

# libFM 1.3 - Manual

Steffen Rendle\*

<http://www.libfm.org/>

December 27, 2012

## Contents

<b>1</b>	<b>Installation</b>	<b>2</b>
1.1	Compiling . . . . .	2
<b>2</b>	<b>Data Format</b>	<b>2</b>
2.1	Text Format . . . . .	2
2.1.1	Converting Recommender Files . . . . .	3
2.2	Binary Format . . . . .	3
2.2.1	Transpose data . . . . .	4
<b>3</b>	<b>libFM</b>	<b>4</b>
3.1	Mandatory Parameters . . . . .	5
3.2	Optional Parameters . . . . .	5
3.2.1	Basic Parameters . . . . .	5
3.2.2	Advanced Parameters . . . . .	5
3.3	Learning Methods . . . . .	6
3.3.1	Stochastic Gradient Descent (SGD) . . . . .	6
3.3.2	Alternating Least Squares (ALS) . . . . .	7
3.3.3	Markov Chain Monte Carlo (MCMC) . . . . .	7
3.3.4	Adaptive SGD (SGDA) . . . . .	7
<b>4</b>	<b>License</b>	<b>8</b>

---

\*Steffen Rendle, University of Konstanz, Social Network Analysis, [steffen.rendle@uni-konstanz.de](mailto:steffen.rendle@uni-konstanz.de)

# 1 Installation

- **Windows:** download the compiled executable from <http://www.libfm.org/libfm-1.34.windows.zip>. You can skip section 1.1.
- **Linux:** compile LIBFM by following the instructions in section 1.1.
- **MacOS X:** compile LIBFM by following the instructions in section 1.1.

## 1.1 Compiling

LIBFM has been tested with the GNU compiler collection and GNU make. Both should be available in Linux and MacOS X.

With the following steps, you can build LIBFM:

1. Download libFM source code: <http://www.libfm.org/libfm-1.34.src.tar.gz>
2. Unzip and untar: e.g. `tar -xzf libfm-1.34.src.tar.gz`
3. Enter the directory `libfm-1.34.src` and compile the tools: `make all`

### Overview of files

- `license.txt`: license for usage of LIBFM
- `history.txt`: version history and changes
- `readme.pdf`: this manual
- `Makefile`: compiles the executables using `make`
- `bin`: the folder with the executables<sup>1</sup>
  - `libFM`: the LIBFM tool
  - `convert`: a tool for converting text-files into binary format
  - `transpose`: a tool for transposing binary data matrices
- `scripts`
  - `triple_format_to_libfm.pl`: a Perl script for converting comma/tab-separated datasets into LIBFM-format.
- `src`: the source files of LIBFM and the tools

# 2 Data Format

LIBFM supports two file formats for input data: a text format and a binary format<sup>2</sup>. Working with the text format is easier and recommended for new LIBFM-users.

## 2.1 Text Format

The data format is the same as in SVMlite [3] and LIBSVM [1]<sup>3</sup>: Each row contains a training case  $(\mathbf{x}, y)$  for the real-valued feature vector  $\mathbf{x}$  with the target  $y$ . The row states first the value  $y$  and then the non-zero values of  $\mathbf{x}$ . For binary classification, cases with  $y > 0$  are regarded as the positive class and with  $y \leq 0$  as the negative class.

---

<sup>1</sup>The executables have to be built with `make`, see sec. 1.1.

<sup>2</sup>Binary format is not supported for Windows.

<sup>3</sup>Note: LIBFM supports files where the variable index starts with 0. In SVMlite and LIBSVM, the index has to start with 1. LIBFM supports both, i.e. files that work with SVMlite or LIBSVM can be used directly in LIBFM.

## Example

```
4 0:1.5 3:-7.9
2 1:1e-5 3:2
-1 6:1
...
```

This file contains three cases. The first column states the target of each of the three case: i.e. 4 for the first case, 2 for the second and -1 for the third. After the target, each line contains the non-zero elements of  $\mathbf{x}$ , where an entry like 0:1.5 reads  $x_0 = 1.5$  and 3:-7.9 means  $x_3 = -7.9$ , etc. That means the left side of INDEX:VALUE states the index within  $\mathbf{x}$  whereas the right side states the value of  $x_{\text{INDEX}}$ , i.e.  $x_{\text{INDEX}} = \text{VALUE}$ .

In total the data from the example describes the following data matrix  $X$  and target vector  $y$ :

$$X = \begin{pmatrix} 1.5 & 0.0 & 0.0 & -7.9 & 0.0 & 0.0 & 0.0 \\ 0.0 & 10^{-5} & 0.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 4 \\ 2 \\ -1 \end{pmatrix} \quad (1)$$

### 2.1.1 Converting Recommender Files

In recommender systems often a file format like `userid itemid rating` is used. A Perl script to convert such datasets (and also more complex ones like in context-aware settings) to the libFM file format is included in the `scripts` directory. E.g. for converting the `ratings.dat` file from the Movielens 1M<sup>4</sup> dataset into LIBFM format, call:

```
./triple_format_to_libfm.pl -in ratings.dat -target 2 -delete_column 3 -separator "::-"
```

The output will be written to a file with extension `.libfm`. In the example the output is written to `ratings.dat.libfm`.

If a single dataset consists of multiple files, e.g. a train and test split, then the conversion script should be called with both files (several files for the `-in` option):

```
./triple_format_to_libfm.pl -in train.txt,test.txt -target 2 -separator "\t"
```

Warning: If you would run the conversion script for each file separately, the variables (ids) would not match. E.g. the  $n$ -th variable in the first file is different from the  $n$ -th variable in the second file.

## 2.2 Binary Format

Besides the standard text format, LIBFM supports a binary data format<sup>5</sup>. The advantages of the binary format are: (1) faster reading, (2) if your data does not fit into main memory, the binary data format supports caching data on hard disc and keeping only a small portion in memory (use the `--cache_size` in LIBFM), (3) if you work with ALS and MCMC, you can precompute the transposed data matrix which saves time when the data set is read.

To convert a file in LIBFM text format to the binary data format, use the tool `convert` in the `bin` folder:

#### Convert

```
Version: 1.3.4
Author:  Steffen Rendle, steffen.rendle@uni-konstanz.de
WWW:    http://www.libfm.org/
License: Free for academic use. See license.txt.
```

```
-----
-help          this screen
-ifile         input file name, file has to be in binary sparse format
               [MANDATORY]
-ofilex        output file name for x [MANDATORY]
-ofiley        output file name for y [MANDATORY]
```

<sup>4</sup><http://www.grouplens.org/node/12>

<sup>5</sup>Binary format is not supported for Windows.

**Example** To convert the Movielens dataset from the example above to binary format:

```
./convert --ifile ratings.dat.libfm --ofilex ratings.x --ofiley ratings.y
```

The output will consist of two files: (1) a file containing the data matrix  $X$ , i.e. the predictor variables and (2) a file containing the prediction targets  $y$ . It is recommended that these files have the ending `.x` for the data matrix and `.y` for the targets.

### 2.2.1 Transpose data

For MCMC and ALS learning, a transposed data matrix is used. If you use the text format, the data is automatically transposed by LIBFM internally. However if you use binary format, the transposed data has to be present in binary format as well. To transpose a data matrix to binary format use the tool `transpose`.

```
-----
Transpose
  Version: 1.3.4
  Author:  Steffen Rendle, steffen.rendle@uni-konstanz.de
  WWW:    http://www.libfm.org/
  License: Free for academic use. See license.txt.
-----

-cache_size  cache size for data storage, default=10000000
-help        this screen
-ifile       input file name, file has to be in binary sparse format
             [MANDATORY]
-ofile       output file name [MANDATORY]
```

**Example** To transpose the Movielens dataset from the example above:

```
./transpose --ifile ratings.x --ofile ratings.xt
```

The output will be a transposed copy of the data matrix. It is recommended that the transposed file has the ending `.xt`.

## 3 libFM

The LIBFM tool trains a Factorization Machine (FM) [4] model from training data `-train` and predicts the test data `-test`. LIBFM has the following options:

```
libFM
  Version: 1.3.4
  Author:  Steffen Rendle, steffen.rendle@uni-konstanz.de
  WWW:    http://www.libfm.org/
  License: Free for academic use. See license.txt.
-----

-cache_size  cache size for data storage (only applicable if data is
             in binary format), default=infty
-dim         'k0,k1,k2': k0=use bias, k1=use 1-way interactions,
             k2=dim of 2-way interactions; default=1,1,8
-help        this screen
-init_stdev  stdev for initialization of 2-way factors; default=0.1
-iter        number of iterations; default=100
-learn_rate  learn_rate for SGD; default=0.1
-meta        filename for meta information about data set
-method      learning method (SGD, SGDA, ALS, MCMC); default=MCMC
-out         filename for output
-regular     'r0,r1,r2' for SGD and ALS: r0=bias regularization,
```

	r1=1-way regularization, r2=2-way regularization
-rlog	write measurements within iterations to a file; default='',
-task	r=regression, c=binary classification [MANDATORY]
-test	filename for test data [MANDATORY]
-train	filename for training data [MANDATORY]
-validation	filename for validation data (only for SGDA)
-verbosity	how much infos to print; default=0

### 3.1 Mandatory Parameters

- The first mandatory parameter to specify is the **-task** which is either classification (**-task c**) or regression (**-task r**).
- Secondly training data (**-train**) and test data (**-test**) has to be present. You can use here a data file in LIBFM-text format or binary format (see sec. 3.2.2 for details about using binary files).
- Third, the dimensionality of the factorization machine has to be specified with **-dim**. This argument consists of three numbers:  $k_0, k_1, k_2$ .
  - $k_0 \in \{0, 1\}$  determines if the global bias term  $w_0$  should be used in the model (see [4] for details).
  - $k_1 \in \{0, 1\}$  determines if one-way interactions (bias terms for each variable), i.e.  $\mathbf{w}$  should be used in the model (see [4] for details).
  - $k_2 \in \mathbb{N}_0$  gives the number of factors that are used for pairwise interactions, i.e. the  $k$  of  $V \in \mathbb{R}^{p \times k}$  (see [4] for details).

**Example** An FM for a regression task using bias, 1-way interactions and a factorization of  $k = 8$  for pairwise interactions:

```
./libFM -task r -train ml1m-train.libfm -test ml1m-test.libfm -dim '1,1,8'
```

### 3.2 Optional Parameters

#### 3.2.1 Basic Parameters

- **out**: After training is finished, you can write all predictions of the test dataset to the file specified by **out**. The out-file is in text format, has as many lines as the test dataset and the  $i$ -th line states the prediction of the  $i$ -th test case. Note that for classification the output is the probability that the case has the positive class.
- **rlog**: A logfile with statistics about each iteration is generated. The file is in CSV format using TAB separated fields. Note that it depends on the learning method which fields are reported.
- **verbosity**: With the argument **-verbosity 1**, LIBFM prints more information. This is useful to check if your data is read correctly, and to find errors.

#### 3.2.2 Advanced Parameters

**Grouping** You can group input variables using the **meta** option. Grouping can be used for MCMC, SGDA and ALS to define a more complex regularization structure. Every group can have an individual regularization parameter. To use grouping, the **meta** parameter expects the name of a textfile with as many lines as there are input variables (columns of the data matrix). Each line specifies the group ID of the corresponding input variable. Please note that IDs for groups should be numerical and start with 0. E.g. a grouping file for the data matrix of the example in eq. (1) (which has 7 columns; the largest ID is 6) could be:

2  
2  
0  
1  
1  
1  
0

Which would read: in total there are three groups. The first two variables (columns in the data matrix) have the same group, the third and last have the same group and the fourth to sixth variable have the same group.

**Binary Data and Caching** In sec. 2.2 it was stated that the filenames for binary files should end with `.x` for the data matrix, `.y` for the target and `.xt` for transposed data. If you want to use binary data in LIBFM, the filenames for the command line arguments of training, test and validation should be specified *without* the `.x`, `.y`, `.xt` ending. I.e. if you have compiled training (`ml1m-train.x`, `ml1m-train.y`, `ml1m-train.xt`) and test data (`ml1m-test.x`, `ml1m-test.y`, `ml1m-test.xt`) call:

```
./libFM -task r -train ml1m-train -test ml1m-test -dim '1,1,8'
```

LIBFM will automatically append the proper file extensions and load the data files that are necessary for the learning algorithm.

If your data does not fit into memory, you can specify how much of a file LIBFM is allowed to keep in memory:

```
./libFM -task r -train ml1m-train -test ml1m-test -dim '1,1,8' -cache_size 100000000
```

In this example, 100,000,000 Bytes (100 MB) would be used as cache for each `.x` or `.xt` file. Note that the `.y` files are always read completely into memory.

If the argument `cache_size` is not specified, all data is loaded into memory. Note: you should use caching only if the data does not fit into memory because caching uses the harddisc which will be much slower then memory access.

### 3.3 Learning Methods

By default MCMC inference is used for learning because MCMC is the most easiest to handle (no learning rate, no regularization). In LIBFM you can choose from the following learning methods: SGD, ALS, MCMC and SGDA. For all learning methods, the number of iterations `iter` has to be specified.

#### 3.3.1 Stochastic Gradient Descent (SGD)

Use `-method sgd` for parameter learning with SGD [4]. For stochastic gradient descent the following parameters have to be chosen:

- `-learn_rate`: the learning rate aka step size of SGD which should have a non-zero or positive value.
- `-regular`: the regularization parameters which should have zero or positive value. For SGD you can specify the regularization values the following way:
  - One value (`-regular value`): all model parameters use the same regularization value.
  - Three values (`-regular 'value0,value1,value2'`): 0-way interactions ( $w_0$ ) use `value0` as regularization, 1-way interactions ( $w$ ) use `value1` and pairwise ones ( $V$ ) use `value2`.
  - No value: if the parameter `-regular` is not specified at all, this corresponds to no regularization, i.e. `-regular 0`.
- `-init_stddev`: the standard deviation of the normal distribution that is used for initializing the parameters  $V$ . You should use a non-zero, positive value here.

Please choose these arguments carefully as the prediction quality largely depends on good choices.

### Example

```
./libFM -task r -train ml1m-train.libfm -test ml1m-test.libfm -dim '1,1,8' -iter 1000  
-method sgd -learn_rate 0.01 -regular '0,0,0.01' -init_stdev 0.1
```

### 3.3.2 Alternating Least Squares (ALS)

Use `-method als` for parameter learning with ALS [7]. The following parameters have to be chosen:

- **-regular**: the regularization parameters which should have zero or positive value. For ALS you can specify the regularization values the following way:
  - One value (**-regular value**): all model parameters use the same regularization value.
  - Three values (**-regular 'value0,value1,value2'**): 0-way interactions ( $w_0$ ) use **value0** as regularization, 1-way interactions (**w**) use **value1** and pairwise ones (**V**) use **value2**.
  - Group specific values (**-regular 'value0,value1g1,...,value1gm,value2g1,...,value2gm'**), i.e. for  $m$  groups there are  $1 + 2m$  many regularization values: if the input variables are grouped, for each group and 1-way and 2-way interaction an individual regularization value can be used.
  - No value: if the parameter **-regular** is not specified at all, this corresponds to no regularization, i.e. **-regular 0**.
- **-init\_stdev**: the standard deviation of the normal distribution that is used for initializing the parameters  $V$ . You should use a non-zero, positive value here.

Please choose these arguments carefully as the prediction quality largely depends on good choices.

### Example

```
./libFM -task r -train ml1m-train.libfm -test ml1m-test.libfm -dim '1,1,8' -iter 1000  
-method als -regular '0,0,10' -init_stdev 0.1
```

### 3.3.3 Markov Chain Monte Carlo (MCMC)

Use `-method mcmc` for parameter learning with MCMC [2]. The following parameters have to be chosen:

- **-init\_stdev**: the standard deviation of the normal distribution that is used for initializing the parameters  $V$ . You should use a non-zero, positive value here.

Please choose this argument carefully as the speed of convergence depends on a good choice.

### Example

```
./libFM -task r -train ml1m-train.libfm -test ml1m-test.libfm -dim '1,1,8' -iter 1000  
-method mcmc -init_stdev 0.1
```

### 3.3.4 Adaptive SGD (SGDA)

Use `-method sgda` for parameter learning with adaptive SGD [6]. With adaptive SGD, regularization values (per group and layer) are found automatically. You have to specify a validation set that is used to tune the regularization values:

- **-validation**: a data set containing training examples used as a validation set to tune the regularization values. This set should be distinct from the training dataset.
- **-learn\_rate**: the learning rate aka step size of SGD which should have a non-zero or positive value.
- **-init\_stdev**: the standard deviation of the normal distribution that is used for initializing the parameters  $V$ . You should use a non-zero, positive value here.

Please choose these arguments carefully as the prediction quality largely depends on good choices.

## Example

```
./libFM -task r -train ml1m-train.libfm -test ml1m-test.libfm -dim '1,1,8' -iter 1000  
-method sgda -learn_rate 0.01 -init_stddev 0.1 -validation ml1m-val.libfm
```

## 4 License

Please see `license.txt` for details. If you use LIBFM in your work, please cite the following paper<sup>6</sup>:

```
@article{rendle:tist2012,  
  author = {Rendle, Steffen},  
  title = {Factorization Machines with {libFM}},  
  journal = {ACM Trans. Intell. Syst. Technol.},  
  issue_date = {May 2012},  
  volume = {3},  
  number = {3},  
  month = May,  
  year = {2012},  
  issn = {2157-6904},  
  pages = {57:1--57:22},  
  articleno = {57},  
  numpages = {22},  
  publisher = {ACM},  
  address = {New York, NY, USA},  
}
```

## References

- [1] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2:27:1–27:27, May 2011.
- [2] Christoph Freudenthaler, Lars Schmidt-Thieme, and Steffen Rendle. Bayesian factorization machines. In *NIPS workshop on Sparse Representation and Low-rank Approximation*, 2011.
- [3] Thorsten Joachims. *Making large-scale support vector machine learning practical*, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.
- [4] Steffen Rendle. Factorization machines. In *Proceedings of the 10th IEEE International Conference on Data Mining*. IEEE Computer Society, 2010.
- [5] Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012.
- [6] Steffen Rendle. Learning recommender systems with adaptive regularization. In *WSDM '12: Proceedings of the third ACM international conference on Web search and data mining*, New York, NY, USA, 2012. ACM.
- [7] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2011.

---

<sup>6</sup>Please check <http://www.libfm.org/> for an up-to-date citation string.