ALS

Suppose Matrix M where rows depict users, the columns items and the values the rating of a user for an item. The intuition of matrix factorization is that there are some latent features that describe users and items that would allow the prediction of ratings given those features. In matrix factorization approaches we assume K such features for users and items which creates two latent feature matrices X (users) and Y(items) that can approximate the original M as:

$$M' = XY^T$$

These features can be seen also as coordinates in a k-dimensional space for each user/item. In that space similar users/items will be close together. Moreover, each rating is a product of a user feature vector with an item feature vector:

$$M_{ij} = X_i Y_j^T = \sum_{1}^{K} X_{ik} Y_{kj}$$

In order to compute X and Y, we need to solve the following optimization problem:

$$MIN_{X,Y}(|M - XY^T| + \lambda(|X|^2 + |Y^2|)$$

Where || is the norm of the matrix; the second factor in the expression above is referred as regularization and it limits the magnitude of X and Y to avoid very large values in the latent features which could in turn cause overfitting.

We can see that the problem requires minimizing two parameters. A common practical approach is to alternate between X and Y. In this manner we fix one of them to be a constant value and optimize the other. We repeat this alternating on which is going to be the constant and which is going to be the variable. By fixing one variable, the optimization problem becomes a liner regression problem (trying to fit a line on the ratings).

Suppose we fix X and we rename Y to β to indicate that is a parameter to be optimized. To simplify the problem more we ignore the regularization factor. Then minimizing:

$$MIN_{X,Y}(|M - X\beta|)$$

requires the derivative of the above expression to be equal to 0. Skipping the full proof, we can find that :

$$\beta = (X^T X)^{-1} X^T M$$

The full solutions -with regularization- for both items and users can be seen in the following formulas. We express them per user/item feature vector as such:

$$Y_j = (X^T X + \lambda I)^{-1} X^T M_j (1)$$

$$X_i = (Y^T Y + \lambda I)^{-1} Y^T M_i (2)$$

Where :

- M_i/M_j are the rows/columns corresponding to $user_i/item_j$ respectively
- I is the identity matrix (ones on the diagonal)

It is easy to see from the two expressions above the computation of each feature is depended on the matrix of the other feature matrix (the features of X are depended on Y etc) and each element is independent from all of the of the other elements of the same matrix. This means we can calculate each feature independently and in parallel given we have the other matrix.

But the expressions above take into account zero values equally to non-zero values. Practically this means we are optimizing the features so that they will predict the zero values as they are. In order to avoid that, we need to optimize the computations so that only the non-zero values are used in the computation of X_i/Y_j. We impose that rule by introducing a weigh matrix W where:

$$W_{ij} = 0 \; if \; M_{ij} = 0 \; , W_{ij} = 1 \; otherwise$$

Then the original minimization problem becomes:

$$MIN_{X,Y}(W|M - X\beta|)$$

And the final equations 1 and 2 are:

$$Y_j = (X^T W_j X + \lambda I)^{-1} X^T W_j M_j \; (3)$$

$$X_i = (Y^T W_i Y + \lambda I)^{-1} Y^T W_i M_i (4)$$

With the expression above, we can see how we can calculate Matrices X and Y with the following algorithm:

Function ALS(M,k):

While (not converged)

1. Initialize randomly X and Y so that each row will have k columns
2. Calculate X given Y for all users (equation 4)
3. Calculate Y given X for all items (equation 3)

Return X,Y

In the algorithm above we essentially take alternating steps where we take as granted the values of one matrix and try to optimize the other. The convergence condition referrers to how different the results are after each loop; so it can be a condition that checks whether the result of the multiplication of XY^T is different from the previous result under a specific threshold or we can replace it with a parameterized number of loops to run.

Improvements of the model

The area of collaborative filtering with NMF has a wide range of improvement techniques. Here we describe essentially the implementation of Spark [1] which:

- tries to optimize the task also for implicit rating matrices
- remove bias from the ratings

Implicit Ratings

Implicit ratings are ratings taken from the other actions of the user on the item (e.g. number of times viewed). In this case, usage patterns can affect the values of the rating matrix. Simply put, the frequency of actions varies for each user and for each item. Additionally, as actions might occur randomly (e.g. viewed an item by accident), we are also unsure about the confidence of the implicit

It becomes apparent that we need to:

1. Remove the bias from the ratings
2. Weight the ratings by a confidence factor [2]

The latter is relatively easy to adjust to the previous models by replacing the W matrix (of 1 and 0) with a confidence matrix (we will denote it with C) where the weight values can be decided based on some function. A common way to define the confidence factor is as such:

$$C_{ij} = 1 + \alpha M_{ij}$$

In order to remove the bias, we deal with each combination between users and items. This means that, during optimization, we need to take into account the bias of the user β_i and the bias of the item γ_j for each cell M_{ij}. We express the previous minimization problem with the bias in cell form:

$$Min_{X_i Y_j} \sum_{i,j} C_{ij}\left(M_{ij} - \beta_i - \gamma_j - X_i Y_j^T\right) + \lambda\left(\sum_i (|X_i| + \beta_i^2) + \sum_j \left(|Y_j| + \gamma_j^2\right)\right)$$

Equations 3 and 4 are then transformed to:

$$\tilde{Y}_j = (\tilde{X}^T C_j \tilde{X} + \lambda I)^{-1} \tilde{X}^T C_j M_j^{\beta} \text{ with } \tilde{X}_i^T = (1 \quad X_i^T) \text{ and } \tilde{Y}_j^T = (\gamma_j \quad Y_j^T)$$

And

$$\tilde{X}_i = (\tilde{Y}^T C_i \tilde{Y} + \lambda I)^{-1} \tilde{Y}^T C_i M_i^{\gamma} \text{ with } \tilde{X}_i^T = (\beta_i \quad X_i^T) \text{ and } \tilde{Y}_j^T = (1 \quad Y_j^T)$$

Where:

- $M_j^{\beta} = M_j - \beta$
- $M_i^{\gamma} = M_i - \gamma$

Matrix Factorization with Context

When we talk about context in collaborative filtering, we refer to properties (features) of users and items. We cannot include the features in the rating matrix as their values will dominate the rating values due to the fact that the latter are much sparser than the former ones. Given a matrix factorization model (here ALS) a popular approach was to filter the matrix based on the feature values.

The idea is to partition the data based on (categorical) values of the features[9,8,7]. The intuition is that different ratings appear for different contexts (items may take different ratings because of a feature value e.g. price range or users may have different ratings due to common features e.g. age). The filtering can happen by both users and items or only one of them. The strategy for partitioning the data is based on how different are the ratings under different values of the features (by t-test, chi-square test or any correlation criterion). For all contextual conditions (sets of values for features) we evaluate the correlation of the two contexts and decide based on threshold whether the two contexts have the same rating pattern or not.

Given the context of the task (Collaborative Filtering in Spark), a straightforward approach to improve predictions with minimal alteration to existing code would be to utilize the existing predictions of the ALS as a feature to a regression problem. In this task, we would build a feature matrix for every rating with the features of the user that rated and the features of the rated item:

X=[features of user, features of item, predicted rating], Y= actual rating

We can then utilize this new feature matrix in a regression task (e.g. with random forests) where we build a model that predicts Y based on X.

Other Matrix Factorization Models on Spark

Spark also contains the SVD++ algorithm [6] (but under the graphx library) which takes into account the Neighborhood of the user based on similarities among them. The minimization task looks similar to that of the ALS with additional factors to indicate the neighborhood.

.

References

[1] Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer, 42(8).

[2] Hu, Y., Koren, Y. and Volinsky, C., 2008, December. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (pp. 263-272). Ieee.

[3] Rendle, S., Gantner, Z., Freudenthaler, C. and Schmidt-Thieme, L., 2011, July. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (pp. 635-644). ACM

[4] Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S. and Schmidt-Thieme, L., 2010, December. Learning attribute-to-feature mappings for cold-start recommendations. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (pp. 176-185). IEEE.

[5] Steffen Rendle (2012): Factorization Machines with libFM, in ACM Trans. Intell. Syst. Technol., 3(3), May[6] Koren, Y., 2008, August. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 426-434). ACM

[7] L. Baltrunas and F. Ricci. Context-based splitting of item ratings in collaborative filtering. RecSys 2009

[8]L. Baltrunas and X. Amatriain. Towards time-dependent recommendation based on implicit feedback. RecSys 2009 Workshop on CARS

[9] A. Said, E. W. De Luca, and S. Albayrak. Inferring contextual user profiles – improving recommender performance. RecSys 2011 Workshop on CARS

[10] Guo, W., Wu, S., Wang, L. and Tan, T., 2016. Personalized ranking with pairwise Factorization Machines. *Neurocomputing*, *214*, pp.191-200.