# **Matrix Factorization**

# 1.LinearNetwork Hypothesis

机器学习的目的就是让机器从数据data中学习到某种能力skill。



- · data: how 'many users' have rated 'some movies'
- skill: predict how a user would rate an unrated movie
- 一个典型的电影推荐系统的例子是2006年Netflix举办的一次比赛。

该推荐系统模型中,用 $\check{x}_n=(n)$ 表示第n个用户,这是一个抽象的特征,常常使用数字编号来代替具体哪个用户。输出方面,使用 $y_m=r_{nm}$ 表示第n个用户对第m部电影的排名数值。

## 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  $\tilde{\mathbf{x}}_n = (n)$ 

 $reve{x}_n = (n)$ 是用户的ID。这类特征被称为类别特征(categorical features)。

常见的categorical features包括: IDs, blood type, programming languages等等。

而许多机器学习模型中使用的大部分都是数值特征 (numerical features) 。例如linear models, NNet模型等。

但决策树 (decision tree) 可以使用categorical features。

所以说,如果要建立一个类似推荐系统的机器学习模型,就要把用户ID这种categorical features转换为numerical features。这种特征转换其实就是训练模型之前一个编码(encoding)的过程。

- · categorical features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, . . .
- 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

一种最简单的encoding方式就是binary vector encoding。也就是说,如果输入样本有N个,就构造一个维度为N的向量。第n个样本对应向量上第n个元素为1,其它元素都是0。下图就是一个binary vector encoding的例子。

#### binary vector encoding:

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

经过encoding之后,输入 $x_n$ 是N维的binary vector,表示第n个用户。

输出 $y_n$ 是M维的向量,表示该用户对M部电影的排名数值大小。

注意,用户不一定对所有M部电影都作过评价,未评价的恰恰是我们要预测的(下图中问号?表示未评价的电影)。

http://127.0.0.1:51004/view/40

encoded data  $\mathcal{D}_m$  for m-th movie:

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

or, joint data  $\mathcal{D}$ 

$$\left\{ (\mathbf{x}_n = \mathsf{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1} ? ? r_{n4} r_{n5} \dots r_{nM}]^T) \right\}$$

总共有N个用户,M部电影。对于这样的数据,需要掌握每个用户对不同电影的喜爱程度及排名。

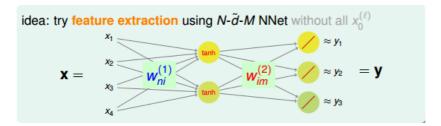
这其实就是一个特征提取(feature extraction)的过程,提取出每个用户喜爱的电影风格及每部电影属于哪种风格,从而建立这样的推荐系统模型。

可供选择使用的方法和模型很多,这里使用的是NNet模型。

NNet模型中的网络结构是 $N-reve{d}-M$ 型,其中N是输入层样本个数, $reve{d}$ 是隐藏层神经元个数,M是输出层电影个数。

该NNet为了简化计算,忽略了常数项。当然可以选择加上常数项,得到较复杂一些的模型。

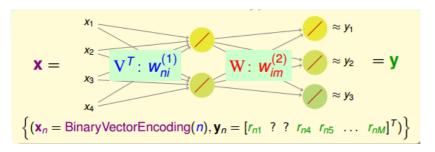
这个结构跟我们之前介绍的autoencoder非常类似,都是只有一个隐藏层。



NNet中隐藏层的tanh函数是否一定需要呢?答案是不需要。

因为输入向量x是经过encoding得到的,其中大部分元素为0,只有一个元素为1。那么,只有一个元素 $x_n$ 与相应权重的乘积进入到隐藏层。由于 $x_n=1$ ,则相当于只有一个权重值进入到tanh函数进行运算。

从效果上来说,tanh(x)与x是无差别的,只是单纯经过一个函数的计算,并不影响最终的结果,修改权重值即可得到同样的效果。 因此,我们把隐藏层的tanh函数替换成一个线性函数y=x,得到下图所示的结构。



由于中间隐藏层的转换函数是线性的,把这种结构称为Linear Network(与linear autoencoder比较相似)。

輸入层到隐藏层的权重 $W_{ni}^{(1)}$ 维度是Nx $\check{d}$ ,用向量 $V^T$ 表示。隐藏层到输出层的权重 $W_{im}^{(2)}$ 维度是 $\check{d}$ xM,用矩阵W表示。把权重由矩阵表示之后,Linear Network的hypothesis 可表示为:

$$h(x) = W^T V x$$

如果是单个用户 $x_n$ ,由于X向量中只有元素 $x_n$ 为1,其它均为0,则对应矩阵V只有第n列向量是有效的,其输出hypothesis为:

$$h(x_n) = W^T v_n$$

• rename: 
$$V^T$$
 for  $\left[w_{ni}^{(1)}\right]$  and  $W$  for  $\left[w_{im}^{(2)}\right]$ 

• hypothesis:  $h(x) = W^T V x$ 

• per-user output:  $\mathbf{h}(\mathbf{x}_n) = \mathbf{W}^T \mathbf{v}_n$ , where  $\mathbf{v}_n$  is *n*-th column of V

#### 2. Basic Matrix Factorization

Vx可以看作是对用户x的一种特征转换 $\Phi(x)$ 。对于单部电影,其预测的排名可表示为:

$$h_m(x) = w_m^T \Phi(x)$$

2/7

http://127.0.0.1:51004/view/40

linear network:

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

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

推导完linear network模型之后,对于每组样本数据(即第n个用户第m部电影),我们希望预测的排名 $w_n^Tv_n$ 与实际样本排名 $y_n$ 尽可能接近。

所有样本综合起来,使用squared error measure的方式来定义 $E_{in}$  ,  $E_{in}$ 的表达式如下所示:

- 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:

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

上式中,灰色的部分是常数,并不影响最小化求解,所以可以忽略。

接下来,就要求出 $E_{in}$ 最小化时对应的V和W解。

目标是让真实排名与预测排名尽可能一致,即 $r_{nm}\approx w_m^Tv_n=v_n^Tw_m$ 。把这种近似关系写成矩阵的形式: $R\approx V^TW$ 。矩阵R表示所有不同用户不同电影的排名情况,维度是NxM。这种用矩阵的方式进行处理的方法叫做Matrix Factorization。

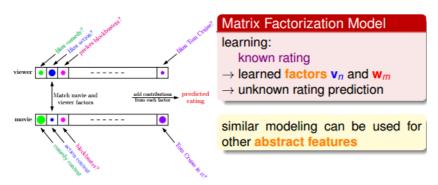
$r_{nm} \approx \mathbf{w}_m^T \mathbf{v}_n = \mathbf{v}_n^T \mathbf{w}_m \Longleftrightarrow \mathbf{R} \approx \mathbf{V}^T \mathbf{W}$											
R	movie <sub>1</sub>	movie <sub>2</sub>		movie <sub>M</sub>		$V^T$					
user <sub>1</sub>	100	80		?		_v <sub>1</sub> '	***				
user <sub>2</sub>	?	70		90	$\approx$	- <b>v</b> <sub>2</sub> -	W	W <sub>1</sub>	W <sub>2</sub>		w <sub>M</sub>
user <sub>N</sub>	?	60		0		_v,_					

希望将实际排名情况R分解成两个矩阵 (V和W) 的乘积形式。

V的维度是 $\check{d}$  xN的,N是用户个数, $\check{d}$  可以是影片类型,例如(喜剧片,爱情片,悬疑片,动作片,…)。

根据用户喜欢的类型不同,赋予不同的权重。W的维度是 $\check{d}$  xM,M是电影数目, $\check{d}$  同样是影片类型,该部电影属于哪一类型就在那个类型上占比较大的权重。

 ${d}$ 维特征不一定就是影片类型,还可以是其它特征,例如明显阵容、年代等等。



那么,Matrix Factorization的目标就是最小化 $E_{in}$ 函数。 $E_{in}$ 表达式如下所示:

$$\min_{\mathbf{W}, \mathbf{V}} E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left(r_{nm} - \mathbf{w}_m^\mathsf{T} \mathbf{v}_n\right)^2$$

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

 $E_{in}$ 中包含了两组待优化的参数,分别是 $v_n$ 和 $w_m$ 。

可以借鉴k-Means的做法,将其中第一个参数固定,优化第二个参数,然后再固定第二个参数,优化第一个参数,一步一步进行优化。

当 $v_n$ 固定的时候,只需要对每部电影做linear regression即可,优化得到每部电影的d维特征值 $w_m$ 。

http://127.0.0.1:51004/view/40

3/7

当 $w_m$ 固定的时候,因为V和W结构上是对称的,同样只需要对每个用户做linear regression即可,优化得到每个用户对 $\check{d}$ 维电影特征的喜爱程度 $v_n$ 。把这种近似关系写成矩阵的形式:

two sets of variables:
 can consider alternating minimization, remember? :-)
 when v<sub>n</sub> fixed, minimizing w<sub>m</sub> = minimize E<sub>in</sub> within D<sub>m</sub>
 —simply per-movie (per-D<sub>m</sub>) linear regression without w<sub>0</sub>
 when w<sub>m</sub> fixed, minimizing v<sub>n</sub>?
 —per-user linear regression without v<sub>0</sub>
 by symmetry between users/movies

这种算法叫做alternating least squares algorithm。它的处理思想与k-Means算法相同,其算法流程图如下所示:

# Alternating Least Squares initialize ã dimension vectors {w<sub>m</sub>}, {v<sub>n</sub>} alternating optimization of E<sub>in</sub>: repeatedly optimize w<sub>1</sub>, w<sub>2</sub>,..., w<sub>M</sub>: update w<sub>m</sub> by m-th-movie linear regression on {(v<sub>n</sub>, r<sub>nm</sub>)} optimize v<sub>1</sub>, v<sub>2</sub>,..., v<sub>N</sub>: update v<sub>n</sub> by n-th-user linear regression on {(w<sub>m</sub>, r<sub>nm</sub>)} until converge

alternating least squares algorithm有两点需要注意。

第一是initialize问题,通常会随机选取 $v_n$ 和 $w_m$ 。

第二是converge问题,由于每次迭代更新都能减小 $E_{in}$ , $E_{in}$ 会趋向于0,则保证了算法的收敛性。

- · initialize: usually just randomly
- · converge:

guaranteed as  $E_{in}$  decreases during alternating minimization

Matrix Factorization与Linear Autoencoder的比较。

# Linear Autoencoder

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

- motivation: special d-d-d linear NNet
- error measure: squared on all x<sub>ni</sub>
- solution: global optimal at eigenvectors of X<sup>T</sup>X
- usefulness: extract dimension-reduced features

#### Matrix Factorization

 $R \approx V^T W$ 

- motivation:
   N-d-M linear NNet
- error measure: squared on known r<sub>nm</sub>
- solution: local optimal via alternating least squares
- usefulness: extract hidden user/movie features

Matrix Factorization与Linear Autoencoder有很强的相似性,都可以从原始资料汇总提取有用的特征。 linear autoencoder可以看成是matrix factorization的一种特殊形式。

# 3. Stochastic Gradient Descent

使用Stochastic Gradient Descent方法来进行求解。

之前的alternating least squares algorithm中,考虑了所有用户、所有电影。

现在使用SGD,随机选取一笔资料,然后只在与这笔资料有关的error function上使用梯度下降算法。

使用SGD的好处是每次迭代只要处理一笔资料,效率很高;而且程序简单,容易实现;最后,很容易扩展到其它的error function来实现。

http://127.0.0.1:51004/view/40 4/7

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

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

对于每笔资料,它的error function可表示为:

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

上式中的err是squared error function,仅与第n个用户 $v_n$ ,第m部电影 $w_m$ 有关。 其对 $v_n$ 和 $w_m$ 的偏微分结果为:

$$abla v_n = -2(r_{nm} - w_m^T v_n)w_m$$

$$abla w_m = -2(r_{nm} - w_m^T v_n) v_n$$

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

 $abla v_n$ 和 $abla w_m$ 都由两项乘积构成。 (忽略常数因子2)。

第一项都是 $r_{nm}-w_{m}^{T}v_{n}$ ,即余数residual。

 $\nabla v_n$ 的第二项是 $w_m$ ,而 $\nabla w_m$ 的第二项是 $v_n$ 。

二者在结构上是对称的。

计算完任意一个样本点的SGD后,就可以构建Matrix Factorization的算法流程。

SGD for Matrix Factorization的算法流程如下所示:

### SGD for Matrix Factorization

initialize  $\tilde{d}$  dimension vectors  $\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$  randomly for t = 0, 1, ..., T

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

$$\mathbf{v}_{n}^{\text{new}} \leftarrow \mathbf{v}_{n}^{\text{old}} + \eta \cdot \tilde{r}_{nm} \mathbf{w}_{m}^{\text{old}}$$
 $\mathbf{w}_{m}^{\text{new}} \leftarrow \mathbf{w}_{m}^{\text{old}} + \eta \cdot \tilde{r}_{nm} \mathbf{v}_{n}^{\text{old}}$ 

在实际应用中,由于SGD算法简单高效,Matrix Factorization大多采用这种算法。

eg: 根据现在有的样本资料, 预测未来的趋势和结果。

这是一个与时间先后有关的预测模型。比如说一个用户三年前喜欢的电影可能现在就不喜欢了。

所以在使用SGD选取样本点的时候有一个技巧,就是最后T次迭代,尽量选择时间上靠后的样本放入到SGD算法中。

这样最后的模型受这些时间上靠后的样本点影响比较大,也相对来说比较准确,对未来的预测会比较准。

http://127.0.0.1:51004/view/40

#### KDDCup 2011 Track 1: World Champion Solution by NTU

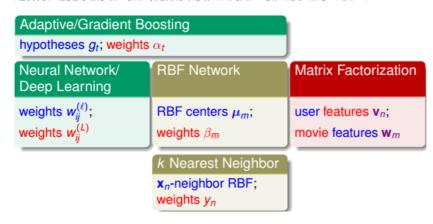
- specialty of data (application need):
   per-user training ratings earlier than test ratings in time
- training/test mismatch: typical sampling bias, remember? :-)
- · want: emphasize latter examples
- last T' iterations of SGD: only those T' examples considered
   —learned {w<sub>m</sub>}, {v<sub>n</sub>} favoring those
- our idea: time-deterministic XGD that visits latter examples last —consistent improvements of test performance

所以,在实际应用中,除了使用常规的机器学习算法外,还需要根据样本数据和问题的实际情况来修改算法,让模型更加切合实际,更加准确。

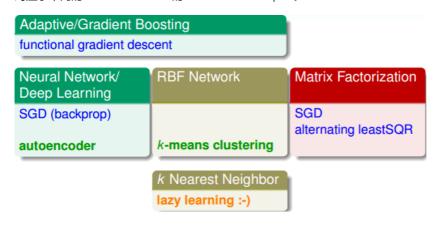
# **4.Summary of Extraction Models**

Extraction Models主要的功能就是特征提取和特征转换,将原始数据更好地用隐藏层的一些节点表征出来,最后使用线性模型将所有节点aggregation。

这种方法能够更清晰地抓住数据的本质,从而建立最佳的机器学习模型。



对应于不同的Extraction Models的Extraction Techniques。



最后,总结一下这些Extraction Models有什么样的优点和缺点。

从优点上来说:

easy: 机器自己完成特征提取,减少人类工作量 powerful: 能够处理非常复杂的问题和特征提取

另一方面,从缺点上来说:

hard:通常遇到non-convex的优化问题,求解较困难,容易得到局部最优解而非全局最优解

overfitting:模型复杂,容易造成过拟合,需要进行正则化处理

所以说,Extraction Models是一个非常强大的机器学习工具,但是使用的时候也要小心处理各种可能存在的问题。

http://127.0.0.1:51004/view/40 6/7

problems in general

regularization/validation

overfitting:

needs proper

Neural Network/
Deep Learning

RBF Network

Cons

• 'easy':
reduces human burden in

designing features

if enough hidden variables

• powerful:

considered

be careful when applying extraction models

http://127.0.0.1:51004/view/40 7/7