

# Lecture 15: Matrix Factorization

课件链接: [Hsuan-Tien Lin - matrix factorization](#)

## Matrix Factorization(矩阵分解)

- Linear Network Hypothesis: 线性网络形式的假说
- Basic Matrix Factorization: 基本矩阵分解
- Stochastic Gradient Descent: 随机梯度下降
- Summary of Extraction Models: 萃取模型的总结

## 1. Linear Network Hypothesis: 线性网络形式的假说

假设我们需要构建一套针对用户的电影**推荐系统**。我们所拥有的数据是一些用户对一些电影的打分, 我们希望这个推荐系统具有的能力是: 预测某个用户对一个没有看过的电影的评分。

对第 $m$ 个电影来说的数据集 $\mathcal{D}_m$ 为:  $n$ 表示用户的index,  $m$ 表示电影的index

$$\{(\tilde{\mathbf{x}}_n = (n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m\}$$

可见, 数据中的输入特征就是一个简单的编号, 例如: 1126, 5566, 6211等。这种ID类型的特征, 被称为**抽象的特征(abstract feature)**, 因为其内在的含义往往不是那么明显。同时, 由于某个人不一定看过所有部电影, 且某部电影不一定被所有人看过, 因此 $r_{nm}$ 可能有一部分是缺失的。

### 对分类特征(categorical feature)进行二元向量编码(binary vector encoding)

一些常见的分类特征有:

- ID: 1126, 5566, 6211.....
- 血型: A, B, AB, O
- 程序语言: Java, C, C++, Python.....

目前所学的大多数模型都是在**数值特征**上操作的, 除了决策树(RF, GBDT)可以直接处理分类特征。因此, 需要将分类特征编码以转换为数值特征——encoding——其中, **binary vector encoding**最常用。例如, 对于血型进行编码:

- $A = [1 \ 0 \ 0 \ 0]^T$
- $B = [0 \ 1 \ 0 \ 0]^T$
- $AB = [0 \ 0 \ 1 \ 0]^T$
- $O = [0 \ 0 \ 0 \ 1]^T$

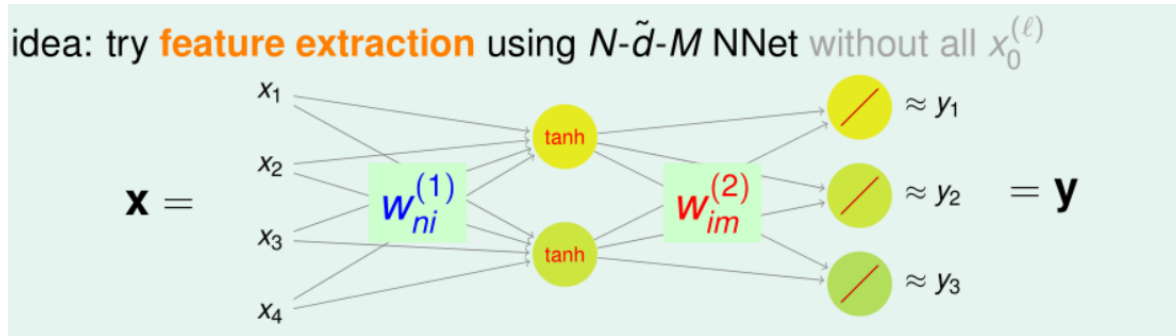
经过编码后, 数据集变为:

$$\{(\tilde{\mathbf{x}}_n = \text{BinaryVectorEncoding}(n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m\}$$

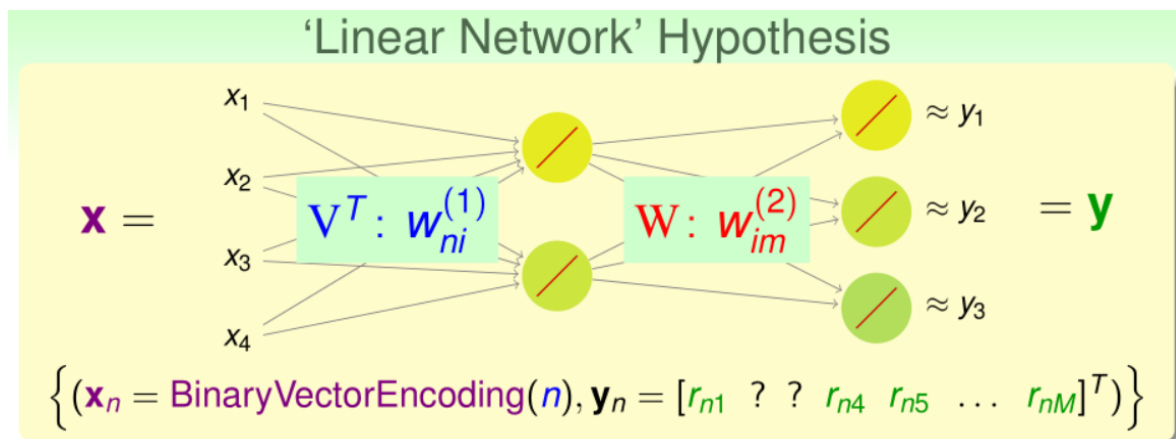
将所有电影的资料整合起来,  $\mathcal{D}$ :

$$\{(\tilde{\mathbf{x}}_n = \text{BinaryVectorEncoding}(n), y_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \cdots \ r_{nM}]^T)\}$$

### 使用神经网络进行特征萃取



其中,  $\mathbf{x}$  是表示某个用户的稀疏向量, 只有一个位置是1, 其余是0;  $\mathbf{y}$  是该用户对于所有  $M$  个电影的评分向量, **中间层就是萃取出的特征**。由于输入向量中只有一个位置是1, 因此可以忽略中间层的  $\tanh$  变换:



重命名权重矩阵:

- 第一层的所有权重:  $\begin{bmatrix} w_{ni}^{(1)} \end{bmatrix} = \mathbf{V}^T$ ,  $\mathbf{V}^T$  是  $N \times \tilde{d}$
- 第二层的所有权重:  $\begin{bmatrix} w_{im}^{(2)} \end{bmatrix} = \mathbf{W}$ ,  $\mathbf{W}$  是  $\tilde{d} \times M$

Hypothesis:

$$h(\mathbf{x}) = \mathbf{W}^T \mathbf{V} \mathbf{x}$$

对某一个用户, 其输出为:

$$h(\mathbf{x}_n) = \mathbf{W}^T \mathbf{V} \mathbf{x}_n = \mathbf{W}^T \mathbf{v}_n$$

其中  $\mathbf{v}_n$  是矩阵  $\mathbf{V}$  的第  $n$  列。

接下来, 就是学习  $\mathbf{V}$  与  $\mathbf{W}$ 。

## 2. Basic Matrix Factorization: 基本矩阵分解

对第  $m$  部电影, 线性模型为(取输出的第  $m$  个分量):

$$h_m(\mathbf{x}) = \mathbf{w}_m^T \mathbf{V} \mathbf{x} = \mathbf{w}_m^T \Phi(\mathbf{x})$$

可以看到, 所有的电影模型共用一个转换  $\Phi$ , 将抽象的特征  $\mathbf{x}$  转换成具体的描述。其中,  $\Phi(\mathbf{x}_n) = \mathbf{v}_n$ 。对于所有  $D_m$  (第  $m$  部电影来说的数据集), 我们希望:

$$r_{nm} = y_n \approx \mathbf{w}_m^T \mathbf{v}_n$$

将求解模型转换为最佳化  $E_{in}$  的问题:

$$E_{in}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \text{constant} \cdot \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2$$

解决上述最佳化问题，等价于通过最上面非常简单的两层线性网络，同时学到了两个东西：

- transform:  $\mathbf{V}$
- linear models:  $\mathbf{W}$

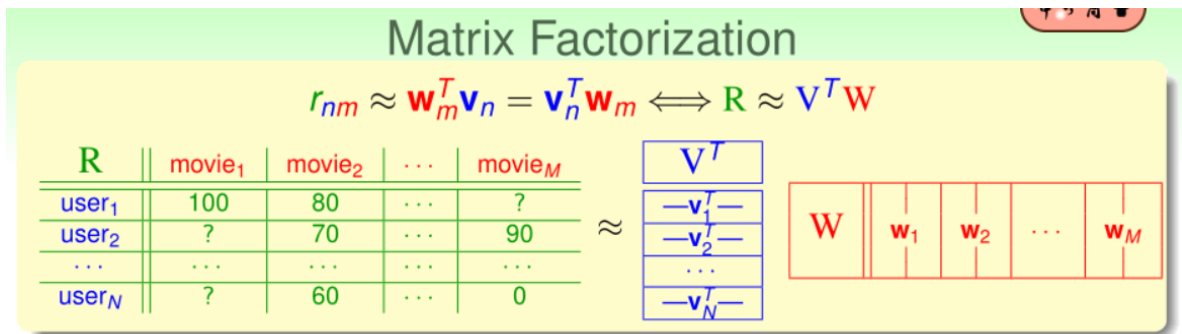
### 矩阵分解(Matrix Factorization)

$$r_{nm} \approx \mathbf{w}_m^T \mathbf{v}_n = \mathbf{v}_n^T \mathbf{w}_m$$

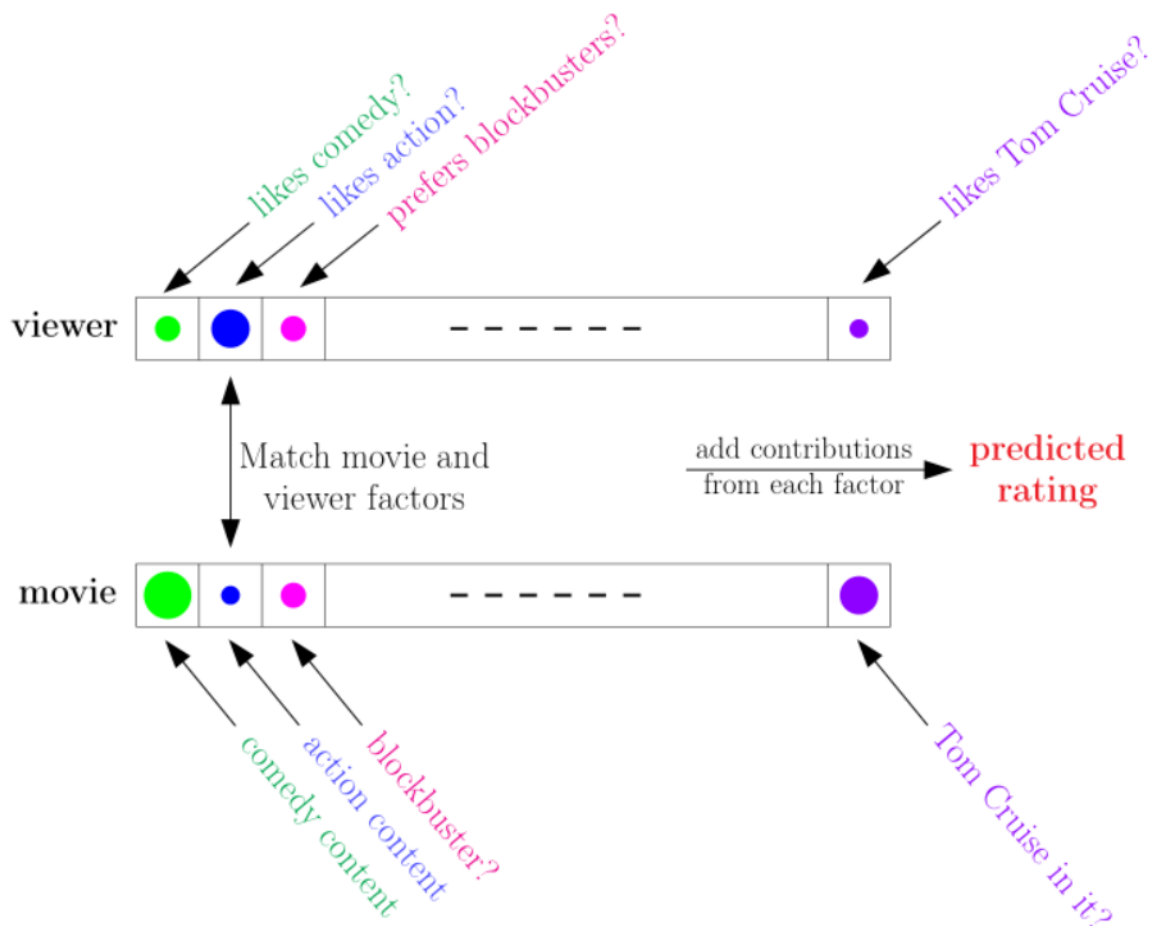
也就是：

$$\mathbf{R} \approx \mathbf{V}^T \mathbf{W}$$

如下图：



从一些已经知道的某些用户对某些电影的评分(known rating)出发，学习一些factors—— $\mathbf{v}_n$ 和 $\mathbf{w}_m$ ，最后能够用他们来做预测：



现在，需要解决的最优化问题是：

$$\min_{\mathbf{W}, \mathbf{V}} E_{in}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \sum_{m=1}^M \left( \sum_{(\mathbf{x}_n, r_{nm}) \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \right)$$

上述最优化问题有两组变量——alternating minimization：

- 当 $\mathbf{v}_n$ 固定，最小化 $\mathbf{w}_m$ ，等价于在电影 $m$ 的数据集 $\mathcal{D}_m$ 上最小化 $E_{in}$ ——per-movie linear regression without  $w_0$ ；
- 当 $\mathbf{w}_m$ 固定，最小化 $\mathbf{v}_n$ ，等价于在用户 $n$ 的数据集 $\mathcal{D}_n$ 上最小化 $E_{in}$ ——per-user linear regression without  $v_0$ ——对称的。

该算法被称为**alternating least squares**：

### Alternating Least Squares

- 1 initialize  $\tilde{d}$  dimension vectors  $\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$
  - 2 **alternating optimization** of  $E_{in}$ : repeatedly
    - 1 optimize  $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_M$ :  
update  $\mathbf{w}_m$  by  **$m$ -th-movie** linear regression on  $\{(\mathbf{v}_n, r_{nm})\}$
    - 2 optimize  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N$ :  
update  $\mathbf{v}_n$  by  **$n$ -th-user** linear regression on  $\{(\mathbf{w}_m, r_{nm})\}$
- until **converge**

与矩阵分解类似的是线性自编码器：前者常用于降维，后者常用于抽取隐藏的特征：

### Linear Autoencoder

$$\mathbf{X} \approx \mathbf{W} (\mathbf{W}^T \mathbf{X})$$

- motivation:  
**special  $d-\tilde{d}-d$**  linear NNet
- error measure:  
squared on **all**  $x_{ni}$
- solution: global optimal at  
**eigenvectors** of  $\mathbf{X}^T \mathbf{X}$
- usefulness: extract  
**dimension-reduced features**

### Matrix Factorization

$$\mathbf{R} \approx \mathbf{V}^T \mathbf{W}$$

- motivation:  
 **$N-\tilde{d}-M$**  linear NNet
- error measure:  
squared on **known**  $r_{nm}$
- solution: local optimal via  
**alternating least squares**
- usefulness: extract  
**hidden user/movie features**

## 3. Stochastic Gradient Descent: 随机梯度下降

除了使用ALS算法，我们还可以考虑使用SGD解决下面的最优化问题。

$$\min_{\mathbf{W}, \mathbf{V}} E_{in}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \sum_{m=1}^M \left( \sum_{(\mathbf{x}_n, r_{nm}) \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \right)$$

SGD：随机选择一个样本，然后用该样本错误衡量err的梯度来更新参数，迭代至收敛。优点是：

- 有效率；
- 易执行；
- 可以轻易地扩展到其他的err。

对某一笔资料的错误衡量：

$$err(user\ n, movie\ m, rating\ r_{nm}) = (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2$$

对所有的变数做偏微分，求得梯度，然后更新：变数是 $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_M$

- $\nabla_{\mathbf{v}_n} err = -2(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n) \mathbf{w}_m$
- $\nabla_{\mathbf{w}_m} err = -2(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n) \mathbf{v}_n$
- 其他的向量不用更新，因为梯度是0。

又可以把要更新的两个向量的梯度写成：

$$-(residual)(the\ other\ feature\ vector)$$

综上，矩阵分解SGD算法为：

### SGD for Matrix Factorization

initialize  $\tilde{d}$  dimension vectors  $\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$  **randomly**

for  $t = 0, 1, \dots, T$

- 1 randomly pick  $(n, m)$  within all known  $r_{nm}$
- 2 calculate residual  $\tilde{r}_{nm} = (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)$
- 3 SGD-update:

$$\begin{aligned}\mathbf{v}_n^{new} &\leftarrow \mathbf{v}_n^{old} + \eta \cdot \tilde{r}_{nm} \mathbf{w}_m^{old} \\ \mathbf{w}_m^{new} &\leftarrow \mathbf{w}_m^{old} + \eta \cdot \tilde{r}_{nm} \mathbf{v}_n^{old}\end{aligned}$$

适用于大数据。

## 4. Summary of Extraction Models: 萃取模型的总结

Extraction Models：能够自动萃取特征的模型，结构一般是：先进行特征转换，最后进行一步线性模型的操作。也就是说，这样的模型将特征转换纳入到学习的范围之内，自动进行特征萃取。

## 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  $\mathbf{v}_n$ ;  
movie features  $\mathbf{w}_m$

### $k$ Nearest Neighbor

$\mathbf{x}_n$ -neighbor RBF;  
weights  $y_n$

extraction models: a **rich** family

在训练Extraction Models时用到的一些技巧:

## 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**

## 5. Summary

- 推荐系统具有两类对象。第一，用户；第二，电影(推荐商品)。因此，我们可以构建这样一个矩阵，该矩阵每一行表示某一个用户对于所有电影的评分情况，每一列表示某一部电影被所有用户的评分情况。我们拿到的训练数据集是该矩阵的一部分。我们需要我们的推荐系统能够“填空”，即根据训练数据预测某用户对于一部没有看过的电影的评分。
- 在这种情形中，我们的特征只有一个，就是用户id，或者说是用户的index。该特征为分类变量，而对于分类变量我们往往将其编码为独热变量，这种编码叫做binary vector encoding。

- 受到神经网络自动学习特征的启发，我们可以借助神经网络进行特征萃取，以获取更有意义、更加具体的特征。我们使用的神经网络结构是  $N - \tilde{d} - M$ 。神经网络的输入是某用户index的独热编码向量，输出是该用户对于所有电影的评分。
- 神经网络第一层的权重矩阵记作  $V^T$ ，第二层的权重矩阵记作  $W$ 。因此有hypothesis：  
 $h(\mathbf{x}) = W^T V \mathbf{x}$ 。因此，用户n对于电影m的评分可以写作  $\mathbf{w}_m^T \mathbf{v}_n$ ，也就是  $\mathbf{v}_n^T \mathbf{w}_m$ 。我们希望，在训练数据上有  $r_{nm} \approx \mathbf{v}_n^T \mathbf{w}_m$ 。这也可以看做是，将"大矩阵"R分解为两个矩阵  $V^T$  与  $W$  的乘积，因此叫做矩阵分解。对应的最优化问题如下：

$$\min_{\mathbf{W}, \mathbf{V}} E_{in}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \sum_{m=1}^M \left( \sum_{(\mathbf{x}_n, r_{nm}) \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \right)$$

- 解决上述最优化问题有两种方法：
  - ALS：交替最小二乘法；
  - SGD：随机梯度下降。