Yanjia Zhang

Yang Sun

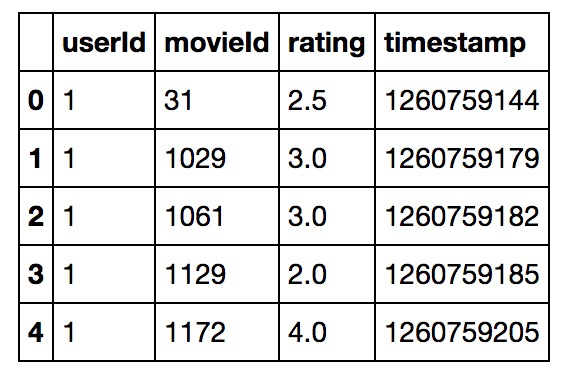
Yichao Chen

**Objective**

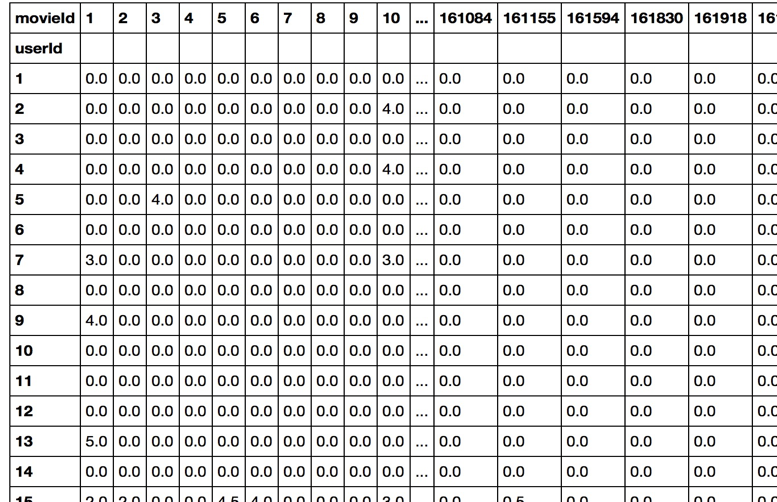
The idea of our Machine Learning project is to build a movie recommendation system using model-based collaborative filtering method, which is based on matrix factorization. This is an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommendation system because it deals better in scalability and sparsity.

**Data Preparation**

The dataset we are going to use for this project is from <https://grouplens.org/datasets/movielens/latest/>. The original dataset has 100,004 ratings applied to 9,125 movies provided by 671 users. All individual users has rated at least 20 movies in the dataset.



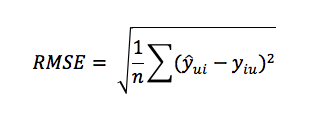
We transform our dataframe into a matrix with userids as rows, movieids as columns, and corresponding ratings as values. Then we have a 671\*9066 matrix with a lot of missing values since each user only rates a small number of movies compared to the total number of movies in our dataset. Thus, we replace all the Nans with 0.



In the end, we shuffle and split our dataset into training dataset (70%), validation dataset (20%) and test dataset (10%).

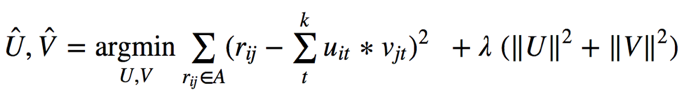
**Evaluation Methods**

The evaluation method we choose is the Root Mean Squared Error(RMSE). RMSE measures the difference between the predicted value by a model and the actual value. In our example, it represents the how far from the true ratings the predictions are.



**Models Selection**

One advantage of Matrix Factorization model is that we only need to use the ratings as the feature, which significantly reduces the feature dimensions. Instead of computing similarity between users and movies, we could predict users’ star ratings by uncovering the latent factors underlying their preferences for movies. The main idea of the Matrix Factorization model is that the ratings matrix can be approximated by a low-rank matrix. Specifically speaking, both the large-scale users and movies can be divided into smaller groups, which is controlled by the hyper-parameter – the number of latent factors. In this case, fewer parameters are needed to describe users’ preferences. In other words, we factorize an m\*n rating matrix into the product of an m\*k matrix and a k\*n matrix and less data are needed to train the model. Therefore, the Matrix Factorization method effectively mitigates the damage of data sparsity.



In the above specification, uit indicates how user i prefers latent factor t, vjt indicates how much latent factor describes movie j, k is the number of latent factors and λ is regularization parameter. We set above function as our objective function. Also, in order to reduce overfitting, we regularized the components of U and V with L-2 norm.

**Performance of Baseline Model**

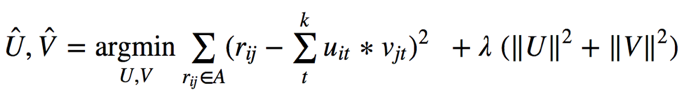
In our Baseline Model, we use stochastic gradient descent (SGD) to minimize the above optimization equation. It works by walking through every point in the training data, predicts the target, and calculates the regularized squared error in the equation. The following table shows performance (regularized squared error) in our baseline model:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Iteration** | **Step Size** | **Regularization** | **Latent Feature** | **Regularized Squared Error** |
| **Model 1** | 1000 | 0.0002 | 0.02 | 5 | 51919 |
| **Model 2** | 1000 | 0.002 | 0.02 | 5 | 40301 |

**Further Improvement**

In the gradient descent model we have utilized so far, we fixed the step size, the regularization term, and the total number of iterations. Later on, we will try different step sizes and regularization terms, and then pick the optimal value with minimum loss. The running time for every epoch is slow(approximately 10 seconds) for our current model. If we try different parameters, the running time might be even slower. If we choose to reduce our matrix’s dimensionality, can we solve this problem without affecting the model accuracy?

Another approach to solve the equation



is alternating least squares (ALS). This method functions by taking turns of fixing u and v and optimizing the other one by solving the least squares problem until convergence is found. We can compare the results from ALS with the results from SGD. Besides that, what other methods do you recommend us to try?