Journal of Machine Learning Research 17 (2016) 1-5 Submitted 7/15; Revised 5/16; Published 10/16

*fast*FM: A Library for Factorization Machines

Immanuel Bayer IMMANUEL.BAYER@UNI-KONSTANZ.DE
University of Konstanz
78457 Konstanz , Germany

Editor: Cheng Soon Ong

Abstract

Factorization Machines (FM) are currently only used in a narrow range of applications and are not yet part of the standard machine learning toolbox, despite their great success in collaborative filtering and click-through rate prediction. However, Factorization Machines are a general model to deal with sparse and high dimensional features. Our Factorization Machine implementation (*fast*FM) provides easy access to many solvers and supports regression, classification and ranking tasks. Such an implementation simplifies the use of FM for a wide range of applications. Therefore, our implementation has the potential to improve understanding of the FM model and drive new development.

Keywords: Python, MCMC, matrix factorization, context-aware recommendation

1. Introduction

This work aims to facilitate research for matrix factorization based machine learning (ML) models. Factorization Machines are able to express many different latent factor models and are widely used for collaborative filtering tasks (Rendle, 2012b). An important advantage of FM is that the model equation

$$w_0 \in \mathbb{R}, x, w \in \mathbb{R}^p, v_i \in \mathbb{R}^k$$

$$\hat{y}^{FM}(x) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j>i}^{p} \langle v_i, v_j \rangle x_i x_j \tag{1}$$

conforms to the standard notation for vector based ML. FM learn a factorized coefficient $\langle v_i, v_j \rangle$ for each feature pair $x_i x_j$ (eq. 1). This makes it possible to model very sparse feature interactions, as for example, encoding a sample as $x = \{\cdots, 0, \overbrace{1}^{x_i}, 0, \cdots, 0, \overbrace{1}^{x_j}, 0, \cdots\}$ yields $\hat{y}^{FM}(x) = w_0 + w_i + w_j + v_i^T v_j$ which is equivalent to (biased) matrix factorization $R_{i,j} \approx b_0 + b_i + b_j + u_i^T v_j$ (Srebro et al., 2004). Please refer to Rendle (2012b) for more encoding examples. FM have been the top performing model in various machine learning competitions (Rendle and Schmidt-Thieme, 2009; Rendle, 2012a; Bayer and Rendle, 2013) with different objectives (e.g. What Do You Know? Challenge[1], EMI Music Hackathon[2]). *fast*FM includes solvers for regression, classification and ranking problems (see Table 1) and addresses the following needs of the research community: (i) easy interfacing for dynamic

1. `http://www.kaggle.com/c/WhatDoYouKnow`
2. `http://www.kaggle.com/c/MusicHackathon`

and interactive languages such as R, Python and Matlab; (ii) a Python interface allowing interactive work; (iii) a publicly available test suite strongly simplifying modifications or adding of new features; (iv) code is released under the **BSD-license** allowing the integration in (almost) any open source project.

2. Design Overview

The *fast*FM library has a multi layered software architecture (Figure 1) that separates the interface code from the performance critical parts (*fast*FM-core). The core contains the solvers, is written in C and can be used stand alone. Two user interfaces are available: a command line interface (CLI) and a Python interface. Cython (Behnel et al., 2011) is used to create a Python extension from the C library. Both, the Python and C interface, serve as reference implementation for bindings to additional languages.

2.1 fastFM-core

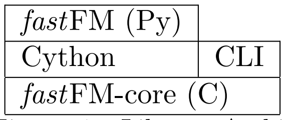

Figure 1: Library Architecture

FM are usually applied to very sparse design matrices, often with a sparsity over 95 %, due to their ability to model interaction between very high dimensional categorical features. We use the standard compressed row storage (CRS) matrix format as underlying data structure and rely on the CXSparse[3] library (Davis, 2006) for fast sparse matrix / vector operations. This simplifies the code and makes memory sharing between Python and C straight forward.

*fast*FM contains a test suite that is run on each commit to the GitHub repository via a continuous integration server[4]. Solvers are tested using state of the art techniques, such as Posterior Quantiles (Cook et al., 2006) for the MCMC sampler and Finite Differences for the SGD based solvers.

2.2 Solver and Loss Functions

*fast*FM provides a range of solvers for all supported tasks (Table 1). The MCMC solver implements the Bayesian Factorization Machine model (Freudenthaler et al., 2011) via Gibbs sampling. We use the pairwise Bayesian Personalized Ranking (BPR) loss (Rendle et al., 2009) for ranking. More details on the classification and regression solvers can be found in Rendle (2012b).

Task	**Solver**	**Loss**
Regression	ALS, MCMC, SGD	Square Loss
Classification	ALS, MCMC, SGD	Probit (MAP), Probit, Sigmoid
Ranking	SGD	BPR (Rendle et al., 2009)

Table 1: Supported solvers and tasks

3. CXSparse is LGPL licensed.
4. `https://travis-ci.org/ibayer/fastFM-core`

2.3 Python Interface

The Python interface is compatible with the API of the widely-used `scikit-learn` library (Pedregosa et al., 2011) which opens the library to a large user base. The following code snippet shows how to use MCMC sampling for an FM classifier and how to make predictions on new data.

```
fm = mcmc.FMClassification(init_std=0.01, rank=8)
y_pred = fm.fit_predict(X_train, y_train, X_test)
```

*fast*FM provides additional features such as warm starting a solver from a previous solution (see MCMC example).

```
fm = als.FMRegression(init_std=0.01, rank=8, l2_reg=2)
fm.fit(X_train, y_train)
```

3. Experiments

libFM[5] is the reference implementation for FM and the only one that provides ALS and MCMC solver. Our experiments show, that the ALS and MCMC solver in *fast*FM compare favorable to libFM with respect to runtime (Figure 2) and are indistinguishable in terms of accuracy. The experiments have been conducted on the MovieLens 10M data set using the original split with a fixed number of 200 iterations for all experiments. The x-axis indicates the number of latent factors (rank), and the y-axis the runtime in seconds. The plots show that the runtime scales linearly with the rank for both implementations. The code snippet

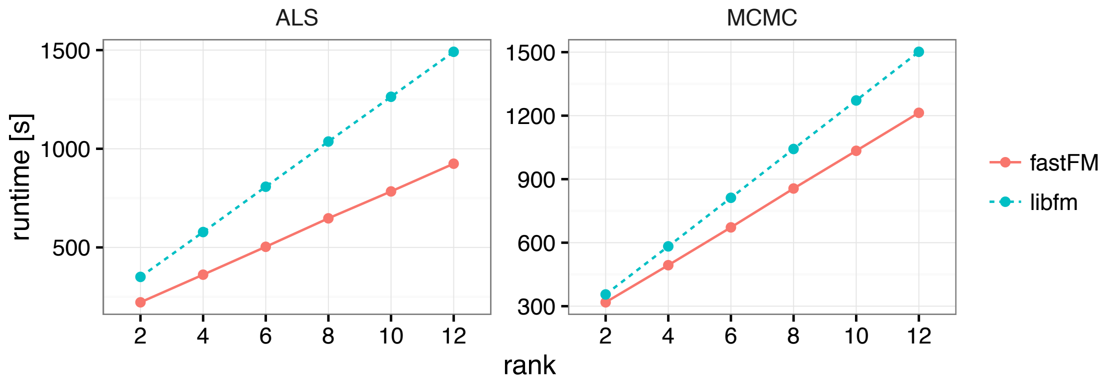

Figure 2: A runtime comparison between *fast*FM and libFM is shown. The evaluation is done on the MovieLens 10M data set.

below shows how simple it is to write Python code that allows model inspection after every iteration. The induced Python function call overhead occurs only once per iteration and is therefore neglectable. This feature can be used for Bayesian Model Checking as demonstrated in Figure 3. The figure shows MCMC summary statistics for the first order hyper parameter σ_w. Please note that the MCMC solver uses Gaussian priors for the model parameter (Freudenthaler et al., 2011).

5. `http://libfm.org`

```
fm = mcmc.FMRegression(n_iter=0)
# initialize coefficients
fm.fit_predict(X_train, y_train, X_test)

for i in range(number_of_iterations):
    y_pred = fm.fit_predict(X_train, y_train, X_test, n_more_iter=1)
    # save, or modify (hyper) parameter
    print(fm.w_, fm.V_, fm.hyper_param_)
```

Many other analyses and experiments can be realized with a few lines of Python code without the need to read or recompile the performance critical C code.

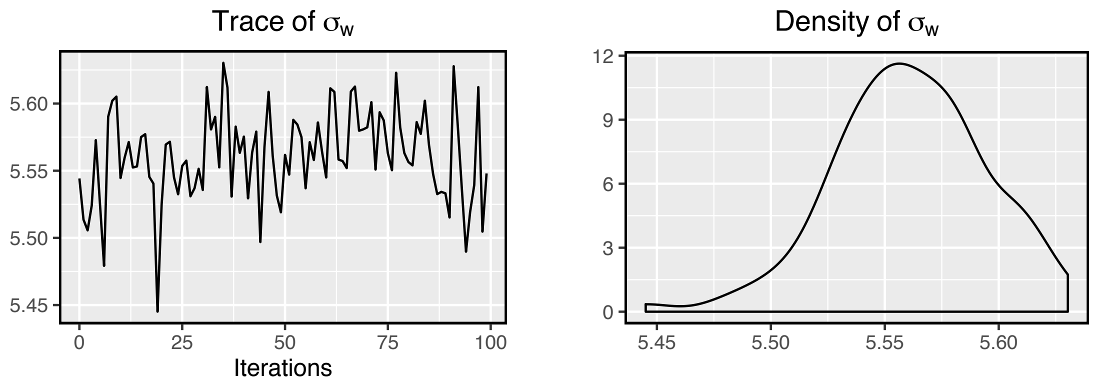

Figure 3: MCMC chain analysis and convergence diagnostics example for the hyperparameter σ_w evaluated on the MovieLens 10M data set.

4. Related Work

Factorization Machines are available in the large scale machine learning libraries GraphLab (Low et al., 2014) and Bidmach (Canny and Zhao, 2013). The toolkit Svdfeatures by Chen et al. (2012) provides a general MF model that is similar to a FM. The implementations in GraphLab, Bidmach and Svdfeatures only support SGD solvers and don't provide a ranking loss. It's not our objective to replace these distributed machine learning frameworks: but to be provide a FM implementation that is easy to use and easy to extend without sacrificing performance.

Acknowledgments

This work was supported by the DFG under grant Re 3311/2-1.

References

Immanuel Bayer and Steffen Rendle. Factor models for recommending given names. In *ECML/PKDD 2013 Discovery Challenge Workshop, part of the European Conference* *on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 81–89, 2013.

Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13 (2):31–39, 2011.

John Canny and Huasha Zhao. Bidmach: Large-scale learning with zero memory allocation. In *BigLearn Workshop, NIPS*, 2013.

Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *The Journal of Machine Learning Research*, 13(1):3619–3622, 2012.

Samantha R Cook, Andrew Gelman, and Donald B Rubin. Validation of software for bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*, 15(3), 2006.

Timothy A Davis. *Direct methods for sparse linear systems*, volume 2. Siam, 2006.

Christoph Freudenthaler, Lars Schmidt-thieme, and Steffen Rendle. Bayesian factorization machines. In *Proceedings of the NIPS Workshop on Sparse Representation and Low-rank Approximation*, 2011.

Yucheng Low, Joseph E Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E Guestrin, and Joseph Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041*, 2014.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Steffen Rendle. Social network and click-through prediction with factorization machines. In *KDD-Cup Workshop*, 2012a.

Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3 (3):57:1–57:22, May 2012b. ISSN 2157-6904.

Steffen Rendle and Lars Schmidt-Thieme. Factor models for tag recommendation in bibsonomy. In *ECML/PKDD 2008 Discovery Challenge Workshop, part of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 235–243, 2009.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI '09*, pages 452–461, Arlington, Virginia, United States, 2009.

Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336, 2004.