Higher-order Factorization Machines

Mathieu Blondel

NTT Communication Science Laboratories
Kyoto, Japan

Joint work with M. Ishihata, A. Fujino and N. Ueda

2016/9/28
Regression analysis

- Variables

\[ y \in \mathbb{R}: \text{target variable} \]
\[ x \in \mathbb{R}^d: \text{explanatory variables (features)} \]

- Training data

\[ y = [y_1, \ldots, y_n]^T \in \mathbb{R}^n \]
\[ X = [x_1, \ldots, x_n] \in \mathbb{R}^{d \times n} \]

- Goal
  - Learn model parameters
  - Compute prediction \( y \) for a new \( x \)
Linear regression

- Model

\[ \hat{y}_{LR}(x; w) := \langle w, x \rangle = \sum_{j=1}^{d} w_j x_j \]

- Parameters

\[ w \in \mathbb{R}^d: \text{feature weights} \]

- Pros and cons

- \( O(d) \) predictions

- Learning \( w \) can be cast as a convex optimization problem

- Does not use feature interactions
Polynomial regression

- Model

\[ \hat{y}_{PR}(x; w) := \langle w, x \rangle + x^T W x = \langle w, x \rangle + \sum_{j,j'=1}^{d} w_{j,j'} x_j x_{j'} \]

- Parameters

\[ w \in \mathbb{R}^{d} : \text{feature weights} \]
\[ W \in \mathbb{R}^{d \times d} : \text{weight matrix} \]

- Pros and cons

😊 Learning \( w \) and \( W \) can be cast as a convex optimization problem

🚫 \( O(d^2) \) time and memory cost
Kernel regression

- **Model**

\[ \hat{y}_{KR}(x; \alpha) := \sum_{i=1}^{n} \alpha_i K(x_i, x) \]

- **Parameters**

\[ \alpha \in \mathbb{R}^n: \text{instance weights} \]

- **Pros and cons**

  - 😊 Can use non-linear kernels (RBF, polynomial, etc...)
  - 😊 Learning \( \alpha \) can be cast as a convex optimization problem
  - 😞 \( O(dn) \) predictions (linear dependence on training set size)
Factorization Machines (FM) (Rendle, ICDM 2010)

- **Model**

  \[ \hat{y}_{FM}(x; w, P) := \langle w, x \rangle + \sum_{j' > j} \langle \bar{p}_j, \bar{p}_{j'} \rangle x_j x_{j'} \]

- **Parameters**

  \( w \in \mathbb{R}^d \): feature weights

  \( P \in \mathbb{R}^{d \times k} \): weight matrix

- **Pros and cons**

  - Takes into account feature combinations

  - \( O(2dk) \) predictions (linear-time) instead of \( O(d^2) \)

  - Parameter estimation involves a non-convex optimization problem
Application 1: recsys without features

- Formulate it as a matrix completion problem

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Movie 3</th>
<th>Movie 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>★★</td>
<td>?</td>
<td>★★★</td>
<td>?</td>
</tr>
<tr>
<td>Bob</td>
<td>★</td>
<td>?</td>
<td>★★</td>
<td>?</td>
</tr>
<tr>
<td>Charlie</td>
<td>★★</td>
<td>?</td>
<td>?</td>
<td>★★</td>
</tr>
</tbody>
</table>

- Matrix factorization: find $U, V$ that approximately reconstruct the rating matrix

$$R \approx UV^T$$
Conversion to a regression problem

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Movie 3</th>
<th>Movie 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>★★</td>
<td>?</td>
<td>★★★</td>
<td>?</td>
</tr>
<tr>
<td>Bob</td>
<td>★</td>
<td>?</td>
<td>★★</td>
<td>?</td>
</tr>
<tr>
<td>Charlie</td>
<td>★★</td>
<td>?</td>
<td>?</td>
<td>★★</td>
</tr>
</tbody>
</table>

\[ \downarrow \text{one-hot encoding} \]

\[
\begin{bmatrix}
★★ \\
★★★★ \\
★ \\
★★ \\
★★ \\
★★
\end{bmatrix}
\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

Using this representation, FMs are equivalent to MF!
Generalization ability of FMs

- The weight of $x_j x_{j'}$ is $\langle \tilde{p}_j, \tilde{p}_{j'} \rangle$ compared to $w_{j,j'}$ for PR
- The same parameters $\tilde{p}_j$ are shared for the weight of $x_j x_{j'} \ orall j > j'$
- This increases the amount of data used to estimate $\tilde{p}_j$ at the cost of introducing some bias (low-rank assumption)
- This allows to generalize to feature interactions that were not observed in the training set
# Application 2: recsys with features

<table>
<thead>
<tr>
<th>Rating</th>
<th>User</th>
<th>Movie</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>**</td>
<td>M</td>
<td>Adventure S. Spielberg</td>
</tr>
<tr>
<td>***</td>
<td>F</td>
<td>Anime H. Miyazaki</td>
</tr>
<tr>
<td>*</td>
<td>M</td>
<td>Drama A. Kurosawa</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

- **Interactions between **categorical variables**
  - Gender $\times$ genre: $\{M, F\} \times \{\text{Adventure, Anime, Drama, ...}\}$
  - Age $\times$ director: $\{0-10, 10-20, ...\} \times \{\text{S. Spielberg, H. Miyazaki, A. Kurosawa, ...}\}$

- *In practice, the number of interactions can be huge!*
# Conversion to regression

<table>
<thead>
<tr>
<th>Rating</th>
<th>User</th>
<th>Movie</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ** Star rating for a male user aged 20-30 watching an Adventure movie directed by S. Spielberg.
- **⭐ Star rating for a female user aged 0-10 watching an Anime movie directed by H. Miyazaki.
- ⭐ Star rating for a male user aged 20-30 watching a Drama movie directed by A. Kurosawa.

\[ \downarrow \text{one-hot encoding} \]

\[
\begin{bmatrix}
\star\star \\
\star\star\star \\
\star \\
\vdots
\end{bmatrix}
\quad
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
\]

very sparse binary data!
FM revisited (Blondel+, ICML 2016)

- ANOVA kernel of degree $m = 2$ (Stitson+, 1997; Vapnik, 1998)

$$A^2(p, x) := \sum_{j' > j} p_j x_j \, p_{j'} x_{j'}$$

- Then

$$\hat{y}_{FM}(x; w, P) = \langle w, x \rangle + \sum_{j' > j} \langle \bar{p}_j, \bar{p}_{j'} \rangle x_j x_{j'}$$

$$= \langle w, x \rangle + \sum_{s=1}^{k} A^2(p_s, x) \uparrow s^{th} \text{ column of } P$$
ANOVA kernel (arbitrary-order case)

- ANOVA kernel of degree $2 \leq m \leq d$
  
  $$A^m(p, x) := \sum_{j_m > \cdots > j_1} (p_{j_1} x_{j_1}) \cdots (p_{j_m} x_{j_m})$$
  
  $\uparrow$ All possible $m$-combinations of $\{1, \ldots, d\}$

- Intuitively, the kernel uses all $m$-combinations of features without replacement: $x_{j_1} \cdots x_{j_m}$ for $j_1 \neq \cdots \neq j_m$

- Computing $A^m(p, x)$ naively takes $O(d^m)$
Higher-order FM s (HOFMs)

- Model

\[ \hat{y}_{HOFM}(x; w, \{ P^t \}_{t=2}^m) := \langle w, x \rangle + \sum_{t=2}^{m} \sum_{s=1}^{k} A^t(p^t_s, x) \]

- Parameters

\[
\begin{align*}
    w \in \mathbb{R}^d &: \text{feature weights} \\
    P^2, \ldots, P^m \in \mathbb{R}^{d \times k} &: \text{weight matrices}
\end{align*}
\]

- Pros and cons

👍 Takes into account higher-order feature combinations

👍 \( O(dkm^2) \) prediction cost using our proposed algorithms

👎 More complex than 2nd-order FM s
Learning HOFMs (1/2)

- We use alternating minimization w.r.t. $w, P^2, \ldots, P^m$
- Learning $w$ alone reduces to linear regression
- Learning $P^m$ can be cast as minimizing

$$F(P) := \frac{1}{n} \sum_{i=1}^{n} \ell \left( y_i, \sum_{s=1}^{k} A^m(p_s, x_i) + o_i \right) + \frac{\beta}{2} \| P \|^2$$

where $o_i$ is the contribution of degrees other than $m$
Learning HOFMs (2/2)

• Stochastic gradient update

\[ p_s \leftarrow p_s - \eta \ell'(y_i, \hat{y}_i) \nabla A^m(p_s, x_i) - \eta \beta p_s \]

where \( \eta \) is a learning rate hyper-parameter and

\[ \hat{y}_i := \sum_{s=1}^{k} A^m(p_s, x_i) + o_i \]

• We propose \( O(dm) \) (linear time) DP algorithms for
  ○ Evaluating ANOVA kernel \( A^m(p, x) \in \mathbb{R} \)
  ○ Computing gradient \( \nabla A^m(p, x) \in \mathbb{R}^d \)
Evaluating the ANOVA kernel (1/3)

• Recursion (Blondel+, ICML 2016)

\[
A^m(p, x) = A^m(p_{\neg j}, x_{\neg j}) + p_j x_j \ A^{m-1}(p_{\neg j}, x_{\neg j}) \ \forall j
\]

where \( p_{\neg j}, x_{\neg j} \in \mathbb{R}^{d-1} \) are vectors with the \( j^{th} \) element removed

• We can use this recursion to remove features until computing the kernel becomes trivial
Evaluating the ANOVA kernel (2/3)

Shortcut:

\[ a_{j,t} := A^t(p_{1:j}, x_{1:j}) \]
\[ p_{1:j} := [p_1, \ldots, p_j]^T \]
\[ x_{1:j} := [x_1, \ldots, x_j]^T \]
Evaluating the ANOVA kernel (3/3)

Ways to avoid redundant computations:
- Top-down approach with memory table
- **Bottom-up dynamic programming (DP)**

<table>
<thead>
<tr>
<th></th>
<th>$j = 0$</th>
<th>$j = 1$</th>
<th>$j = 2$</th>
<th>...</th>
<th>$j = d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t = 0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$t = 1$</td>
<td>0</td>
<td>$a_{1,1}$</td>
<td>$a_{2,1}$</td>
<td>...</td>
<td>$a_{d,1}$</td>
</tr>
<tr>
<td>$t = 2$</td>
<td>0</td>
<td>0</td>
<td>$a_{2,2}$</td>
<td>...</td>
<td>$a_{d,2}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$t = m$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>$a_{d,m}$</td>
</tr>
</tbody>
</table>
Algorithm 1 Evaluating $A^m(p, x)$ in $O(dm)$

**Input:** $p \in \mathbb{R}^d$, $x \in \mathbb{R}^d$

$a_{j,t} \leftarrow 0 \ \forall t \in \{1, \ldots, m\}, \ j \in \{0, 1, \ldots, d\}$

$a_{j,0} \leftarrow 1 \ \forall j \in \{0, 1, \ldots, d\}$

for $t := 1, \ldots, m$ do

  for $j := t, \ldots, d$ do

    $a_{j,t} \leftarrow a_{j-1,t} + p_j x_j a_{j-1,t-1}$

  end for

end for

**Output:** $A^m(p, x) = a_{d,m}$
Backpropagation (chain rule)

Ex: compute derivatives of composite function $f(g(h(p)))$

- **Forward pass**
  
  $$a = h(p)$$
  $$b = g(a)$$
  $$c = f(b)$$

- **Backward pass**
  
  \[
  \frac{\partial c}{\partial p_j} = \frac{\partial c}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial p_j} = f'(b) g'(a) h'_j(p)
  \]

  Only the last part depends on $j$

  Can compute all derivatives in one pass!
Gradient computation (1/2)

- We want to compute \( \nabla \mathcal{A}^m(p, x) = [\tilde{p}_1, \ldots, \tilde{p}_d]^T \)

- Using the chain rule, we have

\[
\tilde{p}_j := \frac{\partial a_{d,m}}{\partial p_j} = \sum_{t=1}^{m} \frac{\partial a_{d,m}}{\partial a_{j,t}} \frac{\partial a_{j,t}}{\partial p_j} = \sum_{t=1}^{m} \tilde{a}_{j,t} a_{j-1,t-1} x_j
\]

since \( p_j \) influences \( a_{j,t} \) \( \forall t \in [m] \)

- \( \tilde{a}_{j,t} \) can be computed recursively in reverse order

\[
\tilde{a}_{j,t} = \tilde{a}_{j+1,t} + p_{j+1} x_{j+1} \tilde{a}_{j+1,t+1}
\]
## Gradient computation (2/2)

<table>
<thead>
<tr>
<th></th>
<th>goal</th>
<th>( j = 1 )</th>
<th>( j = 2 )</th>
<th>( \ldots )</th>
<th>( j = d - 1 )</th>
<th>( j = d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t = 1 )</td>
<td>( \tilde{a}_{1,1} )</td>
<td>( \tilde{a}_{2,1} )</td>
<td>( \ldots )</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( t = 2 )</td>
<td>0</td>
<td>( \tilde{a}_{2,2} )</td>
<td>( \ldots )</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \tilde{a}_{d-1,t-1} )</td>
<td>0</td>
</tr>
<tr>
<td>( t = m )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>( p_{d \times d} )</td>
</tr>
<tr>
<td>( t = m + 1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>


**Algorithm 2** Computing $\nabla A^m(p, x)$ in $O(dm)$

**Input:** $p \in \mathbb{R}^d$, $x \in \mathbb{R}^d$, \{a_{j,t}\}_{j,t=0}^{d,m}

$\tilde{a}_{j,t} \leftarrow 0 \ \forall t \in [m + 1], j \in [d]$

$\tilde{a}_{d,m} \leftarrow 1$

**for** $t := m, \ldots, 1$ **do**

**for** $j := d - 1, \ldots, t$ **do**

$\tilde{a}_{j,t} \leftarrow \tilde{a}_{j+1,t} + \tilde{a}_{j+1,t+1}p_{j+1}x_{j+1}$

**end for**

**end for**

$\tilde{p}_j := \sum_{t=1}^{m} \tilde{a}_{j,t} a_{j-1,t-1}x_j \ \forall j \in [d]$

**Output:** $\nabla A^m(p, x) = [\tilde{p}_1, \ldots, \tilde{p}_d]^T$
HOFMs can be expressed using the ANOVA kernel $A^m$

We proposed $O(dm)$ time algorithms for computing $A^m(p, x)$ and $\nabla A^m(p, x)$

The cost per epoch of stochastic gradient algorithms for learning $P^m$ is therefore $O(dnk)$

The prediction cost is $O(dkm^2)$
Other contributions

• Coordinate-descent algorithm for learning $P^m$ based on a different recursion
  ○ Cost per epoch is $O(dnk m^2)$
  ○ However, no learning rate to tune!

• HOFMs with shared parameters: $P^2 = \cdots = P^m$
  ○ Total prediction cost is $O(dkm)$ instead of $O(dkm^2)$
  ○ Corresponds to using new kernels derived from the ANOVA kernel
Experiments
Application to link prediction

Goal: predict missing links between nodes in a graph

Graph:
- Co-author network
- Enzyme network

Bipartite graph:
- User-movie
- Gene-disease
Application to link prediction

- We assume two sets of nodes $A$ (e.g., users) and $B$ (e.g., movies) of size $n_A$ and $n_B$
- Nodes in $A$ are represented by feature vectors $a_i \in \mathbb{R}^{d_A}$
- Nodes in $B$ are represented by feature vectors $b_j \in \mathbb{R}^{d_B}$
- We are given a matrix $Y \in \{-1, +1\}^{n_A \times n_B}$ such that $y_{i,j} = +1$ if there is a link between $a_i$ and $b_j$
- Number of positive samples is $n_+$
Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n_+$</th>
<th>Columns of $A$</th>
<th>$n_A$</th>
<th>$d_A$</th>
<th>Columns of $B$</th>
<th>$n_B$</th>
<th>$d_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIPS</td>
<td>4,140</td>
<td>Authors</td>
<td>2,037</td>
<td>13,649</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enzyme</td>
<td>2,994</td>
<td>Enzymes</td>
<td>668</td>
<td>325</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD</td>
<td>3,954</td>
<td>Diseases</td>
<td>3,209</td>
<td>3,209</td>
<td>Genes</td>
<td>12,331</td>
<td>25,275</td>
</tr>
<tr>
<td>ML 100K</td>
<td>21,201</td>
<td>Users</td>
<td>943</td>
<td>49</td>
<td>Movies</td>
<td>1,682</td>
<td>29</td>
</tr>
</tbody>
</table>

Features:

- **NIPS**: word occurrence in author publications
- **Enzyme**: phylogenetic information, gene expression information and gene location information
- **GD**: MimMiner similarity scores (diseases) and HumanNet similarity scores (genes)
- **ML 100K**: age, gender, occupation, living area (users); release year, genre (movies)
Models compared

Goal: predict if there is a link between \(a_i\) and \(b_j\)

- **HOFM**: \(\hat{y}_{i,j} = \hat{y}_{HOFM}(a_i \oplus b_j; w, \{P^t\}_{t=2}^m)\)
- **HOFM-shared**: same but with \(P^2 = \cdots = P^m\)
- **Polynomial network (PN)**: replace ANOVA kernel by polynomial kernel
- **Bilinear regression (BLR)**: \(\hat{y}_{i,j} = a_i U V^T b_j\)

vector concatenation
Experimental protocol

- We sample $n_- = n_+$ negatives samples (missing edges are treated as negative samples)
- We use 50% for training and 50% for testing
- We use ROC-AUC (area under ROC curve) for evaluation
- $\beta$ tuned by CV, $k$ fixed to 30
- $P^2, \ldots, P^m$ initialized randomly
- $\ell$ is set to the squared loss
Solver comparison

- Coordinate descent
- AdaGrad
- L-BFGS

AdaGrad and L-BFGS use the proposed DP algorithm to compute $\nabla A^m(p, x)$
NIPS dataset

(a) Convergence when $m = 2$

(b) Convergence when $m = 3$

(c) Convergence when $m = 4$

(d) Scalability w.r.t. $m$