Recommender System for Yelp Dataset

CS6220 Data Mining
Northeastern University

Clara De Paolis Kaluza

Fall 2016

1 Problem Statement and Motivation

The goal of this work is to construct a personalized recommender system for the Yelp dataset that can accurately predict a user's preference for a business. In particular, the system aims to predict the value of a user's rating (from one to five) of a business which they visit. Motivated by the popularity among real-world recommender systems of collaborative filtering, and matrix factorization methods in particular, this work explores these methods to build the recommender system.

2 Background

Methods for recommender systems Recommender systems can be implemented using several different paradigms. These include *content-based* recommendations and *collaborative filtering*[6]. Content-base recommendations rely on calculating item-item or user-user similarities based on constructed or learned item and user profiles. User profiles represent user preferences of each dimension (feature) in the profile. Item profiles indicate the amount or presence of each of those features in that item. For example when considering businesses, some features could be based on its location, its hours of operation, the items they sell, the friendliness of the staff, etc. Then, user profiles would be features that indicate to what extent each user prefers each of those features. Items are recommended that match a user's preferences, which are based on past reviews or purchases by the user. In effect, items are recommended if they are similar to items that a user has liked in the past.[5]. Systems using this approach require accurate profiles for user and items and can fail to recommend items that differ from past items that a user purchased, but that they might still enjoy.

In contrast, collaborative filtering methods produce recommender systems that recommend and item (in this case, a business) based on similarities between users. That is, it relies on the assumption that similar users will like similar items. Specifically, such a recommender system would assess user similarities and recommend business B to user u_i if users similar to u_i liked business B. This method requires plentiful user information but does not rely on extensive information on the items (businesses)[6].

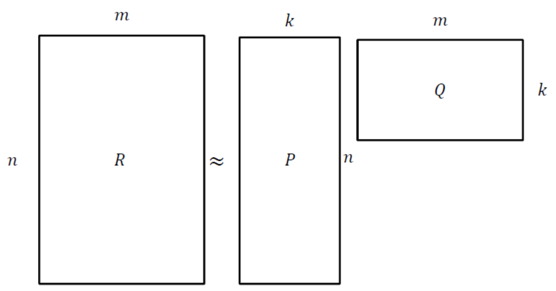
Other methods exist that combine the above methods or introduce additional information, such as social network structure to inform a user's recommendations. As a starting point, this project focuses on collaborative filtering since the dataset in question is large and includes many thousands of user. More details on that dataset will be provided in later sections.

Collaborative filtering through matrix factorization One method for predicting user ratings based on past rating is through matrix factorization. This model assumes that user ratings are the result of only a few factors. For example, we assume that there are only a handful of reasons that account for a customer rating a restaurant, for example, 3/5. Perhaps the speed of service, the quality of the food, and the price are enough to account for all user ratings. That is, as these three factors differ from business to business, the ratings of customers differ accordingly.

The *utility matrix* is a matrix with rows corresponding to users and columns corresponding to items. The entries in the matrix are populated by explicit or implicit user evaluations of items. Implicit evaluations correspond to information gathered through user actions, whereas explicit evaluations correspond to feedback from a user. For example implicit information may be whether a user clicked a link, purchased an item, or visited a business. Explicit information may be a review, a rating, or a "like"/"thumbs-up"/etc provided by a user. In this case, we focus on explicit ratings provided by users. Then, if r_{ij} is the rating user i gave business j the utility matrix $\mathbf{R}$ is defines as

$$\mathbf{R}_{ij} = \begin{cases} r_{ij} & \text{if user } i \text{ reviewed item } j \\ 0 & \text{otherwise} \end{cases}$$

Matrix factorization methods assume that all the information in the utility matrix can be accounted for by some k (latent) factors, where k is much smaller than the number of user and the number of item. If n is the number of users and m is the number of items, the $n \times m$ utility matrix can be factored as $\mathbf{R} = \mathbf{PQ}$, where $\mathbf{P}$ is $n \times k$ and $\mathbf{Q}$ is $k \times m$. To find such a P and Q is to find the

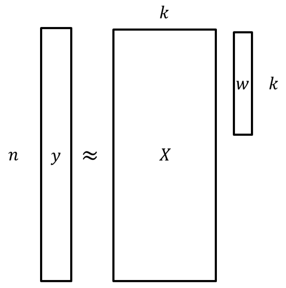


matrices that when multiplied together are as close to $\mathbf{R}$ as possible. Specifically, if $\mathbf{S} = \mathbf{PQ}$, then the root mean squared error, the error that should be minimized, is given by

$$RMSE = \sqrt{\frac{1}{n} \sum_{i,j} (s_{i,j} - r_{i,j})^2}$$

Finding optimal P and Q The problem of finding these matrices is reminiscent of another common problem in machine learning. Namely, (regularized) linear regression. In linear regression, there is a data matrix $\mathbf{X}$ and a target $\mathbf{y}$ and the goal is to find a coefficient vector $\mathbf{w}$ so that

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \lambda \mathbf{w}^T\mathbf{w}$$



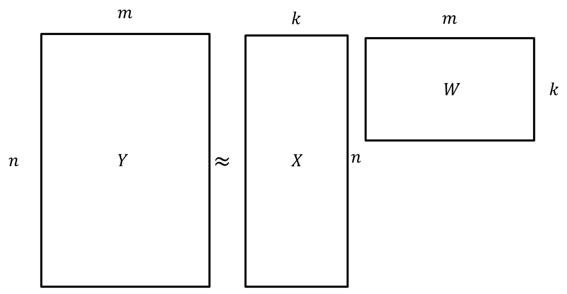
To find $\mathbf{w}$ that minimized the MSE

$$\left(MSE = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2 + \lambda\|\mathbf{w}\|_2^2 \right)$$

there is a closed-formed solution:

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T y$$

While this problem is similar in structure, the dimensionality of $\mathbf{y}$ and $\mathbf{w}$ is quite different than in the matrix factorization problem. We can move a step closer to the problem at hand by considering (regularized) multivariable linear regression, where If instead of predicting a single target value, we want to predict a vector for each instance $\mathbf{x}_i$, we need to find a matrix of coefficients



and the closed-form solution is

$$\mathbf{W} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

The multivariable regression problem

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + \lambda\|\mathbf{W}\|^2$$

looks very similar to the matrix factorization problem

$$\mathbf{R} = \mathbf{P}\mathbf{Q} + \lambda\left(\|\mathbf{P}\|^2 + \|\mathbf{Q}\|^2\right)$$

except that unlike in multivariable linear regression, in matrix factorization both $\mathbf{P}$ and $\mathbf{Q}$ are unknown. Optimizing for both jointly is non-convex, however if one were fixed, solving for the other matrix would be a convex quadratic optimization[3]. This approach is represented in the Alternating Least Squares method of matrix factorization. Specifically, one matrix is fixed and the other is found through the closed form equation listed in the description of the process below

Repeat:

- Fix $\mathbf{Q}$. Solve for $\mathbf{P}$
  - $\mathbf{P} = \left(\mathbf{Q}\mathbf{Q}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{Q}\mathbf{R}$
- Fix $\mathbf{P}$. Solve for $\mathbf{Q}$
  - $\mathbf{Q} = \left(\mathbf{P}\mathbf{P}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{P}\mathbf{R}^T$

The matrix to be inverted is only $k \times k$, and k is much smaller than n and m. Additionally, each column of $\mathbf{Q}$ and each row of $\mathbf{P}$ can be solved for in parallel during each of the steps in the algorithm[1].

To regularize each matrix $\mathbf{P}$ and $\mathbf{Q}$ proportionally with the number of non-zero entries in each row or column of each matrix, we modify the objective value to use weighted-λ-regularization[10]

$$\mathbf{R} = \mathbf{P}\mathbf{Q} + \lambda\left(\mathbf{n}_u\|\mathbf{P}\|^2 + \mathbf{n}_b\|\mathbf{Q}\|^2\right)$$

Where n_u is the number of nonzero entries in each row of $\mathbf{P}$ and n_b is the number of nonzero entries in each column of $\mathbf{Q}$. The process then becomes

Repeat:

- Fix $\mathbf{Q}$. Solve for $\mathbf{P}$
  - $\mathbf{P} = \left(\mathbf{Q}\mathbf{Q}^T + \mathbf{n}_u\lambda\mathbf{I}\right)^{-1}\mathbf{Q}\mathbf{R}$
- Fix $\mathbf{P}$. Solve for $\mathbf{Q}$
  - $\mathbf{Q} = \left(\mathbf{P}\mathbf{P}^T + \mathbf{n}_b\lambda\mathbf{I}\right)^{-1}\mathbf{P}\mathbf{R}^T$

3 The Data: Yelp Dataset Challenge 2016

The Yelp Dataset Challenge data offers a rich collection of data about businesses and users on Yelp. These data include information about users, businesses, reviews, user ratings, and other information collected by Yelp, such as user "tips" for businesses and counts of user "check-ins" to businesses.

The dataset includes businesses and users from ten cities in North American and Europe. These include Edinburgh, U.K., Karlsruhe, Germany, Montreal and Waterloo, Canada, and six cities in the U.S. The number of businesses, users, and reviews in each cities is not uniform. For example, the dataset includes 36,500 businesses in Las Vegas, but only 530 in Toronto. Table 1 shows the counts of types of data by location.

User data includes information about the each user's account, such as the account creation date, the number of reviews generated by the user, the average rating given by the user, the user's

[1]However this parallelization is not implemented in the code accompanying this project

State	restaurants	reviews	check-ins
AZ	9427	622446	8570
NV	5912	662428	5505
QC	3385	63987	2741
NC	2421	112794	2212
PA	1671	78754	1457
EDH	1232	15189	822
WI	1172	49243	1018
BW	571	2714	160
ON	372	4814	297
IL	317	13054	259
SC	143	3532	125

Table 1: Counts of different data by state

Category	count
Restaurants	26729
Shopping	12444
Food	10143
Beauty & Spas	7490
Health & Medical	6106
Home Services	5866
Nightlife	5507
Automotive	4888
Bars	4727
Local Services	4041

Table 2: Counts of top 10 categories in data set

"friends," "compliments" from other users, and votes from other users on reviews written by the user. In addition, each review and tip is linked to the user who wrote the review or tip. Business data includes information about the business category, location, hours of operation, average rating, number of reviews, the count of user check-in by date (though the specific user ID for each check-in is not provided), and lastly every review is linked to the business about which it was written. Business categories are tags provided for each business such as "Restaurants," "Shopping," etc. A business may have multiple categories, for example a single business may be tagged with the categories "Restaurants" as well as "Bars," "Food," and "Sandwiches," etc. There are 1,017 categories of businesses in the dataset, with the top 10 most common ones listed in Table 2.

To narrow the scope of recommendations, we choose to predict ratings of a single category of businesses, namely restaurants. Of the 85,901 businesses in the dataset, 26,729 of them are tagged as "Restaurants," so the choice to focus on this type of business is reasonable. Focusing on one business category also allows the system to model user preferences specific to restaurants that may not apply to other business categories, such as "Automotive." It is reasonable to expect that users judge restaurants and automotive businesses quite differently and users that agree about restaurants may not agree about automotive businesses or vice versa.

To further narrow the scope of the model, and to promote faster computation, we focus on data from a single city. Namely, we choose Pittsburgh,PA as the city for which to build the recommender system because with 78,754 reviews for restaurants it provides abundant information for training without requiring computation on "too much" data[2]. By focusing the model on one type of business and one location, we hope to be able to accurately model the user preferences that may differ by business type or by location, or both, better than trying to model preferences for all categories in all locations at once.

[2]Here we (conspicuously) omit a precise definition or analysis of "too much" data, but suffice it to say that a dataset small enough to work with locally on a laptop was desirable

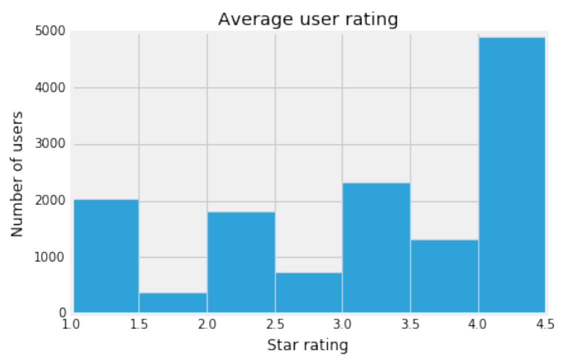


Figure 1: Average user rating distribution of Pittsburgh data

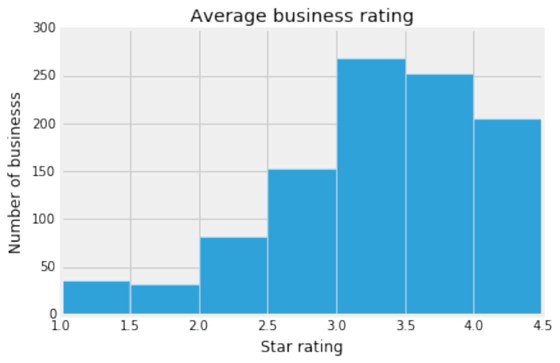


Figure 2: Average restaurant rating distribution of Pittsburgh data

3.1 Constructing the Utility Matrix

To build the recommender system, we do not use any of the check-in, tips, or business and user features provided in the dataset. Instead, we use only the ratings associated with reviews as explicit preferences provided by users. In particular, associated with each review written by a user is a rating from 1 to 5 indicating the user's rating of the business. Figure 1 shows the distribution of the average rating given by users and Figure 2 shows the average rating given to restaurants. These ratings are used to populate the *utility matrix* R. As is common in the setting of recommender systems, each user has only rated a small fraction of the total number of items available, therefore the utility matrix is very sparse. See Figure 3 for the distribution of the number of restaurants reviewed by users and Figure 4 for a representation of the nonzero entries in the utility matrix.

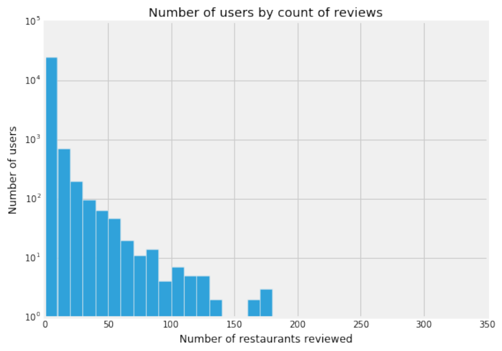


Figure 3: Review counts by user. Note that more than 10,000 users provide 10 or fewer reviews

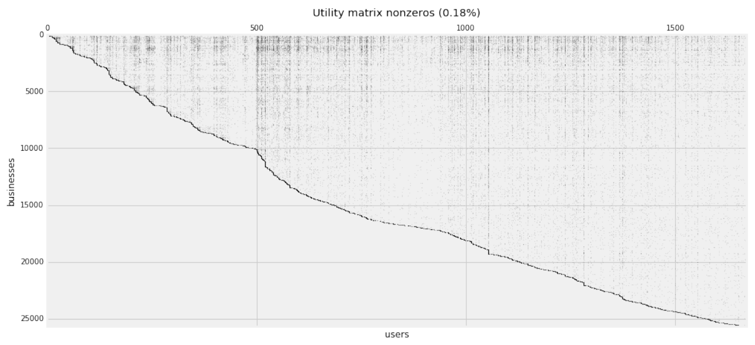


Figure 4: Sparcity of utility matrix. Black dots indicate the 0.18% of entries in the matrix that are non-zero.

4 Experiments

4.1 Separating the data

To train and evaluate the recommender system, the dateset must be split by time. The first time period is used to train the recommender system and the later time period is used to evaluate it. This scenario mimics the real-world scenario where a system can train on past data to predict future ratings. To find the appropriate train-test split, the number of reviews over time is analyzed. The number of reviews in the dataset over time are shown in Figure **??**. The cutoff was identified such that 80% of reviews occurred before the cutoff and 20% occurred after. The test set were the reviews after the cutoff that were made by users with at least one review during the training period.

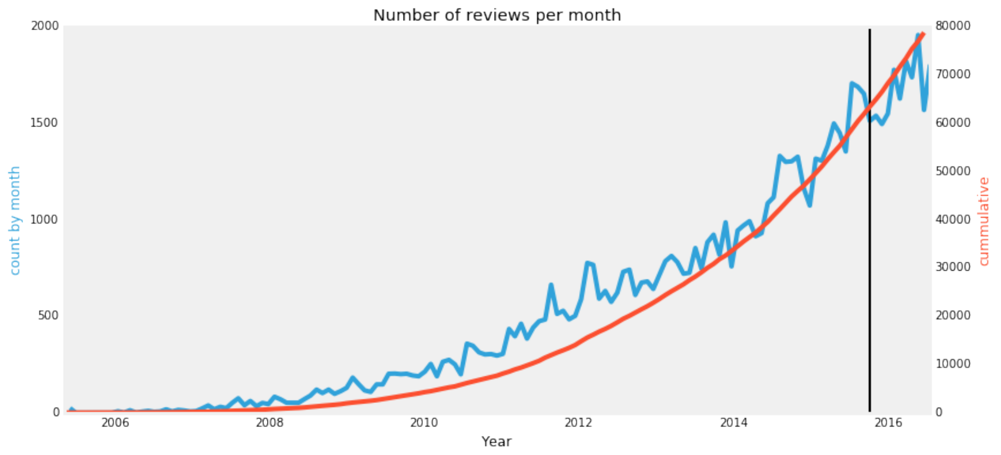


Figure 5: Number of reviews over time. The black line indicates the cut-off such that 80% of reviews occurred before and 20% afterwards

4.2 Training the model

The number of iterations of ALS was fixed to 20 iterations. The algorithm was implemented to find **P** and **Q** to approximate the utility matrix containing the testing ratings. The RMSE for each iteration is shown in Figure 6

4.3 Cross Validation

To cross validate the model hyperparameters, namely the number k of latent factors and the regularization coeffiecient, we set aside 20% of the training data and remove it from the training set. The average MSE was calculated for each parameter combination and used to select the best values in the grid search. The result of this cross validation is shown in Table 3. The best parameters were found to be $\lambda = 1$ and $k = 2$. Figures 7 and 8 show the values of the MSE on the validation set for each combination of hyperparameters. These figures show that for large values of λ, the value of k did not affect the error as much as for smaller values of λ. Furthermore, increasing

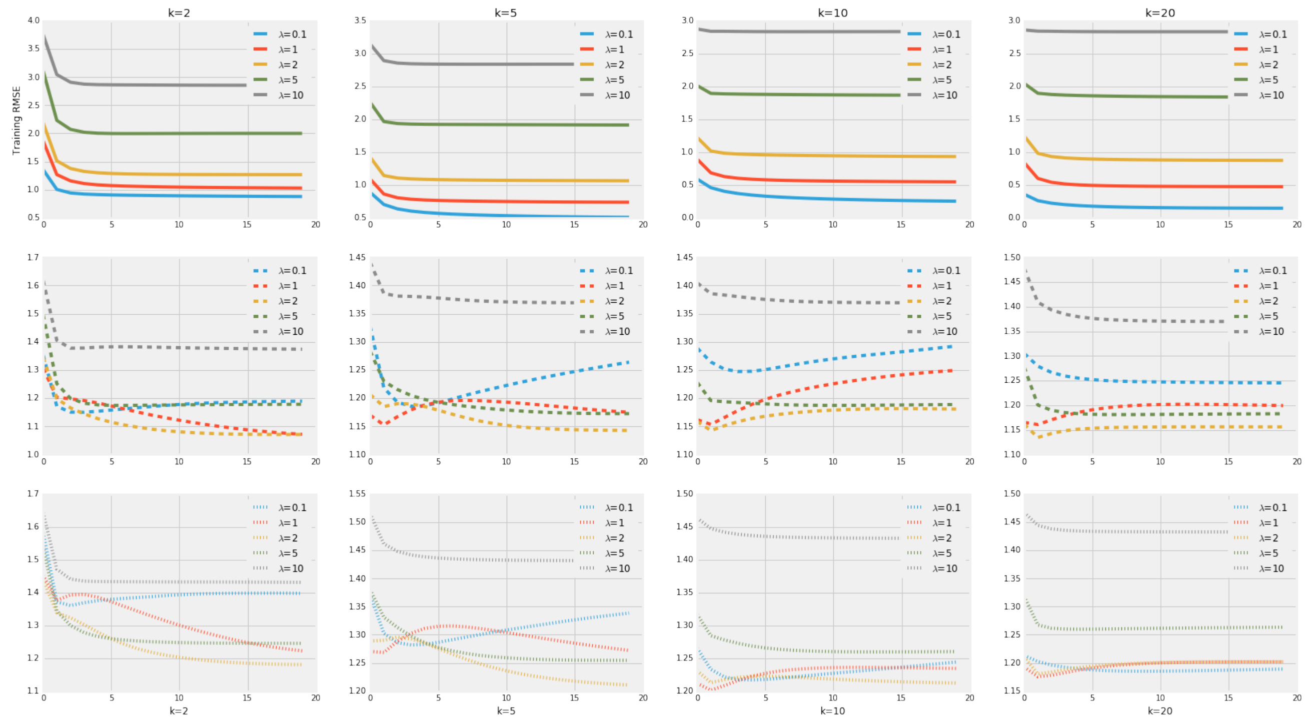


Figure 6: The RMSE for each iteration for different hyperparameter settings. The top row shows the training RMSE. This quickly converges. The middle row shows the validation set RMSE. The bottom row shows the test set RMSE. The ALS algorithm finds $\mathbf{P}$ and $\mathbf{Q}$ to approximate $\mathbf{R}$, which only includes the training set. This explains why predicting values outside the test set does not have the same convergence behavior, though all lines eventually converge as $\mathbf{P}$ and $\mathbf{Q}$ converge

the value of k after some point for a fixed λ did not have much effect on the error. Overall however, Table 3 shows that the average MSE varies a small amount all hyperparameter settings.

4.4 Evaluation

Finally, we evaluate the recommender system against a baseline method to compare the performance. A simple way to recommend items is purely on the item's popularity, disregarding a specific user's preference. This approach is used here as a baseline. For each user i in the test set, the rating for business j is predicted to be the average rating for business j during the training period. If a business does not have an average rating for the training period (because it had no ratings then), the prediction is given by the average for all businesses. Specifically, for this dataset, the average for all businesses in the training set is 2.919 stars out of 5. Across all users, the average Baseline MSE was 1.1550.

The matrix factorization approach predicts the rating user i gave business j as the value in $\mathbf{S}_{ij}$ where $\mathbf{S} = \mathbf{PQ}$. Specifically, we use the $\mathbf{P}$ and $\mathbf{Q}$ found using the hyperparameters chosen through cross validation. With these parameters, the system achieves an average MSE of 1.1531, beating

	$k = 2$	$k = 5$	$k = 10$	$k = 20$
$\lambda = 0.1$	1.0147	1.0973	1.1478	1.1303
$\lambda = 1$	**0.9807**	1.0564	1.1174	1.0953
$\lambda = 2$	0.9940	1.0471	1.0767	1.0682
$\lambda = 5$	1.0715	1.0663	1.0795	1.0749
$\lambda = 10$	1.1944	1.1900	1.1902	1.1904

Table 3: Average MSE on validation set for various parameter settings

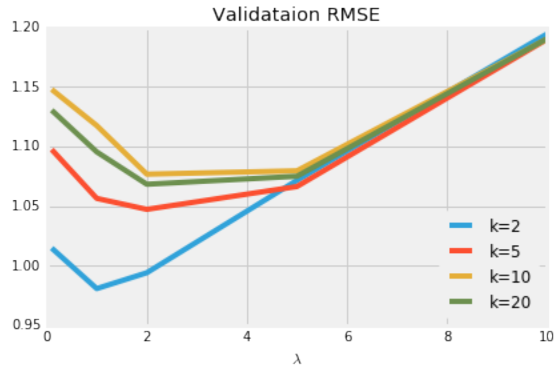


Figure 7: Validation MSE as a function of λ for varying values of k

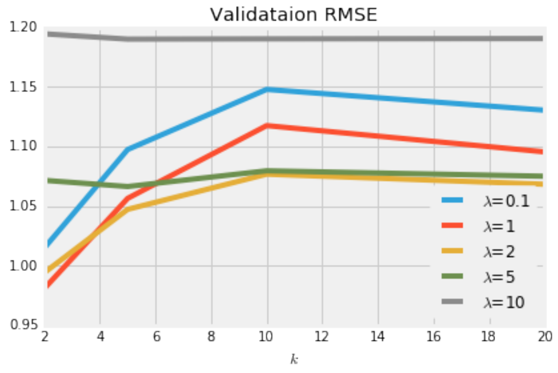


Figure 8: Validation MSE as a function of k for varying values of λ

the baseline (barely). Note that a different setting of hyperparamters ($k = 2$, $\lambda = 2$) achieves an average MSE of 1.1375, a more significant improvement over baseline. These parameters were the second-best in cross validation and when analyzing the convergence behavior of $\lambda = 1$ and $\lambda = 2$ for $k = 2$, it is suggested that these values had not yet converged after 20 iterations. As the RMSE appears to still be decreasing, it is possible that running further iterations would yield better results. This is left to future work.

References

[1] Gediminas Adomavicius, Lior Rokach, and Bracha Shapira, *Recommender Systems Handbook*, vol. 54, 2015.

[2] Yifan Hu, Yehuda Koren, and Chris Volinsky, *Collaborative Filtering for Implicit Feedback Datasets Yifan*, IEEE Int. Conf. Data Min. (2008), 263–272.

[3] Y. Koren, R. Bell, and C. Volinsky, *Matrix Factorization Techniques for Recommender Systems*, Computer (Long. Beach. Calif). **42** (2009), no. 8, 42–49.

[4] Yehuda Koren, Robert Bell, Chris Volinsky, et al., *Matrix factorization techniques for recommender systems*, Computer **42** (2009), no. 8, 30–37.

[5] A. Rajaraman and J.D. Ullman, *Mining of massive datasets*, Cambridge University Press, 2011.

[6] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, *Recommender Systems Handbook*, vol. 53, 2011.

[7] Hsiang-fu Yu, Cho-jui Hsieh, Si Si, and Inderjit S Dhillon, *Parallel Matrix Factorization for Recommender Systems*.

[8] Dave Zachariah, Martin Sundin, Magnus Jansson, and Saikat Chatterjee, *Alternating Least-Squares for Low-Rank Matrix Reconstruction*, 1–4.

[9] Wei Zeng, An Zeng, Ming Sheng Shang, and Yi Cheng Zhang, *Information filtering in sparse online systems: Recommendation via semi-local diffusion*, PLoS One **8** (2013), no. 11.

[10] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan, *Large-scale parallel collaborative filtering for the netflix prize*, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) **5034 LNCS** (2008), 337–348.