

Matrix Factorization

一、Linear Network Hypothesis

1. 回顾一个基石里面的实际应用：Netflix的比赛，通过每个用户给许多电影的打分做一个推荐系统

Recommender System Revisited



- **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 \mathcal{D}_m for m -th movie:
 $\{(\tilde{\mathbf{x}}_n = (n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m\}$
—abstract feature $\tilde{\mathbf{x}}_n = (n)$

how to **learn our preferences** from data?

这里我们的输入就简单的是类别编号 n ——啥意思没有。。。我们回顾以前学习的模型，这些模型往往有一个特点——就是它们都青睐数值型的数据（好像决策树系列除外）那么我们有没有一个好方法来对付这类categorical features（离散类别名称）呢？接下来介绍binary vector encoding。

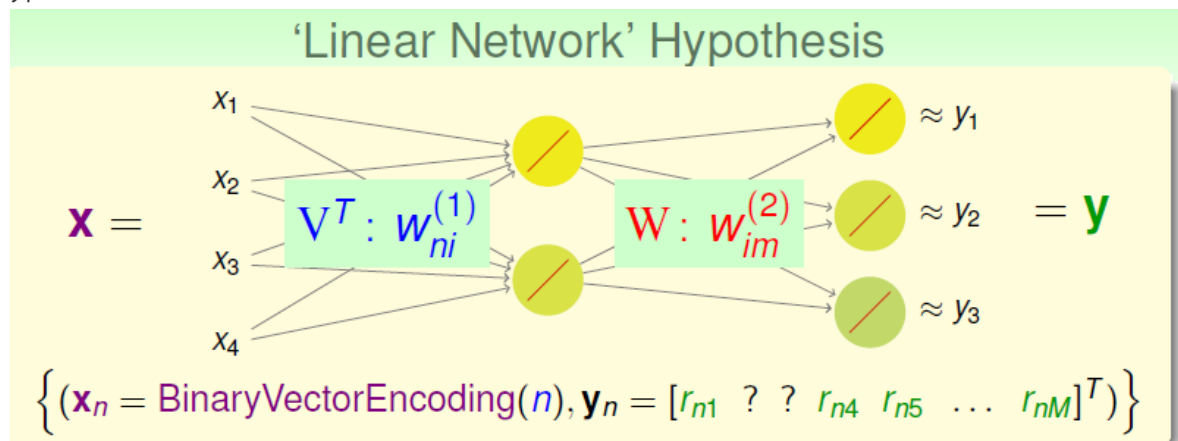
binary vector encoding:

$$\begin{aligned} \mathbf{A} &= [1 \ 0 \ 0 \ 0]^T, \mathbf{B} = [0 \ 1 \ 0 \ 0]^T, \\ \mathbf{AB} &= [0 \ 0 \ 1 \ 0]^T, \mathbf{O} = [0 \ 0 \ 0 \ 1]^T \end{aligned}$$

我们对于输入输出采取这样的操作：

$$\mathbf{x}_n = [0, 0, \dots, 1, \dots, 0]^T, \mathbf{y}_n = [r_{n1}, ?, ?, r_{n4}, \dots, r_{nM}]^T$$

确定好input output之后我们来用一个 $N - \tilde{d} - M$ NNet(为了简便, 放弃bias项)来学习这些特征。注意到输出传入进去的实际只有一项是有效的, 所以非线性在这里没有太大的必要——我们可以放弃non-linear transform:



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

linear network for recommender system:
learn V and W

我们定义两个矩阵来表示这个权重:

- $V^T: W_{ni}^{(1)}$
- $W: W_{im}^{(2)}$

我们作运算,

$$h(x_n) = W^T (V x_n) \quad (1)$$

注意到 $V x_n$ 实际上就是对于 V 的 column space 的线性组合, 那么 x_n 里面只有一项非零, 我们把相应的乘积 (V 中的第 n 列) 叫做 v_n 。那么:

$$h(x_n) = W^T v_n \quad (2)$$

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

linear network for recommender system:
learn V and W

那么我们需要做的事情无非两件事: 学习 V 和 W 。

二、Basic Matrix Factorization

1. 我们因为是在一个线性模型中，所以NN就是一个linear network，我们对于每个电影进行考虑。考虑 E_{in}

$$E_{in}(w_m, v_n) = \frac{1}{\sum_{m=1}^M |D_m|} \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2 \quad (3)$$

Linear Network: Linear Model Per Movie

linear network:

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

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

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

$$E_{in}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \frac{1}{\sum_{m=1}^M |D_m|} \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2$$

linear network: transform and linear models
jointly learned from all D_m

2. Matrix Factorization:

我们列一下电影打分表格：

Matrix Factorization

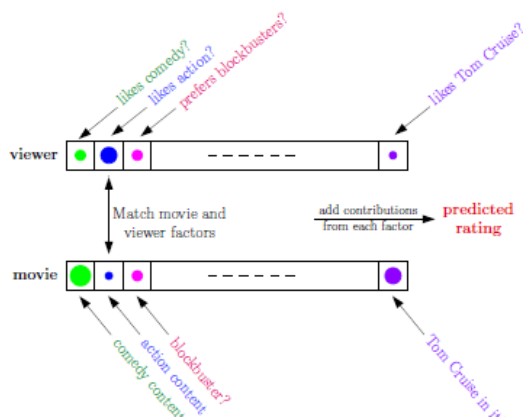
$$r_{nm} \approx \mathbf{w}_m^T \mathbf{v}_n = \mathbf{v}_n^T \mathbf{w}_m \iff \mathbf{R} \approx \mathbf{V}^T \mathbf{W}$$

\mathbf{R}	movie ₁	movie ₂	...	movie _M
user ₁	100	80	...	?
user ₂	?	70	...	90
...
user _N	?	60	...	0

 \approx

\mathbf{V}^T
$-\mathbf{v}_1^T-$
$-\mathbf{v}_2^T-$
...
$-\mathbf{v}_N^T-$

\mathbf{W}	\mathbf{w}_1	\mathbf{w}_2	...	\mathbf{w}_M
--------------	----------------	----------------	-----	----------------



Matrix Factorization Model

learning:

- known rating
- learned factors \mathbf{v}_n and \mathbf{w}_m
- unknown rating prediction

similar modeling can be used for other abstract features

我们预测 $r_{nm} \approx \mathbf{w}_m^T \mathbf{v}_n \iff \mathbf{R} \approx \mathbf{V}^T \mathbf{W}$ 本质就是matrix factorization

3. 看一下最佳化的问题：

Matrix Factorization Learning

$$\begin{aligned} \min_{\mathbf{W}, \mathbf{V}} E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) &\propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \\ &= \sum_{m=1}^M \left(\sum_{(\mathbf{x}_n, r_{nm}) \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \right) \end{aligned}$$

- **two sets** of variables:
can consider **alternating minimization, remember? :-)**
- when \mathbf{v}_n fixed, minimizing $\mathbf{w}_m \equiv$ minimize E_{in} within \mathcal{D}_m
—simply per-movie (per- \mathcal{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

和之前K-Means一样我们都是两组变量的优化问题，我们还是使用调整法：

- 固定V，我们对W（省略bias）进行linear regression
- 固定W，我们对V（省略bias）进行linear regression

实际上！两个矩阵是对称的 😊 这个最优化的策略和之前一样都是一个alternating的策略，特别的治理叫做 **alternating least squares algorithm**

我们总结一下算法的流程：

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**

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

alternating least squares:
the '**tango**' dance between users/movies

- 随机初始化V, W
- alternating optimization: 双重linear regression（矩阵求解）
- 直到收敛就停止（看看 E_{in} 变了没有）

4. 这个方法实际上让我们想起了之前在PCA里面的转化

Linear Autoencoder versus Matrix Factorization

Linear Autoencoder

$$X \approx W(W^T X)$$

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

Matrix Factorization

$$R \approx V^T W$$

- 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

linear autoencoder
≡ special matrix factorization of complete X

我们做一个对比：

	PCA	Matrix Factorization
网络结构	$d - \tilde{d} - d$ linear NNet	$N - \tilde{d} - M$ linear NNet
误差衡量	x_{ni}	r_{nm}
解决办法	$X^T X$ 的最大特征值	轮流均方误差优化
用武之地	降维特征提取	(电影) 特征提取

三、Stochastic Gradient Descent

1. 我们除了用这个矩阵方法解决最优化的问题，实际上我们还可以利用老朋友SGD，实际上这个更流行：

Another Possibility: Stochastic Gradient Descent

$$E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \underbrace{\left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n \right)^2}_{\text{err}(\text{user } n, \text{movie } m, \text{rating } r_{nm})}$$

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

2. 我们先求一下梯度:

Gradient of Per-Example Error Function

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

$$\begin{aligned} \nabla_{\mathbf{v}_{1126}} \quad & \text{err}(\text{user } n, \text{movie } m, \text{rating } r_{nm}) = \mathbf{0} \text{ unless } n = 1126 \\ \nabla_{\mathbf{w}_{6211}} \quad & \text{err}(\text{user } n, \text{movie } m, \text{rating } r_{nm}) = \mathbf{0} \text{ unless } m = 6211 \\ \nabla_{\mathbf{v}_n} \quad & \text{err}(\text{user } n, \text{movie } m, \text{rating } r_{nm}) = -2 \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n \right) \mathbf{w}_m \\ \nabla_{\mathbf{w}_m} \quad & \text{err}(\text{user } n, \text{movie } m, \text{rating } r_{nm}) = -2 \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n \right) \mathbf{v}_n \end{aligned}$$

per-example gradient

$$\propto -(\text{residual})(\text{the other feature vector})$$

求导之后出来的结果是余数项（残差）和向量的内积

3. 总结一下流程：只不过我们这个地方一次优化双份。。。

SGD for Matrix Factorization

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:

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

SGD: perhaps **most popular** large-scale matrix factorization algorithm

4. 鲜活的例子——SGD应用：

对于电影评价的时候，我们往往最近（时间靠后）的电影打分应该来说评分是更重的，但是我们在权重上应该怎么体现呢？我们直到SGD是对选定的一个点进行梯度下降，所以这个点而言梯度下降的意义非凡。我们希望SGD选的点都是靠后时间的点！

SGD for Matrix Factorization in Practice

KDDCup 2011 Track 1: World Champion Solution by NTU

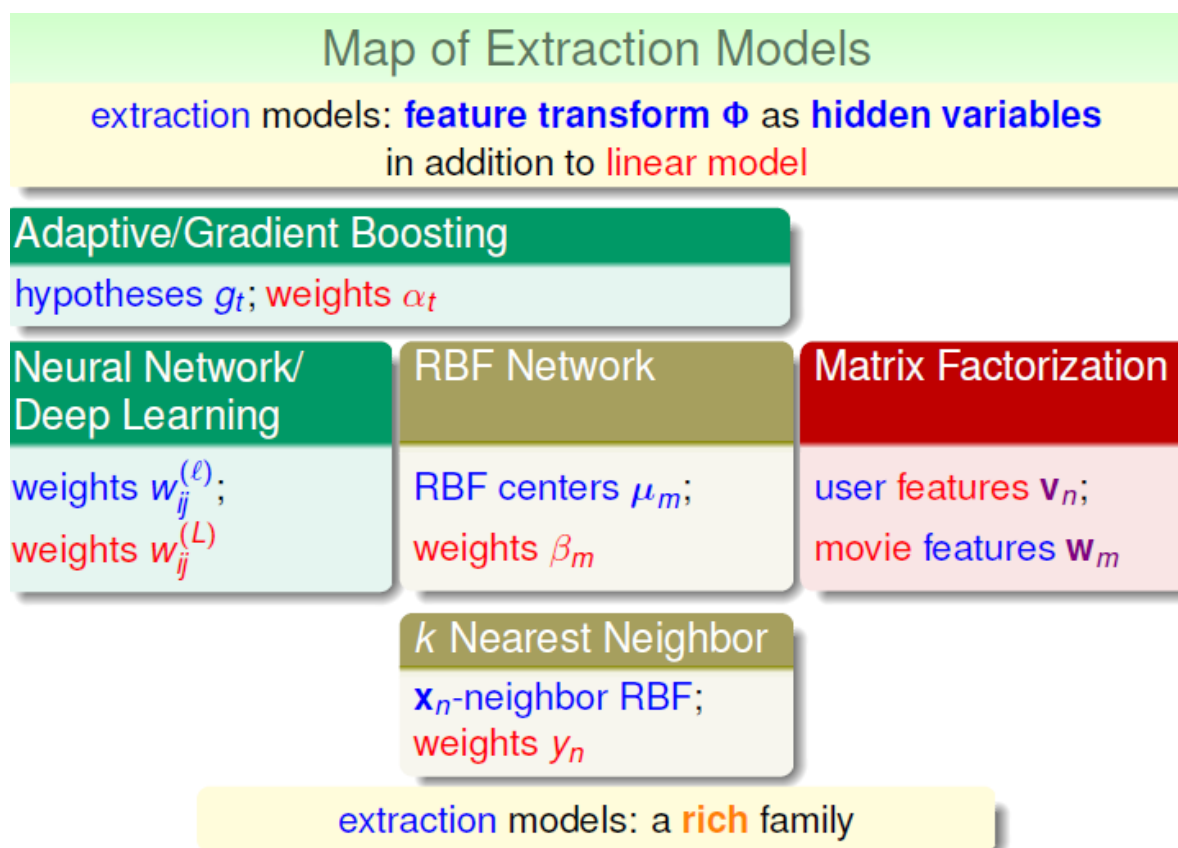
- specialty of data (application need):
per-user training ratings **earlier than** test ratings **in time**
 - training/test mismatch: typical **sampling bias, remember? :-)**
-
- want: **emphasize latter** examples
 - **last** T' iterations of SGD: **only those T' examples** considered
—learned $\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$ **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

妙啊 😊

四、Summary of Extraction Model

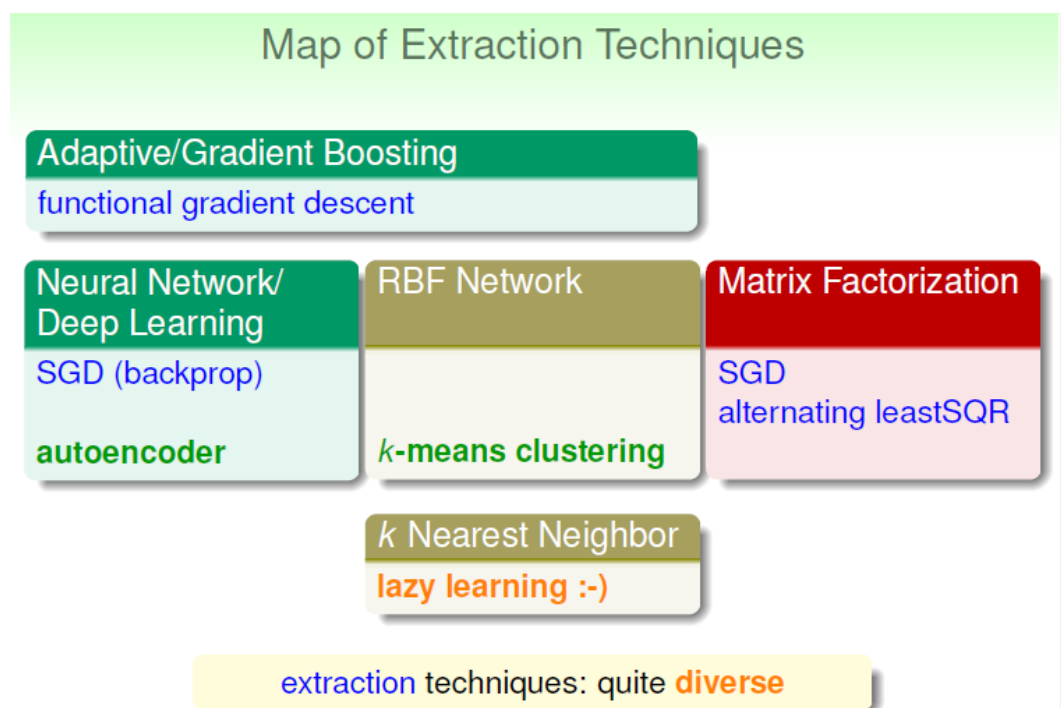
1. 总结一下extraction models!



神经网络、RBF网络、k近邻法、矩阵分解

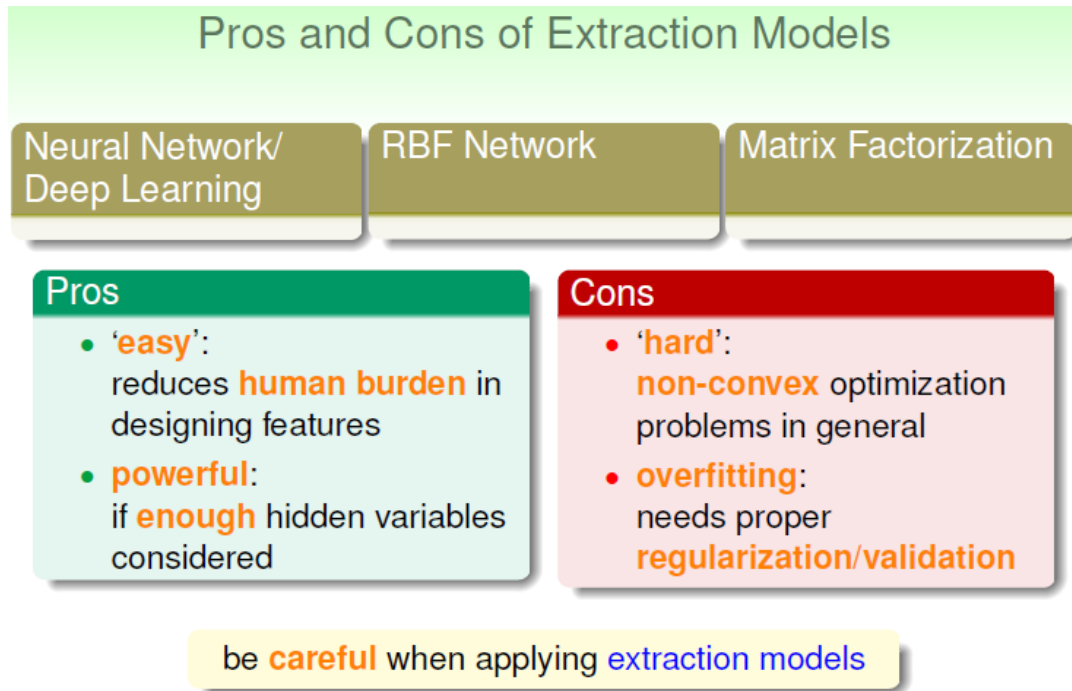
其实 boosting (AdaBoost、Gradient Boost) 的方法也可以看作是一种 extraction (因为实际上权重就是一种特征的提取转换操作!)

2. 同时我们在extraction models中蕴含着很多extraction techniques:



- GB: 函数梯度下降
- NN: SGD + autoencoder (PCA、denoising autoencoder)
- RBF Network: K-Means Clustering
- K-NN (lazy learning emoji 😊)
- Matrix Factorization: SGD + alternating least square optimization

3. extraction models的优缺点:



四字箴言：小心起见