Scaling Factorization Machines to Relational Data

Steffen Rendle
University of Konstanz
78457 Konstanz, Germany
steffen.rendle@uni-konstanz.de

ABSTRACT

The most common approach in predictive modeling is to describe cases with feature vectors (aka design matrix). Many machine learning methods such as linear regression or support vector machines rely on this representation. However, when the underlying data has strong relational patterns, especially relations with high cardinality, the design matrix can get very large which can make learning and prediction slow or even infeasible.

This work solves this issue by making use of repeating patterns in the design matrix which stem from the underlying relational structure of the data. It is shown how coordinate descent learning and Bayesian Markov Chain Monte Carlo inference can be scaled for linear regression and factorization machine models. Empirically, it is shown on two large scale and very competitive datasets (Netflix prize, KDDCup 2012), that (1) standard learning algorithms based on the design matrix representation cannot scale to relational predictor variables, (2) the proposed new algorithms scale and (3) the predictive quality of the proposed generic feature-based approach is as good as the best specialized models that have been tailored to the respective tasks.

1. INTRODUCTION

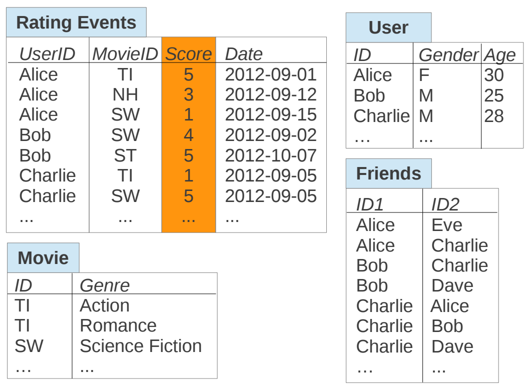
Predictive analytics is an important technique with applications in many fields ranging from business to science. Typically, a predictive model is defined as a function of predictor variables to some target. E.g. *how much* (target) does a *customer* (first predictor variable) like a *product* (second predictor variable). The most common approach is that a data analyst selects some predictor variables (aka *feature engineering*) and applies a machine learning (ML) method to learn the target function from observations of the past. The ML model is then a function of the predictor variables (aka *feature vector*). Many important ML methods are based on this principle incl. linear regression (LR), support vector machines (SVM), decision trees, etc. The runtime of learning and prediction depends on the (sparse) size of the feature vector and is typically linear at best. Nowadays, feature engineering based ML is the dominant technique in predictive analytics. However, if it is applied to relational data, especially involving relations of high cardinality, the feature vectors can grow very large which can make learning and prediction very slow or even infeasible. E.g. to follow the example from above, the friends of a customer might be predictive for his/her taste. Using the variable "friends of a customer" in the feature vector (e.g. for a SVM, LR, etc.) can result in a very long feature vector because all friends (i.e. their IDs) are included in the feature vector.

In this paper, it is shown how prediction and learning algorithms for linear regression and factorization machines can be scaled to predictor variables generated from relational data involving relations of high cardinality. The idea is to make use of repeating patterns over a set of feature vectors. No change is made on the predictive modeling approach and also not on the underlying statistical model. Thus the proposed algorithms learn the same parameters and make the same predictions but with a much lower runtime complexity. The paper starts with linear regression as it is one of the best-known ML models and still achieves high prediction accuracy in competitive problems (e.g. KDDCup 2010 [23]). Moreover the idea of scaling is easier to understand for this basic model first. The main contribution is scaling factorization machines [12] which is a generic factorization model including among others matrix factorization [17], SVD++ [3], PITF [15], timeSVD++ [5], etc. Factorization models have shown great predictive performance in very competitive machine learning problems including the Netflix prize[1], recent KDDCups[2] (2010,2011,2012) as well as other prediction challenges (e.g. 'What Do You Know?' Challenge[3], EMI Music Hackathon[4]). For both models, scaling is shown for coordinate descent (CD) learning and for a Markov Chain Monte Carlo (MCMC) Gibbs sampler. CD is one of the most effective point estimators [2] and MCMC a state-of-the-art Bayesian inference method.

From a practical point of view, the proposed algorithms allow to handle predictive modeling as usual: defining predictor variables (also variables from relations of high cardinality) by feature engineering and applying a feature-vector-based ML algorithm. Internally, the algorithms make use of the repeating patterns stemming from the relational structure of the data to largely speed up computation.

Proceedings of the VLDB Endowment, Vol. 6, No. 5
Copyright 2013 VLDB Endowment 2150-8097/13/03... $ 10.00.

[1]`http://www.netflixprize.com/`
[2]`http://www.sigkdd.org/kddcup/`
[3]`http://www.kaggle.com/c/WhatDoYouKnow`
[4]`http://www.kaggle.com/c/MusicHackathon`



Rating Events

UserID	MovieID	Score	Date
Alice	TI	5	2012-09-01
Alice	NH	3	2012-09-12
Alice	SW	1	2012-09-15
Bob	SW	4	2012-09-02
Bob	ST	5	2012-10-07
Charlie	TI	1	2012-09-05
Charlie	SW	5	2012-09-05
...	...	...	...

User

ID	Gender	Age
Alice	F	30
Bob	M	25
Charlie	M	28
...	...	

Friends

ID1	ID2
Alice	Eve
Alice	Charlie
Bob	Charlie
Bob	Dave
Charlie	Alice
Charlie	Bob
Charlie	Dave
...	...

Movie

ID	Genre
TI	Action
TI	Romance
SW	Science Fiction
...	...

Figure 1: Example database from a movie community.

2. RELATIONAL PREDICTIVE MODELING

In the following, first the standard feature based approach of predictive modeling is shortly recapitulated. Then the limitations of this approach for scaling to data with relational structure are described.

2.1 Predictive Modeling

The most common approach to predictive modeling is to select a set of variables that are assumed to be predictive for a task. Throughout this work, for illustration, the task of predicting rating scores for users on movies is used. E.g. for the data in Figure 1, a data analyst might assume that the user ID, movie ID, date, gender, age, the set of movie genres, the set of friends of a user and the set of all the movies a user has ever watched are predictive for estimating the rating score[5]. The task of machine learning is to learn the functional dependency of the predictor variables on the target (Figure 2(a)).

Learning is based on observed samples of the functional dependency – called *training data*. Each sample can be written as a vector of variable assignments for the predictor variables and the target. Each row in Figure 2(b) shows one of the observed combinations of predictor variable values and the observed target value. E.g. the first row represents that *Alice* who is a *30* year old *female* has watched *Titanic*, *Notting Hill* and *Star Wars* and has *Eve* and *Charlie* as friends rated *Titanic* which is an *Action* and *Romance* movie with *5* stars. A variable assignment of predictor variables is called a *feature vector*. The process of selecting and generating predictor variables is called *feature engineering*.

The process sketched so far is a very generic one and is followed by most of the standard machine learning approaches, incl. linear regression, support vector machines (SVM), decision trees, etc. The difference between the machine learning methods lies in the type of functional dependency that is assumed (e.g. for a linear regression a linear dependency is assumed and for SVMs a linear dependency in a projected space is assumed), optimization aspects such as regularization and the particular learning algorithms.

So far, there is no restriction on the domain (e.g. numerical, categorical, string, graph) of each variable in the feature vector. For a generic formalization, many machine learning methods rely on a numeric vector representation of the variables (either explicitly or implicitly). Also in the remainder of this work, it is assumed that there are p numeric predictor variables, i.e. each case can be represented as a vector $\mathbf{x} \in \mathbb{R}^p$. E.g. for encoding a categorical/nominal variable with m levels, the standard approach is to map the variable to m numerical variables and use a binary encoding. Figure 2(c) shows one possibility of how to represent the training data of 2(b) with numeric variables. In total, the n cases each represented by a feature vector $\mathbf{x} \in \mathbb{R}^p$ can be seen as a matrix $X \in \mathbb{R}^{n \times p}$ which is typically called *design matrix*. The n prediction targets can be represented as an n dimensional vector $\mathbf{y} \in \mathbb{R}^n$ for regression (or $\mathbf{y} \in \{-,+\}^n$ for binary classification).

In general, a highly desirable property of a learning algorithm is a linear runtime complexity in the size of the design matrix X. Often the data is sparse, i.e. contains many 0 values. Let $N_Z(X)$ denote the number of non-zeros in a matrix X. Sparse learning algorithms can make use of the non-zeros in the training data and a desirable property is a runtime complexity linear in $N_Z(X)$. In the following, it is discussed that in relational data even a linear runtime complexity in $N_Z(X)$ can be infeasible.

2.2 Relational Predictive Modeling

The standard approach of predictive modeling to select predictor variables and to learn the dependency on a target is also reasonable for predictive problems on relational data. Actually, the example presented in Figure 2 involves relational data. This means the concepts of feature engineering, feature vectors and design matrices are also applied here and any standard machine learning method can be used.

However, relational data can lead to very large feature vectors/ design matrices with redundant information. Figure 3(a) illustrates this on the running example. Investigating the first case (first row), one can see that parts of the feature vector are repeated in other rows. E.g. the parts highlighted in red reappear in the second and third case. The parts highlighted in blue appear also in the 6th case. It is clear that these repeating patterns stem from the relational structure of the predictor variables: The red block corresponds to the variables describing user *Alice*, the blue block from variables describing movie *Titanic*. Whenever a case uses predictor variables describing *Alice*, the feature vector will include all of *Alice*'s descriptors including age, gender, friends, etc. Regarding the predictive model, this is correct as the case depends on all the predictor variables (no matter from where they come from). However, the design matrix X can get very large and intractable. E.g. in social networks each user often has hundreds of friends, using these friends as predictor variable for the taste of the user is reasonable but will result in a long predictor vector with many non-zero entries (for each friend a non-zero variable) for describing the case. In the evaluation, examples of the size $N_Z(X)$ for a selection of predictor variables in real-world datasets are shown (Figure 5, Table 1). For any machine learning method that relies on feature vectors/ design matrices – which are most ML methods – relational datasets can result in very large design matrices that are infeasible for standard algorithms, even if the algorithm has a linear runtime complexity in $N_Z(X)$. The contribution of this work are new learning algorithms that make use of repeating patterns to speed up learning.

[5]Note that the selected variables are not limited to explicitly stated columns in the original data, but can also be derived.

(a) Predictive function

score : UserID × MovieID × Date × UserGender × UserAge × MovieGenres × UserFriends × ItemsWatched → $\mathbb{R}$

(b) Training Data for Predictive Function

UserID	*MovieID*	*Date*	*Gender*	*Age*	*Genres*	*Friends*	*ItemsWatched*	*Score*
Alice	TI	2012-09-01	F	30	{A,R}	{E,C}	{TI,NH,SW}	5
Alice	NH	2012-09-12	F	30	{C,R}	{E,C}	{TI,NH,SW}	3
Alice	SW	2012-09-15	F	30	{S,A}	{E,C}	{TI,NH,SW}	1
Bob	SW	2012-09-02	M	25	{S,A}	{C,D}	{SW,ST}	4
Bob	ST	2012-10-07	M	25	{S}	{C,D}	{SW,ST}	5
Charlie	TI	2012-09-05	M	28	{A,R}	{A,B,D}	{TI,SW}	1
Charlie	SW	2012-09-05	M	28	{S,A}	{A,B,D}	{TI,SW}	5
...	...	...	...	...	...	...	...	...

(c) Training Data in Numeric Format (*Design Matrix*)

UserID	*MovieID*	*Date*	*Gender*	*Age*	*Genres*	*Friends*	*ItemsWatched*	*Score*
1 0 0	1 0 0 0	1	1 0	30	.5 .5 0 0	0 0 .5 0 .5	.3 .3 .3 0	5
1 0 0	0 1 0 0	12	1 0	30	0 .5 .5 0	0 0 .5 0 .5	.3 .3 .3 0	3
1 0 0	0 0 1 0	15	1 0	30	.5 0 0 .5	0 0 .5 0 .5	.3 .3 .3 0	1
0 1 0	0 0 1 0	2	0 1	25	.5 0 0 .5	0 0 .5 .5 0	0 0 .5 .5	4
0 1 0	0 0 0 1	37	0 1	25	0 0 0 1	0 0 .5 .5 0	0 0 .5 .5	5
0 0 1	1 0 0 0	5	0 1	28	.5 .5 0 0	.3 .3 0 .3 0	.5 0 .5 0	1
0 0 1	0 0 1 0	5	0 1	28	.5 0 0 .5	.3 .3 0 .3 0	.5 0 .5 0	5
...	...	...	...	...	...	...	...	...
A B C	*TI NH SW ST*		*F M*		*A R C S*	*A B C D E*	*TI NH SW ST*	

corresponding levels of categorical variables

Figure 2: Standard approach in predictive modeling (illustrated for data of figure 1): (a) The predicted target (here *score*) is a function over selected variables. (b) Training data for the function is generated from the data base. (c) Many machine learning models work on a numeric/ real valued encoding of the variables. This numeric representation of n cases with p variables is a *design matrix* $X \in \mathbb{R}^{n \times p}$, the associated targets (here *score* variable) is a vector $\mathbf{y} \in \mathbb{R}^n$ of length n (here $n = 7, p = 24$).

2.3 Block Structure

Repeating patterns in X can be formalized by a condensed block structure representation $\mathcal{B}$.

DEFINITION 1 (BLOCK STRUCTURE (BS)). *Let $\mathcal{B} = \{B_1, B_2, \ldots\}$ be a set of <u>blocks</u>, where each block $B = (X^B, \phi^B)$ consists of a <u>design matrix</u> $X^B \in \mathbb{R}^{n^B \times p^B}$ and a <u>mapping</u> $\phi^B : \{1, \ldots, n\} \to \{1, \ldots, n^B\}$ from rows in the original design matrix X to rows within X^B. $\mathcal{B}$ is a <u>block structure representation</u> of X iff for all rows i:*

$$\mathbf{x}_i \equiv (x^{B_1}_{\phi^{B_1}(i),1}, x^{B_1}_{\phi^{B_1}(i),2}, \ldots, x^{B_2}_{\phi^{B_2}(i),1}, x^{B_2}_{\phi^{B_2}(i),2}, \ldots) \quad (1)$$

This means a block structure representation $\mathcal{B}$ of X represents exactly the same design matrix as X. From $\mathcal{B}$, one can reconstruct X by concatenating the corresponding rows of the design matrices $X^{B_1}, X^{B_2}, \ldots$ using the mappings (i.e. applying eq. (1) to each of the n rows).

2.3.1 Simplified Notation

To shorten notation, the index B is dropped from the mapping ϕ^B whenever it is clear which block the mapping belongs to: e.g. $x^B_{\phi^B(i),l}$ is written as $x^B_{\phi(i),l}$. Secondly, the column ordering of variables in the blocks is neglected and an implicit mapping between column spaces X and X^B is assumed: e.g. when referring to the block variable that corresponds to $x_{i,l}$, the column index l is used as well $x^B_{\phi(i),l}$.

2.3.2 Example

Figure 3(b) shows an example how the design matrix can be grouped in three blocks B_1, B_2, B_3 where the first block represents the user, the second the movie and the third the time. Each block consists of a design matrix and a mapping. E.g. X^{B_1} is the design matrix with all the predictor variables of the user and ϕ^{B_1} the mapping from the case to the row of the design matrix of the block. E.g. the fourth case of the original design matrix X can be obtained by concatenating the second row of X^{B_1}, the third row of X^{B_2} and the fourth row of X^{B_3}, i.e. $\mathbf{x}_4 = (\mathbf{x}^{B_1}_{\phi(4)}, \mathbf{x}^{B_2}_{\phi(4)}, \mathbf{x}^{B_3}_{\phi(4)}) = (\mathbf{x}^{B_1}_2, \mathbf{x}^{B_2}_3, \mathbf{x}^{B_3}_4)$.

2.3.3 Complexity of BS

The complexity of a block structure representation is the sum of the complexity of all blocks plus the size of the mappings

$$N_Z(\mathcal{B}) := |\mathcal{B}|\, n + \sum_{B \in \mathcal{B}} N_Z(X^B) \quad (2)$$

If the design matrix X has repeating patterns, $N_z(\mathcal{B}) \ll N_z(X)$. Figure 5 and Table 1 show some examples for the reduction of the complexity in two real-world datasets.

2.3.4 Contribution of this Work

The contribution of this work is to develop learning and prediction algorithms that make use of the repeating pattern and have a linear runtime complexity in $N_Z(\mathcal{B})$. The presented algorithms will be exact, i.e. they give analytically the same solution as with the standard learning approaches – but with a lower runtime complexity. Two state-of-the art learning algorithms, Gibbs sampling and coordinate descent, are scaled for linear regression and factorization machines.

Scaling algorithms to a runtime complexity of $\mathcal{O}(N_Z(\mathcal{B}))$ is not trivial. It means that the algorithm is not allowed

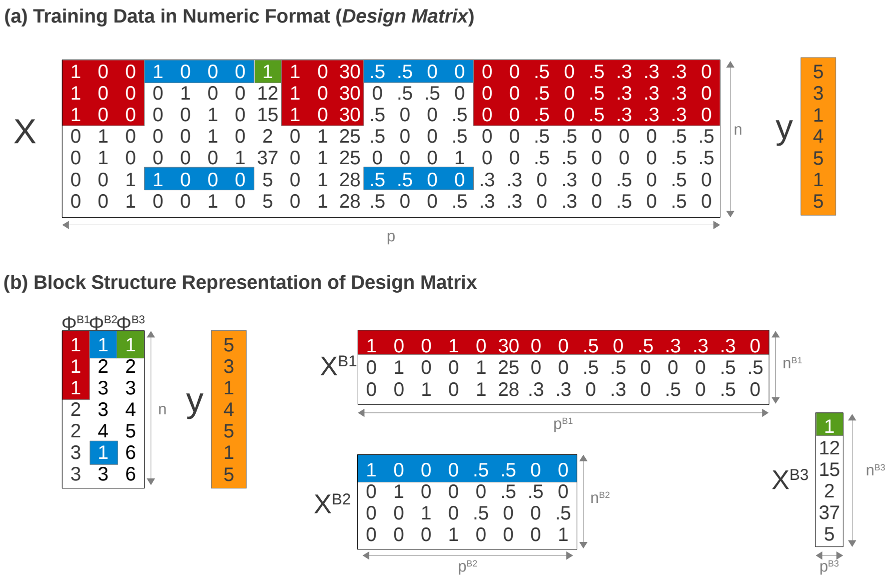


Figure 3: (a) In relational domains, design matrices X have large blocks of repeating patterns (example from Figure 2). (b) Repeating patterns in X can be formalized by a block notation (see section 2.3) which stems directly from the relational structure of the original data. Machine learning methods have to make use of repeating patterns in X to scale to large relational datasets.

to process feature vectors in the original space, i.e. by concatenating the vectors (eq. 1) – not even on-the-fly. On-the-fly concatenating would reduce the *memory* complexity to $\mathcal{O}(N_Z(\mathcal{B}))$ but not the *runtime* complexity. In analogy to compression: if $\mathcal{B}$ is regarded as a compression of X, then a linear runtime complexity in $\mathcal{B}$ means that the algorithms have to do all calculations without decompressing the data at all (not even on-the-fly or partially).

3. SCALING LINEAR REGRESSION

To highlight the basic ideas of scaling learning algorithms, the well-known linear regression model is discussed first.

3.1 Standard Linear Regression

The linear regression (LR) model for the i-th row/ feature vector $\mathbf{x}_i$ of an $n \times p$ design matrix X is

$$\hat{y}(\mathbf{x}_i) = w_0 + \sum_{j=1}^{p} w_j \, x_{i,j}$$

where $\Theta = \{w_0, w_1, \dots, w_p\}$ are the model parameters. Predicting all n cases can be implemented in $\mathcal{O}(N_Z(X))$ by regarding only the non-zero elements in the design matrix.

There are several ways to learn a LR model. The traditional one for least-squares regression is based on solving a $p \times p$ system of linear equations (typically in $\mathcal{O}(p^3)$ time). Iterative approaches scale better to a large number of predictor variables p and coordinate descent (CD) [2] is one of the most efficient iterative algorithms. The CD algorithm starts with an initial (random) guess of Θ, then iterates over each model parameter $w_l \in \Theta$ and performs an update

$$w_l \leftarrow \frac{w_l \sum_{i=1}^n x_{i,l}^2 + \sum_{i=1}^n x_{i,l} \, e_i}{\sum_{i=1}^n x_{i,l}^2 + \lambda_l} \tag{3}$$

where $\lambda_l \in \mathbb{R}_+$ is a predefined regularization constant for the l-th model parameter and $e_i = y_i - \hat{y}(\mathbf{x}_i)$ is the i-th residual ($i \in \{1, \dots, n\}$) which should be precomputed and has to be updated during learning. After a parameter changes from w_l to w_l^* (let $\Delta_l = w_l - w_l^*$ be the difference), each residual changes and can be updated in constant time $e_i \leftarrow e_i + \Delta_l \, x_l$.

This process of updating each model parameter w_l (and updating precomputed residuals) is iterated over all model parameters until convergence. The runtime of CD is dominated (see eq. (3)) by computing the two quantities:

$$\sum_{i=1}^n x_{i,l}^2, \qquad \sum_{i=1}^n x_{i,l} \, e_i. \tag{4}$$

By caching residuals e, each full iteration (i.e. over all Θ) of CD can be implemented efficiently in $\mathcal{O}(N_Z(X))$.

Whereas CD is a point estimator, i.e. the result is a single value for each parameter w_l, Bayesian inference can include uncertainty into the model. Bayesian inference typically improves the prediction quality and also allows to infer regularization values automatically. Bayesian inference with Markov chain Monte Carlo (MCMC), here Gibbs sampling with block size of one, is related to CD. In this case, the Gibbs sampler updates the model parameters by drawing w_l from its conditional posterior distribution:

$$w_l \sim \mathcal{N}\left(\frac{\alpha \, w_l \sum_{i=1}^n x_{i,l}^2 + \alpha \sum_{i=1}^n x_{i,l} \, e_i + \mu_l \, \lambda_l}{\alpha \sum_{i=1}^n x_{i,l}^2 + \lambda_l}, \frac{1}{\alpha \sum_{i=1}^n x_{i,l}^2 + \lambda_l} \right) \tag{5}$$

where α is the precision of the likelihood and μ_l is the mean and λ_l the precision of the normal prior distribution over w_l. These three hyperparameters are found automatically by Gibbs sampling – see [13] for details. As it can be seen, for

Gibbs sampling of a model parameter w_l (eq. 5), the same quantities (eq. 4) have to be computed as for CD (eq. 3), so this work can focus on efficient computation of eq. (4). Both CD and Gibbs sampling can be extended for classification using a link function (also called generalized linear model). Even with a link function, the main computational parts (i.e. eq. (4)) remain the same – (see e.g. [13] for details). Thus all results presented in this paper can be used as well for classification with CD and Gibbs.

3.2 Scaling Linear Regression to Block Structures

For scaling LR, both learning and prediction has to make use of the block structure representation. For achieving a linear runtime in $N_Z(\mathcal{B})$, it is has to be shown how the algorithms can work on the BS representation without (not even on-the-fly) concatenating the block vectors (i.e. eq. 1).

With respect to notation, remind that in block notation (eq. 1), the feature vector $\mathbf{x}_i$ is split into $\mathbf{x}^{B_1}_{\phi(i)}, \mathbf{x}^{B_2}_{\phi(i)}, \ldots$. The same notation will be used for LR model parameters (here $\mathbf{w}$) as well, i.e. $\mathbf{w}$ is split into $\mathbf{w}^{B_1}, \mathbf{w}^{B_2}, \ldots$. Both formulations are mathematically equivalent – however when using a block specific feature vector $\mathbf{x}^B$, it is easier to think of block specific model parameters $\mathbf{w}^B$ as well.

3.2.1 Prediction

Let $\mathcal{B}$ be the block structure representation of design matrix X. With eq. (1), the LR model equation for the i-th case $\mathbf{x}_i$ can be rewritten as

$$\hat{y}(\mathbf{x}_i) = w_0 + \sum_{B \in \mathcal{B}} \sum_{j=1}^{p^B} w_j^B \, x^B_{\phi(i),j} = w_0 + \sum_{B \in \mathcal{B}} q^B_{\phi(i)}$$

where

$$q_i^B = \sum_{j=1}^{p^B} w_j^B \, x^B_{i,j}, \quad \forall i \in \{1, \ldots, n^B\} \tag{6}$$

This directly shows how n cases can be efficiently predicted: (i) compute q^B in $\mathcal{O}(N_Z(X^B))$ for each block B, (ii) predict for each of the n cases $\hat{y}$ using $\mathbf{q}$ in $\mathcal{O}(n\,|\mathcal{B}|)$. To summarize, predicting n cases from the design matrix X can be scaled to $\mathcal{O}(N_Z(\mathcal{B}))$ time (see definition of $N_Z(\mathcal{B})$ in eq. (2)).

3.2.2 Learning

Exploiting the block structure for learning is more complicated. Remind that learning (CD or Gibbs) requires to compute eq. (4) for each model parameter w_l. The goal is to rearrange the sums in eq. (4) to iterate over the n^B elements of the block instead of the n elements in the original design matrix. The basic idea is that for any function f the sum can be rewritten[6]:

$$\sum_{i=1}^{n} f(i) = \sum_{i=1}^{n^B} \sum_{j=1}^{n} \delta(\phi^B(j) = i) f(j). \tag{7}$$

Now a decomposition of f has to be found that allows to replace the inner sum (over n) by a term depending only on i with constant computation time. This is shown in the following for eq. (4) of LR and later for factorization machines.

[6]δ is the indicator function, i.e. $\delta(\text{true}) = 1, \delta(\text{false}) = 0$.

Algorithm 1 LinearRegressionBS($\mathbf{y}$, $\mathcal{B}$)

```
 1: $\Theta \sim \mathcal{N}(0, \sigma^2)$
 2: for $i \in \{1, \ldots, \#\text{Iter}\}$ do
 3:     $\hat{\mathbf{y}} \leftarrow$ predict all cases                  $\triangleright \mathcal{O}(N_z(\mathcal{B}))$
 4:     $\mathbf{e} \leftarrow \mathbf{y} - \hat{\mathbf{y}}$                                  $\triangleright \mathcal{O}(n)$
 5:     update hyperparameter $\alpha$               $\triangleright \mathcal{O}(n)$, see [13]
 6:     for $B \in \mathcal{B}$ do
 7:         update hyperparameter $\mu_w^B, \lambda_w^B$   $\triangleright \mathcal{O}(p^B)$, see [13]
 8:         cache: init $\mathbf{e}^B$ and unsync $\mathbf{e}$                $\triangleright \mathcal{O}(n)$
 9:         for $l \in \{1, \ldots, p^B\}$ do
10:             update $w_l^B$ from eq. (5) using eqs. (8,10)
11:             cache: update $\mathbf{e}^B$, $\mathbf{q}^B$
12:         end for
13:         cache: sync $\mathbf{e}$                                 $\triangleright \mathcal{O}(n)$
14:     end for
15: end for
```

For the first quantity of eq. (4), $f(i) = x_{i,l}^2$

$$\sum_{i=1}^{n} x_{i,l}^2 = \sum_{i=1}^{n^B} \sum_{j=1}^{n} \delta(\phi^B(j) = i) x_{j,l}^2 = \sum_{i=1}^{n^B} (x^B_{i,l})^2 \#_i^B \tag{8}$$

with the constant

$$\#_i^B = \sum_{j=1}^{n} \delta(\phi^B(j) = i) \tag{9}$$

which counts how many mappings go to row i of block B.

For the second quantity of eq. (4), $f(i) = x_{i,l}\, e_i$

$$\sum_{i=1}^{n} x_{i,l}\, e_i = \sum_{i=1}^{n^B} x^B_{i,l} e^B_i, \quad e_i^B := \sum_{j=1}^{n} \delta(\phi^B(j) = i)\, e_j. \tag{10}$$

Here e_i^B is not a constant but a block depending sum of those residuals that are mapped by ϕ^B from X to the i-th row of the block. e_i^B has to be kept in sync with the model parameters. After the value of a model parameter changes from w_l to w_l^* (Δ_l is the difference), the sum of residuals can be updated by $e_i^B \leftarrow e_i^B + \Delta_l\, x^B_{i,l}\, \#_i^B$.

In total, for each model parameter w_l^B, eq. (4) can be computed in $\mathcal{O}(\sum_{i=1}^{n^B} \delta(x^B_{i,l} \neq 0))$ time and updating e_i^B has the same complexity. Thus, for all p^B model parameters in block B, the computational complexity is $\mathcal{O}(N_Z(X^B))$. Additionally, when starting to update model parameters from a block B, $\mathbf{e}^B$ has to be initialized which is in $\mathcal{O}(n)$ time if the global residuals $\mathbf{e}$ are present. The global residuals $\mathbf{e}$ cannot be kept in sync after each model parameter update without sacrificing the runtime complexity, because there are n residuals. However, global residuals $\mathbf{e}$ are not of interest *while* updating parameters within a block but only *before* and *after* all parameters in a block have been updated. Thus, the residuals $\mathbf{e}$ can be unsynchronized from all parameters of the block before model parameters are updated ($e_i \leftarrow e_i + q^B_{\phi(i)}$) and resynchronized afterward ($e_i \leftarrow e_i - q^B_{\phi(i)}$). This has per block a complexity of $\mathcal{O}(n)$.

A fast learning algorithm is sketched in algorithm 1. Here, Gibbs sampling is presented[7]. As discussed above, the algorithm has a linear complexity in $\mathcal{O}(N_Z(\mathcal{B}))$ per iteration

[7]For CD, updates on model paramaters (line 10) should be performed by eq. (3) instead of eq. (5) and the updates on the hyperparameters α, λ, μ should be removed (lines 5,7).

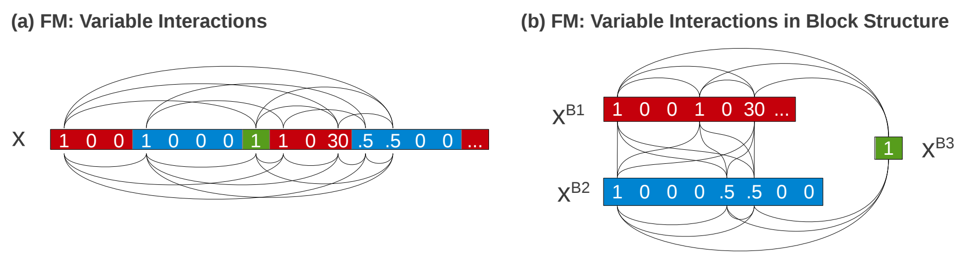


Figure 4: (a) A factorization machine estimates interactions between all variable pairs in the feature vector x (here only interactions between non-zero elements are shown). (b) In the BS representation of the feature vector x, interactions cross different blocks.

and thus makes perfect use of the repeating pattern in X while being analytically exact.

For the data structures involved, X^B should be stored in a sparse representation of its transpose $(X^B)^t$ – i.e. for each column l of X^B an array/ static vector of the non-zero entries (row index) with the corresponding value. This is important in computing eq. (4) because given the column l of a predictor variable, the sum has to iterate over all non-zero entries – which is the l-th row of the sparse transpose. In general, storing only the transpose of X^B is sufficient. For the caches $\mathbf{e} \in \mathbb{R}^n$, $\mathbf{e}^B \in \mathbb{R}^{n^B}$, $\mathbf{q}^B \in \mathbb{R}^{n^B}$, $\#^B \in \mathbb{R}^{n^B}$ as well as for the mappings $\phi^B \in \mathbb{N}^n$ standard arrays/ static vectors can be used.

4. SCALING FACTORIZATION MACHINES

For LR, computation could be divided into blocks because the model does not contain interactions between variables of different blocks. FMs learn variable interactions (e.g. the interaction between age and item category, see Figure 4) which makes efficient computation more challenging.

4.1 Standard Factorization Machines

Like polynomial regression, non-linear SVMs or decision trees, factorization machines (FM) include interactions between predictor variables. This allows to learn more complex functions (esp. non-linear ones) than with linear regression. For example FMs can learn that young customers like action movies and dislike documentaries, whereas the preferences of old customers are the opposite. LR can only model single effects, i.e. young people rate higher in general or action movies are rated low, but not the interaction between two variables.

A second-order FM for the i-th feature vector $\mathbf{x}_i$ of the $n \times p$ design matrix X is defined [13] as

$$\hat{y}(\mathbf{x}_i) := w_0 + \sum_{j=1}^{p} w_j\, x_{i,j} + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_{i,j}\, x_{i,j'} \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle \tag{11}$$

where $\langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle$ models the effect of the variable interaction $x_{i,j}\, x_{i,j'}$ with the dot product of two k-dimensional latent vectors:

$$\langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle = \sum_{f=1}^{k} v_{j,f}\, v_{j',f}. \tag{12}$$

The model parameters of an FM are $\Theta = \{w_0, w_1, \dots, w_p, v_{1,1}, \dots v_{p,k}\}$. According to [12], eq. (11) can be computed efficiently in $\mathcal{O}(k\, N_Z(\mathbf{x}))$ as it is equivalent to

$$\hat{y}(\mathbf{x}_i) = w_0 + \sum_{j=1}^{p} w_j\, x_{i,j} + \frac{1}{2} \sum_{f=1}^{k} \left[\left(\sum_{j=1}^{p} v_{j,f}\, x_{i,j} \right)^2 - \sum_{j=1}^{p} v_{j,f}^2\, x_{i,j}^2 \right]. \tag{13}$$

This makes computing all n cases of a design matrix possible in $\mathcal{O}(k\, N_Z(X))$.

An appealing property of an FM is multilinearity in each model parameter $\theta \in \Theta$:

$$\hat{y}(\mathbf{x}) = g_\theta(\mathbf{x}) + \theta\, h_\theta(\mathbf{x}) \quad \forall \theta \in \Theta \tag{14}$$

where g and h are independent of the value of θ and

$$h_\theta(\mathbf{x}) = \frac{\partial \hat{y}(\mathbf{x})}{\partial \theta} = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_l, & \text{if } \theta \text{ is } w_l \\ x_l \sum_{j \neq l} v_{j,f}\, x_j, & \text{if } \theta \text{ is } v_{l,f} \end{cases} \tag{15}$$

The definition of g_θ is skipped as it is never used directly.

4.1.1 *FM Learning*

Coordinate descent and Gibbs sampling can also be applied for inference with FMs. For CD, a model parameter $\theta \in \Theta$ is updated by the equation:

$$\theta \leftarrow \frac{\theta \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)^2 + \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)\, e_i}{\sum_{i=1}^{n} h_\theta(\mathbf{x}_i)^2 + \lambda_\theta}. \tag{16}$$

Again $\lambda_\theta \in \mathbb{R}_+$ is a regularization constant and e_i the i-th residual.

Similarly, Bayesian inference can be performed by sampling values for θ from its conditional posterior distribution

$$\theta \sim \mathcal{N}\left(\frac{\alpha\, \theta \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)^2 + \alpha \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)\, e_i + \mu_\theta\, \lambda_\theta}{\alpha \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)^2 + \lambda_\theta}, \frac{1}{\alpha \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)^2 + \lambda_\theta} \right). \tag{17}$$

Again α is the precision of the likelihood, μ_θ the prior mean and λ_θ the prior precision of θ.

Comparing these update rules for CD learning and Gibbs sampling, one can see that they are dominated by the computation of the two quantities

$$\sum_{i=1}^{n} h_\theta(\mathbf{x}_i)^2, \qquad \sum_{i=1}^{n} h_\theta(\mathbf{x}_i)\, e_i. \tag{18}$$

For the w_l parameters, these quantities are the same as in linear regression. For second order parameters in FMs, i.e. for model parameters $v_{l,f}$, efficient computation of these quantities requires additional precomputation of the terms $q_{i,f} := \sum_{j=1}^{p} v_{j,f}\, x_{i,j}$ [13]. This allows to compute $h_{v_{l,f}}$ in constant time $\mathcal{O}(1)$. Like residual caches e_i, also $q_{i,f}$ have to be kept in sync after the value of a model parameter $v_{l,f}$ changes by $\Delta_{l,f}$: $q_{i,f} \leftarrow q_{i,f} - \Delta_{l,f}\, x_{i,l}$.

In [13] it is shown that a full iteration on all model parameters Θ can be done in $\mathcal{O}(k\, N_Z(X))$ for CD learning and Gibbs sampling of FMs.

4.2 Scaling Factorization Machines to Block Structures

In the following, it is shown how an FM can make use of repeating patterns in the feature vectors, such that the learning and prediction runtime of the proposed algorithm is linear in the complexity of the BS representation of X: $\mathcal{O}(k\, N_z(\mathcal{B}))$. As noted before, this is more complicated than for LR because FMs contain all pairwise interactions between all predictor variables. Nevertheless due to factorization $V\,V^t$ (eq. 12) of the interaction matrix, a fast computation is possible.

4.2.1 Efficient Prediction

For predicting n cases using the BS representation $\mathcal{B}$ of the design matrix X, the FM eq. (13) can be rewritten:

$$\hat{y}(\mathbf{x}_i) = w_0 + \sum_{B\in\mathcal{B}} q^B_{\phi(i)} + \frac{1}{2}\sum_{f=1}^{k}\left[\left(\sum_{B\in\mathcal{B}} q^B_{\phi(i),f}\right)^2 - \sum_{B\in\mathcal{B}} q^{B,S}_{\phi(i),f}\right] \tag{19}$$

with the caches

$$q^B_{i,f} := \sum_{j=1}^{p^B} v^B_{j,f}\, x^B_{i,j}, \qquad q^{B,S}_{i,f} := \sum_{j=1}^{p^B} (v^B_{j,f}\, x^B_{i,j})^2. \tag{20}$$

Computing the caches $\mathbf{q}^B \in \mathbb{R}^{n^B\times k}$ and $\mathbf{q}^{B,S} \in \mathbb{R}^{n^B\times k}$ is in $\mathcal{O}(k\, N_Z(X^B))$. Computing the n predictions with eq. (19) is in $\mathcal{O}(k\, n\, |\mathcal{B}|)$. The total runtime for predicting n cases is $\mathcal{O}(k\, N_Z(\mathcal{B}))$.

As it can be seen, even though an FM contains interactions between variables of different blocks, block specific parts $\mathbf{q}^B$ and $\mathbf{q}^{B,S}$ can be isolated and precomputed which results in an efficient computation.

With a trivial implementation, the space requirements for the caches are in $\mathcal{O}(n + k\sum_{B\in\mathcal{B}} n^B)$. However, by computing the prediction and caches layer by layer (i.e. first $\mathbf{w}$. then $\mathbf{v}_{\cdot,1}$, then $\mathbf{v}_{\cdot,2}$, etc.) only the caches for one layer have to be stored and the space requirement drops to $\mathcal{O}(n+\sum_{B\in\mathcal{B}} n^B)$.

4.2.2 Efficient Learning

For CD learning and Gibbs sampling (eq. 17), the computational bottlenecks are the computation of the sums in eq. (18). To make use of the repeating patterns in X, these sums have to be rewritten such that computing the whole sum is in $\mathcal{O}(n^B)$ instead of $\mathcal{O}(n)$. This is shown in the following for a model parameter $v^B_{l,f}$ of the second order interactions – see LR (section 3.2) for first order effects (w^B_l). Again, the basic idea is to apply eq. (7) and to reformulate the function $f(i)$ such that it depends only on the block. This will require several caches for which efficient updates have to be derived.

For the first quantity in eq. (18), $f(i) = h_{v_{l,f}}(\mathbf{x}_i)^2$. Applying the definition of h_θ (eq. 15), this allows to rewrite the sum in block specific parts

$$\sum_{i=1}^{n} h_{v_{l,f}}(\mathbf{x}_i)^2 = \sum_{i=1}^{n^B}\Big[\#^B_i (h^B_{v_{l,f}}(\mathbf{x}^B_i))^2 + 2c^B_{i,f} x^B_{i,l} h^B_{v_{l,f}}(\mathbf{x}^B_i) + (x^B_{i,l})^2 c^{B,S}_{i,f}\Big] \tag{21}$$

where

$$h^B_{v_{l,f}}(\mathbf{x}^B_i) := x^B_{i,l}\left(q^B_{i,f} - v^B_{l,f} x^B_{i,l}\right),$$

and the new caches

$$c^B_{i,f} := \sum_{j=1}^{n} \delta(\phi^B(j) = i) \sum_{B'\in(\mathcal{B}\setminus\{B\})} q^{B'}_{\phi(j),f}$$

$$c^{B,S}_{i,f} := \sum_{j=1}^{n} \delta(\phi^B(j) = i)\left(\sum_{B'\in(\mathcal{B}\setminus\{B\})} q^{B'}_{\phi(j),f}\right)^2.$$

It is clear that eq. (21) can be computed efficiently in $\mathcal{O}(n^B)$ provided that the caches $\mathbf{c}^B$ and $\mathbf{c}^{B,S}$ are present. Both caches depend only on variables of other blocks $B \neq B'$, so they can be precomputed before model parameters of a block B should be computed. The third cache that has to be present is $\mathbf{q}^B$. This cache depends on the model parameters of the block (see eq. (20)) but can be updated in constant time by $q^B_{i,f} \leftarrow q^B_{i,f} - \Delta^B_{l,f}\, x^B_{i,l}$.

Efficient computation of the second quantity in eq. (18), $f(i) = h_{v_{l,f}}(\mathbf{x}_i)\, e_i$, is based on the decomposition:

$$\sum_{i=1}^{n} h_{v_{l,f}}(\mathbf{x}^B_i) e_i = \sum_{i=1}^{n^B}\left(h^B_{v_{l,f}}(\mathbf{x}^B_i) e^B_i + x^B_{i,l} e^{B,q}_{i,f}\right) \tag{22}$$

where e^B_i is the same as for LR and

$$e^{B,q}_{i,f} = \sum_{j=1}^{n} \delta(\phi^B(j) = i) e_j \sum_{B'\in(\mathcal{B}\setminus\{B\})} q^{B'}_{\phi(j),f}.$$

Both caches have to be kept in sync while model parameters are learned. The corresponding constant time update rules after the value of a model parameter changes (let $\Delta^B_{l,f}$ be the difference) are:

$$e^B_i \leftarrow e^B_i + \Delta^B_{l,f}\left(h^B_{v_{l,f}}(\mathbf{x}^B_i)\, \#^B_i + x^B_{i,l}\, c^B_{i,f}\right)$$

$$e^{B,q}_{i,f} \leftarrow e^{B,q}_{i,f} + \Delta^B_{l,f}\left(h^B_{v_{l,f}}(\mathbf{x}^B_i)\, c^B_{i,f} + x^B_{i,l}\, c^{B,S}_{i,f}\right)$$

This shows that also the second quantity of FM learning, can be computed in $\mathcal{O}(n^B)$ instead of $\mathcal{O}(n)$ by precomputing and updating caches.

Like for LR, the last problem is to keep the global residuals $\mathbf{e}$ in sync. Again, these residuals cannot be updated efficiently while learning each single model parameter in a block, because there are n global residuals. However as for LR, the global residuals are only necessary to initialize block specific caches (i.e. $\mathbf{e}^B$ and $\mathbf{e}^{B,q}$). This means global residuals $\mathbf{e}$ can be unsynchronized once in $\mathcal{O}(n)$ before entering a block: $e_i \leftarrow e_i + ((q^B_{\phi(i),f})^2 - q^{B,S}_{\phi(i),f})$. And can be resynchronized after leaving the block: $e_i \leftarrow e_i - ((q^B_{\phi(i),f})^2 - q^{B,S}_{\phi(i),f})$.

To summarize, each model parameter can be learned by CD or Gibbs (eqs. 16,17) in $\mathcal{O}(n^B)$ – and by summing only over non-zero predictor variables $x^B_{i,l} \neq 0$, the update is even in $\mathcal{O}(\sum_{i=1}^{n^B} \delta(x^B_{i,l} \neq 0))$. This is done by using the efficient formulation eq. (21) and eq. (22) instead of the standard formulation eq. (18). The efficient formulation requires several caches which can be precomputed once in $\mathcal{O}(n)$ time and (if necessary) updated per model parameter in $\mathcal{O}(\sum_{i=1}^{n^B} \delta(x^B_{i,l} \neq 0))$. Thus, updating all p^B model parameters in a block is in $\mathcal{O}(N_Z(X^B))$ plus for precomputing/ postcomputing $\mathcal{O}(n)$. For all $|B|$ blocks, the computational effort is $\mathcal{O}(N_Z(\mathcal{B}))$ and for all k factorization dimensions the total learning runtime is $\mathcal{O}(k\, N_Z(\mathcal{B}))$ for a full iteration.

Algorithm 2 summarizes the efficient algorithm that makes use of the repeating patterns in the feature vectors of the design matrix. Like for prediction, the storage complexity for the caches is $\mathcal{O}(n + \sum_{B \in \mathcal{B}} n^B)$. Note that also standard CD and standard Gibbs sampling, have a caching complexity of $\mathcal{O}(n)$. In terms of computational complexity, using the proposed BS algorithms, the runtime complexity drops from $\mathcal{O}(k\, N_Z(X))$ to $\mathcal{O}(k\, N_Z(\mathcal{B}))$ with the cost of an increasing storage complexity for caches of $\mathcal{O}(n + \sum_{B \in \mathcal{B}} n^B)$ instead of $\mathcal{O}(n)$. However, with relational data typically $N_Z(X) \gg N_Z(\mathcal{B})$ and $\sum_{B \in \mathcal{B}} n^B \leq n$. Moreover, one should remember, that the BS algorithm also uses the more compact BS representation $\mathcal{B}$ of X which means that the storage for the *data* is only $N_Z(\mathcal{B})$ instead of $N_Z(X)$.

4.3 Discussion

The key point in both LR and FM scaling is that the models allow to isolate each predictor variable x_i and model parameter $v_{l,f}$, conditioned on the remaining ones. In LR, the reason is the linearity of the model and for FMs the multilinearity[8]. Scaling non-decomposable models might be much harder and is a direction of future work.

5. EVALUATION

The evaluation first substantiates the need for new algorithms when feature engineering is applied to relational datasets. This will show the limitation of current feature-based machine learning methods on large scale relational datasets.

Secondly, the prediction quality of FMs using relational predictor variables is investigated. This should underpin that (1) relational predictor variables improve the quality and (2) the scaled FM model can achieve a high prediction quality in very competitive tasks.

5.1 Datasets

All experiments are conducted on two very competitive and large scale datasets:

Netflix Prize: The task of the Netflix prize is to predict how many stars a user assigns to a movie (regression task). The dataset contains about $n = 100,000,000$ ratings from $480,000$ users on $17,770$ movies. The basic predictor variables for a case are *user ID*, *movie ID* and *day*. As derived predictor variables: the number of ratings of a user on a certain day ('*freq.*'), the day as a numeric value for a linear time effect ('*lin. day*') and as relational predictor variable, the set ('*impl*') of all movies ever rated by a user.

[8]Note that multilinearity does not mean that FMs are linear models. In fact, FMs allow nonlinear effects – comparable to polynomial regression or SVMs with a polynomial kernel.

Algorithm 2 FM-BS($\mathbf{y}$, $\mathcal{B}$)

```
1: $\Theta \sim \mathcal{N}(0, \sigma^2)$
2: for $i \in \{1, \ldots, \#\text{Iter}\}$ do
3:   $\hat{\mathbf{y}} \leftarrow$ predict all cases                 $\triangleright O(k\, N_z(\mathcal{B}))$
4:   $\mathbf{e} \leftarrow \mathbf{y} - \hat{\mathbf{y}}$                          $\triangleright O(n)$
5:   update hyperparameter $\alpha$                 $\triangleright O(n)$, see [13]
6:   for $B \in \mathcal{B}$ do
7:     update hyperparameter $\mu^B_w, \lambda^B_w$     $\triangleright O(p^B)$, see [13]
8:     cache: init $\mathbf{e}^B$ and unsync $\mathbf{e}$                  $\triangleright O(n)$
9:     for $l \in \{1, \ldots, p^B\}$ do
10:      update $w^B_l$ from eq. (5) using eqs. (8,10)
11:      cache: update $\mathbf{e}^B$, $\mathbf{q}^B$
12:    end for
13:    cache: sync $\mathbf{e}$                                 $\triangleright O(n)$
14:  end for
15:  for $f \in \{1, \ldots, k\}$ do
16:    cache: init $\mathbf{q}_{\cdot,f}, \mathbf{q}^{B_1}_{\cdot,f}, \mathbf{q}^{B_2}_{\cdot,f}, \ldots$         $\triangleright O(N_z(\mathcal{B}))$
17:    for $B \in \mathcal{B}$ do
18:      update hyperparameter $\mu^B_f, \lambda^B_f$ $\triangleright O(p^B)$, see [13]
19:      cache: init $\mathbf{e}^B, \mathbf{e}^{B,q}, \mathbf{c}^B, \mathbf{c}^{B,S}$, unsync $e$   $\triangleright O(n)$
20:      for $l \in \{1, \ldots, p^B\}$ do
21:        update $v^B_{l,f}$ from eq. (17) using eqs. (21,22)
22:        cache: update $\mathbf{e}^B, \mathbf{e}^{B,q}, \mathbf{q}^B, \mathbf{q}^{B,S}$
23:      end for
24:      cache: sync $\mathbf{e}$                               $\triangleright O(n)$
25:    end for
26:  end for
27: end for
```

KDDCup 2012: The task of the KDDCup 2012 (Track 1) is to predict which tweets a user will follow (classification/ ranking task). There are $n = 73,209,277$ statements about accepted/ rejected tweets from $231,569$ users on $4,992$ different tweeters. There is plenty of relational information available: a table describing the user in terms of gender, age, keywords and tags; a table describing the items in detail; a social network ($50,655,143$ entries) about follower/ followee relationships between users.

5.2 Experimental Reproducibility

The official test set and error/ quality measures provided by the challenge organizers are used. For easy reproducibility of all experiments, FM-BS is implemented in libFM[9]. All FM and FM-BS experiments are run on a single core of an Intel i5-2500 CPU with 32GB RAM.

5.3 Complexity of Feature Engineering with Relational Data

As noted before, standard ML algorithms have typically at best a linear complexity in the number of non-zeros of the design matrix $N_Z(X)$. The proposed new learning algorithms for LR and FM have a linear complexity in the (semantically) equivalent BS representation, i.e. in $N_Z(\mathcal{B})$. First, empirical sizes of $N_Z(X)$ and $N_Z(\mathcal{B})$ for different selections of predictor variables are investigated. Figure 5 and Table 1 show three selections of predictor variables for the KDDCup 2012 and two choices for the Netflix competition. The first selection of predictor variables for KDDCup 2012 includes attributes of the user: age, gender, number

[9]`http://www.libfm.org/`

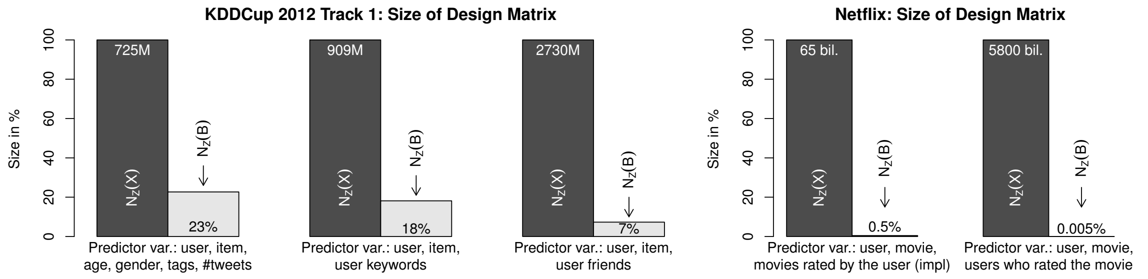


Figure 5: Feature engineering in relational data can result in very large design matrices with repeating patterns ($N_Z(X) \gg N_Z(\mathcal{B})$). See Table 1 for details.

Table 1: Characteristics of feature engineering/ design matrices for three datasets. Computational runtime of standard learning algorithms is linear in $N_Z(X)$. The proposed BS algorithms make use of repeating patterns in X and have a runtime complexity linear in $N_Z(\mathcal{B})$. For k (number of latent dimensions/ model complexity) the same values as in the qualitative study (section 5.4) are used.

Dataset	Predictor Variables	Repr.	Complexity N_z	libFM Runtime in sec
KDDCup 2012	user, item, age, gender, tags, #tweets	X	724,866,951	967 ($k = 22$)
		$\mathcal{B}$	164,358,302	129 ($k = 22$)
	user, item, keywords	X	908,501,618	1417 ($k = 22$)
		$\mathcal{B}$	164,895,145	139 ($k = 22$)
	user, item, friends	X	2,730,323,386	3778 ($k = 22$)
		$\mathcal{B}$	199,399,584	207 ($k = 22$)
Netflix	user, movie, all movies rated by the user ('impl.')	X	65,721,617,393	infeasible ($k = 128$)
		$\mathcal{B}$	304,756,611	1954 ($k = 128$)
	user, movie, all users who rated the movie	X	5,800,969,095,025	infeasible ($k = 128$)
		$\mathcal{B}$	304,756,611	1399 ($k = 128$)
Movielens 1M	user, movie, all movies rated by the user ('impl.')	X	313,708,262	48 ($k = 10$)
		$\mathcal{B}$	2,610,407	0.5 ($k = 10$)
	user, movie, all users who rated the movie	X	654,716,983	248 ($k = 10$)
		$\mathcal{B}$	2,610,407	0.5 ($k = 10$)

of tweets and the user's tags. The amount of repeating information is here less than adding all the friends of a user (third selection). This is because there are more friends per user than the number of attributes of a user. The predictor variables chosen for the Netflix dataset are highly relational and result in large repeating blocks in X because in the first Netflix selection, each case is described with all the movies this user has seen. The second selection of predictor variables for Netflix, where for each case all the users that have rated this movie are part of the predictor variables, has even much more repeating patterns because there are much less users than movies in the Netflix dataset and thus for the first selection less movies are added per case than users are added per case for the second selection.

The number of non-zeros entries in the design matrix range from 725 million to 5800 billion entries – whereas the size of the BS representation is at max. 305 million and 4 to $19,035$ times smaller. This shows that any learning and prediction algorithm that is based on standard feature engineering is slow or even infeasible if it does not handle repeating patterns. It is important to note again that the issue for standard algorithms is not (only) the storage complexity of the design matrices X, rather the problem is the (at least) linear runtime complexity in X because the algorithms require to see each case/ row $\mathbf{x}_i$ of X for learning/prediction. The proposed BS algorithms solve this issue by making use of repeating patterns.

Table 1 shows also empirical runtimes for an FM, either using the standard learning algorithm or the proposed BS algorithm[10]. The empirical runtime reduction from FM to FM-BS shows the same behavior as the improvement from $N_Z(X)$ to $N_Z(\mathcal{B})$.

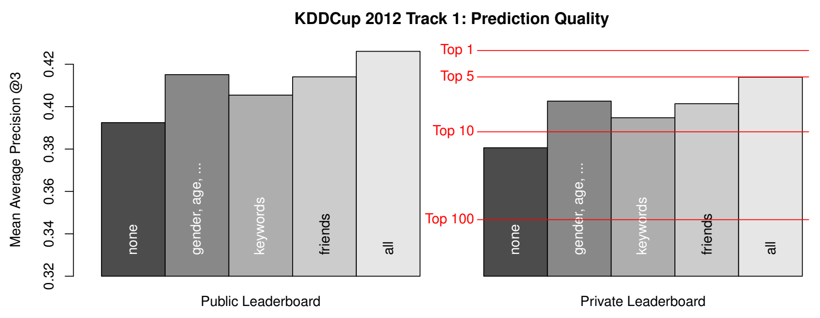
This experiment substantiates that standard learning algorithms for feature engineering based models cannot scale to large datasets if the predictor variables have strong relational patterns.

5.4 Quality of Feature Engineering with Relational Data

Next, it will be substantiated that scaling FMs to relational dataset makes also sense from a prediction quality perspective. Two questions will be addressed: (i) *Do the relational predictors improve prediction quality?* (ii) *How good is the prediction quality of FM?*

The first question is investigated by comparing different choices of variable selections. Figure 6 shows the prediction quality (*mean average precision* for KDDCup 2012 and *root mean squared error* for Netflix). All five KDDCup 2012 variable selections include the basic variables: user ID, item ID as well as sequential variables [14]. In Figure 6, the first measurement '*none*' shows only these basic variables, the second one adds user attributes: gender, age, #tweets, tags (as in Figure 5, left), the third the keywords (Figure 5, middle), the fourth the friends (Figure 5, right) and the last one adds all of these variables, i.e. user attributes, keywords

[10]The average runtime for a full iteration using LIBFM is reported.



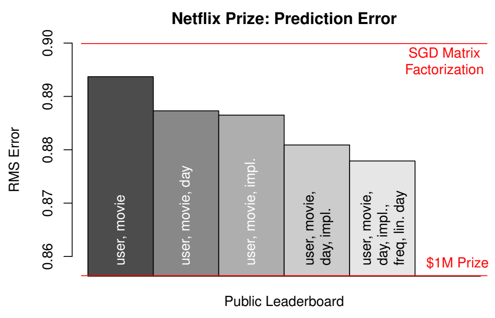


Figure 6: Prediction quality / error of FMs for selections of predictor variables. Relational predictor variables improve the prediction quality / error.

and friends. For the Netflix dataset, the leftmost method is an FM using just the user ID and movie ID as predictor variable. By selecting more predictor variables, the error decreases. Here, the (relational) set variable, *impl.*, that contains all movie IDs a user has ever rated, drops the error largely. These experiments show that additional predictor variables that involve relational patterns improve the prediction quality considerably. In conjunction with the complexity experiments (section 5.3) this confirms the need for BS algorithms for feature-based ML models in general.

The second question is addressed by comparing the FM results to the best approaches in the highly competitive Netflix prize and KDDCup 2012. In each of these competitions, hundreds of teams have developed and evaluated ML methods. In the following, the FM approach is compared to the most successful ones. In Figure 6, the horizontal (red) lines show the quality of the top1 (winning entry), top5 (5th entry), top10 and top100 for the KDDCup 2012. As it can be seen, generic feature engineering with FM as model and FM-BS for scalable learning is highly competitive. Please note that most of the entries (also the horizontal (red) lines in the chart) in KDDCup 2012 are not just single methods (as the FM in the chart), but ensembles of several models. For the Netflix prize, Table 2 shows a detailed comparison of the FMs of Figure 6 to the best non-ensemble approaches developed for the Netflix problem. As it can be seen, feature engineering with FMs is as good as the best specialized models proposed for the Netflix problem. This shows that FMs are a very competitive model that is worth scaling.

It should also be highlighted that the best competing approaches for Netflix (Table 2) and KDDCup 2012 are not generic feature/ design matrix based models but highly specialized models tailored for the specific tasks. In contrast to this, FMs are generic and can be applied to any feature vectors/ design matrices. This is also an interesting research result, as feature engineering does not play a major role in the application area of recommender system (yet). Finally, it should be noted that most of the competing approaches are not only tailored for the tasks (high effort for developing the model and deriving learning algorithms) but also require extensive hyperparameter search for adjusting learning rates ((S)GD based approaches) and regularization values ((S)GD and ALS/CD approaches). The qualitative results for FM-BS in Figure 6 are obtained with much less effort as no important hyperparameters have to be specified due to MCMC inference.

5.5 Discussion

The evaluation has shown that feature engineering from large scale datasets can result in design matrices with very large repeating blocks due to relational structure in the predictor variables. Algorithms with linear runtime in the BS representation of the design matrix can scale to such data. It has been shown on two very competitive datasets, that generic feature engineering based ML methods can create very good results on par with the best specialized and tailored approaches that are known so far. As some of the important predictor variables lead to large repeating patterns, a prerequisite for any feature engineering based algorithm is to handle BS data efficiently.

Scaling feature engineering methods to relational data is highly important for practice as it allows to use the well-established predictive modeling workflow based on feature engineering.

6. RELATED WORK

6.1 Algorithms for Generic ML Models

In [22], linear models are scaled to large design matrices that do not fit in memory by breaking the problem into subproblems and using disk access. This work makes no use of repeating patterns and still has a runtime complexity in $N_Z(X)$. Thus, in terms of our experiments, the algorithm in [22] could be applied but would still be 4 to $19,000$ times slower (plus the additional time for disk-reading) than the proposed BS algorithm.

Another well-studied approach for scaling algorithms is parallelization (e.g. [1]). With perfectly parallelized standard algorithms this would mean that the problems of our evaluation need at least 4 to $19,000$ machines to come up to the same solution as the BS algorithms on a single machine. In contrast to this, the contribution of our work is to make use of structural properties of the design matrix to speed up learning.

6.2 Algorithms for Specialized Models

Restricting the models to a specific setting of predictor variables simplifies developing fast learning algorithms. For rating prediction on the Netflix Prize dataset, Koren and Bell [6] propose fast learning algorithms for the SVD++ and factorized KNN model. From a predictor variable point of view, these models are restricted to two categorical variables and one set variable. The algorithm in [6] makes use

Table 2: **Prediction error on the Netflix prize dataset. A star * indicates that this is the best value reported in the corresponding paper for this method. The methods are grouped by the information that they take into account. The RMSE results are measured on the Quiz dataset (leaderboard scores).**

Method (Name)	Reference	Learning Method	k	Quiz RMSE
Models using user ID and item ID				
Probabilistic Matrix Factorization	[17, 16]	Batch GD	40	*0.9170
Probabilistic Matrix Factorization	[17, 16]	Batch GD	150	0.9211
Matrix Factorization	[7]	Variational Bayes	30	*0.9141
Matchbox	[18]	Variational Bayes	50	*0.9100
ALS-MF	[10]	ALS	100	0.9079
ALS-MF	[10]	ALS	1000	*0.9018
SVD/ MF	[3]	SGD	100	0.9025
SVD/ MF	[3]	SGD	200	*0.9009
Bayesian Probablistic Matrix Factorization (BPMF)	[16]	MCMC	150	0.8965
Bayesian Probablistic Matrix Factorization (BPMF)	[16]	MCMC	300	*0.8954
FM-BS, pred. var: user ID, movie ID	-	MCMC	128	0.8937
Models using implicit feedback				
Probabilistic Matrix Factorization with Constraints	[17]	Batch GD	30	*0.9016
SVD++	[3]	SGD	100	0.8924
SVD++	[3]	SGD	200	*0.8911
BSRM/F	[24]	MCMC	100	0.8926
BSRM/F	[24]	MCMC	400	*0.8874
FM-BS, pred. var: user ID, movie ID, impl.	-	MCMC	128	0.8865
Models using time information				
Bayesian Probabilistic Tensor Factorization (BPTF)	[21]	MCMC	30	*0.9044
FM-BS, pred. var: user ID, movie ID, day	-	MCMC	128	0.8873
Models using time and implicit feedback				
timeSVD++	[5]	SGD	100	0.8805
timeSVD++	[5]	SGD	200	*0.8799
FM-BS, pred. var: user ID, movie ID, day, impl.	-	MCMC	128	0.8809
FM-BS, pred. var: user ID, movie ID, day, impl.	-	MCMC	256	0.8794
Assorted models				
BRISMF/UM NB corrected	[19]	SGD	1000	*0.8904
BMFSI plus side information	[11]	MCMC	100	*0.8875
timeSVD++ plus frequencies	[4]	SGD	200	0.8777
timeSVD++ plus frequencies	[4]	SGD	2000	*0.8762
FM-BS, pred. var: user ID, movie ID, day, impl., freq., lin. day	-	MCMC	128	0.8779
FM-BS, pred. var: user ID, movie ID, day, impl., freq., lin. day	-	MCMC	256	0.8771

of this setting by sorting the data by the set variable and performing batch GD on the set variable and SGD on the remaining two variables. The FM-BS algorithm proposed in our work (1) is more general as it can be applied to any feature vector/ design matrix, (2) results in the analytically same solution as the standard FM algorithm and (3) supports full Bayesian inference incl. hyperparameter selection (with MCMC).

6.3 Dimensionality Reduction

Another generic approach to deal with large design matrices is dimensionality reduction. Many reduction techniques have been proposed from linear transformation (e.g. PCA), non-linear manifold techniques to hashing (e.g. [20]). Even though learning in the transformed space can be fast and approximately exact, the original data X has to be processed at least one time which means at some step the computational complexity is still in $\mathcal{O}(N_Z(X))$. Studying dimensionality reduction techniques that make use of the repeating patterns to scale to $\mathcal{O}(N_Z(\mathcal{B}))$ is a promising direction of future work.

6.4 Statistical Relational Learning

Neville et al. [8] and Perlich and Provost [9] use feature-based classifiers for relational predictive modeling. However, their approaches restrict the modeling flexibility by aggregating predictor variables from relations. Especially, a predictor variable such as '*friends*' or '*movies watched by a user*' is not possible but would have to be processed into a single (numeric or categorical) variable in the feature engineering step. In our work, we focus on using the traditional statistical model (without any adaption) and show that it works well on relational data if one can solve the runtime issue (as proposed in our work).

7. CONCLUSION AND FUTURE WORK

In this work, it was shown that design matrices from relational data can get very large which makes learning and prediction slow or even infeasible for standard machine learning algorithms. As the concept of design matrices is common to most ML methods, this is an inherent problem for all of them.

In this work, based on the observation that feature vectors/ design matrices share block structure, efficient prediction and learning algorithms have been developed for linear regression and factorization machines. The proposed learning algorithms are analytically exact improvements of CD and MCMC. In a study on two very competitive ML tasks, it was shown that the proposed algorithms have a large speedup compared to standard feature-based learning algorithms and a prediction quality as good as the best specialized models develop for the respective tasks.

There are several directions for future work. First of all, other machine learning methods could also be scaled to relational data by exploiting repeating patterns in the feature vectors/ design matrix. Secondly, the proposed BS algorithms could further benefit from parallelization, e.g. by parallelizing posterior draws. A third interesting direction is to compare the predictive accuracy and computational effort to specialized relational learners that do not rely on feature engineering.

8. REFERENCES

[1] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, Cambridge, MA, 2007.

[2] J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2 2010.

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

[4] Y. Koren. The bellkor solution to the netflix grand prize. 2009.

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

[6] Y. Koren and R. M. Bell. Advances in collaborative filtering. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 145–186. Springer, 2011.

[7] Y. J. Lim and Y. W. Teh. Variational Bayesian approach to movie rating prediction. In *Proceedings of KDD Cup and Workshop*, 2007.

[8] J. Neville, D. Jensen, and B. Gallagher. Simple estimators for relational bayesian classifiers. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 609–612, Washington, DC, USA, 2003. IEEE Computer Society.

[9] C. Perlich and F. Provost. Distribution-based aggregation for relational learning with identifier attributes. *Mach. Learn.*, 62(1-2):65–105, Feb. 2006.

[10] I. Pilászy, D. Zibriczky, and D. Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*, pages 71–78, New York, NY, USA, 2010. ACM.

[11] I. Porteous, A. Asuncion, and M. Welling. Bayesian matrix factorization with side information and dirichlet process mixtures. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI 2010, pages 563–568, 2010.

[12] S. Rendle. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 995–1000, Washington, DC, USA, 2010. IEEE Computer Society.

[13] S. Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012.

[14] S. Rendle. Social network and click-through prediction with factorization machines. In *KDD-Cup Workshop*, 2012.

[15] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90, New York, NY, USA, 2010. ACM.

[16] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 880–887, New York, NY, USA, 2008. ACM.

[17] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1257–1264, Cambridge, MA, 2008. MIT Press.

[18] D. H. Stern, R. Herbrich, and T. Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 111–120, New York, NY, USA, 2009. ACM.

[19] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, June 2009.

[20] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

[21] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the SIAM International Conference on Data Mining*, pages 211–222. SIAM, 2010.

[22] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 833–842, New York, NY, USA, 2010. ACM.

[23] H.-F. Yu, H.-Y. Lo, H.-P. Hsieh, J.-K. Lou, T. G. McKenzie, J.-W. Chou, P.-H. Chung, C.-H. Ho, C.-F. Chang, Y.-H. Wei, J.-Y. Weng, E.-S. Yan, C.-W. Chang, T.-T. Kuo, Y.-C. Lo, P. T. Chang, C. Po, C.-Y. Wang, Y.-H. Huang, C.-W. Hung, Y.-X. Ruan, Y.-S. Lin, S. de Lin, H.-T. Lin, and C.-J. Lin. Feature engineering and classifier ensemble for kdd cup 2010. In *Proceedings of KDD Cup and Workshop*, 2010.

[24] S. Zhu, K. Yu, and Y. Gong. Stochastic relational models for large-scale dyadic data using MCMC. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1993–2000, 2009.