Fake Review Identification

And Its Application in Recommender System

-- A study inspired by Amazon

Team Alexa (#5)
Alexa Yang, Dipeeka Bastola
Susie Bai, Fei Hou

Business Understanding

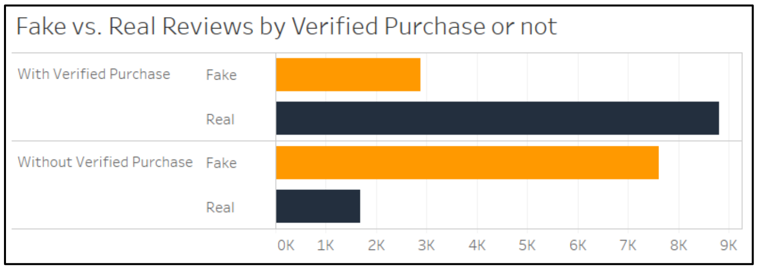
Amazon, the largest online retail platform, is valued by customers not only for the variety of its products and services but for its recommendations. Amazon currently uses item-to-item collaborative filtering to generate recommendations for its users. However, not all Amazon reviews are authentic or real. These fake reviews not only provide the customers with an unrealistic view on an item but also hurt Amazon's ability to recommend relevant products and services to its customers. For this reason, our project will first focus on detecting fake reviews using Amazon reviews and product data. After predicting the fake reviews from real ones, we intend to use the authentic reviews to create a product recommender system.

The first part of the project would add business value because it would help Amazon identify fake reviews and ratings as well as take actions against them. It is important to identify authentic reviews from fake ones because 20% of sales in online platforms are driven by reviews and one-third of online shoppers refuse to buy products that does not have positive reviews (Wesie, 2017). Positive fake reviews (by seller) inflate the value and popularity of a product while negative reviews (placed by competitors) might harm a credible product. Hence, the results of this classification model will help Amazon identify fake reviews as well as take them down from its platform to maintain the credibility of its ratings and reviews. This provides buyers with access to transparent and accurate insights from other customers on various products and services.

The second part of the project involves first building a recommender system using reviews from the luxury beauty category. We will then apply our model from the first part to filter out fake reviews from the same category. With this newly filtered out subset of only genuine reviews, we build another recommender system and compare the results with those of the first recommender system using both real and fake reviews. In addition to showing the impact of fake reviews on recommender system performance, this part will help Amazon identify the best recommender system for their products. The integration of both these steps will help Amazon base its recommendations on real reviews without distortion created from the fake ones.

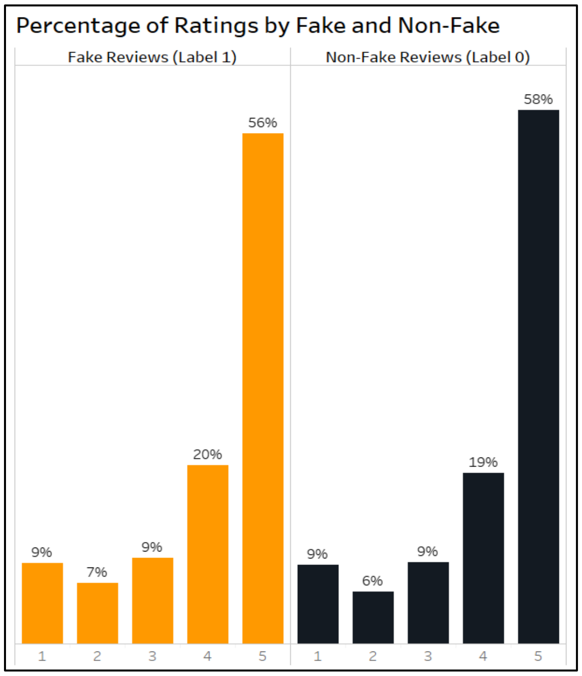
Data Understanding and Exploration

Deception Dataset: This dataset is a text file that contains 21,000 Amazon reviews along with the label about whether the review was fake or not. This dataset was obtained from a GitHub repository. The dataset is balanced with 10,500 instances each for fake and real review, indicating a base rate of 50% for the classification problem. In addition to the data being balanced in terms of target variable, it was also balanced in terms of product categories included, as the data included equal number of observations from each of the 30 categories. After removing irrelevant variables for the model such as document id, there are 8 variables in the dataset – seven potential features and one target variable. This included features such as rating, verified purchase, product information (category, id and title) and review information (title and text). The summary table about data types for feature variables is in Appendix II. Among the features, we wanted to explore how the verified purchase differed based on whether a review was fake or not. The intuition behind this is that most fake reviews must come from unverified purchased than verified purchases. Our intuition was accurate as seen on the table in Appendix III and figure below, where most of the fake reviews come from not verified purchases and most of the real reviews come from actual reviews.



Verified Purchase based on Fake and Real Reviews

Similarly, we wanted to explore the distribution of ratings among the fake and non-fake labels. As seen from the figure below, the distribution of rating among the fake and non-fake reviews are similar, indicating that rating alone might not be a good indicator of whether a review is fake or not.



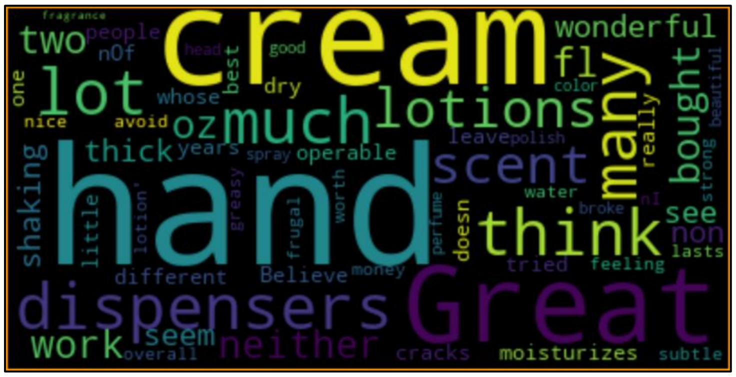
Percentage of Ratings by Fake and Real Reviews

Amazon Reviews Dataset: This dataset is a json/csv file containing the Amazon ratings and reviews for products in the luxury beauty category. It contains 574,228 reviews along with information on time of review, information on reviewer, product, whether the purchase was verified or not and usefulness vote. Considering only the variables relevant to the model, there are 7 variables related to the reviews in addition to the metadata on the products which contain 10 variables. The summary table about data types for the variables is in the Appendix III.

Fake Review Detection Modeling

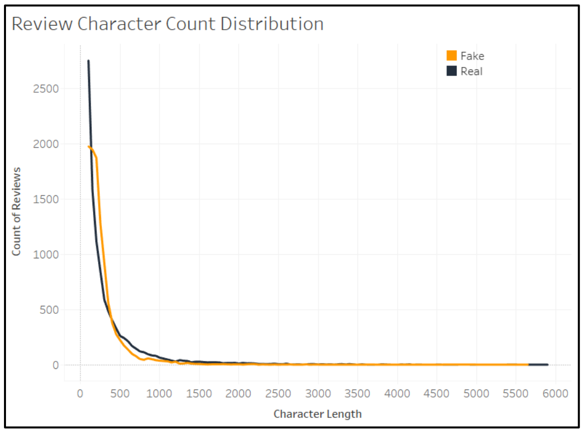
Data Preparation for Fake Review Detection: Based on our business problem formulation, we decided to use the deception dataset as our training and validation data for the fake review identification model because it had labels for whether a review was fake or not. The Amazon reviews data was used as the set where the model was deployed because this dataset did not have fake/non-fake labels. Keeping this in mind, we had to make transformations to data in the Amazon reviews dataset similar to the transformations made to the deception dataset (training). First, we encoded the string labels as binary field – 1 for fake reviews and 0 for non-fake reviews. Further, we identified and retained only the columns that were common to both the datasets, which included columns such as rating, reviews, product ID, whether it was a verified purchase or not, review title and review text. Because of the limited number of features after filtering, we performed text analytics and feature extraction to create additional features for the model.

Text Analytics and Feature Extraction: Since extracting features consist of using text in reviews, we filtered out any rows that did not contain any review text. Generally, text is pre-processed before creating features; however, while analyzing whether reviews were fake on not, we deemed it important to analyze the writing structure of the reviews, such as number of capital letters, punctuation marks and so on. Hence, we created some features before processing the text into meaningful words. These features include length of reviews, count of stop words, count of capital letters, count of punctuation marks and count of emojis. Additionally, the review text was analyzed for a Flesch-Kincaid score. The Flesch–Kincaid Grade Level scores are scores that indicate how difficult to understand a passage is in English; lower the score, the harder it is to understand these words (Hky, 2016). This score is calculated based on word length and sentence length with weighting factors. After creating these features, we then processed the review text by removing punctuations and stop words, stemming and converting to lower case. This was done to retain only the meaningful words for feature extraction. Following this processing step, we created 100 features for most common words in the beauty category along with counts of those words in each review using bag of words. The figure below shows a word cloud representation of the most common words in the luxury beauty reviews. The same features as training data were extracted for the luxury beauty review dataset; the top 100 words from training set kept the same since the model is trained on the features from training data.



Most Frequent Words used in Luxury Beauty Category

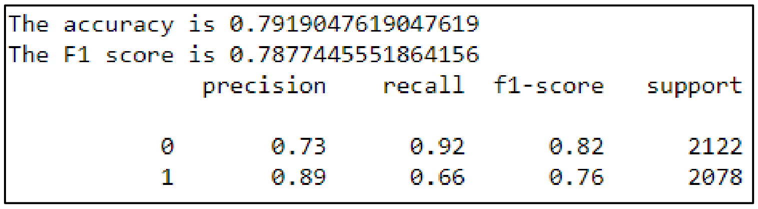
Insights from Text Analytics: Exploring the training data based on the features extracted above, some differences were seen between fake and non-fake reviews. The Flesch-Kincaid score for fake reviews were on average lower (indicating more difficult to understand) than non-fake reviews. Similarly, fake reviews have fewer punctuation marks and capital letters. This shows that some reviewers over-compensate in making their reviews look more authentic by using more complicated language and fewer symbols and capital letters. Finally, as seen in the figure below, large proportion of fake reviews are of shorter length (less than 500 characters) and the average length of fake reviews (316) is shorter than the average non-fake review.



Distribution of Review Character Count by Fake or Non-Fake

Fake Review Classification Model Exploration

The fake review prediction is a supervised binary classification problem, where the target variable is whether the review was fake (1) or not (0). This is a supervised problem because the label for fake review exists in the dataset being used. A data instance in this dataset is an individual customer review for a product. The features that are used for the prediction are product rating, verified purchase, product id and all the features extracted from the text analytics described above. With the model being perfectly balanced, the majority rule has an accuracy of 50%. We then start exploring the modeling process by building a decision tree, which is our baseline model. Using GridSearch to optimize the decision tree hyper-parameters, we have the following performance on the validation data using a decision tree with max_depth of 5 and gini criterion.



```
The accuracy is 0.7919047619047619
The F1 score is 0.7877445551864156
              precision    recall  f1-score   support

           0       0.73      0.92      0.82      2122
           1       0.89      0.66      0.76      2078
```

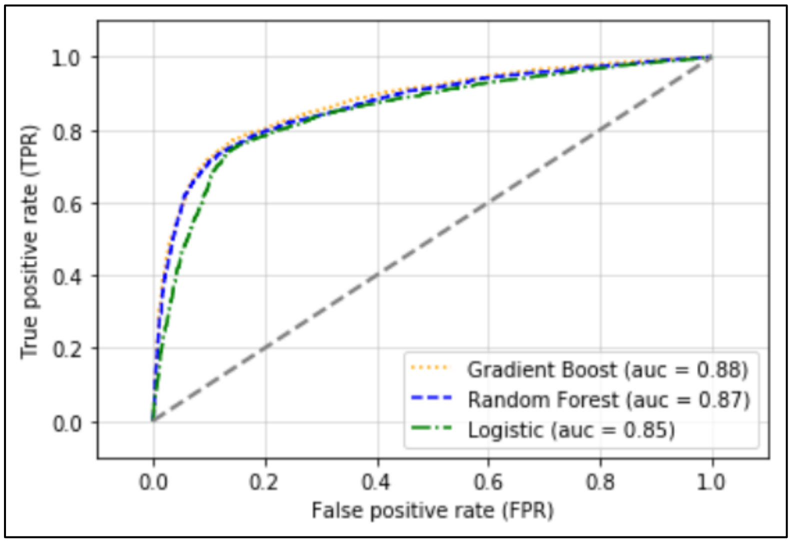
Performance for Base Model: Decision Tree

With a decision tree being the simplest tree-based model, this provides a good performance basis for the other models. We explore other tree-based models with gradient boosting and random forest classifiers as well as linear models such as logistic regression. Using similar GridSearch methods to tune the hyper-parameters of each model, we obtain the following performance measures based on the validation dataset.

Performance Comparison for Linear and Non-Linear Models

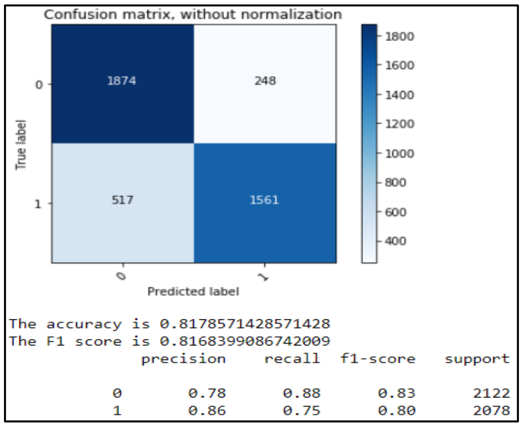
	Accuracy	F1-Measure	Recall - 1	Recall - 0	Precision - 1	Precision - 0
Decision-Tree	0.79	0.79	0.66	0.92	0.89	0.73
Random-Forest	0.82	0.82	0.73	0.90	0.88	0.77
Gradient Boost	**0.82**	**0.82**	**0.75**	**0.88**	**0.86**	**0.78**
Logistic	0.81	0.81	0.75	0.87	0.85	0.78

From the table of performance outputs above, we see that most of the models have similar performance besides the simple decision tree. However, the decision tree has the highest negative recall, meaning it performs the best in capturing genuine reviews in its predictions. On the flipside, its positive recall is terribly low with 0.66, so we exclude it from our candidate list of final models. To gain a more comprehensive view of how the models can detect fake reviews without wrongfully classifying genuine reviews as fake, we plot the ROC curve of the classifiers. As seen below, Gradient Boosting has the highest AUC value of 0.88, followed by Random Forest with 0.87, and Logistic with 0.85. The shape of the ROC curves of all three classifiers are similar. Therefore, the Gradient Boosting classifier is selected as our final model for this prediction.



Model Evaluation

Our final fake review detection model is a Gradient Boosted Tree classification model. Using GridSearch to find the best generalizing performance based on a validation dataset, the optimal max_depth of the individual tree predictors in the model is 4. We then apply the model to a hold-out test set to get a final unbiased estimate of classification performance as shown below.

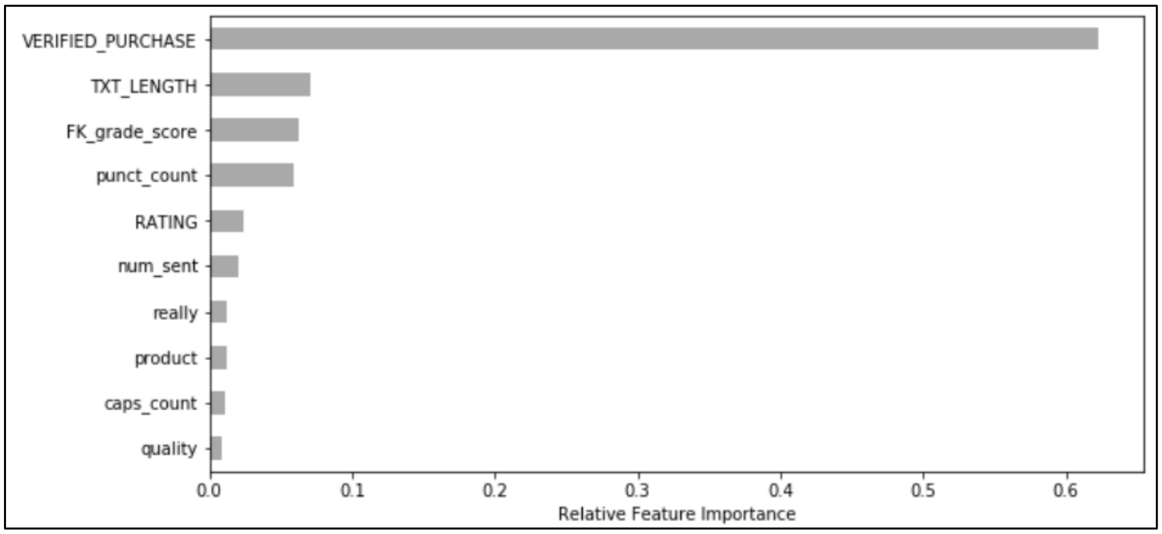


```
The accuracy is 0.8178571428571428
The F1 score is 0.8168399086742009
              precision    recall  f1-score   support

           0       0.78      0.88      0.83      2122
           1       0.86      0.75      0.80      2078
```

Performance of Selected Model: Gradient Boost

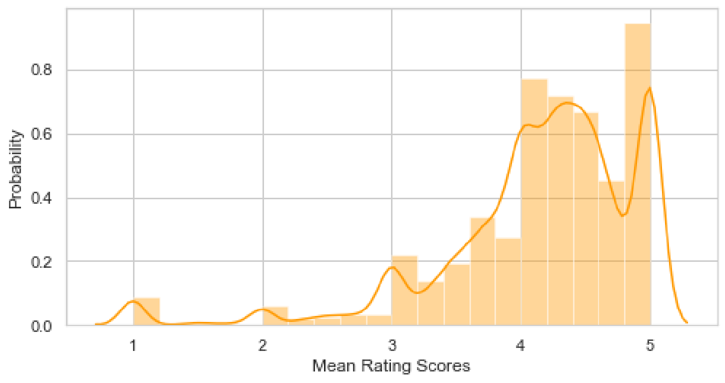
Even though our dataset is perfectly balanced, the model seems to be more biased towards genuine reviews and slightly conservative, classifying 60% of the reviews in the test set as genuine. This is evidenced by the higher negative recall value of 0.88 compared to the positive recall of 0.75. The most important features that provide the model's predictive power are shown in the chart below. Confirming our idea from the data exploration phase that a verified purchase is highly indicative of a genuine review, we see that VERIFIED_PURCHASE holds the most predictive power compared to all other variables. Other important features include the Flesch–Kincaid Grade Level score, character length of the review, as well as punctuation count.



Relative Feature Importance for Final Model

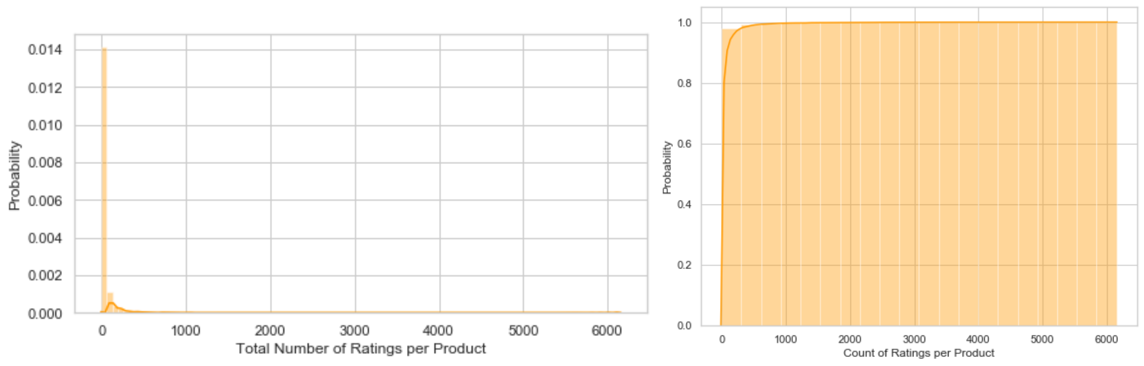
Recommender System Modeling

Data Preparation and Exploration: The dataset used for the recommender system is the Amazon reviews dataset (where we will deploy our fake review detection model). The data is input in the recommender system after predictions on whether the reviews are fake or not is made by the previous model. In order to base the recommendations on actual reviews, the reviews that are predicted fake are filtered out of the dataset before feeding into the recommender system. The filtered dataset has 480,190 reviews, written by over 360,000 users for around 12,000 products. The mean ratings per product range from 1 to 5 with a left skewed distribution as shown below, where most products have ratings above 3.



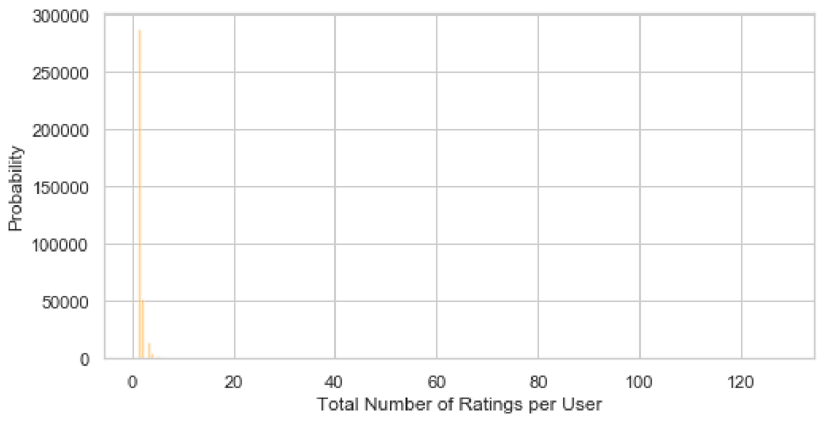
Distribution of Mean Rating per Product

As shown below in cumulative curve of count of ratings per product (right) and distribution of ratings (left), most products have ratings less than 10 with only few products with very high number of ratings.



Distribution of and Cumulative Curve of Number of Rating per Product

When we dive into the percentile distribution as shown in Appendix IV, we see the same long-tail characteristics, where maximum number of reviews are 6,148 but the median is much lower at 9 and the value at the 75th percentile is 30. In order to reduce the computational time required, we only take the top 25% percentile of products with highest number of ratings into account. We filter out products with less than 30 ratings and lose only a small portion of the total review data with number of instances shrinking to 480,190 (14% less than before). Similarly, the figure below shows that even number of ratings per customer is very skewed and has a long tail. We are not dealing with this issue right now, as recommendations to be made for all customers may be of interest first in place in practice; but certain actions would be taken in specific parts of modeling sections later to address this issue.

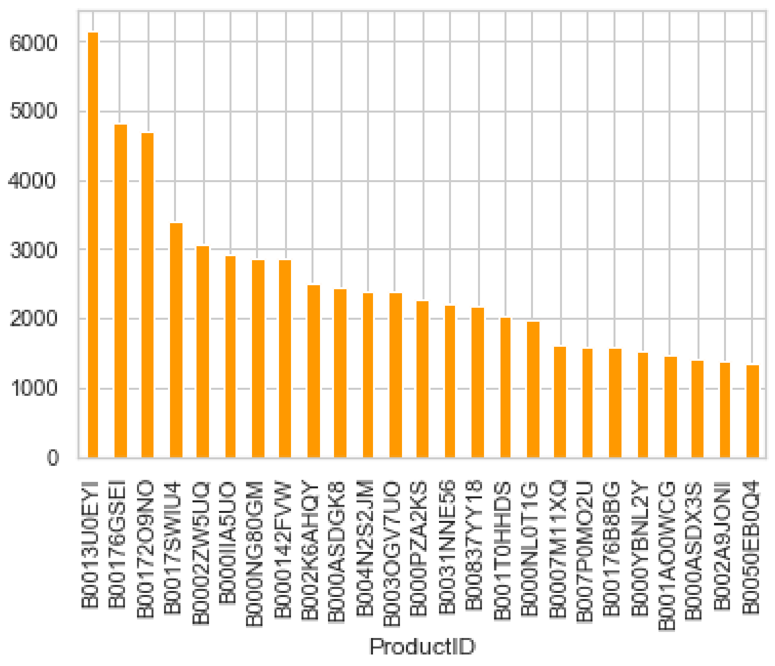


Distribution of Number of Ratings per Customer

The next step after exploring the dataset is to build a recommendation system, in order to predict the best items to be recommended for each customer. This recommendation system could be based on a customer's past records of rating items they purchased, or the purchase/ rating habits of other customers that may share similar preferences. For this analysis, we focus on collaborative filtering mechanisms using past user-item interactions, as we do not have much data on more specific attributes of individual users or items for content-based mechanisms. In this stage, we basically build the recommendation system models by applying two kinds of analytical tools – RapidMiner (with Recommender Extensions) and Python (in Jupyter Notebook). The main evaluation metric for our recommender system models is RMSE.

Cold Start by Most Popular Products

As the data set does not contain any information on characteristics or features of products, memory-based recommendation system would be inaccurate for new users/items. We chose to recommend the most popular items purchased by customers to resolve the cold start problem. We use the total count of reviews as a measurement of popularity of product. The following bar chart plots the top 25 most popular products, which could be recommended to a new customer.



Top 25 Most Reviewed Products in Luxury Beauty Category

Recommender System Algorithms and the "Surprise" Library

Using the Surprise library in Python, we will build the following recommender system algorithms for benchmarking and performance comparison.

Normal Predictor: Normal Predictor algorithm gives a random rating based on the distribution of the training set, which is assumed to be normal. Obviously, it will not perform well and only provide a baseline of other models.

Baseline Only: Baseline Only algorithm predicts the baseline estimate for given user and item.

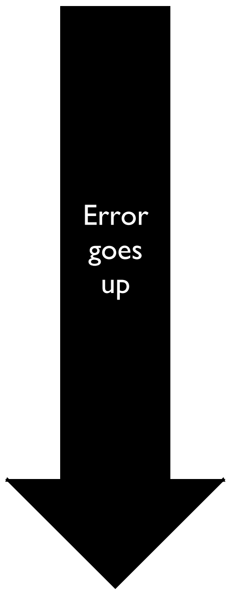
$$\widehat{r_{ui}} = b_{ui} = \mu + b_u + b_i$$

If user u is unknown, then the bias b_u is assumed to be zero. The same applies for item i with b_i

k-NN algorithms: There are four sub-algorithms that were explored: *KNNBasic, KNNBaseline, KNNWithMeans, KNNWithZScore*. *KNNBasic* is a basic collaborative filtering algorithm. *KNNBaseline* is a basic collaborative filtering algorithm considering a baseline rating. Similarly, *KNNWithMeans* is basic collaborative filtering algorithm, considering the mean ratings of each user. *KNNWithZScore* is a basic collaborative filtering algorithm, considering the z-score normalization of each user.

Matrix Factorization-Based Algorithms: Surprise provides five Matrix Factorization-based algorithms: Slope One, Co-clustering, SVD, SVD++, NMF. SVD algorithm is equivalent to Probabilistic Matrix Factorization while the SVDpp algorithm is an extension of SVD that considers implicit ratings. NMF is a collaborative filtering algorithm based on Non-negative Matrix Factorization, which is very similar with SVD. SlopeOne is a straightforward implementation of the SlopeOne algorithm and Coclustering is a collaborative filtering algorithm based on co-clustering. After building the models described above, the RMSE results for each model as the main performance metric is given. The result based on three-fold cross validation is shown below. The recommendations did not consider user-based collaborative filtering because it demands high memory, which was outside the limit of our computational resources.

Recommendation System Algorithms and Performance Comparison



Algorithm	test_rmse	fit_time	test_time
KNNBaseline	1.167857	2.073299	1.402192
SVD	1.191643	14.764405	1.01096
KNNWithZScore	1.197279	0.557455	1.464376
KNNWithMeans	1.197284	0.419854	1.176038
SVDpp	1.198505	25.466587	1.030109
KNNBasic	1.199923	0.318469	1.367486
CoClustering	1.223554	22.115091	0.548205
SlopeOne	1.225996	1.863536	0.768827
NMF	1.242295	27.246579	0.660897
BaselineOnly	1.248616	1.390677	0.602726
NormalPredictor	1.642377	0.424514	0.788906

The table is sorted from lowest RMSE on top, then down to the highest RMSE on bottom. We can see that the best model is KNN Baseline, followed by SVD, SVD++, and three of the KNN approaches. Regarding the fitting time, KNN family are the shortest as well. However, we only trained the model with their default parameters. Next, we will focus on tuning model parameters usning GridSearchCV, which combines Grid Search and cross validation. The best model regarding RMSE is SVD, with the lowest RMSE of 1.165 with the parameters: 'n_factors': 50, 'lr_all': 0.05, 'reg_all': 0.04.

Recommender System Algorithms in RapidMiner

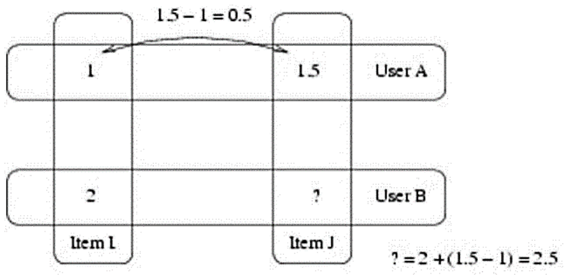
Similar process is undertaken in RapidMiner to select the best recommendation system model. For the training-testing data split across all models explored, we use fixed splits at a 70:30 ratio instead of cross validation (cross validation may simply worsen the issue for some originally time-consuming processes). We mainly apply the recommender models in the "Item Rating Prediction" module in the RapidMiner extension for the sake of comparability of evaluation metrics.

We pick the default *User Item Baseline* RapidMiner operator as our baseline model here, which uses the average rating value accompanied with a regularized user and item bias for prediction. This model does not seem to have very good performance, resulting in an RMSE of 1.246 and a MAE of 0.97, mostly due to it simply using the average. We then try two fundamental methods in the neighborhood-based category of approaches for collaborative filtering – *User k-NN* and *Item k-NN*. For *Item k-NN*, we try the 5-neighbor model(k=5) with cosine similarity as our similarity measure, as generally cosine similarity is considered more appropriate for item-based neighborhood method. The performance gets better than the baseline model regarding RMSE, MAE and NMAE. Additionally, hyperparameter tuning of this model is performed utilizing grid search parameter optimization, which results in optimal parameters of *k=3* with Pearson correlation as similarity measure; the performance of the tuned model improves from an RSE of 0.205 to one of 0.198.

As for *User k-NN*, we need to be particularly attentive to this model here, as we impose some extra handling of data on it. When we first ran this model, we found that it always threw out an error message likely relevant to memory overflow during the runtime process in RapidMiner, no matter how we adjust other potential parameters. This is most likely because many customers have only provided 1 or 2 ratings in the data, which would probably lead to computation of a large sparse matrix in the intermediate result that dashes memory. Given this case, we add an additional filtering operation to our modeling procedure – only retaining those user records with more than 2 item ratings – to increase the performance and reliability of our recommendations. The User k-NN model based on the filtered data seems to perform quite better than most of the other models attempted, resulting in an RMSE of 1.077. This result is obtained after we manually tuned around the hyperparameter (number of neighbors) in the model and found that the performance stabilized around *k=35*.

We also tried another kind of model in the item-based collaborative filtering category – Slope One and

Bi-polar Slope One models offered in RapidMiner. The *Slope One* family of models is the simplest form of non-trivial item-based collaborative filtering approach; it has many advantages over other kinds of models for recommendation system, such as easy implementation, much less storage required, reduction of overfitting, and similarly good performance(or even improved) as other more complicated algorithms. These two models available in RapidMiner do not have hyperparameters to tune. We can see they also perform relatively fine.

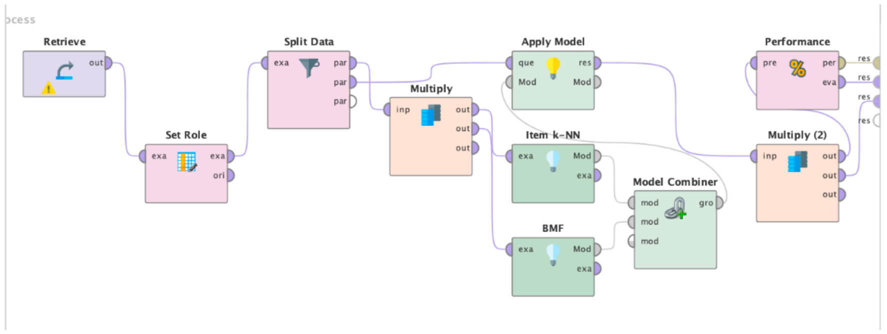


A quick Illustration of Key Idea of Slope One

Besides the category of neighborhood-based approaches, the other large category for collaborative filtering are latent factor collaborative filtering approaches. The main idea of these models is to actively employ matrix factorization to decompose the user-item interaction matrix into product of two lower dimensional rectangular matrices. In RapidMiner, we have options of simple *Matrix Factorization*, *Biased Matrix Factorization* (matrix factorization with explicit user and item bias added), and *Factor Wise Matrix Factorization* (matrix factorization with factor-wise learning) models in this category. The *Biased Matrix Factorization* model with 30 latent factors and 50 iterations performs the best. Parameter tuning and optimization do not improve performance of these models.

Finally, we try the "hybrid model" methodology broadly defined, which provides us with some compelling progress in performance. As usual, basically any type of model for recommendation system building has advantages and limitations. For instance, while memory-based approaches may be more straightforward to implement, the model-based approaches are better in terms of speed and scalability.

The idea of combining individual recommender models for achieving better performance has proven to be a fantastic one in our case. We implement the hybrid structure by using *Model Combiner* available in the RapidMiner Recommenders Extension. We first consider setting up a combined model consisting of the User k-NN model (with additional filtering) and the Item k-NN model built before. Through respectively applying the two well-performing versions of models found previously, we achieve an RMSE as low as 0.949 (together with lowest MAE and NMAE) in this specific case (additionally filtered user data), which has already been a very considerable improvement over the previous User k-NN model with RMSE of 1.077 (on top of additionally filtered user data as well). Then, we also try another hybrid model by incorporating Item k-NN model and Biased Matrix Factorization model, which present relatively good performance previously, and provide a final RMSE of 1.177. Although not a remarkable progress as compared to the prior cases, it still shows a decent leap in performance compared to all the other previous models based on the same data.



Hybrid Model using Item k-NN & Biased Matrix Factorization (on processed data) in RapidMiner

The results of various models described above are presented in the following page, including the main metrics – RMSE and MAE, as well as NMAE (Normalized Mean Absolute Error).

Performances for All Recommendation System Models Explored (Details in Appendix)

Model	**RMSE**	**MAE**	**NMAE**
User Item Baseline (default)	1.246	0.970	0.242
Item kNN	1.205	0.882	0.220
*User kNN	1.077	0.823	0.206
Matrix Factorization	1.214	0.907	0.227
SlopeOne	1.200	0.859	0.215
Bi-Polar Slope One	1.198	0.875	0.219
Factor Wise Matrix Factorization	1.273	1.001	0.250
Optimized ItemkNN	1.198	0.889	0.222
Optimized Matrix Factorization	1.204	0.888	0.222
Optimized Biased Matrix Factorization	1.193	0.890	0.223
Optimized Factor Wise Matrix Factorization	1.271	1.001	0.250
Hybrid Model using *User kNN & Item kNN	0.949	0.685	0.171
Hybrid Model using Item kNN & Optimized Biased Matrix Factorization	1.177	0.881	0.220

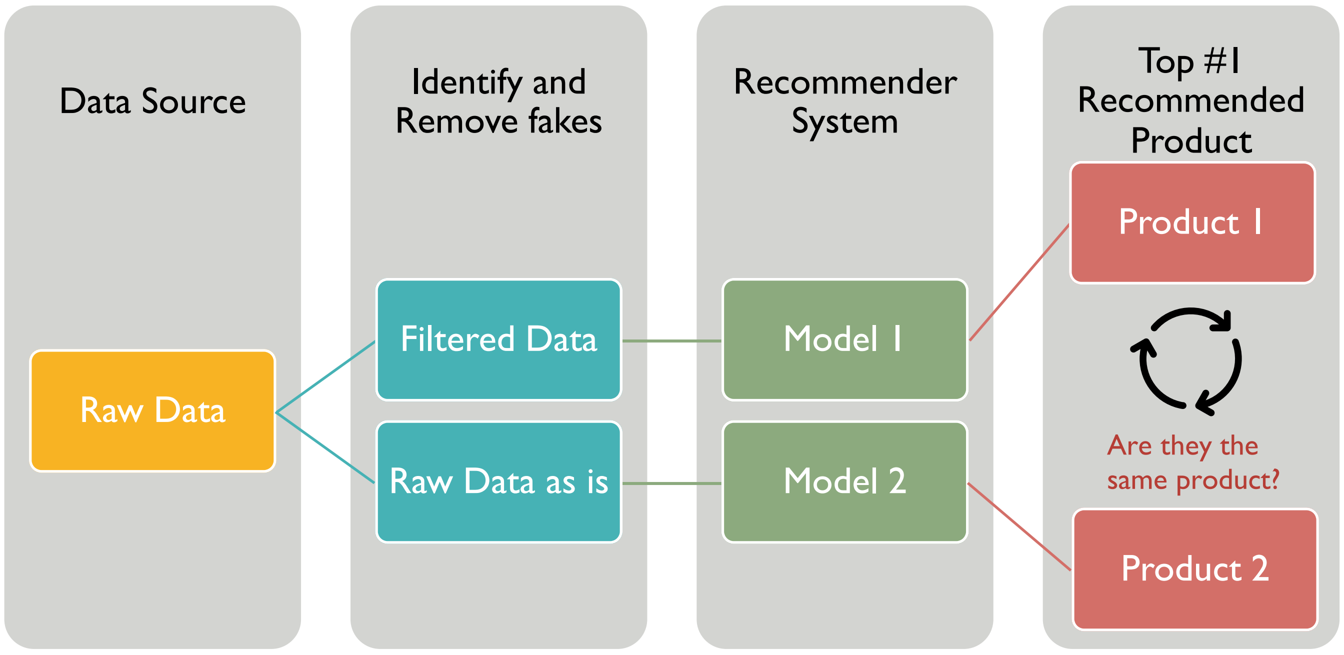
**Note: an additional filter on users is specifically applied for User k-NN model to prevent memory overflows. The filter sifts out the records of users who made less than 2 ratings; 21316 data observations remain after the preprocessing of filtering in this case.*

As a result, we conclude here that the hybrid type models perform better in this recommendation problem. This aligns with the reality in the industry, where more and more companies are leaning towards making hybrid recommenders. In an overview of the construction of recommender systems using Python and RapidMiner, we may infer that the quality and the size of meaningful data about user-item interactions would indeed be one of the determinants of model effectiveness. For instance, simply by ignoring products or customers with only a handful of past records, the overall performance of most models might be better. However, in business practices, such problems would almost always be encountered at the beginning of a product or customer lifecycle, especially for newly founded companies. Decision makers need to clearly realize the tradeoff between making value out of relatively established and complete user-product data and exploring as well as accumulating more knowledge of new domains of users or products to be utilized later. This may be a reminder of the famous exploitation-exploration problem, where approaches like multi-armed bandit models have been developed to relieve the issue in other studies.

Effectiveness of Fake Review Detection in Recommender System

When we initially filtered the dataset for just non-fake reviews, we did so with the hypothesis that training a recommender system based on only genuine reviews would perform better. In order to completely test this hypothesis, A/B testing would be required. Although we could not perform A/B testing, we wanted to justify that the fake review detection had a significant impact on the accuracy and relevancy of products recommended to customer. In order to do that, we compare the output of the top one recommended product generated by the models based on the two different datasets: the raw dataset (with predicted fake and non-fake reviews) and the dataset with only non-fake predicted reviews. The process can be visualized as follows.

Process Flow of Comparing Item Recommended using Two Datasets



To perform this test, we choose 23 customers who have bought the most popular products. The reason behind this is that if we remove a certain number of the reviews from these products, our action will have less impact on them compared to removing the same number of reviews from less popular products. This experiment shows that using data classified as non-fake from our fake review detection model does change the recommendations. The result shows that only 1 out of 23 customers received the same recommended product, showing that model output changes the output of recommender system.

No.	Customer ID	Top #1 Recommended Product		Match or Not?
		Based on Raw Data	Based on Fake Trimmed	
1	AENH50GW3OKDA	B000Q2Y04Y	B00CYHM7NU	No
2	AHN86VFJIJ2JP	B002P3L99G	B001G60EK8	No
3	A2YKWYC3WQJX5J	B000G0JZYQ	B00A9TZMQE	No
4	A3091RP0SPZLMN	B0002MPS7G	B004XMAVFA	No
5	A25QBCHO0KFT0P	B000H0SCXU	B00383HHT8	No
6	A2V5R832QCSOMX	B00LO2RO7K	B011J6HJKM	No
7	A2OW1FKQFPIA5D	B004UMYEIS	B00JRW7QCC	No
8	AQY5XBYSENNZQ	B01DQG3DFI	B0157UMHXK	No
9	A2BU0OBCKFDUKE	B0002CEIQ8	B0006PJRVM	No
10	A2JR5ZTSYSIQYX	B01DQG3DFI	B0015VP9CW	No
11	A2GJX2KCUSR0EI	B01ELUGLUW	B001FBCLS2	No
12	A2J6IZXSLF8VB2	B0001Z66UM	B0015VP9CW	No
13	A24FOVVEIEAIPK	B0015VP9CW	B0015VP9CW	Yes
14	AJL6HX8O2QLVO	B000XYPNMS	B001FBCLS2	No
15	A18VCM7Y7HK6EW	B007NPX4LM	B014UC1QE2	No
16	A11Z3ANM4PQFMH	B0010OKF7S	B000XYPNMS	No
17	A2V1MAUHH1YHKK	B0013A6NFM	B003OGV7UO	No
18	AJC7TSNP5UAWN	B002P3L99G	B002QG6P9G	No
19	A3NQA378KXTZQM	B00014340I	B000Q2Y04Y	No
20	A2NT1LGR2BYP0I	B000Q2Y04Y	B000O5WCN4	No
21	A2H6LHCM3AR3YV	B000Q388UK	B01ELUQJ3Q	No
22	A3NMQEQPT5LDIA	B000Q388UK	B001G60EK8	No
23	A1O4PNHACFEDEM	B000Q388UK	B000H0SCXU	No

If we relax the constraint of matching to top three products instead of one regardless of orders of recommended products, we will have a total of 138 products (23*3*2) recommended by the two models. If the two models give out the same results, we would have 69 unique products (proportion = 0.5) and if the two models give out totally different results, the proportion will be 1. In our case, the proportion is 0.97 (134/138), which is very close to one. That means the two models have different outputs; hence, our fake detection model makes a substantial difference in recommender system.

Deployment

The fake review detection model would be deployed internally and externally in Amazon's existing platform. The internal deployment is more straightforward, where the fake reviews would be removed from data used for most predictive analytics that rely on product ratings, such as recommender systems that rely on ratings information. The external deployment is a bit more challenging because it involves deciding whether to flag and remove fake reviews or present them as unreliable reviews in the Amazon platform. The decision of which route to pursue involves discussions, testing and consideration of impact of such removal or unreliability rating on customers and sellers.

Further, it is important to discuss some risks and future improvements for the models presented here. First, we chose to predict the fake and non-fake reviews as binary outcomes because of timing constraint and presence of such labels in the training dataset. However, probability estimates of belonging to the positive class (fake as 1) can also be predicted. This allows Amazon to create its own cutoff for predicting fake and non-fake reviews based on the cost of false positives and negatives. Additionally, seasonality has not been considered in the analysis. This is important because some product sales might vary based on season and it might be important to include a seasonality variable in the model. Similarly, the fake review detection model relies heavily on the accurate data for training data; hence, the accuracy and completeness of the training data is very crucial. Further, when it comes to the development of recommender systems, we present main statistical accuracy metrics (RMSE and MAE) alone here due to length and time limit; yet in might be important to consider key decision support evaluation metrics in further work, such as precision@k (precision (num of recommendations that are relevant / total number of items recommended) at cutoff of first k items recommended), AP(Average Precision) and MAP(Mean Average Precision), for better understanding of performance of our recommender models. Additionally, the exploitation-exploration dilemma suggested in the cold start and matrix sparsity issues of our recommender system building calls for our further attention and would likely be paramount for new business development and profitability generation for companies in real-world complexity of model employment. Finally, there might be some ethical considerations relating to censorship when flagging and removing fake reviews. This removal of fake reviews would be analogous to removing 'fake news' from social media sites. While on one hand, it is important to flag and remove fake reviews to prevent misrepresentation and false opinions, it is also important to do so carefully in consideration of its potential implications on people's freedom to voice opinions.

References

- Hky, S. (2016, March 29). Flesch-Kincaid Readability Measure. Retrieved from https://datawarrior.wordpress.com/2016/03/29/flesch-kincaid-readability-measure/
- Surprise Documentation. (2015). Retrieved from https://surprise.readthedocs.io/en/stable/index.html
- Wesie, E. (2017, March 20). That review you wrote on Amazon? Priceless. Retrieved from https://www.usatoday.com/story/tech/news/2017/03/20/review-you-wrote-amazon-priceless/99332602/

Data Sources

- Deception Dataset: https://github.com/aayush210789/Deception-Detection-on-Amazon-reviews-dataset
- Amazon Review Dataset: http://jmcauley.ucsd.edu/data/amazon/

Appendix

I: Contribution of Group Members

Group Members	Analysis	Write-Up
Alex Yang Dipeeka Bastola	Text Analytics, Fake Prediction Modeling	
Fei Hou Susie Bai	Recommender Systems	

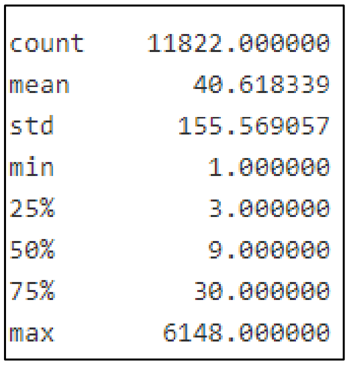
II. Variables in Deception Dataset

Data Type		Number of Variables
Categorical	Binary	I
	Text	5
Numeric	Numbers	I
	Date/ Time	-

III. Variables in the Amazon Reviews Dataset

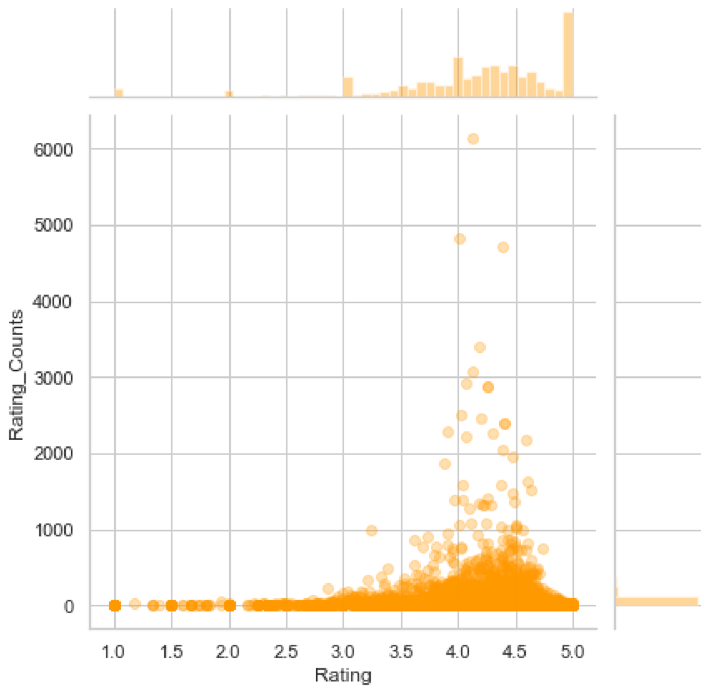
Data Type		Number of Variables
Categorical	Binary	I
	Text	8
Numeric	Numbers	6
	Date/ Time	2

IV. Distribution of Number of Ratings per Product



```
count    11822.000000
mean        40.618339
std        155.569057
min          1.000000
25%          3.000000
50%          9.000000
75%         30.000000
max       6148.000000
```

V. Distribution of Ratings



VII. Performances for All Recommendation System Models Explored with Best Parameters

Model	**RMSE**	**MAE**	**NMAE**
User Item Baseline (default)	1.246	0.970	0.242
Item kNN (k=5; correlation=”cosine”)	1.205	0.882	0.220
(*additionally user-filtered) User kNN (k=35; correlation=”pearson”)	1.077	0.823	0.206
Matrix Factorization (Num Factors=10, Iteration number=30 by default)	1.214	0.907	0.227
Slope One	1.200	0.859	0.215
Bi-Polar Slope One	1.198	0.875	0.219
Biased Matrix Factorization (Num Factors=30, Iteration number=50)	1.194	0.890	0.223
Factor Wise Matrix Factorization (Num Factors=30, Iteration number=50, Sensibility=10)	1.273	1.001	0.250
Optimized ItemkNN (based on grid: {k: [1,20] with 10 steps; correlation_mode: (‘pearson’, ‘cosine’)}) – “k=3, correlation_mode=’pearson’”	1.198	0.889	0.222
Optimized Matrix Factorization (based on grid: {Num_Factors: [20,40,60,80,100]; Iteration_number: [10,40,70,100]}) – “Num_Factors=40, Iteration_number=100”	1.204	0.888	0.222
Optimized Biased Matrix Factorization (based on grid: {Num_Factors: [20,40,60,80,100]; Iteration_number: [10,40,70,100]}) – “Num_Factors=40, Iteration_number=100”	1.193	0.890	0.223
Optimized Factor Wise Matrix Factorization (based on grid: {Num_Factors: [20,40,60,80,100]; Iteration_number: [10,40,70,100]}, Sensibility=10) – “Num_Factors=80, Iteration_number=10”	1.271	1.001	0.250
Hybrid Model using User kNN (*additionally user-filtered) (k=35, correlation_mode=’pearson’) & Item kNN (k=3, correlation_mode=’cosine’)	0.949	0.685	0.171
Hybrid Model using Item kNN (k=3, correlation_mode=’cosine’) & Biased Matrix Factorization (optimal model from previous tuning)	1.177	0.881	0.220

*Note: “*additionally user-filtered”: adding an additional filter (on top of the original preprocessing filter) that sifts out the records of users with less than 2 ratings records.*