8. Recommendation Systems

8.1 Goals

The goal of this exercise is to:

- Build a recommendation system.
- Learn to evaluate its performance.
- Implement alternating least-squares (ALS) algorithm.
- Choose appropriate number of factors, as well as regularization parameters.
- Compare the performance of ALS to a few baselines.

8.2 The Data

We will use the Movielens-100k data set. It is available on the course website (the file movielens100k.mat). The matrix ratings contains ratings of 1682 movies, rated by 943 users. Each user has only rated a few movies, therefore many ratings are missing. Our goal is to build a recommendation system that predicts the missing ratings. More details about the data-set can be found at http://grouplens.org/datasets/movielens/.

The data also contains meta-data about the movies. Even though we will not use this information in this lab, you can look into it for fun. The variable movieInformation contains the ID of the movie, its title, release date, and a link to the corresponding IMDB page. The vector genreNames contains the name of 19 genres, and the matrix movieMetaData contains the genre for all the movies. Note that some movies belong to multiple genres. Finally, the matrix timeStamp contains the actual time when the rating was provided.

8.2.1 Visualization and Creating Train-Test Splits

Since our goal is to predict the unseen ratings, we can create a test set by “punching holes” in the matrix, i.e. we randomly select some of the ratings in the matrix as test points, while we leave the rest for training.

We have provided the code to for this; see rec_sys.m where we randomly choose a maximum of 5 test entries per user. We only consider users and movies that have more than 10 ratings. This way, we keep some minimum training data for users and movies that have very few ratings. The matrix X contains the training data while the matrix Xtest contains the test data. Note that we use sparse matrices to save memory, i.e. we store the indices and values of non-zero entries only. You can read more about sparse matrices in Matlab’s documentation.

When you run rec_sys.m, you will see two plots. The first plot is shown in Figure 8.1 and shows the number of ratings for each user and each movie. We can see that the number of ratings varies quite a lot among users and movies. This is very typical of datasets for recommendation systems, and Big Data in general. The second plot is shown in Figure 8.2 which shows a train-test split created using rec_sys.m.

Note that you can generate many such train-test splits by changing the random seed.
8. Recommendation Systems

Figure 8.1: The left figure shows the number of ratings for each user, while the right figure shows the number of ratings for each movie. Both numbers are shows in a descending order for clarity.

8.2.2 Performance Evaluation

We will use the root mean-square error (RMSE) to measure the performance. Given test ratings $x_{dn}$ for $d$’th movie and $n$’th user and its estimate $\hat{x}_{dn}$, we compute RMSE as follows:

$$\text{RMSE} := \sqrt{\frac{1}{N_t} \sum_{n=1}^{N} \sum_{d \in \Omega_n} (x_{dn} - \hat{x}_{dn})^2}$$  \hspace{1cm} (8.1)

where $N_t$ is total number of test pairs and $\Omega_n$ is the set of test movies for the $n$’th user.

8.2.3 Baseline Models

We will use the following models, that use the mean to predict, as baselines:

Global Mean: $\hat{x} = \frac{1}{N_{tr}} \sum_{n_1=1}^{N} \sum_{d_1 \in \Omega_{n_1}} x_{d_1n_1}$ \hspace{1cm} (8.2)

User Mean: $\hat{x}_n = \frac{1}{N_n} \sum_{d_1 \in \Omega_n} x_{d_1n}$ \hspace{1cm} (8.3)

Movie Mean: $\hat{x}_d = \frac{1}{N_d} \sum_{n_1 \in \Omega_d} x_{d_1n_1}$ \hspace{1cm} (8.4)

where $N_{tr}$ is the total number of non-zero entries in the training data matrix, $N_n$ is the total number of ratings for $n$’th user, $N_d$ is the total number of ratings for $d$’th movie,
8.3 Alternating Least Squares

Exercise 8.1 We will compare the above three baselines first.
- Before implementing, think about the following: which of the three models will give the best performance, and why?
- We have implemented the Global Mean model in rec_sys.m. Implement the other two baselines.
- Now, compare the models. Which model gives you the lowest RMSE? Which is the best model overall? Hint: You can change the random seed and generate several estimates of RMSE to compute a confidence score.

8.3 Alternating Least Squares

Exercise 8.2 We will now implement the ALS algorithm.
- Fill in the code for ALS in rec_sys.m. You only have to write the updates of Z given W and vice-versa. We have provided you the code to compute RMSE.
- How does the test error vary with $M$, $\lambda_z$ and $\lambda_w$?
- Select appropriate values of $M$, $\lambda_z$ and $\lambda_w$. You can use cross-validation, but that might be too expensive (why?). Think of ways to reduce computation.
- How much improvement do you get using ALS?