Project Report

Statistical Learning Theory LAB

Maurice Wenig
Friedrich Schiller University Jena
Germany
maurice.wenig@uni-jena.de

1 INTRODUCTION

The context of this project are rating systems. Users can rate specific items with a score from 0 to 4. The goal is then, to predict the rating of an item for a specific user. To achieve this, we are given training data, that specifies how users rated specific items. This data can be turned into the rating matrix [1, Chapter 1, Section 3], where the items form the columns, and the users form the rows. This rating matrix is naturally sparse, as can be seen in Figure 1, where white pixels are missing data points.

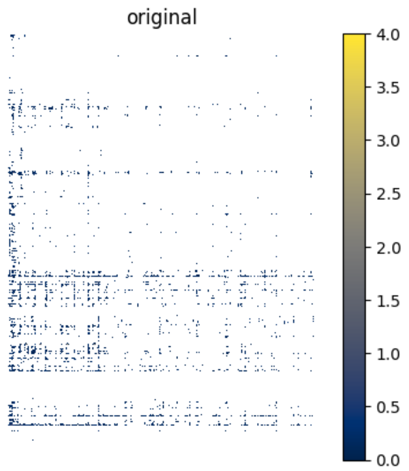

Figure 1: Sparse rating matrix

The user and item ratings also have a long tail, meaning that there are only a few users who rated a lot of items, and a lot of users, who rated very few items. The long tail effect for items is not as strong as it is for users in the training data. This can be observed in Figure 2, where users and items have been sorted by number of ratings.

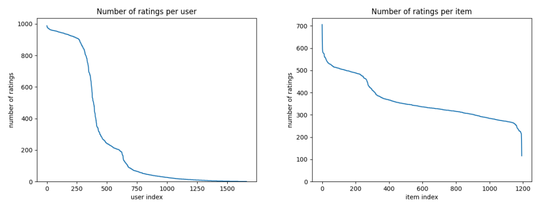

Figure 2: Long tail of user and item ratings

2 METHODS

2.1 Neighbourhood-Based Methods

2.1.1 User-Based Recommender. The user-based recommender compares users based on their rated items. An item rating from a user is then predicted based on ratings of that item from similar users. The similarity measure used to compare users is Pearson:

$$\mathrm{Sim}(u,v) = \frac{\sum_{i \in I_{uv}} w_i \cdot s_{ui} s_{vi}}{\sqrt{\sum_{i \in I_{uv}} w_i \cdot s_{ui}} \cdot \sqrt{\sum_{i \in I_{uv}} w_i \cdot z_{vi}}}$$

where I_{uv} are the items that both users u and v have rated, $s_{ui} = r_{ui} - \mu_u$ is the centered rating from user u. The mean rating μ_u can be calculated individually for every comparison, based on only the common items, or it can be calculated once for every user, based on all their rated items. $w_i = \log \frac{\text{\# users}}{\text{\# users who rated } i}$ are item weights to combat the impact of the long tail in the number of item ratings.

To prefer users with more common items, the similarites of users with less than β common items are made smaller.

$$\mathrm{Sim}(u,v) \leftarrow \mathrm{Sim}(u,v) \cdot \frac{\min\{|I_{uv}|, \beta\}}{\beta}$$

The hyper parameter α can be used to amplify the importance of similarity.

$$\mathrm{Sim}(u,v) \leftarrow \mathrm{Sim}(u,v)^{\alpha}$$

Based on these similarities, a peer group $P_u(i)$ of users is determined. For this, the users are sorted by similarity. To improve efficiency in the online phase, the sorting is done in the offline phase. For this, the similarities between every user u, v are saved in the similarity matrix. The order of peers for a specific user is then determined by sorting the respective row vector.

In the online phase, this order is simply applied. Users are then removed, if they have not rated them item i, or their similarity to user u is below a certain threshold S_{min}. The top k remaining users are then the peer group $P_u(i)$.

The predicted item rating from a user is then calculated as a sum of ratings from similar users, weighted with their relative similarity. The means and variances of the user ratings are normalized to not influence the prediction.

$$\hat{r}_{ui} = \mu_u + \sigma_u \cdot \frac{\sum_{v \in P_u(i)} \mathrm{Sim}(u,v) \cdot z_{vi}}{\sum_{v \in P_u(i)} |\mathrm{Sim}(u,v)|}$$

where $z_{ui} = s_{ui}/\sigma_u$ is the normalized rating, σ_u the variance of the ratings of user u. All of this and the reasoning behind it is described in [1, Chapter 2, Section 3.1].

If the user has only ever given one distinct rating, the variance is set to an arbitrary value of 1.

In special cases where the user or the item has never been seen before, similarities can not be calculated. In those special cases, the ratings were predicted in a different way:

- user and item unknown: predict average between minimum and maximum possible rating
- user unknown, item known: predict average rating of that item
- user known, item unknown: predict average rating of that user

Also, if the original item rating of the user is known, this original rating can be predicted.

The best performing hyper parameters were the following:

- $k = 50$ neighbours in peer group
- $\alpha = 1$, not amplifying importance of similarity
- $S_{min} = 0.4$ minimum similarity
- $\beta = 4$ common dimensions before similarity reduction

2.1.2 Item-Based Recommender. The item-based recommender compares items based on their user ratings. An item rating from a user is then predicted based on ratings of similar items from that user. It functions very similar to the user-based recommender. The similarity function is adapted to still use user means μ_u instead of item means μ_i to normalize user preferences. The predicted rating is then a weighted sum of similar items, the user has rated [1, Chapter 2, Section 3.2].

$$\hat{r}_{ui} = \frac{\sum_{j \in P_i(u)} \text{Sim}(i,j) \cdot r_{uj}}{\sum_{j \in P_i(u)} |\text{Sim}(i,j)|}$$

The best performing hyper parameters were the following:

- $k = 50$ neighbours in peer group
- $\alpha = 1$, not amplifying importance of similarity
- $S_{min} = 0.6$ minimum similarity
- $\beta = 6$ common dimensions before similarity reduction

2.1.3 Clustering. To increase efficiency, the users/items can be clustered before the creation of the similarity matrix. This way, only users/items that are in the same cluster have to be compared.

This is done using an adapted version of K-means [1, Chapter 2, Section 4]. To account for the sparsity of the rankings matrix, the entries of the mean of a cluster are calculated using only vectors, where the entries are not missing in the respective dimensions. The distance measure is also adapted. The distance between two vectors is calculated only with the common dimensions, where ratings are not missing. The result is then divided by the number of common dimensions to get a distance measure that is independent of the number of common dimensions.

The number of clusters was tuned to get the best trade-off of performance and prediction quality. This resulted in clustering with $K = 8$ for the user-based recommender. Such a good trade-off could not be found for the item-based recommender.

2.2 Factorization-Based Recommender

In the factorization-based recommender, the users and items are assumed to be in a common latent factor space R^f of user preferences q_i and item characteristics p_u. The rating r_{ui} is then modelled as the user's interest in the item's characteristics. User and item biases are also taken into account.

$$\hat{r}_{ui} = q_i^\top p_u + \mu + b_i + b'_u$$

The goal is then to learn the factors q_i, p_u, such that the squared error function on the ratings of the training set is minimal.

$$\min_{p,q,b} \sum_{(u,i)\in\kappa} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 + b_i^2 + {b'_u}^2 \right)$$

with κ as the set of (u,i) pairs in the training set, λ as the regularization parameter. This model, its loss function, and ideas for solving this minimization problem are described in [3].

To achieve this minimzation, alternating least squares is used. q, p are initiated randomly. Then, in each step, one of them is fixed and the other one is solved optimally. This is done until the change of the vectors in q, p is below a threshold ϵ or a maximum number of iterations is reached.

To optimally solve the minimization at each step, ridge regression is used [2]. Because they are independent, each of the summands are optimized individually.

$$\min_{p_u, b'_u} \sum_{i\in\kappa_u} \left((r_{ui} - b_i - \mu) - (q_i^\top p_u + b'_u) \right)^2 + \lambda \left(\|p_u\|^2 + {b'_u}^2 \right)$$
$$\min_{q_i, b_i} \sum_{u\in\kappa_i} \left((r_{ui} - b'_u - \mu) - (q_i^\top p_u + b_i) \right)^2 + \lambda \left(\|q_i\|^2 + b_i^2 \right)$$

with κ_u as the set of items that have been rated by u, κ_i as the set of user that have rated i.

The ridge regression the boils down to solving the regularized normal equations:

$$\left(Q_{\kappa_u} {Q_{\kappa_u}}^\top + \lambda \mathbb{1} \right) p_u = Q_{\kappa_u} y_{\kappa_u}$$
$$\left(P_{\kappa_i} {P_{\kappa_i}}^\top + \lambda \mathbb{1} \right) q_i = P_{\kappa_i} y_{\kappa_i}$$

with

$$\begin{aligned} Q_{\kappa_u} &= (q_{i_1}, \ldots, q_{i_n}) \\ y_{\kappa_u} &= \vec{r}_{\kappa_u} - \vec{b} \\ &= (r_{ui_1}, \ldots, r_{ui_n})^\top - (b_{i_1}, \ldots, b_{i_n}) \\ i, \ldots, i_n &\in \kappa_u \end{aligned}$$

for determining p_u and

$$\begin{aligned} P_{\kappa_i} &= (p_{u_1}, \ldots, p_{u_n}) \\ y_{\kappa_i} &= \vec{r}_{\kappa_i} - \vec{b'} \\ &= (r_{u_1 i}, \ldots, r_{u_n i})^\top - (b'_{u_1}, \ldots, b'_{u_n}) \\ u_1, \ldots, u_n &\in \kappa_i \end{aligned}$$

for determining q_i.

The biases can either be fixed or part of the optimization. If the biases are fixed, μ is the overall mean rating in the training set, $b'_u = \mu_u - \mu$ is the mean user rating centered around the overall mean rating, $b_i = \mu_i - \mu$ is the mean item rating centered around the overall mean rating. If the biases are part of the optimization, they are the affine part of the ridge regression at each step. This affine part can be obtained by appending a constant 1 to the training vectors and reading its respective coefficient.

The special cases are handled in the same way as they are in the User Based Recommender.

The best performing hyper parameters were the following:

- $\lambda = 5$ regularization parameter
- $f = 20$ latent dimensions
- biases not fixed
- $\epsilon = 10^{-2}$, max 20 iterations

2.3 Hybrid Recommender

The hybrid combines an array of K recommenders. The predictions are a linear combination of the predictions of those recommenders.

$$\hat{r}_{ui} = \sum_{k=1}^{K} \alpha_k \hat{r}_{ui}^{(k)} = \vec{\alpha}^\top \vec{r}_{ui}$$

To learn $\vec{\alpha}$, the training set is separated into two parts: a fitting set, and a held out set. The recommenders are then trained on the fitting set. Then, the recommenders predict the labels of the held out set. The optimal alpha can either minimize the mean absolute error

$$\mathcal{E}_1 = \frac{1}{m} \left\| R - \vec{R}\vec{\alpha} \right\|_1$$

or the (half) mean squared error (MSE)

$$\mathcal{E}_2 = \frac{1}{2m} \left\| R - \vec{R}\vec{\alpha} \right\|_2^2$$

m as the number of samples in the held out set, $R = (r_1, \dots, r_m)^\top$ as the vector of true ratings of the held out samples, $\vec{R} = (\vec{r}_1, \dots, \vec{r}_m)^\top$ as the matrix of predictions on the held out samples.

$\vec{\alpha}$ is then optimized with gradient descent. Every entry is intialized with $\frac{1}{K}$. At each step, $\vec{\alpha} \leftarrow \vec{\alpha} - \gamma \nabla \mathcal{E}(\vec{\alpha})$ is pushed into the direction of the negative gradient of the error, scaled with a learning rate γ. This is done until the error changes less than a threshold ϵ, or the maxmimum amount of iterations is reached.

$$\nabla \mathcal{E}_1(\vec{\alpha}) = \frac{1}{m} \operatorname{sign}\left(R - \vec{R}\vec{\alpha}\right)^\top \vec{R}$$
$$\nabla \mathcal{E}_2(\vec{\alpha}) = \frac{1}{m} \left(R - \vec{R}\vec{\alpha}\right)^\top \vec{R}$$

This model, its error functions, and the gradient descent method are described in [1, Chapter 6, Section 3].

After $\vec{\alpha}$ has been determined, the predictors are trained on the whole training set to not miss out on information. Special cases do not need to be handled, as they are handled in the individual recommenders.

The best performing hyper parameters were the following:

- hybrid of clustered user-based recommender, unclustered item-based recommender, and factorization-based recommender
- optimize for MSE
- $\gamma = 1$ learning rate
- $\epsilon = 10^{-4}$, max 100 iterations

It is worthy to note that the hybrid didn't improve much compared to the initial equal weights. In Figure 3 you can see mean absolute error (MAE) of the hybrid recommender during the gradient descent of $\vec{\alpha}$.

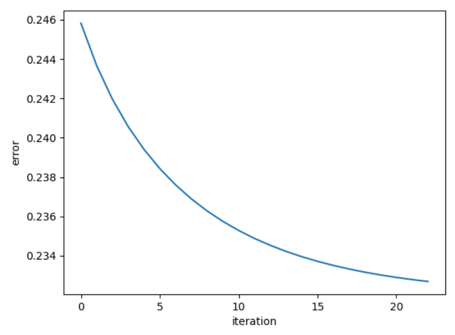

Figure 3: Hybrid error during gradient descent

3 RESULTS

3.1 Performance

For time measurement, the recommenders were trained on `train.csv` (435K samples) and evaluated on `qualifying_blanc.csv` (109K samples).

Table 1: Run Time Comparison

Recommender	offline phase	online phase
user based	15.97s	6.84s
item based	44.3s	4.79s
factorization	85.65s	0.53s
hybrid	304.1s	26.79s

3.2 Error Scores

The error scores were measured with cross validation on the training set, with $R = 8$ rotations. Root mean squared error (RMSE) and MAE were calculated.

Table 2: Error Score Comparison

Recommender	RMSE	MAE
user based	0.670	0.301
item based	0.569	0.222
factorization	0.512	0.207
hybrid	0.496	0.193

3.3 Full Rating Ratrices

The recommenders can be used to fill the rest of the rating matrix with predictions. The resulting structures are visualized in Figure 4.

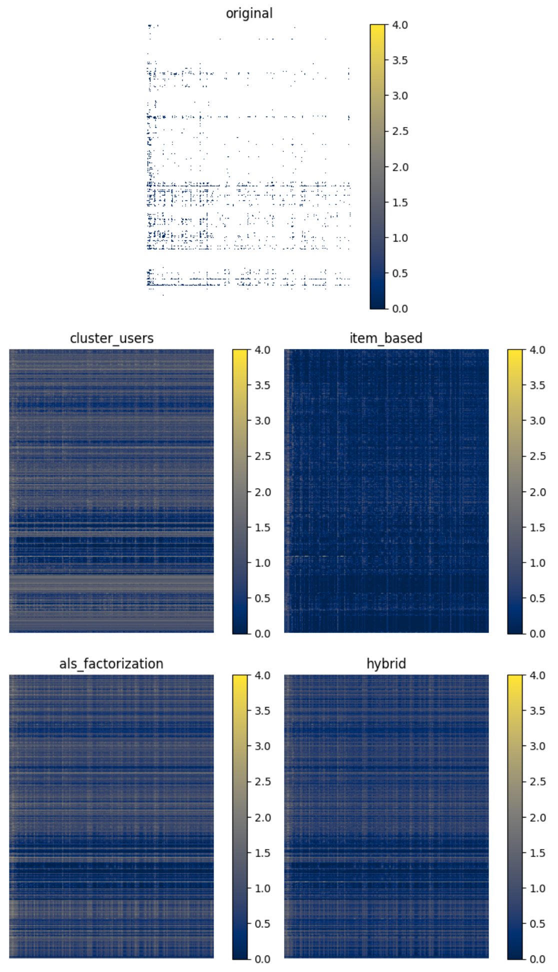

Figure 4: Rating matrix filled by recommenders

4 CONCLUSION

I successfully implemented recommender systems, that predict item ratings for specific users. My recommenders are slow, but the predictions are of high quality. This can be seen on the validation set, where my hybrid recommender is the best scoring recommender at the time of writing, with a score of 0.530.

REFERENCES

[1] Charu C. Aggarwal. 2016. *Recommender Systems*. Springer International Publishing. https://doi.org/10.1007/978-3-319-29659-3
[2] Arthur E. Hoerl and Robert W. Kennard. 1970. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* 12, 1 (feb 1970), 55–67. https://doi.org/10.1080/00401706.1970.10488634
[3] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (aug 2009), 30–37. https://doi.org/10.1109/mc.2009.263