# Hybrid Recommendation Systems for Rental Companies

Anna Dai, Sarwari Das, Tigran Harutyunyan, Surabhi Trivedi

Social Science Research Institute
Duke University, Durham, US

## Abstract

Recommendation engines (RE) are widely used in the e-commerce fashion industry. However, most REs suffer from the cold start problem (where users have no prior shopping history) and the data sparsity problem (lack of user and item ratings). In this paper, we propose a novel method, a combination of clustering, supervised learning models, and Cosine Matrix Factorization (CosMF) to address these problems, specifically in the online fashion rental industry. With this approach we found that clustering the customer base on the basis of size and then recommending items within those clusters while accounting for cold start and data sparsity with CosMF (particularly for the medium cluster) gave us the best results.

## Introduction

This paper outlines a recommendation engine built for a clothing rental website called Rent The Runway. It is an e-commerce platform that allows users to rent, subscribe, or buy designer apparel and accessories. With this paper, we aim to come up with a recommendation system combining matrix factorization with cosine similarity that addresses one of the most common problems found in recommendation engines: the cold start problem and data sparsity. In addition, we sought to examine if market segmentation could help our recommender's performance.

## Background

Recommender systems (RS) are software tools for generating and providing suggestions for a better user experience by exploiting a variety of strategies [1]. Technically all RSs employ one or more recommendation strategies such as Content-Based Filtering (CBF), Collaborative Filtering (CF), Demographic Filtering (DF), Knowledge-Based Filtering (KBF), etc. Below I briefly describe these strategies:

**Collaborative Filtering**: The oldest definition of collaborative filtering is "collaboration between people to help one another perform filtering by recording their reactions to documents they read" [2]. In other words, this filtering approach assumes that people who had same taste in the past will have the same taste in the future. This is leveraged by using information such as ratings or other user generated feedback to recognize commonalities between different groups of users and then give recommendations based on these inter-user similarities [3]. However, this approach suffers from the "cold-start" (new users) and the "gray sheep" (users that can't be grouped in any cluster) problem.

**Content-based Filtering**: This approach relies on the assumption that users who liked items with certain features/ attributes in the past will like similar items in the future. But similar to CF, this approach also suffers from the cold-start problem.

**Knowledge-based Filtering**: This RS uses knowledge about users and products to pursue a knowledge-based approach to reason what items meet the user's requirements [4]. One advantage of using this recommendation strategy is that it avoids the ramp-up problem which is very frequently

encountered in CBF and CF approaches. This problem stems from the fact that both these approaches have to be initialized with large amounts of data for them to work [4]. However, KBF is not able to generate personalized recommendations and capture latent patterns in the user's preferences and habits.

To address problems like cold start, ramp-up, and gray sheep, hybrid recommender systems came into the picture. Hybrid Recommendation Systems put together two or more of the strategies with the goal of reinforcing their advantages and reducing their disadvantages or limitations.

Specific to the online fashion industry, the most commonly used recommendation system is knowledge-based filtering which prompts users to select their preferences from a given set of item attributes such as color, clothing type, size, occasion, etc. However, this systems fails to provide any personalized recommendations.
Along with segmenting our customer base into clusters via agglomerative and K-means clustering and using supervised learning methods for content based filtering we use neural network matrix factorization for collaborative filtering. Matrix factorization helps to overcome issues such as data sparsity and high dimensionality [7] [9]. They are also superior to class recommendation algorithms such as nearest-neighbours as they are able to incorporate implicit feedback, temporal effects and confidence intervals [8] [10] [11].

In matrix factorization, a high correspondence between a user and an item will lead to a recommendation of that particular item to the user [12]. There are a number of ways to calculate the aforementioned correspondence between users and items. In this paper, we use cosine similarity to calculate the correspondence. Following is a brief discussion on why matrix factorization fails to address the data sparsity problem and can benefit from using cosine similarity as a correspondence method instead of a basic inner product or using auxiliary datasets [13]

One way to solve data sparsity is to get auxiliary data using social recommendation or transfer learning [13]. But this approach still has some major drawbacks such as unavailability of auxiliary datasets, available auxiliary datasets may have noise or have different assumptions which could be more harmful than useful [14], and/ or need for fine-tuning between the balance between the target and auxiliary dataset [15]. Instead, in this paper we combine the commonly used cosine similarity with matrix factorization to make the best use of the available dataset.

# Data

### Data sources, cleaning and feature engineering

Data for the project comes from research on recommendation engines at UCSD, and can be accessed by following this link. The data space contains 192,544 rows of unique transactions, and 15 columns with customer metadata, customer rental experience, and rented item details. Each transaction contains a review date, which we treat as a rental date, ranging from April 1, 2011 to September 9, 2017. The dataset contains the rental activities of 105,571 users who rented 5,850 unique items over a period of approximately 6 years. On average each user rented 2 items (see Figure 1:) and each item was rented 33 times (see Figure 1:). These graphs show that the dataset we have is made up primarily of customers who have rented less than 3 items and items that have been rented less than 33 times.

The dataset also contains customer reviews in the "review_text" column and a review summary in "review_summary". We would need to implement Natural Language Processing (NLP) techniques to leverage this data, which is outside the scope of our project, so we remove these two columns. We also have converted several features to more practical format and created new features from existing ones:

- Feature 'bust size'(34d) converted to 'bust_size_num'(34) and 'bust_size_letter'(d)

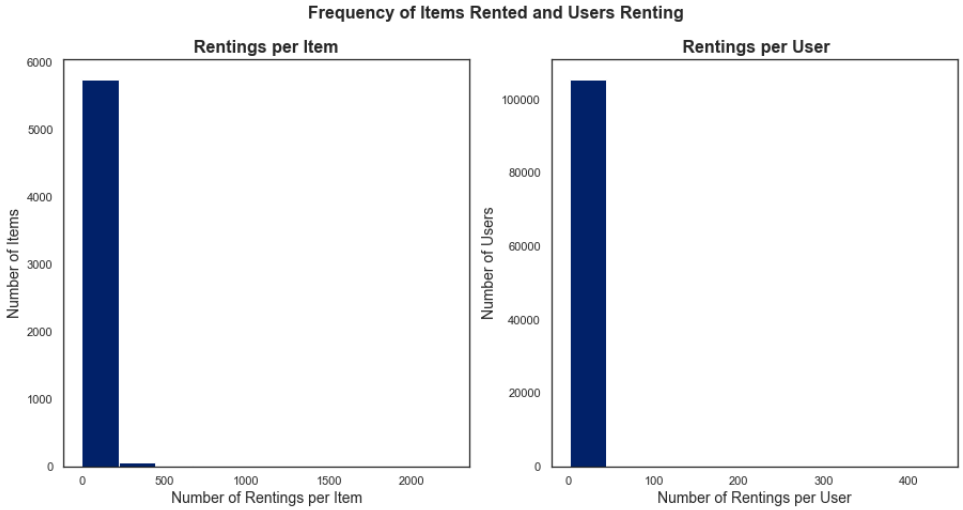


Figure 1: Box-Plots of physical characteristics of customers by ratings.

- Feature 'review_date'(April 20, 2016) converted to 'review_year'(2016)
- Feature 'weight'(137lbs) converted to 'weight_lbs'(137.0)
- Feature 'height'(5' 8") converted to 'height_in'(68.0)

We also checked for missing values, counts of which are shown below 1:

| | is_null_count |
|---|---|
| bust size | 18411 |
| weight | 29982 |
| rating | 82 |
| rented for | 10 |
| body type | 14637 |
| height | 677 |
| age | 960 |

Table 1: Table of features with Null values.

Before discarding or introducing any value, we checked for outliers and erroneous values. As a result 91 transactions were identified with customers over 100 years old and under 2 years old. These transactions were dropped. Since "rating" is our response variable, we also dropped any rows with missing values for "rating". For the rest of the features, we used a multivariate imputation method using IterativeImputer in sklearn linked here. IterativeImputer models each feature with missing values as a function of other features, and uses that estimate for imputation. It does so in an iterated round-robin fashion: at each step, a feature column is designated as output y and the other feature columns are treated as inputs X. A regressor is fit on (X, y) for known y. Then, the regressor is used to predict the missing values of y. This is done for each feature in an iterative fashion, and then is repeated for multiple imputation rounds. The results of the final imputation round are returned.

### Data analysis and Visualisation

One of the approaches for the recommendation in scope of this paper is classification method. We are going to use rating as the response variable to create classification model that predicts whether

a customer will like a product or not, given the customer's rental history. Ratings range from 2 to 10, and the distribution is skewed to the right towards 10, as shown in the Figure 2:. Since it's not practical to generalize customer ratings, we decided to convert the ratings to a binary variable, where 10 equals 1 and 0 otherwise. As a result, we have a data set with 65% of cases when the customer liked the product. (See Figure 2:)

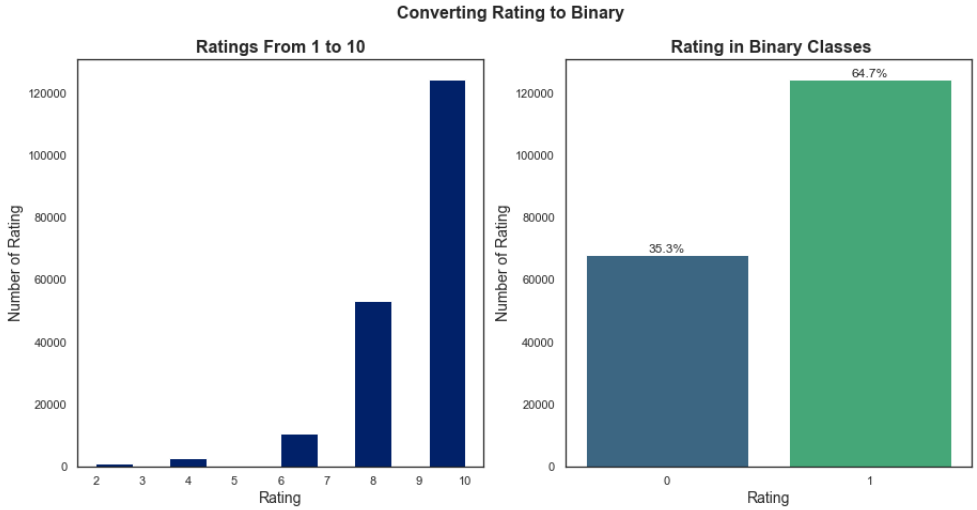


Figure 2: Box-Plots of physical characteristics of customers by ratings.

Analysis of box-plots of numerical variables such as size, height, weight and age with ratings shows mostly similar distributions with similar medians by ratings. With the exception of the size distribution for renters who liked the product, which has more renters with smaller sizes compared to the size distribution for renters who did not like the product (Figure 3).

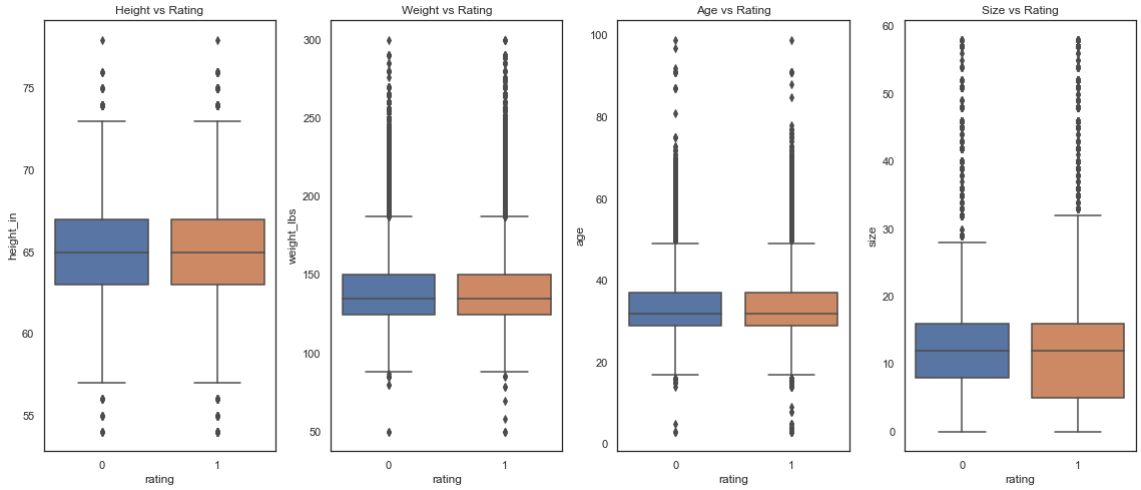


Figure 3: Box-Plots of physical characteristics of customers by ratings.

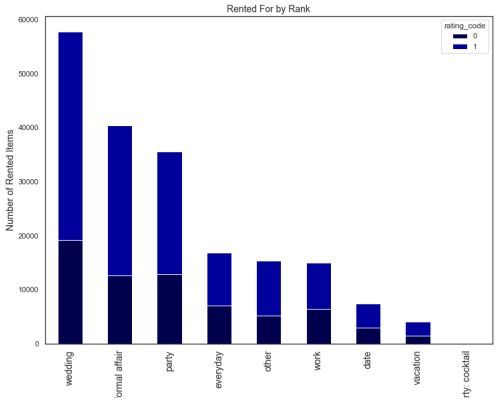


Figure 4

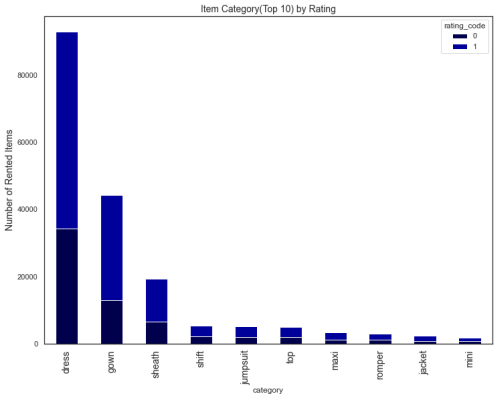


Figure 5

Similar fractions customer with rating 0 and 1 are also observed in categorical variables such as 'rented_for' and 'categories' within different categories.(See Figure 4: and 5:)

These similarities between different categories of categorical variables, as well as similar distributions of numerical variables, mean that they may not contribute significantly to the classification problem. We also looked at possible relationships between the categories of items and what those items were rented out for. In the figure 6: only the top 10 categories are included. This shows that only items in the "Gown" category were rented out mainly for "formal affair" and "wedding" events, while items in the other categories were rented out for various events.

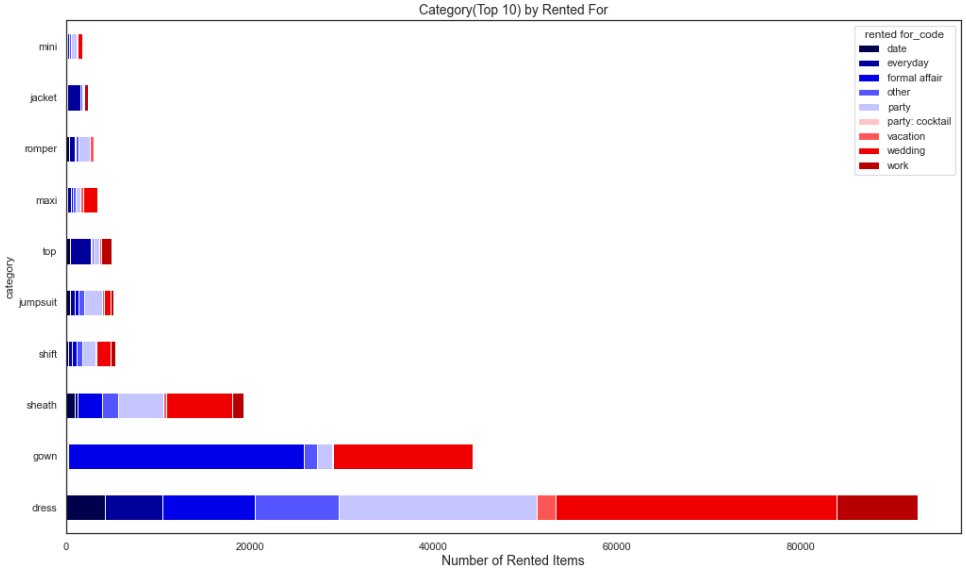


Figure 6: Item Categories by Rented For.

# Experimental Design

## *I Process*

After EDA (which has been covered in the previous section), we used a combination of supervised and unsupervised learning methods to generate a set of recommendations varying on their level of personalization, which we then compared with each other. Figure 10 shows the experimental design employed to process and analyze our data.

Once the features had been prepared for modeling, we separated our features with a 70/30 train-test split. The test data was set aside, and we used a 3-fold CV to hyperparameter tune various supervised models on the train data. The details of these models are given below. The train test contained 134,780 observations while the test set contained 57,764 observations. Once the parameters had been tuned on the train set, they were evaluated using the test set. This is our generalized "baseline" model.

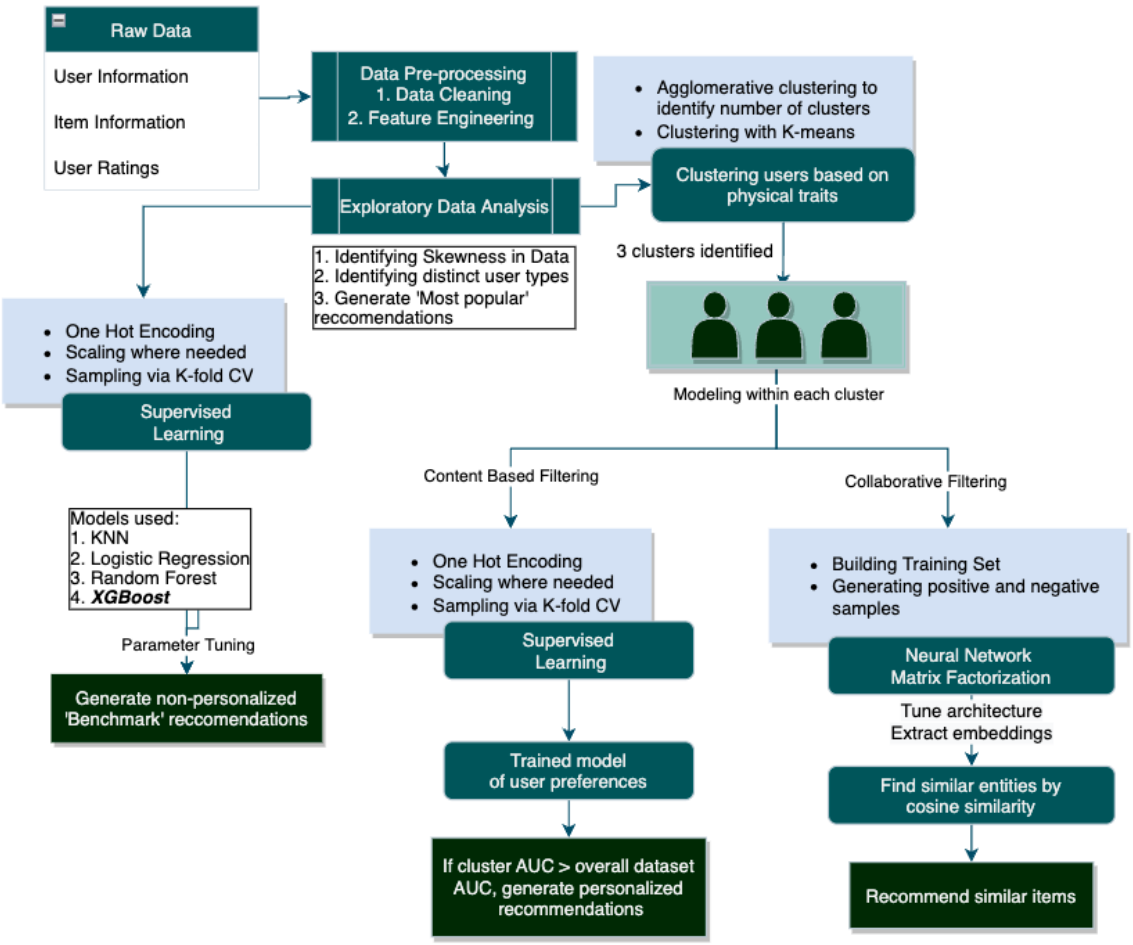


Figure 7: Experimental Design

Separately, we ran clustering techniques to split our entire dataset into clusters. The rest of the analysis was done within these clusters. First, we redid the Supervised Learning techniques within each cluster as a part of out content based filtering framework. Based on our evaluation metrics on the overall dataset, *XGBoost* was seen as the best modeling approach, so these clusters were modeled using only *XGBoost*.

We used the similar 70/30 split with 3-fold CV to tune the parameters here. Performance was each cluster was compared against the performance for the entire dataset to see whether a level of personalization (by segmenting based on user characteristics) can help improve recommendation performance over the general approach. In an applied setting, only if the AUC for a particular cluster was more than than of the general model, we would provide recommendations based on the cluster.
Next, as a part of our collaborative filtering framework, we built a item-based matrix factorization approach that computes cosine similarity between items and recommends items based on how similar they are.

## *II Methods*

### Clustering

Based on our EDA, we choose to cluster people based on their physical characteristics. The number of clusters were chosen with Agglomerative clustering based on visualizing them with a dendogram, and then the actual clustering was done using K-Means clustering. Here, multiple methods were chosen to cluster, like Agglomerative clustering itself, BIRCH (balanced iterative reducing and clustering using hierarchies), DBSCAN (Density-Based Spatial Clustering of Applications with Noise), but we saw that K-Means was most time and memory efficient while also providing separation of clusters.

### Supervised Learning Methods

We trained Logistic Regressions, KNN, Random Forest and XGBoost on the entire dataset to understand how to predict ratings for a user. This was a classification task with a binary 'Recomend' column as the target representing explicit feedback, and all user and item related columns as the features. While KNN and Logistic Regression were chosen as our baseline models, in the domain of online retail, research by Xia, et. al (2017) linked here has shown that Random Forest and XGBoost are two nonparametric learning methods that are resilient to noisy features, robust to outliers, and capable of handling missing features. Research by Dikker, J. (2017) also discussed that online retailers often experience a long tail in their revenue distribution, because of which current recommendation systems focus mainly on accuracy or related metrics and therefore provide recommendations involving mostly popular items. The research supported XGBoost for balanced item recommendation, claiming that initial weights of items are adjusted to emphasize certain items in the growing of the trees, and that as these weights travel along with the items for the duration of the algorithm, it results in a more balanced ensemble.

We evaluate our models based off selected evaluation metrics and proceed with the best performing model to fine tune using Bayesian optimization, which works by constructing a posterior distribution of functions that best describes the model to optimize. As the number of observations grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in parameter space are worth exploring.

### Neural Network Matrix Factorization

We use an item based matrix factorization method to compute similarity between items based on how they're rented by the same user. This forms our collaborative filtering framework. In order to create this representation of similar items, we use the concept of neural network entity embeddings.

The idea of entity embeddings is to map high-dimensional categorical variables to a low-dimensional learned representation that places similar entities closer together in the embedding space. By doing so, we not only get a reduced dimension representation of the items; we also get a representation that keeps similar items closer to each other. A high level visualization of this process can be seen

in Figure 11. As we can see, we first take our sparse User and Item information to create embedding layers, and pass it through the neural network to get the final embeddings. The specifics of the NN architecture is discussed below.

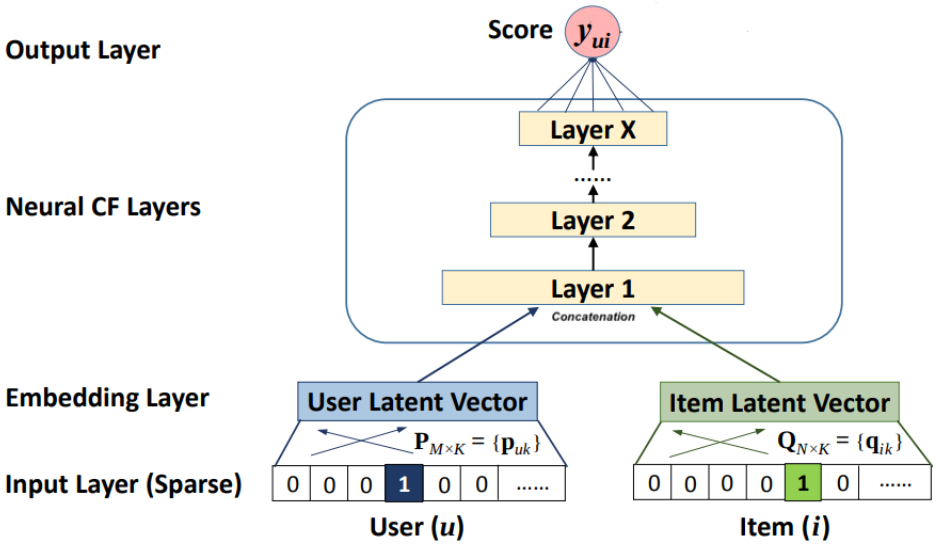


Figure 8: High level Neural Collaborative Filtering framework

To create entity embeddings, we need to build an embedding neural network and train it on a supervised machine learning task that will result in similar items having closer representations in embedding space. The layers of this network as as follows:

- Input: parallel inputs for the item and user
- Embedding: parallel embeddings for the item and user to a 50 number vector
- Dot: computes the dot product between the embeddings to merge them together as a single number
- Reshape: utility layer needed to correct the shape of the dot product

This raw number (after reshaping) is then the ouput of the model for the case of regression. In classification, the model loss function will be binary cross entropy in order to minimize the distance between the prediction and the output. An example of this architecture can be seen in Table 2.

The parameters of the neural network - the weights - are the embeddings, and so during training, these numbers are adjusted to minimize the loss on the prediction problem. We use an Adam optimizer and manually tune parameters like learning rate and batch size.

Once we have the embeddings for the items and the users, we can find the most similar item to a given user by computing the distance between the embedded vector for that item and all the other item embeddings. We'll use the cosine distance which measures the angle between two vectors as a measure of similarity (we also tried Euclidean distance). Results for the NN are provided in the later sections.

| Model Summary | | | |
|---|---|---|---|
| ***Layer (type)*** | ***Output Shape*** | ***Parameters*** | ***Connected to*** |
| item (InputLayer) | [(None, 1)] | 0 | - |
| user (InputLayer) | [(None, 1)] | 0 | - |
| item_embedding (Embedding) | (None, 1, 50) | 283050 | ['item[0][0]'] |
| user_embedding (Embedding) | (None, 1, 50) | 1298500 | ['user[0][0]'] |
| dot_product (Dot) | (None, 1, 1) | 0 | ['item_embedding[0][0]', 'user_embedding[0][0]'] |
| reshape (Reshape) | (None, 1) | 0 | ['dot_product[0][0]'] |

Table 2: Neural Network Architecture

### *Baseline and Evaluation Metrics*

For the general supervised model, our baseline metric is to recommend everything, which is the same as the proportion of 1s in the 'Recommend' column. Further, the baseline model to beat for each of the clusters is the general model (run on the entire dataset). We use Area under the Curve (AUC) to summarize the Receiver Operating Characteristic (ROC) curve measuring and Average Precision (AP), area under the Precision-Recall (PR) curve as the evaluation metrics in our classification models. Since our dataset is imbalanced in terms of positive to negative "Recommends", visualizing the PR Curve becomes especially important to us.

There has been extensive research done to propose baseline models in recommendation systems, of which one is the idea to reference what's been rated highly most often. This becomes the standard 'Most Popular Items, overall'. As an easy to implement and non personalized recommendation method, it is widely used as a baseline to provide a reference performance for a recommender system. (Yitong, et al. 2020). So, for the matrix factorization approach, we will be comparing our results to recommendations provided by the most popular items list.

## Results

We hypothesized that clustering our users into groups and building cluster-specific model could produce better recommendations for RTR users. If this is the case, we can use the chosen clustering method to segment our existing and new users and apply the best performing predictive models to make recommendations.

### *I Clustering*

First, to determine the number of clusters we should use for group our users, we evaluated the dendrogram as shown on in figure X below 9. The vertical axis of the dendrogram represents the dissimilarity between clusters whereas the horizontal represents the users. We determined that three to five clusters could be a reasonable number of clusters to allow enough users to be segmented into well-balanced groups. We examined the Principal Component Analysis (PCA) dimension reduced diagrams 10 for three, four, and five clusters to determine that three clusters best segmented our users and it also produced the best supervised modeling results discussed in the next section.

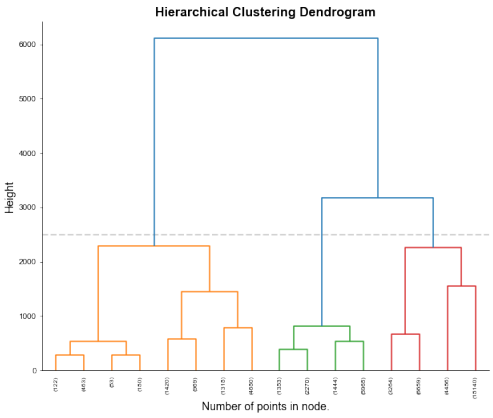


Figure 9


Figure 10

Since our clusters are formed using several features, the exact segmentation is based off latent features that we cannot examine. However we are able to examine data in each cluster to put a "label" on them. In our case, we found that the users primarily differed in size and weight_lbs 3:

| cluster | size | weight_lbs | label |
|---|---|---|---|
| 0 | 30.34 | 192.86 | large (L) |
| 1 | 16.16 | 149.90 | medium (M) |
| 2 | 6.86 | 123.20 | small (S) |

Table 3: Examine Mean Values within Clusters

## *II Supervised Learning Methods*

### General Model Performance

We evaluated our Logistic Regression, KNN, Random Forest, and XGBoost model's performance by comparing the AUC and AP scores. As seen from the table below 4, while all the binary classification models trained on the general data set out performs random chance, the AUC and AP scores of the XGBoost model trained on the general data set is highest, which is consistent with our research.
4:

| | Random Chance | Logistic Regression | KNN | Random Forest | XGBoost |
|---|---|---|---|---|---|
| AUC | 0.500 | 0.520 | 0.490 | 0.540 | 0.560 |
| AP | 0.647 | 0.643 | 0.646 | 0.645 | 0.647 |

Table 4: AUC and AP Scores Across Models

As such, we proceeded with hyper-parameter tuning of our general model as well as separately for cluster-specific models with XGBoost only.

### Cluster-Specific Model Performance

Our binary classification model results can be evaluated in detail using the ROC and PR curves below. We can clearly observe that all three cluster-specific models out performs the general model. While the metrics of small and large user groups are between 1.12 and 1.15 times better than our general model, we can see that our cluster-specific approach does particularly well predicting preferences for the medium cluster group.

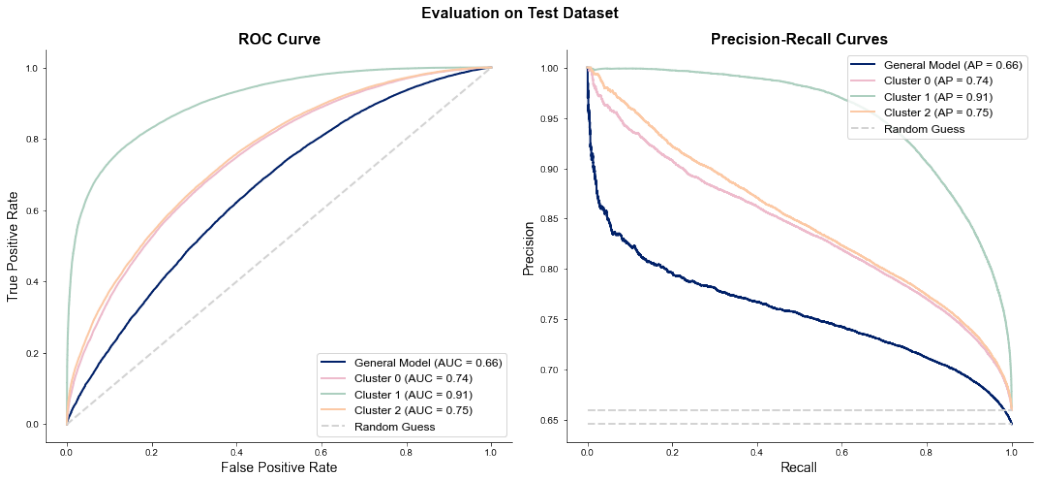


Figure 11: ROC Curves (left) and PR Curves (right) for XGBoost Model Evaluation

| | ***Item ID*** | ***Category*** | ***Top occassion rented for*** | ***Top body type rented for*** | ***Predicted Ratings*** |
|---|---|---|---|---|---|
| **Reference Item** | 123793 | Gown | Formal Affair | Hourglass | - |
| **Reccomendation 1** | 131533 | Gown | Formal Affair | Hourglass | 1 |
| **Reccomendation 2** | 130727 | Dress | Party | Hourglass | 1 |
| **Reccomendation 3** | 132738 | Gown | Formal Affair | Hourglass | 1 |
| **Reccomendation 4** | 136110 | Dress | Wedding | Hourglass | 1 |

Table 5

These results suggest that we should use cluster-specific models to make recommendations for all three clusters, but we can expect best results in the medium group.

### *III Neural Network Matrix Factorization*

For each of the clusters, we evaluated our neural network by finding similar items to a given item_id and comparing how close the recommendations provided by the neural network were. Ideally, we would have chosen to look at a mean recall which would evaluates whether the user's positive item (class rating = 1) is in the top 60 suggested list of recommendations (60 since that is the number of recommendations a user gets on signing onto the platform), but we are severely limited by the lack of users who have bought multiple items in our data. Although we are training for a supervised machine learning task, our end objective via matrix factorization is to learn the best entity embeddings, within our dataset. Hence we use the prediction problem as a means to an end rather than the final outcome.
The table below shows a example of an item and what the the model would predict for it:

## Applications

Following are snippets of the demo demonstrating the application of our recommendation engine:

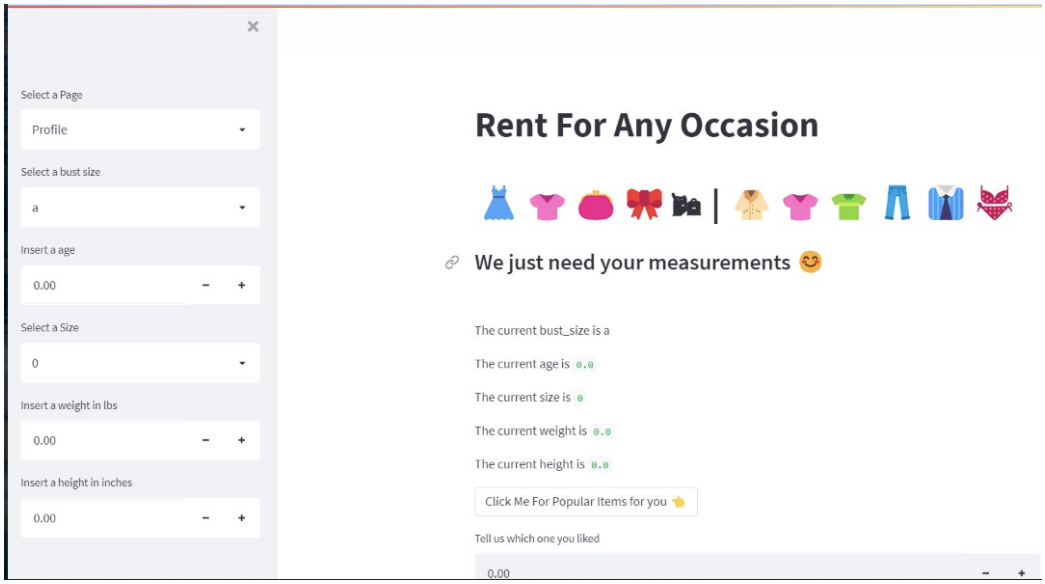


Figure 12: Landing page - requires measurement input and generates a list of popular items best suited to customer after assigning user to cluster

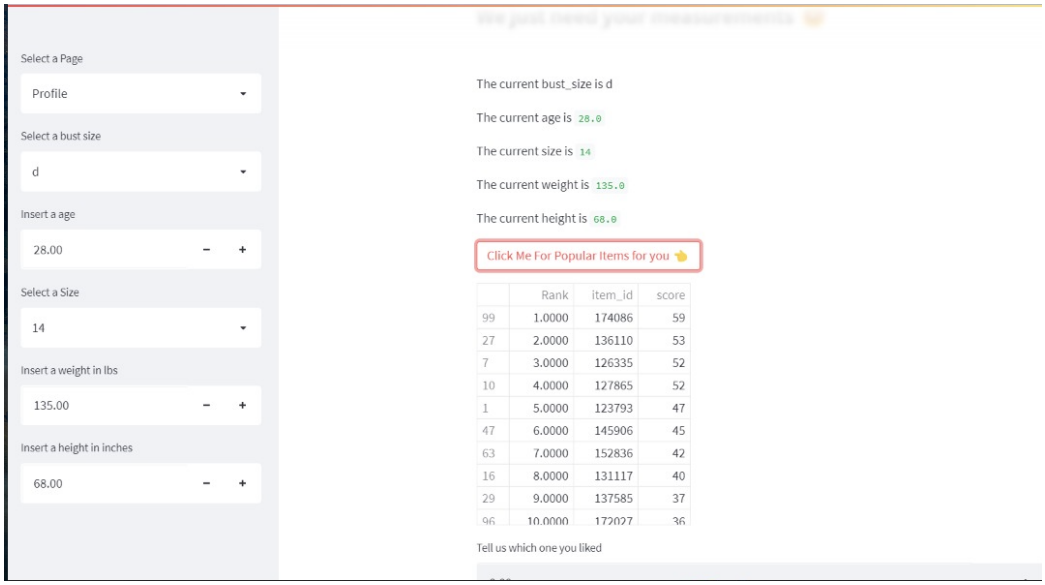


Figure 13: List of popular items generated

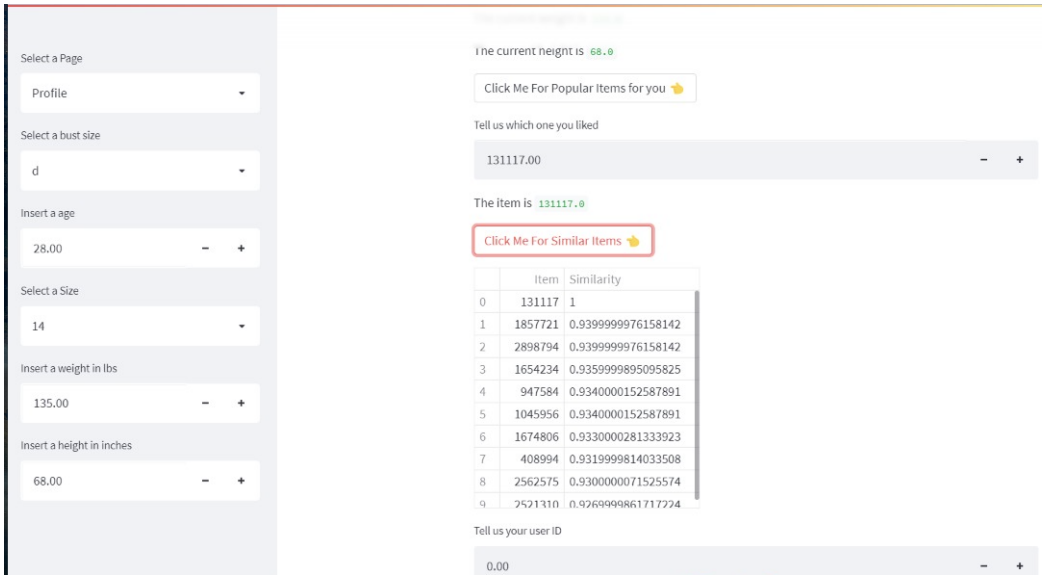


Figure 14: Recommendations are further refined by running cosine similarity

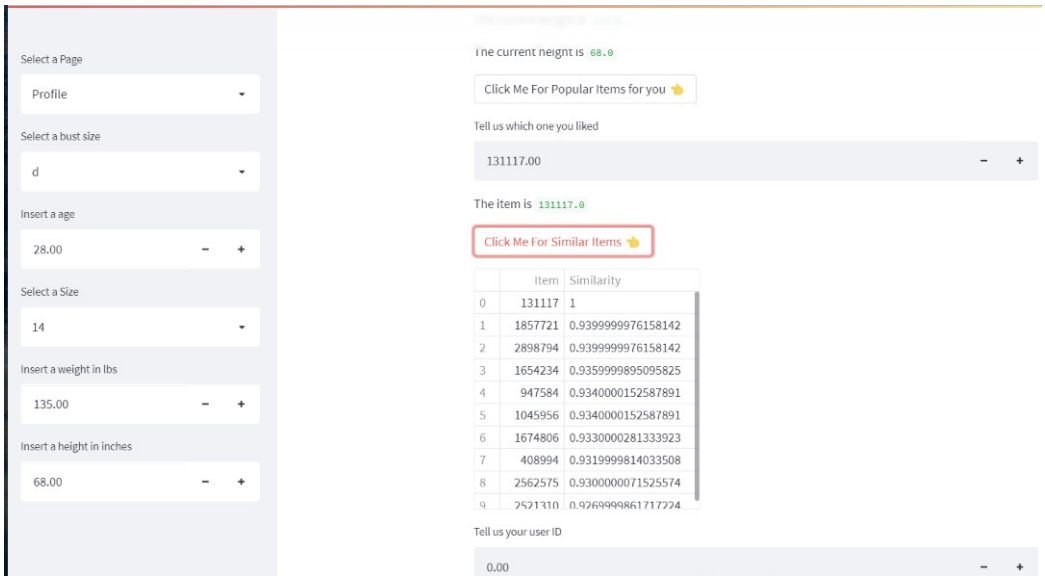


Figure 15: More personalized recommendations by running supervised learning models

## Conclusion and Limitations

We discovered that segmentation of the customer base into clusters based on physical characteristics (*size* in our study) can lead to remarkably better recommendation results. Further, employing techniques such as CosMF (Cosine Similarity Matrix Factorization) within these clusters can help in addressing challenging problems such as cold start and data sparsity. This study also highlighted how relatively computationally efficient and simple techniques such as clustering and cosine similarity can be used to solve complex problems such as recommendation systems. Moreover, this was an exceptional learning experience for the entire team and helped us explore new avenues in machine learning and deep learning.

Our recommendation engine has the following limitations:

1. The dataset we worked with had only 200k observations and $< 15$ features when the industry standard for datasets for building recommendations is millions of rows at the least. Our recommendation engine would potentially perform better with a dataset with more observations and features
2. We did not make use of the *reviews* column. Using NLP for that column can provide us with more insights on user profiles and preferences
3. We dropped nulls instead of imputing those observations
4. The recommendation engine does not combine content-based filtering and collaborative filtering approaches which could lead to even better recommendations

Given more time and resources, we could improve this project by addressing each of our limitations by gathering better data and performing more in depth analysis and tuning. In addition, there has been several studies on deep learning topics on neural network.

# References


[1] Ricci, Francesco, Lior Rokach, and Bracha Shapira. “Introduction to Recommender Systems Handbook.” Recommender Systems Handbook, 2010, 1–35. https://doi.org/10.1007/978-0-387-85820-3_1.

[2] Goldberg, David, David Nichols, Brian M. Oki, and Douglas Terry. 1992. “Using Collaborative Filtering to Weave an Information Tapestry.” Communications of the ACM 35 (12): 61–70.

[3] Ekstrand, Michael D. 2011. “Collaborative Filtering Recommender Systems.” Foundations and Trends® in Human–Computer Interaction 4 (2): 81–173. https://doi.org/10.1561/1100000009.

[4] Burke, Robin. "Knowledge-based recommender systems."

[5] Balabanović, Marko, and Yoav Shoham. 1997. “Fab: Content-Based, Collaborative Recommendation.” Communications of the ACM 40 (3): 66–72. https://doi.org/10.1145/245108.245124.

[6] Schafer, J. Ben, Joseph Konstan, and John Riedl. "Recommender systems in e-commerce." In Proceedings of the 1st ACM conference on Electronic commerce, pp. 158-166. 1999.

[7] Mehta, Rachana, and Keyur Rana. 2017. “A Review on Matrix Factorization Techniques in Recommender Systems.” IEEE Xplore. 2017. https://doi.org/10.1109/CSCITA.2017.8066567.

[8] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 42, no. 8 (2009): 30-37.

[9] Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[10] Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[11] Koren, Yehuda. "Collaborative filtering with temporal dynamics." In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 447-456. 2009.

[12] Koren, Yehuda. "Factorization meets the neighborhood: a multifaceted collaborative filtering model." In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 426-434. 2008.

[13] Wen, Hailong, Guiguang Ding, Cong Liu, and Jianming Wang. "Matrix factorization meets cosine similarity: addressing sparsity problem in collaborative filtering recommender system." In Asia-pacific web conference, pp. 306-317. Springer, Cham, 2014.

[14] Liu, Nathan N., Bin Cao, Min Zhao, and Qiang Yang. "Adapting neighborhood and matrix factorization models for context aware recommendation." In Proceedings of the Workshop on Context-Aware Movie Recommendation, pp. 7-13. 2010.

[15] Shi, Jiangfeng, Mingsheng Long, Qiang Liu, Guiguang Ding, and Jianmin Wang. "Twin bridge transfer learning for sparse collaborative filtering." In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 496-507. Springer, Berlin, Heidelberg, 2013.

[16] Shi, Jiangfeng, Mingsheng Long, Qiang Liu, Guiguang Ding, and Jianmin Wang. "Twin bridge transfer learning for sparse collaborative filtering." In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 496-507. Springer, Berlin, Heidelberg, 2013.

[17] Yitong Ji, Aixin Sun, Jie Zhang and Chenliang Li. "A Re-visit of the Popularity Baseline in Recommender Systems." In Proceedings of Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, China, July 25–30, 2020 (SIGIR '20)

[18] Yandi Xia, Giuseppe Di Fabbrizio, Shikhar Vaibhav, and Ankur Datta. "A Content-based Recommender System for E-commerce Oers and Coupons." In Proceedings of SIGIR eCom 2017, Tokyo, Japan, August 2017, 7 pages.

[19] Dikker, J. "Boosted tree learning for balanced item recommendation in online retail." 18 Dec 2017.

[20] Rishabh Misra, Mengting Wan, Julian McAuley. "Decomposing fit semantics for product size recommendation in metric spaces". 2018.

# Roles

- Anna Dai: Lead Data Merging process with data cleaning and feature engineering. Dived deeply into related references for recommendation systems to understand the use case and explored experimental approaches. Assessed cluster dissimilarity and hyperparameter tuned the XGBoost Models. Contributed to the report and presentation with "Results" section. Brought together plots when appropriate for comparison. Contributed in designing presentation.
- Sarwari Das: In charge of exploratory data analysis and comparing/tuning general supervised learning model. Dealt with missing value imputation, one hot encoding, and creating initial clusters. In charge of collecting and compiling performance metrics. Tuned neural network models within clusters. Contributed to the report and presentation with methodology section. Contributed in designing presentation.
- Tigran Harutyunyan: Led exploratory data analysis of the dataset, worked on feature extraction and engineering. Led process for neural network design, matrix factorization and assessing cosine similarity between items. Created minimum viable product for presentation using StreamLit. Contributed to the report with "Data" section. Contributed in designing presentation.
- Surabhi Trivedi: Conducted exploratory data analysis of the dataset, and performed dimension reduction using PCA to identify clusters within the data. Suggested the final approach for clustering and classification. Conducted extensive research to understand the history and various approaches to recommendation engines. Contributed to the report with "Introduction", "Background", "Limitations", "Conclusion". In charge of organizing reference material. Contributed in designing presentation.