APEX 数据和知识管理实验室
DATA & KNOWLEDGE MANAGEMENT LAB

Feature-Based Matrix Factorization

Tianqi Chen, Zhao Zheng, Qiuxia Lu,
Weinan Zhang and Yong Yu

APEX Technical Report 2011-07-11(version 1.1)

APEX Data & Knowledge Management Lab

311, Yifu Building,
800, Dongchuan Road,
Shanghai, China, 200240.
www.apexlab.org

APEX Technical Report
APEX Technical Report 2011-07-11(version 1.1),

Feature-Based Matrix Factorization

Tianqi Chen, Zhao Zheng, Qiuxia Lu, Weinan Zhang, Yong Yu
{tqchen,zhengzhao,luqiuxia,wnzhang,yyu}@apex.sjtu.edu.cn
Apex Data & Knowledge Management Lab
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240 China
Project page: http://apex.sjtu.edu.cn/apex_wiki/svdfeature

Abstract. Recommender system has been more and more popular and widely used in many applications recently. The increasing information available, not only in quantities but also in types, leads to a big challenge for recommender system that how to leverage these rich information to get a better performance. Most traditional approaches try to design a specific model for each scenario, which demands great efforts in developing and modifying models. In this technical report, we describe our implementation of feature-based matrix factorization. This model is an abstract of many variants of matrix factorization models, and new types of information can be utilized by simply defining new features, without modifying any lines of code. Using the toolkit, we built the best single model reported on track 1 of KDDCup'11.

1 Introduction

Recommender systems that recommends items based on users interest has become more and more popular among many web sites. Collaborative Filtering(CF) techniques that behind the recommender system have been developed for many years and keep to be a hot area in both academic and industry aspects. Currently CF problems face two kinds of major challenges: how to handle large-scale dataset and how to leverage the rich information of data collected.

Traditional approaches to solve these problems is to design specific models for each problem, i.e writing code for each model, which demands great efforts in engineering. Matrix factorization(MF) technique is one of the most popular method of CF model, and extensive study has been made in different variants of matrix factorization model, such as [3][4] and [5]. However, we find that the majority of matrix factorization models share common patterns, which motivates us to put them together into one. We call this model feature-based matrix factorization. Moreover, we write a toolkit for solving the general feature-based matrix factorization problem, saving the efforts of engineering for detailed kinds of model. Using the toolkit, we get the best single model on track 1 of KDDCup'11[2].

This article serves as a technical report for our toolkit of feature-based matrix factorization[1]. We try to elaborate three problems in this report, i.e, what the model is, how can we use such kind of model, and additional discussion of issues in engineering and efficient computation.

2 What is feature based MF

In this section, we will describe the model of feature based matrix factorization, starting from the example of linear regression, and then going to the full definition of our model.

2.1 Start from linear regression

Let's start from the basic collaborative filtering models. The very baseline of collaborative filtering model may be the baseline models just considering the mean effect of user and item. See the following two models.

$$\hat{r}_{ui} = \mu + b_u \tag{1}$$

$$\hat{r}_{ui} = \mu + b_u + b_i \tag{2}$$

[1] http://apex.sjtu.edu.cn/apex_wiki/svdfeature

Here μ is a constant indicating the global mean value of rating. Equation 1 describe a model considering users' mean effect while Equation 2 denotes items' mean effect. A more complex model considering the neighborhood information[3] is as follows

$$\hat{r}_{ui} = \mu + b_i + b_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} s_{ij}(r_{uj} - \bar{b}_u) \tag{3}$$

Here $R(u)$ is the set of items user u rate, $\bar{b}_u$ is a user average rating pre-calculated. s_{ij} means the similarity parameter from i to j. s_{ij} is a parameter that we train from data instead of direct calculation using memory based methods. Note $\bar{b}_u$ is different from b_u since it's pre-calculated. This is a neighborhood model that takes the neighborhood effect of items into consideration.

Assuming we want to implement all three models, it seems to be wasting to write code for each of the model. If we compare those models, it is obvious that all the three models are special cases of linear regression problem described by Equation 4

$$y = \sum_i w_i x_i \tag{4}$$

Suppose we have n users, m items, and h total number of possible s_{ij} in equation 3. We can define the feature vector $x = [x_0, x_1, \cdots, x_{n+m+h}]$ for user item pair $< u, i >$ as follows

$$x_k = \begin{cases} Indicator(u == k) & k < n \\ Indicator(i == k - n) & n \leq k < n + m \\ 0 & k \geq m + n, j \notin R(u), s_{ij} \text{ means } w_k \\ |R(u)|^{-\frac{1}{2}}(r_{uj} - \bar{b}_u) & k \geq m + n, j \in R(u), s_{ij} \text{ means } w_k \end{cases} \tag{5}$$

The corresponding layout for weight w shown in equation 6. Note that choice of pairs s_{ij} can be flexible. We can choose only possible neighbors instead of enumerating all the pairs.

$$w = [b_u(0), b_u(1), \cdots, b_u(n), b_i(1), \cdots b_i(m) \cdots s_{ij} \cdots] \tag{6}$$

In other words, equation 3 can be reformed as the following form

$$\hat{r}_{ui} = \mu + b_i 1 + b_u 1 + \sum_{j \in R(u)} s_{ij} \left[|R(u)|^{-\frac{1}{2}}(r_{uj} - \bar{b}_u) \right] \tag{7}$$

where b_i, b_u, s_{ij} corresponds to weight of linear regression, and the coefficients on the right of the weight are the input features. In summary, under this framework, the only thing that we need to do is to layout the parameters into a feature vector. In our case, we arrange first n features to b_u then b_i and s_{ij}, then transform the input data into the format of linear regression input. Finally we use a linear regression solver to work the problem out.

2.2 Feature based matrix factorization

The previous section shows that some baseline CF algorithms are linear regression problem. In this section, we will discuss feature-based generalization for matrix factorization. A basic matrix factorization model is stated in Equation 8:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i \tag{8}$$

The bias terms have the same meaning as previous section. We also get two factor term p_u and q_i. p_u models the latent peference of user u. q_i models the latent property of item i.

Inspired by the idea of previous section, we can get a direct generalization for matrix factorization version.

$$\hat{r}_{ui} = \mu + \sum_j w_j x_j + b_u + b_i + p_u^T q_i \tag{9}$$

Equation 9 adds a linear regression term to the traditional matrix factorization model. This allows us to add more bias information, such as neighborhood information and time bias information, etc. However, we may also need a more flexible factor part. For example, we may want a time dependent user factor $p_u(t)$ or hierarchical dependent item factor $q_i(h)$. As we can find from previous section, a direct way to include such flexibility is to use features in factor as well. So we adjust our feature based matrix factorization as follows

$$y = \mu + \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j \right) + \left(\sum_j p_j \alpha_j \right)^T \left(\sum_j q_j \beta_j \right) \tag{10}$$

The input consists of three kinds of features $< \alpha, \beta, \gamma >$, we call α user feature, β item feature and γ global feature. The first part of Equation 10. The name of these features explains their meanings. α describes the user aspects, β describes the item aspects, while γ describes some global bias effect. Figure 1 shows the idea of the procedure.

We can find basic matrix factorization is a special case of Equation 10. For predicting user item pair $< u, i >$, define

$$\gamma = \emptyset,\ \alpha_k = \begin{cases} 1 & k = u \\ 0 & k \neq u \end{cases},\ \beta_k = \begin{cases} 1 & k = i \\ 0 & k \neq i \end{cases} \tag{11}$$

We are not limited to the simple matrix factorization. It enables us to incorporate the neighborhood information to γ, and time dependent user factor by modifying α. Section 3 will present a detailed description of this.

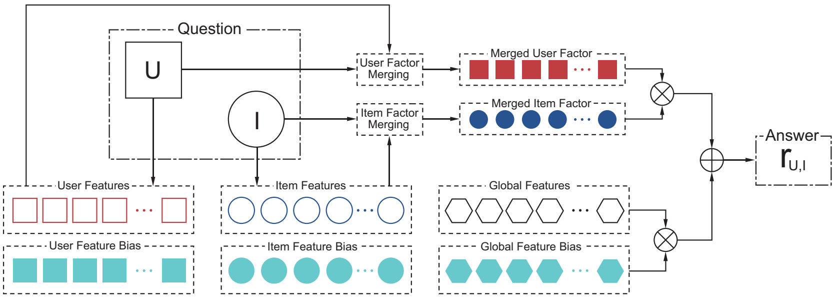

Figure 1: Feature-based matrix factorization

2.3 Active function and loss function

There, you need to choose an active function $f(\cdot)$ to the output of the feature based matrix factorization. Similarly, you can also try various of loss functions for loss estimation. The final version of the model is

$$\hat{r} = f(y) \tag{12}$$

$$Loss = L(\hat{r}, r) + regularization \tag{13}$$

Common choice of active functions and loss are listed as follows:

- identity function, L2 loss, original matrix factorization.

$$\hat{r} = f(y) = y \tag{14}$$

$$Loss = (r - \hat{r})^2 + regularization \tag{15}$$

- sigmoid function, log likelihood, logistic regression version of matrix factorization.

$$\hat{r} = f(y) = \frac{1}{1 + e^{-y}} \tag{16}$$

$$Loss = r \ln \hat{r} + (1 - r) \ln(1 - \hat{r}) + regularization \tag{17}$$

- identity function, smoothed hinge loss[7], maximum margin matrix factorization[8][7]. Binary classification problem, $r \in \{0, 1\}$

$$Loss = h\left((2r - 1)y\right) + regularization \tag{18}$$

$$h(z) = \begin{cases} \frac{1}{2} - z & z \leq 0 \\ \frac{1}{2}(1 - z)^2 & 0 < z < 1 \\ 0 & z \geq 1 \end{cases} \tag{19}$$

2.4 Model Learning

To update the model, we use the following update rule

$$p_i = p_i + \eta \left(\hat{e}\alpha_i \left(\sum_j q_j \beta_j \right) - \lambda_1 p_i \right) \tag{20}$$

$$q_i = q_i + \eta \left(\hat{e}\beta_i \left(\sum_j p_j \alpha_j \right) - \lambda_2 q_i \right) \tag{21}$$

$$b_i^{(g)} = b_i^{(g)} + \eta \left(\hat{e}\gamma_i - \lambda_3 b_i^{(g)} \right) \tag{22}$$

$$b_i^{(u)} = b_i^{(u)} + \eta \left(\hat{e}\alpha_i - \lambda_4 b_i^{(u)} \right) \tag{23}$$

$$b_i^{(i)} = b_i^{(i)} + \eta \left(\hat{e}\beta_i - \lambda_5 b_i^{(i)} \right) \tag{24}$$

Here $\hat{e} = r - \hat{r}$ the difference between true rate and predicted rate. This rule is valid for both logistic likelihood loss and L2 loss. For other loss, we shall modify $\hat{e}$ to be corresponding gradient. η is the learning rate and the λs are regularization parameters that defines the strength of regularization.

3 What information can be included

In this section, we will present some examples to illustrate the usage of our feature-based matrix factorization model.

3.1 Basic matrix factorization

Basic matrix factorization model is defined by following equation

$$y = \mu + b_u + b_i + p_u^T q_i \tag{25}$$

And the corresponding feature representation is

$$\gamma = \emptyset,\ \alpha_k = \left\{ \begin{array}{ll} 1 & k = u \\ 0 & k \neq u \end{array} \right. ,\ \beta_k = \left\{ \begin{array}{ll} 1 & k = i \\ 0 & k \neq i \end{array} \right. \tag{26}$$

3.2 Pairwise rank model

For the ranking model, we are interested in the order of two items i, j given a user u. A pairwise ranking model is described as follows

$$P(r_{ui} > r_{uj}) = sigmoid\left(\mu + b_i - b_j + p_u^T (q_i - q_j) \right) \tag{27}$$

The corresponding features representation are like this

$$\gamma = \emptyset,\ \alpha_k = \begin{cases} 1 & k = u \\ 0 & k \neq u \end{cases},\ \beta_k = \begin{cases} 1 & k = i \\ -1 & k = j \\ 0 & k \neq i, k \neq j \end{cases} \tag{28}$$

by using sigmoid and log-likelihood as loss function. Note that the feature representation gives one extra b_u which is not desirable. We can removed it by give high regularization to b_u that penalize it to 0.

3.3 Temporal Information

A model that include temporal information[4] can be described as follows

$$y = \mu + b_u(t) + b_i(t) + b_u + b_i + (p_u + p_u(t))^T q_i \tag{29}$$

We can include $b_i(t)$ using global feature, and $b_u(t)$, $p_u(t)$ using user feature. For example, we can define a time interpolation model as follows

$$y = \mu + b_i + b_u^s \frac{e-t}{e-s} + b_u^e \frac{t-s}{e-s} + \left(p_u^s \frac{e-t}{e-s} + p_u^e \frac{t-s}{e-s} \right)^T q_i \tag{30}$$

Here e and s mean start and end of the time of all the ratings. A rating that's rated later will be affected more by p^e and b^e and earlier ratings will be more affected by p^s and b^s. For this model, we can define

$$\gamma = \emptyset,\ \alpha_k = \begin{cases} \frac{e-t}{e-s} & k = u \\ \frac{t-s}{e-s} & k = u + n \\ 0 & \text{otherwise} \end{cases},\ \beta_k = \begin{cases} 1 & k = i \\ 0 & k \neq i \end{cases} \tag{31}$$

Note we first arrange the p^s in the first n features then p^e in next n features.

3.4 Neighborhood information

A model that include neighborhood information[3] can be described as below:

$$y = \mu + \sum_{j \in R(u)} s_{ij} \left[|R(u)|^{-\frac{1}{2}} (r_{uj} - \bar{b}_u) \right] + b_u + b_i + p_u^T q_i \tag{32}$$

We only need to implement neighborhood information to global features as described by Section 2.1.

3.5 Hierarchical information

In Yahoo! Music Dataset[2], some tracks belongs to same artist. We can include such hierarchical information by adding it to item feature. The model is described as follows

$$y = \mu + b_u + b_t + b_a + p_u^T(q_t + q_a) \tag{33}$$

Here t means track and a denotes corresponding artist. This model can be formalized as feature-based matrix factorization by redefining item feature.

4 Efficient training for SVD++

Feature-based matrix factorization can naturally incorporate implicit and explicit information. We can simply add these information to user feature α. The model configuration is shown as follows:

$$y = bias + \left(\sum_j \xi_j p_j + \sum_j \alpha_j d_j\right)^T \left(\sum_j \beta_j q_j\right) \tag{34}$$

Here we omit the detail of bias term. The implicit and explicit feedback information is given by $\sum_j \alpha_j d_j$, where α is the feature vector of feedback information, $\alpha_j = \frac{1}{\sqrt{|R(u)|}}$ for implicit feedback, and $\alpha_j = \frac{r_{u,j} - b_u}{\sqrt{|R(u)|}}$ for explicit feedback. d_j is the parameter of implicit and explicit feedback factor. We explicitly state out the implicit and explicit information in Equation 34.

Although Equation 34 shows that we can easily incorporate implicit and explicit information into the model, it's actually very costly to run the stochastic gradient training, since the update cost is linear to the size of nonzero entries of α, and α can be very large if a user has rated many items. This will greatly slow down the training speed. We need to use an optimized method to do training. To show the idea of the optimized method, let's first define a derived user implicit and explicit factor p^{im} as follows:

$$p^{im} = \sum_j \alpha_j d_j \tag{35}$$

The update of d_j after one step is given by the following equation

$$\Delta d_j = \eta \hat{e} \alpha_j \left(\sum_j \beta_j q_j\right) \tag{36}$$

The resulted difference in p^{im} is given by

$$\Delta p^{im} = \eta \hat{e} \left(\sum_j \alpha_j^2 \right) \left(\sum_j \beta_j q_j \right) \tag{37}$$

Given a group of samples with the *same user*, we need to do gradient descent on each of the training sample. The simplest way is to do the following steps for each sample: (1) calculate p^{im} to get prediction (2) update all d_j associates with implicit and explicit feedback. Every time p^{im} has to be recalculated using updated d_j in this way. However, we can find that to get new p^{im}, we don't need to update each d_j. Instead, we only need to update p^{im} using Equation 37. What's more, we can find there is a relation between Δp^{im} and Δd_j as follows:

$$\Delta d_j = \frac{\alpha_j}{\sum_k \alpha_k^2} \Delta p^{im} \tag{38}$$

We shall emphasize that Equation 38 is true even for *multiple updates*, given the condition that the *user is same* in all the samples. We shall mention that the above analysis doesn't consider the regularization term. If L2 regularization of d_j is used during the update as follows:

$$\Delta d_j = \eta \left(\hat{e} \alpha_j \left(\sum_j \beta_j q_j \right) - \lambda d_j \right) \tag{39}$$

The corresponding changes in p^{im} also looks very similar

$$\Delta p^{im} = \eta \left(\hat{e} \left(\sum_j \alpha_j^2 \right) \left(\sum_j \beta_j q_j \right) - \lambda p^{im} \right) \tag{40}$$

However, the relation in Equation 38 no longer holds strictly. But we can still use the relation since it approximately holds when regularization term is small. Using the results we obtained, we can develop a fast algorithm for feature-based matrix factorization with implicit and explicit feedback information. The algorithm is shown in Algorithm 1.

We find that the basic idea is to group the data of the same user together, for the same user shares the same implicit and explicit feedback information. Algorithm 1 allows us to calculate implicit feedback factor only once for a user, greatly saving the computation time.

5 How large-scale data is handled

Recommender system confronts the problem of large-scale data in practice. This is a must when dealing with real problems. For example Yahoo! Music

Algorithm 1 Efficient Training for Implicit and Explicit Feedback

```
for all user $u$ do
  $p^{im} \leftarrow \sum_j \alpha_j d_j$ {calculating implicit feedback}
  $p^{old} \leftarrow p^{im}$
  for all training samples of user $u$ do
    update other parameters, using $p^{im}$ to replace $\sum_j \alpha_j d_j$
    update $p^{im}$ directly , do not update $d_j$.
  end for
  for all $i, \alpha_i \neq 0$ do
    $d_i \leftarrow d_i + \frac{\alpha_i}{\sum_k \alpha_k^2}(p^{im} - p^{old})$ {add all the changes back to $d$}
  end for
end for
```

Dataset[2] consists of more than 200M ratings. A toolkit that's robust to input data size is desirable for real applications.

5.1 Input data buffering

The input training data is extremely large in real application, we don't try to load all the training data into memory. Instead, we buffer all the training data through binary format into the hard-disk. We use stochastic gradient descend to train our model, that is we only need to linearly iterate over the data if we shuffle our data before buffering.

Therefore, our solution requires the input feature to be previously shuffled, then a buffering program will create a binary buffer from the input feature. The training procedure reads the data from hard-disk and uses stochastic gradient descend to train the model. This buffering approach makes the memory cost invariant to the input data size, and allows us to train models over large-scale of input data so long as the parameters fit into memory.

5.2 Execution pipeline

Although input data buffering can solve the problem of large-scale data, it still suffers from the cost of reading the data from hard-disk. To minimize the cost of I/O, we use a pre-fetching strategy. We create a independent thread to fetch the buffer data into a memory queue, then the training program reads the data from memory queue and do training. The procedure is shown in Figure 2

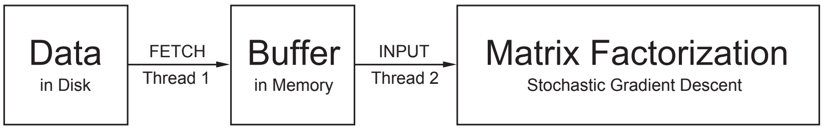

Figure 2: Execution pipeline

This pipeline style of execution removes the burden of I/O from the training thread. So long as I/O speed is similar or faster to training speed, the cost of I/O is negligible, and our our experience on KDDCup'11 proves the success of this strategy. With input buffering and pipeline execution, we can train a model with test RMSE=22.16 for track1 in KDDCup'11[2] using less than 2G of memory, without significantly increasing of training time.

6 Related work and discussion

The most related work of feature based matrix factorization is Factorization Machine [6]. The reader can refer to libFM[3] for a toolkit for factorization machine. Strictly speaking, our toolkit implement a *restricted* case of factorization machine and is more useful in some aspects. We can support global feature that doesn't need to be take into factorization part, which is important for bias features such as user day bias, neighborhood based features, etc. The divide of features also gives hints for model design. For global features, we shall consider what aspect may influence the overall rating. For user and item features, we shall consider how to describe the user preference and item property better. Our model is also related to [1] and [9], the difference is that in feature-based matrix factorization, the user/item feature can be associates with temporal information and other context information to better describe the preference or property in current context. Our current model also has shortcomings. The model doesn't support multiple distinct factorizations at present. For example, sometimes we may want to introduce user vs time tensor factorization together with user vs item factorization. We will try our best to overcome these drawbacks in the future works.

References

[1] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD international confer-*

[2] kddcup.yahoo.com

[3] http://www.libfm.org

ence on Knowledge discovery and data mining, KDD ’09, pages 19–28, New York, NY, USA, 2009. ACM.

[2] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The Yahoo! Music dataset and KDD-Cup’11. In *KDD-Cup Workshop*, 2011.

[3] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’08, pages 426–434, New York, NY, USA, 2008. ACM.

[4] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’09, pages 447–456, New York, NY, USA, 2009. ACM.

[5] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, volume 2007, 2007.

[6] Steffen Rendle. Factorization machines. In *Proceedings of the 10th IEEE International Conference on Data Mining*. IEEE Computer Society, 2010.

[7] Jasson D. M. Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, ICML ’05, pages 713–719, New York, NY, USA, 2005. ACM.

[8] Nathan Srebro, Jason D. M. Rennie, and Tommi S. Jaakola. Maximum-Margin Matrix Factorization. In *Advances in Neural Information Processing Systems 17*, volume 17, pages 1329–1336, 2005.

[9] David H. Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, WWW ’09, pages 111–120, New York, NY, USA, 2009. ACM.