Recommender System via Matrix Factorization and Ensemble Modeling

Abstract

In this paper, we attempt to construct a recommendation system to promote users with new movies that are likely to match their tastes. The two methods discussed in this paper are first, matrix factorization, and secondly, ensemble modeling on top of matrix factorization. In other words, the predicted results from matrix factorization become one of the features for the second stage model. We discover that the ensemble approach is able to decrease the mean absolute error (MAE) of our predictions by a noticeable margin compared the naive matrix factorization approach. One major challenge that we face throughout the experimentation is that given a large dataset, both our matrix factorization algorithm and second stage machine learning models require a long time to execute. Therefore, future work can be focused on running the models with better computation resources, for example, on GPU.

1 Introduction

The recommender problems have been extensively studied in a variety of settings, for example, which restaurants should Yelp recommends to its users, or what movies should Netflix promote on this front page for different users. A traditional machine learning prediction model without sophisticated feature engineering may easily fail this kind of task because often case, besides the ratings that a user have given to the contents (e.g. restaurants, movies) in the past, there does not exist much other information that can be studied by the machine learning model.

So, how could our model recommend movies to users if all we have are the past ratings given by the users? One approach known as collaborative filtering considers all the users at once rather than individually and tries to understand a particular user by leveraging experience of other users. From this definition, a very intuitive way to perform collaborative filtering is through the nearest-neighbor prediction. More specifically, if we want to predict *user a*'s rating on a new *movie i*, we can take the average rating (or most frequent rating) of *user a*'s most similar neighbors who have rated *movie i* in the past. While this method can work very well when the movie preferences are straightforward, it does not extend to the situation that the users have mixed interests in movies. Hence, another collaborative filtering method called matrix factorization attempts to formulating the task of uncovering the full underlying rating matrix Y (see *Figure 1*) as a matrix problem. More specifically, the rating matrix Y can be written as a product of two low-rank matrices, in which the first matrix can represent users' preferences to each (latent) movie categories and the second matrix can represent movies' (latent) categories. Then, by multiplying the two matrices together, we can complete the entire rating matrix and minimize the prediction errors. Since the latter method usually performs the best in complicated recommender problems, this paper will be mainly focusing on matrix factorization and extension of matrix factorization.

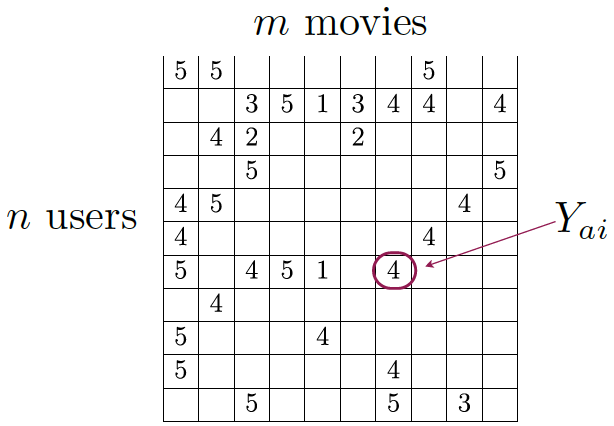


Figure 1: Collaborative Filtering Illustration

2 Dataset and Features

2.1 Dataset

The dataset for this paper is collected from the MovieLens website by GroupLens Research. Since we do not have access to more computation resources, we choose the smaller dataset version that contains 100,836 ratings across 9,742 movies. These data were created by 610 users between March 29th, 1996 and September 24th, 2018. Each rating ranges from 0.5 stars to 5 stars with an increment of half stars. In addition, we further reduce the scope of the dataset by dropping all movies with less than 20 ratings. As a result, the modified dataset contains 67,898 ratings across 1,297 movies created by 610 users. In this dataset, each user and movie is represented by a unique ID, which can be used to join with other tables, such as a movie table that contains the information about movie titles and genres.

2.2 Featurization

Since the recommender problems usually do not have much data besides the ratings, we face a similar issue in our case. The major areas of data preprocessing are as follows:

- Transform a *pandas* dataframe to a *numpy* 2d-array via the *pivot* function in Python.
- (For ensemble model only) Extract movie's release year in the movie title and categorize the release year into 4 time period (before 1980; 1980 - 2000; 2000 - 2010; after 2010) through one-hot-encoding.
- (For ensemble model only) Extract movie's genres and perform hot-encoding via *stack* and *get_dummies* function in Python. Note that a movie may have multiple genres.

3 Modeling

3.1 Matrix Factorization

In this paper, we implement the matrix factorization algorithm from scratch. Let $Y \in R^{n \times m}$ be the observed matrix, and $X \in R^{n \times m}$ be the predicted matrix, we will write $X = UV^T$ such that $U \in R^{n \times k}$, $U = [u^{(1)}, \ldots, u^{(n)}]^T$ and $V \in R^{m \times k}$, $V = [v^{(1)}, \ldots, v^{(m)}]^T$. Here, k represents the rank of X, and it is generally a small number because we want to perform low-rank factorization. Our goal is to minimize the total difference between Y_{ai} and $X_{ai} = [UV^T]_{ai}$ for $(a, i) \in D$, where D represents the set of indices of observations. Hence, we are going to minimize the following loss function with respect to U and V:

$$J(U,V) = \sum_{(a,i)\in D} \left(Y_{ai} - \left[UV^T\right]_{ai}\right)^2 /2 + \frac{\lambda}{2}\sum_{a=1}^{n}\sum_{j=1}^{k} U_{aj}^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{j=1}^{k} V_{ij}^2$$

However, fitting the matrices U and V at once will be usually time-consuming, we thus use a simpler alternative algorithm proposed in the lecture notes as follows:
(0) Initialize the movie feature vectors $v^{(1)}, \ldots, v^{(m)}$ randomly.
(1) Fix $v^{(1)}, \ldots, v^{(m)}$ and separately solve for each $u^{(a)}, a = 1, \ldots, n$ by minimizing:

$$\sum_{i:(a,i)\in D} \left(Y_{ai} - u^{(a)} \cdot v^{(i)}\right)^2 /2 + \frac{\lambda}{2} \left\|u^{(a)}\right\|^2$$

(2) Fix $u^{(1)}, \ldots, u^{(n)}$ and separately solve for each $v^{(i)}, i = 1, \ldots, m$, by minimizing:

$$\sum_{a:(a,i)\in D} \left(Y_{ai} - u^{(a)} \cdot v^{(i)}\right)^2 /2 + \frac{\lambda}{2} \left\|v^{(i)}\right\|^2$$

Step (1) and (2) are performed iteratively until either we hit the maximum number of iterations or the matrices U and V converge. At each minimization step, we use the optimizer from *SciPy* to find the parameters that minimize the loss function.

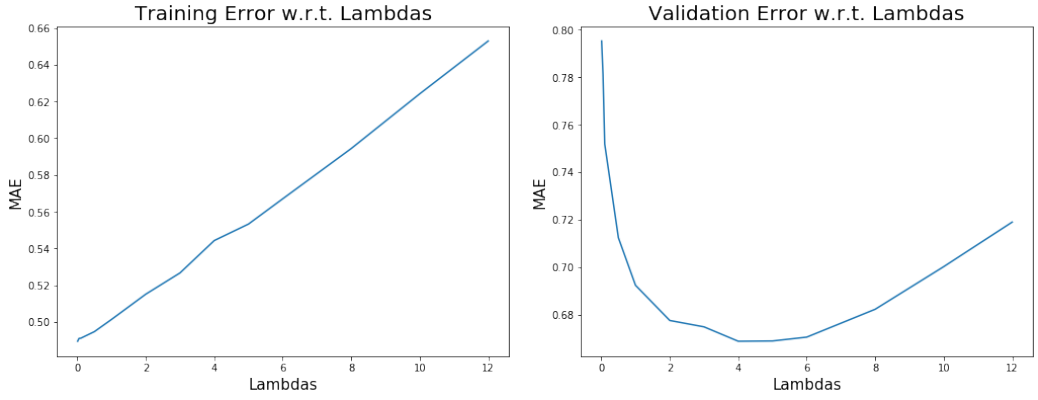
3.2 Ensemble Model

To capture more subtleties, we build an ensemble model that takes the prediction of the movie ratings as one of the features and add other movie-related features that we can obtain, such as movie's release year, genres (feature engineering of these features is discussed in the *Featurization* section). Then we build a random forest regressor to predict the movie ratings again. To obtain the best set of hyperparameters for random forest, we perform a random grid search and three-fold cross-validation on number of trees, maximum depth, minimal sample leafs.

4 Result and Discussion

Given limited computation resources, we first manually tune the rank k and the maximum number of iterations before convergence. From limited experimentation, we set the rank to be 5 and number of iterations to be 10, which allow us to finish updating each iteration of U and V in about 2 minutes. Then we perform cross-validation on our regularization coefficient λ, and as we can observe from *Figure 2*, in training set, without a surprise, the mean absolute error (MAE) of our prediction increases as we penalize more on the complexity of the models. However, in validation set, we see the lowest MAE occurs when we set our λ to be 4. Hence, we will be using this regularization coefficient to train our final model of matrix factorization as well as the ensemble model.

Figure 2: Train and Validation Errors with respect to Regularization Coefficients



(a) Training Error

(b) Validation Error

Then we want to compare the prediction results of the naive matrix factorization model and the ensemble model to see if adding a second stage modeling can help our prediction performances. As we can observe from the below table, the ensemble method outperforms the naive matrix factorization

in both training and testing phases. In particular, the ensemble model has a 1.9% decrease in training MAE and a 16% decrease in testing MAE, which means that the ensemble model overcomes the overfitting issue and can generalize our data's behavior better. Given that our average rating to be 3.63 stars, an error of 0.54 stars shows that our model can learn the behaviors behind ratings to some extent.

	Training MAE	Testing MAE
Matrix Factorization	0.5448	0.6517
Ensemble Method	0.5347	0.5472

5 Conclusion

In this paper, we have seen that using matrix factorization to predict users' ratings (and thus preferences) on movies gives us a descent predictive performance and the algorithm is tractable for a dataset that has about 68k ratings across 1.3k movies and 610 users. Moreover, we see that an ensemble model that incorporates additional features can boost the performance by a noticeable margin. Future work of this paper can be focused on first, gathering more computation resources to run the algorithm on a larger dataset and perform more in-depth cross-validation on all hyperparameters, and secondly, incorporating more features, especially on users so that the second stage model can capture more complicated relationships between users and movies.

References

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872

[2] Jaakkola, Tommi, and Regina Barzilay. 2016. "Introduction to Machine Learning Draft Notes by Topic."