Recommending Books using Goodreads Data

Adam Coviensky, Stephanie Doctor, Marika Lohmus, Mark Salama

The Dataset

We have chosen to build a recommender system for books using the goodbooks-10k dataset (https://github.com/zygmuntz/goodbooks-10k) by Zygmunt Zając. The dataset contains approximately 6 million ratings for ten thousand most popular books from goodreads (based on the total number of ratings for each book). The repository also contains metadata on each book (title, author, average rating, etc.) as well as tags for each book given by users (e.g. “fantasy”, “sci-fi”, “for-fun”, and “all-time-favorites”).

The ratings data consist of 5,976,479 ratings (between 1 and 5) for 10,000 books among 53,424 users. This gives the data a sparsity of 1.12%.

Users rated between 19-200 books each (median 111). Each book was rated between 8-22806 times (median 248). The ratings distributions are shown in Figure 1.

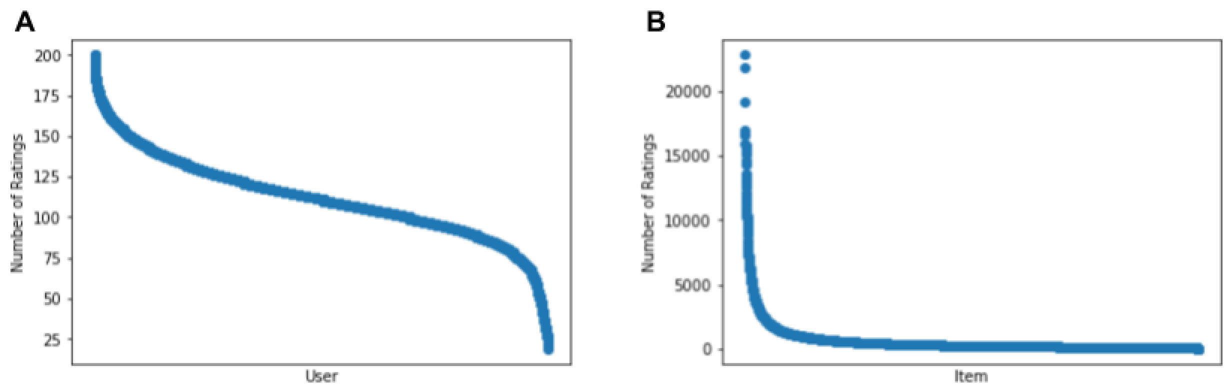

Figure 1. Number of ratings by each user (A) and for each item (B).

Even though this dataset has selected the 10,000 most popular books, we can still see the long tail problem with the ratings wherein a few books have received a lot of ratings and most have received a lower amount.

Looking at the distribution of all ratings within the dataset in Figure 2 shows that most ratings are either a 4 or a 5. This is another reminder that we are dealing with a dataset which only includes the most popular 10K books in Goodreads. It can be argued that a book becomes popular when it has high ratings, and will most likely continue to get higher than average ratings over time.

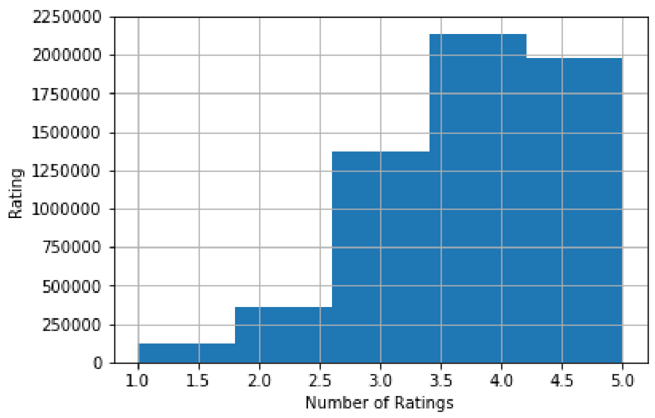

Figure 2. Distribution of all ratings within the dataset.

Looking at the average rating that a user gave books, and the average rating that a book received from users, we can see how the average is around 4 for both. This is expected for the most reviewed books.

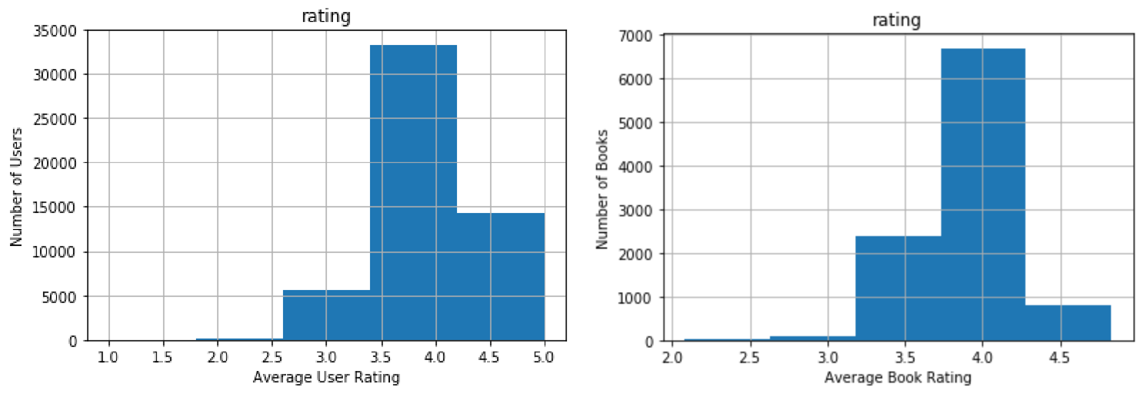

Figure 3. Distribution of average user ratings (left) and average book ratings (right).

Business Goal

Our company is called Goodreads, a book recommendation website owned by Amazon. The goal of our website is to provide our users a social platform where they could record ratings and reviews of the books they have read, interact with other book-loving users, and to recommend them new books that they would ultimately buy from our parent company.

In order to gain and keep new customers, we want to provide our users with a personalized recommender engine that suggests books that they actually want to read. The more they return and trust our recommendations, the more books they will eventually purchase through Amazon. A new customer is more likely to return to Goodreads if their list of recommendations contains books that they may recognize, rather than containing ubiquitous books they have not heard of. Therefore, we will prioritize maximizing the **accuracy** of our models.

We are thus willing to sacrifice coverage and serendipity in order to achieve higher accuracy in our recommendations. Being serendipitous, or suggesting books that the user would be surprised to like, could lead to lower accuracy. The user would be recommended some books which they do not enjoy, which may then lead them to lose faith in the Goodreads recommender system. By focusing on accuracy, we may also sacrifice coverage since we would be recommending many of the same more popular books to our users.

Evaluation Methods

Accuracy Metrics

In order to evaluate accuracy, we focused on three metrics - the root mean squared error (RMSE), the median absolute error (MAE) and the Spearman Rank-Order Correlation Coefficient. These metrics determine whether our algorithms are able to accurately predict whether a user likes a certain book, and whether the magnitude and order of these predicted ratings are similar to the user's preferences.

RMSE

Root Mean Squared Error (RMSE) is an accuracy measure defined by

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n} (y_t - x_t)^2}{n}}$$

where n is the number of ratings, x_t is the true rating for instance t, and y_t is the predicted rating for the same instance. Because errors between predicted and true ratings are squared, outliers (predictions that are very far off) get penalized quite heavily in the RMSE.

MAE

Mean Absolute Error (MAE) is defined by

$$MAE = \frac{\sum_{t=1}^{n} |y_t - x_t|}{n}$$

where n is the number of ratings, x_t is the true rating for instance t, and y_t is the predicted rating for the same instance. The MAE does not penalize errors as heavily as the RMSE since it uses absolute value rather than square but both consider errors in either direction equally.

Spearman Rank-Order Correlation

The Spearman Rank-Order Correlation is a Pearson's correlation between the rank order of the true ratings and that of the predicted ratings. It is defined by the following equation:

$$r_s = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X}\sigma_{rg_Y}}$$

Here, rgX would be the rank order of the true ratings, and rgY would be that of the predicted ratings. The Spearman Rank-Order Correlation is useful because it takes into account only the order in which the items are recommended and not their predicted ratings, thus removing the effect of nonlinearities in the data.

Scalability

We decided to test our algorithms on the most dense dataset we could get, and then slowly scale up. The logic behind this was to determine whether our algorithms gave good results for the most dense subset, and if not, to look into any issues that our data might have.

We created six subsets of the full dataset, described in Table 1. We selected these datasets by first selecting the X most reviewed number of books, and then from the subset of users, we selected the Y most prolific users (those who had reviewed the most). The number of users in Table 1 below is not a nice round number since not all users had reviewed the selected popular books.

For each of these, we then performed an 80-20 train-test split. After tuning hyperparameters, we used them to train each model on each of the six train subsets, and recorded the time required to train the model and the time required to predict on the test set. We then calculated RMSE, MAE, and the Spearman Rank-Order Correlation Coefficient for each model to determine how the accuracy and runtimes scaled with increasing sparsity.

Table 1. Data subsets used in scalability tests.

Number of Users	Number of Books	Number of Ratings	Sparsity
487	20	4311	44.26%
985	35	13932	40.41%
1981	50	37100	37.46%
4980	70	112959	32.40%
7479	85	187770	29.54%
9980	100	271422	27.20%

Coverage

We also looked at the catalog-coverage of our recommendations. This metric measures the fraction of all books that are in the top-k for at least one user. This measure can be a better indicator of both the diversity and serendipity of recommendations, since we can tell whether the same books are being recommended to all users.

If T_u is the set of top-k books recommended to user *u*, *n* is the total number of books, and *m* is the total number of users, then catalog-coverage can be calculated as follows:

$$CC = \frac{\left|\bigcup_{u=1}^{m} T_u\right|}{n}$$

Training and Cross-Validation Methodology

For each model, we used 3-fold cross-validation with grid-search on the largest subset (9980 users and 100 books) to choose optimal hyperparameters. We calculated the RMSE for each run, and selected the hyperparameters and models that performed the best to use going forward.

Recommender Engine Approaches

Baseline Selection

For detailed results and graphs regarding Baseline Model selection, see baseline-algorithm.ipynb.

Normal Prediction

This algorithm predicts a random rating based on the assumed normal distribution of the training set. The mean and variance of the used normal distribution are calculated from the training data using Maximum Likelihood Estimation.

Baseline Bias Estimation

Typically, both items and users have systematic biases, meaning that some users tend to give higher or lower ratings than others, and some items receive higher ratings than others. If μ is the overall average rating within the dataset, then a baseline estimate for an unknown rating is calculated by

$$\widehat{r}_{ui} = \mu + b_i + b_u$$

where b_i is the observed deviations of item i from the average and b_u is the observed deviation of user u. In order to estimate bu and bi for each user and item, one can solve the following least squares problem:

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right)$$

The first part of the equation tries to minimize the error between the estimated and actual error, while the second regularizing term aims to avoid overfitting by penalizing the magnitudes of the parameters. This estimation can be also done faster via alternating least squares or standard gradient descent by alternatively calculating b_i and b_u. Since these alternating methods sacrifice accuracy, we used grid search to reduce the errors in both.

Tuning & Evaluation

Since the Normal Prediction method uses the given testing data to calculate the mean and variance of the data using Maximum Likelihood Estimation, there were no parameters to tune. The RMSE and MAE were obtained by training and testing on the 10K user / 100 item subset using 3-fold cross validation, and they were 1.3562 and 1.0693, respectively. The MAE and RMSE values are quite high, which can be expected from an algorithm that predicts ratings by pulling random values from a normal distribution scaled on the whole dataset, and which doesn't take into account each user's or item's unique properties.

The baseline bias estimates were evaluated using standard gradient descent and alternating least squares methods. In order to tune the hyperparameters, we used grid search to test out values for each.

Baseline Selection

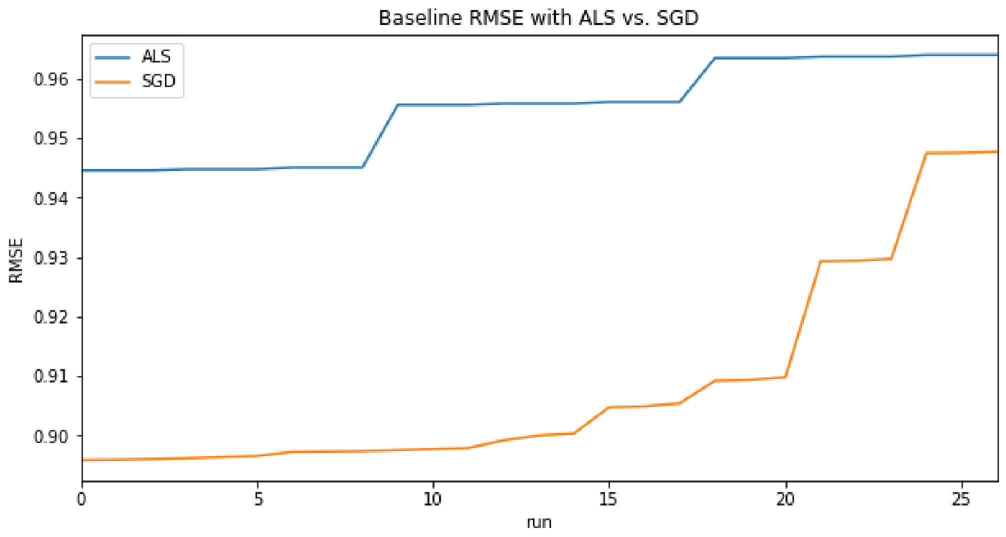

Figure 4. Accuracy of two baseline algorithms.

The best runs of the baseline estimation tuned by standard gradient descent performed better than all alternating least squares results quite significantly (RMSE difference of about .05 for the best runs). Therefore, we used the SGD baseline estimator method to evaluate scalability and to serve as a baseline for comparing the rest of the approaches.

The hyperparameters chosen were a learning rate of .005, 20 epochs, and a regularization parameter of .01.

Neighborhood-Based Approaches

For detailed results and graphs regarding Neighborhood-based model selection, see neighborhood_models.ipynb.

K-Nearest Neighbors

In the neighborhood approaches, we used grid-search to determine the number of nearest neighbors, k, the similarity metric, the minimum support, and whether to use a user or item based model. We tested 10, 50 and 200 neighbors.

We used KNN to make predictions using the neighborhood approaches. We attempted this with both a user-based and an item-based model on the subset of data containing 10000 users and 100 items. Furthermore, we tested KNN scaling by the means and scaling by the Z score of the ratings for the users (in the user-based approach).

We used grid-search to tune the algorithm. We searched over the following parameters and values for each of the algorithms for both user based and item based: k = 10, 50, 200; minimum support = 1, 3; similarity metric = msd, pearson, pearson with baseline.
The full grid-search results can be seen in the notebook along with formulas for how predictions were made. The final best item-based algorithm was KNN scaling by the means with K=10, pearson similarity, and minimum support of 1.

For the user-based model, we found the best algorithm was KNN scaling by the means with K=50, pearson baseline similarity, and minimum support of 1.

We expected the pearson similarity metrics to outperform as they subtract the mean of the user or item’s ratings before calculating similarity. This helps to account for users who tend to rate things higher or items which tend to be rated higher. The pearson baseline similarity metric subtracts the baseline estimate of each user or item from the ratings. The baseline estimates are calculated as described above in Baseline Bias Estimation.

Neighborhood-Based Approach Selection

When looking at the neighborhood approach, we can see an exponential relationship between the runtime and the number of ratings for the user based approach. This relationship also exists for the item-based approach, however since there are so much fewer items in this sample of data, it cannot be observed on the graph.

Since the number of books were restricted to be 100, the runtimes are still very reasonable. However, as the number of books greatly increases if we were to run the algorithm on our whole dataset, it would yield unreasonable results and not be worth putting into production.

In the item-based neighborhood approach, we see an increase in RMSE and MAE until the second largest sampled dataset we used. We actually improved our accuracy at the largest and most sparse dataset. It is also seen that (not including the smallest dataset), our spearman coefficient seems to go up when using more data indicating that as we use more data, we are likely to better predict user preferences.

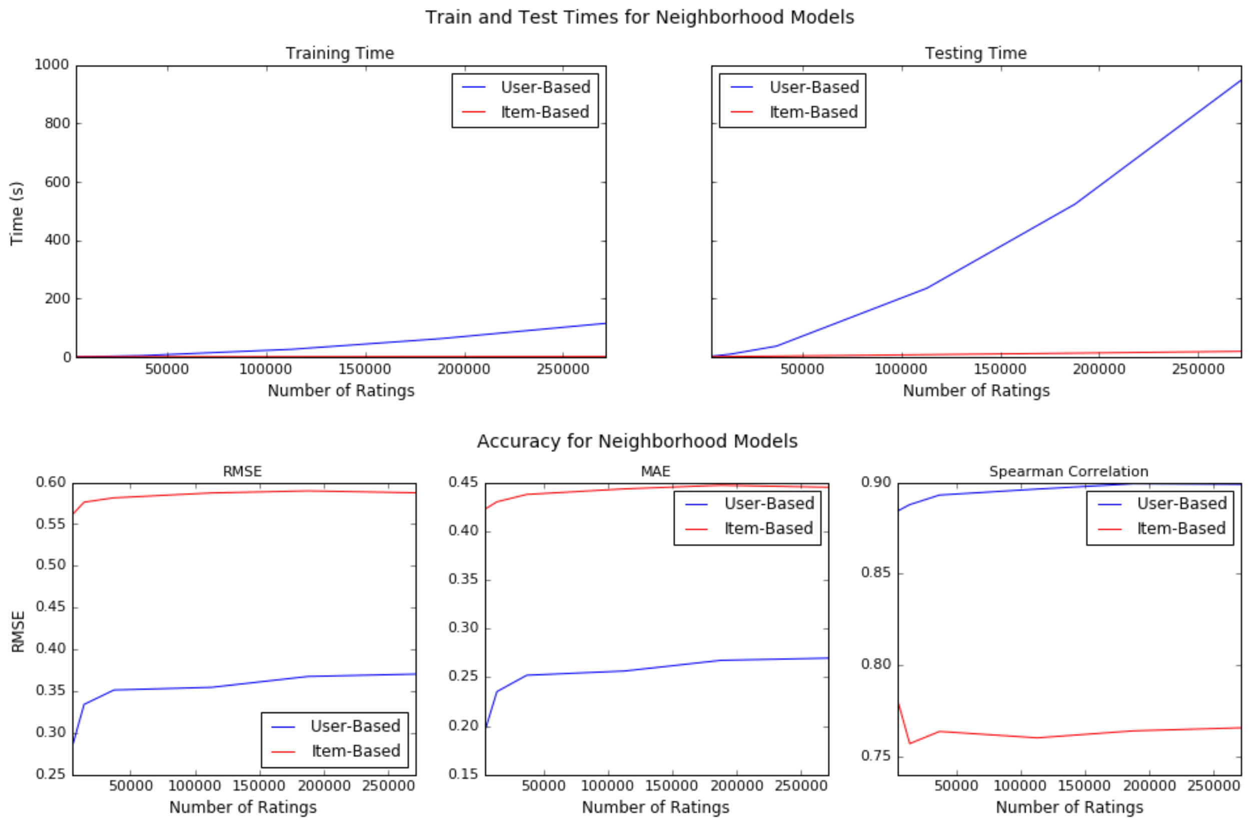

Figure 5. Summary of Neighborhood models, user-based KNN and item-based KNN: on top, running time for training (left) and testing (right); on bottom, accuracy in RMSE (left), MAE (center), and Spearman correlation (right).

Overall, the trends observed in the user-based model are similar to the trends observed in the item-based model. Since we have so many more users however, we can see that the training and testing times are orders of magnitudes larger. We also observe that the Spearman Correlation Coefficient continues to increase with more data until the largest sample size.

Comparing the two models and how they scale, we observe that the training and prediction times for the user-based model are orders of magnitude larger than the item-based model. This is somewhat deceiving however as the number of users being compared is way larger than the number of books. There are 1% of the number of books as users. Furthermore, in our actual dataset for part 2 of the project, there’s 20% the number of books as users. Thus, the time discrepancy won’t be nearly as large.

The catalogue coverage for both models was very impressive. The user-based model obtained catalogue coverages of 94%, 97%, and 100% for top-k recommendations for k = 5, 10 and 20 respectively. The item-based model had catalogue coverages of 95%, 97% and 100% respectively. The numbers are likely so high because we chose the 100 most popular items in our dataset. As we increase the size of our dataset by taking more users and items, we expect this number to drop.

Since our main objective for this project was to maximize our accuracy, specifically our RMSE, we choose the user-based neighbourhood approach over the item-based one. The RMSE for the user-based model is approximately 60% of that of the item-based model when using 10k users and 100 items. We fully expect the RMSE of both to increase significantly as we increase the number of users and items since we chose the densest subsets of our data. Thus, we will be increasing sparsity, likely leading to a decrease in the accuracies.

Modeling-Based Approaches

Matrix Factorization

For detailed results and graphs regarding Matrix Factorization, see matrix-fact-manual-regularized.ipynb.

Matrix Factorization is a method in which two matrices U and V are learned whose dot product resembles the original user-item matrix. Starting with a sparse matrix in which many ratings are missing, we learn U and V to minimize the error between the ratings we do have and their predicted values (from U*V). Then, the dot product can be used to predict the values of the ratings missing in the original dataset.

We implemented matrix factorization directly in Python using Gradient Descent. Specifically, matrices U and V were calculated to minimize the following objective function

$$J = \frac{1}{2} \sum_{(i,j)\in S} {e_{ij}}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} {u_{is}}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} {v_{js}}^2$$

where S is the set of observed ratings, e_{ij} is the error of the prediction for rating r_{ij}, λ is the regularization parameter, m is the number of users, n is the number of items, and k is the rank of the learned matrices U and V. In this way, the matrices are learned based on the error from only the observed ratings in the training data, but are regularized for all values.

To choose optimized parameters, we grid-searched over values rank d = [5, 10, 15], learning rate alpha = [0.00005, 0.0001, 0.00015, 0.0002], and regularization parameter lambda = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]. The optimal parameters were d=5, alpha=0.0001, and lambda=0.001, which gave an RMSE value of 0.958.

SVD with SurPRISE

SVD, or singular value decomposition, is a matrix factorization technique based on the eigenvalues of the original ratings matrix. In the context of recommender systems, it works by decomposing the user-items-ratings matrix such that we learn a set of features for every item and every user in the same dimensional space. We arrive at a recommendation for a user u and and an item i by calculating the dot product of the vector learned for user u and the item learned for item i. We also learn a user and item bias.

For detailed results and graphs regarding the SVD with SurPRISE model, see surprise_SVD_NMF.ipynb.

Ranges tested for the parameters are as follows:

- n_factors: 100 to 200
- n_epochs: 20 to 65
- learning rate: 0.0025 to 0.03
- reg_all: 0.0025 to 0.15

Based on the above, we used the following for evaluation: N_factors = 200, N_epochs = 50, Lr_all = 0.0225, Reg_all = 0.08.

NMF with SurPRISE

Non-negative Matrix Factorization, like SVD, is a matrix decomposition technique. In this case, the user and item matrices are initialized to random values, and then with each iteration of the algorithm, matrix values are adjusted to bring the dot product between a user vector and item vector closer to the actual rated value for that user and item pair. The objection function includes this difference (predicted vs. actual rating) as well as regularization terms to avoid overfitting. Unlike SVD, NMF requires that all learned factors are non-negative.

For detailed results and graphs regarding the NMF with SurPRISE model, see surprise_SVD_NMF.ipynb.

Parameters across the following ranges were tested:

- n_factors: 12 to 60
- n_epochs: 40 to 100
- learning rate: 0.004 to 0.012
- regularization: 0.05 to 0.095

This led to the following set of optimal parameters: n_factors: 60, n_epochs: 80, lr_bu: 0.0085 lr_bi: 0.0085, reg_pu: 0.085, reg_qi: 0.085. Instances with a learning rate of 0.01 and regularization of 0.07 performed similarly well.

Model-Based Approach Selection

In order to choose a model-based approach for the final comparison, we looked at the scalability of these three models by accuracy and running time. As mentioned in the top sections, we selected six different subsets to evaluate scalability on.

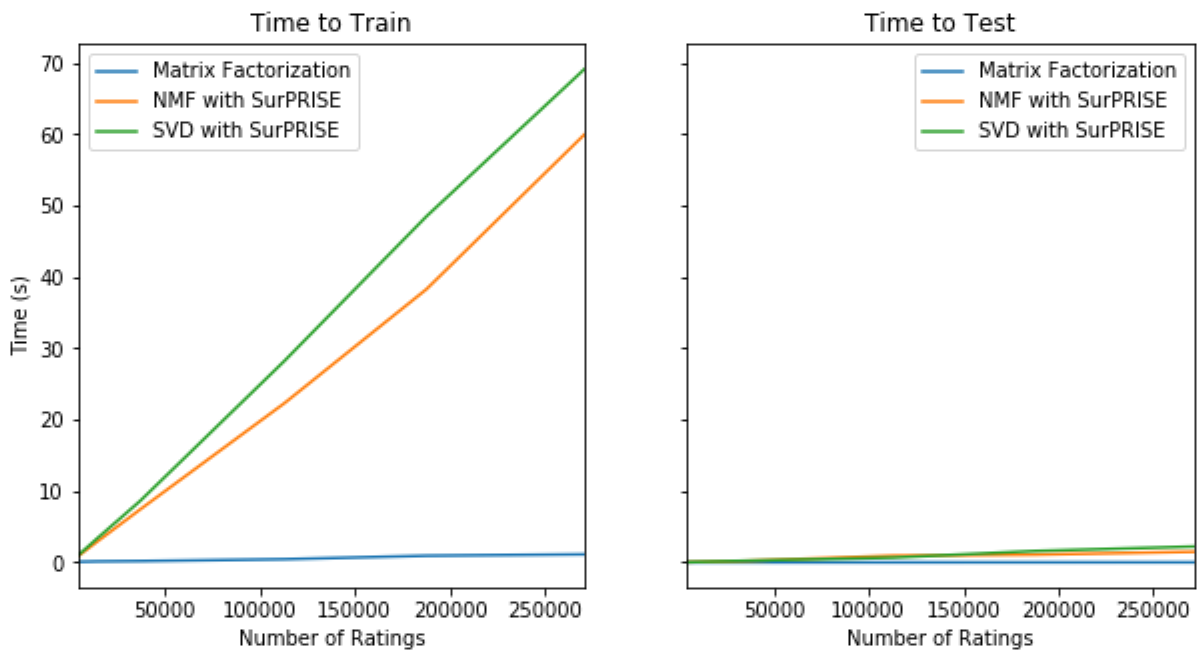

Figure 6. Train (left) and test (right) times for model-based approaches.

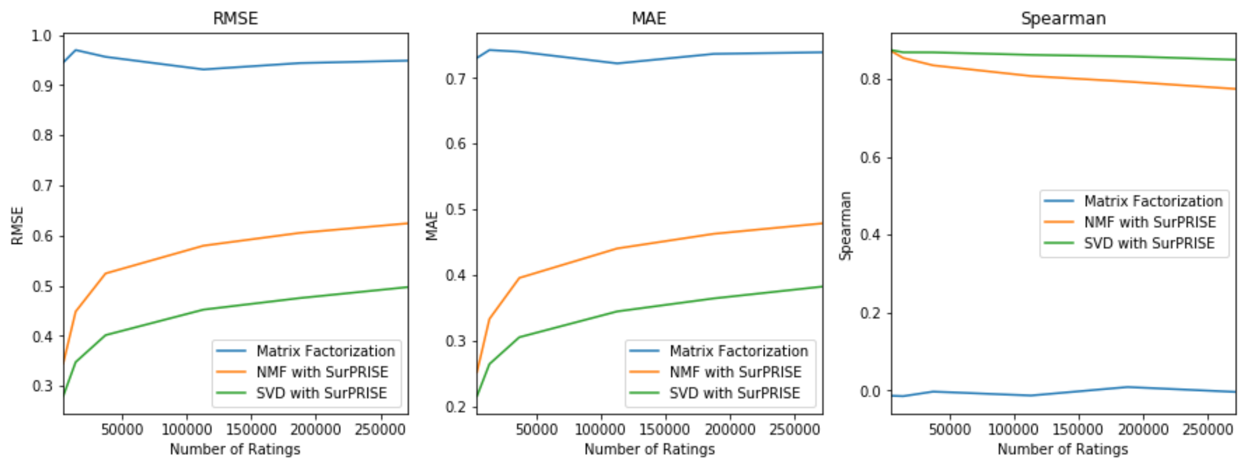

Figure 7. Accuracy measures RMSE (left), MAE (center), and Spearman correlation coefficient (right) for model-based approaches.

The model-based approaches present a clear trade-off in accuracy versus efficiency. Matrix Factorization is by far the fastest method in terms of training and testing time, but has clear deficiencies in accuracy compared to both SurPRISE methods. Of the two SurPRISE models, SVD had the lowest error rates but was slightly slower to train than NMF. Because we have decided to prioritize accuracy for this project, we have chosen the SVD with SurPRISE method to use in our final comparison.

Final Approach Comparison

We will be comparing the scalability (accuracy and runtime) and coverage of the user-based KNN as our selected neighborhood-based approach and the SVD model as our selected model-based approach.

Scalability

We looked at the scalability of both approaches by comparing both running time and accuracy as the number of ratings increased. The results are also compared to the baseline model (bias estimate tuned by standard gradient descent) selected above. As mentioned in the top sections, we selected six increasing in size subsets of the data to evaluate scalability on.

Running time

The training and testing runtimes for both the SVD and User-based KNN increased with the number of ratings. However, the key differentiator is the testing time - or the time it takes to estimate unknown ratings: the SVD model outperforms user-based KNN by a very wide margin, and seems to be more scaleable overall.

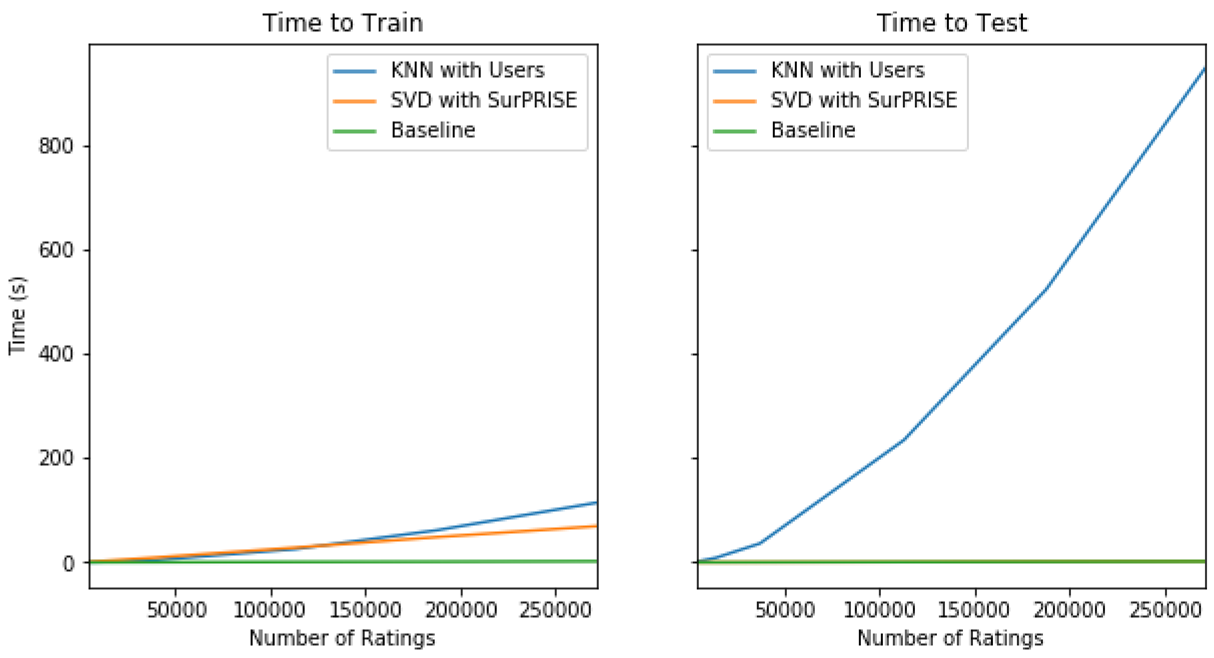

Figure 8. Train (left) and test (right) times for final approaches.

Accuracy

We looked at how the accuracy of our selected approaches would scale with an increasing number of ratings. This is a test of how accurate our system would be when the number of books and users in our system grows, and therefore sparsity grows.

We evaluated the scalability of accuracy with three metrics - RMSE, MAE, and the Spearman Ranking Correlation Coefficient.

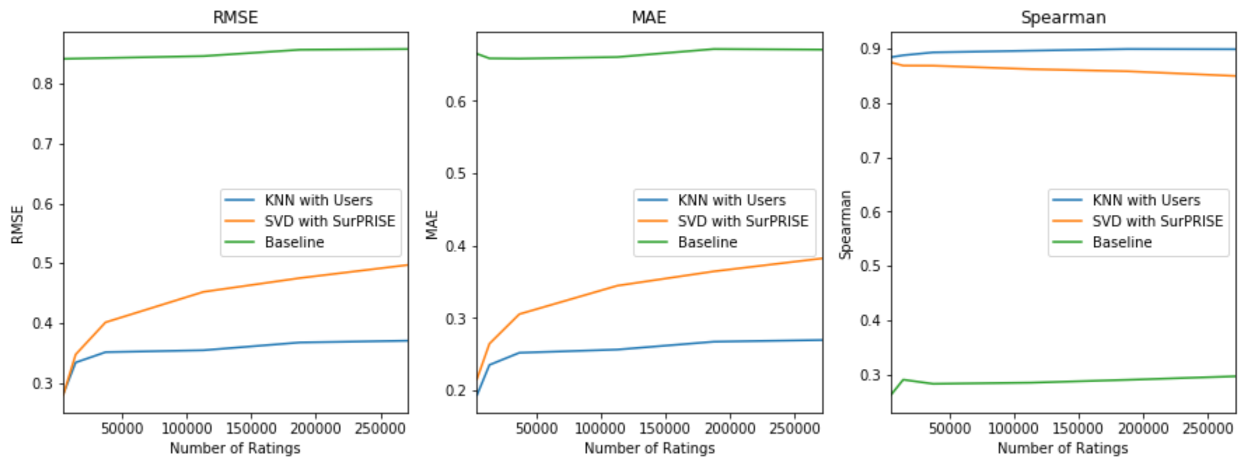

Figure 9. Accuracy measures RMSE (left), MAE (center), and Spearman correlation coefficient (right) for final approaches.

The user-based KNN approach performs better than SVD in all evaluation metrics. Its RMSE and MAE are lower, and its Spearman Rank-Order Correlation Coefficient is higher at all dataset sizes.

Scalability Summary

While user-based KNN does not scale well in terms of training and testing runtime, it strongly outperforms SVD in terms of our accuracy measures. Though we can assume that the accuracy of both methods will decline once we run these models on our full dataset, user-based KNN will most likely continue to perform better than SVD in terms of accuracy.

The main concern with using user-based KNN is the time that it takes to train and to evaluate data (an approximate time complexity of n^2 , where n is the number of users). This makes sense as the similarity metrics need to be calculated for each user-user pair, and the scores need to be calculated for each user based on their k closest neighbors. However, there are certain ways that we can try to mitigate the runtime issue. Some of them include:

- Indexing or hashing the users using methods like a K-D tree or locality sensitive hashing
- Using different search algorithms such as a projected radial search or greedy walk search

- Leveraging partitioning and only calculating item distances within the same partition (could be done by using Apache Spark on Hadoop)

There are also opportunities to try to combine the two algorithms using a hybrid SVD-KNN approach which would attempt to leverage the benefits of each (see this 2009 paper by Siegel et. al. on Hydra, a system which attempts to do just that on MovieLens data with promising results).

Coverage

We also took a look at the catalog-coverage of the resulting recommendations from both of our approaches. This metric answers the following question:

What percentage of all books are suggested in the top-k recommendations for at least one user?

If an approach suggests the same popular books to all users, then we can expect this value to be low. While this is not an indicator of accuracy, it gives us a better idea of the diversity and potentially the serendipity of our recommendation approaches.

Coverage Summary

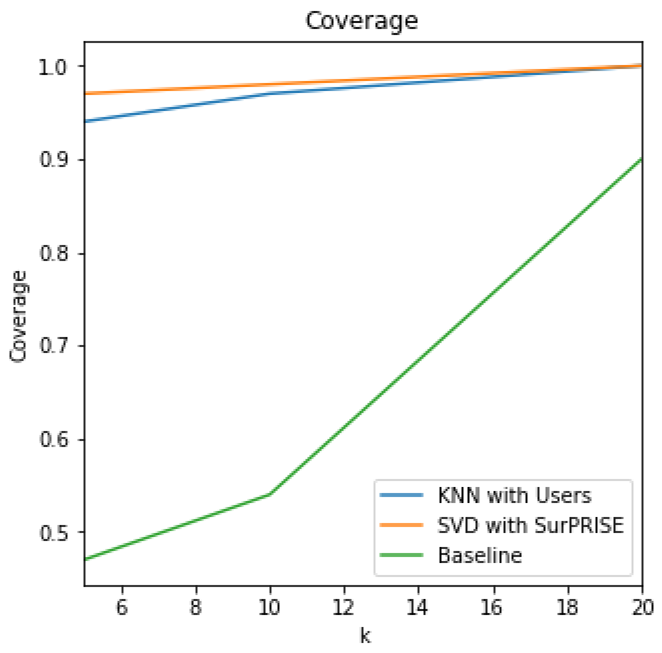

Figure 10. Catalog coverage for final approaches.

Both SVD and user-based KNN have a good catalog-coverage of our itemset, having values above 0.9 even if we only look at the top 5 recommended items. This is a large improvement over our baseline model, which seems to rank the same set of items to many of the users. This high catalog-coverage by our two approaches gives us additional confidence that the recommendations are tailored by some features of the user’s likes and dislikes, rather than just recommending the most popular unread books.

It does look like SVD has slightly higher catalog-coverage at lower values of k, but both of them converge to 1.0 or near-1.0 at top-20 recommendations.

Conclusion

If our overall goal is only to maximize accuracy, then the neighborhood-based approach of a user-based KNN performs best out of all of the approaches we tested. It surpasses our best model-based approach, SVD, by all three accuracy metrics of RMSE, MAE, and the Spearman Rank-Order Correlation Coefficient. We are quite confident that on this 10,000 user, 100 item subset, both approaches would recommend our users relevant items.

However, our user-based KNN does not scale well with increasing number of users, ratings, and overall sparsity. The evaluation portion of finding the k-nearest neighbors and calculating the estimated ratings is extremely time-consuming and would not be a feasible solution to put into production. No user would return to our website if we told them to come back in 20+ minutes for their actual book recommendations.

If we had to choose one system to put into production tomorrow, we would select the SVD approach. It performs faster with reasonable accuracy on our smaller subsets, and has great catalog-coverage so we know that it recommends diverse subsets (out of a 100 books) to each user. We also hypothesize that user-based KNN will not perform as well on the full dataset which is considerably sparser than what we used for this report.

To be able to recommend an approach to put into production, we need to do additional evaluation on the full dataset to be confident about our approaches’ true accuracy and performance on a large, sparse, dataset. We would also need to address our concerns with runtime scalability, perhaps employing approximation methods and using hybridization to improve accuracy.

Caveats / Watch Items

When looking at the results of this data, we need to keep in mind that the overall dataset is a subset of all Goodreads data, selecting the 10K most popular items and the most prolific reviewers. This means that our dataset is artificially dense, and in the real world we may have a lot more users and items that have much fewer reviews.

This issue is compounded by the way that we subset our data. We decided to choose the most dense subset of the data. As a first step, we wanted to first assure that our most prolific users will get useful recommendations. This leads to some skewed accuracy results. It is likely that our accuracies will start dropping as we add more users and many more items creating much sparser datasets.Some algorithms perform better on sparser data than dense data. It may no longer be true that user-based KNN is the best performing model in terms of accuracy. SVD in general performs well on sparse ratings matrices. Given these results, we might not have selected it as a final model due to how well KNN performed.

Next Steps

The next step for this project is to expand our model to the full ratings dataset. Because this will require many more resources, we plan to use PySpark to see how distributed computing can increase efficiency.

Given that user-based KNN consistently performed the best but had scalability issues, we would like to see if we can mitigate this issue and have outlined potential approximation approaches above (under Scalability Summary). SVD also performed well and it could be that in larger datasets, it would be superior in performance as well as time. We plan on testing both SVD and user-based KNN on the full dataset, as well as explore a hybrid approach. After transforming our explicit ratings matrix into an implicit one, we can also explore if incorporating implicit data through SVD ++ improves our results.

In addition, using the GoodReads API, we would like to see if additional content based data (e.g., book genre and book summary data similar to the Wikipedia MovieLens paper) improves recommendations.