CSE 255 Assignment 7

Alexander Asplund, William Fedus

September 25, 2015

1 Introduction

In this paper we implement a collaborative filtering algorithm on the MovieLens dataset to predict movie ratings for the users. The original matrix, which contains the movie ratings on a 1-5 scale for the users, has many missing entries. Rank-K factorization is used to construct the filtering algorithm and alternating least squares is then performed on the two lower rank matrices in order to fill in the missing entries of the original matrix and thus predict all movie ratings for all users. The final model is found to be one with rank-40 decomposition and a regularization parameter of $\lambda = 2^1$. This model both minimizes the total root mean squared error (RMSE) and the mean absolute error (MAE) of the ratings predictions, generating an average RMSE on cross-validation folds of 0.930 and an average MAE of 0.739.

2 Data Statistics and Preprocessing

The MovieLens dataset consists of a randomly ordered set of 100,000 ratings provided by 943 users on 1682 movies, with each user providing at least 20 ratings. Ratings for the movies are provided on an integral scale of $\{1..5\}$. This data, collected over a seven-month period, is already cleaned by eliminating users with fewer than 20 ratings and those who had not provided complete demographic information. Each movie included a list of parameters including title, release date, video release date as well as binary indicator variables for classification within 19 genres, where an individual film could be in several genres at once. Five cross-validation folds are already provided for the MovieLens dataset, so no partition is necessary.

3 Training Methodology

The input to the collaborative filtering algorithm is an incomplete matrix of ratings, matrix A, with each row corresponding to a user $i \in \{1..m\}$ and each column corresponding to a movie $j \in \{1..n\}$. Each element $a_{i,j}$ in matrix A corresponds to user $i's$ rating of movie j.

Alternating Least Squares (ALS) is used to select an optimal K-rank matrix factorization $A = UV$, an approach which is computationally tractable in large scale problems. U is an m-by-K matrix and V is a K-by-n matrix. We initialize these matrices according to a Gaussian distribution with $\mu = 0$ and $\sigma = 1$. The goal is to minimize the sum of square errors

$$\sum_{\text{known } a_{ij}} (V_{:j}^T U_{i:} - a_{ij})^2 \tag{1}$$

Where u_i and v_j is the ith row of U and v_j is the jth column of V.

Alternating Least Squares optimizes U and V by first fixing V and optimizing U and then fixing U and optimizing V. These two steps are performed iteratively until U and V are sufficiently close to convergence. The optimization step in ALS can also be done by optimizing the rows or columns of the target matrix sequentially since their optimal values are independent of each other given the values of the fixed matrix. We optimize U by individually optimizing each row $U_{u:}$ such that we minimize

$$\sum_{\text{known } u,j} (V_{:j}^T U_{u:} - a_{uj})^2 + \lambda^2 \|\beta\|_2^2 \tag{2}$$

We achieve this by constructing a matrix V_u and column vector y and taking the row vector $U_{u:}$ to be the linear least squares solution to $V_u x = y$. The rows of V_u are the transpose of columns $V_{:j}$ and the rows of y are the known values a_{uj}.

In the second step of of the iteration we optimize V by individually optimizing each column $V_{:v}$ such that we minimize

$$\sum_{\text{known } i,v} (V_{:v}^T U_{i:} - a_{iv})^2 \tag{3}$$

In the same manner as when optimizing U we set the columns of V to be the linear least squares solution to $U_v x = y$.

To avoid overfitting the known values regularization must be introduced to the objective. We introduce a regularization constant λ such that the contribution of a single (i, j) to the global objective, when $V_{:v}$ the current target of optimization, is

$$\sum_{\text{known } a_{iv}} (V_{:v}^T U_{i:} - a_{iv})^2 + \lambda^2 \|V_{:v}\|_2^2 \tag{4}$$

We next consider how to adapt the optimization of U when solving $U_v x = y$ so that it optimizes for this regularized objective. The original solution will optimize for the first term. We add the second term to consideration by appending the K-by-K identity matrix multiplied with λ to the bottom of U_v and a K-by-1 zero vector to the bottom of y. To ensure that the regularization constant puts equal pressure independent of the number of known values for a given row or column we instead use local lambda λ_L for each local optimization problem:

$$\lambda_L = \lambda\sqrt{N/K} \tag{5}$$

N is the number of known ratings and thus observations for the given local problem. This is necessary because K is fixed resulting in a constant regularization pressure relative to K while N changes. This change ensures that the regularization pressure is constant relative to N. Dividing by K is not necessary, since it is constant and thus can be absorbed into the original λ, nor is it necessarily mathematically well motivated. The equation locallambda is presented with division by K as this is how it was done for the experiments presented in this paper and theore is necessary to reproduce our exact results.

As a final step, regression determined ratings a_{ij} which fall outside the possible range of the movie rating scale of $\mathbb{R} \in [1,5]$ are mapped to the corresponding position within the allowable range, i.e. movies ratings $a_{ij} > 5$ are mapped to a rating of 5 and $a_{ij} < 1$ are mapped to a rating of 1.

4 Design Considerations

4.1 Selecting Hyperparameters

One of the first design considerations is the selection of the two hyperparameters of the model, k and λ. In order to find the values that generalize best to new test data, we first perform a coarse grid search over the two-dimensional k-λ space. To find the approximate optimal values, we train the model over a k range of {1,20,40,60,80,100} and a λ range of 2^i for $i \in \{-7..7\}$ on the 80% training partition. With each k and λ value established via training on the 80% partition, the RMSE of this model on the corresponding disjoint cross-validation fold is then recorded after a 10 iterations or less if improvement in RMSE between iterations falls below 0.001. The coarse grid search determines an optimal k of 40 and λ of 2^1. We choose not to implement finer grid search so as to minimize the risk of overfitting the model to the training folds.

We also experimented with the addition of separate regularization parameters for the U and V matrices, λ_1 and λ_2 respectively.

This approach did marginally improve the average performance on cross-validation folds by $\approx 5\%$, however, it came at the expense of dramatically increased computation time (hyperparameter search space is in a +1 higher dimension) and it increases the risk of overfitting the training data. Theore, with these considerations, we chose to use the original single λ model described above.

4.2 Normalizing for User and Item Means

An additional design consideration is the normalization of the entries according to the user $i's$ mean, item $j's$ mean and the global matrix mean. The motivation is to get all features on an equal range; for instance, for user normalized version, 0 for element $a_{i:}$ would represent a item rated exactly in line with user $i's$ total average rating. In a item normalized version, 0 for element $a_{:j}$ would represent a user rating that item in line with item $j's$ total average rating.

To test the efficacy of these three approaches, we utilize the optimal k selected via the coarse grid search, do a new search for optimal lambda and then try pair testing for each approach. First, we train the final model on the five training partition without user means normalized and determine the average RMSE on the five cross validation partitions. Next, we train each of the three normalization models on the five training partitions and determine the average RMSE on the five cross validation partitions.

The result of these experiments generates an average RMSE without normalization of 0.930, with user normalization of 0.969, with item normalization of 0.986 and with global normalization of 0.941; given these results, we theore proceeded without using normalization as this generated the lowest RMSE on cross-validation folds.

4.3 Additional Design Considerations

The experiments above do not comprehensively cover all design considerations and additional decisions must be made in the construction of the algorithm. Initialization of the U and V matrices is one design consideration and we choose to initialize the matrices with small values distributed evenly on $\mathbb{R} \in [-1, 1]$. We felt this approach created sufficiently distributed small values without heavily biasing the final model towards any particular result.

Finally, one last design consideration is whether to train the dimensions sequentially or simultaneously. We chose to train sequentially since the problem should decompose to approximately the same form and the memory access time of smaller sparse matrices in the sequential formulation is faster.

5 Results and Investigations

5.1 Results

To assess the performance of the collaborative filtering algorithm, we measured the root mean squared error (RMSE) and absolute error (MAE) as a function of the rank decomposition parameter, k.

From these graphs, one sees that the lowest absolute error and lowest mean squared error as a function of k occured for $k = 40$. The final model with hyperparameters $k = 40$ and $\lambda = 2^1$ achieved an RMSE of 0.930.

5.2 Investigations

For our final model, we can identify and analyze the learned latent vectors in order to better understand the mechanics behind the algorithm. The learned

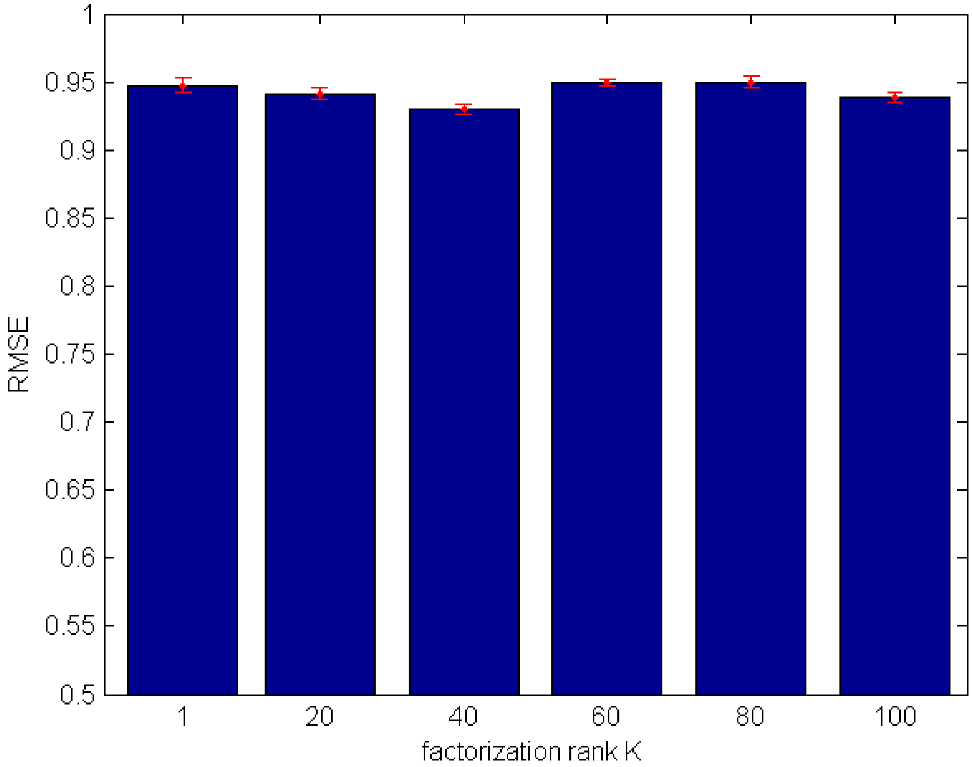


Figure 1: The above bar chart is the measured RMSE, as a function of the k factorization rank, on the five cross-validation folds. The error bars represent one standard deviation, obtained by measuring the variance over the cross-validation folds.

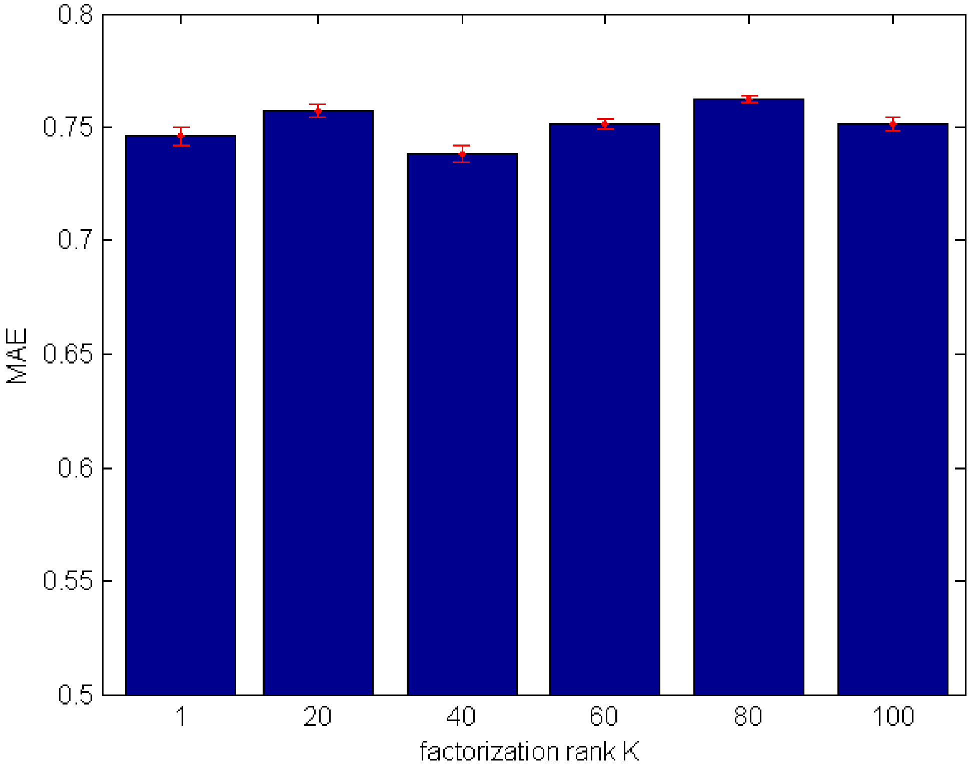


Figure 2: The above bar chart is the measured MAE, as a function of the k factorization rank, on the five cross-validation folds. The error bars represent one standard deviation, obtained by measuring the variance over the cross-validation folds.

latent vectors are the rows for i users in matrix U or the columns for j movies in the matrix V. A user latent vector is interpretable as the perence vector for user i over the set of k features and the item latent vector is interpretable as the properties of movie j as defined by the set of k features. More specifically, the value of feature k for movie j is described by the coefficient v_{kj} of the V matrix and the predicted coefficient of feature k for user i is described by the coefficient u_{ik} of the U matrix. For our final model, 40 learned dimensions are found.

5.2.1 Determining Similar Movies

To begin this investigation of latent vectors, we first determine which movies have the highest similarity, s, via the cosine of of the angle between them. Theore, a $j \times j$ matrix is created, which we will call matrix S, which is the entire list of calculated similarities of movie a with all other movies b, including its self where $s_{aa} = 1.0$.

Exploration of the data begins by grouping movies with the highest similarities into sets to investigate the algorithm. Movies which often shared similar explicit feature vectors of genres are often grouped into the same sets, indicating a correlation of learned and provided explicit features. For instance, below are two examples of the top selected similar movies for Star Wars and Die Hard with a Vengeance.

Star Wars: {The Empire Strikes Back, Return of the Jedi, Raiders of the Lost Ark, Indiana Jones and the Lost Crusader, The Outlaw}

Die Hard with a Vengeance: {Die Hard 2, Die Hard, The Rock, The Hunt for Red October, Crimson Tide}

The first similar selected set for the film, Star Wars, appears to have successfully grouped movies according to genre, director and sequels. The second similar selected set for the film, Die Hard with a Vengeance appears successful as well, selecting the prequel and sequel to the film as well as other action movies with a famous male star (Sean Connery).

5.2.2 Assessing Learned Dimensions

To approach this dataset differently, we also aimed to interpret the k learned dimensions in V of the algorithm. To do so, for each k dimension, we ranked the top films and created subsets of 50 for those. Below is an example for some of the films which had the highest value for the learned dimension, $k = 40$.

{Wings of Desire, Lost Highway, Thin Line Between Love and Hate, Les Miserables, Double vie de Veronique, Boogie Nights, The Celluloid Closet, Jude, Boys Life 2, Ruby in Paradise}

This learned dimension may be likened to the explicit given feature binary indicator for drama. In the subset of 50 films that scored the highest on $k = 40$ learned dimension, 45% have the drama label, far in excess of the 25% drama label base rate for all films rated. However, it should be noted that other learned dimensions appear to be grouping movies on more abstract considerations and thus did not correspond to conventional film classification methodologies.

5.2.3 Correlations with Explicit Features

Each movie had an explicit genre feature vector provided and thus we were able to analyze the correlations of the virtual features with this provided information. First, for the top 50 movies ranked according to each of the 40 learned dimensions, the 19 indicator variables are summed and normalized. As anticipated, the high base rate of drama (25% of labels) and comedy (17% of labels) influenced the virtual learned features and almost all of the 40 learned dimensions demonstrated a strong correlation to these genres.

5.2.4 Movie Outliers

Finally, to conclude the investigations, we sought to define and search for outliers of the movies. In our first approach, we use the initial similarity matrix, S and determine which films are the least similar to all others. More specifically, we ordered the films for which the next most similar movie was the lowest according to their dot product. The result is the eclectic group of movies in the set below, films that the algorithm was unable to easily match to any others.

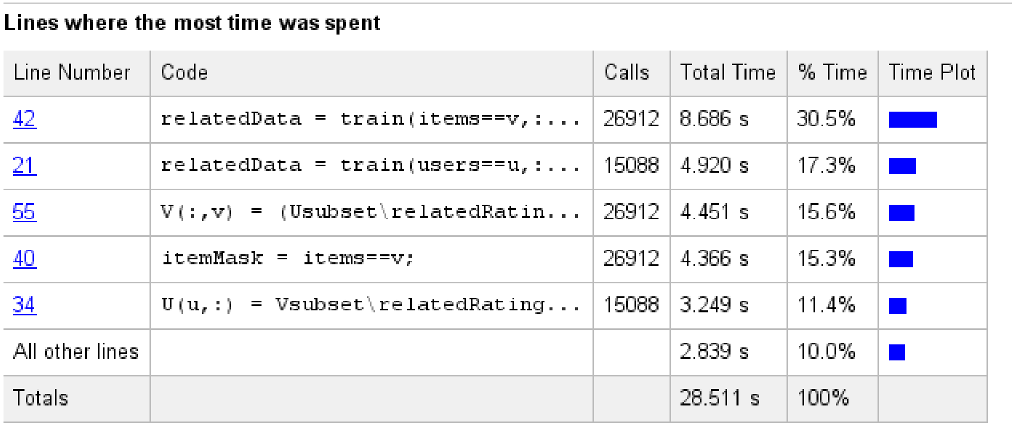
{Amityville 2: The Possession, Surviving Picasso, The Stupids, Kazaam, Showgirls}

For a second approach, we simply determined for which films our algorithm had the worst absolute error in predicting ratings per rating. The set below represents the films for which our algorithm performed the worst, and interestingly, the algorithm had a positive bias to each of these five films.

{Natural Born Killers, Koyaanisqatsi, The Cook The Thief, His Wife & Her Lover, Pink Floyd, the Wall, Nightmare on Elm Street}

5.3 Time complexity of ALS

Performing the linear least squares method with D known values and z predictors takes $O(Dz^2)$ time. In both cases, if the full matrix has a total of C known values, optimizing a single row of U then takes $O(C_{i:}k^2)$ and optimizing a single column of V takes $O(C_{:j}k^2)$ time. Optimizing the entire matrix in each step takes $O(\sum_x C_x k^2) = O(Ck^2)$ Completing one full iteration of ALS then takes $O(Ck^2 + Ck^2) = O(Ck^2)$ time. The number of iterations is constant with regards to the input size, and so the final running time is $O(Ck^2)$.



Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
42	`relatedData = train(items==v,:...`	26912	8.686 s	30.5%	■
21	`relatedData = train(users==u,:...`	15088	4.920 s	17.3%	■
55	`V(:,v) = (Usubset\relatedRatin...`	26912	4.451 s	15.6%	■
40	`itemMask = items==v;`	26912	4.366 s	15.3%	■
34	`U(u,:) = Vsubset\relatedRating...`	15088	3.249 s	11.4%	■
All other lines			2.839 s	10.0%	■
Totals			28.511 s	100%	

Line 1, 2 and 4 in the profiling information is the time spent retrieving retrieving users that have rated a given movie and retrieving movies a given user has rated. Line 3 and line 5 is the time spent working on solving local least squares problems. These time percentages are global to the entire procedures, and so these are clearly the main bottlenecks. This is not surprising since they are all lines from the innermost loops of the implementation.

5.4 Feasibility on the Netflix dataset

The algorithm is linear in the number of known values C. In the Netflix data $C = 10^8$ and so the run time is quadratic in k with a very large constant factor. Given the running times of 4.94 seconds for our dataset of $C = 10^5$ with $K = 10$, we project a running time of 82 minutes for the Netflix data with a rank 10 factorization and thereore believe this is a scalable algorithm for a dataset of that size. If we instead consider a rank-40 factorization the projected time increases to 7.5 hours. All experiments were performed with an Intel i7 3635QM processor.

6 Conclusions

The final chosen model of $k = 40$ and $\lambda = 2$ performed the best on average over the cross-validation folds by both the RMSE and MAE measures. However, it should be noted that the mechanism behind the precise shape of the error graphs as a function of factorization rank k remains unclear. With the RMSE measure, the performance appears to be improving with higher $k > 60$ but with the MAE measure, this trend does not hold. Without the time constraints, additional testing over the factorization rank k variable may lead to insights about when a simpler model may be substituted for more complex models with a minimal drop in performance.

Additionally, the final collaborative filtering algorithm performs surprisingly similar to the naive classifier, which simply predicts user $i's$ average rating for all movies j or simply predicts movie $j's$ average rating for all users i. The naive classifier, averaging users ratings, generated an RMSE of 1.063 and the naive classifier, averaging items ratings, generated an RMSE of 1.041. This compares to an RMSE of 0.930 for our final model, indicating the difficulty of achieving each percentage point of improvement on this set. Though rank-k factorization is a time efficient approach that scales well to larger databases, more sophisticated methods may be necessary to dramatically improve RMSE.