Factorization Machines

1、场景描述

眼睛、鼻子、嘴巴、肤色、身高、体重、性格，世界观、人生观、爱情观等等可以作为对一个人的**"描述"**，这些描述以下我们统称为***特征（Feature)***，那么我们可以认为"每个人都可以使用不同类型的特征进行描述"。将以上人的各种特征使用单一的数值表示时，我们可以将一个人简单的表示为一下形式：

$$[X_1, X_2, X_3, X_4 \cdots \quad X_n] \tag{1}$$

公式（1）中n表示人的特征数量一共有n种，其中X1 , X2 , X3 , X4分别表示眼睛、鼻子、嘴巴、肤色，等

在人与人相处过程中，我们经常会依据对方身上的各种**特征**来给出某一方面的评价，比如男女相亲时，第一眼会很肤浅的扫视对方的长相，然后在接下来的接触中再依据性格等其他因素会给对方有个**衡量**（以下衡量统称为 **得分**,**score**），如果将这种打分方式进行线性描述：

$$y = WX + b \tag{2}$$

其中b是一个数值（偏置）， W 就是每个人心中的一个描述权重，W是一个向量，形式为：$[W_1, W_2, W_3, W_4 \cdots \quad W_n]$, 下标n 的大小与公式（1）中一致，则公式（2）详细计算方式为：

$$y = W_1 * X_1 + W_2 * X_2 + ... + W_n * X_n + b \tag{3}$$

公式（3）稍稍专业一点表示：

$$y = \sum_{i=1}^{n} W_i X_i + b \tag{4}$$

公式（4）的向量表示：

$$y = W \cdot X^T + b \tag{5}$$

综上所述，人的各种特征可以使用**(1,n)**的一维向量表示，且每个人心中都会有一个权重向量W，权重与特征的乘积求和 可以作为给彼此打分的依据，y的数值越高，表示高感度得分越高，以上这种描述score方式称之为"一阶线性描述"

2、问题分析

一阶线性方式考虑了单个维度的特征，即人的" 眼睛、鼻子....性格，世界观、人生观 "等特征只使用了独立的权重进行计算，没有考虑：眼睛 与 鼻子、眼睛与世界观、....、 性格与世界观、...等特征之间的组合

特征两两之间的组合描述可称之为 "二阶描述"，特征之间的组合可能性有 $C_n^2 = \frac{n(n-1)}{2!}$,当然也可以有特征之间的三阶描述则有 $C_n^3 = \frac{n(n-1)(n-2)}{3!}$,以此类推还有 四阶，五阶等特征描述。

基于多阶的特征描述，我们是不是可以猜想一下，一阶描述与多阶描述的组合是不是会能更好的去刻画一个人，得到的效果会不会更好？

3、特征的二阶描述

在描述两两特征之间的二阶关系时，不得不联想到一个$n \times n$的权重矩阵W，其中n是特征的数量，即维度。W矩阵的具体形式如下：

$$W = \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & W_{2,3} & \dots & W_{2,n} \\ & & \ddots & & \vdots \\ W_{1,n-1} & \dots & & & W_{n-1,n} \\ W_{1,n} & \dots & & & W_{n,n} \end{bmatrix}_{n \times n} \tag{6}$$

其中数值$W_{i,j}$表示 一个人的特征 X_i 与特征 X_j之间的权重，则一个人的二阶特征数学描述如下：

$$y = \sum_{i=1}^{n} \sum_{j=i+1}^{n} W_{i,j} \times X_i \times X_j \tag{7}$$

公式（7）中j 的索引值是从 $i+1$ 开始，即一个人中不同特征无先后顺序，且同一特征的组合在这里无意义，所以最终矩阵W可以写成是一个下三角全为0的矩阵，最终使用的只是非0的部分，详情如下：

$$W = \begin{bmatrix} 0 & W_{1,2} & W_{1,3} & \dots & W_{1,n} \\ \vdots & 0 & W_{2,3} & \dots & W_{2,n} \\ & & \ddots & & \vdots \\ 0 & \dots & & & W_{n-1,n} \\ 0 & \dots & & & 0 \end{bmatrix}_{n \times n} \tag{8}$$

最终在描述一个人的时候，我们可以使用用户的一阶与 二阶特征，具体的数学公式如下：

$$y = b + \sum_{i=1}^{n} W_i^* X_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} W_{i,j} \times X_i \times X_j \tag{10}$$

其中 W^*是一阶特征权重矩阵

4、Factorization Machines

Factorization Machines翻译过来的大白话就是**“因子分解的机器”**，简称 **FM。**这里超前提一下，FM其实分解的就是二阶矩阵W，即公式（8）

真实场景中直接求解矩阵 W 是很难的，难度主要来自于：

（1）可以一个人的特征维度是超级大，即n可能非常大，可能达到百万、千万；
（2）用户的上百万维的特征中，会出现90%为零的情况，则矩阵W很难求解

```
特征中90%+为零的情况，可以了解下one-hot编码
```

在聊FM的因子分解之前，先简单聊一下矩阵的乘法，在公式（8）中，我们可以知道二阶权重矩阵W的维度是$n \times n$，但是如何通过其他方式进行得到矩阵W呢？请看矩阵的乘法，如下：

$$
W_{n,n} = \begin{bmatrix} 0 & W_{1,2} & W_{1,3} & \dots & W_{1,n} \\ \vdots & 0 & W_{2,3} & \dots & W_{2,n} \\ & & \ddots & & \vdots \\ 0 & \dots & & & W_{n-1,n} \\ 0 & \dots & & & 0 \end{bmatrix}_{n\times n} = [V_1, V_2 \cdots \quad X_n]^T \cdot [V_1, V_2 \cdots \quad X_n] \tag{11}
$$

将公式（11）简化表示，可以得到$W[n,n] = V^T[n,1] \cdot V[1,n]$

```
ps，w[n,n]是指个人的一个习惯表述，表示有矩阵/向量/张量 w，他的维度/shape 是 n x n
```

划重点了

如果再将公式（11）中的向量 V的第一维扩展一番也是成立的，则可以表示为：

$$
W[n,n] = V^T[n,k] \cdot V[k,n] \tag{12}
$$

由公式（12）可以进行一次比较，将特征的维度 n=100万，k=100，那么我们的参数量 $n \times k << n \times n$, 所以上述就是将 特大号二阶 矩阵 W 分成小号矩阵的乘法，因此我们要具体表述 W 中的某个值时，我们可以由 V计算得到，如下(红色标出）：

$$
\begin{bmatrix} \cdots & & & \cdots & \\ \vdots & \dots & {\color{red}W_{i,j}} & \dots & W_{i,n} \\ & & & & \vdots \\ & \dots & & & \\ & \dots & & & W_{n,n} \end{bmatrix}_{n\times n} = \begin{bmatrix} & & \cdots & & \\ \vdots & & & \cdots & \\ & & & & \vdots \\ {\color{red}V_{i,1}} & {\color{red}V_{i,1}} & {\color{red}\dots} & & {\color{red}V_{i,k}} \\ \vdots & & & & \vdots \end{bmatrix}_{n\times k} \times \begin{bmatrix} & \dots & {\color{red}V_{1,j}} & & \\ \vdots & & {\color{red}V_{2,j}} & & \\ & & & \vdots & \\ & & {\color{red}\vdots} & & \\ \vdots & & {\color{red}V_{k,j}} & & \vdots \end{bmatrix}_{k\times n} \tag{13}
$$

<u>下面主要来描述一下FM针对矩阵W的优化思路：</u>

首先先转变一下 矩阵 V的认知，即，使用FM 论文中的表述方式，同时在这里再次强调 V是 一个 $k \times n$ 的矩阵，n是“人”特特征维度，k是超参（k是正整数）

$$
W_{i,j} =< V_i, V_j >:= \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f} \tag{14}
$$

公式（14）中$W_{i,j}$与公式（13）中的表述一致，$< V_i, V_j >$ 是向量的点成，详情见公式（13）

最终FM中 二阶方程表达式如下：

$$
y = b + \sum_{i=1}^{n} w_i^* x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < V_i, V_j > x_i \times x_j \tag{15}
$$

再多些一步，将公式14、15揉在一起得：

$$
y = b + \sum_{i=1}^{n} w_i^* x_i + \sum_{f=1}^{k} \sum_{i=1}^{n} \sum_{j=i+1}^{n} v_{i,f} \times v_{j,f} \times x_i \times x_j \tag{16}
$$

> 逼逼叨，逼逼叨到现在一直是在说二阶矩阵W的分解过程，FM的最核心的部分还没有上来（不要急，马上来，上述都是铺垫）

OK，下面是最FM的核心部分，也是最简单直接的部分，上公式：

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle x_i x_j \\
=&\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle x_i x_j-\frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_i\rangle x_i x_i \\
=&\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{j,f}\, x_i x_j-\sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{i,f}\, x_i x_i\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)\left(\sum_{j=1}^{n} v_{j,f}\, x_j\right)-\sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2-\sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right)
\end{aligned}
$$

Ending

哈哈哈，是不是超级简单 !!!!（优化公式来自论文截图）

最后的最后都是落在了最后的这个公式优化中，公式（16）可以重新成

$$y=b+\sum_{i=1}^{n} w_i^* x_i+\frac{1}{2}\sum_{k}^{f=1}\left(\left(\sum_{i=1}^{n} v_{i,f}\times x_i\right)^2-\sum_{i=1}^{n} n_{i,f}^2\times x_i^2\right) \tag{17}$$

关于FM再叨一句，为啥要写成这样：线性计算快速，二阶矩阵特征更丰富，具体的自己look look paper哈，哈哈哈哈！！！！

5、上代码

作为一名程序员，一切的一切都要落到代码中，所以接下来的时间就是上代码的路子啦！！！！

```
def __init__(self, dims, k_dims):
    super(FM, self).__init__()
    # linear
    self.linear_bias = tf.get_variable(name="bias_linear", shape=[1], initializer=tf.zeros_initializer)
    self.linear_weight = tf.get_variable(name="weight_linear", initializer=tf.random.truncated_normal(shape=[dims]))
    # matrix factorization factors
    self.factor_V = tf.get_variable(name="factor_V", initializer=tf.random.truncated_normal(shape=[k_dims, dims]))

def call(self, inputs, **kwargs):
    # 线性特征计算
    # bias + W0 * X
    linear_feature = tf.add(
        self.linear_bias, tf.reduce_sum(
            tf.multiply(self.linear_weight, inputs), axis=1, keep_dims=True
        )
    )
    # 二阶特征计算
    interaction_feature = tf.multiply(0.5, tf.reduce_sum(
        tf.subtract(
            tf.pow(tf.matmul(inputs, tf.transpose(self.factor_V)), 2),
            tf.matmul(tf.pow(inputs, 2), tf.pow(tf.transpose(self.factor_V), 2))),
        1, keepdims=True))
    inference = tf.add(linear_feature, interaction_feature)
    # 均方误差
    loss = tf.reduce_mean(tf.square(tf.subtract(kwargs["Y"], inference)))
```

耶耶耶，代码可以先look、look，具体解释就到下次了。今天（20200829）起了个大早！！！！