计算广告CTR预估系列(三)--FFM理论与实践

原创 李宁宁 机器学习荐货情报局 2018-05-14

计算广告CTR预估系列(三)--FFM理论与实践

> 首先，非常抱歉，上一篇文章计算广告CTR预估系列(二)—DeepFM实践里面github的链接挂掉了。原因是文章中不能有外链。这里重新贴一下：https://github.com/gutouyu/ML_CIA另外，大家也可以在公众号中点击"资料获取"，获得更详细的介绍。

1. 发展

在CTR预估中，**Logistic Regression**应该是最早被应用而且应用最广泛的模型了。输入是one-hot之后的特征，输出是点击广告的概率。对于类别型特征，one-hot之后，每一个取值都变成了一维新的特征。线性模型有一个致命的缺点：对于每一个维度特征权重的学习是独立的，很难有效的学习到组合特征的权重。

为了解决这个问题，相继提出了改进模型**Poly2和FM**。Poly2又叫做degree-2 polynomial mappings，它对于每一对组合特征都会学习一个权重，从名字上应该能看出来，最高考虑2维组合特征；FM则是通过把特征组合分解成两个隐向量的内积来学习组合特征的权重，理论上可以提取任意高维组合特征，但是出于计算复杂度的考虑，实际往往也只是到2维组合特征。

关于FM，在CTR预估系列的第四篇文章中我们会给出详细介绍，敬请期待~

FFM全称是**Field-aware Factorization Machines**，是从PITF（pairwise interaction tensor factorization）改进而来的。PITF限制了特征维度为User、Item、Tag这三个维度，而且主要关注的问题是个性化标签推荐。FFM去掉了对于特征维度的限制，并且专注于CTR预估问题，更加泛化通用。这也是两者间仅有的区别。

2. 理论公式

CTR预估模型中，LR，FM，FFM的目标函数或者说是损失函数，都可以统一为：

$$\min_{\boldsymbol{w}} \quad \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{m} \log(1+\exp(-y_i \phi_{\mathrm{LM}}(\boldsymbol{w}, \boldsymbol{x}_i))).$$

包括两部分：正则项 + 经验损失。经验损失用的是**logistic loss**.

对于线性模型有：

$$\phi_{\mathrm{LM}}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}.$$

决策函数为：

$$\phi_{\mathrm{Poly2}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} w_{h(j_1,j_2)} x_{j_1} x_{j_2},$$

对于FM有：

$$\phi_{\mathrm{FM}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\boldsymbol{w}_{j_1} \cdot \boldsymbol{w}_{j_2}) x_{j_1} x_{j_2}.$$

对于FFM有：

$$\phi_{\mathrm{FFM}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\boldsymbol{w}_{j_1,f_2} \cdot \boldsymbol{w}_{j_2,f_1}) x_{j_1} x_{j_2},$$

其中，j1，j2表示特征维度，f1，f2分别表示j1，j2所属的field。

公式有点多，是不是有点乱？没关系，让我们来理一理！

首先，为了简单，上面公式中省略了常数项和一次项。
其次，我们知道**统计学习三要素：模型+策略+算法**，模型就是要学习的**条件概率分布或决策函数**。由决策函数表示的模型为非概率模型，由条件概率分布表示的模型为概率模型。我们可以认为上面的函数就是**决策函数**，针对不同的模型**线性模型、Poly2、FM、FFM，决策函数自然也不同。**。里面的x对应**一条样本**，x的下标表示该条样本中，不同维度特征的取值。
最后，我们的目标函数就是最小化：**正则项 + logistic_loss**。其中logistic_loss自然就和模型的决策函数有关了。

3. 思想

3.1 Poly2

对于Poly2：

1. 相比于kernel method，训练时间要短
2. 为每一个组合特征都学习一个权重。特征权重之间是**独立**的
3. 又叫做degree-2 polynomial mapping，多项式线性模型，主要是2维组合特征

决策函数为：

$$\phi_{\text{Poly2}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} w_{h(j_1,j_2)} x_{j_1} x_{j_2},$$

Poly2最naive的版本，认为每一对组合特征都是一个新特征。但是这样模型大小是O(n^2)，模型太大了无法学习。
Vowpal Wabbit (VW)是一个库，通过对j1和j2做**hash**，解决了这个问题。

换句话说，函数 `h(j1, j2)` 是一个**Hash函数**。作用是**将j1和j2，映射到一个合理的自然数上。**为什么会有这样奇怪的举动那？原因是：**CTR问题的输入维度一般比较高，Poly2考虑所有的2维组合特征，为每个组合特征都要学习一个权重，这个维度是O(n^2)。所以我们把原来的维度通过Hash，转换到一个合理的维度范围，再进行学习。**

VW和FFM中实现的Hash函数如下：

$$h(j_1, j_2) = (\frac{1}{2}(j_1 + j_2)(j_1 + j_2 + 1) + j_2) \bmod B,$$

3.2 FM

> 这里简单介绍下，详细介绍CTR系列后续文章会更新，敬请关注

FM的做法：

1. 为每一维特征都学习一个隐向量
2. 每一个隐向量都包含k个latent factors。k是人为设定的超参数
3. 每一个组合特征对于最终预测结果的影响，或者说权重，都通过对应的隐向量的内积来表示

FM的论文里有提到，FM效果比Poly2要好的原因，个人认为这也是FM的核心所在：最主要的原因可以说是**样本不够充分造成的**。 有人会说，CTR问题的训练集一般很大，样本怎么会不够用那？输入数据one-hot之后，维度会很高，而且非常稀疏，相比之下样本数量就没那么多了。

举个例子，假设这是我们的训练数据：

		Publisher	Advertiser
+80	−20	ESPN	Nike
+10	−90	ESPN	Gucci
+0	−1	ESPN	Adidas
+15	−85	Vogue	Nike
+90	−10	Vogue	Gucci
+10	−90	Vogue	Adidas
+85	−15	NBC	Nike
+0	−0	NBC	Gucci
+90	−10	NBC	Adidas

具体来说，原因有两点：

> 1， **样本不充分。** Poly2学习到的权重不准确。

在ESPN的广告中，Adidas的样本只有一个，预测结果是-1。那么Poly2对于这个组合特征会学到一个非常大的负的权重 $W_{ESPN,Adidas}$. 因为只有这一个样本，而且label是-1. 模型就认为只要这个pair出现，那么基本上预测结果就是-1， 所以对应的weights会是一个非常大的负值。
但是，FM不同，对于样本(ESPN,阿迪)的预测，是由两个向量决定的：W_espn * W_adidas 。 而这两个向量，可以单独的去从其他的pair中学习，比如(ESPN,Nike),

(BBC, Adidas). 所以FM这样学习的更加准确。
也就是说，样本的不充分会导致Poly2学习的权重不准确，而FM的会更加准确。

> 2，**样本中根本没有出现这个组合特征。** Poly2学习不到权重。

比如(NBC, Gucci)这一对样本取值，在给出的训练集上根本就没有。
Poly2会什么都学不到，或者认为weights是0，所以给出的预测也就没有意义。但是FM不同，可以单独的从其他的pair中学习到W_nbc, W_gucci 所以FM仍然可以给出有意义的预测。

这里面的关键在于：Poly2中对于每个**组合特征的权重是独立的**，比如$W_{NBC,Gucci}$和其他的任何权重都没有关系，他们是相互独立的，你无法从其他的权重中得到关于$W_{NBC,Gucci}$的任何信息。

FM中**组合特征的权重不再是独立的**，比如$W_{NBC,Gucci}$，它是由两部分组成的 V_NBC, V_Gucci。虽然(NBC, Gucci)没有这个样本。但是可以从包含NBC的其他样本中学习到V_NBC, 从包含Gucci的其他样本中学习到V_Gucci。那么就可以估计出(NBC, Gucci)的权重了。
这也更加符合逻辑。（NBC，Gucci）和（NBC，Adidas）都含有NBC，所有他们的权重应该是有一些联系，而不应该是完全独立的。

3.3 FFM

3.3.1 FFM模型—思想

FFM是从PITF改进来的，后者用于推荐系统中的个性化标签。

PITF起初是应用在推荐系统中的，开始只考虑三个维度的特征：User、Tag、Item。并且在三个不同的latent space学习了组合特征：（User，Tag），(User, Item), (Tag, Item). 后来又加上了更多的特征：AdId, UserId, QueryId等，并且也被应用于CTR。但是，PITF的目标毕竟还是推荐系统，而且限制特征维度为User Tag Item这三个维度。

所以，FFM应运而生！

CTR大部分的数据集的 特征都可以被分组到field中。**FFM就是在FM的基础上利用了这些分组的信息。**

举例来说，假设训练集只有一条训练样本：

Clicked	Publisher (P)	Advertiser (A)	Gender (G)
Yes	ESPN	Nike	Male

FM模型的决策函数是：

$$\boldsymbol{w}_{\text{ESPN}} \cdot \boldsymbol{w}_{\text{Nike}} + \boldsymbol{w}_{\text{ESPN}} \cdot \boldsymbol{w}_{\text{Male}} + \boldsymbol{w}_{\text{Nike}} \cdot \boldsymbol{w}_{\text{Male}}.$$

> 注意：这里不要写成 $W_P W_A W_G$。
> 这关乎到FM中一个重要的问题，**隐向量矩阵V到底表示的是什么？**
> V的一行表示一个特征，但是这个特征是one-hot之后的，上表中给出的是one-hot之前的。也就是说，one-hot之后，之前那些取值比如：ESPN Nike Male都变成了特征，**他们一个取值就是一个维度！！**惊不惊喜，意不意外！那么FM学习的就是每个取值（每个维度）的隐向量，所以这里要这么写。

在FM中，对于任意一个特征维度，比如ESPN，它只有一个latent vector。 无论是在（ESPN， Nike），还是（ESPN， Male）都用同一个latent vector。但是Nike Male属于不同的field，所以可能需要使用不同的latent vector。

这就是FFM的出发点：每一个特征，针对不同的field使用不同的latent vector.

FFM函数的决策函数是：

$$\boldsymbol{w}_{\text{ESPN,A}} \cdot \boldsymbol{w}_{\text{Nike,P}} + \boldsymbol{w}_{\text{ESPN,G}} \cdot \boldsymbol{w}_{\text{Male,P}} + \boldsymbol{w}_{\text{Nike,G}} \cdot \boldsymbol{w}_{\text{Male,A}}.$$

在FFM中，特征ESPN不再是一个latent vector用到底，而是针对Field A和Field G使用不同的隐向量来计算。因为隐向量变多了，所以隐向量的维度会远远小于FM的。

3.3.2 FFM模型—方程

根据FFM对Field敏感的特性，可以导出其模型方程为：

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_{i,f_j}, \mathbf{v}_{j,f_i} \rangle x_i x_j$$

其中，fj是第j个特征所属的field，fi是第i个特征所属的field。如果隐向量的维度是k，field的个数是f，那么FFM的参数个数是nfk。FFM的二次项并不能化简，计算复杂度就是O(kn^2)

3.3.3 FFM模型—学习算法

FFM模型使用**logistic loss**作为损失函数，加上L2惩罚项，因此只能用于**二分类**。损失函数：

$$\min_{\mathbf{w}} \sum_{i=1}^{L} \log\left(1+\exp\{-y_i\phi(\mathbf{w},\mathbf{x}_i)\}\right)+\frac{\lambda}{2}\|\mathbf{w}\|^2$$

$$\phi(\mathbf{w},\mathbf{x})=\sum_{j_1,j_2\in\mathcal{C}_2}\langle\mathbf{w}_{j_1,f_2},\mathbf{w}_{j_2,f_1}\rangle x_{j_1}x_{j_2}$$

其中，yi是-1，+1表示label，L是样本数量，lambda是惩罚项系数。模型采用SGD优化，优化流程如下：

Algorithm 1 SGD(tr, va, pa)

$model = \textbf{init}(tr.n, tr.m, pa)$
$R_{tr} = \mathbf{1}, R_{va} = \mathbf{1}$
if $pa.norm$ **then**
  $R_{tr} = \textbf{norm}(tr), R_{va} = \textbf{norm}(va)$
end if
for $it = 1, \cdots, pa.itr$ **do**
  if $pa.rand$ **then**
    $tr.X = \textbf{shuffle}(tr.X)$
  end if
  for $i = 1, \cdots, tr.l$ **do**
    $\phi = \textbf{calc}\Phi(tr.X[i], R_{tr}[i], model)$
    $e\phi = \exp\{-tr.Y[i] * \phi\}$
    $L_{tr} = L_{tr} + \log\{1 + e\phi\}$
    $g_\Phi = -tr.Y[i] * e\phi/(1 + e\phi)$
    $model = \textbf{update}(tr.X[i], R_{tr}[i], model, g_\Phi)$
  end for
  for $i = 1, \cdots, va.l$ **do**
    $\phi = \textbf{calc}\Phi(va.X[i], R_{va}[i], model)$
    $L_{va} = L_{va} + \log\{1 + \exp\{-va.Y[i] * \phi\}\}$
  end for
end for

tr,va,pa分别是训练样本集，验证样本集，参数设置。流程如下：

1. 根据样本特征数量（tr.n）,样本field数量(tr.m)，以及参数pa来初始化model。也就是随机的生成模型的参数
2. 如果pa.norm为真，就对训练集tr和验证集va，在**样本维度**进行归一化。也就是除以每个样本的模
3. 对每一轮迭代，如果pa.rand为真，就先打乱训练集tr的顺序。一共迭代pa.itr轮
4. 对**训练集中每一个样本**执行如下操作：
   a. 根据模型当前参数，计算当前样本FFM的输出项f(x)
   b. 使用交叉熵损失函数，计算当前样本的损失 log(1 + exp(-y*f(x)))
   c. 利用单个样本的损失函数，计算梯度，再根据梯度来更新模型参数
5. 对于验证集每一个样本，计算FFM输出和损失
6. 重复3~5，直到到达设置迭代轮数，或验证误差达到最小

为了加快SGD算法的速度，FFM的实现中采用了如下四种**优化**：

1. 梯度分布计算。

$$\mathcal{L} = \mathcal{L}_{err} + \mathcal{L}_{reg} = \log\left(1 + \exp\{-y_i\phi(\mathbf{w}, \mathbf{x}_i)\}\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}_{err}}{\partial \phi} \cdot \frac{\partial \phi}{\partial \mathbf{w}} + \frac{\partial \mathcal{L}_{reg}}{\partial \mathbf{w}}$$

损失函数求导中，$\frac{\partial \mathcal{L}_{err}}{\partial \phi}$ 这一部分和单个W无关，可以只计算一次，之后更新每一个W的时候都用这一个值。W参数的个数为nfk，这样做可以极大的减少运算量。

2. AdaGrad自适应调整学习速率。

普通的SGD调整学习率用指数递减，而FFM是参考AdaGrad来更新的。

$$w'_{j_1,f_2} = w_{j_1,f_2} - \frac{\eta}{\sqrt{1 + \sum_t (g^t_{w_{j_1,f_2}})^2}} \cdot g_{w_{j_1,f_2}}$$

其中，Wj1,f2是特征j1针对field f2的隐向量的一个因素(k中的一个，为了简单k维度的下标未给出)。t表示每一轮的迭代。从式子中可以看出，随着迭代的进行，不断的累积梯度，使得学习速率逐渐下降。但是，每个参数学习速率的更新速度是不同的，与其历史梯度有关。根据AdaGrad的特点，对于样本比较稀疏的特征，学习速率要大于样本比较密集的特征。所以参数可以较快速达到最优，又不会造成损失误差的大幅震荡。

3. OpenMP多核并行计算。

> OpenMP基于**共享内存**实现多核并行计算。**FFM并行化的点**：在上面算法的第四步，对每一个样本点执行，求FFM输出，求损失，求梯度，更新梯度模型参数。是在**样本层面的并行化**。

4. SSE指令集加快**向量内积**运算。

> SSE是CPU对数据层并行的关键指令，常用于多媒体和游戏的应用程序中。FFM中有大量的向量运算，采用SSE来加快向量内积的运算速度，对模型训练非常有利。

3.3.4 FFM模型—多值类别型特征

原始的FFM模型建模时，对于离散特征做one-hot处理，这样同一个样本，同一个field只会有一个特征取值为1，其余的都是0.
但是，当原始的离散特征取值有多个时，该怎么处理那？比如：

User	Movie	Genre	Price
YuChin	3Idiots	Comedy, Drama	$9.99

对于Genre这一个类别型变量来说，一条样本中它可能有多个取值。我们的做法是：**依旧认为他们属于同一个Field，但是属于不同的特征**。
对于上述训练集，我们会得到5个特征，其中 `Genre=Comedy` 和 `Genre=Drame` 是属于同一个Field的两个不同特征维度。FFM要求我们对field进行编号，然后对feature进行编号。这两个编号是单独进行的。Price是连续型数值，不用one-hot。如下：

Field name	Field index	Feature name	Feature index
User	1	User=YuChin	1
Movie	2	Movie=3Idiots	2
Genre	3	Genre=Comedy	3
Price	4	Genre=Drama	4
		Price	5

对应的FFM输出为：

$$\begin{aligned}\langle \mathbf{v}_{1,2}, \mathbf{v}_{2,1}\rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{3,1}\rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{4,1}\rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,4}, \mathbf{v}_{5,1}\rangle \cdot 1 \cdot 9.99 \\ + \langle \mathbf{v}_{2,3}, \mathbf{v}_{3,2}\rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{2,3}, \mathbf{v}_{4,2}\rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{2,4}, \mathbf{v}_{5,2}\rangle \cdot 1 \cdot 9.99 \\ + \langle \mathbf{v}_{3,3}, \mathbf{v}_{4,3}\rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{3,4}, \mathbf{v}_{5,3}\rangle \cdot 1 \cdot 9.99 \\ + \langle \mathbf{v}_{4,4}, \mathbf{v}_{5,3}\rangle \cdot 1 \cdot 9.99\end{aligned}$$

下标中，蓝色对应feature编号，红色对应field编号。绿色是特征取值。

4. 各模型计算复杂度

	#variables	complexity
LM	n	$O(\bar{n})$
Poly2	B	$O(\bar{n}^2)$
FM	nk	$O(\bar{n}k)$
FFM	nfk	$O(\bar{n}^2k)$

FFM是计算复杂度最高的。FM很好的平衡了计算开销和效果。其实这是一个trade off。

5. 优缺点

FFM优点:
细化隐向量的表示，同一特征针对不同field使用不同隐向量，模型建模更加准确

FFM缺点:
计算复杂度比较高，参数个数为nfk，计算复杂度为O(k * n^2)

6. 使用FFM需要注意的地方

1. 样本归一化。即pa.norm为真，对样本进行归一化，否则容易造成数据溢出，梯度计算失败。
2. 特征归一化。为了消除不同特征取值范围不同，量纲不同造成的问题，需要对特征进行归一化。
3. Early stopping。一定要设置早停策略，FFM很容易过拟合。
4. 省略零值特征。零值特征对模型没有任何贡献，无论是1次项还是2次项都为0。这也是系数样本采用FFM的显著优势。

7. 代码实战

FFM的原理不是特别复杂，优化算法AdaGrad也给出了详细的算法流程。但是，自己实现的话，很难加入SSE、OpenMP以及梯度计算的优化。所以自己实现的往往速度很慢。论文中作者给出了实现libffm，github：https://github.com/guestwalk/libffm
需要先编译成可执行文件，再提供ffm格式的训练集和验证集来运行:

```
./ffm-train -l 0.0001 -k 15 -t 30 -r 0.05 -s 10 --auto-stop -p libffm_toy/criteo.va.r100.gbdt0.ffm libffm_toy/criteo.tr.r100.gbdt0.ffm model
```

参数说明如下：

参数	说明
l	正则项系数
k	隐向量维度
t	迭代代数
r	初始学习速率
s	并行化线程数量
p	指定valid dataset路径
—auto-stop	验证集loss最小时提前停止,必须配合-p使用
—no-norm	关掉**样本维度**归一化

生成的模型保存在model中，然后使用model进行预测：

```
./ffm-predict ./libffm_toy/criteo.va.r100.gbdt0.ffm model output
```

output保存预测结果

libffm要求输入文件格式为：

```
<label> <field1>:<feature1>:<value1> <field2>:<feature2>:<value2> ...
```

训练：

```
ubuntu@LiNingNing0_121:~/projects/libffm/playground$ ./ffm-train -l 0.0001 -k 15 -t 30 -r 0.05 -s 10 --auto-stop -p libffm_toy/criteo.va.r100.gbdt0.ffm libffm_toy/criteo.tr.r100.gbdt0.ffm model
First check if the text file has already been converted to binary format (0.2 seconds)
Binary file found. Skip converting text to binary
First check if the text file has already been converted to binary format (0.0 seconds)
Binary file found. Skip converting text to binary
iter   tr_logloss   va_logloss      tr_time
   1      0.52733      0.50369          4.2
   2      0.49283      0.49698          8.6
   3      0.48680      0.49302         13.0
   4      0.48262      0.49010         17.4
   5      0.47940      0.48786         21.8
   6      0.47673      0.48620         26.2
   7      0.47447      0.48473         30.9
   8      0.47246      0.48344         35.5
   9      0.47069      0.48251         39.8
  10      0.46903      0.48142         44.1
  11      0.46749      0.48055         48.4
  12      0.46604      0.47985         52.9
  13      0.46472      0.47901         57.1
  14      0.46339      0.47836         61.4
  15      0.46215      0.47772         65.5
  16      0.46096      0.47720         70.0
  17      0.45978      0.47666         74.5
  18      0.45866      0.47615         78.9
  19      0.45754      0.47569         83.4
  20      0.45646      0.47530         87.9
  21      0.45543      0.47483         92.3
  22      0.45436      0.47446         96.8
  23      0.45335      0.47407        101.3
  24      0.45233      0.47373        105.8
  25      0.45130      0.47342        110.3
  26      0.45031      0.47318        114.8
  27      0.44934      0.47286        119.5
  28      0.44836      0.47259        124.2
  29      0.44738      0.47236        128.7
  30      0.44641      0.47211        133.4
```

预测：

```
ubuntu@LiNingNing0_121:~/projects/libffm/playground$ ./ffm-predict ./libffm_toy/criteo.va.r100.gbdt0.ffm model output
logloss = 0.47211
```

代码和文件可以参考github： https://github.com/gutouyu/ML_CIA

Reference

1. Field-aware Factorization Machines for CTR Prediction
2. 美团点评团队 https://tech.meituan.com/deep-understanding-of-ffm-principles-and-practices.html
3. libffm https://github.com/guestwalk/libffm

计算广告CTR预估系列往期回顾：
计算广告CTR预估系列(一)—DeepFM理论
计算广告CTR预估系列(二)—DeepFM实践

获取更多机器学习干货、荐货，欢迎关注**机器学习荐货情报局**，加入荐货大家庭！