The Perfume Recommendation System

By Jingya Cao, Siyuan Chen, Xin Shen, Shixuan Wan

December, 2019

1. Introduction

1.1 Background information

For the people who want to buy the perfume, you will often find yourself overwhelmed by different kinds of perfumes when you search online. Even you search them based on certain filters, you still have to select one from several web pages by reading the long, unusual and highly descriptive introductions. Those poetic introductions describe the feelings emotionally. Those introductions even use a scene to describe the moment someone smells the perfume, which takes some time to understand. Picking the perfume by reading introductions is absolutely time-consuming. As a perfume customer who is confused by the introductions and want to find the right perfume in a short time, our perfume recommendation system is a good choice for them. This project is intended to apply the NLP in perfume selection and check the validation of the NLP model.

1.2 Goals of the system

The aim of this system is to give 5 recommendations based on the description input, which increases the choosing efficiency. The special corpus of the perfume is good for NLP. It contains the emotion, the feeling and the personality description. When the people input a descriptive text, the model can return the perfume that matches the input text. What's more, our system includes sentiment analysis. The system can detect your likes and dislikes.

1.3 Format of Report

In the following part of the report, the first part introduces the web crawler used to get the

perfume information. The source data comes from *https://www.luckyscent.com/*. The second part shows the data details, which include the data type and data format. The third part is about the lemmatization of the source data. The fourth part gives the detailed introduction of the models used, which include the tf-idf and doc2vec. And then we run the model and print the outcomes. We will give some sample inputs and have a look on the outputs. Finally, in the conclusion part, we give our insights and list the weakness of our models so that others can have a comprehensive understanding of our work. Under each part, the code file is listed near the subtitle. All the files can be found on the GitHub repository at ….

1.4 How to Run

Module Requirement:

Scipy >= 1.3.0
Numpy >=1.16.0
Matplotlib >= 3.1.0
Skimage > 0.15.0
Nltk >=3.4.0
Genism >=3.8.0
Sklearn > 0.21.0

Run the model:

We have scraped the raw data, trained the model and persist them into files so simply run the 'run_model.ipynb' file and input the sentence into the box, you would get perfume recommendation.

1.5 Task Division

The team works on NLP model designing and training together. Each of the team member is assigned the responsibility in specialized aspects.

1. Design the thesis of this project. Coding data scraping and part of model designing. Reporting of Introduction and Conclusion (Jingya Cao)

2. Coding of data scraping, data cleaning and part of model designing. Reporting of Data Preparation (Siyuan Chen)

3. Responsible for all model designing and model training processes. Choose the training models and do the model optimization. Reporting of model training. (Xin Shen)

4. Coding of information retrieval, model running, UI implementation and the part of model designing. Reporting of Information Retrieval and UI Implementation. (Shixuan Wan)

2. Data Preparation

2.1 Files description

get_data.py: A web scraping script to get the data frame of perfume source data. we have already scrape the data we need and store it in the 'perfume_data.pkl' file so you do not have to scrape the data again. Simply load the data using 'pickle.load' function is fine.
model_training.ipynb: Include text pre-processing and two model training(TF-IDF and Doc2Vec). We have vectorized the raw data and persist them into several pickle files so as to be used directly.
run_model.ipynb: Show the query results in the interface.
information_retrieval_model.py: Run the model we have trained

2.2 Web scrapping

To recommend perfumes to customers, we needed data sources which include perfume names, descriptions, images and notes. We gained these information by scraping data from a popular niche perfume website(https://www.luckyscent.com). The crawling script with the scrapped URLs can be found inside the Python project as ***get_data.py*** file. In this step, we successfully transformed those unstructured source data to a structured data frame, which is convenient for our model training.

2.3 Data Description

In the crawling script file, we scrapped four sources of data(which are highlighted by a red rectangle) from the website :

- Name
- Description
- List of Notes in the Perfume
- Image

Then we concatenated these into a data frame with 2288 rows and 4 columns, and pickled this data frame to a ***perfume_data.pkl*** file to load it directly next time.

Here is an example of all text data sources for a perfume, Divine Vanille:

Here is an example of data frame with first 4 perfumes:

	title	imgae	description	notes
0	Creation-E Parfum Cologne	https://www.luckyscent.com//images/products/74...	Roja's much loved Creation-E, or Enigma as it'...	bergamot, geranium, rose de mai, neroli, jasmi...
1	Baccarat Rouge 540	https://www.luckyscent.com//images/products/49...	In the Fall of 2014, Baccarat, the most presti...	Citrus, jasmine, saffron, sage, ambergris, oak...
2	Bee	https://www.luckyscent.com//images/products/76...	We don't usually play favorites. We love all o...	orange, ginger syrup, royal jelly accord, broo...
3	Pacific Rock Moss	https://www.luckyscent.com//images/products/79...	A bona fide fragrance sensation, Pacific Rock ...	Australian coastal moss, lemon, sage, geranium...
4	Ani	https://www.luckyscent.com//images/products/77...	An ancient metropolis now abandoned to the age...	Bergamot, green notes, blue ginger, pink peppe...

The main content above are descriptions and notes. Because the language of perfume description is so rich and notes concludes many keywords(raw materials for making perfumes), so we could then train a model to recommend perfumes that match a description of a mood, a feeling, a personality, or an event like vacation.

2.4 Data Cleaning

Because the data we got now is a nearly cleaned structured data frame, so we did the data cleaning mainly by pre-processing the raw texts with ***nltk*** package, which include stemming, lowering case, removing stop words and punctuations. This was done by 4 functions – *stem_words(), make_lower_case(), remove_stop_words(), remove_punctuation()* -- and that would make our recommendation more simplified and better.

3. Model Training

3.1 TF-IDF Model:

Since we cannot analyze raw documents data using machine learning algorithms, we have to vectorize the information these documents provide. A common way to do so is to calculate tf-idf value.

In information retrieval, tf–idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today. (from Wikipedia)

tf (Term Frequency)

The most frequently used tf(t,d) value are to use raw count or binary 'frequencies' (term occurs yes/no). However, considering the different number of terms in each document in a corpus. We usually adjust the 'tf' value for document length so we have:

$$f_{t,d}/|d|$$

$f_{t,d}$ is the raw count of term t in document d

d is the number of terms or document length

idf (inverse document frequency)

It describes how much information the term provides. We have:

$$idf(t,d) = log_2\left(\frac{N}{N_t}\right) + 1$$

N is the total number of documents

N_t is the number of documents that contains term t.

Therefore,

$$tfidf(t,d) = (f_{t,d}/|d|) \cdot (log_2\left(\frac{N}{N_t}\right) + 1)$$

We use 'TfidfVectorizer' function in 'sklearn' module to get the 'Tfidf' matrix. The input is just a string containing information from 'description' and 'notes' columns. We set the parameters in 'TfidfVectorizer' function as min_df = 10, ngram_range = (1,2), stop_words = 'english'. And the output 'tfidf' matrix size is (2288, 3615). Then we use pickle module to store the 'tfidf' data into file.

Truncated SVD

This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with scipy.sparse matrices efficiently.

In our case, we use truncated SVD in sklearn.decomposition to get a truncated version of data matrix to mitigate the burden when training models. We set parameter n_components = 500 and then store it. The truncated matrix size is (2288, 500).

3.2 Doc2Vec Model:

Doc2vec is an unsupervised algorithm using neural network to generate vectors for sentence/paragraphs/documents. And these vectors can find similarity between sentences and documents. Due to its architecture, the model contains context and semantics inside the document. The context of the document and relationships between words are preserved in the

learned embedding. We use Doc2Vec function in gensim.models.doc2vec module and set the parameters vector_size = 15, min_count = 5, epochs = 100, seed = 0, dm = 0 and all else by default. The model training time is about 30s. Then we persist it as a file named 'doc2vec_model'. The feature size we get is (2288, 15). The feature matrix for the first 5 perfumes is shown as below:

title	0	1	2	3	4
Creation-E Parfum Cologne	0.549590	1.854232	-1.173573	0.613501	-3.034639
Baccarat Rouge 540	-0.134542	1.043279	-0.720354	1.385473	-4.060413
Bee	-1.214550	0.215745	-1.128564	1.886470	-1.579626
Pacific Rock Moss	-2.778608	0.873175	2.453677	-0.872026	-2.752821
Ani	-1.542883	0.728623	-0.996456	0.900571	-1.520029

title	5	6	7	8	9
Creation-E Parfum Cologne	-0.474585	0.065858	3.367525	1.314173	0.830975
Baccarat Rouge 540	-0.892275	-1.711593	-0.822155	1.224066	-0.152057
Bee	-1.473644	-2.101403	0.004945	-0.695259	2.744827
Pacific Rock Moss	-0.661539	-0.696913	0.155676	-1.631206	-0.318072
Ani	-2.467376	2.277984	2.207020	0.451579	0.947415

title	10	11	12	13	14
Creation-E Parfum Cologne	0.152633	0.918908	-0.938568	1.438391	2.299377
Baccarat Rouge 540	2.217224	1.177417	-0.985075	-2.540311	0.562573
Bee	1.009817	0.016830	-0.406918	0.204303	3.234323
Pacific Rock Moss	1.842871	0.239168	-1.813368	0.252105	2.846847
Ani	1.491260	-1.622022	-1.103055	-2.405279	2.326366

4． Information Retrieval

We expect that customers will input several sentences to describe the perfume he/she wants and negate some feature he/she dislikes. We vectorize the description sentences and compare the cosine similarities with vectors stored in the models we previous changed to pick up top 5 choices that matches the description. The more detailed steps are as follows:

4.1 Preprocessing the description text

Similar to the model training part, we use Snowballstemmer as word stemmer and remove all punctuation and stop words and put all words into lower case.

4.2 Sentiment analysis for negation

For this part, we assume that customers will use negative sentences to show that they do not want certain features to be shown in the final query results. We expect sentences like “I like scent of cinnamon. I do not want vanilla or patchouli.” or “I enjoy osmanthus, but not benzoin or cedar.” To deal with such descriptions, we first divide input into several parts by periods or words like “but”. Then we use SentimentIntensityAnalyzer in nltk.sentiment.vader to get the general sentiment of this message(positive or negative). We didn’t train the sentiment analyzer manually because there are not any pre-existing labels for supervised learning. The results are classified as “love message” and “hate message”.

4.3 Similarity analysis

We will use two feature sets to analyze the similarity(or “dissimilarity”) between input messages and perfume data sets. We transform messages into vectors both using BoW model(tf-idf) and doc2vec model, according to feature sets we built in the last part and calculate the similarity scores using cosine similarity. When caculating the similar scores, we use of the average of BoW similarity(using TF-IDF feature sets, not considering the order of words) and semantic similarity scores(using doc2vec feature sets, considering the order of words through neural networks). Since we can not get the feedback from customers for how our model performs when picking the requested perfumes, we cannot decide whether BoW or Doc2Vec performs better, and that maybe improved with additional feedbacks when applied to the real market.
For “love messages”, we view high similarity score as a positive factor to put it into final query results; as for “hate messages”, we view high similarity as a negative factor, and will exclude those perfume data with dissimilarity score higher than 0.3 from query results.

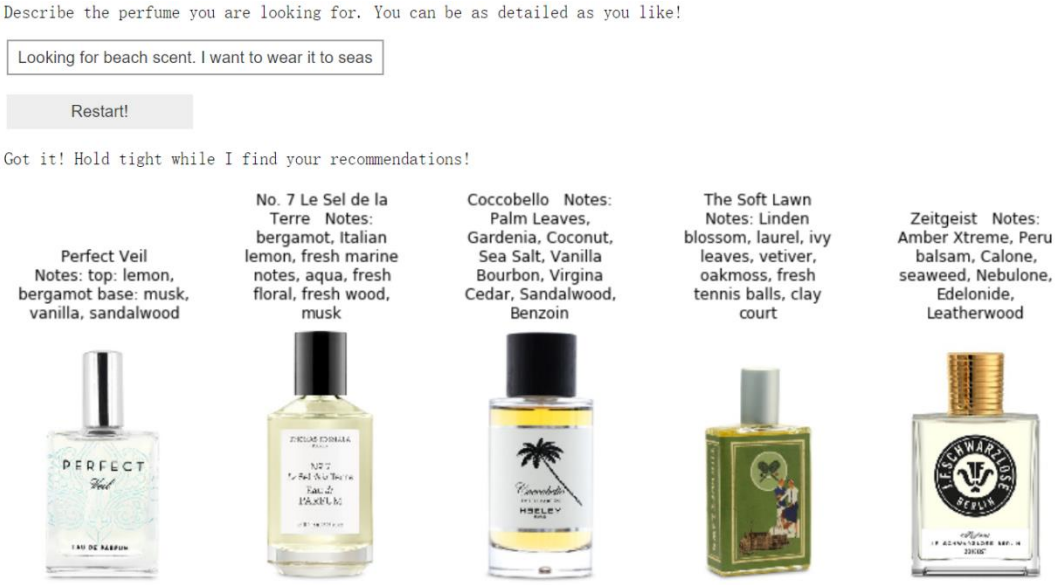
4.4 Output results

We use subplots to print out 5 perfumes in the data sets with highest similarity scores on “love messages” from input and exclude ones with high similarity score with negated “hate messages”. The plots includes pictures, perfume name and notes(mainly scent) of the result perfume. Customers can easily see their input descriptions pick up the relevant results.

5. User Interface

The user interface is designed to be a chatbot, we use IPython.display and ipywidgets to implement this. The user is expected to input text descriptions into the widgets module and press Enter. Then program then will analyze the text message and print the query results output from the information retrieval part. If user want to open another query, he/she can just clean the result by clicking the “Restart” button and then input another query request.

Here is an example of the user interface and a simple query result:

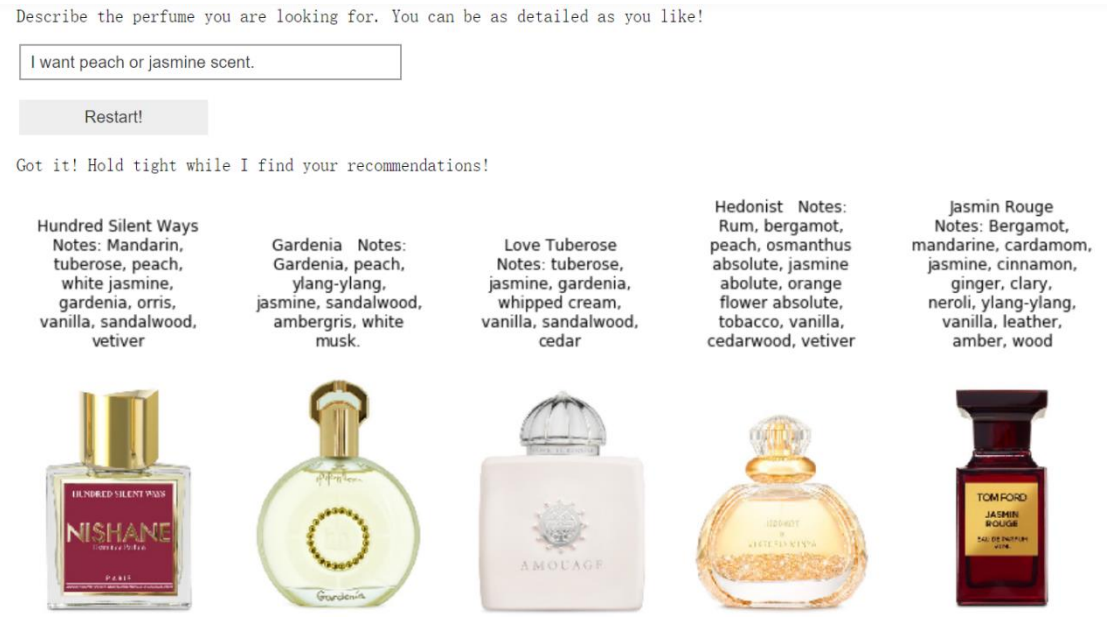


The query message is “Looking for beach scent. I want to wear it to seaside or pools.”

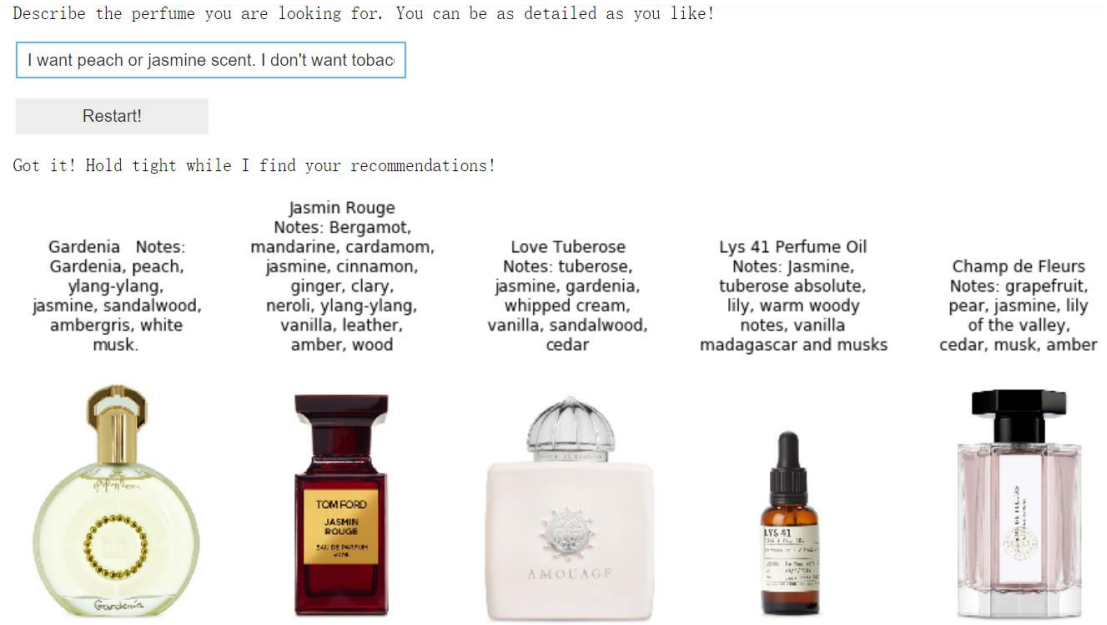
The results shows some relevant scents related to beach or sea like “fresh marine”, “aqua”, “sea salt”, ”seaweed”.

Testing the Negation:

Here is another test on how the negation works:



This query is “I want peach or jasmine scent” and correctly print out some relevant results.



This query add “I don’t want tobacco.” to the last query, thus the result excludes the “Hedonist” perfume in the last query with tobacco scent.

6. Conclusion and future work

Basically, This project built a perfume recommendation system. Whatever customers say about their desires, the recommendation system would offer 5 choices for them. The more precise the input about perfume ingredients, the more accurate this system would be. However, The model used cannot provide an accuracy. This is based on unsupervised learning, measuring the performance of this kind of algorithm may be challenging. One way to test the system is to check if the recommended perfumes are relative, given many times of input description. This might provide a rough accuracy about this system. Despite of this, we know that many works need to be done to further improve this system. First, we do not use reviews data and other useful information to train our model, so generally speaking, collecting more data and re-train the model should shape the system in a better way. Second, the data cleaning and preparation part also need to be improved. We checked the lemmatization part which is a bit unsatisfying so improving this part might lead to a better feature set. Third, we reduced the feature size in order to shorten the model training time, so more parameters and models should be used. This should optimize the model further.