推荐系统入门系列(一)-FM算法理论与实践

何无涯 何无涯的技术小屋 5月7日

> 千里之行，始于足下！
>
> —— 老子

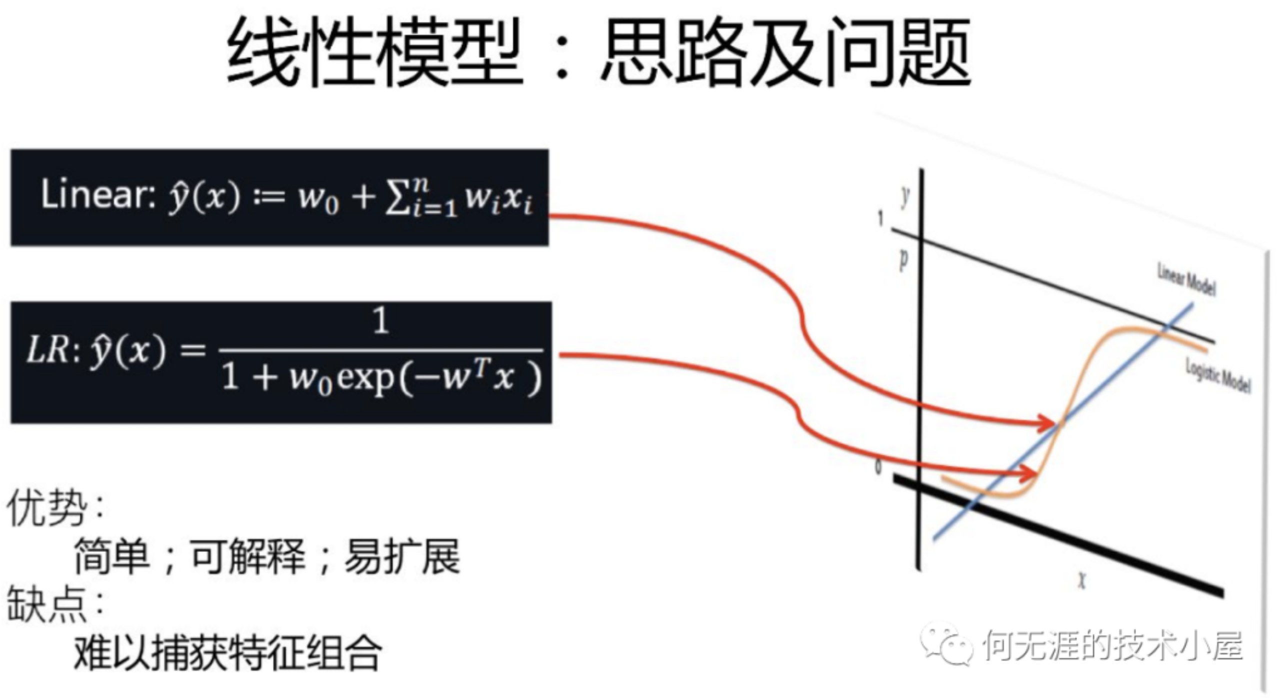
一、FM算法背景

FM算法（Factor Machine），又叫因子分解机算法。在推荐系统中，有一个非常重要的环节叫点击率预估（CTR，click-through rate），即根据所提供的一些特征信息来判断用户是否会有点击行为，这其中FM算法就有很广泛的应用。

二、FM算法思想

如果直接利用所提供的特征信息，**线性模型**将是最简单直接的方法。如下图所示，xi就是某个特征值，线性模型需要为每一个特征值学习一个权重wi，最终的模型预测值就是所有的特征值乘以这个权重，加起来求和。

如果是**逻辑回归**（LR）的话，会在上面求和的基础上套一个sigmoid函数，也就是图中的黄色曲线。直观上很好理解，使用sigmoid将线性模型的取值范围压缩到0-1，这样就可以很容易的判断是正面的结果还是负面的结果（对应着点击与不点击）。



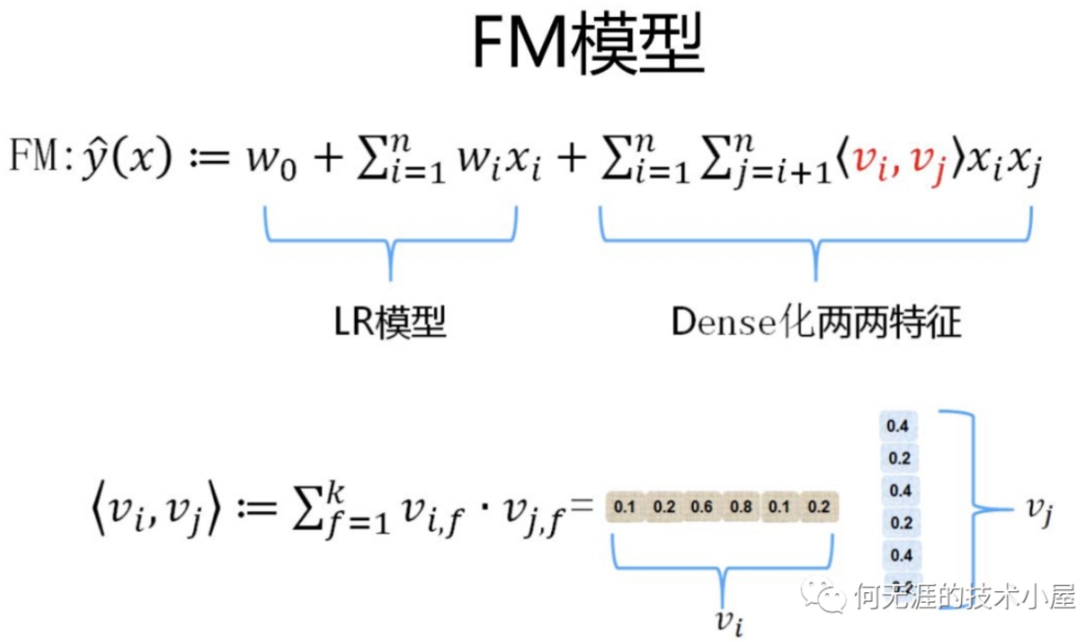
然而线性模型的缺点也很明显，**线性模型**没有捕获到任何的特征组合信息，然而这些特征组合信息对于CTR又非常重要。

那么很直观的想法就是直接引入两两组合特征融入到模型中，在线性模型的基础上得到：

$$y(x) = \omega_0 + \sum_{i=1}^{n} \omega_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \omega_{ij} x_i x_j$$

然而这样有两个比较大的问题，第一，这样处理组合特征的参数个数为n(n-1)/2，如果n=1000（对原始数据连续特征离散化，并且one-hot编码，特征很容易达到此量级），那么参数个数将达到50w，参数量巨大；第二，由于对原始数据的one-hot处理，数据将特别稀疏，满足xi，xj都不等于0的个数很少，这样就很难去学习到参数wij，导致组合特征的泛化能力差。

这就催生了FM算法的出现，FM算法是怎么干的呢？FM算法在上式的基础上**将wij替换成了vivj的点积**，



vi和vj又是什么含义呢？vi的意思是：对于xi这个特征来说它会学到一个embedding向量，特征组合权重是通过两个单特征各自的embedding的内积呈现的，因为它内积就是个数值，可以代表它的权重，这就是FM算法。

三、FM算法求解

FM算法的本质是利用**近似矩阵分解**，将参数权重矩阵W分解成两个向量相乘，从而将参数从平方级别减少到线性级别。FM算法的式子其实是可以推导得到的，接下来让我们看看FM算法具体是怎么一步一步推导而来的。

首先看直接将两两组合特征融入到模型里的式子：

$$y(x) = \omega_0 + \sum_{i=1}^{n} \omega_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \omega_{ij} x_i x_j$$

接下来将所有的参数wij组合成一个矩阵，

$$\mathrm{y}(x)=\omega_0+\sum_{i=1}^{n}\omega_i\times x_i+\begin{pmatrix}\omega_{11},\omega_{12},\ldots,\omega_{1n}\\ \omega_{21},\omega_{22},\ldots,\omega_{2n}\\ \cdots\\ \omega_{n1},\omega_{n2},\ldots,\omega_{nn}\end{pmatrix}\cdot\begin{pmatrix}0,x_1x_2,\ldots,x_1x_n\\ x_2x_1,0,\ldots,x_2x_n\\ \cdots\\ x_nx_1,x_nx_2,\ldots,0\end{pmatrix}$$

很显然，参数矩阵W是实对称矩阵，而且至少是半正定矩阵，这里假设为正定矩阵。根据正定矩阵的性质，正定矩阵可以分解，得到

$$W=V\cdot V^T,V\in R^{n\times k}$$

代入后，模型如下：

$$\mathrm{y}(x)=\omega_0+\sum_{i=1}^{n}\omega_i\times x_i+\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\mathrm{v}_{i1},v_{i2},\cdots,v_{ik})\cdot\begin{pmatrix}v_{1j}\\ v_{2j}\\ \cdots\\ v_{kj}\end{pmatrix}\cdot x_i\cdot x_j$$

即得到以下式子：

$$y(x)=w_0+\sum_{i=1}^{n}w_ix_i+\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle v_i,v_j\rangle x_ix_j$$

其中

$$\langle v_i,v_j\rangle=\sum_{f=1}^{k}v_{i,f}\cdot v_{j,f}$$

FM算法最核心的就是后面这个交叉项，为了计算方便，对该交叉项进行化简得到：

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

这样化简之后，计算就会简单多了，更具体的可以看接下来的算法实现。

四、FM算法实战

使用PyTorch深度学习框架实现FM算法，对于FM算法有两种实现：

1. 单独使用fm时的实现，这种写法中权重是单独的，作为需要训练的参数

```
class FM(nn.Module):
    """Pure Factorization Machine."""
    def __init__(self, n, k):
        super(FM, self).__init__()
        self.n = n  # num_fileds (all items and features)
        self.k = k  # feature emb dim
        self.linear = nn.Linear(self.n, 1, bias=True)
        self.v = nn.Parameter(torch.randn(self.k, self.n))

    def forward(self, x):
        # x [batch, n]
        linear_part = self.linear(x)
        # matmul [batch, n] * [n, k]
        inter_part1 = torch.mm(x, self.v.t())
```

```
        square_of_sum = torch.sum(torch.pow(inter_part1, 2))

        # matmul [batch, n]^2 * [n, k]^2
        inter_part2 = torch.mm(torch.pow(x, 2), torch.pow(self.v, 2).t())
        sum_of_square = torch.sum(inter_part2)

        # out_size = [batch, 1]
        output = linear_part + 0.5 * (square_of_sum - sum_of_square)
        return torch.sigmoid(x.squeeze(1))
```

2. 是在深度神经网络中应用的fm，FM算法中的权重向量也体现在前面的embedding部分，写法也就简单了

```
class FactorizationMachine(nn.Module):
    """Factorization Machine Layer"""
    def __init__(self, reduce_sum=True):
        super().__init__()
        self.reduce_sum = reduce_sum

    def forward(self, x):
        """
        :param x: Float tensor of size ``(batch_size, num_fields, embed_dim)``
        """
        square_of_sum = torch.sum(x, dim=1) ** 2
        sum_of_square = torch.sum(x ** 2, dim=1)
        ix = square_of_sum - sum_of_square
        if self.reduce_sum:
            ix = torch.sum(ix, dim=1, keepdim=True)
        return 0.5 * ix
```

完整的代码可以参考我的github: https://github.com/yyHaker/RecommendationSystem。

小结：FM算法本质上就是为每一个特征学习到一个特征向量，组合特征的特征向量的内积为组合特征的权重。