Factorization Models for Recommender Systems and Other Applications

Part II

Lars Schmidt-Thieme, Steffen Rendle

Tutorial at KDD Conference, 12th August 2012, Beijing
Outline

Tensor Factorization
- Problem Setting
- Models
- Learning
- Examples for Applications
- Summary

Time-aware Factorization Models

Factorization Machines
Problem Setting

▶ Predictor variables: \( m \) variables of categorical domain \( I_1, \ldots, I_m \).

▶ Target \( y \): Real-valued (regression), binary (classification), scores (ranking).

▶ Supervised task: set of observations \( S = \{(i_1, \ldots, i_m, y), \ldots\} \)
Example: Social Tagging

Tagging can be expressed as a function over three categorical domains:

\[ y: U \times I \times T \rightarrow \{0, 1\} \]

\[ \text{http://last.fm} \]
Example: Social Tagging

Tagging can be expressed as a function over three categorical domains:

\[ y: U \times I \times T \rightarrow \{0, 1\} \]

\(^1http://last.fm\)
Example: Social Tagging

Tagging can be expressed as a function over three categorical domains: $y: U \times I \times T \rightarrow \{0, 1\}$

1http://last.fm
Example: Social Tagging\textsuperscript{1}

Tagging can be expressed as a function over three categorical domains:
\[
y : U \times I \times T \rightarrow \{0, 1\}
\]

\textsuperscript{1}http://last.fm
Example: Querying Incomplete RDF-Graphs

▶ **Task:** Answer queries about subject-predicate-pairs. E.g. What is McCartney member of?
Example: Querying Incomplete RDF-Graphs

An RDF-Graph can be expressed as a function over three categorical domains: $y : S \times P \times O \rightarrow \{0, 1\}$
Notation: Tensors and Functions

Models in this setting are functions:

\[ \hat{y} : I_1 \times \ldots \times I_m \rightarrow \mathcal{Y} \]
Notation: Tensors and Functions

Models in this setting are functions:

\[ \hat{y} : I_1 \times \ldots \times I_m \rightarrow Y \]

All possible targets and predictions can be written equivalently as a \textit{m-order tensor} / multiway array:

\[ Y \in \mathcal{Y}^{|I_1| \times \ldots \times |I_m|}, \quad \hat{Y} \in \mathcal{Y}^{|I_1| \times \ldots \times |I_m|} \]

where

\[ y(i_1, \ldots, i_m) = y_{i_1,...,i_m}, \quad \hat{y}(i_1, \ldots, i_m) = \hat{y}_{i_1,...,i_m} \]
Notation: Tensor-Matrix product

- Let $T \in \mathbb{R}^{k_1 \times \ldots \times k_m}$ be a $m$-order tensor and $V \in \mathbb{R}^{n \times k_l}$ be a matrix.
- The \textit{mode-$l$ tensor-matrix product} $\times_l$ is defined as:

$$ ( T \times_l M )_{i_1, \ldots, i_{l-1}, j, i_{l+1}, \ldots, i_m} := \sum_{i_l=1}^{k_l} t_{i_1, \ldots, i_m} m_{j, i_l} $$

![Diagram of tensor-matrix product](image-url)
Notation: Tensor-Matrix product

- Let \( T \in \mathbb{R}^{k_1 \times \cdots \times k_m} \) be a \( m \)-order tensor and \( V \in \mathbb{R}^{n \times k_l} \) be a matrix.
- The \textit{mode-}l \textit{tensor-matrix product} \( \times_l \) is defined as:

\[
(T \times_l M)_{i_1, \ldots, i_{l-1}, j, i_{l+1}, \ldots, i_m} := \sum_{i_l=1}^{k_l} t_{i_1, \ldots, i_m} m_{j, i_l}
\]

- The result is a tensor \( T^* \) of dimension \( \mathbb{R}^{k_1 \times k_{l-1} \times n \times k_{l+1} \times \cdots \times k_m} \)
Notation: Tensor-Matrix product

- Let $T \in \mathbb{R}^{k_1 \times \ldots \times k_m}$ be a $m$-order tensor and $V \in \mathbb{R}^{n \times k_l}$ be a matrix.
- The *mode-*l tensor-matrix product $\times_l$ is defined as:

\[
\left( T \times_l M \right)_{i_1, \ldots, i_{l-1}, j, i_{l+1}, \ldots, i_m} := \sum_{i_l=1}^{k_l} t_{i_1, \ldots, i_m} m_j, i_l
\]

- The result is a tensor $T^*$ of dimension $\mathbb{R}^{k_1 \times k_{l-1} \times n \times k_{l+1} \times \ldots \times k_m}$
- The size of the $l$-th mode changes from $k_l$ to $n$. 
Notation: Tensor-Matrix product

- Let $T \in \mathbb{R}^{k_1 \times \ldots \times k_m}$ be a $m$-order tensor and $V \in \mathbb{R}^{n \times k_l}$ be a matrix.
- The *mode-$l$ tensor-matrix product* $\times_l$ is defined as:

$$
(T \times_l M)_{i_1,\ldots,i_{l-1},j,i_{l+1},\ldots,i_m} := \sum_{i_l=1}^{k_l} t_{i_1,\ldots,i_m} m_j,i_l
$$

- The result is a tensor $T^*$ of dimension $\mathbb{R}^{k_1 \times k_{l-1} \times n \times k_{l+1} \times \ldots \times k_m}$
- The size of the $l$-th mode changes from $k_l$ to $n$. 
Notation: Tensor-Matrix product

- Let $T \in \mathbb{R}^{k_1 \times \cdots \times k_m}$ be a $m$-order tensor and $V \in \mathbb{R}^{n \times k_l}$ be a matrix.
- The *mode-$l$ tensor-matrix product* $\times_l$ is defined as:

$$ (T \times_l M)_{i_1, \ldots, i_{l-1}, j, i_{l+1}, \ldots, i_m} := \sum_{i_l=1}^{k_l} t_{i_1, \ldots, i_m} m_j, i_l $$

- The result is a tensor $T^*$ of dimension $\mathbb{R}^{k_1 \times k_{l-1} \times n \times k_{l+1} \times \cdots \times k_m}$
- The size of the $l$-th mode changes from $k_l$ to $n$. 
Notation: Tensor-Matrix product

- Let $T \in \mathbb{R}^{k_1 \times \cdots \times k_m}$ be a $m$-order tensor and $V \in \mathbb{R}^{n \times k_l}$ be a matrix.
- The *mode-$l$ tensor-matrix product* $\times_l$ is defined as:

  \[
  (T \times_l M)_{i_1, \ldots, i_{l-1}, j, i_{l+1}, \ldots, i_m} := \sum_{i_l=1}^{k_l} t_{i_1, \ldots, i_m} m_j, i_l
  \]

- The result is a tensor $T^*$ of dimension $\mathbb{R}^{k_1 \times k_{l-1} \times n \times k_{l+1} \times \cdots \times k_m}$
- The size of the $l$-th mode changes from $k_l$ to $n$. 
Outline

Tensor Factorization
  Problem Setting
  Models
  Learning
  Examples for Applications
  Summary

Time-aware Factorization Models

Factorization Machines
Parallel Factor Analysis (PARAFAC)

$m$-order PARAFAC in tensor product notation:

\[ \hat{Y} := C \times_1 V^{(1)} \times_2 \ldots \times_m V^{(m)} \]

with model parameters

\[ V^{(l)} \in \mathbb{R}^{l_1 \times \ldots \times l_m \times k}, \quad \forall l \in \{1, \ldots, m\} \]

and where \( C \) is the identity tensor:

\[ C \in \mathbb{R}^{k \times \ldots \times k}, \quad c_{j_1, \ldots, j_m} := \delta(j_1 = \ldots = j_m) \]

[Harshman 1970, Carroll 1970]
Parallel Factor Analysis (PARAFAC)

\[ \hat{Y} = \mathbf{V}^1 \times_1 \mathbf{V}^2 \times_2 \mathbf{V}^3 \]

*m*-order PARAFAC in element-wise notation:

\[ \hat{y}(i_1, \ldots, i_m) := \sum_{f=1}^{k} v_{i_1,f}^{(1)} \cdots v_{i_m,f}^{(m)} = \sum_{f=1}^{k} \prod_{l=1}^{m} v_{i_l,f}^{(l)} \]

with model parameters

\[ V^{(l)} \in \mathbb{R}^{l_i \times k}, \quad \forall l \in \{1, \ldots, m\} \]

[Harshman 1970, Carroll 1970]
Parallel Factor Analysis (PARAFAC)

Notes

- For a $m = 2$-order tensor (i.e. a matrix), PARAFAC is the same as matrix factorization.

[e.g. Kolda et al. 2009, Cichocki et al. 2009]
Parallel Factor Analysis (PARAFAC)

Notes

▶ For a $m = 2$-order tensor (i.e. a matrix), PARAFAC is the same as matrix factorization.

▶ Sometimes a modified PARAFAC with diagonal core $c_f,...,f =: \lambda_f$ and factors of unit length is used:

$$
\hat{y}(i_1, \ldots, i_m) := \sum_{f=1}^{k} \lambda_f \ v_{i_1, f}^{(1)} \cdots v_{i_m, f}^{(m)} = \sum_{f=1}^{k} \lambda_f \prod_{l=1}^{m} v_{i_l, f}^{(l)}
$$

[e.g. Kolda et al. 2009, Cichocki et al. 2009]
Parallel Factor Analysis (PARAFAC)

Notes

▶ For a $m = 2$-order tensor (i.e. a matrix), PARAFAC is the same as matrix factorization.

▶ Sometimes a modified PARAFAC with diagonal core $c_f,...,f =: \lambda_f$ and factors of unit length is used:

$$\hat{y}(i_1,...,i_m) := \sum_{f=1}^{k} \lambda_f \ v_{i_1,f}^{(1)} \cdots v_{i_m,f}^{(m)} = \sum_{f=1}^{k} \lambda_f \prod_{l=1}^{m} v_{i_l,f}^{(l)}$$

▶ Other constraints, e.g. non-negativity or symmetry can be imposed.

[e.g. Kolda et al. 2009, Cichocki et al. 2009]
Parallel Factor Analysis (PARAFAC)

Notes

- For a $m = 2$-order tensor (i.e. a matrix), PARAFAC is the same as matrix factorization.
- Sometimes a modified PARAFAC with diagonal core $c_{f,...,f} =: \lambda_f$ and factors of unit length is used:

$$\hat{y}(i_1, \ldots, i_m) := \sum_{f=1}^{k} \lambda_f v_{i_1,f}^{(1)} \cdots v_{i_m,f}^{(m)} = \sum_{f=1}^{k} \lambda_f \prod_{l=1}^{m} v_{i_l,f}^{(l)}$$

- Other constraints, e.g. non-negativity or symmetry can be imposed.
- PARAFAC is also called Canonical Decomposition (CANDECOMP).

[e.g. Kolda et al. 2009, Cichocki et al. 2009]
Tucker Decomposition (TD)

\[
\hat{Y} = \times_{1}^{C} \times_{2}^{V^{(1)}} \times_{3}^{V^{(2)}} \times_{4}^{V^{(3)}}
\]

\(m\)-order Tucker Decomposition in tensor product notation:

\[
\hat{Y} := C \times_{1} V^{(1)} \times_{2} \ldots \times_{m} V^{(m)}
\]

where \(V\) and \(C\) are model parameters:

\[
C \in \mathbb{R}^{k_{1} \times \ldots \times k_{m}}, \quad V^{(l)} \in \mathbb{R}^{I_{l} \times k_{l}}, \quad \forall l \in \{1, \ldots, m\}
\]

[Tucker 1966]
Tucker Decomposition (TD)

\[
\hat{Y} = C \times_1 V^1 \times_2 V^2 \times_3 V^3
\]

\(m\)-order Tucker Decomposition in element-wise notation:

\[
\hat{y}(i_1, \ldots, i_m) := \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} c_{f_1,\ldots,f_m} v_{i_1,f_1}^{(1)} \cdots v_{i_m,f_m}^{(m)} = \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} c_{f_1,\ldots,f_m} \prod_{l=1}^{m} v_{i_l,f_l}^{(l)}
\]

with model parameters:

\[
C \in \mathbb{R}^{k_1 \times \cdots \times k_m}, \quad V^{(l)} \in \mathbb{R}^{|I_l| \times k_l}, \quad \forall l \in \{1, \ldots, m\}
\]

[Tucker 1966]
Tucker Decomposition (TD)

Notes

- For a $m = 2$-order tensor (i.e. a matrix), TD is different from matrix factorization:

\[ \hat{y}^{TD}(i_1, i_2) = \sum_{f_1=1}^{k_1} \sum_{f_2=1}^{k_2} c_{f_1, f_2} v_{i_1, f_1}^{(1)} v_{i_2, f_2}^{(2)} \neq \sum_{f=1}^{k} v_{i_1, f}^{(1)} v_{i_2, f}^{(2)} = \hat{y}^{MF}(i_1, i_2) \]

[e.g. Kolda et al. 2009, Cichocki et al. 2009]
Tucker Decomposition (TD)

Notes

▶ For a $m = 2$-order tensor (i.e. a matrix), TD is different from matrix factorization:

$$\hat{y}_{TD}(i_1, i_2) = \sum_{f_1=1}^{k_1} \sum_{f_2=1}^{k_2} c_{f_1,f_2} v_{i_1,f_1}^{(1)} v_{i_2,f_2}^{(2)} \neq \sum_{f=1}^{k} v_{i_1,f}^{(1)} v_{i_2,f}^{(2)} = \hat{y}_{MF}(i_1, i_2)$$

▶ Sometimes orthogonality constraints on $V$ are imposed.

[e.g. Kolda et al. 2009, Cichocki et al. 2009]
Tucker Decomposition (TD)

Notes

- For a $m = 2$-order tensor (i.e. a matrix), TD is different from matrix factorization:

$$\hat{y}_{TD}(i_1, i_2) = \sum_{f_1=1}^{k_1} \sum_{f_2=1}^{k_2} c_{f_1,f_2} v_{i_1,f_1}^{(1)} v_{i_2,f_2}^{(2)} \neq \sum_{f=1}^{k} v_{i_1,f}^{(1)} v_{i_2,f}^{(2)} = \hat{y}_{MF}(i_1, i_2)$$

- Sometimes orthogonality constraints on $V$ are imposed.
- Other constraints, e.g. non-negativity or symmetry can be imposed.

[e.g. Kolda et al. 2009, Cichocki et al. 2009]
PARAFAC vs. TD

- **PARAFAC**

\[
\hat{y}(i_1, \ldots, i_m) := \sum_{f=1}^{k} v_{i_1,f}^{(1)} \cdots v_{i_m,f}^{(m)} = \sum_{f=1}^{k} \prod_{l=1}^{m} v_{i_l,f}^{(l)}
\]

- **TD**

\[
\hat{y}(i_1, \ldots, i_m) := \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} c_{f_1,\ldots,f_m} v_{i_1,f_1}^{(1)} \cdots v_{i_m,f_m}^{(m)} = \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} \prod_{l=1}^{m} v_{i_l,f_l}^{(l)}
\]

TD is more general, as \(C\) is free.
PARAFAC vs. TD

▸ PARAFAC

\[ \hat{y}(i_1, \ldots, i_m) := \sum_{f=1}^{k} v_{i_1,f}^{(1)} \cdots v_{i_m,f}^{(m)} = \sum_{f=1}^{k} \prod_{l=1}^{m} v_{i_l,f}^{(l)} \]

▸ TD

\[ \hat{y}(i_1, \ldots, i_m) := \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} c_{f_1, \ldots, f_m} v_{i_1,f_1}^{(1)} \cdots v_{i_m,f_m}^{(m)} = \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} \prod_{l=1}^{m} v_{i_l,f_l}^{(l)} \]

▸ TD is more general, as \( C \) is free.
PARAFAC vs. TD

- PARAFAC

\[ \hat{y}(i_1, \ldots, i_m) := \sum_{f=1}^{k} v_{i_1,f}^{(1)} \cdots v_{i_m,f}^{(m)} = \sum_{f=1}^{k} \prod_{l=1}^{m} v_{i_l,f}^{(l)} \]

- TD

\[ \hat{y}(i_1, \ldots, i_m) := \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} c_{f_1,\ldots,f_m} v_{i_1,f_1}^{(1)} \cdots v_{i_m,f_m}^{(m)} = \sum_{f_1=1}^{k_1} \cdots \sum_{f_m=1}^{k_m} c_{f_1,\ldots,f_m} \prod_{l=1}^{m} v_{i_l,f_l}^{(l)} \]

- TD is more general, as \( C \) is free.

- Computational complexity:
  - PARAFAC: \( O(k m) \).
  - TD: \( O(k^m) \) if \( k_1 = \ldots = k_m =: k \).
Tensor Factorization as Machine Learning Models

- PARAFAC and TD model m-ary interactions directly.
- PARAFAC and TD have problems when the number of observations for some levels is small:
  - E.g. assume that there are no observations for a level \( l \), then for the estimated factors \( v_l = 0 \) (in case of L2 regularization) and thus all predictions involving this level will always be 0 as well (for PARAFAC and TD).
  - Similar problems can occur if the number of observations of a level is small.
Tensor Factorization as Machine Learning Models

- PARAFAC and TD model m-ary interactions directly.
- PARAFAC and TD have problems when the number of observations for some levels is small:
  - E.g. assume that there are no observations for a level $l$, then for the estimated factors $v_l = 0$ (in case of L2 regularization) and thus all predictions involving this level will always be 0 as well (for PARAFAC and TD).
  - Similar problems can occur if the number of observations of a level is small.
- Standard L2-regularization alone cannot solve this problem.
Tensor Factorization as Machine Learning Models

▶ PARAFAC and TD model m-ary interactions directly.
▶ PARAFAC and TD have problems when the number of observations for some levels is small:
  ▶ E.g. assume that there are no observations for a level \( l \), then for the estimated factors \( \mathbf{v}_l = 0 \) (in case of L2 regularization) and thus all predictions involving this level will always be 0 as well (for PARAFAC and TD).
  ▶ Similar problems can occur if the number of observations of a level is small.
▶ Standard L2-regularization alone cannot solve this problem.
▶ If a m-ary interaction cannot be estimated reliably, often a lower-level interaction (e.g. \((m - 1)\)-ary) can be estimated reliably.
TF with Lower-level Interactions

Model equation of m-ary tensor factorization with nested lower-level interactions

\[ \hat{y}^\text{LLTF}(i_1, \ldots, i_m) := c + \sum_{l=1}^{m} w^{(l)}_{i_l} + \sum_{l_1=1}^{m} \sum_{l_2 > l_1}^{m} \hat{y}^\text{TF}(i_{l_1}, i_{l_2}) + \ldots + \hat{y}^\text{TF}(i_1, \ldots, i_m) \]

[e.g. Rendle et al. 2010; Cai et al. 2011]
TF with Lower-level Interactions

Model equation of m-ary tensor factorization with nested lower-level interactions

\[ \hat{y}^{\text{LLTF}}(i_1, \ldots, i_m) := c + \sum_{l=1}^{m} w_{i_l}^{(l)} + \sum_{l_1=1}^{m} \sum_{l_2 > l_1}^{m} \hat{y}^{\text{TF}}(i_{l_1}, i_{l_2}) + \ldots + \hat{y}^{\text{TF}}(i_1, \ldots, i_m) \]

Model parameters

\[ c \in \mathbb{R}, \quad w^{(l)} \in \mathbb{R}^{|l_l|}, \ldots, \quad \mathbf{V}^{(l)} \in \mathbb{R}^{|l_l| \times k} \]

[e.g. Rendle et al. 2010; Cai et al. 2011]
TF with Lower-level Interactions

Model equation of m-ary tensor factorization with nested lower-level interactions

\[
\hat{y}^{\text{LLTF}}(i_1, \ldots, i_m) := c + \sum_{l=1}^{m} w^{(l)}_{i_l} + \sum_{l_1=1}^{m} \sum_{l_2 > l_1}^{m} \hat{y}^{\text{TF}}(i_{l_1}, i_{l_2}) + \ldots + \hat{y}^{\text{TF}}(i_1, \ldots, i_m)
\]

Model parameters

\[
c \in \mathbb{R}, \quad w^{(l)} \in \mathbb{R}^{|I_l|}, \quad \ldots, \quad V^{(l)} \in \mathbb{R}^{|I_l| \times k}
\]

- Estimating a lower level effect (e.g. a pairwise one) reliably is easier than estimating a higher level one.

- Often lower level effects can explain the data sufficiently and higher level ones can be dropped completely.

[e.g. Rendle et al. 2010; Cai et al. 2011]
Outline

Tensor Factorization
  Problem Setting
  Models
  Learning
  Examples for Applications
  Summary

Time-aware Factorization Models

Factorization Machines
Standard Fitting Algorithms

Standard algorithms assume:

- $Y$ is observed completely, i.e. for all combinations $(i_1, \ldots, i_m) \in I_1 \times \ldots \times I_m$, $y_{i_1,\ldots,i_m}$ is known.
  - Missing values are imputed.

- Optimization is done with respect to least squares:

$$\arg\min_{\Theta} \sum_{(i_1,\ldots,i_m) \in I_1 \times \ldots \times I_m} (y_{i_1,\ldots,i_m} - \hat{y}_{i_1,\ldots,i_m})^2$$

- No regularization/prior assumptions.
Standard Fitting Algorithms

Standard algorithms assume:

- $Y$ is observed completely, i.e. for all combinations $(i_1, \ldots, i_m) \in I_1 \times \ldots \times I_m$, $y_{i_1,\ldots,i_m}$ is known.
  - Missing values are imputed.
  - *In ML problems most elements are missing (often $> 99.9\%$).*
- Optimization is done with respect to least squares:
  \[
  \arg \min_{\Theta} \sum_{(i_1, \ldots, i_m) \in I_1 \times \ldots \times I_m} (y_{i_1,\ldots,i_m} - \hat{y}_{i_1,\ldots,i_m})^2
  \]

- *ML: Other losses are also of interest, e.g. classification, ranking, . . . .*
- No regularization/ prior assumptions.
  - *ML: Prior knowledge should be included.*
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- Model: Tucker decomposition
- Algorithm:
  - For each mode $l$
    - Unfold $Y$ to matrix form.
    - Compute SVD.
    - $V^{(l)}$ are the left singular vectors of the SVD.
  - Compute core tensor
    $$ C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T. $$

[Tucker 66, Lathauwer et al. 2000]
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- **Loss:** Least-squares loss without regularization; no missing value treatment.
- **Model:** Tucker decomposition
- **Algorithm:**
  - For each mode $l$
    - Unfold $Y$ to matrix form.
    - Compute SVD.
    - $V^{(l)}$ are the left singular vectors of the SVD.
  - Compute core tensor $C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T$.

[Tucker 66, Lathauwer et al. 2000]
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- **Loss:** Least-squares loss without regularization; no missing value treatment.
- **Model:** Tucker decomposition
- **Algorithm:**
  - For each mode \( l \)
    - Unfold \( Y \) to matrix form.
    - Compute SVD.
    - \( V^{(l)} \) are the left singular vectors of the SVD.
  - Compute core tensor
    \[ C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T. \]

[Tucker 66, Lathauwer et al. 2000]
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- **Loss:** Least-squares loss without regularization; no missing value treatment.
- **Model:** Tucker decomposition
- **Algorithm:**
  - For each mode $l$
    - Unfold $Y$ to matrix form.
    - Compute SVD.
    - $V^{(l)}$ are the left singular vectors of the SVD.
  - Compute core tensor $C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T$.

[Tucker 66, Lathauwer et al. 2000]
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

▶ Loss: Least-squares loss without regularization; no missing value treatment.
▶ Model: Tucker decomposition
▶ Algorithm:
  ▶ For each mode $l$
    ▶ Unfold $Y$ to matrix form.
    ▶ Compute SVD.
    ▶ $V^{(l)}$ are the left singular vectors of the SVD.
  ▶ Compute core tensor
    \[ C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T. \]

[Tucker 66, Lathauwer et al. 2000]
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- **Loss**: Least-squares loss without regularization; no missing value treatment.
- **Model**: Tucker decomposition
- **Algorithm**:
  - For each mode $l$
    - Unfold $Y$ to matrix form.
    - Compute SVD.
    - $V^{(l)}$ are the left singular vectors of the SVD.
  - Compute core tensor
    $$ C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T. $$
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- Model: Tucker decomposition
- Algorithm:
  - For each mode $l$
    - Unfold $Y$ to matrix form.
    - Compute SVD.
    - $V^{(l)}$ are the left singular vectors of the SVD.
  - Compute core tensor
    $$C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T.$$  

[Tucker 66, Lathauwer et al. 2000]
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- Model: Tucker decomposition
- Algorithm:
  - For each mode $l$
    - Unfold $Y$ to matrix form.
    - Compute SVD.
    - $V^{(l)}$ are the left singular vectors of the SVD.
  - Compute core tensor
    \[ C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T. \]

[Tucker 66, Lathauwer et al. 2000]
Example: Higher-Order SVD (HOSVD)

HOSVD is such an approximative fitting algorithm:

- **Loss**: Least-squares loss without regularization; no missing value treatment.
- **Model**: Tucker decomposition
- **Algorithm**:
  - For each mode $l$
    - Unfold $Y$ to matrix form.
    - Compute SVD.
    - $V^{(l)}$ are the left singular vectors of the SVD.
  - Compute core tensor
    $$C = Y \times_1 (V^{(1)})^T \times_2 (V^{(2)})^T \times_3 \ldots \times_m (V^{(m)})^T.$$  

Additional Alternating Least-Squares (ALS) steps can improve the fit.

[Tucker 66, Lathauwer et al. 2000]
Machine Learning with TF Models

- Optimize only w.r.t. observed elements of $Y$.
  - Comparable to MF: *Weighted Low-Rank Approximations* [Srebro et al. 2003]
- Choose loss/ likelihood according to the target variables/ task.
  - E.g. logit for classification, pairwise classification for ranking, etc.
- Add priors / regularization to model parameters.
  - E.g. L2/ Gaussian priors.
- Model lower-level interactions.
  - E.g. add factorized pairwise interactions [Rendle et al. 2010]

TF models are multilinear $\Rightarrow$ simple SGD or ALS algorithms can be used for optimization.
Outline

Tensor Factorization
  Problem Setting
  Models
  Learning
Examples for Applications
Summary

Time-aware Factorization Models

Factorization Machines
Personalized Tag Recommendation

**Task:** Recommend a user a (personalized) list of tags for a specific item.

[Hetke et al., 2006]
Personalized Tag Recommendation

- $U$ ... users
- $I$ ... items
- $T$ ... tags
- $S \subseteq U \times I \times T$ ... observed tags
- $P_S = \{(u, i) | \exists t \in T : (u, i, t) \in S\}$ ... observed tagging posts

[Hotho et al., 2006]
Evaluation: Prediction Quality

▶ adapted PageRank for tag recommendation/ Folkrank [Hotho et al. 2006]
▶ HOSVD: TD for least squares, no missing values, no reg. [Symeonidis et al. 2008]
▶ RTF-TD: TD model optimized for regularized ranking [Rendle et al. 2009]
▶ BPR-PITF, BPR-CD: PITF/ PARAFAC model optimized for regularized ranking [Rendle et al. 2010]

[Rendle et. al 2010]
Evaluation: Learning Runtime

Last.fm: Prediction quality vs. learning runtime

[Graph showing prediction quality vs. learning runtime for BPR−PITF 64, BPR−CD 64, and RTF−TD 64 over 30 days and 120 minutes.]

[Source: Rendle et. al 2010]
ECML/PKDD Discovery Challenge 2009

<table>
<thead>
<tr>
<th>Rank</th>
<th>Method</th>
<th>Top-5 F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>BPR-PITF</strong> + adaptive list size</td>
<td>0.35594</td>
</tr>
<tr>
<td>-</td>
<td><strong>BPR-PITF (not submitted)</strong></td>
<td>0.345</td>
</tr>
<tr>
<td>2</td>
<td>Relational Classification  [Marinho et al. 09]</td>
<td>0.33185</td>
</tr>
<tr>
<td>3</td>
<td>Content-based [Lipczak et al. 09]</td>
<td>0.32461</td>
</tr>
<tr>
<td>4</td>
<td>Content-based [Zhang et al. 09]</td>
<td>0.32230</td>
</tr>
<tr>
<td>5</td>
<td>Content-based [Ju and Hwang 09]</td>
<td>0.32134</td>
</tr>
<tr>
<td>6</td>
<td>Personomy translation [Wetzker et al. 09]</td>
<td>0.32124</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Task 2: ECML/ PKDD Challenge 2009,  
http://www.kde.cs.uni-kassel.de/ws/dc09/results

[Rendle et. al 2010]
Querying Incomplete RDF-Graphs

▶ Task: Answer queries about subject-predicate-pairs. E.g. What is McCartney member of?

Querying Incomplete RDF-Graphs

An RDF-Graph can be expressed as a function over three categorical domains: $y : S \times P \times O \rightarrow \{0, 1\}$

Prediction Quality

- CD Dense: PARAFAC optimized for least-squares, no missing values, no reg.
- CD-BPR: PARAFAC optimized for regularized ranking.
- PITF-BPR: PITF (pairwise interactions) optimized for regularized ranking.

[Drumond et al. 2012]
Other Applications: Examples

- **Multiverse Recommendation** [Karatzoglou et al. 2010]
  - Task: Context-aware Rating prediction.
  - Model: Tucker Decomposition.
  - Missing values are handled.
  - Loss: task dependent, e.g. MAE, RMSE.
  - Regularization: L1, L2.
  - Algorithm: Stochastic Gradient Descent (SGD).

- **CubeSVD** [Sun et al. 2005]
  - Task: Clickthrough prediction.
  - Approach: HOSVD.
Outline

Tensor Factorization

Problem Setting
Models
Learning
Examples for Applications
Summary

Time-aware Factorization Models

Factorization Machines
Summary

- Prediction functions with \( m \) categorical variables can be modeled with tensor factorization.

- Parallel Factor Analysis (PARAFAC) generalizes matrix factorization to \( m \) modes.

- Tucker Decomposition allows a free core tensor. (High computational complexity!)

- Lower-order interactions, e.g. pairwise ones should be integrated for better prediction quality in sparse settings.

- For learning: missing values, loss/likelihood and regularization/priors should be considered.
Summary

- Prediction functions with $m$ categorical variables can be modeled with tensor factorization.

- Parallel Factor Analysis (PARAFAC) generalizes matrix factorization to $m$ modes.

- Tucker Decomposition allows a free core tensor. (High computational complexity!)

- Lower-order interactions, e.g. pairwise ones should be integrated for better prediction quality in sparse settings.

- For learning: missing values, loss/likelihood and regularization/priors should be considered.

**Problem**: Only categorical variables can be handled.
Outline

Tensor Factorization

Time-aware Factorization Models

Summary

Factorization Machines
Time-Aware: Problem Setting

- 3 predictor variables:
  - two variables of categorical domain $I$ and $J$.
  - one numerical variable (time), $t \in \mathbb{R}$.

- Target $y$: Real-valued (regression), binary (classification), scores (ranking).

- Supervised task: set of observations $S = \{(i, j, t, y), \ldots\}$

- Modelling: function $\hat{y} : I \times J \times \mathbb{R} \to \mathcal{Y}$. 

Observations over $I$ and $J$

Observations over $I$ and $J$

Observations over $I$ and $J$

Observations over $I$ and $J$
Tensor Factorization

1. Discretize time variable, e.g. by binning. ⇒ 3 cat. domains: I, J, T.

   \[ b : \mathbb{R} \rightarrow T, \quad \text{e.g. } b(t) := \lfloor t / (24 \times 60 \times 60) \rfloor \]
Tensor Factorization

1. Discretize time variable, e.g. by binning. \(\Rightarrow\) 3 cat. domains: I, J, T.

\[ b : \mathbb{R} \rightarrow T, \quad \text{e.g. } b(t) := \lfloor t/(24 \times 60 \times 60) \rfloor \]

2. Apply tensor factorization, e.g. Tucker Decomposition, PARAFAC.

\[ \hat{y}(i, j, t) := \sum_{f=1}^{k} v_{i,f}^I v_{j,f}^J v_{b(t),f}^T \]

[Xiong et al. 2010]
Tensor Factorization

1. Discretize time variable, e.g. by binning. ⇒ 3 cat. domains: I, J, T.

   \[ b : \mathbb{R} \to T, \quad \text{e.g. } b(t) := \left\lfloor \frac{t}{24 \times 60 \times 60} \right\rfloor \]

2. Apply tensor factorization, e.g. Tucker Decomposition, PARAFAC.

   \[ \hat{y}(i, j, t) := \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f} \cdot v_{b(t),f}^T \]

3. Smooth time factors \( V^T \), s.th. nearby points in time have similar factors. E.g. by regularization:

   \[ v_{t+1,f}^T \sim \mathcal{N}(v_{t,f}^T, 1/\lambda_T), \quad \forall t \in T, f \in \{1, \ldots, k\} \]

[Xiong et al. 2010]
Tensor Factorization

1. Discretize time variable, e.g. by binning. ⇒ 3 cat. domains: I, J, T.

   \[ b : \mathbb{R} \to T, \text{ e.g. } b(t) := \lfloor t / (24 \times 60 \times 60) \rfloor \]

2. Apply tensor factorization, e.g. Tucker Decomposition, PARAFAC.

   \[ \hat{y}(i, j, t) := \sum_{f=1}^{k} v_{i,f}^I v_{j,f}^J v_{b(t),f}^T \]

3. Smooth time factors \( V^T \), s.th. nearby points in time have similar factors. E.g. by regularization:

   \[ v_{t+1,f}^T \sim \mathcal{N}(v_{t,f}^T, 1/\lambda_T), \quad \forall t \in T, f \in \{1, \ldots, k\} \]

For learning/ inference, e.g. a MCMC sampler can be used.

[Xiong et al. 2010]
Time-Aware Matrix Factorization

\[ \hat{y}(i, j, t) := \sum_{f=1}^{k} w_{i,f}(t) h_{j,f}(t) \]

where the factor matrices \( H \) and \( W \) depend on the time \( t \):

\[ W : \mathbb{R} \rightarrow \mathbb{R}^{I \times k}, \quad H : \mathbb{R} \rightarrow \mathbb{R}^{J \times k} \]

[Koren 2009]
Time-Aware Matrix Factorization

Modeling time dependent factors, e.g. for $\mathcal{W}$:

- Constant

$$w_{i,f}(t) := \tilde{w}_{i,f}, \quad \tilde{\mathcal{W}} \in \mathbb{R}^{l \times k}$$
Time-Aware Matrix Factorization

Modeling time dependent factors, e.g. for $W$:

- **Constant**

  \[ w_{i,f}(t) := \tilde{w}_{i,f}, \quad \tilde{W} \in \mathbb{R}^{I|\times k} \]

- **Linear**

  \[ w_{i,f}(t) := \tilde{w}_{i,f} + z_{i,f} t, \quad \tilde{W} \in \mathbb{R}^{I|\times k}, \quad Z \in \mathbb{R}^{I|\times k} \]

[Koren 2009]
Time-Aware Matrix Factorization

Modeling time dependent factors, e.g. for $W$:

- **Constant**
  
  $$w_{i,f}(t) := \tilde{w}_{i,f}, \quad \tilde{W} \in \mathbb{R}^{l \times k}$$

- **Linear**
  
  $$w_{i,f}(t) := \tilde{w}_{i,f} + z_{i,f} t, \quad \tilde{W} \in \mathbb{R}^{l \times k}, Z \in \mathbb{R}^{l \times k}$$

- **Binning with function $b$**
  
  $$w_{i,f}(t) := \tilde{w}_{i,f,b}(t), \quad \tilde{W} \in \mathbb{R}^{l \times k \times \text{img}(b)}$$
Time-Aware Matrix Factorization

Modeling time dependent factors, e.g. for $W$:

- **Constant**

  $$w_{i,f}(t) := \tilde{w}_{i,f}, \quad \tilde{W} \in \mathbb{R}^{I \times k}$$

- **Linear**

  $$w_{i,f}(t) := \tilde{w}_{i,f} + z_{i,f} t, \quad \tilde{W} \in \mathbb{R}^{I \times k}, Z \in \mathbb{R}^{I \times k}$$

- **Binning with function $b$**

  $$w_{i,f}(t) := \tilde{w}_{i,f,b(t)}, \quad \tilde{W} \in \mathbb{R}^{I \times k \times |img(b)|}$$

- **Spline with $m_i$ predefined control points at position $t_{i,1}, \ldots, t_{i,m}$**

  $$w_{i,f}(t) := \frac{\sum_{l=1}^{m_i} \tilde{w}_{i,f,l} \exp(-\gamma |t - t_{i,l}|)}{\sum_{l=1}^{m_i} \exp(-\gamma |t - t_{i,l}|)}, \quad \tilde{W} \in \mathbb{R}^{I \times k \times m_i}$$

[Koren 2009]
Time-Aware Matrix Factorization

Modeling time dependent factors, e.g. for $W$:

- **Constant**
  \[ w_{i,f}(t) := \tilde{w}_{i,f}, \quad \tilde{W} \in \mathbb{R}^{l \times k} \]

- **Linear**
  \[ w_{i,f}(t) := \tilde{w}_{i,f} + z_{i,f} t, \quad \tilde{W} \in \mathbb{R}^{l \times k}, Z \in \mathbb{R}^{l \times k} \]

- **Binning with function $b$**
  \[ w_{i,f}(t) := \tilde{w}_{i,f,b}(t), \quad \tilde{W} \in \mathbb{R}^{l \times k \times |\text{img}(b)|} \]

- **Spline with $m_i$ predefined control points at position $t_{i,1}, \ldots, t_{i,m}$**
  \[ w_{i,f}(t) := \sum_{l=1}^{m_i} \frac{\tilde{w}_{i,f,l} \exp(-\gamma|t - t_{i,l}|)}{\sum_{l=1}^{m_i} \exp(-\gamma|t - t_{i,l}|)}, \quad \tilde{W} \in \mathbb{R}^{l \times k \times m_i} \]

- **Linear combinations of the functions above.**

[Koren 2009]
Time-Aware Matrix Factorization

Choices for the \textit{timeSVD++} model for the Netflix challenge:

- User factors $W$: linear combination of
  - linear effect
  - binning with bin size 1

- Item factors $H$:
  - constant

- Additional (time-unaware) implicit indicators (from SVD++ [Koren, 2008])

[Koren 2009]
Choices for the *timeSVD++* model for the Netflix challenge:

- **User factors W:** linear combination of
  - linear effect
  - binning with bin size 1
- **Item factors H:**
  - constant
- Additional (time-unaware) implicit indicators (from SVD++ [Koren, 2008])

For learning, e.g. a SGD algorithm can be used.
Comparison

- Time-aware MF with binning (TAMF) and tensor factorization with discretization (TF) treat the time variable similarly:

\[
\hat{y}_{TAMF}(i, j, t) := \sum_{f=1}^{k} w_{i, f, b(t)} h_{j, f}
\]

\[
\hat{y}_{TF}(i, j, t) := \sum_{f=1}^{k} w_{i, f} h_{j, f} z_{b(t), f}
\]

- Main difference:
  - In tensor factorization, the (i,t)-interaction is factorized.
  - In time-aware MF, the (i,t)-interaction is modeled unfactorized.
Discussion

- Binning and splines cannot make use of time for future events.
  - Future bins are empty and variables cannot be estimated.
  - Variables in (future) control points of splines cannot be estimated.

- Seasonal time indicators can help, e.g. weekday, holiday, Christmas, etc,

- Other approach: use qualitative/ sequential information
Sequential Prediction

Task: Which items will be selected next?

[e.g. Zimdars et al. 2001, Rendle et al. 2010]
Markov Chains

Markov chain of order 1:

\[ p(j_t | l_{t-1}) \]

- \( t \) is a sequential index.
- \( l_{t-1} \) is the item selected previously.
- The Markov chain is defined by a transition matrix \( A \in \mathbb{R}^{|J| \times |J|} \).
Markov Chains

Markov chain of order 1:

\[ p(j_t | l_{t-1}) \]

- \( t \) is a sequential index.
- \( l_{t-1} \) is the item selected previously.
- The Markov chain is defined by a transition matrix \( A \in \mathbb{R}^{J \times |J|} \).
- Model is (weakly) personalized by taking the last item selected by a user into account.
Factorized Personalized Markov Chain

Model equation

\[ \hat{y}(i, j, t) := \hat{z}(i, j, s(i, t)) \]

where \( s(i, t) \) is the previously (w.r.t. \( t \)) selected entity (by \( i \)).

- \( \hat{z} \) can be modeled by TD, PARAFAC, PITF, . . .
- For product recommendation \( i \) is the user and \( j \) the current item.

[Rendle et al. 2010]
Factorized Personalized Markov Chain

Model equation

\[ \hat{y}(i, j, t) := \hat{z}(i, j, s(i, t)) \]

where \( s(i, t) \) is the previously (w.r.t. \( t \)) selected entity (by \( i \)).

- \( \hat{z} \) can be modeled by TD, PARAFAC, PITF, . . .
- For product recommendation \( i \) is the user and \( j \) the current item.
- If a set of items can be selected previously, one can average over this set:

\[ \hat{y}(i, j, t) := \frac{1}{|s(i, t)|} \sum_{l \in s(i, t)} \hat{z}(i, j, l) \]

For learning, e.g. a SGD algorithm can be used.

[Rendle et al. 2010]
Outline

Tensor Factorization

Time-aware Factorization Models

Models

Summary

Factorization Machines

Steffen Rendle 37 / 75 Social Network Analysis, University of Konstanz
Summary

- Time can be taken into account by:
  - Discretization and applying Tensor Factorization.
  - Time-variant factors, e.g. binning, linear effects, splines, ...
  - Sequential indicators, e.g. last item selected.

- With time-variables, the dataset split should be considered:
  - Random split: absolute time can be modeled.
  - Time split: binning not effective, time transformation that are predictive for future points in time should be chosen; e.g. seasonal or sequential.
Outline

Tensor Factorization

Time-aware Factorization Models

Factorization Machines
Problem Setting
Standard Models
Factorization Machines
Applications
Summary
Motivation

All the presented factorization models work empirically very well, but:

- For each new problem a new model, a new learning algorithm and implementation is necessary.
- For some of the models there are dozens of improved learning algorithms proposed (that work only with this particular model).
- For non-experts in factorization models this is not applicable.
- How does this relate to standard models?
Data and Variable Representation

Many standard ML approaches work with real valued input data (a *design matrix*). It allows to represent, e.g.:

- any number of variables
- categorical domains by using dummy indicator variables
- numerical domains
- set-categorical domains by using dummy indicator variables

Using this representation allows to apply a wide variety of standard models (e.g. linear regression, SVM, etc.).
Data and Variable Representation: Example

<table>
<thead>
<tr>
<th>User</th>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Titanic</td>
<td>5</td>
</tr>
<tr>
<td>Alice</td>
<td>Notting Hill</td>
<td>3</td>
</tr>
<tr>
<td>Alice</td>
<td>Star Wars</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>Star Wars</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>Star Trek</td>
<td>5</td>
</tr>
<tr>
<td>Charlie</td>
<td>Titanic</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>Star Wars</td>
<td>5</td>
</tr>
</tbody>
</table>

2 categorical variables
Data and Variable Representation: Example

<table>
<thead>
<tr>
<th>User</th>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Titanic</td>
<td>5</td>
</tr>
<tr>
<td>Alice</td>
<td>Notting Hill</td>
<td>3</td>
</tr>
<tr>
<td>Alice</td>
<td>Star Wars</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>Star Wars</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>Star Trek</td>
<td>5</td>
</tr>
<tr>
<td>Charlie</td>
<td>Titanic</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>Star Wars</td>
<td>5</td>
</tr>
</tbody>
</table>

2 categorical variables

$$|U| + |I|$$ real valued variables
Problem Setting

- Predictor variables: $p$ variables of real-valued domain $X_1, \ldots, X_p \in \mathbb{R}$.

- Target $y$: Real-valued (regression), binary (classification), scores (ranking).

- Supervised task: set of observations $S = \{(x_1, \ldots, x_p, y), \ldots\}$
Problem Setting

- **Predictor variables:** $p$ variables of real-valued domain $X_1, \ldots, X_p \in \mathbb{R}$.

- **Target $y$:** Real-valued (regression), binary (classification), scores (ranking).

- **Supervised task:** set of observations $S = \{(x_1, \ldots, x_p, y), \ldots\}$

This is the most common machine learning task.
Outline

Tensor Factorization

Time-aware Factorization Models

Factorization Machines
  Problem Setting
  Standard Models
  Factorization Machines
  Applications
  Summary
Standard Machine Learning Models

- Categorical variables can be represented with real-valued ones.
- There are many well-studied standard ML models that can work with real-valued variables.
- Why shouldn’t we work with them? Why do we need factorization models?
Linear Regression

- Let \( \mathbf{x} \in \mathbb{R}^p \) be an input vector with \( p \) predictor variables.
- Model equation:

\[
\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{p} w_i x_i
\]

- Model parameters:

\[
w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^p
\]

\( \mathcal{O}(p) \) model parameters.
Polynomial Regression

- Let \( x \in \mathbb{R}^p \) be an input vector with \( p \) predictor variables.
- Model equation (degree 2):

\[
\hat{y}(x) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j \geq i}^{p} w_{i,j} x_i x_j
\]

- Model parameters:

\[
w_0 \in \mathbb{R}, \quad w \in \mathbb{R}^p, \quad W \in \mathbb{R}^{p \times p}
\]

\( \mathcal{O}(p^2) \) model parameters.
Application to Large Categorical Domains

<table>
<thead>
<tr>
<th>User</th>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Titanic</td>
<td>5</td>
</tr>
<tr>
<td>Alice</td>
<td>Notting Hill</td>
<td>3</td>
</tr>
<tr>
<td>Alice</td>
<td>Star Wars</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>Star Wars</td>
<td>4</td>
</tr>
<tr>
<td>Bob</td>
<td>Star Trek</td>
<td>5</td>
</tr>
<tr>
<td>Charlie</td>
<td>Titanic</td>
<td>1</td>
</tr>
<tr>
<td>Charlie</td>
<td>Star Wars</td>
<td>5</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Applying regression models to this data leads to:

- Linear regression:
  \[ \hat{y}(x) = w_0 + w_u x_u + w_i x_i \]

- Polynomial regression:
  \[ \hat{y}(x) = w_0 + w_u x_u + w_i x_i + w_{u,i} x_u x_i \]

- Matrix factorization (with biases):
  \[ \hat{y}(u, i) = w_0 + w_u x_u + h_i + \langle w_u, h_i \rangle \]
Application to Large Categorical Domains

Applying regression models to this data leads to:

Linear regression: \( \hat{y}(x) = w_0 + w_u + w_i \)
Application to Large Categorical Domains

Applying regression models to this data leads to:

Linear regression:
\[ \hat{y}(x) = w_0 + w_u + w_i \]

Polynomial regression:
\[ \hat{y}(x) = w_0 + w_u + w_i + w_{u,i} \]
## Application to Large Categorical Domains

Applying regression models to this data leads to:

### Linear regression:

\[ \hat{y}(x) = w_0 + w_u + w_i \]

### Polynomial regression:

\[ \hat{y}(x) = w_0 + w_u + w_i + w_{u,i} \]

### Matrix factorization (with biases):

\[ \hat{y}(u, i) = w_0 + w_u + h_i + \langle w_u, h_i \rangle \]
Application to Large Categorical Domains

For the recommender data of the example:

- Linear regression has no user-item interaction.
Application to Large Categorical Domains

For the recommender data of the example:

- Linear regression has no user-item interaction.
  - \( \Rightarrow \) Linear regression is not expressive enough.
Application to Large Categorical Domains

For the recommender data of the example:

- Linear regression has no user-item interaction.
  - ⇒ Linear regression is not expressive enough.

- Polynomial regression includes pairwise interactions but cannot estimate them from the data.
Application to Large Categorical Domains

For the recommender data of the example:

- Linear regression has no user-item interaction.
  - $\Rightarrow$ Linear regression is not expressive enough.

- Polynomial regression includes pairwise interactions but cannot estimate them from the data.
  - $n \ll p^2$: number of cases is much smaller than number of model parameters.
Application to Large Categorical Domains

For the recommender data of the example:

- Linear regression has no user-item interaction.
  - ⇒ Linear regression is not expressive enough.

- Polynomial regression includes pairwise interactions but cannot estimate them from the data.
  - $n \ll p^2$: number of cases is much smaller than number of model parameters.
  - Max.-likelihood estimator for a pairwise effect is:

$$w_{i,j} = \begin{cases} y - w_0 - w_i - w_u, & \text{if } (i, j, y) \in S. \\ \text{not defined,} & \text{else} \end{cases}$$
Application to Large Categorical Domains

For the recommender data of the example:

- Linear regression has no user-item interaction.
  - $\Rightarrow$ Linear regression is not expressive enough.

- Polynomial regression includes pairwise interactions but cannot estimate them from the data.
  - $n \ll p^2$: number of cases is much smaller than number of model parameters.
  - Max.-likelihood estimator for a pairwise effect is:
    \[
    w_{i,j} = \begin{cases} 
    y - w_0 - w_i - w_u, & \text{if } (i,j,y) \in S. \\
    \text{not defined}, & \text{else}
    \end{cases}
    \]
  - Polynomial regression cannot generalize to any unobserved pairwise effect.
Factorization Models and Real-valued Variables

- Factorization models work well for categorical variables of large domain.

- Standard Models are more flexible as they allow real-valued predictor variables that can be used for encoding several kind of variables.
Factorization Models and Real-valued Variables

- Factorization models work well for categorical variables of large domain.

- Standard Models are more flexible as they allow real-valued predictor variables that can be used for encoding several kind of variables.

- How can these advantages be combined?
Outline

Tensor Factorization

Time-aware Factorization Models

Factorization Machines
  Problem Setting
  Standard Models

Factorization Machines
  Applications
  Summary
Factorization Machine (FM)

- Let $\mathbf{x} \in \mathbb{R}^p$ be an input vector with $p$ predictor variables.
- Model equation (degree 2):

$$
\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j>i} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j
$$

- Model parameters:

$$
w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^p, \quad \mathbf{V} \in \mathbb{R}^{p \times k}
$$

[Rendle 2010, Rendle 2012]
Factorization Machine (FM)

- Let $x \in \mathbb{R}^p$ be an input vector with $p$ predictor variables.
- Model equation (degree 2):

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

- Model parameters:

$$w_0 \in \mathbb{R}, \quad w \in \mathbb{R}^p, \quad V \in \mathbb{R}^{p \times k}$$

Compared to Polynomial regression:
- Model equation (degree 2):

$$\hat{y}(x) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j \geq i} w_{i,j} x_i x_j$$

- Model parameters:

$$w_0 \in \mathbb{R}, \quad w \in \mathbb{R}^p, \quad W \in \mathbb{R}^{p \times p}$$

[Rendle 2010, Rendle 2012]
Factorization Machine (FM)

- Let $x \in \mathbb{R}^p$ be an input vector with $p$ predictor variables.
- Model equation (degree 2):

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

- Model parameters:

  $$w_0 \in \mathbb{R}, \quad w \in \mathbb{R}^p, \quad V \in \mathbb{R}^{p \times k}$$

[Rendle 2010, Rendle 2012]
Factorization Machine (FM)

- Let $x \in \mathbb{R}^p$ be an input vector with $p$ predictor variables.
- Model equation (degree 3):

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

$$+ \sum_{i=1}^{p} \sum_{j>i}^{p} \sum_{l>j}^{p} \sum_{f=1}^{k} v_{i,f}^{(3)} v_{j,f}^{(3)} v_{l,f}^{(3)} x_i x_j x_l$$

- Model parameters:

$$w_0 \in \mathbb{R}, \quad w \in \mathbb{R}^p, \quad V \in \mathbb{R}^{p \times k}, \quad V^{(3)} \in \mathbb{R}^{p \times k}$$

[Rendle 2010, Rendle 2012]
Factorization Machines: Discussion

- FMs work with real valued input.
- FMs include variable interactions like polynomial regression.
- Model parameters for interactions are factorized.
- Number of model parameters is $O(kp)$ (instead of $O(p^2)$ for poly. regr.).
Factorization Machines: Discussion

- FMs work with real valued input.
- FMs include variable interactions like polynomial regression.
- Model parameters for interactions are factorized.
- Number of model parameters is $O(k p)$ (instead of $O(p^2)$ for poly. regr.).
- How are FMs related to the factorization models we have seen so far?
Matrix Factorization and Factorization Machines

Two categorical variables encoded with real valued predictor variables:

With this data, the FM is identical to MF with biases:

\[
\hat{y}(x) = w_0 + w_u + w_i + \langle v_u, v_i \rangle
\]
Tag-Recommendation with Factorization Machines

Three categorical variables encoded with real valued predictor variables:

With this data, the FM is a tensor factorization model with lower-level interactions (here up to pairwise ones):

$$\hat{y}(x) := w_0 + w_i + w_u + w_t + \langle v_u, v_t \rangle + \langle v_i, v_t \rangle + \langle v_u, v_i \rangle$$
Time with Factorization Machines

Two categorical variables and time as linear predictor:

\[ \hat{y}(x) := w_0 + w_i + w_u + t w_{\text{time}} + \langle v_u, v_i \rangle + t \langle v_u, v_{\text{time}} \rangle + t \langle v_i, v_{\text{time}} \rangle \]
Time with Factorization Machines

Two categorical variables and time discretized in bins \((b(t))\):

With this data, a three-order FM includes the time-aware tensor factorization model described before:

\[
\hat{y}(x) := w_0 + w_i + w_u + w_{b(t)} + \langle v_u, v_i \rangle + \langle v_u, v_{b(t)} \rangle + \langle v_i, v_{b(t)} \rangle + \sum_{f=1}^{k} v^{(3)}_{u,f} v^{(3)}_{i,f} v^{(3)}_{b(t),f}
\]
Time with Factorization Machines

Two categorical variables and time discretized in bins ($b(t)$):

With this data, an FM includes the time-aware matrix factorization model with binned user-time interactions:

$$\hat{y}(x) := w_0 + w_i + w_{u,b(t)} + \left\langle v_{u,b(t)}, v_i \right\rangle$$

[MF with time variant factors]

[Koren, 2009]
With this data, the FM is identical to:

$$\hat{y}(x) = w_0 + w_u + w_i + \langle v_u, v_i \rangle + \frac{1}{\sqrt{|N_u|}} \sum_{l \in N_u} \langle v_i, v_l \rangle$$

$$+ \frac{1}{\sqrt{|N_u|}} \sum_{l \in N_u} \left( w_l + \langle v_u, v_l \rangle + \frac{1}{\sqrt{|N_u|}} \sum_{l' \in N_u, l' > l} \langle v_l, v_{l'} \rangle \right)$$

[Koren, 2008]
Factorization Machines: Discussion II

- Representing categorical variables with real-valued variables and applying FMs is comparable to the factorization models that have been derived individually before (e.g. (bias) MF, tensor factorization, SVD++).

- FMs are much more flexible and can handle also non-categorical variables.

- Applying FMs is simple, as only data preprocessing has to be done (defining the real-valued predictor variables).
Computation Complexity

Factorization Machine model equation:

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

- Trivial computation: $O(p^2 k)$
Computation Complexity

Factorization Machine model equation:

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

- Trivial computation: \( O(p^2 k) \)
- Efficient computation can be done in: \( O(p k) \)
Computation Complexity

Factorization Machine model equation:

\[
\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j>i} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j
\]

- Trivial computation: \(O(p^2 k)\)
- Efficient computation can be done in: \(O(pk)\)
- Making use of many zeros in \(\mathbf{x}\) even in: \(O(N_z(\mathbf{x}) k)\), where \(N_z(\mathbf{x})\) is the number of non-zero elements in vector \(\mathbf{x}\).
Efficient Computation

The model equation of an FM can be computed in $O(pk)$.
Efficient Computation

The model equation of an FM can be computed in $O(pk)$.

Proof:

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

$$= w_0 + \sum_{i=1}^{p} w_i x_i + \frac{1}{2} \sum_{f=1}^{k} \left[ \left( \sum_{i=1}^{p} x_i v_{i,f} \right)^2 - \sum_{i=1}^{p} \left( x_i v_{i,f} \right)^2 \right]$$
Efficient Computation

The model equation of an FM can be computed in $O(p k)$.

Proof:

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

$$= w_0 + \sum_{i=1}^{p} w_i x_i + \frac{1}{2} \sum_{f=1}^{k} \left[ \left( \sum_{i=1}^{p} x_i v_{i,f} \right)^2 - \sum_{i=1}^{p} (x_i v_{i,f})^2 \right]$$

- In the sums over $i$, only non-zero $x_i$ elements have to be summed up
  $\Rightarrow O(N_z(x) k)$.
- (The complexity of polynomial regression is $O(N_z(x)^2)$.)
Multilinearity

FMs are multilinear:

\[
\forall \theta \in \Theta = \{w_0, w, V\} : 
\hat{y}(\mathbf{x}, \theta) = h(\theta)(\mathbf{x}) \theta + g(\theta)(\mathbf{x})
\]

where \(g(\theta)\) and \(h(\theta)\) do not depend on the value of \(\theta\).
Multilinearity

FMs are multilinear:

$$\forall \theta \in \Theta = \{w_0, w, V\} : \hat{y}(x, \theta) = h(\theta)(x) \theta + g(\theta)(x)$$

where $g(\theta)$ and $h(\theta)$ do not depend on the value of $\theta$.

E.g. for second order effects ($\theta = v_{l,f}$):

$$\hat{y}(x, v_{l,f}) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} \sum_{f'=1}^{k} V_{i,f'} V_{j,f'} x_i x_j$$

$$+ v_{l,f} x_l \sum_{i=1, i \neq l}^{p} V_{i,f} x_i$$

$$h(v_{l,f})(x)$$
Learning

Using these properties, learning algorithms can be developed:

- L2-regularized regression and classification:
  - Stochastic gradient descent [Rendle, 2010]
  - Alternating least squares/ Coordinate Descent [Rendle et al., 2011, Rendle 2012]
  - Markov Chain Monte Carlo (for Bayesian FMs) [Freudenthaler et al. 2011, Rendle 2012]

- L2-regularized ranking:
  - Stochastic gradient descent [Rendle, 2010]

All the proposed learning algorithms have a runtime of $O(k N_z(X) i)$, where $i$ is the number of iterations and $N_z(X)$ the number of non-zero elements in the design matrix $X$. 
Stochastic Gradient Descent (SGD)

- For each training case \((x, y) \in S\), SGD updates the FM model parameter \(\theta\) using:

\[
\theta' = \theta - \alpha \left( (\hat{y}(x) - y) h_\theta(x) + \lambda_\theta \theta \right)
\]

- \(\alpha\) is the learning rate / step size.
- \(\lambda_\theta\) is the regularization value of the parameter \(\theta\).
- SGD can easily be applied to other loss functions.

[Rendle, 2010]
### Alternating Least Squares (ALS)

- Elementwise ALS updates each FM model parameter $\theta$ using:

  $$
  \theta' = - \frac{\sum_{(x,y) \in S} (g(\theta)(x) - y) h(\theta)(x)}{\sum_{(x,y) \in S} h^2(\theta)(x) + \lambda(\theta)}
  $$

- Using caches of intermediate results, the runtime for updating all model parameters is $O(k N_z(X))$.

- The advantage of ALS compared to SGD is that no learning rate has to be specified.

- ALS can be extended to classification [Rendle, 2012].

[Rendle et al., 2011]
Bayesian FMs (BFM)

\[ w_0 \sim \mathcal{N}(\mu_{w_0}, 1/\lambda_{w_0}), \quad \forall j \in \{1, \ldots, p\} : \quad w_j \sim \mathcal{N}(\mu_w, 1/\lambda_w), \quad v_j \sim \mathcal{N}(\mu_v, \Lambda_v) \]

\[ \mu_w \sim \mathcal{N}(\mu_0, \gamma_0\lambda_w), \quad \lambda_w \sim \Gamma(\alpha_\lambda, \beta_\lambda), \quad \mu_v, f \sim \mathcal{N}(\mu_0, \gamma_0\lambda_v, f), \quad \lambda_v, f \sim \Gamma(\alpha_\lambda, \beta_\lambda) \]

[Freudenthaler et al., 2011]
Bayesian FMs (BFM)

- The SGD and ALS models correspond to the left model.
- The right side is a two level model that integrates priors.

[Freudenthaler et al., 2011]
Bayesian FMs (BFM): Inference

- For Bayesian inference an efficient Gibbs sampler can be derived.
- The Gibbs posterior distribution for each model parameter $\theta$ is related to the ALS.
- Sampling all model parameters once can be done in $O(k N_z(X))$ as well.
- Introducing hyperpriors and integrating over priors has the advantage over ALS that the values of the priors are ‘automatically’ found.
- BFM$s$ can be extended to classification [Rendle, 2012].

[Freudenthaler et al., 2011]
Outline

Tensor Factorization

Time-aware Factorization Models

Factorization Machines
  Problem Setting
  Standard Models
  Factorization Machines

Applications

Summary
Applications

FM$s$ are especially suited for ML problems:

- Categorical variables of large domain.
- Number of predictor variables is large.
- Interactions between predictor variables are of interest.
- Several variables involved.
(Context-aware) Recommender Systems

- Main variables:
  - User ID (categorical)
  - Item ID (categorical)

- Additional variables:
  - time
  - mood
  - user profile
  - item meta data
  - ...

- Examples: Netflix prize, Movielens, KDDCup 2011
Clickthrough Prediction

- **Main variables:**
  - User ID
  - Query ID
  - Ad/ Link ID

- **Additional variables:**
  - query tokens
  - user profile
  - ...

- **Example:** KDDCup 2012 Track 2 (FM placed 3rd/171)
Student Performance Prediction

- Main variables:
  - Student ID
  - Question ID

- Additional variables:
  - question hierarchy
  - sequence of questions
  - skills required
  - . . .

- Examples: KDDCup 2010, Grockit Challenge\(^2\) (FM placed 1st/241)

\(^2\)http://www.kaggle.com/c/WhatDoYouKnow
Link Prediction in Social Networks

- **Main variables:**
  - Actor A ID
  - Actor B ID

- **Additional variables:**
  - profiles
  - actions
  - ...

- **Example: KDDCup 2012 Track 1 (FM placed 2nd/658)**
libFM Software

libFM is an implementation of FMs

- Model: second-order FMs
- Learning/ inference: SGD, ALS, MCMC
- Classification and regression
- Uses the same data format as LIBSVM, LIBLINEAR [Lin et. al], SVMlight [Joachims].
- Supports variable grouping.
- Available with source code.

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

Tensor Factorization

Time-aware Factorization Models

Factorization Machines
  Problem Setting
  Standard Models
  Factorization Machines
  Applications
  Summary
Summary

- Real-valued predictor variables can encode information from variables of other domains, e.g. categorical variables.
- Applying linear regression to large categorical domains results in too little expressiveness; applying polynomial regression results in too much expressiveness.
- Factorization Machines (FM) are a polynomial regression model with factorized interaction parameters.
- FMs bring together the generality of standard machine learning methods with the prediction quality of factorization models.
- FMs are multilinear and can be computed efficiently.


L. Drumond, S. Rendle, and L. Schmidt-Thieme. Predicting rdf triples in incomplete knowledge bases with tensor factorization.


Y. Koren.  
Collaborative filtering with temporal dynamics.  

L. D. Lathauwer, B. D. Moor, and J. Vandewalle.  
A multilinear singular value decomposition.  

S. Rendle.  
Factorization machines.  

S. Rendle.  
Factorization machines with libfm.  

S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme.  
Factorizing personalized markov chains for next-basket recommendation.


L. Tucker. Some mathematical notes on three-mode factor analysis.
