Content-based and Collaborative Filtering Recommender System for Airbnb

Project Final Report

Hannah HO-LE Yifan JIANG Kai KANG Miao WANG

1 ABSTRACT

This project we have chosen is an Airbnb recommendation system. In this report we will examine two approaches we used: a content-based recommendation system and a collaborative filtering one. These are standard approaches in machine learning when trying to produce recommendations for consumers based on limited or sparse information.

For the content-based recommender, we implemented a similar title recommendation based on TF.IDF and feature-based similar listing recommendation. In its collaborative counterpart, we mainly implemented the neighborhood method with simple weighted average and latent feature model with SVD. This report will explain the methodology that was used including all steps explained in detail and how each type of recommendation system is evaluated.

Finally, we will examine the pros and cons of these two methods and note some further work we consider important and should be followed.

2 MOTIVATION

Data is accumulated at an unprecedented speed nowadays, especially in mobile apps and websites. We know that the power of big data is uncapped - in the case of machine learning, we are able to learn from a huge mass of historical data and predict the unknown, and for this reason we are very interested in understanding how this works in practice.

Recommender systems are a typical example of how machine learning techniques can be translated into lucrative business tricks. Almost all tech giants have utilised these tools to provide a personalised search result and related recommendation. Netflix even hosts a competition for collaborative filtering algorithms (Wikipedia, 2007).

It is omnipresent in our daily life: Netflix recommends similar movies based on our watch history, Amazon identifies products that we like or need, and Spotify knows our music tastes better than ourselves.

The recommender engine not only brings users better experiences, as it can provide personalised content for consumption and filter out irrelevant information (which means a user can never get bored and can become addicted to the service), but it also garners huge advantages for tech giants: a highly customised system implies a higher probability of retention and revenue generation. Usually, at the heart of their success is their ability to drive engagement through personalised recommendations for each user. In this project, we want to delve into this and demystify this buzzword.

3 PROBLEM DEFINITION

A recommender system is an intelligent system that predicts the rating and preferences of users for different products. The primary application of recommender systems is finding a relationship between user and products in order to maximise the user-product engagement.

In reality, the recommender system is a complicated problem, and often a group of models are combined to achieve the best possible performance. There is continuous investment and research going on to improve the models. Although there are multiple models for this topic, two most prevalent ones are content-based and collaborative filtering recommenders.

The general idea behind content-based recommenders is that by relying on item metadata we can describe the similarity of two items and accordingly recommend the most similar items to a user.

For its collaborative filtering counterpart, we will not consider item similarity but rather the user preference similarity. For example, after analysing the activities of user A and B, we find they watch the same type of movies. Now, between these users, if A has seen a movie that B hasn't watched yet, that movie will get recommended to B and vice versa. In other words, this engine works based on the collaboration between similar user's preferences (thus, the name "Collaborative Filtering"). One typical application can be seen in Amazon, where users can see the "Customers who viewed this item also viewed" and "Customers who bought this item also bought" list.

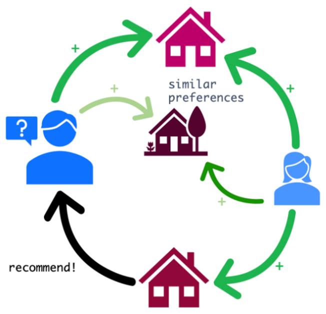


Figure 1: An illustration of user-based collaborative filtering

4 RELATED WORK

Recommender systems are an extensive and continuously expanding field.

In our research we found its major application is in suggesting related videos or music for generating a playlist for the user while they are engaged with a related item, such as Spotify and Netflix.

A similar application is in the field of e-commerce where customers are recommended with the related products, but this application involves some other techniques such as association rule learning. It is also used to recommend contents based on user behaviours on social media platforms and news websites.

However, finding and recommending many suitable items that would be liked and selected by users is always a challenge. There are many techniques used for this task.

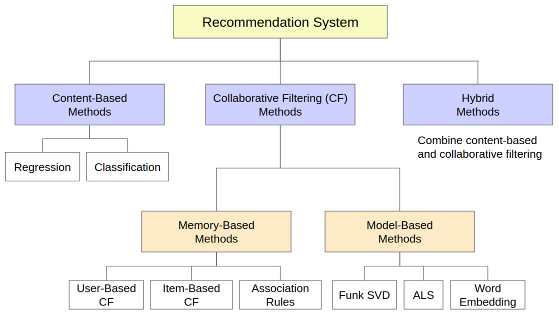


Figure 2: An illustration of most prevalent recommender methods (Arga, 2020)

4.1 Content Based Filtering

In the case of content-based methods, the information contains additional knowledge about the recommendable items, e.g., their features, metadata, category assignments, relations to other items, user-provided tags and comments, or related textual or multimedia content (Lops, P., Jannach, D., Musto, C. et al.).

These types of recommendation systems suggest similar items based on a particular item. For example, to recommend a movie to a user the system uses item metadata such as genre, director, description, actors, etc. to make these recommendations. The general idea behind these recommender systems is that if a person likes a particular item, they will also like an item that is similar to it.

In a content-based recommendation system, the value of an item i to a user u is calculated by using the values assigned by the user to other items that are similar to item i (Adomavicius, G., & Tuzhilin, A. T., 2005). The features associated with the items play an important role in forming recommendations (Pérez-Almaguer, Y. et al. , 2021).

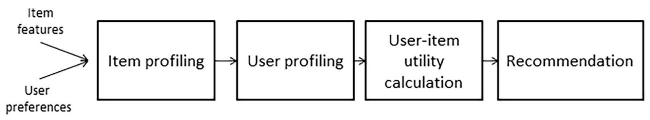


Figure3: General Scheme of Content-Based Recommendation Systems

Several approaches have been proposed:

- The creation of binary profiles to directly represent the presence/absence of a feature
- Weighting schemes taken from text-based fields
- The use of latent semantic analysis to represent textual items
- The use of tag-based profiles

4.2 Collaborative Filtering

Several conventional methods are available for collaborative filtering, and they appear in the Netflix Prize competition.

The most intuitive way to predict the score of an object given by a user is to ask other users who have scored the object. Instead of merely using the mean score of that object among all users, we consider only those users similar to the target user. This method is known as K-Nearest Neighbour, abbreviated to KNN (Paterek, 2007), which is also called memory-based.

Another type of method is one that directly finds the feature values of each user and object, and predicts the unknown scores by a prediction function using those features. Usually, such algorithms involve a matrix factorization which constructs a feature matrix for users and for objects, respectively. These kinds of algorithms include Non-Negative Matrix Factorization and Singular Value Decomposition (Bell et al, 2010), which is also called model based.

Besides the widely used algorithms described above, there are also other methods which gave good performance in the competition of the Netflix Prize. One of them uses a model called Restricted Boltzmann Machines (Salakhutdinov et al., 2007), which was performed either by itself or as a linear combination with other algorithms. Another approach is regression, which predicts the unknown scores by taking the scores in the training data as observations.

In our project, we will try to implement these two most ubiquitous algorithms and try to solve a real-world problem, listing recommendations. We notice in reality, several algorithms are deployed at the same time. ‘Other users also like this’ is supported by collaborative filtering while the recommendations popped at the bottom of housing details tend to be from content-based recommenders. Plus, we want to put two key concepts we learnt in data mining into practice, namely TF.IDF and sentiment analysis.

5 METHODOLOGIES

5.1 Content-based recommender: TF-IDF

5.1.1 Introduction.. TF-IDF is a weight factor in which a word displays importance in a document and has been calculated with a statistical method. TF-IDF method is used in a lot of domains (sentiment analysis, RS, stop words filter in etc.)

5.1.2 Term frequency. We will analyse Term Frequency (TF). We use `WordCloud` to show the frequency of column ‘name’ and ‘description’. The most frequent words occurring in the name or title of listings are: ‘Seattle’, ‘Capitol Hill’, ‘View’, ‘Home’, ‘Cozy’, etc. This obviously represents the Seattle area data with common words in room listings. Unlike for the description, here some of the tops are already specific like: house, home, apartment, living room, space. They are typical words for hosts when describing their listings.

5.1.3 TF-IDF matrix. Then we create a TF-IDF matrix of unigrams and bigrams for each id or room. The “stop words” parameter tells the TF-IDF module to ignore common English words like 'the', ‘about’, etc. TF-IDF will parse through the descriptions, identify distinct phrases in each item's description, and then find similar contents based on those phrases. Formula is:

$$W_{ij} = tf_{ij} \times \log\left(\frac{N}{df_i}\right)$$

Notation:

- tf_{ij} = number of occurrences of i in j;
- df_i= number of documents containing i;
- N = total number of documents)

5.1.4 Clean data and setup model. Firstly, we choose 6 columns which describe the room, the columns are ‘name’, ‘description’, ‘property_type’, ‘room_type’, ‘bed_type’ and ‘amenities’. These all ensure consumers know what is included in the room, and they can choose their preferences.

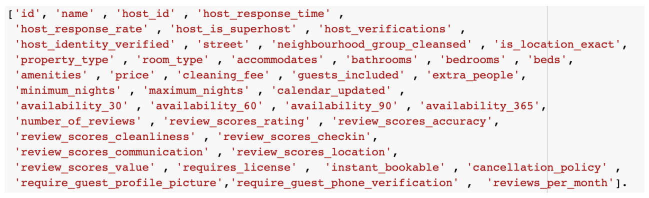
To compute the cosine similarity, we need to deal with the words in a uniform way. We convert every word into lower case letters and fill the null values with ‘null’. Then, we retain the column ‘name’, which is the room name given by hosts, and is most appealing to consumers. Other columns are the description of the room. Finally, we use combine to merge them into one column.

We can use all the listing descriptions to compute cosine similarity and comprise the cosine similarity of each room and based on the user's choice we recommend the one which has the highest cosine similarity.

5.2 Content-based recommender：k-means

5.2.1 Data Analysis and Cleaning. There are 3818 rows x 92 columns in the Seattle Airbnb listing dataset that was chosen. We have seven types of columns: url, int, str (binary and paragraph), int combined str ($100), percentage, list and NaN. Some columns such as ‘url’ cannot be used as features, and also others that already contain the same values as the other columns like ‘space’ and ‘descriptions’. Furthermore, other fields such as ‘experiences_offered’ have no effect once used as features as they only have one value for all rows. These columns were removed in this first step.

5.2.2 Processing Data. After the data cleaning process, we have the following columns left:



```
['id', 'name' , 'host_id' , 'host_response_time' ,
 'host_response_rate' , 'host_is_superhost' , 'host_verifications' ,
 'host_identity_verified' , 'street' , 'neighbourhood_group_cleansed' , 'is_location_exact',
 'property_type' , 'room_type' , 'accommodates' , 'bathrooms' , 'bedrooms' , 'beds',
 'amenities' , 'price' , 'cleaning_fee' , 'guests_included' , 'extra_people',
 'minimum_nights' , 'maximum_nights' , 'calendar_updated' ,
 'availability_30' , 'availability_60' , 'availability_90' , 'availability_365',
 'number_of_reviews' , 'review_scores_rating' , 'review_scores_accuracy',
 'review_scores_cleanliness' , 'review_scores_checkin',
 'review_scores_communication' , 'review_scores_location',
 'review_scores_value' , 'requires_license' ,  'instant_bookable' , 'cancellation_policy' ,
 'require_guest_profile_picture','require_guest_phone_verification' ,  'reviews_per_month'].
```

Figure 4: selected features for further analysis

We now consider methods to deal with the null values and convert all types of features to integers so they can be used as vectors to compute cosine similarity.

Firstly, the command drop_na is not a good way to handle null values, because we only have about 2000 rows remaining after dropping whole rows that contain ‘NaN’. Secondly, from observation of some columns, we found that the missing value itself contains some information: for example, a value for ‘host_is_superhost’ is always missing if other host related columns like ‘host_acceptance_rate’ are missing. So we can infer this host is not a superhost, and thus fill null with ‘f’.

Furthermore, for columns which contain less than 1% null values, we ignore missing values. Lastly, for integer fields which contain values that are fairly evenly distributed, we replace null values with the median or mean.

Several methods can be applied to convert non-integer type values into integers like one-hot encoding, dummy variables and factorization. To avoid high dimensionality of features and multicollinearity, we chose factorization. But for columns such as ‘amenities’ that contain lists, we used the length of the list to represent the feature.

After applying these steps, all columns were converted into integer type.

5.2.3 Dimensionality Reduction. Now we are left with forty columns which may result in overfitting when used to predict similarities, and furthermore the computation would be extremely slow. **Principal component analysis** (PCA) is the process of computing the principal components and using only these components to perform a change of basis on the underlying data.

To decide how many features should be included, we can use k-means clustering method to check how many clusters we have. Calculating the Euclidean distance between nodes to perform k-means clustering may not work if the range of column values is very large. Consequently, we use the scaler `preprocessing.MinMaxScaler()` to normalise data. We then plot the best k value for the model. We observe that the gradient becomes smaller after seven, so we chose k = 7.

In the PCA function, the component is 7 to get 7 features.

So far we can have a matrix with 3818 rows x 9 columns. Among the columns, two are indexes: ‘id’ and ‘name’, and the other seven are ‘new’ features that we have created.

5.2.4 Cosine Similarity. To calculate the similarity between items, we divide every row of the matrix with its norm and then compute the similarity between every pair of columns with the function `cosine_similarities = linear_kernel()`. We then obtain a 3818 x 3818 symmetric matrix. Lastly, we iterate through each item's similar items and store the 100 most-similar pairs.

5.2.5 Prediction. Lastly, to recommend an item for the user, one only has to input an item ID, and read the results from the dictionary, and output the score of the similar pair.

5.3 Collaborative Filtering Recommendation System

There is an alternative to content-based filtering which relies only on past user behaviors and does not require the creation of explicit profiles. This approach is known as collaborative filtering, a term coined by the developers of Tapestry, the first recommender system (Huttner, 2009). Collaborative filtering analyzes relationships between users and interdependencies among products to identify new user-item associations.

Unlike content-based recommenders where detailed attributes of listings are involved, collaborative filtering doesn't need anything else except users' historical preference on a set of items. Because it's based on historical data, the core assumption here is that the users who have agreed in the past on items tend to also agree in the future. Collaborative filtering captures the underlying pattern of interests of like-minded users and uses the choices and preferences of similar users to suggest new items.

To build a recommender based on similar users' preferences we can break down our task into two major steps:

Step 1. Find the indicator to represent similarity between users.

There are many choices to measure the similarity between users, such as Pearson correlation, cosine similarity, etc. Now that our dataset includes the review of each reviewer with respect to their individual stay, an intuitive idea is that we can gauge the user preferences by estimating how positive or negative the sentiment of their reviews, thanks to `nltk.sentiment.vader`, one of the most prevalent NLP python packages that is able to extract sentiment analysis from text-heavy, unstructured data.

One thing that should be kept in mind is that user' reviews are not always representative of users' preferences, because these subjective reviews tend to be biased, simply because users are prone to writing down something when they feel overwhelmingly positive or negative about the experience. Mostly, average users won't even bother to write down something after they leave the Airbnb accommodation.

In reality, we assume that companies should rely more on inferring users' preferences by their behaviours like their clicks or their bounce rate. However, due to lack of access to behavioural data, here we will mainly use their published reviews.

Step 2. Make recommendations based on those preferences.

Once we have translated all reviews into sentiment polarity scores, we can build our recommendation engine to predict sentiment polarity scores for all reviewer-listing pairs. And then, given a particular user, we can predict their preferences towards all items in our list and recommend the top N items with the highest sentiment scores, and assume that particular user will prefer these.

We'll therefore split our approach into three parts:

5.3.1 Extract sentiment from the original data

Based on the typical steps involved with text normalisation in the NLP pipeline as well as the constraints around Airbnb housing data, we break down our preprocessing task into following steps.

1. Drop rows with null comments
2. Drop rows where reviews contain the information 'Host cancelled this reservation...'
3. Drop those rows which are purely numeric.
4. Detect reviews that are not in English, and decide if to translate or simply to drop them based on their size.

We will keep capitalizations and punctuation because not only is it handled well with VADER , but it's also incorporated in the sentiment analysis. VADER works so well is that it adds value to words that are capitalised, that have degree modifiers (e.g. 'very'), are accompanied by punctuation (e.g. '!!!') or emoticons (e.g. ':)'), and it can even handle shifts in sentiment in a single sentence, also known as conjunctions.

After exploration, we found there are 1040 non-English records and we decided to translate them with the `googletrans` package.

Then, after the data is prepared, we are able to implement VADER. VADER is a module in the `nltk.sentiment` Python library that was specifically created to work with text produced in a social media setting, but also works well within our context. VADER is able to detect the polarity of sentiment (how positive or negative) of a given body of text when the data being analysed is unlabelled. In traditional sentiment analysis, the algorithm is given the opportunity to learn from the labelled training data. VADER uses a lexicon of sentiment-related words to determine the overall sentiment of a given body of text (Malde, 2020). Below is an example of how the lexicon is structured, with each word having a valence rating:

Word	Sentiment rating
Tragedy	-3.4
Rejoiced	2
Insane	-1.7
Disaster	-3.1
Great	3.1

Figure 5: An example of how Vader works

VADER has built this labelled lexicon using Amazon's Mechanical Turk, which is a crowdsourcing platform that pays 'crowdworkers' to perform tasks en masse, resulting in an impressively efficient method for doing this. After this step, we successfully translate all reviews into a sentiment polarity score.

Taking this one step further, we want to explore the correlation between this sentiment score and other housing attributes used above. We use Pearson correlation here and with the help of `seaborn`, we are able to visualise the correlations by a heatmap, shown below.

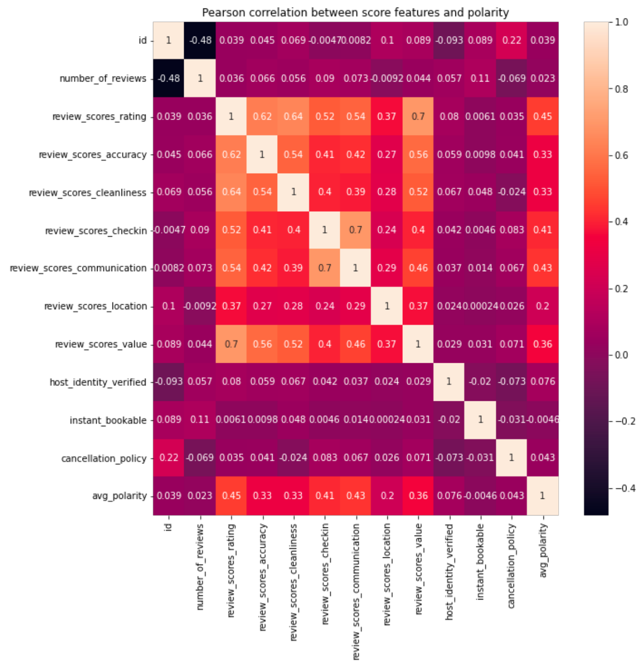


Figure 6: The heatmap describing the correlation

5.3.2 Build our recommendation engine to predict sentiment score for all reviewer-listing pairs

To predict our reviewer's polarity scores on listings they haven't visited, there are many methods in existence, among which, matrix factorization seems to yield the best results. As the Netflix Prize competition has demonstrated, matrix factorization models are superior to classic nearest-neighbour techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels (Koren et al, 2009).

The two primary areas of collaborative filtering are the **neighbourhood methods** and **latent factor models**. Neighbourhood methods are centered on computing the relationships between items or, alternatively, between users. The item oriented approach evaluates a user's preference for an item based on ratings of "neighbouring" items by the same user. A product's neighbours are other products that tend to get similar ratings when rated by the same user. For example, consider the movie Saving Private Ryan. Its neighbours might include war movies, Spielberg movies, and Tom Hanks movies, among others. To predict a particular user's rating for Saving Private Ryan, we would look for the movie's nearest neighbours that this user actually rated (Koren et al, 2009). As Figure 1 illustrates, the user-oriented approach identifies like-minded users who can complement each other's ratings.

On the other hand, the latent factor model tries to explain the ratings by characterising both items and users on, say, 20 to 100 factors inferred from the ratings patterns. For movies, the discovered factors might measure obvious dimensions such as comedy versus drama, amount of action, or orientation to children; less well-defined dimensions such as depth of character development or quirkiness; or completely uninterpretable dimensions. For users, each factor measures how much the user likes movies that score high on the corresponding movie factor. We are mainly interested in implementing latent feature models in collaborative filtering here.

We will explore the SVD technique further, as we have been exposed to it during our course. SVD is well known for its efficiency in dimensionality reduction, and at the same time it performs pretty well in matrix completion. Essentially, SVD states that any given matrix M can be factored as a product of three matrices: U, Σ, and V. U is the n x k user-latent features matrix, V is the m x k listing-latent features matrix, and Σ is a k x k diagonal matrix containing the singular values of our original matrix M, which just means that each value will represent how important a latent feature is to to predict user preferences. Using these three matrices, we can therefore recover the full M matrix of polarity values for all reviewer-listing pairs.

Before training a model, we need to split our 'ratings' data frame into two parts — a train set to train the algorithm to predict ratings and a second part to test whether the rating predicted is close to what was expected. This will help in evaluating our models.

Another prerequisite step is to create a utility matrix, where each row represents a user, each column an item. The entries of this matrix are ratings given by users to items. In this step, we used a trick by `pivot_table` to generate the matrix from origin data.

Firstly, we tried to implement a recommender based on the weighted average of similar listings as our baseline model, where we use the weighted average of the ratings and use the cosine similarity as the weights. The users who are more like the 'input_user' will have a higher weight in our rating computation for the 'input_user'. Through the weighted average method, we can achieve a 0.29 RMSE as evaluation result.

We improved on this by the help of Singular Value Decomposition (SVD). Here, we used `svds` embedded in `scipy.sparse.linalg` and took the utility matrix as its input and returned the predictions on polarity scores.

5.3.3 Make personalised recommendations

Now we will try to generate a random new user to simulate the recommendation process.

We scripted a `get_recommendations()` function, which has three arguments: utility matrix, the 'reviewer_id' to be predicted, and N which refers to how many top recommendations to be returned.

It will return top N recommended listings to a certain user ordered by polarity score.

6 EVALUATION

For predictions of content-based system we use the Precision ratio. For each list of top n recommended items, Precision is defined as the ratio between the number of recommended items that were actually preferred by the current user, and the overall number of recommended items (in this case n). The formula is:

$$Precision = \frac{|recommended\ items \cap preferred\ items|}{|recommended\ items|}$$

The performance of collaborative filtering can be measured by the error between the prediction values and the ground-truth (as we split the origin dataset into train and test parts, we are able to evaluate the test set). A common and efficient measure is Root Mean Square Error (RMSE). Consider the prediction matrix $P \in R_{n\times m}$ and the ground-truth answer matrix A $\in R_{n\times m}$. The RMSE between the prediction P and the answer A is defined as

$$RMSE = \sqrt{\frac{\sum_{i,i\in K}\left(P_{ij} - A_{ij}\right)^2}{|K|}}$$

Notation:

- K: set of all pairing user i and rating j
- P_{ij}: predicted sentiment polarity score
- A_{ij}: the actual sentiment polarity score

And from the experiments, we can clearly see that SVD performs much better than simple weighted average, scoring 0.16 (with 150 features selected) and 0.29 respectively.

7 CONCLUSION

7.1 Advantages and weaknesses

Though either content-based recommenders or its collaborative filtering counterparts is widely used in applications, there still exist many challenges faced by these two methods:

7.1.1 Content-based recommendation

Pros:

1. There is no need for data on other users, hence no sparsity problems as attributes describing a house are always available.
2. Able to recommend to users with unique tastes: Because it is based on items, even though a user has unique preferences, similar items can be found.
3. Able to recommend unpopular item.
4. Interpretability: content-based recommender is based on attributes of items, therefore it's able to provide explanations– it can list the key attributes that lead a house to be recommended.

Cons:

1. Difficulty to find appropriate features: finding features that can satisfy the topic and help solve the problem isn't always easy.
2. Cold start problems: it cannot recommend items to a new user as we have no idea about his previous preferences.
3. Overspecialization: Content-based algorithm will never recommend items outside a user's consumption profile.

7.1.2 Collaborative filtering

Pros:

1. Works for any kind of item: doesn't require features about the items or users to be known, i.e no feature selection is needed.
2. Combat overspecialization : can help recommenders to not overspecialize in a user's profile and recommend items that are completely different from what they have seen before. If you want your recommender to not suggest a pair of sneakers to someone who just bought another similar pair of sneakers, then try to add collaborative filtering to your recommender.

Cons:

1. Cold start: When a new item comes in, until it has to be rated by a substantial number of users, the model is not able to make any personalised recommendations.
2. Popularity bias: Similarly, for items that don't have much data, the model tends to give less weight to them and has a popularity bias by recommending more popular items.
3. Lack of transparency and explainability: The underlying tastes expressed by latent features are actually not interpretable because there are no content-related properties of metadata. For example, in Airbnb's case, it doesn't necessarily have to be the cleanliness of a house, it can be something like how far it is to the nearest takeaway shop or how often the host edits their profiles, and so on.
4. Sparsity: the user/review matrix is always sparse and sometimes, it is hard to find users that have rated the same items.

In a nutshell,

- Content-based model is more interpretable, we can clearly tell which features contribute more to a positive remark.
- A major appeal of collaborative filtering is that it is domain free, yet it can address data aspects that are often elusive and difficult to profile using content filtering.
- While generally more accurate than content-based techniques, collaborative filtering suffers from what is called the cold start problem, due to its inability to address the system's new products and users. In this aspect, content filtering is superior.

8 FUTURE WORK

There is a long list of variants of the recommender system: collaborative filtering with graph-based implicit feedback, neighborhood-based filtering, The SVD technique was introduced into the recommendation system domain by Brandyn Webb, much more famously known as Simon Funk during the Netflix Prize challenge. Here we aren't doing Funk's iterative version of SVD or FunkSVD, which is not an exhaustive list. We would love to expand more on other topics in the future

Meanwhile, while every type of recommender algorithm has its own list of pros and cons, it's usually a hybrid recommender that comes to the rescue.

Hybrid recommender: To overcome problems encountered in using pure content based or collaborative filtering systems, such as cold-start or sparsity problems, we can attempt to merge the two approaches. This has been shown to be more effective at recommending items to users (Lenhart, 2016), as it returned a more diverse range of options than the content-based approach, and also displays more new listings than the pure collaborative approach. The benefits of multiple algorithms working together or in a pipeline can help reach more accurate recommenders. In fact, the solution of the winner of the Netflix prize (Wikipedia, 2019) was also a complex mix of multiple algorithms.

Several simple techniques could be chosen:

- Weighted average
- Switching - choosing between either approach
- Mixed - show listings from both

More complex techniques could also be considered:

- Feature combination from both systems
- Cascade - Give one recommender priority, with the second one used when several listings are given the same score.

Apart from the possible improvements on models, we can try other evaluation measurements, for example business-related metrics such as A/B tests to validate any improvements in the in-app experience and whether the number of active users increases daily, in order to better evaluate business objectives and whether the recommendation systems are getting better at recommending listings to Airbnb users.

Meanwhile, solving cold start problems is a bottleneck to most recommendation algorithms. Some research indicates that assigning lower constraints to the latent factors associated with the items or users that have more information and setting higher constraints to the others can partially remedy the problem (Chen, 2019).

REFERENCES

[1] Paterek, Arkadiusz. "Improving regularized singular value decomposition for collaborative filtering." (2007).

[2] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In Proceedings of the 24th international conference on Machine learning (ICML '07). Association for Computing Machinery, New York, NY, USA, 791–798. DOI:https://doi.org/10.1145/1273496.1273596

[3] Lops, P., Jannach, D., Musto, C. et al. Trends in content-based recommendation. User Model User-Adap Inter 29, 239–249 (2019).

[4] Adomavicius, G., & Tuzhilin, A. T. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17, 734–749

[5] Pérez-Almaguer, Y. et al. (2021). Content-based group recommender systems: A general taxonomy and further improvements. Expert systems with applications. [Online] 184115444–.

[6]c Arga. "Recommendation System : Matrix Factorization with Funk SVD." Rstudio-Pubs-Static.s3.Amazonaws.com, 20 Sept. 2020, rstudio-pubs-static.s3.amazonaws.com/668748_d374e1a7f0584c74af50cf7b0aac79ca.html#1_Introduction. Accessed 26 Dec. 2021.

[7] Bell, Robert M., et al. "All Together Now: A Perspective on the Netflix Prize." CHANCE, vol. 23, no. 1, Jan. 2010, pp. 24–29, 10.1080/09332480.2010.10739787..

[8] Chen, Hung-Hsuan, and Pu Chen. "Differentiating Regularization Weights -- a Simple Mechanism to Alleviate Cold Start in Recommender Systems." ACM Transactions on Knowledge Discovery from Data, vol. 13, no. 1, 29 Jan. 2019, pp. 1–22, 10.1145/3285954. Accessed 26 Dec. 2021

[9] Huttner, Joseph. From Tapestry to SVD: A Survey of the Algorithms That Power Recommender Systems. 8 May 2009..

[10] Koren, Yehuda, et al. "Matrix Factorization Techniques for Recommender Systems." Computer, vol. 42, no. 8, Aug. 2009, pp. 30–37, 10.1109/mc.2009.263.

[11] Lenhart, Philip, and Daniel Herzog. "Combining Content-based and Collaborative Filtering for Personalized Sports News Recommendations." CBRecSys@ RecSys. 2016.

[12] Malde, Ravi. "A Short Introduction to VADER." Medium, 6 July 2020, towardsdatascience.com/an-short-introduction-to-vader-3f3860208d53.

[13] qutbuddin. "An Exhaustive List of Methods to Evaluate Recommender Systems." Medium, 2 Oct. 2020, towardsdatascience.com/an-exhaustive-list-of-methods-to-evaluate-recommender-systems-a70c05e121de.

[14] Wikipedia Contributors. "Netflix Prize." Wikipedia, Wikimedia Foundation, 24 Sept. 2019, en.wikipedia.org/wiki/Netflix_prize.