AIRBENDER PROJECT
Airbnb Listings Recommender System

Problem

Airbnb is an American vacation rental online marketplace company based in San Fransisco, California, United States. Airbnb offers arrangement for lodging, primarily homestays, or tourism experiences.

From the website or mobile application, travelers can search and filter the listings in Airbnb that are suitable for them based on location, price, number of accommodations, and many other criteria. However, sometimes it still returns us hundreds of listings as the result. To determine the best place, usually we read all the descriptions and reviews from other travelers to ensure the listing is suitable. It takes a lot of time considering the total available options. It would be wonderful if the listings can be filtered automatically. Hence, the Airbender project, Airbnb listings recommender system.

There are some questions when travelers access Airbnb, for instance:

- "I've never been to this city, but I know what kind of place I like to stay. How can I filter it based on my preference?"
- "I stayed once in this listing, and I love it! Where can I find other similar listings?"
- "I came to this city a lot. I gave comments to those listings and I believe others have too. Where do they usually stay?"

Based on those problems, my goal of this project is a recommendation system based on user preference/query, content similarity from previous visits, and preferences from similar travelers. This system would be beneficial not only for travelers, but also for Airbnb because this could increase their customer user experience.

Data Description

The dataset can be acquired from insideairbnb.com. They are categorized by city and Vancouver city data was chosen due to my familiarity with the city. There are 4 datasets as the initial data source, but it ended up with 2 datasets for this project:

- Listings.csv: This file is in CSV format and contains all information about the listings in Vancouver, such as name, description, summary, price, etc. with total of 106 columns, both numerical and categorical data type.
- Reviews.csv: This is also in CSV format and represents the review for each listing from different users. It has listing id, id, date, reviewer id, reviewer name, and comments

Method used for Recommender System

For the first and second goals, content-based filtering method is used to find the similarity between user query or traveler's previous visit and all other listing's content. Regarding the third goal, collaborative filtering is applied to get the recommendation from similar user preferences.

Content-Based Filtering

Preprocessing and EDA

Listings dataset was used for the first method. There is no data duplication for this dataset. Since several categories can be filtered directly from the website (i.e. price, # of accommodations, # of beds, accommodation type, etc.), only descriptive information columns were considered as the final dataset, which are name, summary, space, description, neighborhood overview, notes, transit, access, and house rules. All these columns values were checked using *worldcloud* library to see the most frequent words in each column. Worth to note that there is a similarity of information between summary, space, and description column, and after further checking, those 3 columns were still considered to be included. In addition to that, I also included neighborhood column, which is a categorical column that describes the neighborhood location of the listing, into consideration. Finally, the null values were filled with blank space and all columns were combined into 1 single column called 'content'. To understand the distribution of the words in this new column, a histogram was shown to see how many words in each listing content. With an average of 430 words, it seems that many hosts want to describe their listings to the best to attract travelers.

Modeling

First, the content column was tokenized using TF-IDF vectorizer from scikit learn to get the numerical statistic to show the importance of a word within a document. Next, the tokenized data was transformed into a matrix. The standard NLTK stop words were used during this process. When it came up to the most frequent word, the word 'Vancouver' was there, and I decided to put it as an additional stop word considering all listings are in Vancouver.

Following that, a function for content-based recommendation system was built to compare the similarity between listings. Cosine similarity function from scikit learn was used to compare the matrix. There were several steps to build the function, such as: optimizing the performance, including the user query, considering multiple listings as the argument, and combine all the logic into a single function with different arguments. The function was tested by multiple scenarios (one listing, multiple listings, user query, and multiple listings + user query) and it gave us the results with similarity scores.

Next, neighborhood column, with categorical information, was transformed to get the numerical value using pandas get_dummies function. After I combined this data to the previous matrix and test the same function, we got a better similarity score. However, all the result of top recommendations is from the same neighborhood. It seems that this is not a good feature to add. There is a possibility that travelers want to find a similar place, but not in the same neighborhood. Considering this, the neighborhood feature was not included in the final matrix.

As a conclusion, we have 1 final function called "content_recommender_fin" as our content-based filtering recommendation system and a matrix that contains only the descriptive information.

Collaborative Filtering

Preprocessing and EDA

Reviews dataset was used for this process. There is a total of 0.04% of null values for the comments column, so the rows were dropped. However, there was no data duplication here. Since there is no review score in this dataset that will be useful for our collaborative filtering, I did sentiment analysis using 2 commonly used libraries, NLTK VADER and TEXTBLOB. Both functions gave polarity scores for each review to determine if the review is having positive or negative sentiment. After seeing the distribution histogram, I decided to use TEXTBLOB function because it had more normal distribution. The result of this function were values between -1 and 1. I did scale process using MinMaxScaler from scikit learn to put the value between 0 and 1, in addition to standardize the polarity score.

When it came to exploration, I found that 93% of the reviewers only reviewed 1 listing. It seems those are tourists that come to the city once only. In this case, I can not use all the data for collaborative model because it will be useless. For example, we have reviewer A that has 1 review to listing X. Then, we try to find another reviewer that also reviews listing X, let's say reviewer B. But reviewer B only has 1 review too, which is this listing. We can not find other listings to be compared. Considering this, I filtered the data for listings and reviewers that have minimum 2 reviews.

Modeling

At first, I tried to build a function for user-item and item-item filtering, but it did not work well. Then I decided to use Surprise library, a package that is commonly used to develop recommendation system for collaborative filtering. I transformed the dataset into a user matrix format, the format that is required for this library, using reader and dataset function from the library. The library has several algorithms and I ran all of those with basic parameters to see which algorithm has a better RMSE score, accuracy metric for the predictions. Lower score means better accuracy.

From the simulation, I found BaselineOnly and KNNBaseline are the two best options. I did hyperparameter optimization for both algorithms, and BaselineOnly with parameter als method, reg_i = 20, reg_u = 5, and n_epochs = 5 are the best. I did train test split and modeling using this parameter, and it gave good accuracy score on the test set. However, when I did full modeling and did the prediction for the missing ratings, it predicts the same result for all users. This model can not be used for the recommender system. I run the same steps for KNNBaseline, and it also gave us the same problem. Finally, I tried an algorithm that mostly used for collaborative filtering, which is matrix factorization – SVD. This model gave better results and can be used for the recommendation system.

Summary

As the final product, a prototype demo was built using Streamlit app where we can test all 3 scenarios. Both modeling, content-based and collaborative filtering, are useful to handle these problems. In real life scenario, these recommendation systems can be used by Airbnb to help travelers choose their preferable listings easily.

So far, we have not considered the listings that the travelers do not like. What if they stay in a listing and they do not like it? Our recommender system still suggests other listings that are similar to his previous visit. In that case, my potential next steps are to put this into consideration and scale the model to other cities.

END