Recommender Systems for Implicit Feedback: Collaborative Filtering augmented with Side-Information

C S Krishna, DataSaurus Rex

Problem Definition

A common use case is to generate product recommendations from an item catalogue to an extant user base.

In coming up with these recommendations, the algorithm must take into account:

- user purchase behavior
- user and item attributes.

In Collaborative Filtering (CF), the model directly learns from the purchase behavior of users to recommend additional items. It does so by analyzing the relationships between users and interdependencies among items to identify new user-item associations. When a user has exhibited sufficient purchase behavior, the CF based approach in general is superior to content based approaches that make predictions based on the user and item's observable features.

Within Collaborative Filtering, matrix factorization techniques have shown good performance [1]. In this approach, the model fits f-dimensional latent factor vectors x_u, y_i per each user and item based on the known ratings in the user-item ratings matrix. We can think of these latent factors as the user's underlying DNA that cannot be directly observed and only inferred from the user's purchases.

User\Item	**Item 1**	**Item 2**	**Item 3**	**Item 4**	**…**	**….**	**Item N-2**	**Item N-1**	**Item N**
User 1	1		1						
User 2									
User 3	1		1		1		1		
User 4									1
…									
…	1			1					
User M-2	1								
User M-1			1			1			
User M		1				1	1		

The latent factor vectors are be estimated by optimizing the following objective function:

$$\min_{\{x_u, y_i\}} \sum_{u,i} (p_{ui} - < x_u, y_i >)^2 + \lambda_x \sum_u \|x_u\|^2 + \lambda_y \sum_i \|y_i\|^2$$

where p_{ui} is the observed rating for a user indexed by *u* and an item indexed by *i*

After training a model, we can predict ratings for items that users haven't rated as per

$$r_{ui} = < x_u, y_i >$$

Alternating Least Squares (ALS): Matrix Factorization for Implicit Feedback datasets

In [2], the Matrix Factorization framework takes into account the implicit nature of feedback datasets. In such datasets, users don't give explicit ratings for products they have purchased. Rather, the purchase has to be interpreted as a 'like' signal whereas the non-purchase has to be interpreted as a 'dislike' signal. Therefore, in estimating latent factors, the model needs to take non-purchases into account as well which increases computational complexity. A key design feature is to give greater weightage to 'like' signals. Mathematically, a greater weight implies lesser uncertainty associated with the corresponding signal. This matches with our intuition that the dislike signal associated with a non-purchase comes with greater uncertainty since the user may not even have heard of the product.

The model estimates item and user specific latent factors by optimizing following objective function:

$$\min_{\{x_u, y_i\}} \sum_{u,i} c_{ui} (p_{ui} - < x_u, y_i >)^2 + \lambda_x \sum_u \|x_u\|^2 + \lambda_y \sum_i \|y_i\|^2$$

Here, p_{ui} is now a binary variable where '1' indicates the user has purchased the product. c_{ui} is the weight term which is greater whenever $p_{ui} = 1$.

Augmented Alternating Least Squares (Augmented-ALS):

The matrix factorization model's performance can be improved by incorporating user and item side information. A common approach to doing so is to fit a function that transforms the user/item's l-dimensional feature vector into the f-dimensional factor space. If the transformation function is linear, this formulation can be seen as a special case of factorization machines [4]. We can also learn non-linear transformations of the side information based on neural networks to augment the user and item specific latent factor vectors [5].

Incorporating use and item side information has many advantages. It helps us overcome the cold-start problem where a user has very few or zero purchases and a purely CF based approach based on purchase behavior would generate poor recommendations. Further, learning weights on the observable feature vectors capture global effects which can act as regularizers for learning user and item specific latent factors. This can improve model generalization.

In Augmented-ALS, we learn global weights for linear transformations of the observable user and item-feature vectors. We will refer to these weights as embedding parameters in the rest of the document. We also introduce user and item bias terms.

The objective function to be optimized can then be formulated as

$$\min_{\{\theta_{ui}\}} \sum_{u,i} c_{ui} (p_{ui} - \langle(\alpha_u G + x_u), (\beta_i D + y_i)\rangle - \langle g, \alpha_u \rangle - b(user)_u - \langle d, \beta_i \rangle - b(item)_i)^2$$

$$+\lambda_x \sum_u \|x_u\|^2 + \lambda_y \sum_i \|y_i\|^2 + \lambda_x \left(\sum_u b(user)_u^2\right) + \lambda_y \left(\sum_i b(item)_i^2\right)$$

$+\lambda_G \|G\|^2 + \lambda_D \|D\|^2 + \lambda_G \|g\|^2 + \lambda_D \|d\|^2$

We set up the following notation:

$\theta_{ui} := (x_u, y_i, b_u, b_i, G, D, g, d)$ refers to the set of trainable parameters required to estimate the propensity for a user-item pair.

u indexes the user and ranges from 1 to *M*, the total number of users.

i indexes the item and ranges from 1 to *N*, the total number of items.

We have the following trainable parameters:

$x_u, y_i \in \mathbb{R}^f$ the latent factor vectors for the user and the item respectively.

$b(user)_u,\ b(item)_i \in \mathbb{R}$, the user and item biases

$D \in \mathbb{R}^{l \times f}$, the embedding to transform item features into the f-dimensional factor space.

$d \in \mathbb{R}^l$, the embedding vector to compute the item features' contribution to its bias.

$g \in \mathbb{R}^k$, the embedding vector to compute the item features' contribution to its bias.

$G \in \mathbb{R}^{k \times f}$, the embedding matrix to transform item features into the f-dimensional factor space.

The data-set comprises:

$\alpha_u \in \mathbb{R}^k$, the observable feature vector for the user. This encodes the user's demographics, age and other observable characteristics.

$\beta_i \in \mathbb{R}^l$, the observable feature vector for the item.

p_{ui} denotes a user purchase (1 for purchase, 0 for non-purchase)

c_{ui} is a design parameter set to 41 when the user has purchased the item and 1 otherwise. It thus encodes the level of confidence in p_{ui} with the dislike signal associated with a non-purchase being down-weighted.

$\{\lambda_*\}$ are the set of *l2* regularization parameters that will be set through cross-validation.

Augmented ALS: Model Training

For training model parameters, we follow the co-ordinate ascent algorithm where during a training iteration, we fix a block of parameters and optimizing the objective function with respect to the set of parameters that aren't fixed. When the item and embedding factors have been fixed, the model is linear with respect to the user factors and can be updated in close form.

Model training proceeds as follows:

- Randomly initialize all model trainable parameters
- Keeping all other parameters fixed, update $x_u, b(user)_u$ for each user (each user's parameters can be updated independently and this is a massively parallelizable step)
- Keeping all other parameters fixed, update $y_i, b(item)_i$ for each item (each item's parameters can be updated independently and this is a massively parallelizable step)

- Keeping all other parameters fixed, update G, g
- Keeping all other parameters fixed, update D, d

We outline the procedure updating user factors in the next section:

User Factors Update:

Definite the following terms:

$$Y \in \mathbb{R}^{N \times f} := \begin{bmatrix} y_1^T \\ \vdots \\ y_n^T \end{bmatrix}$$

$$B \in \mathbb{R}^{N \times l} := \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_n^T \end{bmatrix}$$

$$B' \in \mathbb{R}^{N \times f} := BD$$

$$Bias(user) \in \mathbb{R}^{M} := \begin{bmatrix} b(user)_1 \\ \vdots \\ b(user)_M \end{bmatrix}$$

$$Bias(item) \in \mathbb{R}^{N} := \begin{bmatrix} b(item)_1 \\ \vdots \\ b(item)_N \end{bmatrix}$$

Then,

$$Y' \in \mathbb{R}^{N \times (f+1)} := [(Y + B') \quad \mathbf{1}]$$

$p(u)\ is\ the\ (N \times 1)\ matrix\ encoding\ customer\ u's\ purchases\ of\ products$

$q(u) \in \mathbb{R}^{N \times 1} := p(u) - (Y + B')\alpha_u G - Bd - g\alpha_u - Bias(item)$ [use broadcasting to efficiently implement this whenever dimensions don't match]

$C^u\ is\ the\ (n \times n)\ diagonal\ matrix\ such\ that\ C^u_{ii} = c_{ui}$

Given these definitions, the update step for a user u's factors is given by:

$$\begin{bmatrix} x_u \\ b_u \end{bmatrix} = \left(Y'^T C^u Y' + \lambda_x I\right)^{-1} Y'^T C^u q(u)$$

We note that each user factor can be updated independently of the other user factors. Hence this step is massively parallelizable.

We can derive update steps for other model parameters similarly.

Model Prediction

After model training, the user-item propensity score is computed as:

$$\langle(\alpha_u G + x_u), (\beta_i D + y_i)\rangle +$$

$\langle g, \alpha_u \rangle + b(user)_u + \langle d, \beta_i \rangle + b(item)_i$

A higher propensity score indicates a greater likelihood of the user purchasing the item. These scores can be used to prioritize targets for various marketing interventions.

Model Evaluation Framework

To assess model performance, we define the evaluation metric, AUC, as follows:

Test Metric: AUC

For the set of users U and set of items I, define S as the set of user-item purchases:

$$S \subseteq U \times I$$

For every user, we randomly transfer one purchase from the original set to the test set. This results in the disjoint sets, S_{train} and S_{test}. The model is trained on S_{train} and used to make predictions on user-item tuples in S_{test}.

From the model's predictions, we then compute the AUC metric, defined in [3], as follows:

$$AUC = \frac{1}{|U|}\sum_u \frac{1}{|E(u)|}\sum_{(i,j)\in E(u)} \delta\left(\hat{r}_{ui} > \hat{r}_{uj}\right)$$

In this equation,

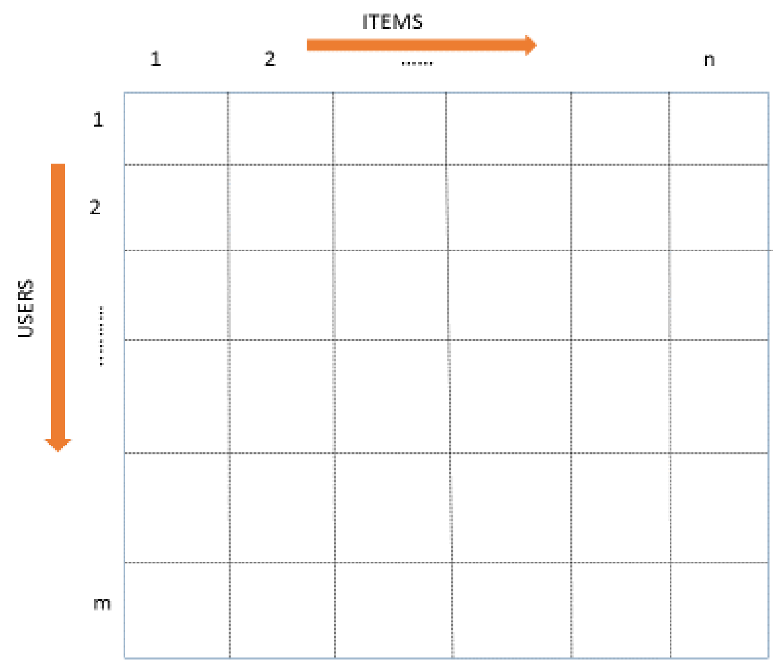
$$E(u) := \{(i,j) \mid (u,i) \in S_{test} \wedge (u,j) \notin (S_{test} \cup S_{train})\}$$

AUC of 0.5 indicates the model is no better than a random predictor and a value closer to 1 indicates perfect predictions.

Spark Implementation

The bottle neck computation in updating user/item factors are terms such as ${Y'}^T C^u Y'$ has a space/time complexity of $O(M)$, where M is the number of users/items.

To address this bottleneck, we partition the users and items among the C nodes of the cluster such that each node has M/m user factors, and N/n item factors. With this partitioning, any customer's factors will be present in exactly n nodes while any item's factors will be present in exactly m modes. Also note that $mn = C$



Given such a partitioning, in a given iteration, we can update in parallel at each node the customer's factors with respect to the item factors present in that node. This is followed by a shuffling step in which updates for every user across nodes are grouped together so that each user's final factor update can be computed. Then each of the user factors is propagated back to the m nodes it was present in. We repeat the procedure to update all the item factors.

The wall clock time complexity of one iteration is

$$k_1 Mn + k_2 \frac{M}{C} n + k_1 Nm + k_2 \frac{N}{C} m + overhead$$

subject to,

$$m \leq C$$

$$n \leq C$$

$$mn = C$$

The overhead term is a function of the number of factors, items and users which is fixed and can be ignored for the reminder of the analysis. The first and third terms capture the time complexity in the shuffling step (grouping factor updates from all the nodes and then propagating back) while the second and fourth terms capture time complexity of updating all the factors in a given node. k_1 is the proportionality constant for inter-node communication and will be orders of magnitude greater than k_2, the proportionality constant for processing a job within a node. Hence, the third and fourth terms can be ignored.

Then the optimal values of m, n such that total wall clock time per iteration is minimized are

$$m^* = min\left(C, \sqrt{\frac{MC}{N}}\right)$$

$$n^* = min\left(C, \sqrt{\frac{NC}{M}}\right)$$

For the common case of $M \gg N$, the optimal values are

$$m^* = C$$

$$n^* = 1$$

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems", IEEE Computer Society.

[2] Y.F. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," Proc. IEEE Int'l Conf. Data Mining (ICDM 08), IEEE CS Press, 2008, pp. 263-272.

[3] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian Personalized Ranking from Implicit Feedback", UAI, 2009.

[4] S. Rendle, "Factorization Machines", ACM Transactions on Intelligent Systems and Technology, Volume 3 Issue 3, 2012

[5] T. Bansal, D. Belanger, A. McCallum, "Ask the GRU: Multi-Task Learning for Deep Text Recommendations", arXiv:1609.02116v2 [stat.ML]