Recommender Systems for Amazon User-Product Recommendations

Luisa Silva, Nafessa Jaigirdar, Aaron Finkelstein
TA: Crescent Xiong
University of Pennsylvania

Abstract
In this project, our group investigates the problem of recommending electronic product purchases to different consumers on the e-commerce website Amazon using machine learning algorithms. In order to minimize search times and maximize efficiency for consumers online, companies are increasingly leveraging their website data to provide useful recommendations to online users. The same algorithms used to recommend e-commerce products will dictate what content people consume on streaming platforms like YouTube, which matches they receive on dating apps such as Tinder, advertisements they see on Google, and more. As a means to solve this problem, we utilize a data-set compiled from empirical reviews of electronic product purchases from Amazon found on Kaggle, a website for sharing and testing data. We employ SVD and kNN algorithms to find that the SVD model without timestamp is the most accurate model.

Contents

1 Introduction

Our project's data-set consists of reviews for electronics products by users on Amazon. Although the table contains 7,824,482 rows of data, only 4 columns are present: userID, productID, rating and timestamp. Rating is a float value ranging from 1.0 to 5.0, incrementing by 1. The primary goal of our project is to evaluate these inputs to create a recommender system, primarily one that relates to Amazon's own system, collaborative filtering. Given the sparsity of the data-set, this will involve a key filtering of data phase, in which certain users and timestamps may be removed from the data-set for various reasons. EDA displays a few key features of the data-set that may prove fruitful in the filtering process: timestamp is essentially normally-distributed, ratings are left-skewed, over 50% of the reviews are ratings of 5.0, and there are multiple products with less than 10 reviews. All of these statistical discrepancies will be taken into account in the creation of our models.

We use 2 different machine learning methods to create the recommendation system: K-nearest neighbors (KNN) and Singular-Value Decomposition (SVD). Each of these models employ collaborative filtering to ensure the greatest amount of predictive power for the purpose of our project. Both take in ratings as an input and output predictions of ratings and preferences of users on products. We use different types of kNN models, including basic kNN, kNN with Z-Score, kNN with Means User-User, and kNN with Means Item-Item. to find the one most suited for our analysis. Timestamp is additionally utilized as a new input contrarily to previous analyses using this data-set. The metrics used to evaluate both will be standard accuracy measures for collaborative filtering: Root Mean Squared Error, Mean Absolute Error, fit time and test time.

For the collaborative filtering implementations, the hyper-parameters will be reevaluated using Randomized Search to optimize our own model. The recommender system is built upon by factoring in timestamp into the evaluation of popularity ranking. For all models, pre-processing will be especially important given the lack of features to use in the data. Timestamp will be factored in as a feature in contrast to the baseline model, which removed it entirely. Additionally, all products with less than 15 ratings and ratings by users with less than 50 reviews are removed. We test on a 70/30 split of the input data for training and testing respectively.

For the evaluation contributions, given the fact that our baseline model acts a comprehensive recommender system with low error, we attempt to build on the model by varying hyper-parameter search and including timestamp in our own. Due to this, our main question becomes whether or not the inclusion of timestamp is necessary in recommending electronic user products, considering the baseline model's accuracy without it. Additionally, we test if our different subsets of the data through logical filtering will result in increased performance, and whether or not Randomized Search is the best method to choose hyper-parameters for this application.

1.1 Proposal Feedback

Given the feedback from our prior project milestones, we have implemented new steps to account for our initial shortcomings.

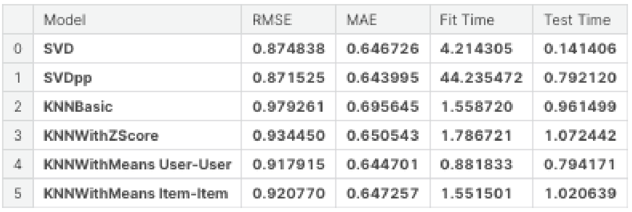
In order to acknowledge the importance of data cleaning to our model, we filtered out all data points of users that made less than 50 reviews, which makes the data-set distinctively more sparse while still allowing for enough data per user to create adequate recommendations. In addition, we filtered out any products with less than 10 ratings, as we want to avoid cases where the product's number of ratings is very low (such 1 or 2) but the rating value is unrealistically high. We justify the choice of our thresholds for SVD product and user ratings with the two graphs seen in the appendix, which both display that the RMSE values decrease with higher thresholds. Although the filters chosen are similar to our baseline model, they are optimally selected to improve accuracy– we chose threshold values that were high enough to lower the RMSE, but not too high as to filter out too much data.

Secondly, we incorporated ways to differentiate your project implementation with the those already in Kaggle. We use random search to tune our hyper-parameters, instead of grid search. This makes more sense in the context of our project, as the data is extremely low dimensional and gives the highest probability of finding the optimal hyper-parameters for a given model. Additionally, we add timestamp to our utility matrix as an extra parameter to differentiate from the baseline models used as a basis of comparison.

2 Background

Given the fact that the data-set is posted to Kaggle, a leading online data-provider that hosts competitions for data analysis, the bulk of the leading work comes from the codebase of user submissions attempting to most accurately model the problem at hand. Given that only collaborative filtering can be utilized to create a recommender system given such a sparse data-set, the majority of submissions all employ the same techniques, namely SVD and kNN, the same models we use in this report. The main problem with all of such reports include their use of Grid Search to find optimal hyper-parameters and not utilizing timestamp as an additional feature in the model.

The most relevant and comprehensive analysis relating to this problem happens to be from the Kaggle user who posted the data-set in the first place: https://www.kaggle.com/code/pritech/product-recommendation-systems/notebook. We mainly build upon his analysis, which similarly employs SVD, basic kNN, kNN with Z-Score, kNN with Means User-User, and kNN with Means Item-Item [Choudary, 2020]. The submission includes exploratory data analysis on the data-set before filtering. Similarly, users with 50 reviews or more are filtered for in this analysis, as are products with more than 10 reviews. Grid Search is used as the method of finding optimal hyper-parameters, as opposed to our Random Search. The main difference is his lack of timestamp as a feature in the data, which we investigate to see if it has any beneficial effect on the recommender system.



	Model	RMSE	MAE	Fit Time	Test Time
0	SVD	0.874838	0.646726	4.214305	0.141406
1	SVDpp	0.871525	0.643995	44.235472	0.792120
2	KNNBasic	0.979261	0.695645	1.558720	0.961499
3	KNNWithZScore	0.934450	0.650543	1.786721	1.072442
4	KNNWithMeans User-User	0.917915	0.644701	0.881833	0.794171
5	KNNWithMeans Item-Item	0.920770	0.647257	1.551501	1.020639

Given his highest RMSE is a measly 0.920770 for kNN with Means Item-Item, it will be difficult for use to match his strong results with new implementations, but that does not hamper our motivation.

3 Contributions

3.1 Summary

For our implementation contributions, we compare multiple collaborative filtering algorithms as contenders for the most predictive recommender system, including SVD, basic kNN, kNN with Z-Score, kNN with Means User-User, and kNN with Means Item-Item. The hyper-parameters are explicitly varied with Random Search and timestamp is used as an additional features.

For our evaluation contributions, we mainly aim to answer whether or not timestamp as an additional feature in our recommender system improves its predictive power. Additionally, we investigate why Random Search is optimal for choosing hyper-parameters in the context of collaborative filtering. Last, we display the reasons behind maintaining shifts in our data pertaining to users with a certain number of given reviews and products with reviews.

3.2 Implementation Contributions

SVD:
Using the cleaned and filtered data, we implement a collaborative filtering system using the SVD model from the *surprise* package. The first step in the implementation was running Random Search cross-validation to find optimal parameters for our final model. Using Grid Search, we tried to optimize 2 main metrics Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). Having the optimal parameter values for *n_epochs*, lr_all and *reg_all*, we trained the final model and tested it on our held-out data. This tuned model was used for testing the data in one iteration including timestamp as a feature, and another without.

kNN:
As mentioned above, we needed to first transition our data such that the surprise library

could interpret and extract insights from it. With our updated data, we first ran Grid Search to determine the optimal parameters under three versions of the kNN algorithm: the basic approach, the means approach and the z-score approach. Per each of its names, the basic approach is the standard collaborative filtering algorithm, the with means takes into account the mean ratings of each product or user respectively depending on whether we opt for item on item or user on user filtering and the z-score approach which accounts for the z-score normalization of each product or user again depending on whether item on item or user on user filtering is used. In this case, we will be testing both to see if performance is better when providing recommendations based on items that are rated similarly versus users who tended to rate the same items in the same general way. In running grid search, we aimed to optimize for RMSE and MAE by adjusting the *bsl_options*, which controls how the baseline are computed, reg, which is the regularization parameter of the cost function, k which is the number of neighbors to base the recommendation upon and *sim_options*, which is the metric by which similarity is calculated, through which we tested cosine, msd and pearson similarity. This was in contrast to our baseline model, which only employed cosine similarity.

In terms of how we updated the model, we first relied on random search as opposed to grid search. Random search is often considered better than grid search because it can explore a larger search space in a more efficient way. In a grid search, a fixed set of hyper-parameters and their values, and then perform an exhaustive search over all possible combinations of these values. This approach can quickly become computationally expensive and impractical when dealing with a large number of hyper-parameters.

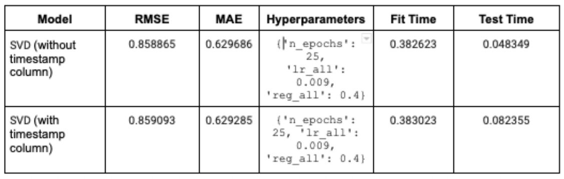
On the other hand, random search randomly selects hyper-parameters and their values from a predefined range. This approach allows for a more comprehensive exploration of the hyper-parameter space, and often yields better results than grid search with fewer iterations. Additionally, random search can be more efficient because it can prioritize sampling the most important hyper-parameters, based on previous results.

We also made a cognizant effort to include timestamp as well as feature engineering which significantly improved our RMSE. Specifically, we first included code that preprocesses the Timestamp column by converting it to a datetime object and extracting the year, month, day, hour, minute, and second features. Secondly, we did the following: We perform feature engineering by creating new columns in the data-set based on the mean and count of ratings given by each user and received by each product.

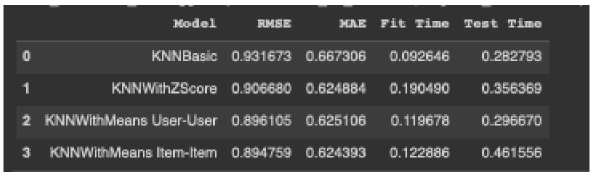
user_rating_mean: This feature is the mean rating given by a user to all the products they have rated. It helps to capture the general rating behavior of each user.
product_rating_mean: This feature is the mean rating received by a product from all the users who have rated it. It helps to capture the general popularity of each product.
user_rating_count: This feature is the count of ratings given by a user to all the products they have rated. It helps to capture the level of user engagement in rating products.
product_rating_count: This feature is the count of ratings received by a product from all the users who have rated it. It helps to capture the level of popularity of each product. By including these features in conjunction with randomized search, these new features provide additional information that was not present in the original data and can help the machine learning model to make more accurate predictions.

3.3 Evaluation Contributions

With this project, we set out to answer four key components to our larger problem: “how can we use feature engineering to increase the performance of our recommender system?”, “how can we tune the hyperparameters of our model more effectively to increase our RMSE and MSE metrics?" "how can we innovate and add to the current Kaggle submission implementations for the recommender system problem?", and "how do we include timestamp as a suitable feature in our model?". The baseline of our approach will be comparing the results of different implementations of the recommender system: The performance metrics we decided to focus on are Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). Below are the preliminary results for our SVD and kNN model: SVM, basic kNN, kNN with Z-Score, kNN with Means User-User, and kNN with Means Item-Item. The results of our different models can be found below:



Model	RMSE	MAE	Hyperparameters	Fit Time	Test Time
SVD (without timestamp column)	0.858865	0.629686	{'n_epochs': 25, 'lr_all': 0.009, 'reg_all': 0.4}	0.382623	0.048349
SVD (with timestamp column)	0.859093	0.629285	{'n_epochs': 25, 'lr_all': 0.009, 'reg_all': 0.4}	0.383023	0.082355

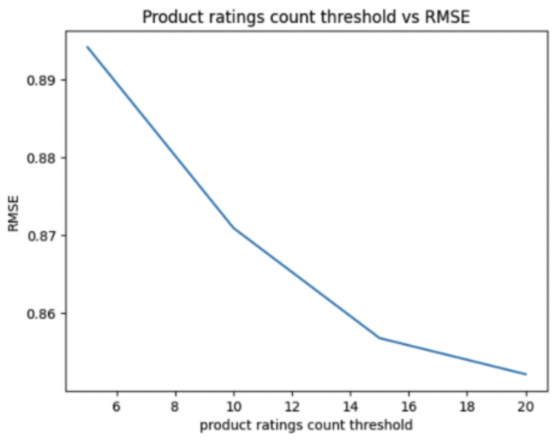


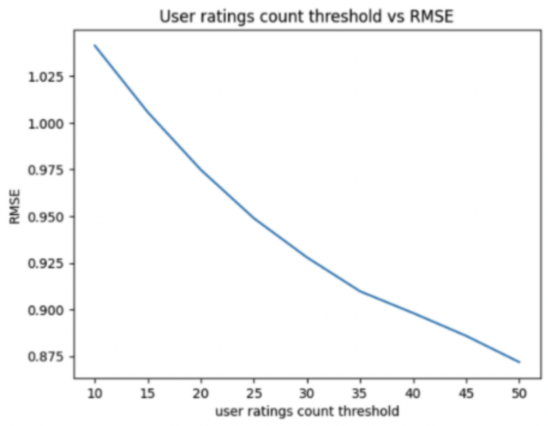
	Model	RMSE	MAE	Fit Time	Test Time
0	KNNBasic	0.931673	0.667306	0.092646	0.282793
1	KNNWithZScore	0.906680	0.624884	0.190490	0.356369
2	KNNWithMeans User-User	0.896105	0.625106	0.119678	0.296670
3	KNNWithMeans Item-Item	0.894759	0.624393	0.122886	0.461556

Most notably after looking at the data-set, the SVD that does not include timestamp results in the lowest value for RMSE, while kNN with Means Item-Item provides the lowest value in regards to MAE. Given the SVD's vast improvement in fit and test time, it is the obvious candidate for an actionable collaborative filtering model. When considering kNN, it is likely that the item to item formulation outperformed the others because users are solely compared based upon their liked items anyways, so there is no advantage to relaying this information back to compare users. There is likely a higher variance among users' preferences, and items are solely compared on the basis of their rating, which by definition, cannot be variant at all.

This proves that although timestamp can be adequately transformed to be included in the model, it is not necessarily necessary to build a proper recommender system. It is interesting to consider exactly why an extra feature lowers the accuracy, even by the minuscule amount that we observed with our results, when it is quite easy to speculate that timestamp's addition may have caused overfitting instead. The worsening effect is likely due to how the system evaluated similarity with timestamp combined with rating. Simply put, for two similar users A and B, a collaborative filtering model recommends the liked products of B to A. Thus, it is possible that many of the users that made reviews at similar times in fact had different tastes in products entirely.

When considering data-set shifts, we decided to implement two types of filtering to our original data. First, we filtered out all data points of users that made less than 50 reviews, as well as any products with less than 10 ratings. These decisions were validated by computing constant versions of each model using different subsets of the data-set. The graphs of each can be found below:





As you can see in both cases, the RMSE is reduced linearly as the number of user ratings and the number of product ratings increase. Thus, 10 product ratings and 50 user ratings were chosen to eliminate products with unjustified review scores while not reducing the data by a large margin.

Despite all of these considerations, our model has the same flaw of any collaborative filtering recommender system: its interpretability. It is difficult to consider a recommendation's effectiveness without understanding the content and motivation behind it, something that a collaborative filtering system does not take into account. These are considerations that our evaluation metrics could not assess, and depend more on human contemplation of which model is best for the task at hand. All in all, our model performs well based on our own defined metrics, but its actionable predictive ability is undeniably dubious.

4 Resources Used

The sole resource used in the creation of this project was the main Kaggle submission described above. In addition, the lectures on recommender systems in class were key resources for the creation of our models and their evaluation. Additionally, Google Colab hosts the code where each of our models were built.

5 Conclusion

Overall, the main outcome from our project is that more attributes included does not necessarily translate into a stronger model. This is mainly seen with our implementation of the timestamp feature; despite its inclusion, the SVD model with the same hyper-parameters results in a lower overall MSE. However, the significant feature engineering displays how a similar model may have used this feature in the past. In addition, it displays how Random Search is an effective choice for choosing the best parameters.

In hindsight, we ended up relying heavily on the previous work due to its accuracy and the relative sparseness of our dataset. Although we ended up proving on our own why certain parameters were chosen, methods were used, and filters were applied, it would have been interesting to truly attack this problem from scratch with multiple features to experiment with. However, implementing timestamp and choosing our own hyper-parameters still resulted in a fulfilling analysis that allowed us to practically use concepts taught in class.

Looking beyond the horizon for collaborative filtering, we envision a world of real-time machine learning, where different suggestions for improvement can be made for humans in any activity. A foreseeable part of this issue could be solved with models similar to what we employed in this project.

All in all, this project effectively allowed our group to interface directly with the source material and the Penn community. It was amazing working together and meeting our TA, amounting to a great opportunity in the end!

Cited Works: Pritam Choudary, "Product Recommendation Systems", Kaggle 2020