SVDFeature : User Reference Manual

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

APEX Technical Report 2011-09-17(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 2011-09-17(version 1.1),

SVDFeature : User Reference Manual

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. This document serves as a user manual for *SVDFeature* toolkit.

Contents

1 Introduction

This document serves as a user manual for *SVDFeature* [1] toolkit. SVDFeature is a toolkit developed by Apex Data & Knowledge Management Lab during the competition of KDDCup'11[2]. It is designed to solve the feature-based matrix factorization efficiently. New models can be developed just by defining new features. The feature-based setting allows us to include many kinds of information into the model, making the model *informative*. Using the toolkit, we can easily incorporate information such as temporal dynamics, neighborhood relationship, and hierarchical information into the model. Besides rate prediction, it is also capable of doing pairwise ranking tasks.

2 Model

2.1 Model Setting

The model of the toolkit is defined as follows (please refer to our technical report[1] in project page for more details):

$$y = \mu + \left(\sum_j b_j^{(g)}\gamma_j + \sum_j b_j^{(u)}\alpha_j + \sum_j b_j^{(i)}\beta_j\right) + \left(\sum_j p_j\alpha_j\right)^T \left(\sum_j q_j\beta_j\right) \tag{1}$$

The input consists of three kinds of features $< \alpha, \beta, \gamma >$, where we call α user feature, β item feature and γ global feature. The final version of the model is

$$\hat{r} = f(y) \tag{2}$$

$$Loss = L(\hat{r}, r) + regularization \tag{3}$$

Common choices of active function and loss function are listed below:

- identity function, L2 loss, original matrix factorization.

$$\hat{r} = f(y) = y \tag{4}$$

$$Loss = (r - \hat{r})^2 + regularization \tag{5}$$

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

$$\hat{r} = f(y) = \frac{1}{1 + e^{-y}} \tag{6}$$

$$Loss = r \ln \hat{r} + (1 - r) \ln(1 - \hat{r}) + regularization \tag{7}$$

[1] http://apex.sjtu.edu.cn/apex_wiki/svdfeature
[2] http://kddcup.yahoo.com

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

$$Loss = h\left((2r-1)y\right) + regularization \tag{8}$$

$$h(z) = \begin{cases} \frac{1}{2} - z & z \leq 0 \\ \frac{1}{2}(1-z)^2 & 0 < z < 1 \\ 0 & z \geq 1 \end{cases} \tag{9}$$

2.2 Update Rule

To update the model, we use the following update rules

$$p_i = p_i + \eta\left(\hat{e}\alpha_i\left(\sum_j q_j\beta_j\right) - \lambda_1 p_i\right) \tag{10}$$

$$q_i = q_i + \eta\left(\hat{e}\beta_i\left(\sum_j p_j\alpha_j\right) - \lambda_2 q_i\right) \tag{11}$$

$$b_i^{(g)} = b_i^{(g)} + \eta\left(\hat{e}\gamma_i - \lambda_3 b_i^{(g)}\right) \tag{12}$$

$$b_i^{(u)} = b_i^{(u)} + \eta\left(\hat{e}\alpha_i - \lambda_4 b_i^{(u)}\right) \tag{13}$$

$$b_i^{(i)} = b_i^{(i)} + \eta\left(\hat{e}\beta_i - \lambda_5 b_i^{(i)}\right) \tag{14}$$

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

2.3 Model Setting for Pairwise Rank

Pairwise rank setting is a special case of our model. If we are interested in comparing two items i and j for a given user, we will use the following model settings.

$$P(rank(i) > rank(j)) = f(y(i) - y(j)) \tag{15}$$

Here $y(i)$ and $y(j)$ are defined as Equation 1 with the corresponding features $< \alpha(i), \beta(i), \gamma(i) >$ and $< \alpha(j), \beta(j), \gamma(j) >$. Note that $\alpha(i) = \alpha(j)$ for the same user. This enables us to define a new feature vector

$$\gamma(i,j) = \gamma(i) - \gamma(j), \alpha(i,j) = \alpha(i), \beta(i,j) = \beta(i) - \beta(j) \tag{16}$$

Table 1: Usage for two kinds of input

input type	used by
random ordered input	basic usage of feature-based matrix factorization
user grouped input	rank-model, efficient SVD++

and the rank-prediction problem naturally becomes a special case as follows

$$P(rank(i) > rank(j)) = f(y(i,j)) \tag{17}$$

where $y(i,j)$ is defined by using feature $< \alpha(i,j), \beta(i,j), \gamma(i,j) >$. However, the definition of $y(i,j)$ will take undesirable user bias $\sum_j b_j^{(u)} \alpha_j$ into the model. To solve this problem, the toolkit provides an option(no_user_bias=1) to remove this bias when using pairwise rank model.

3 Input Data Format

This section describes the input data format *SVDFeature* . The reader is also refered to the examples in demo folder. There are two kinds of input data format used by the toolkit, random ordered input and user grouped input. This two kinds of input is used by different training method of *SVDFeature* (see Table 1). For basic usage of the toolkit, random ordered input is sufficient. User grouped input is used when we need efficient training with explicit/implicit feedback, or when we need to build rank model.

3.1 Random Ordered Input

3.1.1 Feature Format

The input format is in sparse feature format similar to SVM format. For a training sample, we need to specify three kinds of features $< \alpha, \beta, \gamma >$ as well as prediction target. The format is as follows

```
line:= r k n m <global features> <user features> <item features>
r   := prediction target
k   := number of nonzero global features
n   := number of nonzero user features
m   := number of nonzero item features
<global features> := gid[1]:gvalue[1] ... gid[k]:gvalue[k]
<user features>   := uid[1]:uvalue[1] ... uid[n]:uvalue[n]
<item features>   := iid[1]:ivalue[1] ... iid[m]:ivalue[m]
```

Here id and value correspond to feature id and feature value of nonzero entry. The feature file first specifies the prediction target, and then it states number of nonzero entries in global, user and item feature vector. Then it lists the nonzero global, user and item features in sparse feature format. For example, if we use basic matrix factorization model, the sample of user 0 and item 10 with rate 5 is as follows

```
5  0  1  1  0:1  10:1
```

Here $< 0, 1, 1 >$ states that there are no global feature, 1 user feature and 1 item feature. $0 : 1$ represents the user feature and $10 : 1$ represents the item feature.

3.1.2 Random Order

Random ordered input file should be *randomly ordered*(shuffled) before taken by training program. The program **line_shuffle** can shuffle the lines of text feature files.

3.1.3 Binary Buffer Generation

By default, the training program reads data from a binary format file for efficiency reason. **make_feature_buffer** can generate the binary format file from text feature format.

3.1.4 Summary for Random Ordered Input Generation

- Generate feature file according to feature format.
- Shuffle the feature file.
- Generate binary buffer file from the shuffled feature file.

3.2 User Grouped Input

3.2.1 User Grouped Feature File

The rank-model and efficient training for SVD++ requires the user to be grouped together. Please also refer to Section 2.3 and Section 8 for the corresponding description of the two models. The feature file has the same format as random ordered feature file. The difference is that the lines corresponding to same user shall be grouped consecutively. The idea of user grouped order is shown intuitively shown as follows:

```
[line 1: feature of user 3, item 2]
[line 2: feature of user 3, item 1]
[line 3: feature of user 1, item 0]
[line 4: feature of user 1, item 3]
[line 5: feature of user 1, item 2]
```

The key requirement is summarized as follows:(1) All the lines corresponding to same user are in consecutive order; (2) The order of users, and the order of item in each user group should be shuffled. We provide two programs to achieve this ordering:

- svdpp_randorder takes a file with user id in the first column, and shuffles the file while ensuring same users are grouped together, it will output an order file specifying the shuffled order.
- line_reorder takes the generated order file to reorder another specified file in the generated order.

Assume we have a training data ua.base in the following format:

```
uid iid rate ...\n
```

We will need to do following steps to generate the grouped feature file

```
svdpp_randorder ua.base ua.svdpporder
line_reorder    ua.base ua.svdpporder ua.base.shuffle
generate feature from ua.base.shuffle
```

3.2.2 User Implicit/Explicit Feedback Information

When we want to add implicit/explicit feedback information to the model. We will need to provide an additional file to specify users' implicit/explicit feedback information. We call this file user feedback file. This file contains one line for each user, the order is same as the user order of the user grouped feature file. The format of file is shown as follows:

```
[num lines in group] n fid[1]:fval[1] ... fid[n]:fval[n]
```

The first column gives the number of lines in grouped feature file of corresponding user. Then it specifies the implicit feedback information using the sparse feature format. n gives number of nonzero entries, and then it is followed by n index value pairs. fid gives the index of implicit/explicit feedback information arranged. $fval$ gives the corresponding feature value. A normal setting is $fval = \frac{1}{\sqrt{|R(u)|}}$ for implicit feedback and $fval = \frac{r_{u,j}-b_u}{\sqrt{|R(u)|}}$

for explicit feedback. For example, assume user 3 have rated item 2,4,6,7, and he has 3 lines in the grouped training file, then the extra file for implicit feedback is formatted as follows:

```
3  4  2:0.5 4:0.5 6:0.5 7:0.5
```

Here $fval = \frac{1}{\sqrt{|R(u)|}} = 0.5$. We shall note that when we add the implicit feedback information to the extra file, we no longer need to include them in the grouped feature file.

3.2.3 Binary Buffer File Generation

The same as the random ordered format. We provide a program to create binary buffer from two text files. **make_ugroup_buffer** takes in the grouped feature file and (optionally) user feedback file to create a binary buffer that can be further used by *svd_feature*. When user feedback file is provided, the first column of the file is used to define the user group. If there is no user feedback file, the program use the smallest nonzero entry in user feature to decide the user and define the group.

3.2.4 Summary for User Grouped Input Generation

- create a file obeying the rule of user grouped format.
- generate feature file according to feature format.
- generate user feedback file if necessary.
- generate binary buffer file.

4 Basic Usage

There are two major programs in the toolkit: svd_feature and svd_feature_infer. A basic example usage of the toolkit is like this:

```
svd_feature config.conf
```

Here svd_feature takes configure file config.conf as input to train the model. Detailed description of usage will be elaborated in the following.

4.1 Configure File and Input Parameters

svd_feature and svd_feature_infer both take a configure file as input. All the information about training is defined in the configure file. The following is a part of configure file:

```
# number of each kind of features
num_item   = 1682
num_user   = 943
```

In this example, we define the parameter **num_item** and **num_user**, which are number of item features and number of user features. In general, configure file format can be described as follows:

```
# comment here
[parameter name]   = [parameter value]
```

Besides specifying parameters in configure file, it's also possible passing parameters directly through console. An example is as follows:

```
svd_feature config.conf num_item=10 num_user=10
```

This example defines the two parameters through console. We shall note that there shouldn't be space in each parameter so that num_item=10 can be taken in as a single argument. For detailed description of parameters, please refer to Section 5. Examples of configure files are also provided in our demo folder.

4.2 Model Training

We use following command to explain our training program.

```
svd_feature config.conf num_round=10 model_out_folder=./
```

This command will run the stochastic gradient descent training procedure for 10 rounds. A model file is saved to **model_out_folder** (in our example current folder) after each round. Other parameters are specified in config.conf. After 10 rounds of running, we can get the following model files.

```
0000.model 0001.model ... 0010.model
```

Each model file stores the model of corresponding round, and can be used to generate prediction.

4.3 Continue Training

Assume we currently have the following model files,

```
0000.model 0001.model ... 0007.model
```

we want to continue training from 7 round to get 0010.model, then we can use the following command:

```
svd_feature config.conf num_round=10 continue=1
```

The option **continue=1** will search from 0000 to the latest model to continue running. If there is no model in the **model_out_folder**, then the program will start training from the beginning. However, if the models in the folder are:

```
0000.model 0001.model 0003.model ... 0007.model
```

The program will only starts from 0001.model since 0002 is missing. To start from 0007 in this case, use the following command:

```
svd_feature config.conf num_round=10 continue=1 start_counter=7
```

start_counter specifies the beginning of search index. When we set it to 7, the program starts search from 0007 instead of 0000.

Using the continue training option, we can use a shell script to run the toolkit without saving the models of all the rounds

```
for(( i=$start; i < $nround; i ++ ))
do
  svd_feature config.conf num_round=$((i+1)) start_counter=$i continue=1
  do some other things such as evaluation
  rm `printf %04d.model $i`
done
```

This script runs the training program for one round each time and removes model of the previous round.

4.4 RMSE Evaluation

Suppose we have the following model files at present,

```
0000.model 0001.model ... 0007.model
```

we have already specified the true rating in the test file. Then we can use the following command to do RMSE evaluation:

```
svd_feature_infer config.conf
```

This command will evaluate and output RMSE for each of the model. If we only want RMSE from 0005 to 0007, just use the following command

```
svd_feature_infer config.conf start=5 end=8
```

4.5 Test Prediction

If having those model files

```
0000.model 0001.model ... 0007.model
```

and we want to get prediction using the 0006.model, we can use the following command

```
svd_feature_infer config.conf pred=6 name_pred=pred.txt
```

The program will write the prediction to pred.txt, with one predicted rate in each line and in the same order as the test file.

4.6 Usage Example Summary

```
model training:
    svd_feature config.conf num_round=10
continue training:
    svd_feature config.conf num_round=10 continue =1
rmse evaluation:
    svd_feature_infer config.conf
test prediction:
    svd_feature_infer config.conf pred=6 name_pred=pred.txt
```

5 Parameters

In this section, we will describe the parameters of the toolkit. All these parameters can be specified in the config file. When a parameter has a default value, it's OK not to specify it.

5.1 Data Parameters

5.1.1 input_type

Specify the input file type for training. *default=0*

0 : binary file created by *make_feature_buffer*.

1 : text file in feature format.

Recommend setting: use 0 for efficient training.

5.1.2 buffer_feature

Input binary file name for training which is used by *svd_feature* when input_type=0.

5.1.3 data_in

Input text file name for training which is used by *svd_feature* when input_type=1.

5.1.4 test:input_type

Specify the input file type for test prediction, the same definition as **input_type**, *default=0*.

5.1.5 test:data_in

Specify the test file for RMSE evaluation and test prediction which is used by *svd_feature_infer* when test:input_type=1. The test file should be in text feature format.

5.1.6 test:buffer_feature

Input binary file name for test prediction, the same definition as **buffer_feature**.

5.2 Model Parameters

5.2.1 active_type

active_type defines function f and loss function. *default=0*

0 : identity function, L2 loss

2 : sigmoid function, log-likelihood, output $\hat{r}$ in prediction

3 : sigmoid function, log-likelihood, output y in prediction(instead of $\hat{r}$)

5 : smooth-hinge loss, max-margin matrix factorization

Recommended setting: using 0 and 2 for rate prediction, using 3 for pairwise rank and binary classification.

5.2.2 num_user

Number of user features, no default value.

5.2.3 num_item

Number of item features, no default value.

5.2.4 num_global

Number of global features, no default value.

5.2.5 num_factor

Number of latent factors, no default value.

5.2.6 base_score

Global average of the rate prediction, *default=0.5*.

5.3 Training Parameters

5.3.1 num_round

Total number of round to run the training program, no default value. Corresponding model file will be generated after each round.

5.3.2 train_repeat

How many times to repeat over the entire training data for a round. If train_repeat=10, the program will iterate over the entire training data for 10 times before saving a model out. *default=1*

5.3.3 learning_rate

Learning rate, represented by η in section 2.2, *default=0.01*.

5.3.4 wd_user

Regularization parameter for user factor, represented by λ_1 in section 2.2, *default=0*.

5.3.5 wd_item

Regularization parameter for item factor, represented by λ_2 in section 2.2, *default=0.*

5.3.6 wd_global

Regularization parameter for global bias, represented by λ_3 in section 2.2, *default=0.*

5.3.7 wd_user_bias

Regularization parameter for user bias, represented by λ_4 in section 2.2, *default=0.*

5.3.8 wd_item_bias

Regularization parameter for item bias, represented by λ_5 in section 2.2, *default=0.*

6 Regularization

This section provides a detailed description for regularization method available in *SVDFeature* . The default regularization for *SVDFeature* is L2 regularization, and has already been described in the previous sections. This section can be skipped if the user choose L2 regularization(which commonly works well).

6.1 Discussion of Regularization Update

The first thing we shall point out is that the regularization update in Section 2.2 *is not* the true gradient of loss function. This is because we only add the regularization term when a parameter is updated. But strictly speaking, we shall add the regularization to the parameter even when the parameter is not updated by gradient of loss function. However, directly regularize all the parameters in every step costs too much time. On the other hand, the update style in Section 2.2 works well in most cases, and is commonly referred as(though no strictly true) *L2 regularization.* So in *SVDFeature* , we also use the notation L2 regularization for this kind of regularization.

As we have mentioned, it's not a good choice to directly regularize very parameters at each update. Besides accepting the "L2 regularization" as a

compromised solution, we can also use another trick called lazy decay to solve the problem. The idea is to record the number of rounds between two updates of the parameters, and performs all the regularization together before a new gradient update occurs. For example, if a parameters θ is updated 3 rounds before this update, then we shall multiply θ by $(1-\eta\lambda)^3$ before performing current update. This lazy regularization can help saves the computation cost and make it possible to regularize every parameters after each round. *SVDFeature* supports lazy regularization as well.

6.2 Max-norm Regularization for Factors

Besides the common regularizations and lazy regularizations described in previous subsection. *SVDFeature* also support max-norm regularization for user/item factors. We pose a max-norm bound constraint on the user/item factor matrix. We implemented the projected gradient descent method described in [2]. When we use max-norm regularization, *wd_user* and *wd_item* will be used as constraint parameters for the *bound* of max-norm. They shall be set to a large number if the user want less constraint on the factors.

6.3 Related Parameters

6.3.1 reg_global

regularization for global bias.

0. L2 regularization, only regularize the parameters that has nonzero gradient.
1. L1 regularization, only regularize the parameters that has nonzero gradient.
4. L2 regularization using lazy regularization, regularize all the parameters.
5. L1 regularization using lazy regularization, regularize all the parameters.

6.3.2 reg_method

regularization for user/item factor(matrix)

0. L2 regularization, only regularize the parameters that has nonzero gradient.

1. L1 regularization, only regularize the parameters that has nonzero gradient.

2. max-norm regularization, use project gradient descent method.

4. L2 regularization using lazy regularization, regularize all the parameters.

5. L1 regularization using lazy regularization, regularize all the parameters.

6.4 Detailed Regularization Setting

In common setting of *SVDFeature* , we can use the regularization parameters to set the regularization we want. However, sometimes the regularization parameters provided are not enough. For example, we may include temporal user factor into user feature, and we want to impose a stronger regularization to temporal user factor than basic user factor. On the other hand, parameter *wd_user* only allows us to set the same regularization for all the user factors. In this subsection, we will introduce the detailed regularization setting, which allows us to set different regularization parameters for different range of feature. The following lines gives an example configure for detailed regularization setting.

```
# detailed user feature regularization
up:wd    = 0.02
up:bound = 100
up:wd    = 0.3
up:bound = 300
up:wd    = 0.2
up:bound = 350
```

In this example, we set regularization(*wd_user*) to 0.02 when user feature index is in $[0, 100)$, 0.3 when it's in $[100, 300)$ and 0.2 when it's in $[300, 350)$. By including these lines in configure file, *SVDFeature* will impose the corresponding regularization for each range of feature. We shall note that the when detailed regularization setting is introduced, the original parameter *wd_user* will be ignored, and user shall set the detailed regularization parameters for all the user features (i.e the last *up* : *bound* shall be same as *num_user*). Similar setting is available for global feature and item feature, with prefix *up* changed to *gp* and *ip*.

7 Pairwise Rank Training

7.1 Training Data Generation

Training data should be generated in pairs. For each user, item pairs are randomly sampled and corresponding features are generate (described in section 2.3). An example to show the data generation step is given as below. Suppose we have 2 users and 4 movies, the data is as follows

```
uid   movies watched by user
0     0, 1
1     1, 3
```

Both users have watched 2 movies. We want to build a learning to rank model. For each user, we will pick a movie he has watched, and randomly pick a movie he has not watched. We will construct a pairwise example stating that the user prefers the movie he watched. For user 0, pick movie 0 or 1 as positive sample and movie 2 or 3 as negative sample. If we pick movie 0 and 2 as a pair, the feature format for basic pairwise ranking will be

```
1.0  0  1  2 0:1 0:1 2:-1
```

1.0 means the user prefers positive sample to negative one. The first 0:1 means the user factor is p_0, and the last two features mean item factor is $(q_0 - q_2)$.

7.2 Test Data Generation

We don't need to generate pairwise features for getting ranking order of test data. This is because we use $y_1 - y_2 > 0$ to stand for order pairwise prediction of $rank(1) > rank(2)$. We can simply use y_1 and y_2 for the ranking order. To continue our example in previous section, if we want to know the order preference of user 0 on movie 2 and 3. We can use the following features for test prediction (note the first column takes no meaning here).

```
1.0  0  1  1 0:1 2:1
1.0  0  1  1 0:1 3:1
```

7.3 Rank Pair Generation

There are two choices when we need to do pairwise training using the toolkit.

1. The user randomly generate the ranking pairs from the training data, and let the toolkit take the input feature generated from the pairs.

2. The user gives the training features without generating the ranking pair, the toolkit takes the input and generate the pairs from provided rate information.

Method 1 gives the freedom of rank pair specification to the user, while method 2 can help save user's effort of rank pair generation. In this section, we describe two kinds of rank pair generation used by toolkit in method 2.

7.3.1 Positive-negative Pair Sampling Strategy

Positive-negative pair sampling strategy divides the data into positive and negative set. Pairs are generated by select one sample from positive set and another one from negative set. This pairing method is commonly used in rated/unrated (clicked/not clicked) prediction, when we want to predict whether a user would rate a movie or not. The positive sample set comes from the movies that user has rated, while the negative set is generated by the user using sampling strategy. In this setting, user shall provide the positive set with rating 1 and negative set with rating 0.

7.3.2 Rate Based Pairing

We also provide a random pairing that compares the rate of the random selected pairs. If $r_1 - r_2 \geq rank\_sample\_gap$, we will accept this as a sample indicating first bigger than second one, and if $r_2 - r_1 \geq rank\_sample\_gap$, we will accept this as a sample indicating second one is bigger than first one. **rank_sample_gap** is a parameter specifying the gap that is required to accept the significance of comparison. We can find rate order pairing can possibly learn the difference of the user rating, and can be useful in some scenarios.

7.4 Input Data Type

If the ranking pairs are generated by the user, then any kinds of input format is OK since it has no difference from basic usage. If the user leaves the ranking pair generation to the toolkit, then the input format shall be user grouped format (Section 3.2).

7.5 Related Parameters Setting

7.5.1 active_type

Because we are doing pairwise rank prediction, it is actually a classification problem. To classify whether one item's rank is higher than another

for the same user. We can use active_type = 3 or 5. Recommend setting: active_type=3

7.5.2 no_user_bias

no_user_bias=1 will remove the user bias term $\sum_j b^{(u)}\alpha_j$ from model. Set **no_user_bias=1** for ranking model. *default=0*

7.5.3 input_type

When using ranking pair generation by the toolkit, input_type shall be set to 2 or 3 depending the type (binary/text) of input.

0 : rank pair generated by user, binary input file

1 : rank pair generated by user, text input file

2 : rank pair generation by toolkit, binary input file (generated by *make_ugroup_buffer*)

3 : rank pair generation by toolkit, text input file

7.5.4 model_type

Specifies the different kinds of training algorithms, *default=0*

0 : original version of SVDFeature

1 : rank-model when using toolkit pair generation or optimized update for implicit feedback

Set it to 1 if the user want the rank pair to be generated by the toolkit.

7.5.5 rank_sample_method

This option specifies the pair generation method used by the toolkit, *default=0*

0 : positive negative pairing

1 : rank order pairing

7.5.6 rank_sample_gap

Specifies the gap to accept the comparison as a positive or negative pair, *default=0.0001*

7.6 Key Points Summary

- set $no\_user\_bias = 1$, $active\_type = 3$
- if ranking pair generation by user
  - everything else is similar as the basic usage
- if ranking pair generation by toolkit
  - set $model\_type = 1$, $input\_type = 2$
  - use user grouped input format
  - pay attention to the rank_sample_method

8 Efficient SVD++ Training

8.1 Training Algorithm

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

We use the efficient training method described in Algorithm 1. For more detail explanations of the algorithm, please refer to our technical report [1]. The basic idea is to group the data of same user together. Because the same user shares the same implicit/explicit feedback information. Algorithm 1 allows us to calculate implicit feedback factor only once for a user, and saves the computation time.

8.2 Input Data Type

User grouped feature file is required to do efficient SVD++ training. The data format is described in Section 3.2.

8.3 Related Parameters

8.3.1 model_type

Specifies the different kinds of training algorithms, *default=0*

0 original version of SVDFeature

1 optimized update for implicit feedback and rank-model

Set model_type=1 to when using SVD++.

8.3.2 buffer_feature

input buffer file for training, created by *make_ugroup_buffer*

8.3.3 test:data_in

input text file of grouped feature file for test prediction

8.3.4 test:feedback_in

input text file of implicit feedback information.

8.3.5 num_ufeedback

number of implicit feedback features, most commonly set to number of item.

8.3.6 scale_lr_ufeedback

the scale of learning rate of updating p^{im} during the same user update, *default=1*

$$\eta^{im} = learning\_rate * scale\_lr\_ufeedback$$

8.3.7 wd_ufeedback

regularization parameter during the same user update of p^{im}, *default=0*

8.3.8 wd_ufeedback_bias

regularization parameter during the same user update of implicit feedback bias, *default=0*

9 Common Feature Space

SVDFeature model in Equation 1 uses different parameter space for user feature and item feature. Sometimes we may want to have common parameter space for user feature and item feature. The model is presented by the following equation.

$$y = \mu + \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(iu)} \alpha_j + \sum_j b_j^{(iu)} \beta_j \right) + \left(\sum_j p_j \alpha_j \right)^T \left(\sum_j p_j \beta_j \right) \tag{18}$$

The key difference between Equation 1 and Equation 18 is we use common set of parameters to associate with α and β. This setting can be useful for some problems. For example, if we want to learn the matching scores between users(friendship prediction), we can take α as the feature of first user and β as feature of second user, and it's reasonable to use the common feature space since friendship relation is symmetric. We shall also note that common feature space setting subsumes Equation 1 by feature index arrangement.

9.1 Related Parameters

9.1.1 common_latent_space

Whether we use common feature space for user/item features, *default=0*

0. use original setting of *SVDFeature* as 1
1. use common feature space as Equation 18

9.1.2 num_uiset

Size of user/item features in common feature space setting.

9.1.3 wd_uiset

Regularization for user/item latent factor parameters in common feature space setting.

9.1.4 wd_uiset_bias

Regularization for user/item bias parameters.

9.1.5 Detail Regularization Prefix

Detailed regularization prefix for common latent factor is *uip*, see also Section 6.4.

References

[1] Tianqi Chen, Zhao Zheng, Qiuxia Lu, Weinan Zhang, and Yong Yu. Feature-based matrix factorization. Technical Report APEX-TR-2011-07-11, Apex Data & Knowledge Management Lab, Shanghai Jiao Tong University, July 2011.

[2] Jason Lee, Ben Recht, Ruslan Salakhutdinov, Nathan Srebro, and Joel Tropp. Practical large-scale optimization for max-norm regularization. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1297–1305. 2010.