Fast Variational Learning of Factorization Machines for Large-Scale Recommender Systems

Abstract

Factorization machines (FMs) are a powerful tool for regression and classification in the context of sparse observations, that has been successfully applied to collaborative filtering, especially when side information over users or items is available. Bayesian formulations of FMs have been proposed to provide confidence intervals over the predictions made by the model, however they usually involve Markov-chain Monte Carlo methods that require many samples to provide accurate predictions, resulting in slow training in the context of large-scale data. In this paper, we propose a variational formulation of factorization machines that allows us to derive a simple objective that can be easily optimized using standard stochastic gradient descent, making it amenable to large-scale data. Our algorithm learns an approximate posterior distribution over the user and item parameters, which in turn provides confidence intervals over the predictions. We show, using several datasets, that it outperforms existing methods in terms of prediction accuracy, and suggest future applications in active learning strategies, e.g. preference elicitation techniques.

1 Introduction

Data collected from a crowd is imperfect: ratings in recommender systems, outcomes of students over educational exercises, annotations provided by humans for crowdsourcing tasks. It is crucial to model it properly in order to achieve social impact: provide interesting recommendations to users, identify misconceptions of students, determine the true label. The challenge here relies in modeling both the items being annotated, and the users annotating them.

In the particular case of recommender systems, we have access to some ratings provided by users over items, and we want to generalize to new user-item pairs. Collaborative filtering [Zhou *et al.*, 2008] is a famous technique that consists in learning an embedding for each user and item, so that the rating can be expressed as a function of the user and item embeddings. In various applications, side information is also available on either users or items, such as: the different genres of the movies, or their actors, directors, etc. The challenge now becomes, how to use this extra information to improve the recommendations? This is why factorization machines were developed [Rendle, 2012]. They generalize collaborative filtering in the presence of side information.

Models typically encountered in collaborative filtering, such as the latent factor model, learn point estimates of the user and item parameters; but it is sometimes more useful to learn distributions over these parameters, in order to have confidence intervals over predictions that can guide decision making. In educational applications, this is particularly useful: what is the uncertainty over the algebra level of this student? Should I continue to ask to them algebra questions, or should I switch to geometry? On a recommender system, one can even let the user control whether they want recommendations with high confidence, or more risky ones in order to explore more the movie database. One way to do so is to use probabilistic matrix factorization.

However, when we learn these models using Bayesian inference, the posterior distribution over the parameters is usually intractable, and we resort to MCMC techniques like Metropolis-Hastings [Cai, 2010] or Gibbs sampling [Porteous *et al.*, 2010; Wang *et al.*, 2018]. These techniques have the advantage to recover the exact distribution asymptotically, at the cost of many samples. Variational inference [Hoffman *et al.*, 2013; Kingma and Welling, 2014] have been proposed to perform at a cheaper cost an approximate inference of the posterior.

In this paper, we propose a variational approach to learn factorization machines. Distributions over the user and item parameters are estimated easily, and likelihood maximization is done by increasing a lower bound. We show that learning becomes several orders of magnitude faster than existing approaches to learn factorization machines. By acting directly on factorization machines, our results naturally extend to the latent factor models typically encountered in collaborative filtering such as probabilistic matrix factorization.

Our contribution is a simple algorithm for learning Bayesian factorization machines that works both for classification and regression tasks. Although our model does not have hyperpriors, we show using several datasets of various sparsity that it achieves higher accuracy at predicting unseen pairs than existing methods, especially when the data is sparse. Also, the time to perform each epoch is reduced by at least half compared to other models on large datasets.

This paper is built as follows. First, we expose related work, then we present factorization machines, our variational objective, our algorithm, experiments and results.

2 Related Work

Factorization machines were described in [Rendle, 2012], where they were applied for classification and regression. For classification tasks, the model can be seen as a generalization of multidimensional item response theory [Reckase, 2009]. For regression tasks, it can be seen as a generalization of the latent factor model.

Rendle [2012] describe several algorithms to train several variants of factorization machines, notably a Bayesian version where every feature (user, item, or other) has a Gaussian prior, and Beta hyperpriors. So using a Markov-chain Monte Carlo method called Gibbs sampling, they can sample the hyperpriors, the parameters, then the predictions.

We here propose to relax the goal of learning the exact posterior. Instead, we will learn an approximation of the posterior of the model parameters. As we observe more points, our approximation of the posterior will fit the true posterior better.

Even more variants of factorization machines have been proposed, such as deep factorization machines [Vie, 2018], convex factorization machines [Blondel *et al.*, 2015; Yamada *et al.*, 2017] and higher-order factorization machines [Blondel *et al.*, 2016]. In all these works, the MCMC version is recognized as a hard baseline, that sometimes beats deep counterparts [Vie, 2018], which is why it is our principal competitor in this paper.

To the best of our knowledge, the only independent attempts at developing variational learning for factorization machines are an implementation in Chainer on GitHub [Moody, 2016]. However, they have more constrained priors and their implementation is slower than ours. We also found an unpublished preprint on a similar method [Saha *et al.*, 2017], but they did not report their results and have lower results on Movie10M than Movie1M according to their figures. We have different graphical models, that makes our objective function simpler, as well as the resulting algorithms.

3 Factorization Machines

Let d be an integer. Every feature $k = 1, \dots, K$, be it a user, an item, a tag, etc. are parameterized using one scalar (bias) w_k and a vector (embedding) $\boldsymbol{v}_k \in \mathbb{R}^d$. We will note V the matrix of all embeddings $(v_1, \dots, v_K)$.

Each sample $i = 1, \dots, n$ is described by a pair $(\boldsymbol{x}, y)$ where y is a label (say, a rating) and $\boldsymbol{x}$ is a sparse vector which determines the presence ($x_k \neq 0$) or absence ($x_k = 0$) of each feature k, see Table 1. For example, if there are N_u users, N_i items and no side information, one can encode the event "user i gave item j 5 stars" with the pair

$$((0, \dots, 0, \underbrace{1}_{i}, 0, \dots, 0, \underbrace{1}_{N_u+j}, 0, \dots, 0), 5).$$

The expression at the core of factorization machines is the following:

	Sparse features $\boldsymbol{x}$						$\boldsymbol{y}$
	U_0	U_1	U_2	I_0	I_1	I_2	
User 1 Item 1: 5/5	0	1	0	0	1	0	5
User 1 Item 2: 4/5	0	1	0	0	0	1	4
User 2 Item 1: 2/5	0	0	1	0	1	0	2
User 2 Item 1: 2/5	0	0	1	0	1	0	2
User 2 Item 2: 5/5	0	0	1	0	0	1	5

Table 1: Example of sparse features encoded for a factorization machine. Each column of $\boldsymbol{x}$ corresponds to either a user or an item.

$$\begin{aligned} y(\boldsymbol{x}) &= \mu + \sum_{k=1}^{K} w_k x_k + \sum_{1 \leq k < \ell \leq K} x_k x_\ell \langle \boldsymbol{v}_k, \boldsymbol{v}_\ell \rangle \\ &= \mu + \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \frac{1}{2} \left(||\boldsymbol{V}\boldsymbol{x}||_2^2 - \sum_{c=1}^{d} ||(\boldsymbol{V}^T)_c \circ \boldsymbol{x}||_2^2 \right) \\ &= \mu + \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \frac{1}{2} \left(||\boldsymbol{V}\boldsymbol{x}||_2^2 - ||(\boldsymbol{V} \circ \boldsymbol{V})^T (\boldsymbol{x} \circ \boldsymbol{x})||_2^2 \right) \end{aligned}$$

where $\circ$ denotes the pairwise product and μ is a global bias. This latter expression is easier to compute as $\boldsymbol{x}$ is sparse, which also makes it convenient for computing the predictions of one batch.

In the particular case where there is no side information, N_u users, N_i items, and each sample is a pair $(\boldsymbol{x}_{ij}, r_{ij})$ where r_{ij} is the rating given by user i over item j and $\boldsymbol{x}_{ij}$ is the concatenation of one 1-hot vector of size N_u and one 1-hot vector of size N_i, one can recover collaborative filtering:

$$y(\boldsymbol{x}) = \mu + w_i + w_{N_u+j} + \langle \boldsymbol{v}_i, \boldsymbol{v}_{N_u+j} \rangle.$$

Regression For a new sample x of sparse features, factorization machine will output a prediction $\hat{y} \in \mathcal{N}(y(\boldsymbol{x}), \sigma)$ where σ is a regularization parameter. A convex loss function can be chosen for training such as least squares.

Classification For a new sample x of sparse features, factorization machine will output a prediction $\hat{y} \in \mathcal{B}(\Phi(y))$ where Φ is the sigmoid function and $\mathcal{B}$ is the Bernoulli distribution. If we just compute $\boldsymbol{w}$, not the embeddings $\boldsymbol{V}$, we recover logistic regression. A convex loss function is chosen such as cross-entropy for training.

3.1 Training of FMs

Training of FMs can be done using standard algorithms such as stochastic gradient descent (SGD), alternating least squares (ALS) or Gibbs sampling [Rendle, 2012], in their Bayesian version where parameters w_k and $\boldsymbol{v}_k$ are random variables.

The lowest error is usually achieved by training FMs using MCMC, with priors $w_k \sim \mathcal{N}(0, 1)$ $\boldsymbol{v}_k \sim \mathcal{N}(\mu, (1/\lambda) I_d)$ and hyperpriors: $\mu \sim \mathcal{N}(\mu, 1\lambda)$ $\lambda \sim \Gamma(1, 1) = \mathcal{U}(0, 1)$ where $\Gamma, \mathcal{U}$ respectively denote the Gamma, uniform distributions.

4 Our Contribution

θ denotes random variables $\boldsymbol{w}$ and $\boldsymbol{V}$. We will denote sample i by its features $\boldsymbol{x}_i$ and outcome y_i.

For regression, $p(y_i|\boldsymbol{x}_i, \theta) = \mathcal{N}(y_{FM}(\boldsymbol{x}_i), \sigma)$ where σ is a regularization parameter that can be learned using cross validation.

For classification, $p(y_i|\boldsymbol{x}_i, \theta) = \mathcal{B}(\Phi(y_{FM}))$. where $\mathcal{B}$ is the Bernouilli distribution and Φ is the sigmoid function.

4.1 Priors

We tried three different priors.

Simplest prior

This is the simplest prior. All means are 0, all standard deviations are 1.

$$p(w_k) = \mathcal{N}(0, 1) \qquad p(\boldsymbol{v}_k) = \mathcal{N}(0, I_d).$$

Hyperpriors

This is the prior proposed in [Rendle, 2012], with hyperpriors. The Gamma hyperprior is convenient because it is the conjugate of the Gaussian distribution.

$$p(w_k) = \mathcal{N}(\mu, \lambda) \qquad p(\boldsymbol{v}_k) = \mathcal{N}(\mu, (1/\lambda)I_d)$$

where $\mu \sim \mathcal{N}(0, 1)$ and $\lambda \sim \Gamma(1, 1) = U(0, 1)$. Furthermore, $1/\sigma \sim \Gamma(1, 1) = U(0, 1)$.

Adaptive prior

We now describe a prior that considers that features for which there are many samples should have a more precise prior.

$$p(w_k) = \mathcal{N}(0, 1/N_k) \qquad p(\boldsymbol{v}_k) = \mathcal{N}(0, (1/N_k)I_d)$$

where N_k is the number of samples that involve feature k.

For example, $\log p(w_k) = N_k||w_k||_2^2$. So if we have more samples that involve feature k, its bias and embedding will be more regularized. This prior is related to alternating least squares with weighted regularization [Zhou *et al.*, 2008], which achieves good performance.

4.2 Approximate posteriors

Our variational approximation:

$$q(w_k) = \mathcal{N}(\mu_k^w, \sigma_k^w) \quad q(\boldsymbol{v}_k) = \mathcal{N}(\boldsymbol{\mu}_k^v, \boldsymbol{\sigma}_k^v)$$

where μ_k^w, σ_k^w, $\boldsymbol{\mu}_k^v$, $\boldsymbol{\sigma}_k^v$ are parameters that we will learn. Hence, we have $2K(d + 1)$ parameters to estimate, see Figure 1.

Sampling is easy because we just need to do:

$$w_k \leftarrow \mu_k^w + \varepsilon \cdot \sigma_k^w \quad \boldsymbol{v}_k \leftarrow \boldsymbol{\mu}_k^v + \boldsymbol{\varepsilon} \circ \boldsymbol{\sigma}_k^v$$

for some $\varepsilon \sim \mathcal{N}(0, 1)$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, I_d)$.

4.3 Variational Training of VFMs

What we would like to increase is the likelihood over all training samples:

$$\log p(\boldsymbol{y}) = \sum_{i=1}^{N} \log p(y_i)$$

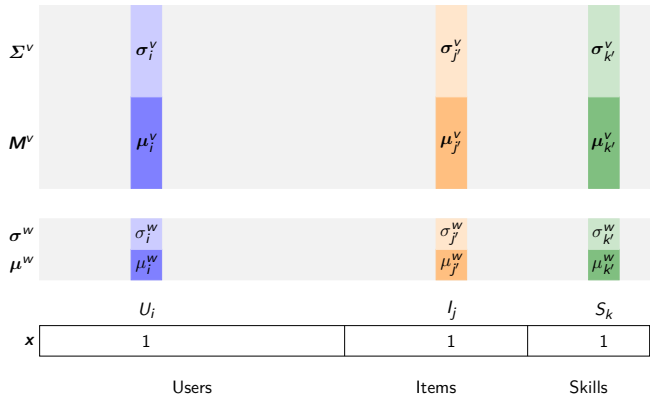

Figure 1: Parameters of a VFM. For each feature we learn a distribution over a bias and a distribution over the embedding. Together, they form an approximate posterior distribution from which we can sample the parameters, and the predictions.

But it's intractable. So instead we will try to maximize a lower bound over $\log p(\boldsymbol{y})$:

$$\begin{aligned}\log p(y_i) &\geq \mathbb{E}_{q(\theta)}[\log p(y_i|x_i, \theta)] - D_{KL}(q(\theta)||p(\theta)) \\ &= \underbrace{\mathbb{E}_{q(\theta)}[\log p(y_i|x_i, \theta) + \log p(\theta) - \log q(\theta)]}_{\text{Evidence Lower Bound (ELBO) } \mathcal{L}_b}\end{aligned}$$

Hopefully, by increasing the Evidence Lower Bound (ELBO), we will indirectly increase the log-likelihood. This is at the core of variational inference.

In our case, the ELBO $\mathcal{L}_b$ is given by:

$$\begin{aligned}\mathcal{L}_b &= \mathbb{E}_{q(\theta)}\left[\sum_i \log p(y_i|x_i, \theta) + \log p(\theta) - \log q(\theta)\right] \\ &= \mathbb{E}_{q(\theta)}\left[\sum_i \log p(y_i|x_i, \theta)\right] \\ &\quad + \mathbb{E}_{q(\theta)}\left[\log p(w)p(V) - \log q(w)q(V)\right]\end{aligned}$$

When we compute it for a single batch of samples $B \subset \{1, \ldots, N\}$, we have to rescale the ELBO estimate. We denote X_B the batch of sparse features, and $F(X_B)$ the multiset of features that are present in the batch: $k \in F(X_B) \iff \exists i \in B, x_{ik} > 0$.

$$\begin{aligned}\mathcal{L}_b &= \frac{N}{|B|}\mathbb{E}_{q(\theta)}\left[\sum_{i \in B} \log p(y_i|x_i, \theta)\right. \\ &\quad \left. + \sum_{k \in F(X_B)} \frac{\log p(w_k)p(\boldsymbol{v}_k) - \log q(w_k)q(\boldsymbol{v}_k)}{N_k}\right] \\ &\simeq \frac{N}{|B|}\sum_{i \in B} \log p(y_i|x_i, \theta) \\ &\quad + \frac{N}{|B|}\sum_{k \in F(X_B)} \frac{\log p(w_k)p(\boldsymbol{v}_k) - \log q(w_k)q(\boldsymbol{v}_k)}{N_k} \\ &\quad \theta \sim q(\theta)\end{aligned}$$

We only need one sample of each feature if the batch is big enough [Kingma and Welling, 2013].

In the case of regression with the adaptive prior, ELBO has the following expression:

$$\begin{aligned}\mathcal{L}_b &= C_{\epsilon_k,\boldsymbol{\epsilon_k}} + \frac{N}{|B|}\left[\sum_{i\in B}\frac{||y_{FM}(x_i)-y_i||_2^2}{\sigma} + \sum_{k\in F(X_B)}||w_k||_2^2 + ||\boldsymbol{v}_k||_2^2\right] \\ &= C_{\epsilon_k,\boldsymbol{\epsilon_k}} + \frac{N}{|B|}\sum_{i\in B}\frac{||y_{FM}(x_i)-y_i||_2^2}{\sigma} \\ &\quad + \frac{N}{|B|}\sum_{k\in F(X_B)}||\mu_k^w + \varepsilon_k\sigma_k^w||_2^2 + ||\boldsymbol{\mu}_k^v + \boldsymbol{\varepsilon}_k\circ\boldsymbol{\sigma}_k^v||_2^2\end{aligned}$$

for the sampled values of ε_k and $\boldsymbol{\varepsilon}_k$, where $C_{\epsilon_k,\boldsymbol{\epsilon_k}}$ is a constant because it only depends on randomness, not the optimized parameters, $|B|$ is the batch size and N_k is the number of samples that involve feature k on the whole train set.

The gradients of $\mathcal{L}_b$ with respect to parameters $\mu_k^w, \sigma_k^w, \boldsymbol{\mu}_k^v, \boldsymbol{\sigma}_k^v$ are easy to compute because y_{FM} is linear in them in the regression case, log-linear in the classification case. So $\mathcal{L}_b$ is polynomial in the parameters.

We learn the parameters using stochastic gradient descent over minibatches. We only update the parameters of features that are present in the batch, which simplifies computation over one batch. Our algorithm is displayed in Algorithm 1.

Algorithm 1 Variational Training of FMs

for $\boldsymbol{X}_B, y_B$ **do**
  for k feature involved in this batch $\boldsymbol{X}_B$ **do**
    Sample $w_k \sim q(w_k)$, $\boldsymbol{v}_k \sim q(\boldsymbol{v}_k)$
  end for
  for k feature involved in this batch $\boldsymbol{X}_B$ **do**
    Update parameters $\mu_k^w, \sigma_k^w, \boldsymbol{\mu}_k^v, \boldsymbol{\sigma}_k^v$ to increase ELBO estimate $\mathcal{L}_b$ for this batch
  end for
end for

4.4 Complexity

The libFM MCMC implementation [Rendle, 2012] has complexity $O(H + kN_z(\boldsymbol{X}))$ per epoch where H is the number of hyperparameters and $N_z(\boldsymbol{X})$ is the number of nonzero entries of training data $\boldsymbol{X}$. We have similar complexity $O(kN_z(\boldsymbol{X}_B))$ but for batch $\boldsymbol{X}_B$ so it is easier to compute.

MCMC does $H + (k+1)M$ samples per epoch where M is the total number of features involved in the training set. We do only $(k+1)F_1(\boldsymbol{X}_B)$ per batch where $F_1(X_B)$ is the number of features involved in batch $\boldsymbol{X}_B$: we have only one hyperparameter to cross-validate, which is σ.

5 Experiments

We conducted experiments on real datasets. Given existing entries, the task consisted in predicting the remaining ones.

5.1 Tasks and Datasets

The datasets are described in Table 2. We distinguish two tasks: regression and classification.

	#users	#items	#ratings	Sparsity
fraction	536	20	10720	0.000
movie100k	944	1683	100000	0.937
movie1M	6041	3707	1000209	0.955
movie10M	69879	10678	10000054	0.987

Table 2: Datasets used for the experiments.

Regression This task is related to collaborative filtering in recommender systems. Ratings are on an ordinal scale, between 1 and 5. Predictions are compared using root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{|D|}\sum_{i\in D}(y_i - \hat{y}_i)^2}.$$

The items may have side information, like tags. For example, in the Movielens 100k dataset[1], all items are mapped to their genres. There are 19 possible genres (Animation, Comedy, etc.). In the Movielens 1M and 10M datasets, no side information is available.

Classification This task is related to item response theory. The entries are binary: 1 means a positive outcome, 0 a negative one. Predictions are compared using area under the curve (AUC). Items may be mapped to side information.

Fraction data represents 536 students answering 20 items related to fraction subtraction, it comes from [DeCarlo, 2010] and is available online. The matrix of answers is fully specified: all students answered all 20 questions. The (i, j) entry is 1 when student i answered item j correctly, 0 otherwise. Items are mapped to knowledge components: skills that are required to be mastered in order to solve the question correctly.

5.2 Models

The main baselines are the libFM[2] implementation of Gibbs sampling (MCMC) for training FMs [Rendle, 2012] that relies on hyperpriors, the libFM implementation of ALS (alternating least squares) with hyperpriors and $\lambda_0 = 0$, $\lambda_1 = \lambda_2 = 1$, respectively the regularization of the global bias, biases and embeddings. Those values are concordant with the parameters learned by the MCMC method.

We compared two versions of the proposed method with the adaptive prior, one that ignores side information besides user and item features (VFM) and one that takes side information into account (VFM+si). For completeness, we also reported the results of maximum a posteriori (MAP) point estimation of FMs with the simplest prior, where the objective is $\sum_i \log p(y_i|x_i,\theta) + \log p(\theta)$, also known as ALS-WR, which is typically encountered in probabilistic matrix factorization for collaborative filtering [Zhou *et al.*, 2008].

On all regression models, the latent dimension was 20, which is consistent with existing work [Yamada *et al.*, 2017]. For the Fraction dataset, we fixed the latent dimension to 3.

[1] https://grouplens.org/datasets/movielens/

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

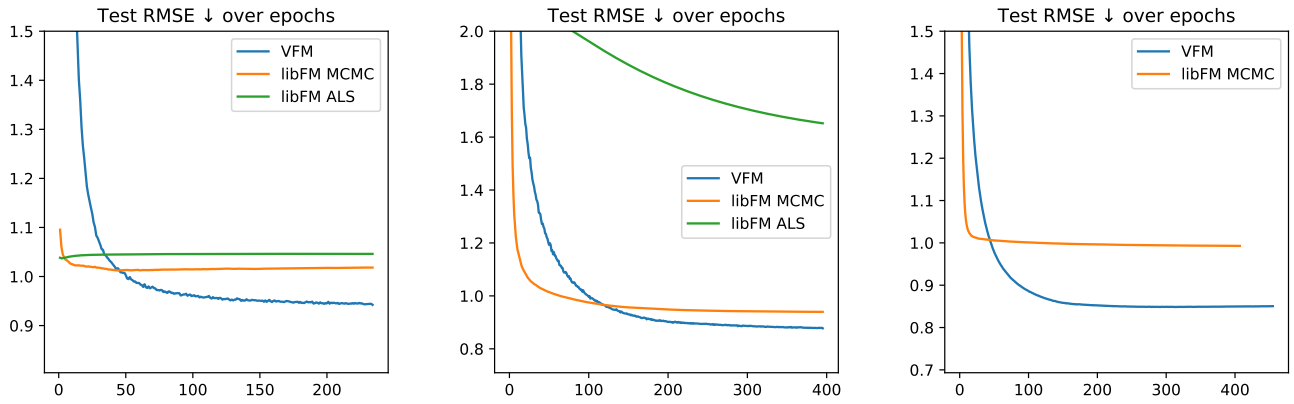

Figure 2: Test RMSE of VFM on all Movielens datasets over training epochs, compared to MCMC and ALS. On the Movielens 10M, ALS has RMSE 1.8 so it was not represented.

5.3 Framework

We split each dataset into a training set with 64% of the ratings, a validation set with 16% of the ratings and a testing set with 20% of the ratings, in order to ensure the ratio 4:1 of training over validation and training + validation over testing. In all experiments we optimized our ELBO estimate using stochastic gradient descent over 10 or 100 minibatches sampled at random. We trained our model using an Adam optimizer [Kingma and Ba, 2014] initialized with $\gamma = 0.01$.

For VFM, VFM+si and MAP, the hyperparameter σ was learned using cross validation. The stopping rule for training was whether the ratio of generalization error on the validation set over the training progress is over 0.20 [Prechelt, 1998]. Then, we refit our model on both training and validation set using the best value of σ, and the stopping rule becomes that the ELBO estimate should not decrease during 4 consecutive steps. Our implementation is written in TensorFlow using TensorFlow Distributions and TensorFlow Probability [Dillon *et al.*, 2017] and is available online on GitHub.

For libFM MCMC, the hyperparameter σ was directly optimized during training. For libFM ALS, we used the same value of σ than the one learned by MCMC.

5.4 Results

The results are reported in Figure 2 and Tables 3 and 4.

Classification On the Fraction dataset, with $d = 3$, see Table 4. VFM has similar performance than MCMC, and MAP has better generalization: 0.90 AUC. The worst performing model is libFM ALS, which may be because there are several hyperparameters to optimize. MCMC, on the other hand, learns the regularization parameter during training.

Regression On all datasets, VFM achieves lower RMSE than MCMC. MAP has lower RMSE than VFM only on the Movielens 100k dataset. MAP needs a higher value of σ (1) to be trained properly, which means, higher uncertainty over the predictions, and weaker regularization.

On the Movielens 10M dataset, our algorithm converges to RMSE 0.849 after 300 epochs, while MCMC struggles to get below 0.992 after 500 epochs.

On the Movielens 100k dataset, VFM+si has a slightly lower performance than VFM that ignores side information (here, the genre of the movies).

Training speed On the Movielens 10M dataset, each epoch takes around 24 seconds for MCMC (C++ libFM implementation), and 13 seconds for VFM (Python code calling C++ operators). This comparison is unfair for our implementation in Python because libFM is entirely written in C++ but still, we are faster on the Movielens 100k and 10M datasets. It may be because we only need to sample $\boldsymbol{w}$ and $\boldsymbol{V}$ using at most $K(d+1)$ samples from the standardized normal distribution while MCMC has to sample the hyperpriors, the priors, the variables, and the outputs as well, and there are 10M of the latter.

We tried the available implementation of [Moody, 2016] of VFM but it took respectively 7 and 68 seconds per epoch to run on the Movielens 100k and 1M datasets with the same batch size, while our implementation took 0.07 s and 1.247 s. The implementation by [Saha *et al.*, 2017] took respectively 1, 5 and 60 seconds per epoch for the Movielens 100k, 1M and 10M datasets.

VFM+si is slower than VFM because of the time required to feed sparse matrices to TensorFlow. We plan to optimize this part.

6 Discussion

On all datasets, VFMs, without any hyperprior, achieve lower RMSE or higher AUC than MCMC which is the best method for training FMs, according to existing published work. With higher sparsity, the difference between both models is particularly visible, which seems to indicate that, although MCMC asymptotically converges to the true distribution, when there are too few observations, it seems easier to learn an approximation of the posterior.

VFMs perform better than MAP when the sparsity is higher, see Figure 2, and reciprocally: on the Fraction dataset, where all entries are known, MAP has AUC 0.90 while all other models have 0.80. Please note that on that dataset, the simplest and adaptive priors are identical, because all features

		FM ALS	FM MCMC	MAP	VFM	VFM+si	OVBFM [Saha *et al.*, 2017]
movie100k	RMSE	**1.046**	**0.991**	**0.931**	**0.942***	**0.961**	**1.001**
	sigma	–	–	1.0	0.5	0.5	1
	stopped at epoch	439	439	145	234	439	100
	time per epoch (s)	0.121	0.141	0.075	0.072	0.541	1
	total time (s)	53.000	62.000	10.811	16.912	237.427	
movie1M	RMSE	**1.635**	**0.938**	**0.933**	**0.879**	**0.877**	**0.846***
	sigma	–	–	1.0	0.2	0.2	1
	stopped at epoch	441	441	192	371	395	100
	time per epoch (s)	1.190	0.737	1.408	1.247	4.413	5
	total time (s)	524.596	325.000	270.268	462.688	1742.938	
movie10M	RMSE	**1.819**	**0.992**	**0.870**	**0.849**	–	**0.828***
	sigma	–	–	1.0	0.2	–	1
	stopped at epoch	407	407	225	371	–	21
	time per epoch (s)	24.154	–	14.490	12.818	–	60
	total time (s)	9830.521	–	3260.297	4755.509	–	

Table 3: Results on all datasets for the regression task. The best results for Bayesian models are annotated with an asterisk.

		FM MCMC	MAP 10	MAP 100	VFM 10	VFM 100	VFM+si 10	VFM+si 100
fraction	AUC	**0.800**	**0.900**	**0.887**	**0.808***	**0.804**	**0.807***	**0.805**
	#batches	–	10	100	10	100	10	100
	sigma	–	0.0	0.0	0.0	0.0	0.0	0.0
	stopped at epoch	500	500	500	500	500	500	500
	time per epoch (s)	–	0.083	0.434	0.093	0.501	0.084	0.399
	total time (s)	–	41.452	217.238	46.291	250.730	41.845	199.265

Table 4: Results on the Fraction dataset for the classification task. The best results for Bayesian models are annotated with an asterisk.

are present the same number of times. MAP may have a better performance than VFM on the Fraction dataset because there are fewer parameters to estimate, and no uncertainty to take into account.

We also tried to compare our results to the other existing variants of VFM. [Moody, 2016] was way slower than ours and used hyperpriors, and [Saha *et al.*, 2017] used the simplest prior and $\sigma = 1$, and was slower than our method on the biggest dataset Movielens 10M. To optimize, we use Adam while [Saha *et al.*, 2017] use Robbins-Monro to adapt their learning rate. We claim that, with our adaptive priors, we recover simpler forms of the unbiased estimates of the lower bound, which induces simpler formulas for updating the parameters. [Saha *et al.*, 2017] got better results than our method on the Movielens 100k and 1M datasets, but with a slower method.

In our experiments, the side information degrades the performance of the model on the Movielens 100k data, and only slightly increases the performance on the Fraction dataset. In existing work such as [Vie, 2018], side information could improve estimation, but it was richer than just a few tags. We plan to conduct more experiments using richer data.

7 Future work

We showed that on sparse datasets, variational training of FMs could result in better training. Our training relies on batches of data, which could be used in online inference of FMs. This is why we plan to do active learning experiments, where ratings are fed one at a time.

Preference elicitation Using VFM we can assume a positive/negative answer from a student, and simulate the corresponding updates, in order to evaluate if a question has high information gain or not (or low reduction of variance), before we ask it. Different information criteria can be used, such as maximum fisher information or maximum expected posterior variance [Liu *et al.*, 2015]. We leave this for future work.

Graph encoding [Berg *et al.*, 2018] have shown that learning on a graph can be described as message-passing. The variational stochastic gradient updates in this paper can also be framed this way. We plan to make more contributions in this direction, where side information could be embedded in the graph.

Normalizing flows We showed that a Gaussian approximation performs well, but it can be replaced with other, more complex families using normalizing flows, as long as the Jacobian is easy to compute [Dinh *et al.*, 2016].

8 Conclusion

In this paper we showed how factorization machines could be trained easily by optimizing a simple variational objective. By recovering an approximation of the posterior distribution over the parameters of the model, one can identify which features need extra data points. This work opens the door to new techniques of active learning (or multi-armed bandits), because now reward functions can be expressed in function of the uncertainty of the model. We hope to see many applications in adaptive education and interactive recommender systems.

References

[Berg *et al.*, 2018] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. Presented at the KDD 2018 Deep Learning Day, 2018.

[Blondel *et al.*, 2015] Mathieu Blondel, Akinori Fujino, and Naonori Ueda. Convex factorization machines. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 19–35. Springer, 2015.

[Blondel *et al.*, 2016] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. Higher-order factorization machines. In *Advances in Neural Information Processing Systems*, pages 3351–3359, 2016.

[Cai, 2010] Li Cai. High-dimensional exploratory item factor analysis by a metropolis–hastings robbins–monro algorithm. *Psychometrika*, 75(1):33–57, 2010.

[DeCarlo, 2010] Lawrence T. DeCarlo. On the analysis of fraction subtraction data: The dina model, classification, latent class sizes, and the q-matrix. *Applied Psychological Measurement*, 2010.

[Dillon *et al.*, 2017] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

[Dinh *et al.*, 2016] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[Hoffman *et al.*, 2013] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kingma and Welling, 2013] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[Kingma and Welling, 2014] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*, 2014.

[Liu *et al.*, 2015] Jingchen Liu, Zhiliang Ying, and Stephanie Zhang. A rate function approach to computerized adaptive testing for cognitive diagnosis. *Psychometrika*, 80(2):468–490, 2015.

[Moody, 2016] Christoph E. Moody. Variational factorization machines. 2016.

[Porteous *et al.*, 2010] Ian Porteous, Arthur U Asuncion, and Max Welling. Bayesian matrix factorization with side information and dirichlet process mixtures. In *AAAI*, 2010.

[Prechelt, 1998] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[Reckase, 2009] Mark D Reckase. Multidimensional item response theory models. In *Multidimensional Item Response Theory*, pages 79–112. Springer, 2009.

[Rendle, 2012] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57:1–57:22, 2012.

[Saha *et al.*, 2017] Avijit Saha, Rishabh Misra, Ayan Acharya, and Balaraman Ravindran. Scalable variational bayesian factorization machine. 2017.

[Vie, 2018] Jill-Jênn Vie. Deep factorization machines for knowledge tracing. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 370–373, 2018.

[Wang *et al.*, 2018] Chao Wang, Qi Liu, Run-ze Wu, Enhong Chen, Chuanren Liu, Xunpeng Huang, and Zhenya Huang. Confidence-aware matrix factorization for recommender systems. In *AAAI*, 2018.

[Yamada *et al.*, 2017] Makoto Yamada, Wenzhao Lian, Amit Goyal, Jianhui Chen, Kishan Wimalawarne, Suleiman A Khan, Samuel Kaski, Hiroshi Mamitsuka, and Yi Chang. Convex factorization machine for toxicogenomics prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1215–1224. ACM, 2017.

[Zhou *et al.*, 2008] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*, pages 337–348. Springer, 2008.