osmanthus',
             'oud','patchouli','rose','sandalwood','tuberose',
             'vetiver','ylang-ylang'
]

print(len(word_list))
display_pca_scatterplot(model, word_list)
```

```
34
```



#This is the result of mapping words.

#How Words are in 2D Space and Clustered

#PCA is a way to reduce dimensions while maximizing the variance of high-dimensional data. Through this process, similar words can be placed in similar location in 2D space. There are mainly six clusters in 2D space. All of clusters have less elements than four. Words in each cluster have more to do with than words from outside. For example, in C1, "vetiver", "frankincense", "myrrh" are all woody, earthy, and intense. While in C2, words "neroli," "ylang-ylang", "tuberose", and "osmanthus" are likely placed in close to each other, since they are words that indicate ingredients that perform floral, sweet, and fresh scent.

#Suitability of Word Clustering and Unexpected Examples

#There is reasonable relationship between each word in same cluster, but this clustering still has unexpected drawbacks. For example, though "bergamot" and "mandarin" are all citruses and similar scent, the model did not catch their similarities. Another case is remoted location of 'rose'. It was expected that the 'rose' assigned near of floral ingredients (e.g. neroli, tuberose, etc.). Since the pre-trained model 'glove-wiki-gigaword-300' is not especially trained to cover some fragrance problem, it is likely that detailed analysis is not performed.

2. Train New Word2Vec

#To handle drawbacks found previous problem and make more precise classification of ingredients, I will use another model which is trained by another text file.

#At first, to accomplish the purpose, I found text file related to perfume in 'kaggle'.

The url is. https://www.kaggle.com/datasets/nandini1999/perfume-recommendation-dataset

#This is dataset of information of more than 2000 perfumes. It was originally csv file, which has columns such as:

1. Name = Name of the perfume
2. Brand = Brand or Company to which the perfume belongs to
3. Description = Some text describing the feel and features of the perfume.
4. Notes = A list of fragrance notes present in the perfume
5. Image URL = URL of the perfume image

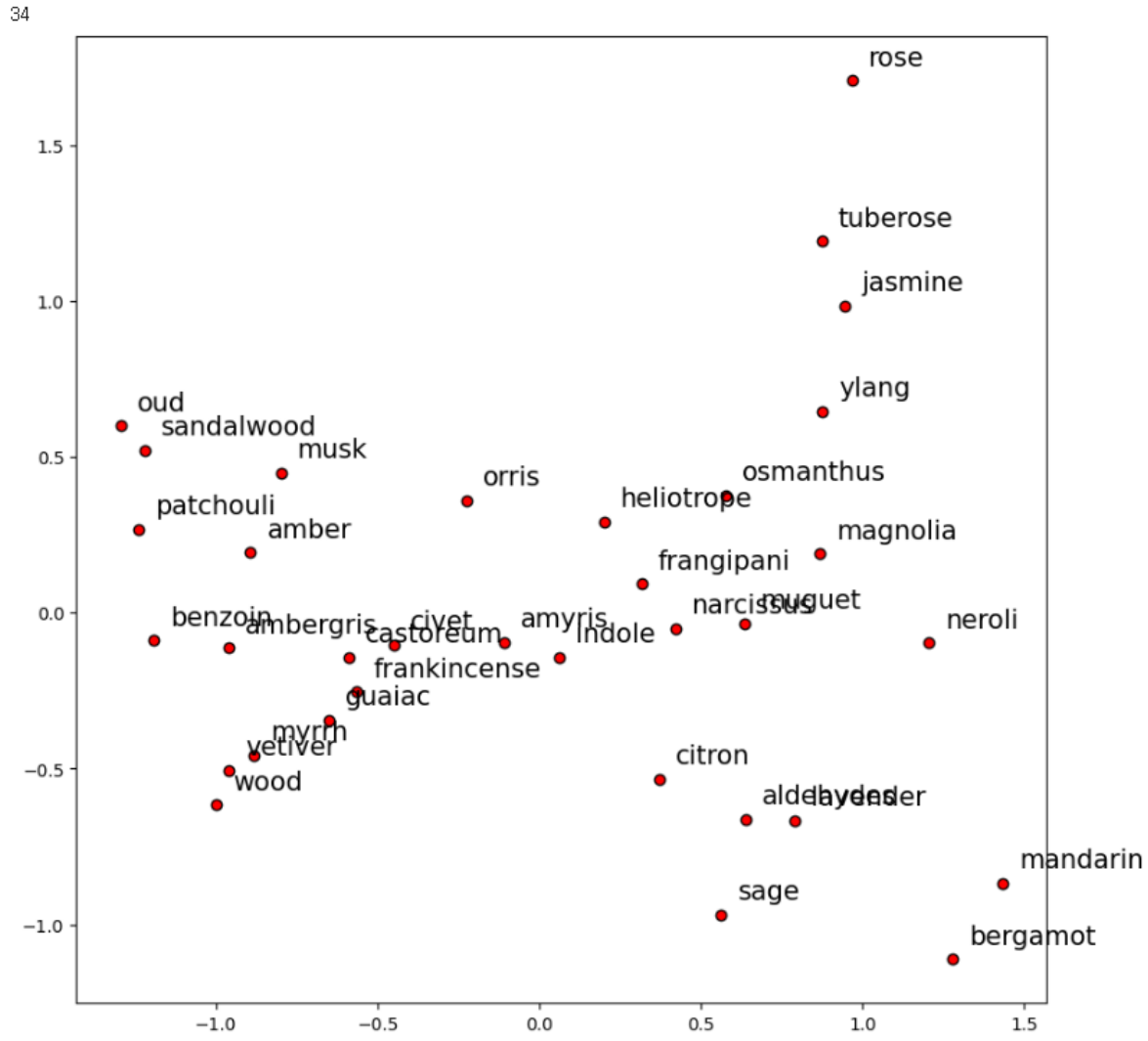
#However, to train the model with text file, which is composed by sentences, I only extracted the information from the third column: description. Then I add the article that I used in previous problem and saved it as 'perfume_decription.txt'.

```
your_text_fn = '/content/perfume_description.txt' # Enter your text file name here

#with open(your_text_fn, 'r') as f:
with open(your_text_fn, 'r', encoding='utf-8', errors='ignore') as f:
  strings = f.readlines()
corpus = make_tokenized_corpus(strings)
'''
```

```
model1 = Word2Vec(sentences=corpus,
                  vector_size=100,
                  window=5,
                  min_count=2,
                  sg=1,
                  negative=5,
                  workers=4)

model1 = model1.wv # To match with pr
```

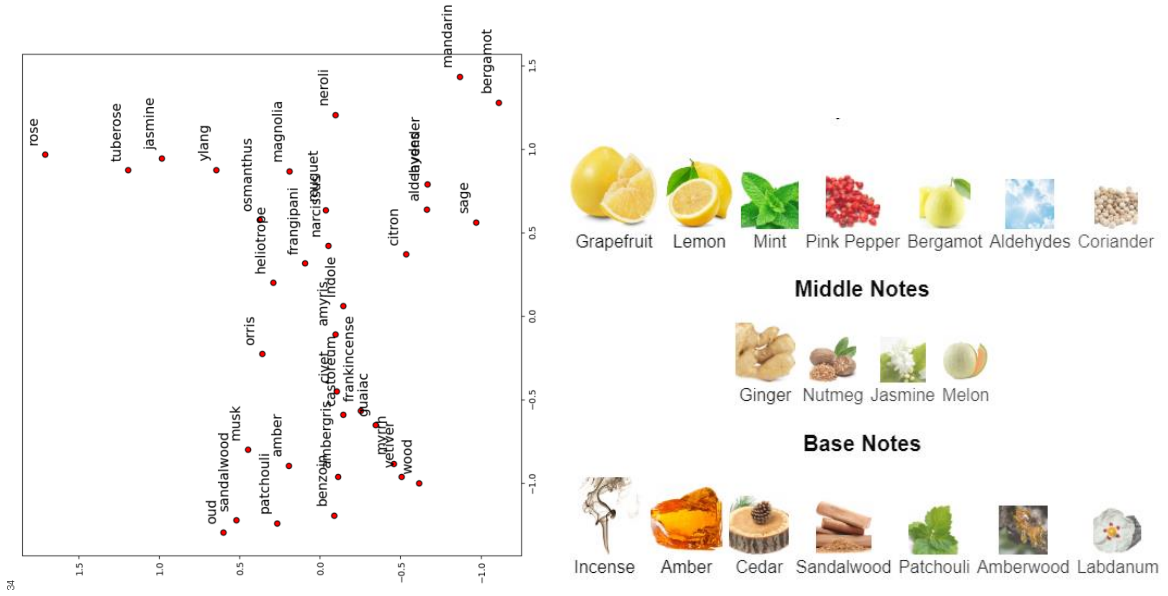
#Since amount of word from dataset is much smaller than "glove-wiki-gigaword-300", I reduced some parameters of Word2ved such as 'min_count' or 'negative'.

#The work in question 4 was carried out in the same way. The result is



#The first interesting difference derived from using another dataset is that it seems to understand each feature of ingredients. Looking at x-axis, the aroma of the raw material

becomes lighter as the x value increases. On the left side of the graph, some heavy material such as 'oud', 'amber', 'sandalwood' are located while on the right side, there are light ingredients which perform floral or citrus scent. Usually, heavy materials are used in base note and light materials are used in top note. So, If you rotate the graph counterclockwise 90 degree, you will find a similarity to the picture on the right below.



#The second interesting thing about new model is that it prints out more similar words when typing word of an ingredient in function 'most_similar'.

```
target_word = 'bergamot' # Enter your word string here
# check the word is in the vocabulary of the model
assert model.has_index_for(target_word), f"The selecte
model.most_similar(target_word)
```

```
[('chamomile', 0.4372253417968875),
 ('patchouli', 0.4317514896392822),
 ('lavender', 0.405318021774292),
 ('neroli', 0.40015318989753723),
 ('spearmint', 0.3657273650169372 6),
 ('corallina', 0.3653071820735931 4),
 ('horndon', 0.3649813234806061),
 ('lidcombe', 0.364951103925704 96),
 ('broadbeach', 0.3599428236484527 6),
 ('thymol', 0.3595306873321533)]
```

```
target_word = 'bergamot' # Enter your word string here
# check the word is in the vocabulary of the model
assert model1.has_index_for(target_word), f"The selecte
model1.most_similar(target_word)
```

```
[('grapefruit', 0.9601817727088928),
 ('lemon', 0.9556978940963745),
 ('mandarin', 0.9434173703193665),
 ('lime', 0.942919135093689),
 ('petitgrain', 0.9335858225822449),
 ('zesty', 0.9315468072891235),
 ('blackcurrant', 0.9287005066871643),
 ('tangy', 0.9284946322441101),
 ('pink', 0.9220525026321411),
 ('basil', 0.9212995171546936)]
```

#Left image is output of the function when using previous model and left is that of new model. In this case, since 'grapefruit', 'lemon', 'mandarin', and 'lime' is all orange-shaped fruits, it seems that new model better understands characteristics of specific words.

#The third interesting thing is that cosine similarity is much higher than existing model. When comparing result, the similarity was higher in left case. To confirm, the similarity of words not related to perfume was obtained and the results were compared.

```
target_word = 'king' # Enter your word string here
# check the word is in the vocabulary of the model
assert model.has_index_for(target_word), f"The sele
model.most_similar(target_word)
```

```
[('queen', 0.6336469054222107),
 ('prince', 0.6196622848510742),
 ('monarch', 0.5899620652198792),
 ('kingdom', 0.5791266560554504),
 ('throne', 0.5606487989425659),
 ('ii', 0.5562329292297363),
 ('iii', 0.5503199100494385),
 ('crown', 0.5224862694740295),
 ('reign', 0.5217353701591492),
 ('kings', 0.5066401958465576)]
```

```
target_word = 'king' # Enter your word s
# check the word is in the vocabulary of
assert model1.has_index_for(target_word)
model1.most_similar(target_word)
```

```
[('child', 0.9845563769340515),
 ('1920s', 0.9834492206573486),
 ('isabey', 0.9826719164848328),
 ('angels', 0.9824622869491577),
 ('wrote', 0.9824082255363464),
 ('success', 0.9820895195007324),
 ('londons', 0.9816744923591614),
 ('mother', 0.9813142418861389),
 ('novels', 0.9811524748802185),
 ('grew', 0.9810875654220581)]
```

#When putting 'king' to each function, the result is same. It means that no matter which word is used, the function outputs high similarity.

#The fourth interesting example is related to the case of word 'king'. When word has nothing to do with perfume, the model outputs less accurate result. We can easily judge that 'king' is not like 'child' or '1920s' as having more than 98% similarity. Maybe this happens because of lack of words in dataset.

```
print(len(model1))
```

```
12313
```

#Only 12313 words are in the model, and it is much smaller than original. That is reason why this kind of problem occurs. To improve it, fine tuning the pre-existing model with given text is needed.

#The last interesting thing is that it performs better in term of estimating scent.

```
print(analogy(model, 'neroli', 'floral', 'sandalwood'))
print(analogy(model1, 'neroli', 'floral', 'sandalwood'))
```

```
[('flowers', 0.4777246415615082),
 ('flower', 0.461261510848999),
 ('adorned', 0.45864105224609375),
 ('ornaments', 0.4481959342956543),
 ('incense', 0.4429246783256531),
 ('decorative', 0.4398514926433563),
 ('bouquet', 0.4307063817977905),
 ('embroidered', 0.4271883070468902 6),
 ('lace', 0.42621755599975586),
 ('carvings', 0.4248176813125610 4)]
None
[('patchouli', 0.69670569896698),
 ('vanilla', 0.6926760673522949),
 ('amber', 0.6913019418716431),
 ('benzoin', 0.691146969795227),
 ('tonka', 0.6843622326850891),
 ('cedarwood', 0.6829566359519958),
 ('animalic', 0.6752651929855347),
 ('base', 0.6719072461128235),
 ('labdanum', 0.6650844216346741),
 ('bean', 0.6644149422645569)]
None
```

#From the bottom results, we see that this model can better extract scent information. Just as 'floral' refers to the scent of 'neroli', 'animalic' refers to the scent of 'sandalwood'. Other examples are also related to the scent of 'sandalwood'.