Convex Factorization Machines

Mathieu Blondel

Joint work with A. Fujino and N. Ueda

NTT Communication Science Laboratories
Kyoto, Japan

2015/9/14

Problem setting

- This talk is concerned with the traditional **supervised learning** setting

  From training set

  $$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d \quad \text{and} \quad y_1, \ldots, y_n \in \mathbb{R}$$

  we want to learn a prediction function

  $$\hat{y} \colon \mathbb{R}^d \to \mathbb{R}$$

- We want $\hat{y}(\boldsymbol{x})$ to take into account **second-order interaction features**

Second-order interaction features

- **Second-order interaction features** have a significant effect on the response in many regression problems
- For instance, interactions of multiple genes can play an important role in the expression of certain phenotypes
- Classical approach: **polynomial regression**

Polynomial regression

- Polynomial regression uses one parameter per interaction feature

$$\hat{y}(\boldsymbol{x}) := w_0 + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \sum_{j=1}^{d}\sum_{j'=j}^{d} z_{jj'} x_j x_{j'}$$

- Drawbacks:
  - Quadratic number of parameters to estimate
  - $z_{jj'}$ is zero if interaction never occurred in the training set (likely if $\boldsymbol{x}$ is high-dimensional and sparse)

Factorization machines

- Proposed by S. Rendle (ICDM 2010)
- Efficient way to model **feature interactions** in **high-dimensional** spaces
- Contains several factorization models as special case
- Popular in the recsys community
- Open-source implementation: www.libfm.org

Factorization machines

- Use a **factorized** matrix for **interaction feature** weights

$$\hat{y}(\boldsymbol{x}) := w_0 + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \sum_{j=1}^{d}\sum_{j'=j+1}^{d} (\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}})_{jj'} x_j x_{j'}$$

$$w_0 \in \mathbb{R}, \quad \boldsymbol{w} \in \mathbb{R}^d, \quad \boldsymbol{V} \in \mathbb{R}^{d \times k} \quad k \ll d$$

- Advantages over polynomial regression
  - Number of parameters to estimate is now $O(dk)$ instead of $O(d^2)$
  - Prediction cost is now $O(n_z(\boldsymbol{x})k)$ instead of $O(n_z(\boldsymbol{x})^2)$
  - $(\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}})_{jj'}$ can be non-zero even if $x_j x_{j'}$ never occurred in training set

Factorization machines

- Objective function

$$\min_{w_0 \in \mathbb{R}, \boldsymbol{w} \in \mathbb{R}^d, \boldsymbol{V} \in \mathbb{R}^{d \times k}} \frac{1}{2} \sum_{i=1}^{n} \Big(y_i - \hat{y}(\boldsymbol{x}_i) \Big)^2 + \frac{\alpha}{2} \| \boldsymbol{w} \|^2 + \frac{\beta}{2} \| \boldsymbol{V} \|_F^2$$

- Typically solved by SGD or coordinate descent

Factorization machines

- Important detail: prediction function of FMs ignores diagonal elements $x_1^2, \ldots, x_d^2$ since $j' > j$

$$\hat{y}(\boldsymbol{x}) := w_0 + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + \sum_{j=1}^{d} \sum_{j'=j+1}^{d} (\boldsymbol{V} \boldsymbol{V}^{\mathrm{T}})_{jj'} x_j x_{j'}$$

- Can we use diagonal elements instead?

$$\hat{y}(\boldsymbol{x}) := w_0 + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + \sum_{j=1}^{d} \sum_{j'=j}^{d} (\boldsymbol{V} \boldsymbol{V}^{\mathrm{T}})_{jj'} x_j x_{j'}$$

Factorization machines

- New (non-)convexity results w.r.t. $\boldsymbol{V} \in \mathbb{R}^{d \times k}$

	use diag	ignore diag
Full matrix	non-convex	non-convex
Element-wise	non-convex	convex

$\Rightarrow$ Ignore diag case is easier to solve than use diag case

$\Rightarrow$ Element-wise coordinate descent is a good method in the ignore diag case

Convex Factorization Machines

- We propose a **convex formulation** of FMs
- Benefits of convexity
  - **Global solution** can be found $\Rightarrow$ insensitive to initialization
  - One less hyper-parameter to decide (no rank hyper-parameter)
  - Convex, whether we use diagonal elements or not

Prediction function

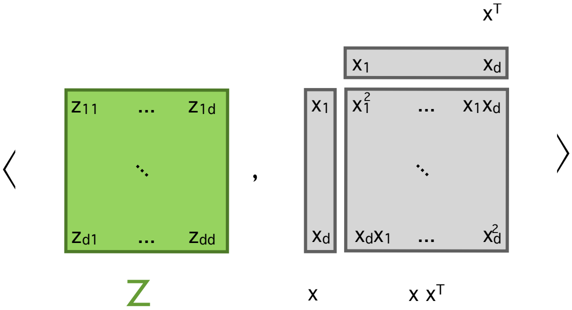
- We rewrite the prediction function as

$$\begin{aligned}\hat{y}(\boldsymbol{x}) &= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \sum_{j=1}^{d}\sum_{j'=1}^{d} z_{jj'} x_j x_{j'} \\ &= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{x} \\ &= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \langle \boldsymbol{Z}, \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} \rangle\end{aligned}$$

- $\boldsymbol{Z}$ is a $d \times d$ **symmetric** matrix
- $z_{jj'}$ is the weight of $x_j x_{j'}$ for predicting y
- Bias term omitted for simplicity

Quadratic forms

- $\boldsymbol{x}^{\mathrm{T}} \boldsymbol{Z} \boldsymbol{x} = \langle \boldsymbol{Z}, \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} \rangle$ is called a **quadratic form**

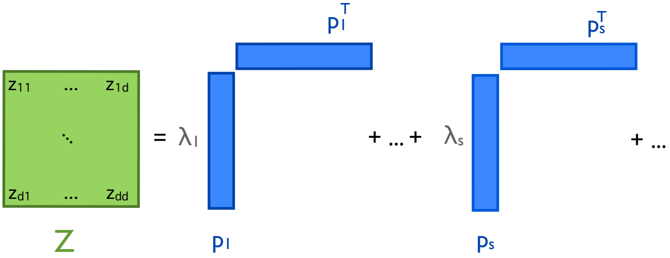


- Advantage: we can enforce $\boldsymbol{Z}$ to be **low-rank**

Eigendecomposition

- Any real symmetric matrix $\boldsymbol{Z}$ can be decomposed as a sum of rank-one matrices

$$\boldsymbol{Z} = \sum_{s=1}^{d} \lambda_s \boldsymbol{p}_s \boldsymbol{p}_s^{\mathrm{T}} = \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}$$

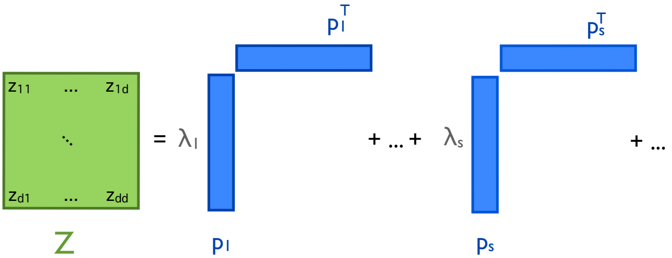


Eigendecomposition

- **Low-rank matrix** = sum of a small number of rank-one matrices

$$\boldsymbol{Z} = \sum_{s=1}^{k} \lambda_s \boldsymbol{p}_s \boldsymbol{p}_s^{\mathrm{T}} \text{ where } k = \mathrm{rank}(\boldsymbol{Z}) \ll d$$

(assuming $\lambda_1, \ldots, \lambda_d$ are sorted in decreasing order)



Nuclear norm (a.k.a trace norm)

- To promote low-rank solutions, we use the **nuclear norm**
- Nuclear norm of a symmetric matrix $\boldsymbol{Z} = \sum_{s=1}^{d} \lambda_s \boldsymbol{p}_s \boldsymbol{p}_s^{\mathrm{T}}$

$$\|\boldsymbol{Z}\|_* = \mathrm{Tr}(\sqrt{\boldsymbol{Z}\boldsymbol{Z}}) = \|\boldsymbol{\lambda}\|_1$$

$\Rightarrow$ nuclear norm $= \ell_1$ norm of eigenvalues

- sparse $\boldsymbol{\lambda} \Rightarrow$ low-rank $\boldsymbol{Z}$

Sparse vs. low-rank

	ℓ_1 norm	nuclear norm
Definition	$\|\boldsymbol{w}\|_1$	$\|\boldsymbol{Z}\|_* = \|\boldsymbol{\lambda}\|_1$
Surrogate of	$\|\boldsymbol{w}\|_0$	$\mathrm{rank}(\boldsymbol{Z})$
Effect	sparse	low-rank
Decomposition	$\boldsymbol{w} = \sum_{j=1}^{d} w_j \boldsymbol{e}_j$	$\boldsymbol{Z} = \sum_{s=1}^{d} \lambda_s \boldsymbol{p}_s \boldsymbol{p}_s^{\mathrm{T}}$
Atoms	$\boldsymbol{e}_j$ (standard basis)	$\boldsymbol{p}_s \boldsymbol{p}_s^{\mathrm{T}}$ (rank-one matrix)

Objective function

- Proposed objective:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d, \boldsymbol{Z}\in\mathbb{R}^{d\times d}} \sum_{i=1}^{n} \ell\Big(y_i, \hat{y}(\boldsymbol{x}_i)\Big) + \frac{\alpha}{2}\|\boldsymbol{w}\|^2 + \beta\|\boldsymbol{Z}\|_*$$

  where ℓ is a twice-differentiable convex loss function

- **Jointly** convex in $\boldsymbol{w}$ and $\boldsymbol{Z}$
- The larger β, the smaller rank($\boldsymbol{Z}$)
- Optimal $\boldsymbol{Z}$ is always symmetric

Algorithm outline

- Two-block coordinate descent

  1. Minimize w.r.t. $\boldsymbol{w}$
  2. Minimize w.r.t. $\boldsymbol{Z}$
  3. Repeat until convergence
  4. Return $\boldsymbol{w}^*$ and $\boldsymbol{Z}^* = \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}$

- Converges to a global solution

Minimizing w.r.t. $\boldsymbol{Z}$

- Standard nuclear norm penalized objective

$$\min_{\boldsymbol{Z} \in \mathbb{R}^{d \times d}} L(\boldsymbol{Z}) + \beta \|\boldsymbol{Z}\|_*$$

  where L is twice-differentiable convex

- Proximal methods and ADMM do not scale well
- State-of-the-art: **greedy coordinate descent**
- Can exploit symmetry to derive more efficient solver

Algorithm outline

- $\boldsymbol{P} \leftarrow [\,]$ $\quad \boldsymbol{\lambda} \leftarrow [\,]$ $\quad$ (equivalent to $\boldsymbol{Z} \leftarrow \mathbf{0}$)
- Repeat until convergence
  1. Find $\boldsymbol{p}$ which most violates KKT conditions
  2. Find optimal λ (closed form solution for squared loss)
  3. $\boldsymbol{P} \leftarrow [\boldsymbol{P}\ \boldsymbol{p}]$ $\quad \boldsymbol{\lambda} \leftarrow [\boldsymbol{\lambda}\ \lambda]$ $\quad$ (equivalent to $\boldsymbol{Z} \leftarrow \boldsymbol{Z} + \lambda \boldsymbol{p}\boldsymbol{p}^{\mathrm{T}}$)
  4. Periodically: refit objective restricted to current subspace
- Return $\boldsymbol{Z}^* = \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}$

Refitting

- Given the current iterate $\boldsymbol{Z} = \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}$
- Diagonal refitting

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^k} L(\boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}) + \beta\|\boldsymbol{\lambda}\|_1$$

- Fully-corrective refitting

$$\min_{\boldsymbol{A}\in\mathbb{R}^{k\times k}} L(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{\mathrm{T}}) + \beta\|\boldsymbol{A}\|_*$$

since $\|\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^{\mathrm{T}}\|_* = \|\boldsymbol{A}\|_*$ when $\boldsymbol{P}$ is an orthogonal matrix

Quadratic kernel interpretation

- We can rewrite the prediction function as

$$\begin{aligned}
\hat{y}(\boldsymbol{x}) &= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \langle \boldsymbol{Z}, \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} \rangle \\
&= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \langle \sum_{s=1}^{k} \lambda_s \boldsymbol{p}_s \boldsymbol{p}_s^{\mathrm{T}}, \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} \rangle \\
&= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \sum_{s=1}^{k} \lambda_s (\boldsymbol{p}_s^{\mathrm{T}}\boldsymbol{x})^2
\end{aligned}$$

- $(\boldsymbol{p}_s^{\mathrm{T}}\boldsymbol{x})^2$ is the homogeneous quadratic kernel between $\boldsymbol{p}_s$ and $\boldsymbol{x}$

Quadratic kernel interpretation

- Compare

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \sum_{s=1}^{k} \lambda_s (\boldsymbol{p}_s^{\mathrm{T}}\boldsymbol{x})^2$$

  with a kernelized regression model

$$\hat{y}(\boldsymbol{x}) = \sum_{i=1}^{n} a_i \kappa(\boldsymbol{x}_i, \boldsymbol{x})$$

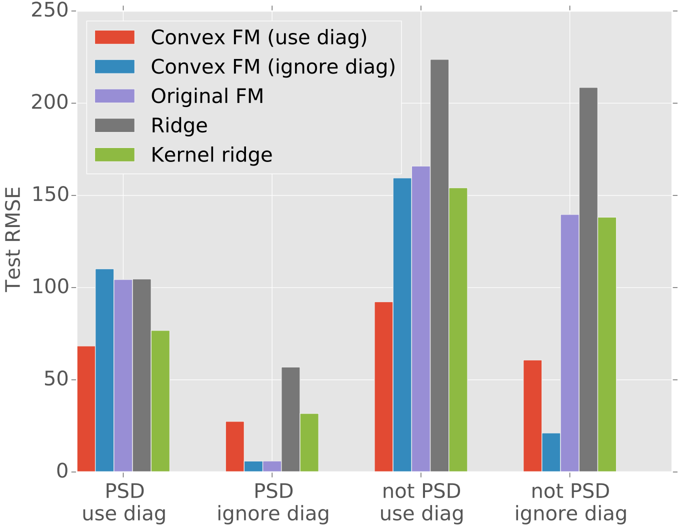
- By learning a low-rank $\boldsymbol{Z}$, we are indirectly learning basis vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k$ and their weights $\lambda_1, \ldots, \lambda_k$

Experiments

Synthetic data

- Generate $\boldsymbol{X}$ using $x_{ij} \sim \mathcal{N}(0, 1)$
- Generate $\boldsymbol{w}$ using $w_j \sim \mathcal{N}(0, 1)$
- Generate $\boldsymbol{P}$ using $p_{js} \sim \mathcal{N}(0, 1)$
- Generate $\boldsymbol{\lambda}$
  - $\lambda_s \sim \mathcal{N}(0, 1)$ if not PSD
  - $\lambda_s \sim \mathcal{U}(0, 1)$ if PSD
- Generate $\boldsymbol{y}$
  - $y_i = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + \langle \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}, \boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}} \rangle + \epsilon$ if use diag
  - $y_i = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + \langle \boldsymbol{P}\,\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{P}^{\mathrm{T}}, \boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}} - \mathrm{diag}(\boldsymbol{x}_i)^2 \rangle + \epsilon$ if ignore diag

Synthetic experiment



Application to collaborative filtering

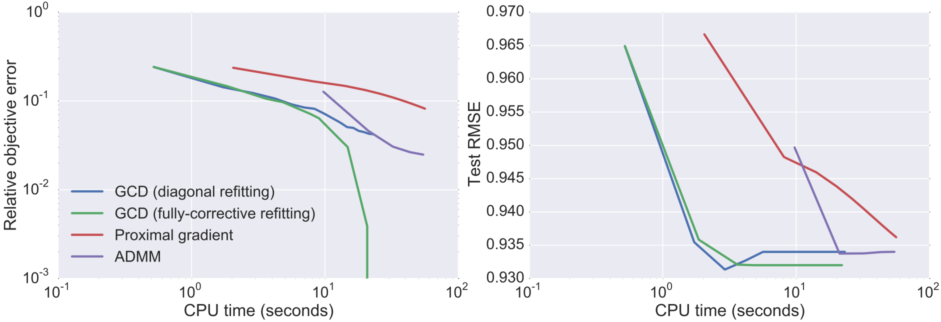
- If user $u \in \{1, \ldots, U\}$ gave 3 stars to movie $i \in \{1, \ldots, I\}$, we can set

$$\boldsymbol{x} := [\underbrace{0, \ldots, 0, \overbrace{1}^{u}, 0, \ldots, 0}_{U}, \underbrace{0, \ldots, 0, \overbrace{1}^{U+i}, 0, \ldots, 0}_{I}]^{\mathrm{T}}$$

$$y := 3$$

- Number of training pairs $(\boldsymbol{x}_i, y_i)$ is number of ratings
- Number of features is $d = U + I$
- Then factorization machines are **equivalent** to matrix factorization

Solver comparison



Movielens 100k

$\alpha = 10^{-9}$, $\beta = 10$

Comparison with original FMs

	Convex FMs (use diag)	Convex FMs (ignore diag)	Original FMs
ML 100k	0.93	0.93	0.93
ML 1m	0.87	0.85	0.86
ML 10m	0.84	0.82	0.81
Last.fm	2.21	2.05	2.13

Test RMSE with hyper-parameters tuned by 3-fold CV

Conclusion

- Factorization machines are useful for leveraging **feature interactions** even with **high-dimensional sparse** data
- We proposed a **convex formulation** of factorization machines
- Although they are especially popular in the recsys community, we emphasize that factorization machines are **general-purpose**
- In particular, more applications using **biological** data would be welcome