

Recommender system

**Shopify App Store**



Presented by EBAC5002 Team Zoom:

Di Boyu | A0195329U

Li Xinxin | A0149193X

Tan Yong Ern Paul | A0195389H

Wang Ruochen | A0087653U

Wang Shumin | A0195334B

Zhong Xin | A0195276R

Tale of Contents

[1 Business Context 2](#_Toc23454790)

[1.1 Background 2](#_Toc23454791)

[1.2 Business problem 2](#_Toc23454792)

[2 Data Exploration 3](#_Toc23454793)

[2.1 Variable Identification 3](#_Toc23454794)

[2.2 Statistics 4](#_Toc23454795)

[2.3 Missing Value Exploration 5](#_Toc23454796)

[2.4 Textual Data Exploration 5](#_Toc23454797)

[3 Recommender System Modeling 6](#_Toc23454798)

[3.1 Content-Based Recommender 6](#_Toc23454799)

[*3.1.1 Content Analyzer 6*](#_Toc23454800)

[*3.1.2 Profile Learner 10*](#_Toc23454801)

[*3.1.3 Filtering Component 11*](#_Toc23454802)

[*3.1.4 Recommendation Results 11*](#_Toc23454803)

[3.2 Collaborative Filtering 13](#_Toc23454804)

[*3.2.1 Data preparation 13*](#_Toc23454805)

[*3.2.2 User-based System 13*](#_Toc23454806)

[*3.2.3 Item-based System 17*](#_Toc23454807)

[*3.2.4 Model-based System 19*](#_Toc23454808)

[*3.2.5 Method Comparison 21*](#_Toc23454809)

[3.3 Hybrid System 22](#_Toc23454810)

[4 Conclusion 23](#_Toc23454811)

[4.1 Limitation 23](#_Toc23454812)

[4.2 Further Improvement 23](#_Toc23454813)

# Business Context

## Background

Shopify is an e-commerce company that offers online retailers a suite of services to simplify the process of running an online store through multiple channels. The suite of services helps entrepreneurs to design their own online stores by providing pre-defined functions and customized applications.

In order to provide store owners with more diversified and customized functionalities to improve their online store, Shopify introduced the Shopify App Store to allow developers post/sell add-on features to store owners. It is a platform in which store owners can download appropriate extensions to beautify their online stores or adding features to ensure smooth shopping experiences to their customers.

Store owners on the Shopify can browse the Shopify App Store, listed with application information from developers and user reviews and ratings which were continuously gathered.

As a platform, Shopify would charge $29/month from store owners to access basic functions on Shopify to support the online store operations. The store owners can pay $79-229/month, if the store owner want to unlock more advanced functions or apps on Shopify.

## Business problem

As Shopify supports an estimated 800,000 potential businesses and the App Store has a wide range of more than 3000 applications of varies quality and usability, ensuring a positive and satisfied customer experience becomes a crucial problem for Shopify. With increasing numbers of application in the App store, it becomes time consuming to search for the desired application through the standard searching flow. Therefore, a need to recommend the most appropriate applications to Shopify users has raised.

A recommender system is essential as it would recommend appropriate apps to every store owner based on their preferences to improve the user experience of existing store owners to boost the retention rate and revenue as well as attracting new customers to open online stores on Shopify.

# Data Exploration

As Shopify App Store has existed for 10 years and it gathers information from various perspectives, the data is sufficient to build an app recommender system. The dataset, we are using to build the recommender system, consists of 2 parts: application information and user reviews. Below shows the data exploration approaches in preparation to feed the recommender model development.

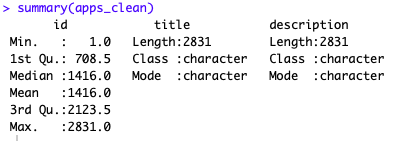
## Variables Identification

This section shows the data dictionary for the datasets, Application Information and User Reviews According to the assignment purpose, we only keep relevant information by dropping unnecessary columns:

1. Application Information

|  |  |  |
| --- | --- | --- |
| Variable Name | Description | Data Type |
| id | The unique ID of each application | Integer |
| title | The title of each application | Character |
| description | A brief description of each application, containing the features and functionality of each application | Character |

*Table 1. Application Information*

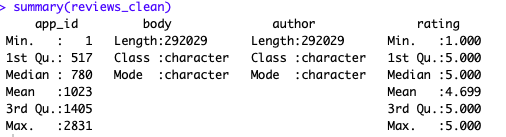


*Figure 1. Application Information*

1. User Reviews

|  |  |  |
| --- | --- | --- |
| Variable Name | Description | Type |
| app\_id | The unique ID of each application | Integer |
| author | Name of user who provides reviews, 1 author could rate multiple applications | Character |
| body | Reviews related to each application by individual author | Character |
| rating | Author’s rates on an application. Scale from 1 to 5 | Integer |

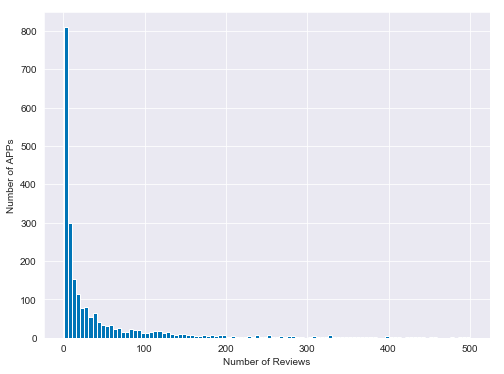
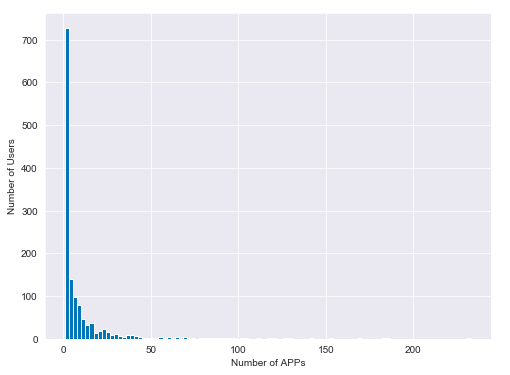
*Table 2. User Reviews*



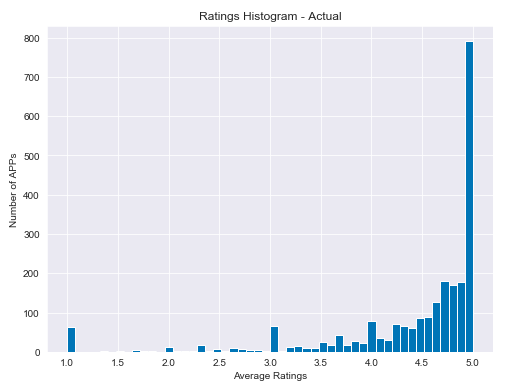
*Figure 2. User Reviews*

## Statistics

Shopify dataset contains 2831 unique applications with 292029 reviews gathered from 191782 unique authors. As shown in histograms below, most users rated less than 5 apps and most apps were reviewed by less than 100 unique users.

*Figure 3. Distribution of rating counts from users on apps Figure 4. Distribution of review counts on apps*

The app ratings have a range from 1 to 5. The histogram below shows the distribution of average ratings for each app. Most apps have received an average rating greater than 4 and a few apps have received between 1 and 3 average ratings.



*Figure 5. Distribution of ratings*

Upon normalization of each user’s ratings for User-based Collaborative Filtering model, the average normalized ratings after subtracting each user’s mean ratings from the individual ratings have a left-skewed distribution with most ratings equal to zero. The reason for this phenomenon is that most users rated the APPs with same ratings without too much deviations. Therefore, the normalization of individual ratings for each user is not applied to the subsequent Collaborative Filtering model because it is not necessary.

## Missing Value Exploration

Looking at the summary of each dataset, we found there are several values missing from the author column of the reviews table. It shows that the platform allows users to leave anonymous reviews. We have decided to drop these reviews created by anonymous authors as it is meaningless if we recommend apps to someone we cannot identify.

## Textual Data Exploration

As Shopify is well-known worldwide, we expect multiple languages to be found in text feeds. After examining description attribute in Product information, we found the data set is multi-linguistic containing 10 languages where English accounts for 99% of total observations. The words from these languages will be removed when doing text mining as we are not able to analysis non-English text.





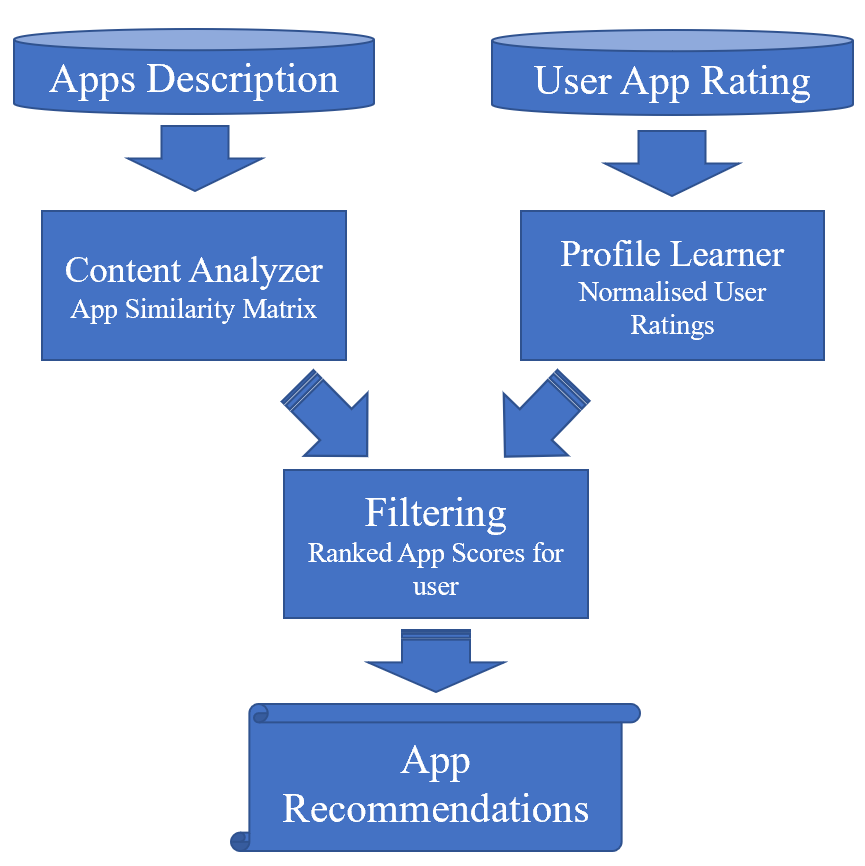
*Figure 6. Reviews’ language variations*

# Recommender System Modeling

## Content-Based Recommender

One of the approaches we can take to build an app recommender system is the content-based approach, by recommending similar apps based on the apps which the user have used and liked. To build a content-based recommendation system, our 3 main components:

1. Content Analyzer: to find similarity between apps.
2. Profiler Learner: to find which apps the user used and liked.
3. Filtering Component: to combine the app similarity and the user profile and rank potentially interesting apps to the user.



*Figure 7. Content-based Recommender Architecture*

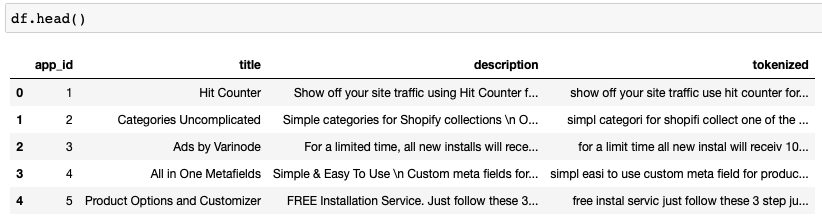
### Content Analyzer

The Content Analyzer should give us a structured representation of each app in which we can use for comparisons between apps. To build the content analyzer, we must extract features of each app which we can measure similarity between apps. In our dataset, quantitative product features are few and too generalized. We decided to use text mining techniques on the product description of the apps to build the vector space model to represent the apps which we can use to calculate the similarity score between apps.

Firstly, we used text pre-processing techniques used on the app descriptions dataset to build the corpus for indexing:

* 1. Uni-gram tokenization
  2. Stop words removal
  3. English words from the tm package standard list
  4. Non-English words from other languages
  5. Punctuation
  6. Numbers
  7. Additional spacing
  8. Escape sequences such as \n
  9. Domain-related words commonly appear in the app descriptions such as ‘app’, ’shopifi’, ‘store’, ‘product’ etc.
  10. Text normalization

1. Convert to lower case
2. Stemming



*Figure 8. data exploration after text preprocessing on apps’ descriptions*

Below is the word cloud of the final corpus of all the apps’ description after completing text pre-processing.



*Figure 9. Word cloud based on apps’ descriptions*

To measure the similarity between apps, TF-IDF weighting is used in the term vector model and to calculate the cosine similarity score between apps. The content analyzer will be represented by a matrix of the similarity score between apps.

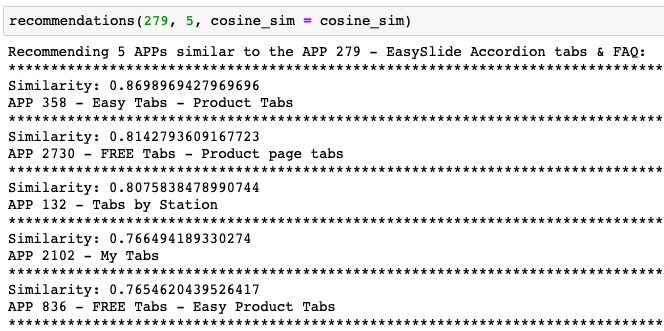
Why choosing TF-IDF?

TF-IDF is a simple and efficient weighting method which can bring out words that are more unique among documents. This results in document terms that are more relevant and can better describe the document given a higher weightage.

Why choosing cosine similarity?

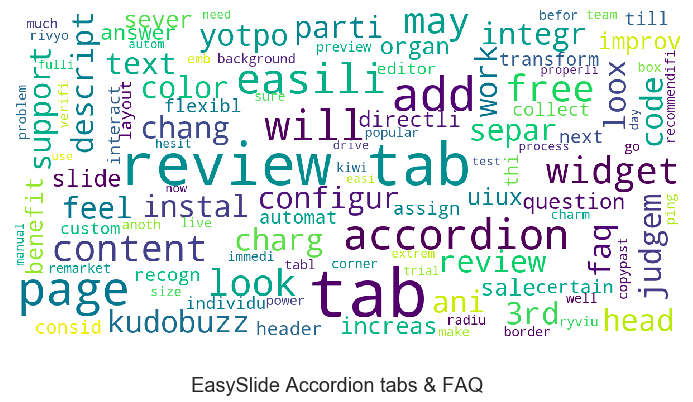
Cosine similarity emphasize on the difference in angle between vectors rather than magnitude of the vectors. This allows documents with similar document term profiles to have similar score without taking the word frequency into consideration, make it ideal for comparing similarity content between application description.

With the apps’ similarity matrix, we created a function to calculate the similarity scores between the selected app and the rest apps based on the app descriptions. Taking the app called *“Easy Slide Accordion tabs & FAQ”* with App ID 279 as an example, the 5 most similar apps with it are:



*Figure 11. Top 5 similar apps for app: Easyslid Accordion tabs & FAQ (app\_id: 279)*

To test whether the recommendations make sense, we’ve built the word cloud for the selected app and each recommended app to visualize whether the apps are truly similar to the selected app. The result below shows these apps look quite similar to the selected app.



*Figure 11. Word cloud for selected app ‘easyslide’ Accordion tabs & FAQ*:

|  |  |  |  |
| --- | --- | --- | --- |
| Rank | Similarity Score | App ID and Title | Word Cloud |
| 1 | 0.870 | APP 358  Easy Tabs ‑ Product Tabs |  |
| 2 | 0.814 | APP 2730  FREE Tabs ‑ Product page tabs |  |
| 3 | 0.808 | APP 132  Tabs by Station |  |
| 4 | 0.766 | APP 2102  My Tabs |  |
| 5 | 0.765 | APP 836  FREE Tabs ‑ Easy Product Tabs |  |

*Table 3. Word cloud of top 5 similar apps to selected app ‘easyslide’*

### Profile Learner

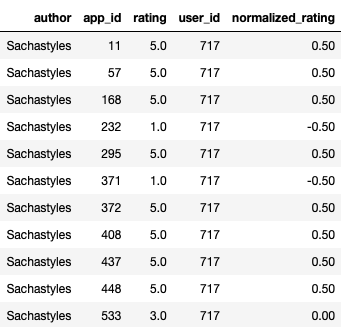
The Profile Learner should give us a user profile representation of users’ app liking. To build our Profiler Learner, we are using explicit feedback, users’ rating on the apps. The users’ rating data will give us a good measurement of which apps the users have used and liked. The rating range is from 1 to 5 and we normalized the actual rating first to a range from -0.5 to 0.5 so that Apps with ratings above 3 have positive normalized rating while Apps with ratings below 3 have negative normalized rating.

Thus, the actual ratings are normalized to:

|  |  |
| --- | --- |
| Actual Ratings | Normalized Ratings |
| 1 | -0.5 |
| 2 | -0.25 |
| 3 | 0 |
| 4 | 0.25 |
| 5 | 0.5 |

*Table 4. Relation between normalized rating and actual rating*

The user profiles are represented by a list of apps the users have rated positively with the normalized rating. As long as the user have given an app a rating of 3 and above that results in a positive normalized rating, we are able to build the user’s profile.



*Figure 12. Users’ profile for sample user ‘Sachastyles’*

### Filtering Component

With the app similarity matrix from the Content Analyzer and the user profiles from the Profile Learner, we will combine them to calculate the app recommendation score for the user.

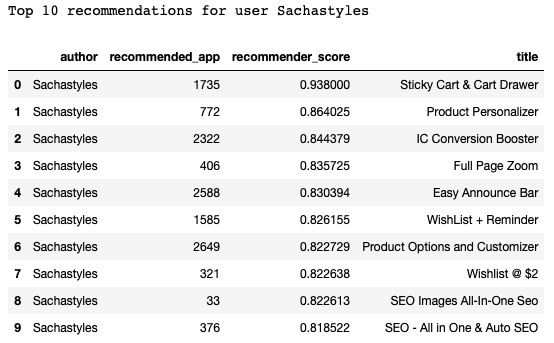
Multiplying a weighted component using the normalized rating to the matrix of each rated apps’ similarity matrix with the other apps, we can calculate the recommendation scores for all the apps for each user.

After removing the apps that the user has rated, the normalized recommendation scores are sorted in descending order and the top apps are recommended.

### Recommendation Results

The outcome from the development of the 3 components is a recommender system that can give application recommendation based on application description which the user has rated positively in the past. As long as the user has rated positively at least 1 application, this content-based recommender will give its top recommendations.

Recommendations of top 10 APPs for user – Sachastyles:



*Figure 14. Recommendation of top 10 APPs for user – Sachastyles*

## Collaborative Filtering

### Data preparation

The target dataset (Shopify’s review data) were loaded in and anonymous users were dropped from the data.

A close up of a logo

Description automatically generated

*Figure 14. Drop anonymous users from reviews data*

The reviews data were aggregated by users to select the active users, as the active users are defined by giving 10 or more comments. After that, the reviews data were aggregated by app id, and apps which have 10 or more reviews are chosen to be active apps. The final reviews data used to generate rating matrix were the overlap part of reviews with active users and active apps. There were 13849 reviews being selected.

A screenshot of a cell phone

Description automatically generated

*Figure 15. Data exploration for reviews’ data*

The rating matrix with 977 users and 1100 apps were generated with fill rate at 1.29%, and 200 users were selected as test users for later user-based system and item-based system approaches.

A screenshot of a cell phone

Description automatically generated

*Figure 16. Rating matrix generated from reviews’ data*

After conducted normalization test, we found out normalization is not preferred in this case. As shown in the report’s statistics part, the rating distribution is highly left skewed, which means most users tend to give rating at only 4 or 5 when they liked any app, these ratings would fall near to 0 after normalization, which indicates neutral rather than favorable. For example, 67% of the users who gave single rating of 5 will be indicated as neutral after normalization. Therefore, normalization is not applied to the subsequent Collaborative Filtering model.

A screenshot of a cell phone

Description automatically generated

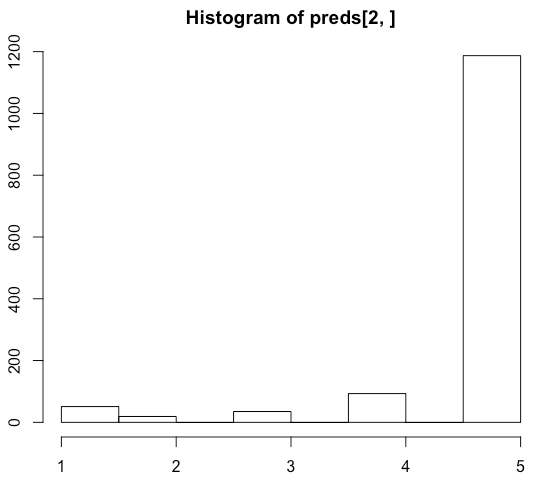
*Figure 17. Distribution of ratings after normalization*

### User-based System

User-based collaborative filtering approach is recommending apps by finding similar users to the target active user. It finds active user’s K-nearest neighbors (KNN) and measures the distance between each pair of users first, followed by predicts the rating of unused apps based on other similar users’ preferences.

In Shopify’s reviews data, we followed 80/20 rules to split train and test data into 777 train users and 200 test users, with 200 test applications. We performed 3 ways in calculating similarity matrix between users and found the best approach by comparing error terms and precision/recall/accuracy with defined threshold.

To identify the threshold, firstly let’s look at the existing rating distribution. According to the histogram shown below, we found the distribution is highly left skewed, which means the Shopify users tend to give high ratings for apps, so the threshold should be set higher than normal. Thus, we set the threshold to be 4.5 and comparing different methods using MAE and confusion matrix.

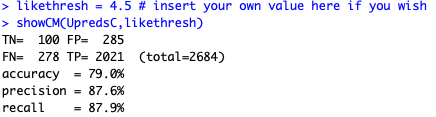


*Figure 18. Distribution of original ratings*

* Cosine Similarity



*Figure 19. MAE result of Cosine Sim*



*Figure 20. Confusion matrix of Cosine Sim*

A close up of a map

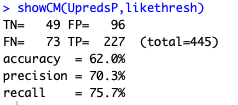
Description automatically generated

*Figure 21. ROC curve of Cosine Sim*

* Pearson Similarity



*Figure 22. MAE result of Pearson Sim*



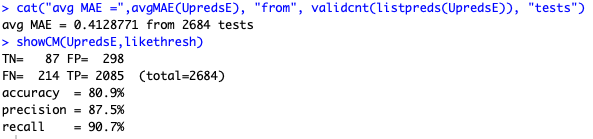
*Figure 23. Confusion matrix of Pearson Sim*

A close up of a map

Description automatically generated

*Figure 24. ROC curve of Pearson Sim*

* Euclidean Similarity



*Figure 25. MAE result and Confusion matrix of Euclidean Sim*

A close up of a map

Description automatically generated

*Figure 26. ROC curve of Euclidean Sim*

* Comparison of Different Similarity Measures

We used MAE, Accuracy, Precision, Recall and AUC to measure the performance of the three similarity measures, where Accuracy, Precision and Recall were derived from the confusion matrices.

|  |  |  |
| --- | --- | --- |
|  | Predictions | |
| Actual | Won’t Like | Will Like |
| Rated Poor | TN | FP |
| Rated High | FN | TP |

*Table 5. Confusion matrix*

* + TN (True Negative): The apps we predict customers would not like are also rated negatively by customers
  + FP (False Positive): The apps we predict customers would like but actually rated negatively by customers
  + FN (False Negative): The apps we predict customers would not like but actually rated positive by the customers
  + TP (True Positive): The apps we predict customers would like are also rated positive by the customers

The definitions of Accuracy, Precision and Recall are:

To summarize the performance between predictive models using different similarity matrices, we created a comparison table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Similarity Method | MAE | Accuracy | Precision | Recall | AUC |
| Cosine Similarity | 0.437 | 79% | 87.6% | 87.9% | 0.595 |
| Pearson Similarity | 0.784 | 62% | 70.3% | 75.7% | 0.561 |
| Euclidean Similarity | 0.413 | 80.9% | 87.5% | 90.7% | 0.605 |

*Table 6. Comparison of similarity methods for User-based model*

In this case, to give customers better user experience as recommending some apps the user actually dislikes would be less tolerable for customers, we prefer to minimize False Positive and optimize the Precision.

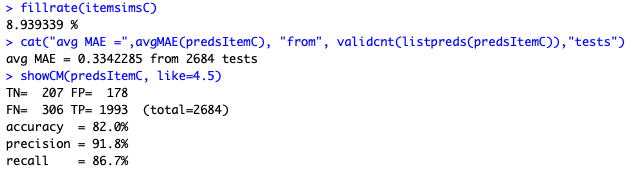
As shown in above table, both Cosine and Euclidean methods had similar performance metrics. Thus, both Cosine and Euclidean similarity measures could be used to build the User-based model.

### Item-based System

Instead of finding similar users on-line, Item-based collaborative filtering pre-calculates similarities between any two applications and then recommend nearby items according to target user’s downloaded apps.

Similar to User-based approach, we divided data into 80% training and 20% test data set to validate the prediction. 3 ways are performed to compute item-item similarity matrix for comparison.

* Cosine Similarity



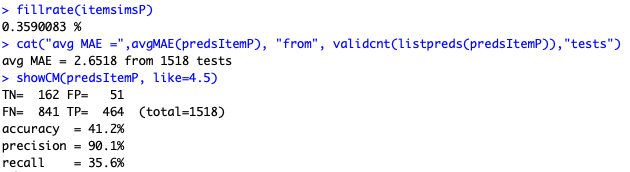
*Figure 27. MAE result and Confusion matrix of Cosine Sim*

A close up of a map

Description automatically generated

*Figure 28. ROC curve of Cosine Sim*

* Pearson Similarity



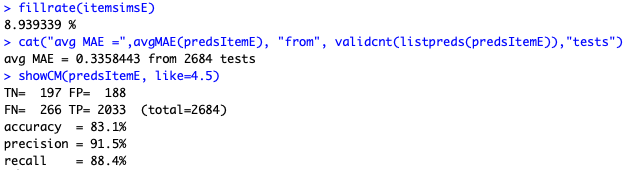
*Figure 29. MAE result and Confusion matrix of Pearson Sim*

A close up of a map

Description automatically generated

*Figure 30. ROC curve of Pearson Sim*

* Euclidean Similarity



*Figure 31. MAE result and Confusion matrix of Euclidean Sim*

A close up of a map

Description automatically generated

*Figure 32. ROC curve of Euclidean Sim*

* Comparison of Different Similarity Measures

To summarize the performance between prediction model using different item-item similarity matrices, we created a comparison table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Similarity Method | MAE | Accuracy | Precision | Recall | AUC |
| Cosine Similarity | 0.334 | 82% | 91.8% | 86.7% | 0.816 |
| Pearson Similarity | 2.652 | 41.2% | 90.1% | 35.6% | 0.530 |
| Euclidean Similarity | 0.336 | 83.1% | 91.5% | 88.4% | 0.780 |

*Table 7. Comparison between similarity methods for Item-based model*

As shown in the table, Cosine similarity gave the best prediction performance with highest Precision and lowest MAE while having relatively high Accuracy, so Cosine Similarity should be chosen in this case as well.

### Model-based System

We decide to apply model-based approach as the sparsity of rating matrix is high and matrix factorization with Alternative Least Square (ALS) algorithm solves data sparseness problem very well.

RecoSystem in R only takes index into ALS model, so firstly we assign index to author. Secondly, we split reviews data into training (90%) and testing (10%). Thirdly, we load the data into RecoSystem input format and run the model. Different parameters for number of latent variables and learning rate are set for model optimization. After finding the most suitable model, we predict the ratings by multiple user and application matrixes. Lastly, confusion matrix and MAE were used to evaluate the prediction.

* Set user index



*Figure 33. Set user index*

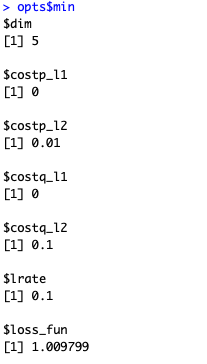
* Split data and feed into RecoSystem



*Figure 34. Train test split*

* Parameters selection by choosing smallest loss function value. Latent variables are set to be 5,10,15,20,25,30 and 35 with learning rate between 0.1 and 0.2.

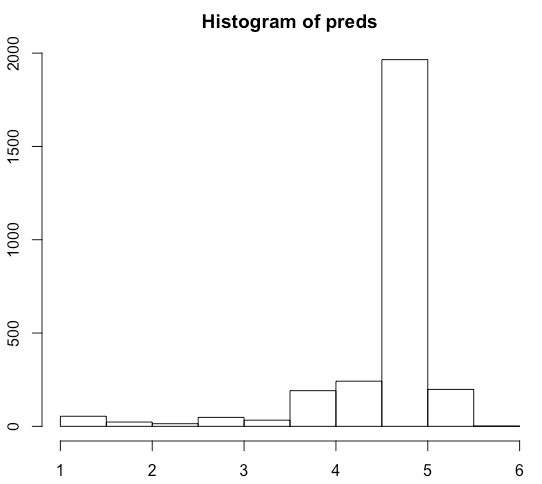
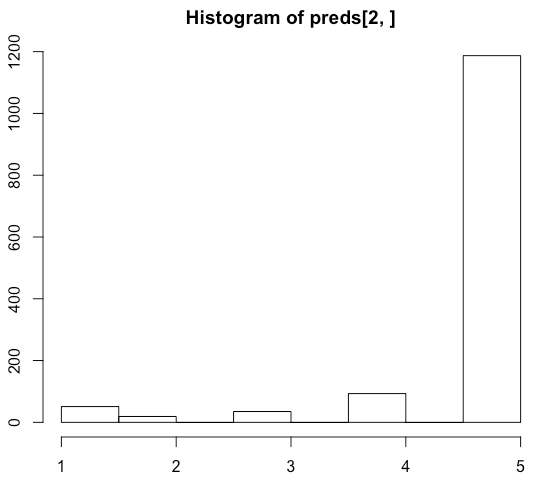


*Figure 35. Fine tuning for model*

The left chart shows the optimal combination of ALS model with minimal loss function. The best model shows 5 latent variables and 0.1 learning rate, in addition, user matrix and app matrix are built with 0.01 and 0.1 penalty term respectively to avoid overfitting.

*Figure 36. Output the best parameter*

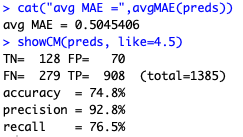
* Generate user and application matrix and predict ratings by multiply 2 matrixes together.

*Figure 37.1 Distribution of Predicted Ratings* *Figure 37.2 Distribution of Original Ratings*

As shown in above charts, the distribution trend is similar between predicted and original ratings. With large number of ratings falls between 4-5, we could still use same threshold (4.5) in other CF methods to compute confusion matrix.

* Evaluates the model by looking at MAE and confusion matrix



*Figure 38. MAE result and Confusion matrix of model-based approach*

A close up of a map

Description automatically generated

*Figure 39. ROC curve of model-based approach*

### Method Comparison

In Collaborative Filtering, we performed user-based, item-based and model-based approach to predict ratings for active users in Shopify. Since all of them used same set of data, we could compare error terms and confusion matrix to evaluate the performance. As cosine similarity matrix is the most suitable method in memory-based system, we will only use cosine similarity in user and item based for comparison.

Below shows the statistic from 3 systems:

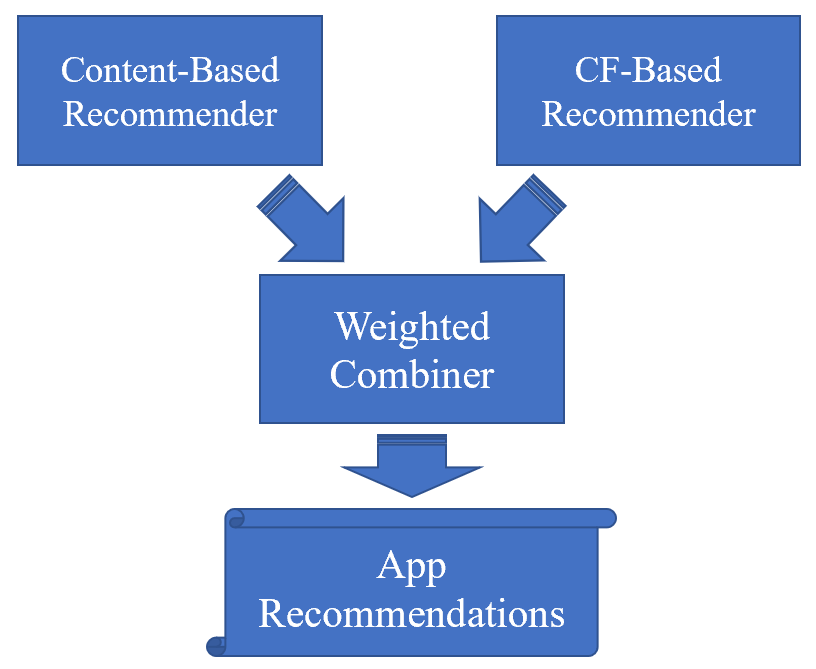
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Approach | Method | MAE | Precision | Recall | AUC |
| User-based System | Cosine Similarity | 0.437 | 87.6% | 87.9% | 0.605 |
| Item-based System | Cosine Similarity | 0.334 | 91.8% | 86.7% | 0.816 |
| Model-based System | Alternative Least Square | 0.505 | 92.8% | 76.5% | 0.822 |

*Table 8. Comparison of different systems*

Item-based approach provides the highest Precision and smallest MAE, therefore it is most suitable for Shopify to recommend most relevant apps to target users. Model-based system could be used if new users joined without existing ratings or if Shopify wants to ensure diversity in recommendation.

## Hybrid System

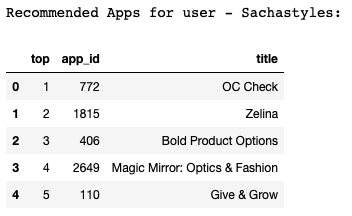
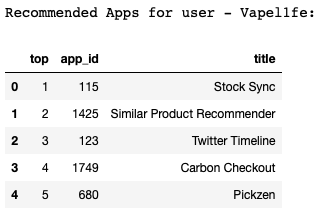
A hybrid recommender system can combine different systems to generate better recommendations. For the hybrid system, we will combine the previously developed content-based recommender and CF-based recommender using a weighted combiner. By giving weighted percentage to content-based model and CF-based ALS model, the system calculates a new rating on individual item for specific users.



*Figure 40. Hybrid Recommender System Architecture*

In our hybrid system, we combined scores from content-based model and predicted ratings from ALS model to derive new ratings with equal weighted factor (0.5). The calculated recommendation scores are normalized to the range from 0 to 1 for both content-based and collaborative filtering approach, and then the normalization rating would be substitute in following equation to predict for final score:

Test on recommendations for two users:



*Figure 42. App Recommendations for User Vapel1fe. Figure 43. App Recommendations for User Sachastyles*

# Conclusion

To summarize, we found out hybrid model would be the most suitable recommender system to solve our business problem. As the hybrid model is the combination of both content-based and collaborative filtering, it can take advantage of strengths of both models. The content-based model can recommend items for new users, and it performs better on sparse rating matrix compared with collaborative filtering, while the collaborative filtering approach doesn’t require user and item profile like content-based model, and it can achieve higher accuracy with user-item interaction increment. Thus, the hybrid model can achieve the optimized performance compared using either content-based or collaborative filtering approach alone.

## Limitation

When we are using content-based model to make recommendation for a user, if all the ratings given by the user are lower than 3 which would have negative normalized ratings, our current approach is to recommend the apps which are the least similar to the apps that the user dislike. However, we cannot guarantee that user would like the app which is not similar to the app user rated poor. Besides, for content-based approach, TF-IDF is used for text analytics, which is based on the bag-of-words (BoW) model, so it does not capture position in text, semantics, co-occurrences in different documents, etc.

In addition, the common shortcoming for collaborative filtering and hybrid model is that recommending for a new user would still be difficult. Since if there is no historical rating given from the user, it would be hard for system to understand his/her preference.

## Further Improvement

For the new user and the user gives rating which are all lower than 2, we may recommend them the most popular 10 apps among all existing users. After discovered these users’ preferences, we may use our model to conduct further recommendation for these users. Besides, currently we are assigning equal weightage on content-based and collaborative filtering approach when building hybrid model. We may adjust the weightage distribution continuously based on the users, apps and reviews variation to achieve the optimized result when using the hybrid model.