Lecture 15: Matrix Factorization

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science & Information Engineering
National Taiwan University
(國立台灣大學資訊工程系)
Roadmap

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 14: Radial Basis Function Network

- Linear aggregation of distance-based similarities using \( k \)-Means clustering for prototype finding

Lecture 15: Matrix Factorization

- Linear Network Hypothesis
- Basic Matrix Factorization
- Stochastic Gradient Descent
- Summary of Extraction Models
Recommender System Revisited

**data** → **ML** → **skill**

- **data**: how ‘many users’ have rated ‘some movies’
- **skill**: predict how a user would rate an unrated movie

**A Hot Problem**

- competition held by Netflix in 2006
  - 100,480,507 ratings that 480,189 users gave to 17,770 movies
  - 10% improvement = 1 million dollar prize
- data $D_m$ for $m$-th movie:
  \[
  \{ (\tilde{x}_n = (n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m \} 
  \]
  —abstract feature $\tilde{x}_n = (n)$

how to learn our preferences from data?
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) \]: user IDs, such as 1126, 5566, 6211, \ldots —called categorical features

- **categorical** features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, \ldots
- many ML models operate on **numerical** features
  - linear models
  - extended linear models such as NNet
    —except for **decision trees**
- need: **encoding (transform)** from categorical to numerical

**binary vector encoding**:

\[
A = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T, \quad B = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T, \\
AB = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T, \quad O = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T
\]
**Feature Extraction from Encoded Vector**

**encoded** data $D_m$ for $m$-th movie:

$$\left\{ (x_n = \text{BinaryVectorEncoding}(n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m \right\}$$

or, joint data $D$

$$\left\{ (x_n = \text{BinaryVectorEncoding}(n), y_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \ldots \ r_{nM}]^T) \right\}$$

**idea**: try **feature extraction** using $N-\tilde{d}-M$ NNet without all $x_0^{(\ell)}$.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$W_{ni}^{(1)}$$

$$W_{im}^{(2)}$$

$$y = \begin{bmatrix} \approx y_1 \\ \approx y_2 \\ \approx y_3 \end{bmatrix}$$

**is tanh necessary? :-)**
Matrix Factorization

Linear Network Hypothesis

‘Linear Network’ Hypothesis

\[
\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad \mathbf{V}^T : \mathbf{w}^{(1)}_{ni} \quad \mathbf{W} : \mathbf{w}^{(2)}_{im} \quad \approx y_1 \\ \approx y_2 \quad = \mathbf{y} \\ \approx y_3
\]

\[
\begin{cases}
(x_n = \text{BinaryVectorEncoding}(n), y_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ldots \ r_{nM}]^T) \end{cases}
\]

- rename: \( \mathbf{V}^T \) for \( \left[ \mathbf{w}^{(1)}_{ni} \right] \) and \( \mathbf{W} \) for \( \left[ \mathbf{w}^{(2)}_{im} \right] \)
- hypothesis: \( \mathbf{h}(\mathbf{x}) = \mathbf{W}^T \mathbf{V} \mathbf{x} \)
- per-user output: \( \mathbf{h}(\mathbf{x}_n) = \mathbf{W}^T \mathbf{v}_n \), where \( \mathbf{v}_n \) is \( n \)-th column of \( \mathbf{V} \)

linear network for recommender system:

learn \( \mathbf{V} \) and \( \mathbf{W} \)
For $N$ users, $M$ movies, and $\tilde{d}$ ‘features’, how many variables need to be used to specify a linear network hypothesis $h(x) = W^T V x$?

1. $N + M + \tilde{d}$
2. $N \cdot M \cdot \tilde{d}$
3. $(N + M) \cdot \tilde{d}$
4. $(N \cdot M) + \tilde{d}$
For $N$ users, $M$ movies, and $\tilde{d}$ ‘features’, how many variables need to be used to specify a linear network hypothesis $h(x) = W^T V x$?

1. $N + M + \tilde{d}$
2. $N \cdot M \cdot \tilde{d}$
3. $(N + M) \cdot \tilde{d}$
4. $(N \cdot M) + \tilde{d}$

Reference Answer: \(\boxed{3}\)

simply $N \cdot \tilde{d}$ for $V^T$ and $\tilde{d} \cdot M$ for $W$
Linear Network: Linear Model Per Movie

linear network:

\[ h(x) = W^T \Phi(x) \]

— for \( m \)-th movie, just linear model \( h_m(x) = \mathbf{w}_m^T \Phi(x) \)

subject to shared transform \( \Phi \)

- for every \( \mathcal{D}_m \), want \( r_{nm} = y_n \approx \mathbf{w}_m^T \mathbf{v}_n \)
- \( E_{in} \) over all \( \mathcal{D}_m \) with squared error measure:

\[
E_{in}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \frac{1}{\sum_{m=1}^{M} |\mathcal{D}_m|} \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - \mathbf{w}_m^T \mathbf{v}_n \right)^2
\]

linear network: transform and linear modelS jointly learned from all \( \mathcal{D}_m \)
Matrix Factorization

\[ r_{nm} \approx w_m^T v_n = v_n^T w_m \iff R \approx V^T W \]

<table>
<thead>
<tr>
<th></th>
<th>movie(_1)</th>
<th>movie(_2)</th>
<th>\ldots</th>
<th>movie(_M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>user(_1)</td>
<td>100</td>
<td>80</td>
<td>\ldots</td>
<td>?</td>
</tr>
<tr>
<td>user(_2)</td>
<td>?</td>
<td>70</td>
<td>\ldots</td>
<td>90</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>user(_N)</td>
<td>?</td>
<td>60</td>
<td>\ldots</td>
<td>0</td>
</tr>
</tbody>
</table>

Matrix Factorization Model

learning:
- known rating
  - \( \rightarrow \) learned factors \( v_n \) and \( w_m \)
  - \( \rightarrow \) unknown rating prediction

similar modeling can be used for other abstract features
Matrix Factorization Learning

\[
\min_{\mathbf{W}, \mathbf{V}} E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \\
= \sum_{m=1}^M \left( \sum_{(x_n, r_{nm}) \in D_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \right)
\]

- two sets of variables: can consider alternating minimization, remember? :-)
- when \( \mathbf{v}_n \) fixed, minimizing \( \mathbf{w}_m \equiv \) minimize \( E_{\text{in}} \) within \( D_m \)
  —simply per-movie (per-\( D_m \)) linear regression without \( w_0 \)
- when \( \mathbf{w}_m \) fixed, minimizing \( \mathbf{v}_n \)?
  —per-user linear regression without \( v_0 \)
  by symmetry between users/movies

called alternating least squares algorithm
Alternating Least Squares

1. Initialize $\tilde{d}$ dimension vectors $\{w_m\}, \{v_n\}$

2. Alternating optimization of $E_{in}$: repeatedly

   1. Optimize $w_1, w_2, \ldots, w_M$:
      update $w_m$ by $m$-th-movie linear regression on $\{(v_n, r_{nm})\}$

   2. Optimize $v_1, v_2, \ldots, v_N$:
      update $v_n$ by $n$-th-user linear regression on $\{(w_m, r_{nm})\}$

until converge

- initialize: usually just randomly
- converge: guaranteed as $E_{in}$ decreases during alternating minimization

Alternating least squares:
the ‘tango’ dance between users/movies
Linear Autoencoder versus Matrix Factorization

<table>
<thead>
<tr>
<th>Linear Autoencoder</th>
<th>Matrix Factorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \approx W (W^T X)$</td>
<td>$R \approx V^T W$</td>
</tr>
<tr>
<td>• motivation: special $d$-$\tilde{d}$-$d$ linear NNet</td>
<td>• motivation: $N$-$\tilde{d}$-$M$ linear NNet</td>
</tr>
<tr>
<td>• error measure: squared on all $x_{ni}$</td>
<td>• error measure: squared on known $r_{nm}$</td>
</tr>
<tr>
<td>• solution: global optimal at eigenvectors of $X^T X$</td>
<td>• solution: local optimal via alternating least squares</td>
</tr>
<tr>
<td>• usefulness: extract dimension-reduced features</td>
<td>• usefulness: extract hidden user/movie features</td>
</tr>
</tbody>
</table>

linear autoencoder $\equiv$ special matrix factorization of complete $X$
How many least squares problems does the alternating least squares algorithm needs to solve in one iteration of alternation?

1. number of movies $M$
2. number of users $N$
3. $M + N$
4. $M \cdot N$
How many least squares problems does the alternating least squares algorithm needs to solve in one iteration of alternation?

1. number of movies $M$
2. number of users $N$
3. $M + N$
4. $M \cdot N$

Reference Answer: 3

simply $M$ per-movie problems and $N$ per-user problems
Another Possibility: Stochastic Gradient Descent

\[ E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2 \]

SGD: randomly pick one example within the \( \sum \) & update with gradient to per-example err, remember? :-)

- ‘efficient’ per iteration
- simple to implement
- easily extends to other err

next: SGD for matrix factorization
Gradient of Per-Example Error Function

\[
\text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2
\]

\[
\nabla v_{1126} \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126
\]

\[
\nabla w_{6211} \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211
\]

\[
\nabla v_n \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) w_m
\]

\[
\nabla w_m \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) v_n
\]

\[
\text{per-example gradient} \propto - (\text{residual})(\text{the other feature vector})
\]
initialize $\tilde{d}$ dimension vectors $\{w_m\}, \{v_n\}$ randomly
for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
2. calculate residual $\tilde{r}_{nm} = (r_{nm} - w_m^T v_n)$
3. SGD-update:

$$
\begin{align*}
\textbf{v}_{n}^{\text{new}} & \leftarrow \textbf{v}_{n}^{\text{old}} + \eta \cdot \tilde{r}_{nm} \textbf{w}_{m}^{\text{old}} \\
\textbf{w}_{m}^{\text{new}} & \leftarrow \textbf{w}_{m}^{\text{old}} + \eta \cdot \tilde{r}_{nm} \textbf{v}_{n}^{\text{old}}
\end{align*}
$$

SGD: perhaps most popular large-scale matrix factorization algorithm
Matrix Factorization
Stochastic Gradient Descent

SGD for Matrix Factorization in Practice

KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings *earlier than* test ratings in time
- training/test mismatch: typical *sampling bias, remember? :-)*

- want: *emphasize latter* examples
- *last* $T'$ iterations of SGD: *only those* $T'$ examples considered —learned $\{w_m\}, \{v_n\}$ favoring those
- our idea: *time-deterministic SGD* that visits *latter* examples *last* —consistent improvements of test performance

if you *understand* the behavior of techniques, easier to *modify* for your real-world use
If all $w_m$ and $v_n$ are initialized to the 0 vector, what will NOT happen in SGD for matrix factorization?

1. all $w_m$ are always 0
2. all $v_n$ are always 0
3. every residual $\tilde{r}_{nm} = \text{the original rating } r_{nm}$
4. $E_{in}$ decreases after each SGD update
If all $\mathbf{w}_m$ and $\mathbf{v}_n$ are initialized to the $\mathbf{0}$ vector, what will NOT happen in SGD for matrix factorization?

1. all $\mathbf{w}_m$ are always $\mathbf{0}$
2. all $\mathbf{v}_n$ are always $\mathbf{0}$
3. every residual $\tilde{r}_{nm} =$ the original rating $r_{nm}$
4. $E_{in}$ decreases after each SGD update

Reference Answer: 4

The $\mathbf{0}$ feature vectors provides a per-example gradient of $\mathbf{0}$ for every example. So $E_{in}$ cannot be further decreased.
Matrix Factorization

Summary of Extraction Models

**Map of Extraction Models**

**Extraction models:** feature transform $\Phi$ as hidden variables in addition to linear model

**Adaptive/Gradient Boosting**
- Hypotheses $g_t$; weights $\alpha_t$

**Neural Network/Deep Learning**
- Weights $w_{ij}^{(\ell)}$
- Weights $w_{ij}^{(L)}$

**RBF Network**
- RBF centers $\mu_m$
- Weights $\beta_m$

**Matrix Factorization**
- User features $v_n$
- Movie features $w_m$

**k Nearest Neighbor**
- $x_n$-neighbor RBF
- Weights $y_n$

**Extraction models:** a rich family
Map of Extraction Techniques

- **Adaptive/Gradient Boosting**: functional gradient descent
- **Neural Network/Deep Learning**: SGD (backprop), autoencoder
- **RBF Network**: k-means clustering
- **Matrix Factorization**: SGD, alternating leastSQR

**k Nearest Neighbor**: lazy learning :-)

Extraction techniques: quite diverse
Pros and Cons of Extraction Models

Pros

- 'easy': reduces human burden in designing features
- powerful: if enough hidden variables considered

Cons

- 'hard': non-convex optimization problems in general
- overfitting: needs proper regularization/validation

be careful when applying extraction models
Which of the following extraction model extracts Gaussian centers by *k*-means and aggregate the Gaussians linearly?

1. RBF Network
2. Deep Learning
3. Adaptive Boosting
4. Matrix Factorization
Which of the following extraction model extracts Gaussian centers by $k$-means and aggregate the Gaussians linearly?

1. RBF Network
2. Deep Learning
3. Adaptive Boosting
4. Matrix Factorization

Reference Answer: 1

Congratulations on being an expert in extraction models! :-)

Hsuan-Tien Lin (NTU CSIE)
Summary

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 15: Matrix Factorization
- Linear Network Hypothesis
  - feature extraction from binary vector encoding
- Basic Matrix Factorization
  - alternating least squares between user/movie
- Stochastic Gradient Descent
  - efficient and easily modified for practical use
- Summary of Extraction Models
  - powerful thus need careful use

• next: closing remarks of techniques