Summary for the RecSys 2017 Online Ranking Tutorial

1 Rating and ranking prediction

A common definition for the recommendation task is the *rating prediction* problem. For a given user u and a given item i the system should return a predicted relevance $\hat{r}_{ui}$. The actual preference of the user on the item is r_{ui}. While the rating prediction task is highly popular, the top-K *ranking prediction* problem is closer to the actual industrial use-cases. In this setting, for a given user u the recommender should retrieve an ordered set of relevant items $\{i_1, i_2, ..., i_K\}$, where the number of returned items is K.

2 Classes of recommendation algorithms

- **Content-based** recommenders use user profiles and item content for recommendation, for example the description of a product, or information on the user in a social network.
- **Collaborative filtering** (CF) methods use historical data that contain the past interactions of the users and the items logged by the service provider. One type of CF algorithms are matrix factorization (MF) models that map the users and the items to a lower k dimensional space.
- **Context-aware** recommenders not only investigate user-item interactions, but also their conditions. Contextual information mainly describes the user and in general user-item interactions, e.g. the updated location information of the user.

3 Explicit and implicit recommendation

Recommendation scenarios can differ based on the type of data available for training. Collaborative filtering methods learn by investigating the interactions between the users and the items. When the data contains ratings e.g. on a scale between 1-5, the problem is an explicit recommendation task. In case of implicit feedback, there is no exact information about the user preferences, e.g. only the click history of the user is known.

4 Matrix factorization

Matrix factorization models [2] are collaborative filtering methods that use the past interactions between the users and the items. Those can be mapped to a user-item matrix, the utility matrix R (see Fig. 1). Each row of R corresponds to a given user, and each column corresponds to a given item. In case of explicit feedback, r_{ui} is the value of the rating given by user u for item i. For implicit feedback, r_{ui} is equal to 1 if user u previously interacted (clicked, viewed, ...) item i.

Note that R is a sparse matrix, and most of its elements are unknown. The task of the recommender algorithm is to predict correctly the unknown part of the matrix. To achieve this, MF models decompose R into two dense matrices P and Q. For a given user u, the corresponding row in P is the user vector p_u. For item i, the corresponding column of Q is the item vector q_i. The predicted relevance of item i for user u is then

$$\hat{r}_{ui} = p_u q_i^T. \tag{1}$$

4.1 Optimization by stochastic gradient descent

We introduce an algorithm that learns the model parameter matrices P and Q [2]. For the utility matrix R that contains a set of ratings T, we intend to minimize the L2 norm of the difference between

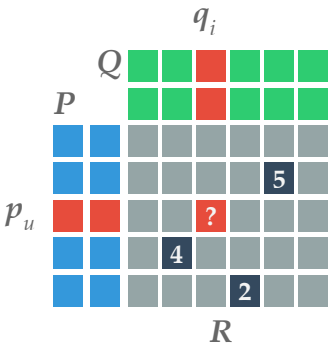

Figure 1: The utility matrix R and the matrix factorization model built from matrix P and Q.

predicted relevances and the actual ratings,

$$F = \sum_{r_{ui} \in \mathrm{T}} \left(r_{ui} - \hat{r}_{ui} \right)^2, \tag{2}$$

i.e. the objective function that we optimize for is the mean squared error (MSE) on the training set. For searching the minimum, we use stochastic gradient descent (SGD) that iterates several times on the known user-item ratings in the training set. For a given (u, i), the steps of the algorithm are the following.

- We compute the gradient of the objective function F with respect to the model parameters,

$$\frac{\partial F}{\partial p_u} = -2(r_{ui} - \hat{r}_{ui}) q_i, \qquad \frac{\partial F}{\partial q_i} = -2(r_{ui} - \hat{r}_{ui}) p_u. \tag{3}$$

- Then we take a step and update the model parameters towards the direction opposite to the gradient. The step is proportional to the learning rate η,

$$p_u \leftarrow \eta(r_{ui} - \hat{r}_{ui}) q_i, \qquad q_i \leftarrow \eta(r_{ui} - \hat{r}_{ui}) p_u. \tag{4}$$

4.2 Variants of matrix factorization

Learning algorithms can vary by the their model, objective function, and their optimization algorithm. Previously we learned a matrix factorization model by applying SGD to minimize the MSE in equation (2). Next we list some popular modifications of MF models.

L2 regularization: A general way to avoid overfitting is to modify the objective function with a regularization term [2],

$$\tilde{F} = F + F_{\mathrm{reg}} = \sum_{r_{ui} \in \mathrm{T}} \left(r_{ui} - \hat{r}_{ui} \right)^2 + \lambda \left[|p_u|^2 + |q_i|^2 \right]. \tag{5}$$

The modified update rules are then

$$p_u \leftarrow \eta(r_{ui} - \hat{r}_{ui}) q_i - \eta\lambda p_u, \qquad q_i \leftarrow \eta(r_{ui} - \hat{r}_{ui}) p_u - \eta\lambda q_i. \tag{6}$$

Bias terms: One can modify the model by introducing scalar terms that describe the biased behavior of the users and the items [2],

$$\hat{r}_{ui} = b_u + b_i + p_u q_i^T, \tag{7}$$

where b_u is the user bias term and b_i is the item bias term. Both parameters can be fit by SGD.

Negative sampling: In case of implicit feedback, the known part of the utility matrix only contains constant 1 elements. To fit a model, one has to assume that at least part of the unknown part is 0, i.e. for some unknown elements, the user has not interacted with the items because she did not prefer them. In case of SGD, we sample C times more samples than positive ones from the unknown elements uniformly and learn $r_{ui} = 0$ for those "negative samples".

Confidence values: In an implicit ratings prediction task, one can introduce lower confidence values for the unknown negative events [1]. In contrast to negative sampling, we include all user-item pairs in the objective function, but with different confidence c_{ui} as

$$\tilde{F}' = \sum_{u,i} c_{ui} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left[\sum_u |p_u|^2 + \sum_i |q_i|^2\right], \tag{8}$$

where we set $c_{ui} = c > 1$ for user-item pairs where $r_{ui} > 0$ (e.g. $c := 40$), and $c_{ui} := 1$ when $r_{ui} = 0$.

ALS: The alternating least squares method [2, 4] (ALS) is another preferred algorithm besides SGD. The concept of ALS is that for a fixed Q it computes the best P i.e. P that minimizes the objective function. Then for a fixed P it calculates the best Q. We continue this iteration until we get satisfactory objective values. More specifically, we use the objective function in (5). When we fix Q, this defines a separate optimization problem for each user u. The objective for a particular user u is then

$$\tilde{F}_u = \sum_{i:(u,i)\in \mathrm{T}} 2\left(r_{ui} - p_u q_i^T\right)^2 + \lambda |p_u|^2. \tag{9}$$

Next we introduce some notations for user u,

- $S_u = \{r_{ui_1}, r_{ui_2}, ..., r_{ui_s}\}$ is the set of the items rated by u, $s := |S_u|$,
- $b_u := [r_{ui_1}, r_{ui_2}, ..., r_{ui_s}]$ is the rating vector of u,
- $Q_u := \left[q_{i_1}^T, q_{i_2}^T, ..., q_{i_s}^T\right]$ is the matrix constructed from the latent vectors of the items rated by u.

Equation (9) can be rewritten as

$$\tilde{F}_u = |b_u - p_u Q_u|^2 + \lambda |p_u|^2. \tag{10}$$

The optimal p_u can be found by solving $\frac{\partial \tilde{F}_u}{\partial p_u} := 0$,

$$\frac{\partial \tilde{F}}{\partial p_u} = 2Q_u^T(p_u Q_u - b_u) + 2\lambda p_u = 0, \tag{11}$$

therefore the final solution for p_u is

$$p_u = (Q_u Q_u^T + \lambda I_s)^{-1} Q_u^T b_u, \tag{12}$$

where I_s is the s-dimensional identity matrix. Similarly, the update step for each item i is

$$q_i = (P_i P_i^T + \lambda I_s)^{-1} P_i^T b_i, \tag{13}$$

where the definitions of P_i and b_i are similar to the definitions of Q_u and b_u.

iALS: Koren et al. [1] proposed a variant of the original ALS algorithm for an implicit setting. We use the objective function with confidence values and regularization (8) that contains *all* elements in R. We apply a similar alternating algorithm for computing the user and the item vectors as in case of ALS. Equation (8) can be rewritten as

$$\tilde{F}' = \sum_u \left|\sqrt{C^u} r_u - \sqrt{C^u} Q p_u\right|^2 + \lambda \left[\sum_u |p_u|^2 + \sum_i |q_i|^2\right], \tag{14}$$

where

- vector $r_u := [r_{u1}, r_{u2}, ..., r_{uM}]$ contains the preferences of user u and M is the number of items,
- C_u is a $N \times N$ diagonal matrix containing the confidence values of u in the diagonal.

The optimal p_u can be set by solving $\frac{\partial \tilde{F}'}{\partial p_u} := 0$,

$$\frac{\partial \tilde{F}'}{\partial p_u} = 2\left[Q^T C^u Q p_u - Q^T C^u r_u\right] + 2\lambda p_u. \tag{15}$$

As for a fixed Q for the optimal p_u $\frac{\partial \tilde{F}'}{\partial p_u} = 0$,

$$p_u = \left[Q^T C^u Q + \lambda I\right]^{-1} Q^T C^u r_u. \tag{16}$$

Similarly for a fixed P, the optimal q_i is

$$q_i = \left[P^T C^i P + \lambda I\right]^{-1} P^T C^i r_i, \tag{17}$$

where vector r_i contains the preferences for item i, and C_i is a diagonal matrix containing the confidence values for i.

To sum up, we compute the user vectors and the item vectors in an alternating way based on (16) and (17). Note that a significant speedup can be achieved by rewriting these expressions to

$$p_u = \left[Q^T Q + Q^T [C^u - I] Q + \lambda I\right]^{-1} Q^T C^u r_u \text{ and } q_i = \left[P^T P + P^T [C^i - I] P + \lambda I\right]^{-1} P^T C^i r_i. \tag{18}$$

5 Evaluation metrics

A usual setting for evaluation is to divide the available data set into a training and an evaluation set. We learn the parameters of our model on the training set T, and evaluate its performance on a separated evaluation set E.

5.1 Explicit rating prediction

The metric used in the Netflix Prize competition is a natural choice: the mean squared error simply computes the squared difference of the predicted and actual ratings,

$$MSE = \sum_{(u,i) \in \mathrm{E}} (\hat{r}_{ui} - r_{ui})^2, \tag{19}$$

where the error is computed on the evaluation set of ratings E.

5.2 Implicit ranking prediction

In this setting the recommender should return an ordered top-K list of items L_u for a given user u. We compare the top list L_u in question against the items consumed by the user in the evaluation set E_u. Precision and recall are standard information retrieval metrics that can be applied for ranking evaluation. For a given user u,

$$\text{Precision@K}_u = \frac{|E_u \cap L_u|}{K}, \qquad \text{Recall@K}_u = \frac{|E_u \cap L_u|}{|E_u|}. \tag{20}$$

While precision@K and recall@K are widely used, they do not consider the rank of the relevant items. In contrast, mean average precision@K (MAE@K) depends on the ranking of the items in the list,

$$\text{MAE@K} = \frac{1}{K} \sum_{k=1}^{K} \text{precision@k}. \tag{21}$$

A more standard ranking based metric is discounted cumulative gain@K (DCG@K),

$$\text{DCG@K} = \sum_{k=1}^{K} \frac{\text{rel}(i_k)}{\log_2(1+k)} \tag{22}$$

where $\text{rel}(i_k)$ notes the relevance of the i-th item in the list. For the implicit task, the relevance is 1 if the user considered the item in the evaluation set, otherwise it is 0.

6 The online ranking prediction problem

In a time sensitive or online recommender that potentially re-trains the model after each and every new item, we have to generate a new top-k recommendation list for every single event in the evaluation period. In an online setting, as seen in Figure 2,

1. we query the recommender for a top-k recommendation for the active user,
2. we evaluate the list in question against the single relevant item that the user interacted with,
3. we allow the recommender to train on the revealed user-item interaction.

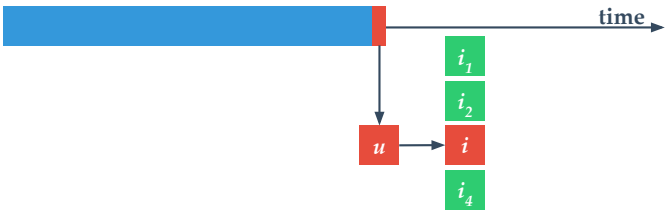

Figure 2: Temporal evaluation of the online ranking prediction problem.

7 Average DCG for online evaluation

We show that DCG computed *individually* for each event and averaged in time is an appropriate measure for real-time recommender evaluation. If i is the next consumed item by the user, the online DCG@K is defined as the following function of the rank of i returned by the recommender system,

$$\mathrm{DCG@K}(i) = \begin{cases} 0 & \text{if } \mathrm{rank}(i) > K; \\ \dfrac{1}{\log_2(\mathrm{rank}(i)+1)} & \text{otherwise.} \end{cases} \tag{23}$$

The overall evaluation of a model is the average of the DCG@K values over all events of the evaluation period. Furthermore, evaluating each event in the time series allows us to measure timely averages over the dataset, e.g. we may compute daily average DCG scores.

8 Online matrix factorization

As originally designed, stochastic gradient descent methods may iterate several times over the training set until convergence. In an real-time recommendation task, however, the model needs to be re-trained after each new event and hence re-iterations over the earlier parts of the data may be computationally infeasible. We may implement an online recommender algorithm by allowing a *single* iteration over the training data only when we process the events in *temporal* order.

Specifically for online matrix factorization we may set MSE as the objective function and we can apply online SGD for learning. In an implicit setting we may use the following sampling technique:

- When we update the model for the most recent positive record, we generate C random negative samples for the given user.
- After performing the update steps for the positive record, we process the negative samples with rating assigned to be 0.
- We exclude items that appeared before the time of the positive interaction when generating negative samples.

References

[1] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[3] Róbert Pálovics, András A Benczúr, Levente Kocsis, Tamás Kiss, and Erzsébet Frigó. Exploiting temporal influence in online recommendation. In *Proceedings of RecSys 2014*, pages 273–280. ACM, 2014.

[4] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 71–78. ACM, 2010.