FM (因子分解机) [2010]

首先，为了防止歧义和自己犯迷糊，这里先对“特征(feature)”和“域(field)”进行定义。推荐中的categorical 特征是高度稀疏的，这里的“特征”指的不是诸如性别、地区之类的特征，而是经过one-hot之后，每个维度是一个特征。例如“性别 == 男”，“性别 == 女”是两个特征，“地区 == 北京”，“地区 == 上海”，“地区 == 重庆”，这算三个特征。更极端一点的，我们工业界常用的是ID类特征，例如京东活跃用户有一亿个，那么就是一亿个特征，所以推荐中的特征是高度稀疏的。

> The data is mostly categorical and contains multiple fields; a typical representation is to transform it into a high-dimensional sparse binary feature representation via one-hot encoding.

FM就是针对这种极度稀疏的特征提出的方法，其核心在于把每个特征用稠密embedding来表示。

1. 手动特征交叉+LR

FM之前的模型都没有**自动**进行**特征交叉**，而一般依赖**人的经验**来手动构造交叉特征，比如LR。这样的缺点十分明显：

- 人工构造，效率低下，且依赖于人的经验
- 特征交叉难以穷尽
- 由于**数据稀疏**，对于训练集中没有出现的交叉特征，模型就无法学习。例如，我们来显式的构造二阶交叉特征+LR，其公式如下：

$$w_0+\sum_{i=1}^{n} w_i x_i+\sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j$$

例如x_i表示特征“性别 == 男”，x_j表示“城市 == 重庆”。

这样有两个缺点：

一是，要学习的二阶参数 wij 太多了，假如有n个特征，就有O(n^2)种特征交叉。而实际应用中，n会非常大，例如ID类特征很容易达到100万维。

二是，这些权重之间没有任何联系(泛化能力差)。对于那些从来没有一起不为0的特征x_i, x_j, 其对应的权重w_ij就没有在训练中得到更新。假如推荐一个火锅，先遇到一个样本是 “性别 == 男”x“城市 == 重庆” ，标签为点击，那么“性别 == 男"x“城市 == 重庆”这个二阶特征的权重w_ij得到了更新，下次再遇到一个“性别==女”x“城市 == 重庆”的样本，但是 “性别 == 女”x“城市 == 重庆”的二阶特征的权重却没有在梯度下降中得到更新，因为在训练集中这两个特征从来没有一起出来过。可是从人的理解来说，吃不吃辣其实重庆这个特征占了很大的权重，难道我就不可以猜女x重庆也应该有一个较大的起始值才对嘛？

树模型由于推荐场景很多特征是稀疏的效果不是很好；而逻辑回归只考虑了一阶特征。所以，我们引入了FM。

2. FM思想

【关键词】稀疏数据、泛化能力

FM把特征的交叉做了一步**分解**，使用了隐含的embedding，所以叫做“因子分解机”。先来看公式：

$$\hat{y}(x) = \mathbf{w}_0 + \sum_{i=1}^{d} \mathbf{w}_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

其中，y是预测的label，例如"click","non-click".这里的第二项就是用LR去学习一阶特征的参数，第三项是二阶交叉特征，其中v_i, v_j 是特征 x_i, x_j 对应的embedding。

$V \in R^{d \times k}$是d个特征的embedding table，embedding维度为k。$\langle v_i, v_j \rangle$直接做向量内积来做交叉特征的系数（一个数），最后得到$n(n-1)/2$个二阶交叉特征的值，将它们相加就得到了最后一项 $\sum_{i=1}^{d} \sum_{j=i+1}^{d} < v_i, v_j > x_i x_j$,这是一个数。最后，将一阶特征输入到LR中，再加上二阶交叉项，得到最后的输出logit。

1）这样的分解，让模型有了**泛化能力**，对于新的特征组合，也能够进行比较好的推断。这是因为虽然x_i, x_j 没有同时不为零的出现过，但是他们毕竟和别的特征同时出现过，于是它们对应的隐向量 v_i, v_j 必定已经更新过，在这里直接求二者的点积就可以了。

> 1.如果说LR是复读机，那么FM可以算作是电子词典了。 2.**泛化**就是我没见过你，我也能懂你，但是泛化有时候和个性化有点矛盾，属于此消彼长的关系 3.实践中的泛化往往来源于**拆解**，没见过组成的产品，但是见过各种**零件**，就能推断出很多的信息

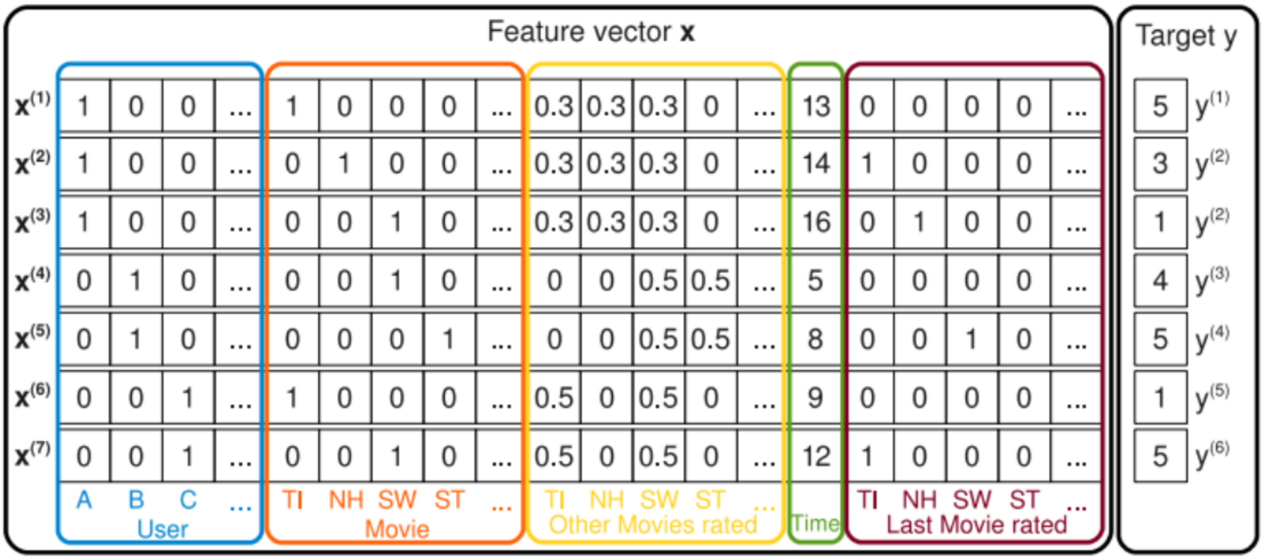
FM初步具备了**泛化**能力，对于**新的特征组合**有很好的推断性质，它所需要的可学习参数也小于交叉特征很多的LR。在这个DNN的时代，FM的交叉性质也没有被完全替代，还能站在时代的浪尖上。所以说，"在DNN时代，LR打不过也加入不了；FM打不过，但是它可以加入:)"

2）Factorization Machine也能够很好的解决**数据稀疏**问题，这也是FM比SVM的优势（由于维度太高，SVM会过拟合）。

$x \in R^d$ 表示d个特征，这些特征一般都是**极度稀疏的**--例如一些categorical feature，比如商品品类，可能有1000多维特征，每个维度都是0/1，而一个商品最多不过属于两三个品类，所以只有两三个特征是1，其他全都是0。当然，特征也可以是numerical feature，这样就是一个实数，FM也支持此类特征的embedding分解。 FM的优势就在于，对于如此稀疏的特征，它也不会像SVM那样过拟合（高维空间是线性可分的）。

> We deal with problems where x is highly sparse, i.e. almost all of the elements x_i of a vector x are zero.

例如，我们根据用户之前对电影的打分来预测他对现在这部电影的打分，在实际应用中，这种ID类特征都是非常稀疏的：

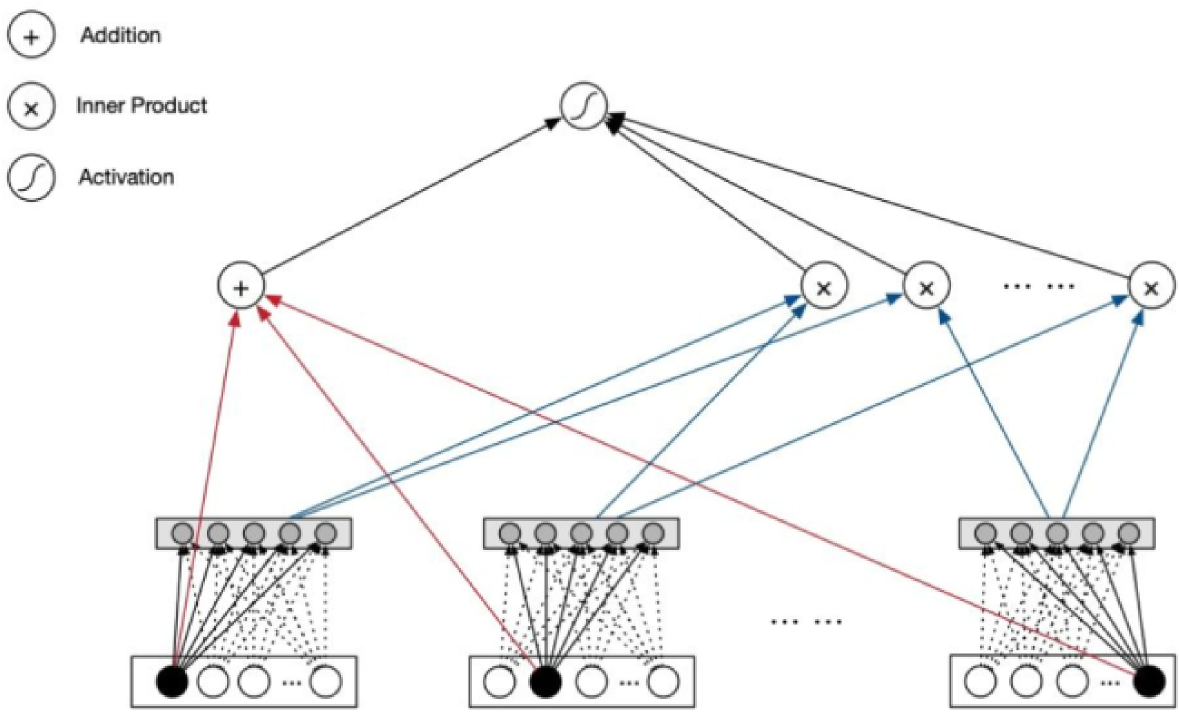


Feature vector **x**

	User				Movie					Other Movies rated					Time	Last Movie rated					Target y	
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...		
$x^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$x^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$x^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$x^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$x^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$x^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$x^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$

FM通过给每个特征做embedding，把它们都变成稠密表示。

整个模型的示意图如下所示：



红色代表一阶特征+LR；蓝色表示二阶特征。

3. FM复杂度

FM有一个复杂度问题，也是这篇文章的一个卖点，经常出现在面试中。

然而，如果只用上述这种暴力方法来计算二阶特征交互，其复杂度为 $O(kd^2)$,其中 d为特征的个数，因此是平方复杂度。当特征很多的时候，这样是不可取的。可以用下面的方法降为 $O(kd)$ 。其中d为特征的个数，k为特征embedding维度，因此FM可以为线性复杂度。

$$
\begin{aligned}
&\sum_{i=1}^{d}\sum_{j=i+1}^{d}\langle\mathbf{v}_i,\mathbf{v}_j\rangle x_i x_j \\
&=\frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\langle\mathbf{v}_i,\mathbf{v}_j\rangle x_i x_j-\frac{1}{2}\sum_{i=1}^{d}\langle\mathbf{v}_i,\mathbf{v}_i\rangle x_i x_i \\
&=\frac{1}{2}\Big(\sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{l=1}^{k}\mathbf{v}_{i,l}\mathbf{v}_{j,l}x_i x_j-\sum_{i=1}^{d}\sum_{l=1}^{k}\mathbf{v}_{i,l}\mathbf{v}_{i,l}x_i x_i\Big) \\
&=\frac{1}{2}\sum_{l=1}^{k}\Big((\sum_{i=1}^{d}\mathbf{v}_{i,l}x_i)(\sum_{j=1}^{d}\mathbf{v}_{j,l}x_j)-\sum_{i=1}^{d}\mathbf{v}_{i,l}^2x_i^2\Big) \\
&=\frac{1}{2}\sum_{l=1}^{k}\Big((\sum_{i=1}^{d}\mathbf{v}_{i,l}x_i)^2-\sum_{i=1}^{d}\mathbf{v}_{i,l}^2x_i^2\Big)
\end{aligned}
$$

假如有两个field，它们的特征取值个数分别为n1,n2，那么按照原来的方法就是要求(n1+n2)^2个参数；用FM进行embedding的方法，只需求 (n1+n2)d 个参数。

4. 总结

优点： 对训练集中未出现的交叉特征也可以进行泛化（embedding table）。

缺点： 只考虑了二阶交叉特征，没有考虑**更高阶**的交叉特征。

5. 代码实现

来源：Deepctr项目

二阶交叉项:

```
concated_embeds_value = inputs  ###输入的特征[batchsize, 26(feat_num), 4(embed_size)]

square_of_sum = tf.square(reduce_sum(concated_embeds_value, axis=1, keep_dims=True))#(?, 1, 4)
sum_of_square = reduce_sum(concated_embeds_value * concated_embeds_value, axis=1, keep_dims=True)##(?, 1, 4)
cross_term = square_of_sum - sum_of_square##(?, 1, 4)
cross_term = 0.5 * reduce_sum(cross_term, axis=2, keep_dims=False)##(?, 1)
```

一阶交叉项和常数项直接交给LR去做即可。