Hybrid Recommender System

By Neha Saraf and Kiran Saini

1. Introduction

Recommendation systems are used widely nowadays in the e-commerce and content-streaming industries to boost up their sales. It provides a way to retain the customers by recommending them products such as movies, apparel, etc.

Collaborative filtering is a technique used to build recommendation system. Collaborative filtering uses the technique of collaborating the ratings of different users and then using that knowledge to filter out the products which a particular user might like.

Content based recommendation is a different technique used to recommend items to the user. Content-based Recommender system uses the technique of using the past preferences of user to recommend similar items to the user. Pure content-based recommender system uses solely the profile built by the user and its previous preferences of items to recommend items in the future.

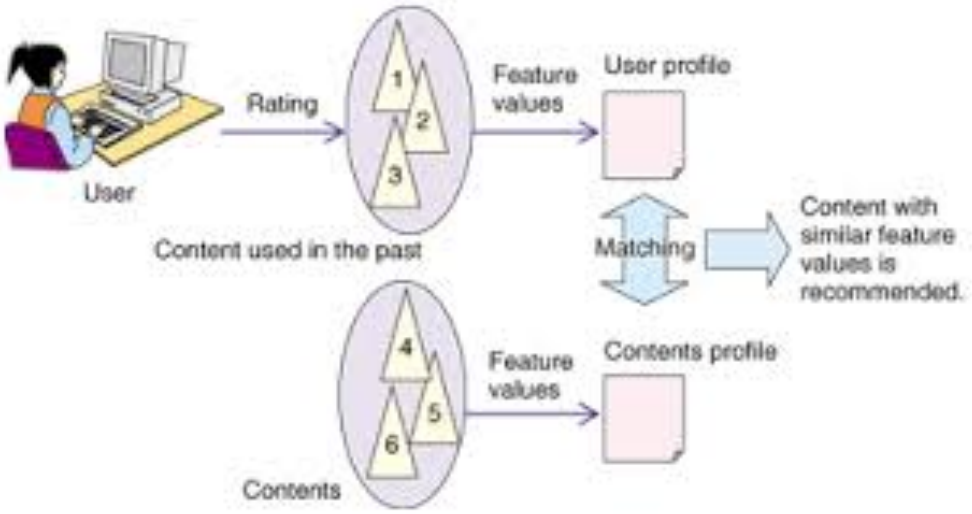
Hybrid Recommender systems are those systems which are built by combining both content-based and collaborative based methods. It takes the advantage from both the representation of content of a user and similarity among the other users. Hybrid Recommender Systems are generally built to overcome the disadvantages of both the systems.

The types of Hybrid Recommendation systems are:

1. **Weighted:** The ratings from multiple recommendation techniques are combined into single rating.
2. **Switching:** Ratings from the recommendation technique that is best for the current context are used. When we do not have a lot of information on user behavior, content-based system is used and then as user behavior information becomes available, we switch to collaborative filtering.
3. **Mixed:** Top-k recommendations from multiple systems are used at the same time. For e.g. We might get 5 recommendations from a content-based system and 5 recommendations from a collaborative filtering system.
4. **Cascade:** The ratings output from one model is used as input for the next model.

Content-based models

Content-based recommendation techniques use a predictive model based on user's past preferences to predict ratings of unrated items. The items have features such as description, brand, summary, genre, category. The user model is trained to recognize similar items based on these features. For e.g. The user might favor a certain brand. The user model can use classification or regression to predict ratings of items on the basis of features and user ratings available. Ratings from the past and their feature values are used to build the user profile. The content profile is the feature values for that item. Thus the content profile matching the user profile is recommended for that user. Some classification and regression techniques used for content-based recommendation are Logistic Regression, Naive Bayes Classifier, Random Forest Regression.



Content-based Recommendation

Pros:

- The model can predict long-term preferences of user
- The model solves the cold start problem for new items. Since we have item features available for new items, we can check which user profile it matches and recommend that item for matched users.

Cons:

- The model can only give meaningful recommendations when we have user ratings for at least a few items. This presents a cold start problem for a new user.
- The model only outputs items similar to the ones already purchased. This can result in a lack of serendipity and diversity.

LSH Spark + Collaborative Filtering

Let us first introduce Spark:
Apache Spark is the general-purpose clustering computing system. We have used the PySpark package which is the Python API for spark. Spark implementation will enable us to scale up the dataset and other computational work. Pyspark package contains pyspark.ml package which in turn help us to implement the Bucketed Random Projection and Minhash Aprroaches for LSH Algorithm.

Pros of Pyspark:

- Simple to write parallelized code for simple problems.
- Algorithm done with proper documentation.

Cons of Pyspark:

- Writing map and lambda functions to perform small operations like reading csv is little tedious and difficult.
- Due to its lazy programming nature, Pyspark is slow performance wise.

LSH has only been used to deal with high dimensional data so that finding nearest neighbors is computationally easier as compared to previous approach of finding neighbors by using its pearson similarity matrix. The ratings are still predicted using the same neighborhood-based collaborative filtering approach. There are following pros and cons of Collaborative filtering approach which we found out from our previous implementation:

Pros of Neighborhood-based:

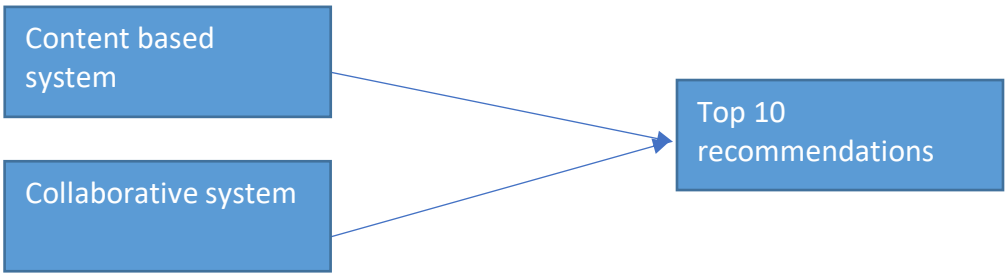
- Easier to interpret the concept of how the ratings are predicted and hence play around with its parameters.
- Moreover, in collaborative filtering models, we do not have to depend on the context of items or users. CF algorithms only depend on liking of other items and users.

Cons of Neighborhood-based:

- Cold Start Problem is difficult to resolve as we will have no or very little information of the new users as compared to other users.

Our Hybrid Approach:

We have built a mixed model of collaborative filtering and content-based recommendation to balance out the cons of each model. The content-based model gives top 5 recommendations and the collaborative filtering model gives top 5 recommendations and these are combined to obtain top 10 recommendations for each user.



2. Dataset

The dataset: We have used the dataset for Amazon Clothing, Shoes and Jewelry. Two datasets have been used: the items metadata and the 5-core ratings dataset. The 5-core metadata contains the ratings given by different users to different items whereas the metadata contains more detail information about the items.
Note: Please refer to the link above to know more about the dataset.

Data Sampling: The 5-core rating dataset contains 39387 unique users and 23033 unique items with ratings for 1393385 user-item combinations. We have explicit ratings with scale of 1-5. As before we implemented the collaborative filtering model with three different sizes of train and test set, this time also we have tried following the same procedure.

3. Visualizations

Before we dive into our objective and explaining our model, we will dive into the datasets with few visualizations to have a deeper understanding of the data.

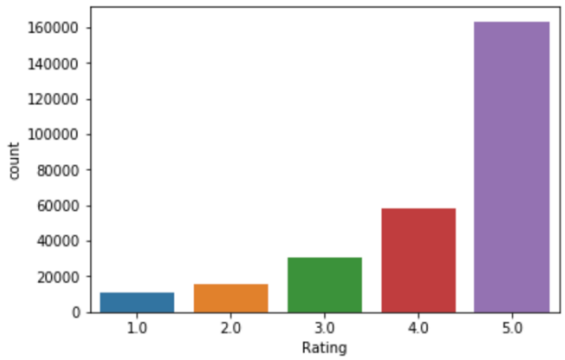


Figure 1: Count of each rating

Figure 1 shows that the count of rating 5 is much more than the count of other ratings which means that users have given mostly 5 to the available items in the dataset.

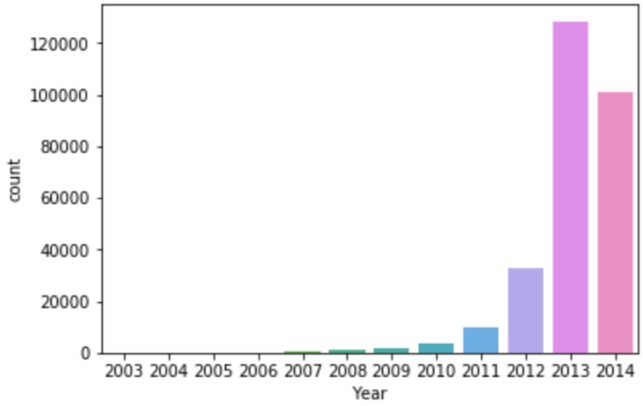


Figure 2: Count of ratings in each year

Figure 2 shows that the number of reviews increased over the years. The data was collected till July 2014 only. This explains the decrease for 2014.

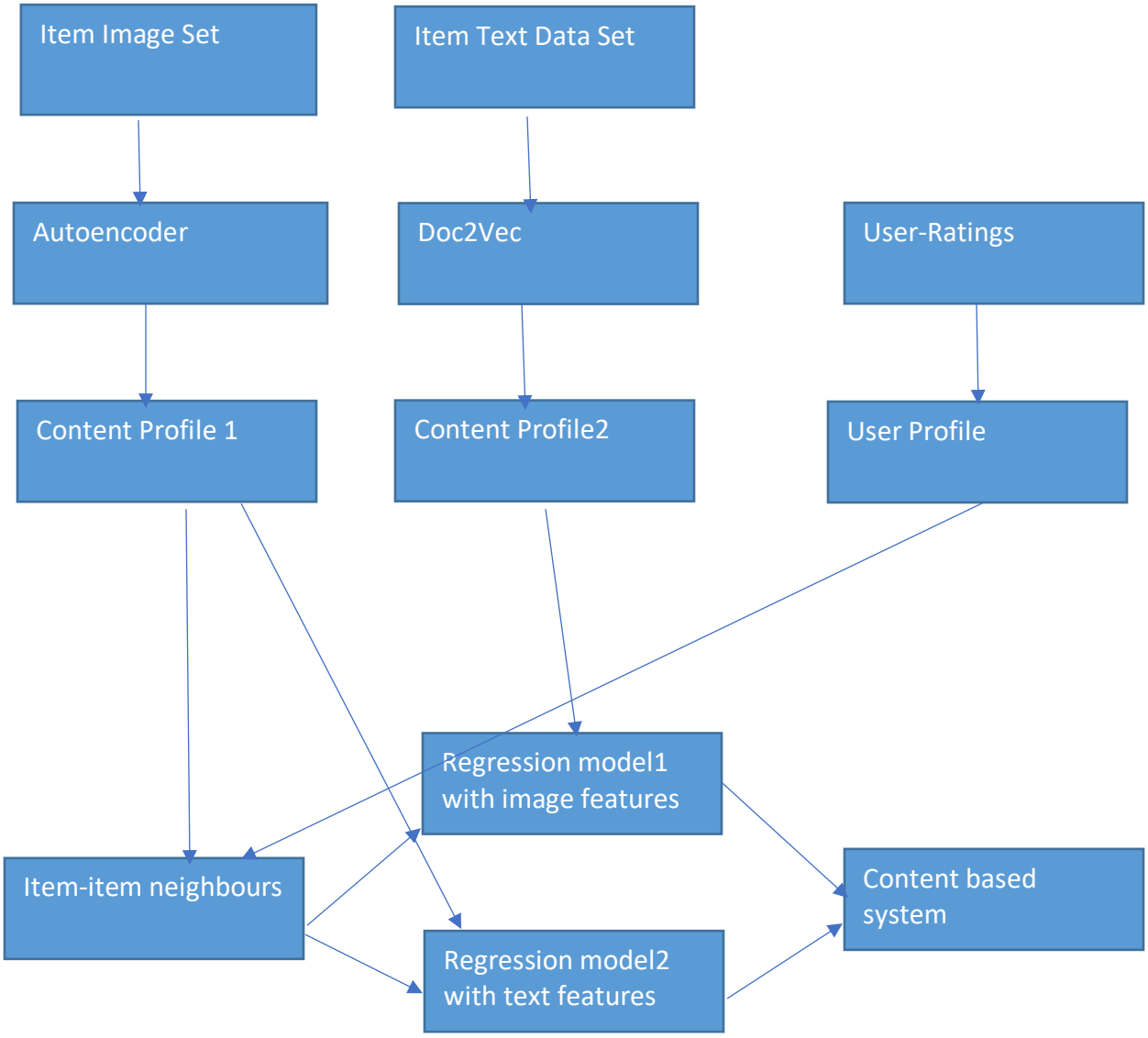
There are many more things which can be observed for the 5-core dataset of Amazon Clothes. Please refer to the notebook: Recommendation_system_EDA.ipynb to get more insights of the dataset.

4. Objective

The objective is to predict the user item ratings for all the users in our dataset. We have implemented the combination of two different approaches: the first one is the content-based model. The second approach is where we have use Locality Sensitive Hashing (LSH) to generate top 10 neighbors of a user in combination with the neighborhood based collaborative filtering model. LSH implementation has been done using Spark so that it can be scaled up when required.

5. Content-based Model

The architecture of our content-based model is as follows:



We created two content profiles for the items. One content profile is based on image of the item where as the other content profile is based on the text information available for the item.

Content Profile based on Image:

The dataset contains an image URL for each item. Each image URL was opened and the image was stored as 28*28*3 image. Then an autoencoder was used to reduce the dimension of image from 28*28*3 to 4*4*1 while trying retain as much information as possible. Autoencoders are used to get an encoding for data. This is typically used in dimension reduction. Autoencoders is an unsupervised method that tries to copy the input to the output. It tries to reduce the input to its latent space representation and then uses the latent space representation to recreate the input as the output.

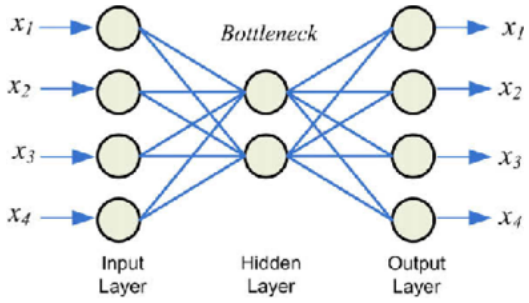


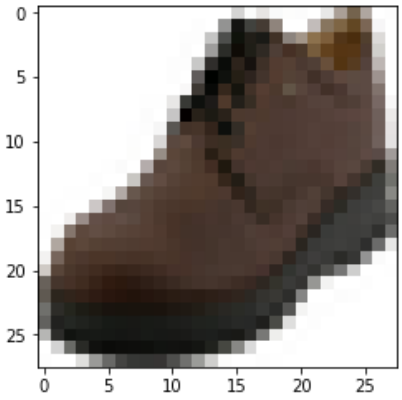
Figure 3: Structure of Autoencoders

Layer (type)	Output Shape	Param #	
input_3 (InputLayer)	(None, 28, 28, 3)	0	
conv2d_19 (Conv2D)	(None, 28, 28, 16)	448	
max_pooling2d_7 (MaxPooling2	(None, 14, 14, 16)	0	
conv2d_20 (Conv2D)	(None, 14, 14, 8)	1160	
max_pooling2d_8 (MaxPooling2	(None, 7, 7, 8)	0	
conv2d_21 (Conv2D)	(None, 7, 7, 8)	584	
max_pooling2d_9 (MaxPooling2	(None, 4, 4, 8)	0	
conv2d_22 (Conv2D)	(None, 4, 4, 1)	73	
encoder (MaxPooling2D)	(None, 4, 4, 1)	0	
conv2d_23 (Conv2D)	(None, 4, 4, 1)	10	
up_sampling2d_9 (UpSampling2	(None, 4, 4, 1)	0	
conv2d_24 (Conv2D)	(None, 4, 4, 8)	80	
up_sampling2d_10 (UpSampling	(None, 8, 8, 8)	0	
conv2d_25 (Conv2D)	(None, 8, 8, 8)	584	
up_sampling2d_11 (UpSampling	(None, 16, 16, 8)	0	
conv2d_26 (Conv2D)	(None, 14, 14, 16)	1168	
up_sampling2d_12 (UpSampling	(None, 28, 28, 16)	0	
conv2d_27 (Conv2D)	(None, 28, 28, 3)	435	

Total params: 4,542 Trainable params: 4,542 Non-trainable params: 0

Architecture of Auto encoder

The latent space representation obtained for each 28*28*3 image is a vector of length 4*4*1. The latent space representation was obtained by reducing the RMSE error such that the output equals the input. To test the effectiveness of the latent space representation of item images, a nearest neighbors' model was trained on all latent space representations in the test set. We found the nearest neighbors for the test item. The result obtained was as follows:



Test Image

Nearest neighbors

The nearest neighbors obtained are very close to the input image. Thus, we can say that items with similar latent space representations built using our autoencoder have similar images. The auto encoder model was used to find the latent space representation for all items and the representation was stored as the content profile.

Content Profile based on Text:

We have textual information for each item such as brand, description and categories. These columns in the dataset were combined to create one column which represents all textual information available for that item as a paragraph. A Doc2Vec model was trained on all such item paragraphs to represent each paragraph as a vector of size 15. If two item vectors are similar that means their paragraphs are similar. This means they might have a common brand or similar description or similar category.

The doc2vec model is used to find the latent space representation for each item such that two items with similar vectors are similar. To test the doc2vec model a test item was chosen and items with similar item vectors were found. The results were as follows:

Test Input: CASIO LA11WB-1 Ladies Watch Casio Splash-resistant, classically casual Woman's digital watch with stopwatch and timer Clothing Shoes & Jewelry Women Watches Wrist Watches

Output:

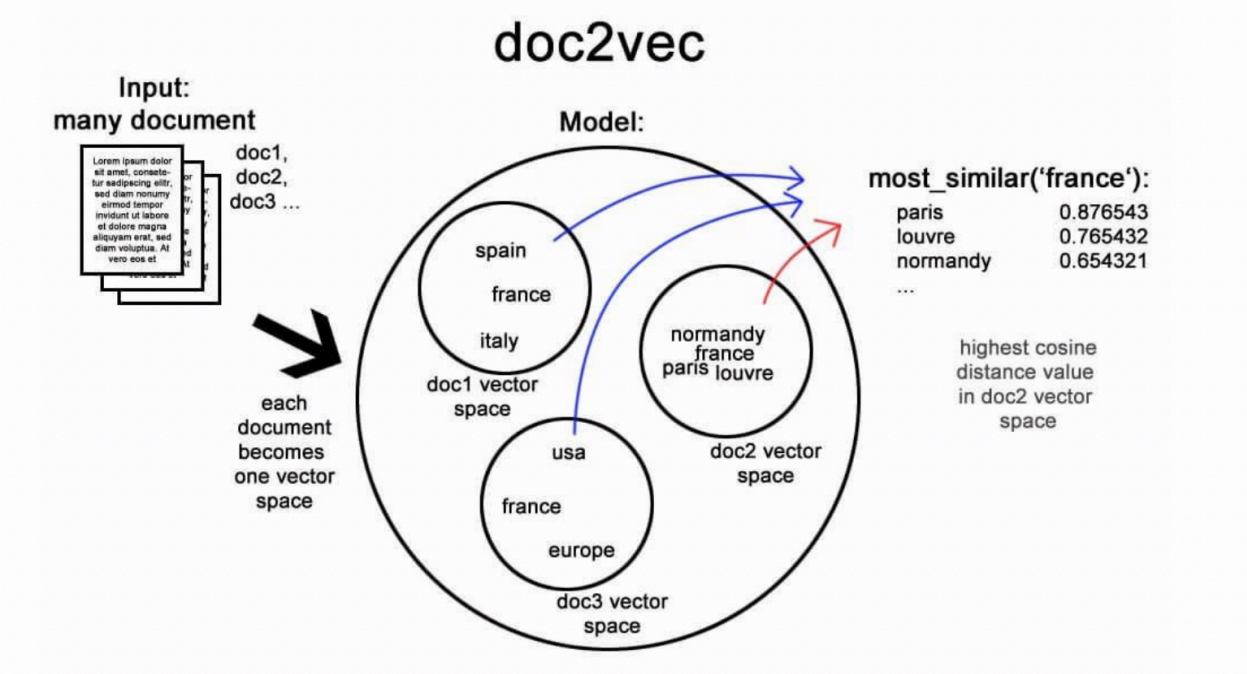


Figure 4: Doc2Vec Model

```
"Tommy Hilfiger Men's 1790859 Black Leather Analog Quartz Watch with Grey
Dial Clothing Shoes & Jewelry Fashion Watches Clothing Shoes & Jewelry T
Tommy Hilfiger Clothing Shoes & Jewelry Men Watches Wrist Watches "
```

Out[90]:

```
"Casio Men's DW5600BB-1 Black Resin Quartz Watch with Digital Dial Clothin
g Shoes & Jewelry Men Watches Wrist Watches "
```

Out[91]:

```
"Casio Men's PRGS510-1 Black Resin Quartz Watch with Grey Dial Clothing Sh
oes & Jewelry Men Watches Wrist Watches "
```

We can see that the similar items have a similar brand or category. Thus we can say that items with similar latent space representations built using our doc2vec model have similar textual information. The Doc2vec model was used to find the latent space representation for all items and the representation was stored as the content profile.

Item-item neighbors:

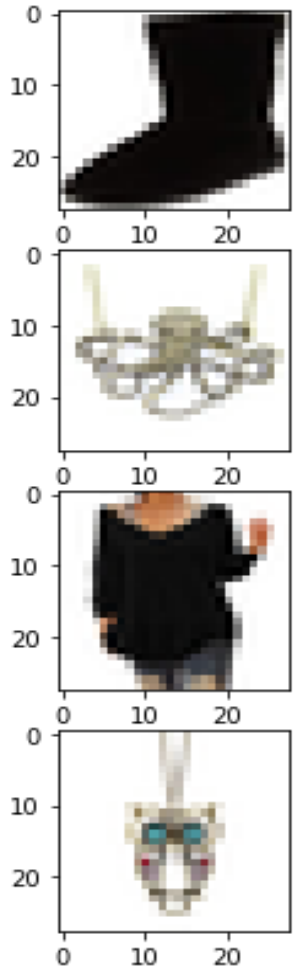
Once we had the content profile for all items, we built the user-profile. On average each user has approximately 5 rated items. This is not enough training examples to build a regression model for each user. Thus, to decrease the sparsity of the matrix we used user ratings and nearest neighbors model trained on all item image vectors to find the 20 nearest neighbors of each item rated for each user. Then we predicted the rating for the nearest neighbors based on cosine similarity to the items rated by that user. Thus, we obtained about 50 training examples for each user. This technique was used to reduce sparsity of user-item matrix.

Random Forest Regression based on item text features

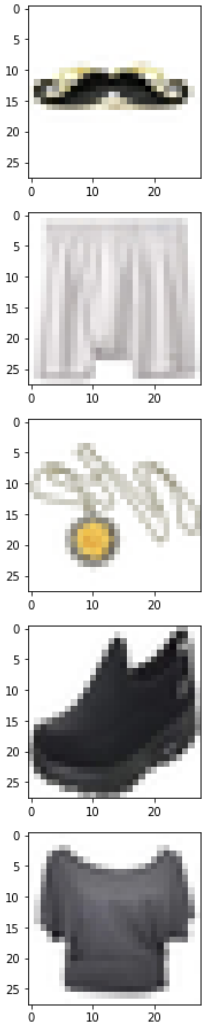
A regression model was trained for each user with the item text features as the features and the rating as the label. Using item-item neighbors' method we got about 40 training examples per user. Using these rated examples, the model predicted the rating for all unrated items since we have the feature values for all items from the content profile created.

Cross-validation and Hyper-parameter tuning:

GridSearchCV was used to perform 3-fold cross validation and hyper parameter tuning to get the optimum parameter values for our Random Forest Regression model.
The results obtained were as follows:



Items purchased by test user



Items recommended for test user

As we can see the items recommended are similar to the ones purchased by the user.

Random Forest Regression based on item image features

A regression model was trained for each user with the item image features as the features and the rating as the label. Using item-item neighbors' method we got about 40 training examples per user. Using these rated examples, the model predicted the rating for all unrated items since we have the feature values for all items from the content profile created based on image information for each item.

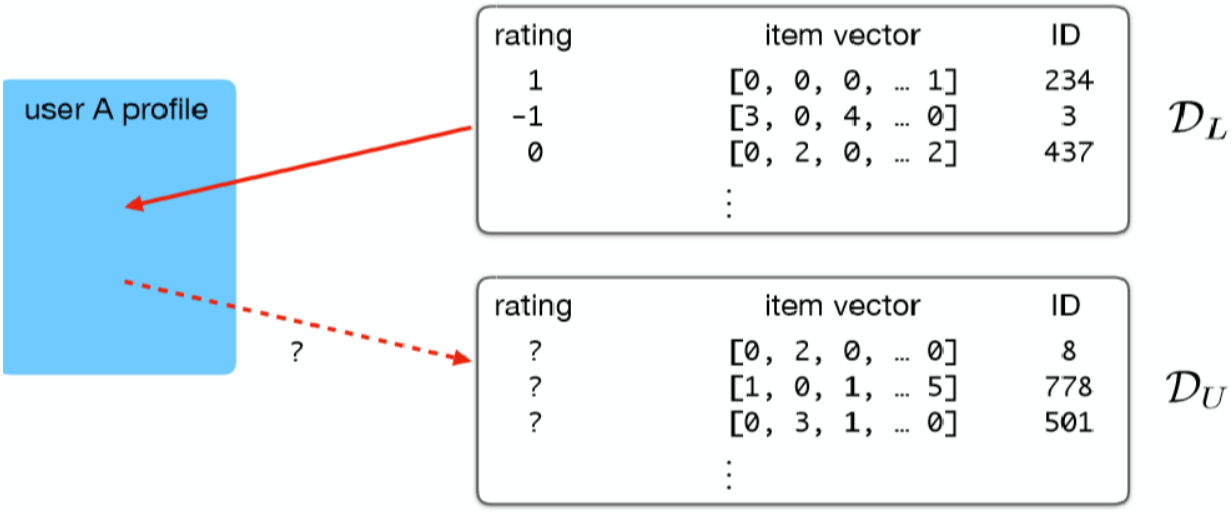
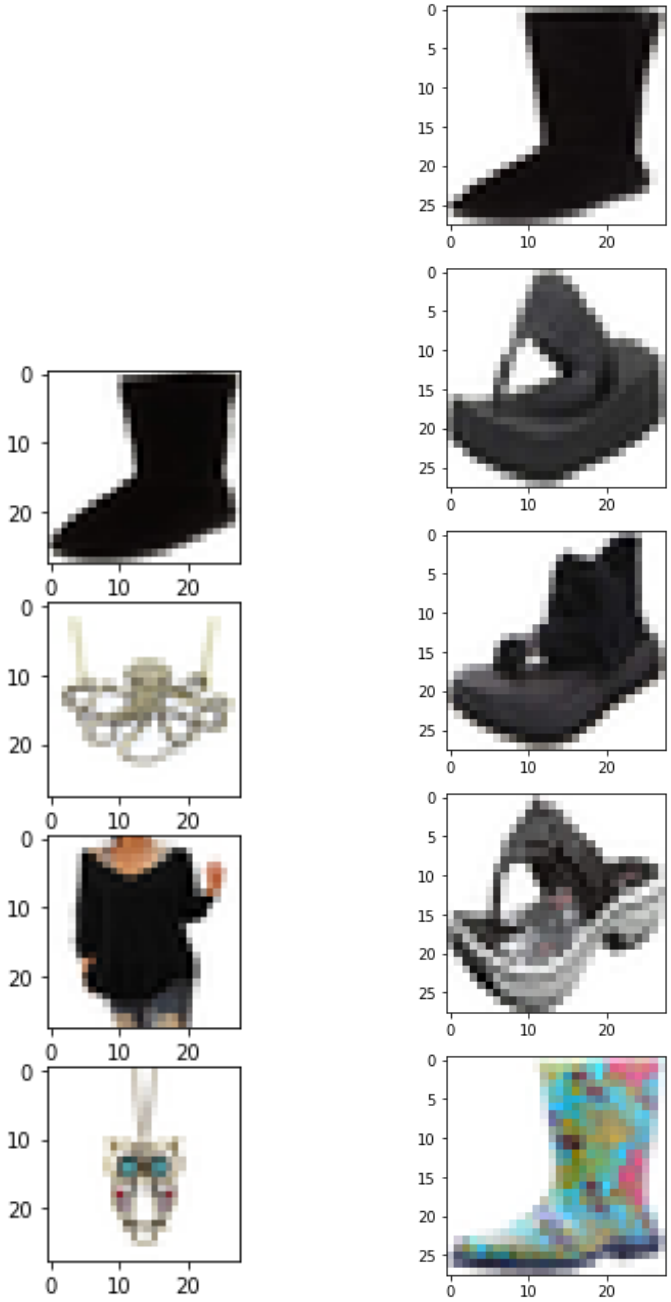


Figure 5: Profile learning

Cross-validation and Hyper-parameter tuning:

GridSearchCV was used to perform 3-fold cross validation and hyper parameter tuning to get the optimum parameter values for our Random Forest Regression model.
The results obtained were as follows:

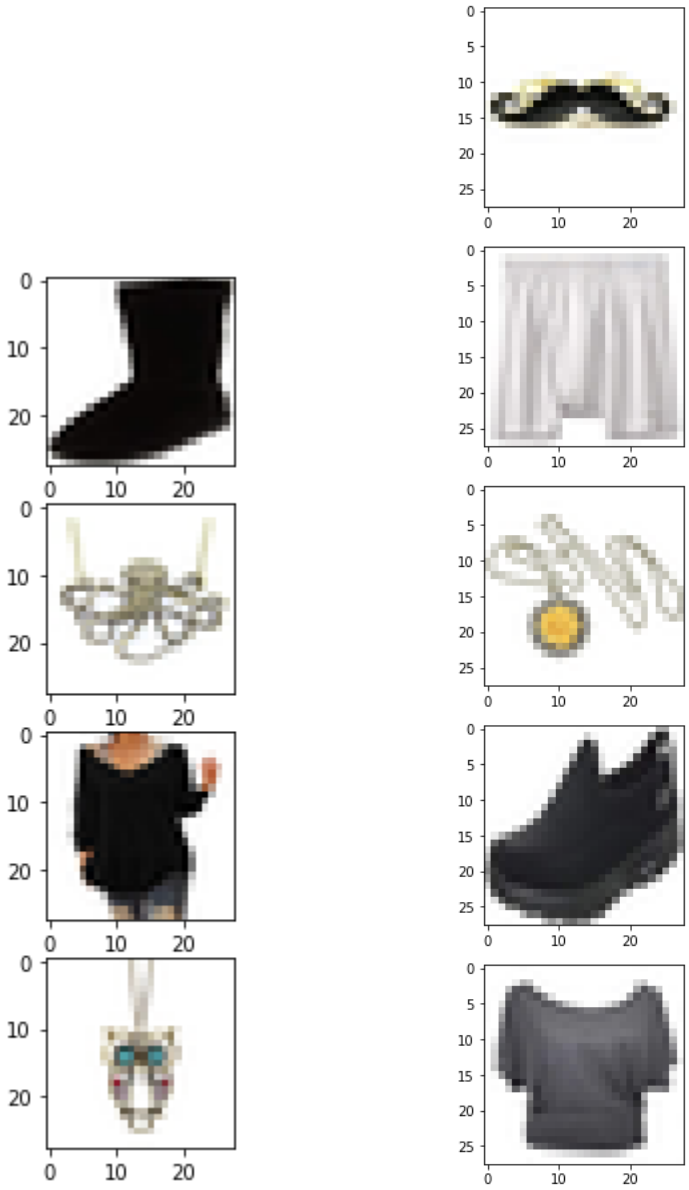


Items bought by test user *Items recommended for test user*

We can see that recommended items have a similar shape or color to the items bought by the user and hence are similar to the items bought by the user.

Content-based model:

The content-based model takes the average of the ratings predicted using Regression model with item image features and Regression model using item text features.
The results obtained were as follows:



Items bought by test user *Items recommended for test user*

We can observe that the recommended items are similar to items by the user.

6. Locality Sensitive Hashing + Collaborative Filtering (Spark Implementation)

LSH algorithm is use to reduce the dimensionality of high-dimensional data. The technique used by this algorithm is that it hashes the input so that the similar items maps into the same buckets with high probability (Wikipedia definition). It is a randomized-algorithm used for solving the nearest neighbors problem in high-dimensional data. Hence, we use this fact to combine it with our famous neighborhood-based model of collaborative filtering to predict the ratings of user-item combination.

To implement LSH we first got the feature vector for every row of the 5-core dataframe. The feature vector for every row will give us the mapped user index, mapped item index and their corresponding ratings. To implement this, we have used Indexer and Assembler feature transformers of spark framework. These feature vectors are then used to create the LSH model for hashing. The LSH algorithms which spark provides for implementation are: Bucketed Random Projection for Euclidean Distance and Minhash for Jaccard distance. Here for our implementation purpose, we have used the first approach of Bucketed Random Projection for Euclidean distance.
As per the documentation of this algorithm, Bucketed Random Projection is LSH family for Eulidean distance. The Euclidean distance is defined as follows:

$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

Its LSH family projects feature vectors x onto a random unit vector v and portions the projected results into hash buckets:

$$h(x) = \left\lfloor \frac{x.v}{r} \right\rfloor$$

where r is a user-defined bucket length. The bucket length can be used to control the average size of hash buckets (and thus the number of buckets). A larger bucket length (i.e., fewer buckets) increases the probability of features being hashed to the same bucket (increasing the numbers of true and false positives).

Bucketed Random Projection accepts arbitrary vectors as input features, and supports both sparse and dense vectors.

Then after we implement this model, we can use the approxNearestNeighbors method of this model to get the top k neighbors for a particular user or item with a distance column. These neighbors are now used with the neighborhood-based collaborative filtering model to predict the ratings of the user-item combination. The distance column will now act as the similarity score for the weighted average (in place of the Pearson similarity distance).
Note: The implementation of neighborhood-based model is exactly same as the one implemented in the previous homework.

Advantages:
The major drawback which we overcome here by combining LSH with neighborhood-based model is the issue of scalability. Since LSH has been implemented in Spark framework and it's a randomized algorithm which has been designed in a way so that the scalability issue can be solved.

Major Concerns:
When we implemented this algorithm, the major concern we faced is combining the results of LSH with neighborhood-based model. Since Spark follows the lazy programming, it gives MemoryOut exception a lot of times, which makes the implementation of it little difficult and time taking process.

7. Evaluation

a) Accuracy

Accuracy is a good method to check how our model is performing on the test data. It tests our predicted values against the original observed ratings and the according to that give a score how good our model is. Here, we have used two accuracy metrics:

Root Mean Square Error (RMSE):

Let r_{uj} be the predicted rating for user 'u' and item 'j'

Let o_{uj} be the observed rating for user 'u' and item 'j'

$$RMSE = \sqrt{\frac{\sum_{j=1}^{n}(r_{uj} - o_{uj})^2}{n}}$$

where, n = number of ratings in test data.

RMSE can be used to measure the spread of the observed values of ratings about the predicted value of ratings. Lesser the value of RMSE, better the quality of predicted ratings.

Mean Absolute Error (MAE):

Using the same ratings notation:

$$MAE = \frac{\sum_{j=1}^{n} |r_{uj} - o_{uj}|}{n}$$

b) Coverage Ratio

Coverage ratio in recommender system is the percentage of items or users which were successfully recommended by the system. We have used here two types of coverage: Item coverage and Catalogue Coverage.

Item Coverage:

Item Coverage is the ratio of items included in the recommendation list over the number of potential items.

$$Coverage_{item} = \frac{n}{N}$$

Catalog Coverage:

Catalog Coverage is the ratio of recommended user-item pairs over the total number of potential pairs. Let L be the number of recommended user-item pairs.

$$Coverage_{catalog} = \frac{L}{N * U}$$

c) Qualitative analysis

To perform qualitative analysis, we compare the items bought by user and the items recommended for the user. For a content-based system the recommended items must be similar to the items bought. Thus, the recommended items might have the same brand or same description or similar style or similar color.

8. Results

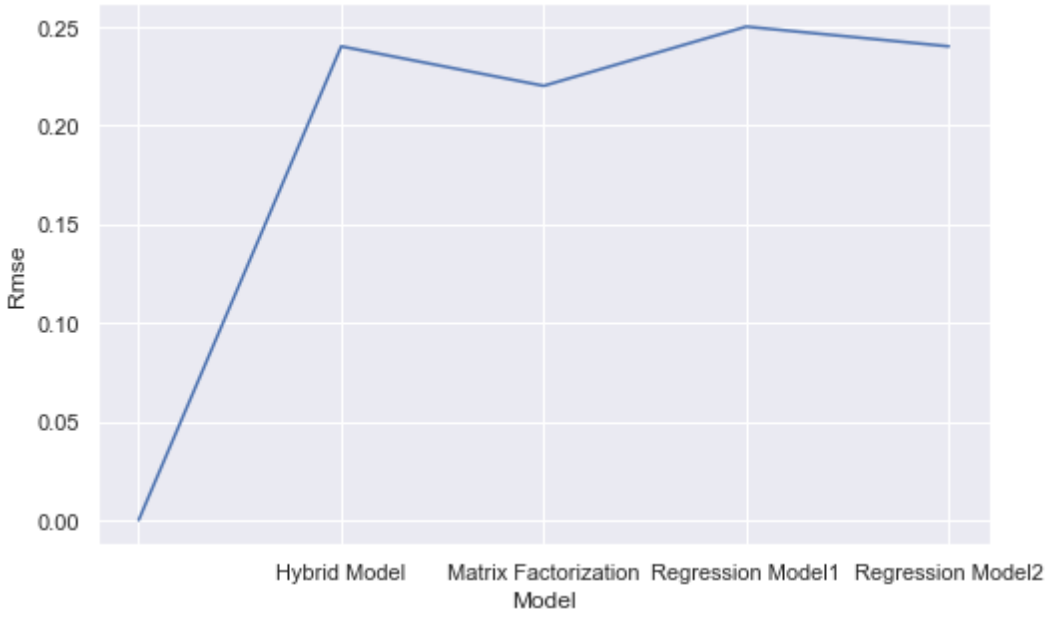
Content based system:

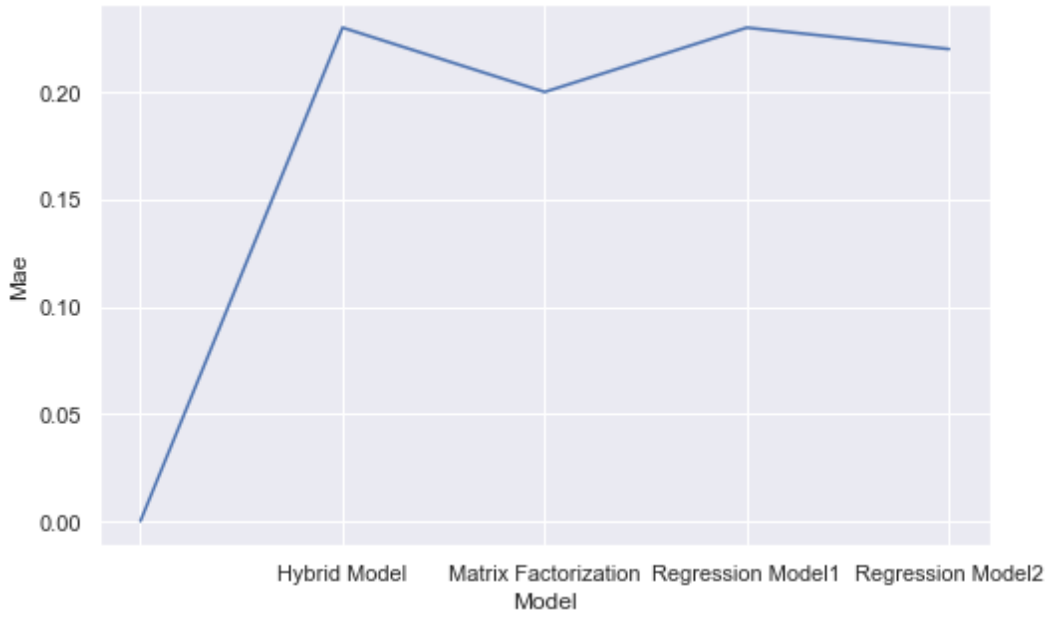
The content-based system was tested on subsets of increasing size. The baseline model used was Matrix Factorization using SVD. The item and catalog coverage are calculated for the scenario that each user was recommended his/her top 10 highest rated options using that model.

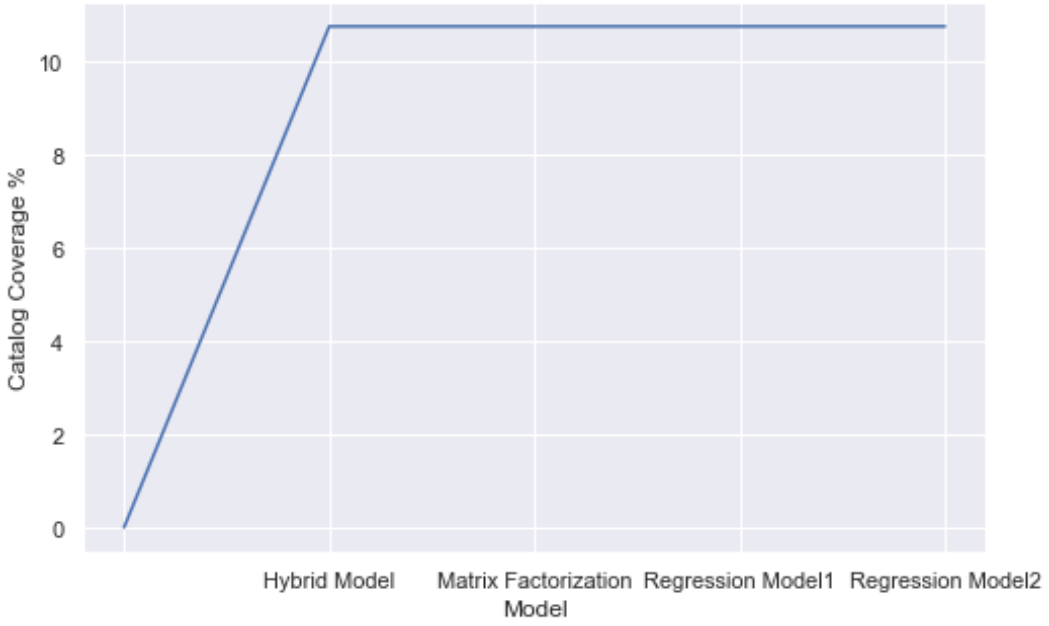
Subset1:

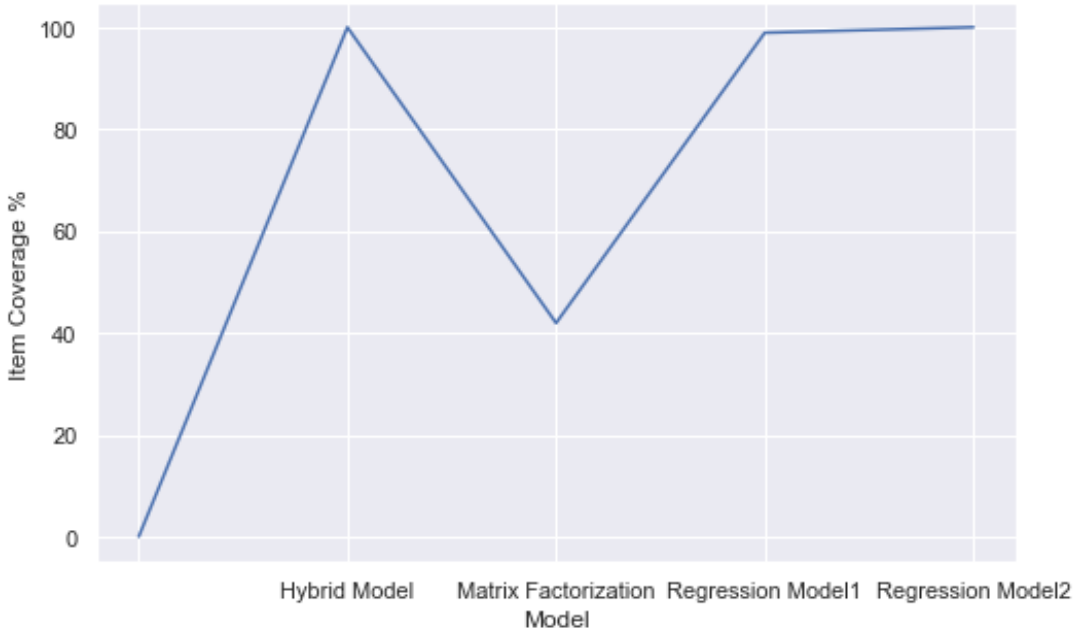
	RMSE	MAE	Catalog Coverage	Item Coverage
Matrix Factorization	0.228086445391 54113	0.201245102210 40084	0.107526881720 43011	0.419354838709 67744
Random Forest Regression based on item text features	0.250112189368 64127	0.235654293983 221	0.107526881720 43011	0.989247311827 957
Random forest regression based on item image features	0.242401342522 15426	0.226379783071 16371	0.107526881720 43011	1.0
Hybrid model	0.246151792492 57121	0.231017038527 19227	0.107526881720 43011	1.0

Graphical Results:




Mae
0.20
0.15
0.10
0.05
0.00
Hybrid Model
Matrix Factorization
Regression Model1
Regression Model2
Model


Catalog Coverage %
10
8
6
4
2
0
Hybrid Model
Matrix Factorization
Regression Model1
Regression Model2
Model


100
80
60
40
20
0
Item Coverage %
Hybrid Model
Matrix Factorization Model
Regression Model1
Regression Model2

Qualitative Results:

Items bought

Recommended using regression on Text features

Recommended using regression on image features

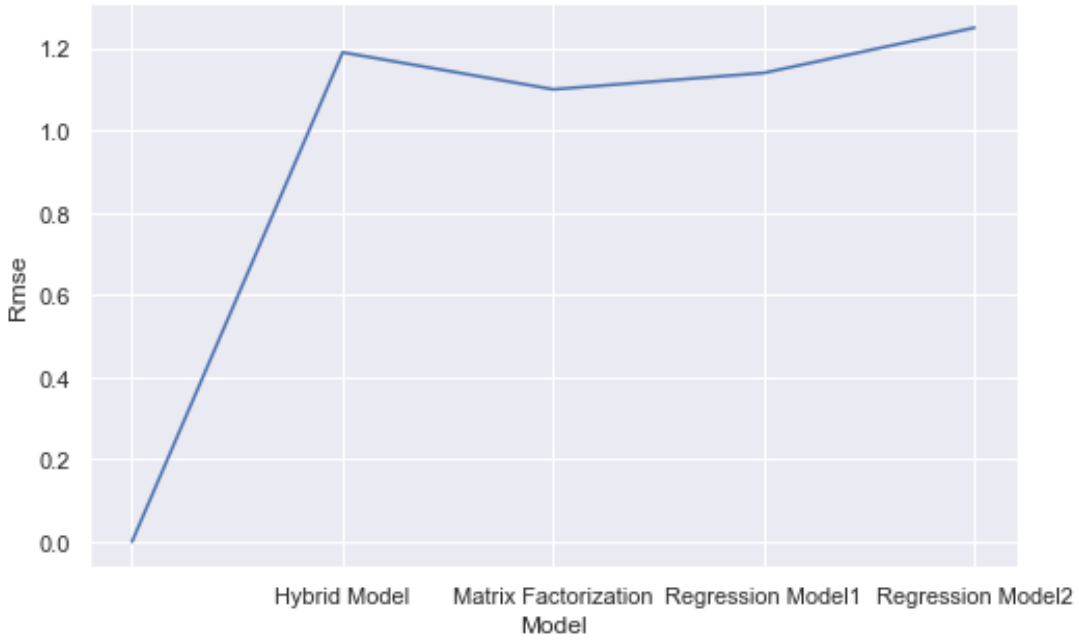
Recommended by hybrid content system

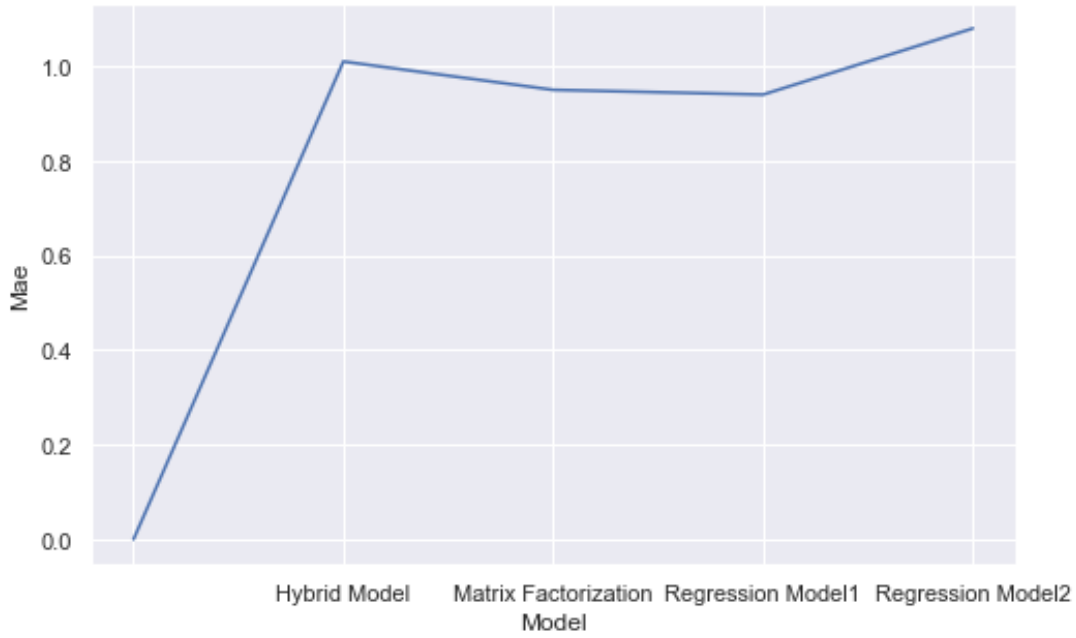
We can see that the user has bought jewelry, black boots and a black top. The recommended items are similar to these items. Our hybrid model recommends jewelry, a white top, black boots and black shoes.

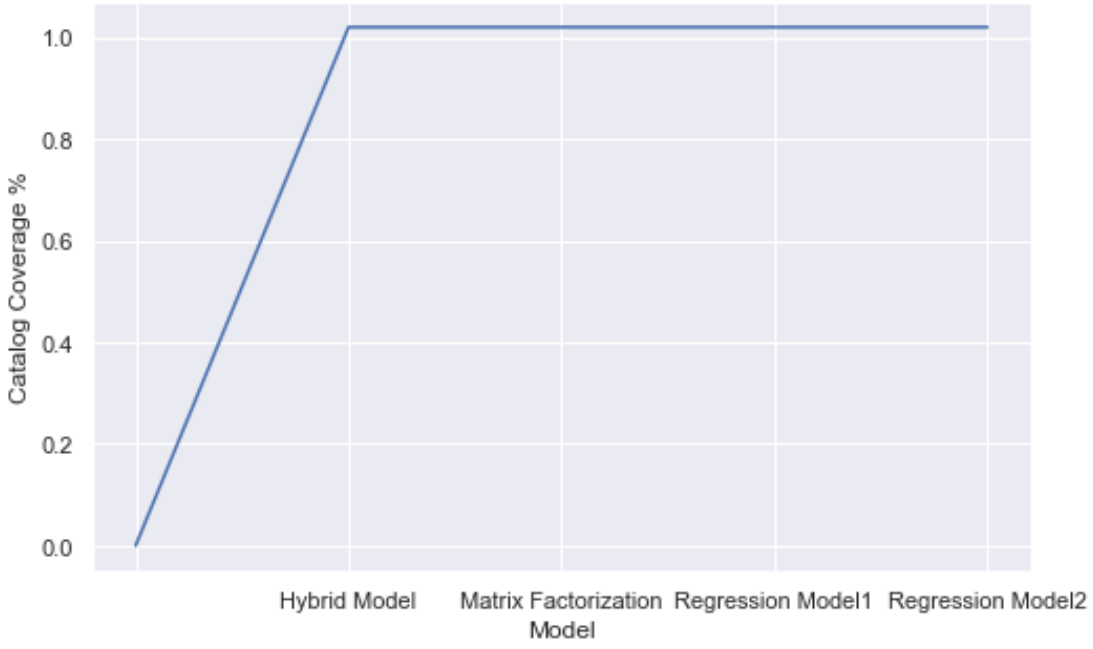
Subset2:

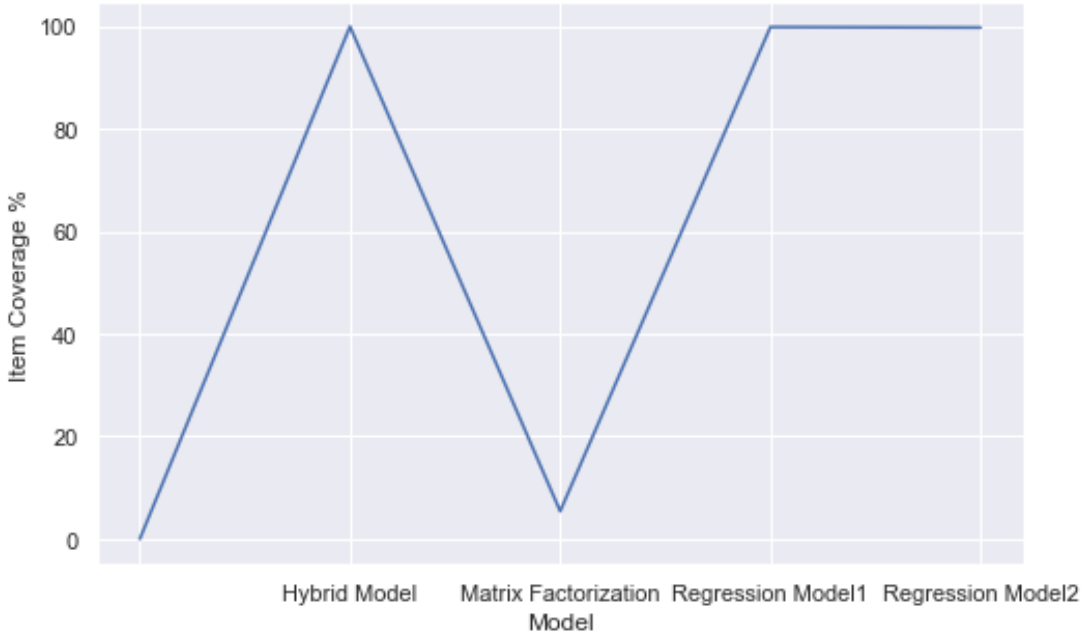
	RMSE	MAE	Catalog Coverage	Item Coverage
Matrix Factorization	1.100702170262623	0.956923633758772	0.010224948875255624	0.054192229038854 8
Random Forest Regression based on item text features	1.142753421581759	0.944284754274021 9	0.010224948875255624	0.997955010224948 9
Random forest regression based on item image features	1.253154416819592 7	1.080046512611596	0.010224948875255624	0.996932515337423 3
Hybrid model	1.190968768117731 4	1.012165633442590 8	0.010224948875255624	0.998977505112474 4

Graphical Results:




Mae
1.0
0.8
0.6
0.4
0.2
0.0
Hybrid Model
Matrix Factorization
Regression Model1
Regression Model2
Model


Catalog Coverage %
1.0
0.8
0.6
0.4
0.2
0.0
Hybrid Model
Matrix Factorization
Regression Model1
Regression Model2
Model


Item Coverage %
100
80
60
40
20
0
Hybrid Model
Matrix Factorization Model
Regression Model1
Regression Model2

Qualitative Results:

Items bought | *Recommended using regression on Text features* | *Recommended using regression on image features* | *Recommended by hybrid content system*

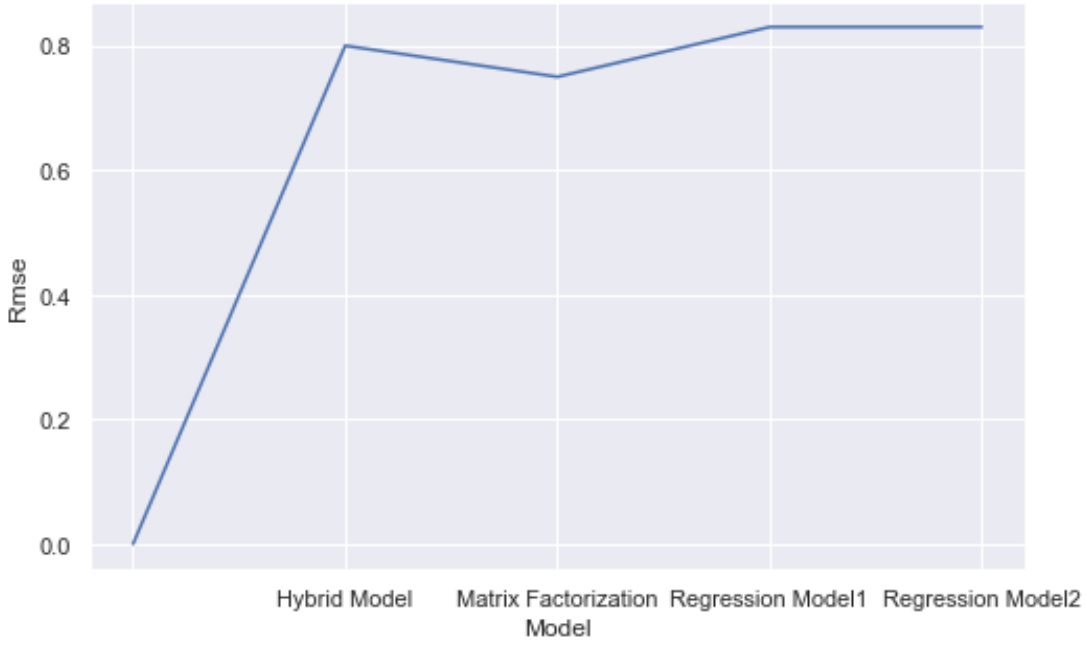
We can see that the items recommended to the user are either based on a style preference or color preference or category preference. The items recommended are very similar to the items bought by

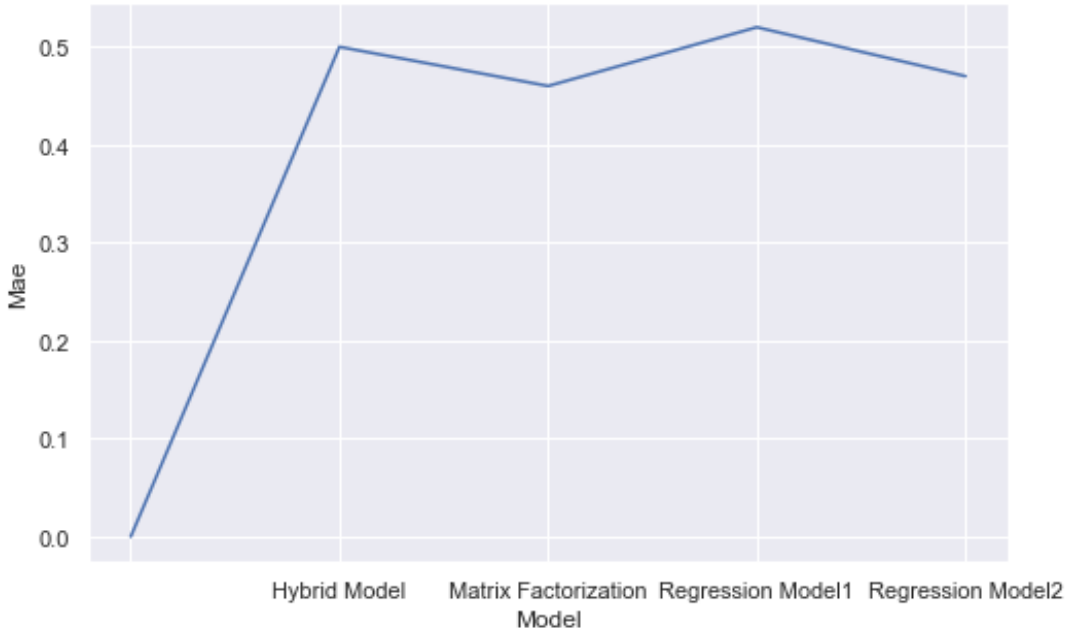
the user. Here we have a serendipitous result as well. Boots have been recommended for the user even though the user has not bought boots.

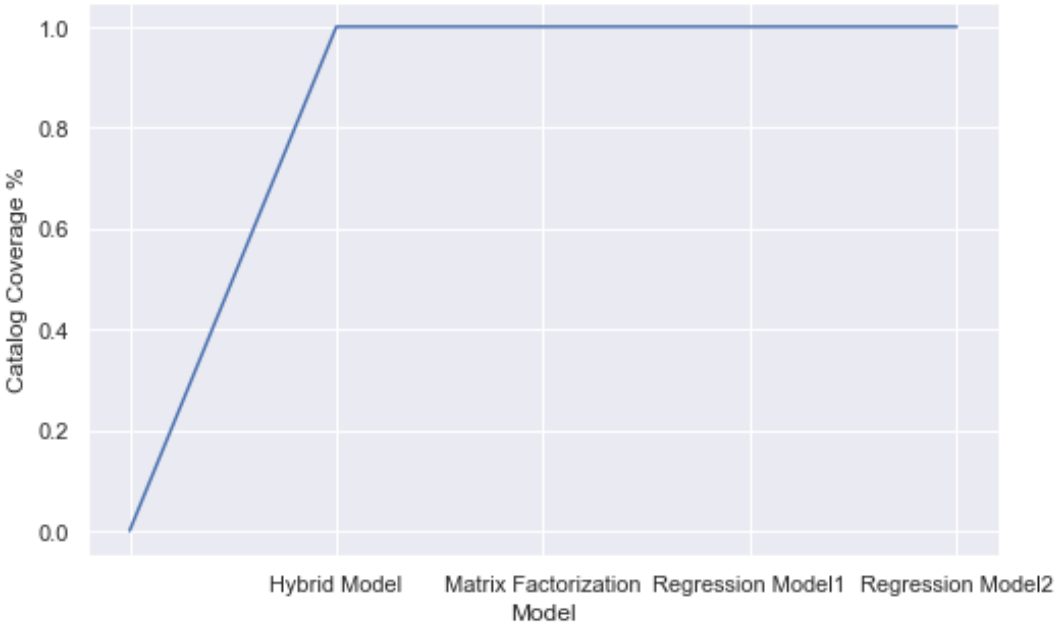
Subset3:

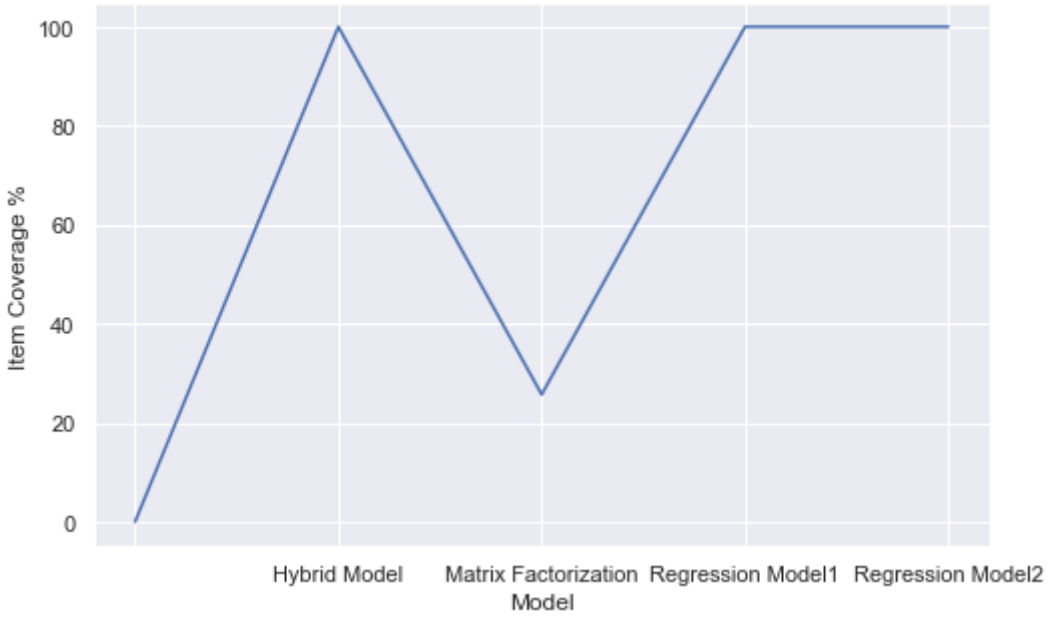
	RMSE	MAE	Catalog Coverage	Item Coverage
Matrix Factorization	0.7459938377296096	0.46726651660557417	0.01	0.257
Random Forest Regression based on item text features	0.8384407228439325	0.529589603175292	0.01	1.0
Random forest regression based on item image features	0.8315855617724708	0.4733229084129558	0.01	1.0
Hybrid model	0.8088444794299012	0.501456255794124	0.01	1.0

Graphical Results:




0.5
0.4
0.3
0.2
0.1
0.0
Mae
Hybrid Model
Matrix Factorization
Regression Model1
Regression Model2
Model


1.0
0.8
0.6
0.4
0.2
0.0
Catalog Coverage %
Hybrid Model
Matrix Factorization
Regression Model1
Regression Model2
Model


100
80
60
40
20
0
Item Coverage %
Hybrid Model
Matrix Factorization Model
Regression Model1
Regression Model2

Qualitative Results:

Items bought | *Recommended using regression on Text features* | *Recommended using regression on image features* | *Recommended by hybrid content system*

The user bought shoes and jewelry. The items recommended by the hybrid model include earrings similar to the ones bought by the user. It also includes a necklace and socks even though user has not bought these items. So, we get serendipitous results. The model also recommends shoes like the one's user has bought.

9. Overall Results:

We observe that we get comparable performance to Matrix Factorization with our Hybrid Model when the metric used are RMSE and MAE. We get the same catalog coverage for both models. But the Hybrid Model outperforms the Matrix Factorization method by a significant margin in item coverage. We get 97-100% item coverage using our Hybrid Model. This means that almost all items were recommended to at least one user. While the Matrix Factorization method gives a 5-41% item coverage. This means that more than half the items were not recommended to any user.

User facing challenges:

- Accuracy: The items recommended are relevant since it is based on user history or user preferences. We get comparable performance to matrix factorization when the metric used is MAE or RMSE.
- Interpretability: The recommendations can be interpretable. The recommendations made by the content-based model should be arranged under a Based on Your Purchase History section and the recommendations by the collaborative filtering method should be arranged under People who bought this also bought this section.
- Diversity: There is a balance between items recommended based on past behavior and items recommended based on user preferences. This leads good diversity in recommendations.
- Rich get richer: The model does not suffer from this problem. We get a high item-coverage (97% to 100%). This means that almost all items have been recommended to users. This implies that the popular items are not the only items recommended. This solves the rich get richer problem.
- Filter bubble: The content-based system can lead to a filter bubble. This is because items similar to the items bought get recommended. Thus, only items in the same category as the items bought might get recommended. This is why we also get recommendations from a collaborative model.
- Serendipity: Content based models can lack serendipity. But we did not see this our results. We did get a few surprising recommendations. This is because we did not just use item-item neighborhood-based recommendation. We used a regression model for each user.
- Time variance: Content-based system can pick up on long term preferences. This solves the time variance problem.
- Aspirational vs actual: Recommendations are based on user history and preferences, as indicated by the user's purchase history. Hence, we do not get aspirational recommendations.

Engineering concerns:

- Cold start: The content-based model solves the cold start problem for new items. Since we have the item features for the new item, we can match it users based on their user profile.
- Scalability: Training the regression model for each user is a time-consuming process. It must be done offline and stored. Once we have a user model for each user the prediction of rating for any item is very quick. Hence the content-based model is scalable with offline training of user models. LSH has been implemented in Spark framework and it's a randomized algorithm which has been designed in a way so that the scalability issue can be solved.
- Real-time vs batch: Recommendations are real-time depending on items in subset. As new items get added the recommendations may change.

10. Future Work:

Instead of using a mixed hybrid model of content based model and collaborative model, we can use a switching hybrid model that initially uses a content-based model and then switches to a behavior-based model as we get more information regarding the user behavior.

We can also perform time-series analysis of our dataset to get seasonal trends and style trends in apparel. This is because the apparel we buy depends highly on current season and current trends.