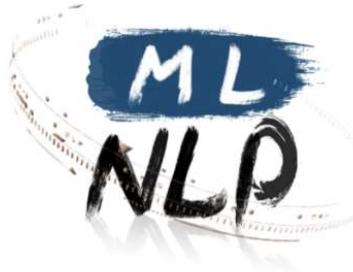


史上最全推荐系统传统算法合集

机器学习算法与自然语言处理 2022-01-19 08:55



机器学习算法 与自然语言处理

MLNLP(机器学习算法与自然语言处理)社区是国内外最大的自然语言处理社区之一，汇聚超过50w订阅者，受众覆盖国内外NLP硕博生、高校老师以及企业研究人员。

社区的愿景是促进国内外自然语言处理，机器学习学术界、产业界和广大爱好者之间的交流和进步。

来源 | PaperWeekly

我花了半个多月将推荐系统传统算法分别进行了总结归纳，应该时目前全网最全的版本了。希望对大家了解推荐系统传统算法有所帮助。

推荐系统的传统算法主要包括：

- 基于邻域的算法
- 隐语义模型
- 决策树模型
- 逻辑回归

01

基于邻域的算法

主要介绍了 user-based CF (协同过滤)，item-based CF 的原理以及他们的对比，另外还有相关代码和优化实验结果。详细内容：

1.1 基于邻域的算法 (协调过滤)

1.1.1 UserCF

算法步骤：

1. 找到和目标用户兴趣相似的用户集合；
2. 将集合中用户喜欢的未出现在目标用户的兴趣列表中的 item 以一定的权值排序后推荐给用户

用户相似度计算：

$$w_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

改进：

1. 由于很多用户两两之间并没有对同样的物品产生过行为，用用户-物品表计算用户相似度很多时候计算都是浪费的，可以利用物品-用户倒排表简化计算。
2. 加入热门商品惩罚，用户对冷门物品采取相同行为更能说明两者的相似性

1.1.2 ItemCF

算法步骤：

1. 计算物品之间的相似性
2. 根据物品的相似度和用户的历史行为给用户生成推荐列表

物品相似度计算：

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|}$$

1. 余弦相似度

改进：

1. 加入热门用户惩罚，或者直接忽略过分活跃用户
2. 利用用户-物品倒排表
3. 同类物品相似度归一化

哈利波特问题：物品太热门

1. 加加大对热门物品的惩罚

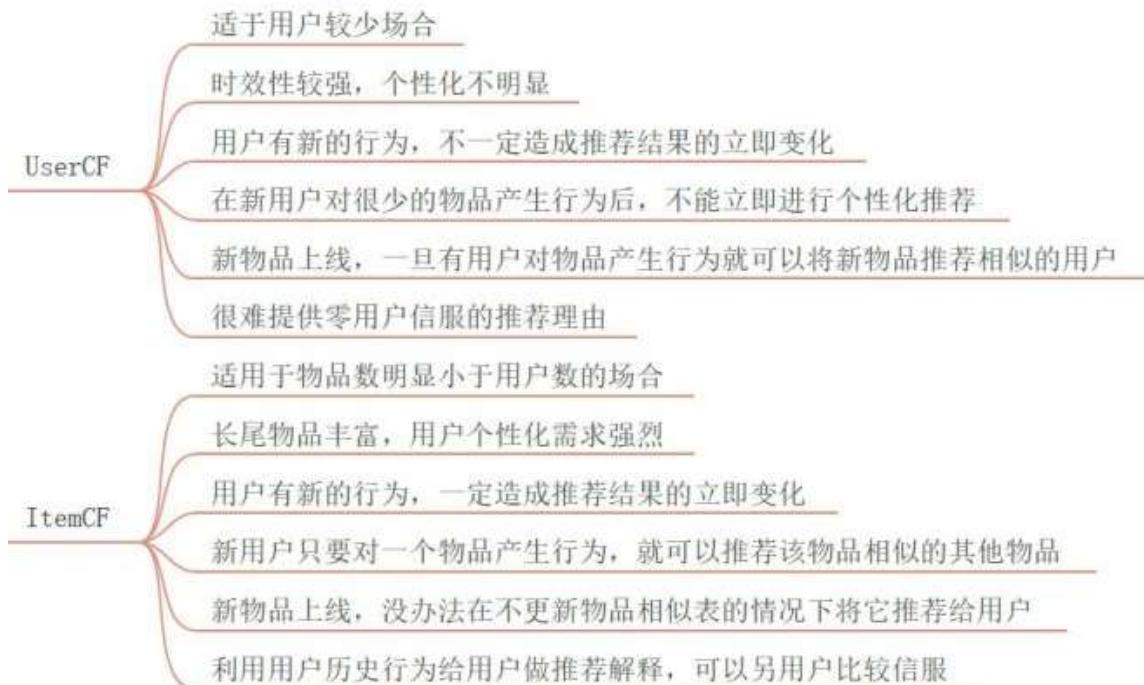
$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|^{1-\alpha} |N(j)|^\alpha}$$

2. 两个不同领域的热门商品有很高的相似度

解决方法：

- 1.引入物品的内容数据
- 2.对不同领域的物品降权

1.1.3 UserCF与ItemCF比较



1.1.4 UserCF与ItemCF实验分析（基于movielens数据集）

代码链接：github.com/mercurialgh/

1.2 UserCF 实验

K 代表最当前用户最相似的 k 个用户，nitems 代表推荐的商品数目

不加入热门商品惩罚：

K=8

1.nitems=5

```
recall: 0.06346, precision: 0.23404, coverage: 0.27570, popularity: 7.18761
```

1.nitems=10

```
recall: 0.10619, precision: 0.19579, coverage: 0.36066, popularity: 7.08531
```

1.nitems=20

```
recall: 0.16985, precision: 0.15659, coverage: 0.46834, popularity: 6.96057
```

1.nitems=40

```
recall: 0.25839, precision: 0.11911, coverage: 0.59876, popularity: 6.81221
```

1.nitems=160

```
recall: 0.51439, precision: 0.05928, coverage: 0.87716, popularity: 6.43734
```

选择与用户相似度最高的 8 个用户，随着推荐数目 k 的升高，召回率升高，准确率下降，覆盖率升高，流行度降低，符合直观理解。

nitems=10

1.k=4

```
recall: 0.09363, precision: 0.17265, coverage: 0.44183, popularity: 6.97563
```

1.k=8

```
recall: 0.10619, precision: 0.19579, coverage: 0.36066, popularity: 7.08531
```

1.k=16

```
recall: 0.11516, precision: 0.21233, coverage: 0.29329, popularity: 7.17953
```

1.k=32

```
recall: 0.12032, precision: 0.22185, coverage: 0.23593, popularity: 7.26256
```

1.k=160

```
recall: 0.11658, precision: 0.21497, coverage: 0.13014, popularity: 7.43525
```

每次推荐的数目为 10，选取的相似用户的数量 k 从 4 到 32，召回和准确率都提升了，但是 k 从 32 到 160，对于模型没有性能提升，反而损失了一部分性能，k 的增大会使得覆盖率降低，商品的流行度上升。

加入热门商品惩罚：

1.k=8,n=10，不热门商品惩罚：

```
recall: 0.10619, precision: 0.19579, coverage: 0.36066, popularity: 7.08531
```

1.k=8,n=10，热门商品惩罚

```
recall: 0.10523, precision: 0.19402, coverage: 0.37689, popularity: 7.03856
```

与书中描述的不同的是，我加入了惩罚项反而使得召回和准确率稍微下降了，覆盖率提升了一点。

1.3 ItemCF实验

K 代表与当前用户感兴趣的物品最相似的 k 个物品，nitems 表示推荐的数量

不加入活跃用户惩罚：

K=8

1.n=5

```
recall: 0.06645, precision: 0.24507, coverage: 0.13555, popularity: 7.36820
```

1.n=10

```
recall: 0.10963, precision: 0.20215, coverage: 0.19129, popularity: 7.28785
```

1.n=20

```
recall: 0.17471, precision: 0.16108, coverage: 0.28111, popularity: 7.17559
```

1.n=40

```
recall: 0.26480, precision: 0.12207, coverage: 0.39881, popularity: 7.02130
```

选择最相似的 8 个物品，推荐数目递增，跟 UserCF 第一个实验的结果一样，召回提高，准确率下降，覆盖率提升，商品流行度下降。

nitems=10

1.k=4

```
recall: 0.10673, precision: 0.19679, coverage: 0.21050, popularity: 7.20988
```

1.k=8

```
recall: 0.10963, precision: 0.20215, coverage: 0.19129, popularity: 7.28785
```

1.k=16

```
recall: 0.11010, precision: 0.20301, coverage: 0.17208, popularity: 7.36693
```

1.k=100

```
recall: 0.42877, precision: 0.07906, coverage: 0.58712, popularity: 6.85058
```

推荐 10 个物品，选择最相似商品的个数 k 从 4 到 16 的时候，召回和准备率都提升了，覆盖率下降，流行度升高。但是 k 过大时，模型的性能反而会有损失。同时响应时间也会慢很多

加入活跃用户惩罚：

1.k=8,n=10,不加入活跃用户惩罚：

```
recall: 0.10963, precision: 0.20215, coverage: 0.19129, popularity: 7.28785
```

1.k=8,n=10,加入活跃用户惩罚：

```
recall: 0.11084, precision: 0.20437, coverage: 0.17641, popularity: 7.37799
```

加入活跃用户惩罚后，模型性能有了较小的提升。

同类物品相似度归一化

1.k=8,n=10,不进行归一化：

```
recall: 0.10963, precision: 0.20215, coverage: 0.19129, popularity: 7.28785
```

1.k=8,n=10,进行归一化：

```
recall: 0.11591, precision: 0.21373, coverage: 0.23187, popularity: 7.27046
```

可以看出加入归一化后所有指标都提升了，尤其是覆盖率提升了很多，说明同类物品归一化是有效的。

02

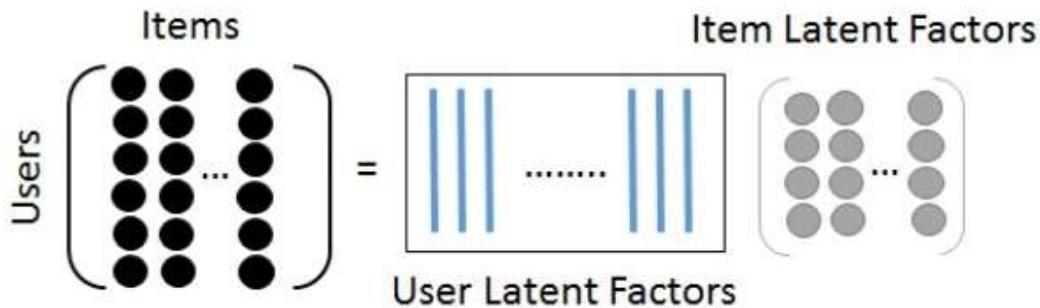
隐语义模型

主要介绍了隐语义模型原理，矩阵分解，以及 Factorization Machine (FM) 和 Field-aware Factorization Machine (FFM) 的原理和推导。详细内容：

在隐语义模型中，我们使用同样的维度来表征 (Embedding) item 和用户。对于 item，这个表征就是 item 表现出的对应维度的特征强度；对于用户，就是用户表现出的对对应维度特征的偏好强度。用用户的表征向量点乘 item 的表征向量，就可以得到用户对该条目的偏好描述。

$$R(u, i) = p_u^T q_i = \sum_{k=1}^K p_{u,k} q_{i,k}$$

表示用户 u 对 item i 的喜好程度，其中关于用户和条目的描述维度有 k 个，这个参数是自定义的。



我们对模型的优化目标：

$$\min \sum \|\hat{R}_{u,i} - R_{u,i}\|_F^2$$

其中 $\hat{R}_{u,i}$ 为实际喜好值， $\|\cdot\|_F$ 为 Frobenius 范数。为了抑制过拟合，需要加入正则化参数：

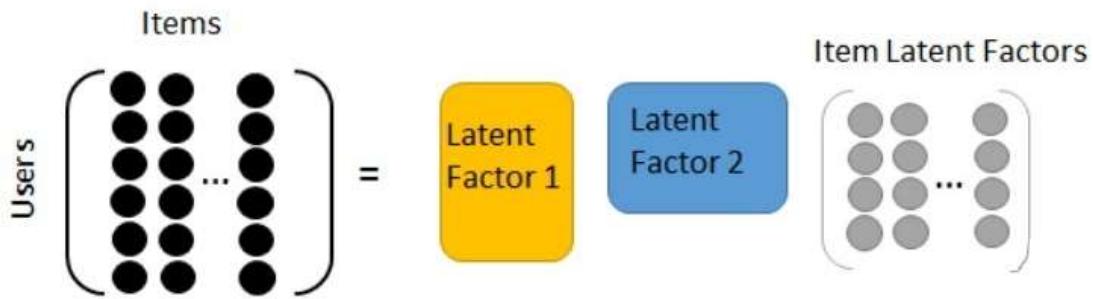
$$\min \sum \left(\|\hat{R}_{u,i} - R_{u,i}\|_F^2 + \lambda (\|p_u\|_F^2 + \|q_i\|_F^2) \right)$$

本质上讲，隐语义模型是不含非线性过程的单层图神经网络。一般来讲，我们观测到的用户对 item 的行为是很少的，用户对大部分 item 没有互动和喜好参数，特别是没有显式的负样本。

负样本构造应该遵循的原则：

1. 对每个用户，要保证正负样本均衡
2. 对每个用户采样负样本时，要选取那些热门，而用户没有行为的物品

由于有训练过程，LFM 不能用于实时推荐，但可以作为 Hybrid System 中的一个选项。当然，也可以将 LFM 演变成 2 层神经网络，增加其表征能力：



这种表征类似于矩阵分解 (Matrix Factorization)。不过这种改进一般使 Recall 的提升在几个百分点内，而计算量上大幅提升。为了避免计算导致的延迟，可以在模型确定之后对每个用户计算一个推荐条目表，在部署时直接查表以免除计算过程。

2.1 Factorization Machine (FM)

针对的问题：

1. 类别特征经过 one-hot 编码后数据稀疏性
2. 一些特征两两组合之后往往与 label 有更强的关联性

多项式模型是比较直观的包含特征组合的模型，二阶形式如下：

$$y(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^n w_{ij} x_i x_j$$

特征 $x_i x_j$ 的组合用 $x_i x_j$ 表示，只有当两者都非零时才有意义，该模型的主要问题：

1. 特征数量为 N ，二次项系数为 $N(N-1)/2$ ，复杂度太高
2. one-hot 特征太稀疏，特征组合之后更加稀疏

FM 的模型方程：

$$y(x) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \vec{v}_i \vec{v}_j \rangle w_i w_j$$

利用了推荐系统中的矩阵分解思想，实际就是 LFM (latent factor model) 隐因子模型。
参数量由 $N(N-1)/2$ 降到 $k^2 n$ 。



$$\begin{aligned}
\sum_{i=1}^n \sum_{j=i+1}^n < V_i \cdot V_j > x_i x_j &= \frac{1}{2} \left(\sum_{i=1}^n \sum_{j=1}^n < V_i \cdot V_j > x_i x_j - \sum_{i=1}^n < V_i \cdot V_i > x_i x_i \right) \\
&= \frac{1}{2} \left(\sum_{i=1}^n \sum_{j=1}^n \left(\sum_{f=1}^k v_{if} v_{jf} \right) x_i x_j - \sum_{i=1}^n \left(\sum_{f=1}^k v_{if} v_{if} \right) x_i x_i \right) \\
&= \frac{1}{2} \sum_{f=1}^k \left[\left(\sum_{i=1}^n v_{if} x_i \right) \cdot \left(\sum_{j=1}^n v_{jf} x_j \right) - \sum_{i=1}^n v_{if}^2 x_i^2 \right] \\
&= \frac{1}{2} \sum_{f=1}^k \left[\left(\sum_{i=1}^n v_{if} x_i \right)^2 - \sum_{i=1}^n v_{if}^2 x_i^2 \right]
\end{aligned}$$

原来只有 $x_i x_j$ 都不为 0 时才能计算，样本量很少，现在只要 x_i 不为 0 就可以计算，解决了样本量的问题，并且参数量减少。

注意这里公式之所以能这么化简，是因为我们做了假设：即每一个特征都可以用一个 k 维的隐向量来表示。但是这个假设不一定成立，例如行业 (industry) 特征做 one-hot 后，不同行业都用 k 维向量表示还行，但此时性别 (gender) 特征也用 k 维向量表示这显然不太合理。若该假设有问题，则公式不成立。

2.2 Field-aware Factorization Machine (FFM)

FFM 将特征按不同的规则分到多个场 (field)，特征 x_i 属于某个特定的场 f ，每个特征 x_i 可以被映射为多个隐向量 v_{i1}, \dots, v_{if} ，每个隐向量对应一个场，当特征组合时，用特征对应的场的隐向量进行内积。

$$w_{ij} = v_{i,f_j}^T v_{j,f_i}$$

FFM 由于引入了场，使得每两组特征交叉的隐向量都是独立的，可以取得更好的组合效果，但是使得计算复杂度无法通过优化变成线性时间复杂度，每个样本预测的时间复杂度为，不过 FFM 的 k 值通常远小于 FM 的 k 值。FM 可以看做只有一个场的 FFM。

2.3 LFM和基于邻域的方法的比较

1. LFM 具有比较好的理论基础，它是一种学习方法，通过优化一个设定的指标建立最优的模型。基于邻域的方法更多的是一种基于统计的方法，并没有学习过程。
2. LFM 在生成一个用户推荐列表时速度太慢，因此不能在线实时计算。
3. ItemCF 算法支持很好的推荐解释，它可以利用用户的历史行为解释推荐结果。但 LFM 无法提供这样的解释。

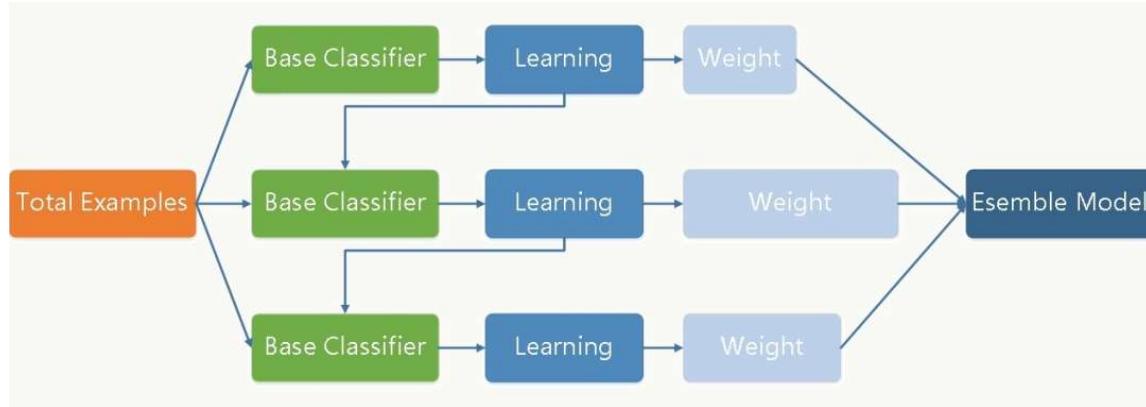
03

决策树模型



主要介绍了 GBDT 的原理和推导，包括 GBDT 回归算法，GBDT 分类算法（二分类，多分类）以及正则化。XGB 的原理和推导，正则化，节点分类准则（贪心准则，近似算法，稀疏感知等）以及工程优化原理。详细内容：

3.1 GBDT



bdt 通过多轮迭代，每轮迭代产生一个弱分类器，每个分类器在上一轮分类器的残差基础上进行训练。对弱分类器的要求一般是足够简单，并且是低方差和高偏差的。因为训练的过程是通过降低偏差来不断提高最终分类器的精度。

弱分类器一般会选择为 CART（也就是分类回归树）。由于上述高偏差和简单的要求每个分类回归树的深度不会很深。最终的总分类器是将每轮训练得到的弱分类器加权求和得到的（加法模型）。

算法流程如下：

1. 初始化 $f_0(x)=0$
2. 对 $t=1, 2, 3, \dots, T$ ，
 - 计算残差 $r=y_i-f_{t-1}(x)$
 - 拟合残差学习得到弱分类器 $h_t(x)$
 - 得到新的强学习器 $f_t(x)=f_{t-1}(x)+h_t(x)$
3. 树的深度达到阈值或者残差小于阈值，得到最终的强学习器

$$f_t(x) = f_0(x) + \sum_1^T h_t(x)$$

3.1.1 GBDT回归算法

假设训练集样本 $T=(x,y_1)(x,y_2)\dots(x,y_m)$ ，最大迭代次数 T ，损失函数 L ，输出的强学习器 $f(x)$ ，回归算法的流程如下：

1. 初始化弱学习器， c 的值可以设为样本 y 的均值

$$f_0(x) = \operatorname{argmin}_c \sum_{i=1}^m L(y_i, c)$$

2. 对迭代次数 $t=1, \dots, T$;

- 对样本 $i=1, \dots, m$; 计算梯度

$$r_{ti} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \quad f(x) = f_{t-1}(x)$$

- 利用 (x_i, r_{ti}) , 拟合一棵 CART 树, 得到第 t 棵回归树, 其对应的叶子节点区域为 R_{tj} , $j=1, 2, \dots, J$ 。其中 J 为回归树 t 的叶子节点个数。
- 对叶子区域 $j=1, 2, 3, \dots, J$, 计算最佳拟合值:

$$c_{tj} = \operatorname{argmin}_c \sum_{x \in R_{tj}} L(y_i, f_{t-1}(x_i) + c)$$

- 更新强学习器

$$f_t(x) = f_{t-1}(x) + \sum_{j=1}^J c_{tj}, I(x \in R_{tj})$$

3. 得到最终的强学习器

$$f(x) = f_0(x) + \sum_{t=1}^T \sum_{j=1}^J c_{tj}, I(x \in R_{tj})$$

3.1.2 GBDT 分类算法

GBDT 分类算法在思想上和回归算法没有区别, 但是由于样本输出不是连续的值, 而是离散的类别, 导致我们无法直接从输出类别去拟合类别输出的误差。为解决此问题, 我们尝试用类似于逻辑回归的对数似然损失函数的方法, 也就是说我们用的是类别的预测概率值和真实概率值来拟合损失函数。对于对数似然损失函数, 我们有二元分类和多元分类的区别。

1. 二分类

对于二元 GBDT, 如果用类似于逻辑回归的对数似然损失函数, 则损失函数表示为 $L(y, f(x)) = \log(1 + e^{-y f(x)})$, 其中 $y \in \{-1, 1\}$ 此时的负梯度误差为

$$r_{ti} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \quad f(x) = f_{t-1}(x) = \frac{-y_i \exp(-y_i f(x_i))}{1 + \exp(-y_i f(x_i))}$$

由于上式比较难优化, 我们一般使用近似值代替:

$$c_{tj} = \frac{\sum_{x_i \in R_{tj}} r_{ti}}{\sum_{x_i \in R_{tj}} |r_{ti}|(1 - |r_{ti}|)}$$

除了负梯度计算和叶子节点的最佳残差拟合的线性搜索外，二元 GBDT 分类和 GBDT 回归算法过程相同。

2. 多分类

多元 GBDT 要比二元 GBDT 复杂一些，对应的是多元逻辑回归和二元逻辑回归的复杂度差别。假如类别数为 K，则我们的对数似然函数为：

其中如果样本输出类别为 k，则 $y_{ik}=1$ ，第 k 类的概率 $p_k(x)$ 的表达式为：

集合上两式，我们可以计算出第 t 轮的第 i 个样本对应类别 l 的负梯度误差为：

其实这里的误差就是样本 i 对应类别的真实概率和 t-1 轮预测概率的差值。对于生成的决策树，我们各个叶子节点的最佳残差拟合值为：

由于上式比较难优化，我们用近似值代替：



除了负梯度计算和叶子节点的最佳残差拟合的线性搜索，多元 GBDT 分类和二元 GBDT 分类以及 GBDT 回归算法过程相同。

3.1.3 正则化

和 Adaboost 一样，我们也需要对 GBDT 进行正则化，防止过拟合。GBDT 的正则化主要有 3 种方式：learning_rate（学习率）；subsample（子采样比例）；min_samples_split（叶子结点包含的最小样本数）。

针对 GBDT 正则化，我们通过子采样比例方法和定义步长 v 方法来防止过拟合。

1. learning_rate。学习率是正则化的一部分，它可以降低模型更新的速度（需要更多的迭代）。通常我们用 learning_rate 和最大迭代次数一起来决定算法的拟合效果。

- 经验表明：一个小的学习率 ($v < 0.1$) 可以显著提高模型的泛化能力（相比较于 $v=1$ ）。
- 如果学习率较大会导致预测性能出现较大波动。

2. subsample，子采样比例。Freidman 从 bagging 策略受到启发，采用随机梯度提升来修改了原始的梯度提升树算法。

每一轮迭代中，新的决策树拟合的是原始训练集的一个子集（而并不是原始训练集）的残差。这个子集是通过对原始训练集的无放回随机采样而来。无放回抽样的子采样比例 (subsample)，取值为 (0,1]。如果取值为 1，则与原始的梯度提升树算法相同，即使用全部样本。

如果取值小于 1，则使用部分样本去做决策树拟合。较小的取值会引入随机性，有助于改善过拟合，因此可以视作一定程度上的正则化；但是会增加样本拟合的偏差，因此取值不能太低。推荐在 [0.5, 0.8] 之间。这种方法除了改善过拟合之外，另一个好处是：未被采样的另一部分子集可以用来计算包外估计误差。因此可以避免额外给出一个独立的验证集。

3. min_samples_split，叶子结点包含的最小样本数。梯度提升树会限制每棵树的叶子结点包含的样本数量至少包含 m 个样本，其中 m 为超参数。在训练过程中，一旦划分结点会导致子结点的样本数少于 m ，则终止划分。

3.2 XGB

3.2.1 算法原理

和 GBDT 一样，XGBoost 是一个加法模型，在每一步迭代中只优化当前步中的子模型。在第 m 步中：

目标函数为经验风险+结构风险（正则项）：



由

可以得到上式二阶泰勒展开：

前 $m-1$ 个子模型已经确定了，故上式中除了关于 $f_m(x)$ 的部分都是常数，不影响对 $f_m(x)$ 的优化求解。目标函数可转化为：

其中：

这里的 L 是损失函数，度量一次预测的好坏。在 $F_{m-1}(x)$ 确定了的情况下，对每个样本点 i 都可以轻易计算出一个 g_i 和 h_i 。

3.2.2 xgb的正则化

为防止过拟合，XGBoost 设置了基于树的复杂度作为正则项：

T 为树 f 的叶节点个数， w 为所有叶节点输出回归值构成的向量，由 (1)，(2) 可知：



接下来通过一个数学处理，可以使得正则项和经验风险项合并到一起。经验风险项是在样本层面上求和，我们将其转换为叶节点层面上的求和。

3.2.3 节点分裂准则

1. 贪心准则

XGBoost 的子模型树和决策树模型一样，要依赖节点递归分裂的贪心准则来实现树的生成。除此外，XGBoost 还支持近似算法，解决数据量过大超过内存、或有并行计算需求的情况。

基本思路和 CART 一样，对特征值排序后遍历划分点，将其中最优的分裂收益作为该特征的分裂收益，选取具有最优分裂收益的特征作为当前节点的划分特征，按其最优划分点进行二叉划分，得到左右子树。

上图是一次节点分裂过程，很自然地，分裂收益是树 A 的评分减去树 B 的评分。虚线框外的叶节点，即非分裂节点的评分均被抵消，只留下分裂后的 LR 节点和分裂前的 S 节点进行比较，因此分裂收益的表达式为：

2. 近似算法

XGBoost 还提供了上述贪心准则的近似版本，简言之，将特征分位数作为划分候选点。这样将划分候选点集合由全样本间的遍历缩减到了几个分位数之间的遍历。

具体而言，特征分位数的选取有 global 和 local 两种可选策略：global 在全体样本上的特征值中选取，在根节点分裂之前进行一次即可；local 则是在待分裂节点包含的样本特征值上选取，每个节点分裂前都要进行。通常，global 由于只能划分一次，其划分粒度需要更细。

在 XGB 原始论文中，作者在 Higgs Boson 数据集上比较了精确贪心准则、global 近似和 local 近似三类配置的测试集 AUC，用 eps 代表取分位点的粒度，如 eps=0.25 代表将数据

集划分为 $1/0.25=4$ 个 buckets，发现 global ($\text{eps}=0.05$) 和 local ($\text{eps}=0.3$) 均能达到和精确贪心准则几乎相同的性能。

这三类配置在 XGBoost 包均有支持。

3. 加权分位数

查看 (1) 式表示的目标函数，令偏导为 0 易得

此目标函数可理解为以 h_i 为权重， $-g_i/h_i$ 为标签的二次损失函数：

因此，在近似算法取分位数时，实际上 XGBoost 会取以二阶导 h_i 为权重的分位数。

4. 列采样和学习率

XGBoost 还引入了两项特性：列采样和学习率。

列采样，即随机森林中的做法，每次节点分裂的待选特征集合不是剩下的全部特征，而是剩下特征的一个子集。是为了更好地对抗过拟合（我不是很清楚 GBDT 中列采样降低过拟合的理论依据。原文这里提到的动机是某 GBDT 的软件用户反馈列采样比行采样更能对抗过拟合），还能减少计算开销。

学习率，或者叫步长、shrinkage，是在每个子模型前（即在每个叶节点的回归值上）乘上该系数，削弱每棵树的影响，使得迭代更稳定。可以类比梯度下降中的学习率。XGBoost 默认设定为 0.3。

5. 稀疏感知

缺失值应对策略是算法需要考虑的。特征稀疏问题也同样需要考虑，如部分特征中出现大量的 0 或干脆是 one-hot encoding 这种情况。XGBoost 用稀疏感知策略来同时处理这两个问题：概括地说，将缺失值和稀疏 0 值等同视作缺失值，再将这些缺失值“绑定”在一起，分裂节点的遍历会跳过缺失值的整体。这样大大提高了运算效率。



分裂节点依然通过遍历得到，NA 的方向有两种情况，在此基础上对非缺失值进行切分遍历。或者可以理解 NA 被分到一个固定方向，非缺失值在升序和降序两种情况下进行切分遍历。

如上图所示，若某个特征值取值为 1,2,5 和大量的 NA，XGBoost 会遍历以上 6 种情况（3 个非缺失值的切分点 × 缺失值的两个方向），最大的分裂收益就是本特征上的分裂收益，同时，NA 将被分到右节点。

3.2.4 工程优化

1. 并行块设计

XGBoost 将每一列特征提前进行排序，以块（Block）的形式储存在缓存中，并以索引将特征值和梯度统计量 g_i, h_i 对应起来，每次节点分裂时会重复调用排好序的块。而且不同特征会分布在独立的块中，因此可以进行分布式或多线程的计算。

2. 缓存访问

特征值排序后通过索引来取梯度 g_i, h_i 会导致访问的内存空间不一致，进而降低缓存的命中率，影响算法效率。为解决这个问题，XGBoost 为每个线程分配一个单独的连续缓存区，用来存放梯度信息。

3. 核外块计算

数据量过大时，不能同时全部载入内存。XGBoost 将数据分为多个 blocks 并储存在硬盘中，使用一个独立的线程专门从磁盘中读取数据到内存中，实现计算和读取数据的同时进行。为了进一步提高磁盘读取数据性能，XGBoost 还使用了两种方法：一是通过压缩

block，用解压缩的开销换取磁盘读取的开销；二是将 block 分散储存在多个磁盘中，有助于提高磁盘吞吐量。

3.2.5 与GBDT比较

- 性质：GBDT 是机器学习算法，XGBoost 除了算法内容还包括一些工程实现方面的优化。
- 基于二阶导：GBDT 使用的是损失函数一阶导数，相当于函数空间中的梯度下降；而 XGBoost 还使用了损失函数二阶导数，相当于函数空间中的牛顿法。
- 正则化：XGBoost 显式地加入了正则项来控制模型的复杂度，能有效防止过拟合。
- 列采样：XGBoost 采用了随机森林中的做法，每次节点分裂前进行列随机采样。
- 缺失值处理：XGBoost 运用稀疏感知策略处理缺失值，而 GBDT 没有设计缺失策略。
- 并行高效：XGBoost 的列块设计能有效支持并行运算，提高效率。

04

逻辑回归

主要介绍了逻辑回归的原理和如何在推荐上应用。详细内容：

在推荐系统中，可以将是否点击一个商品看成一个概率事件，被推荐的商品无非两种可能性：1.被点击；2.不被点击。那么就可以将这个推荐问题转换成一个分类问题。逻辑回归是监督学习中的分类算法，所以可以使用逻辑回归来进行一个分类预测。

逻辑回归模型能够综合利用用户，物品，上下文等多种不同的特征生成较全面的推荐结果。

算法步骤

- (1) 将用户年龄、性别、物品属性、物品描述、当前时间、当前地点等特征转换成数值型特征向量；

- (2) 确定逻辑回归模型的优化目标（以优化点击率为例），利用已有样本数据对逻辑回归模型进行训练，确定逻辑回归模型的内部参数
- (3) 在模型服务阶段，将特征向量输入逻辑回归模型，经过逻辑回归模型的推断，得到用户“点击”物品的概率
- (4) 利用“点击概率”对所有候选物品进行排序，得到推荐列表

LR的数学形式如下：

其中 $\theta=(\theta_1, \theta_2, \dots, \theta_n)$ ，假设 $\text{logistic}(x)=1/(1+e^{-x})$ ，则 $f(x)=\text{logistic}(\text{linear}(x))$ 。

所以逻辑回归实际上就是在线性回归的基础上加了 $\text{logistic}(x)$ ，而这个就是所谓的 sigmoid 函数。可以看这篇详细的解答：

<https://www.zhihu.com/question/23666587/answer/462453898>

LR的缺陷：

LR 假设各特征间是相互独立的，忽略了特征间的交互关系，因此需要做大量的特征工程。同时 LR 对非线性的拟合能力较差，限制了模型的性能上限。通常LR可以作为 Baseline 版本。

这里有用逻辑回归做电影推荐的 github 项目，有兴趣的小伙伴可以实操一下：

<https://github.com/LawsonAbs/MovieRecommend>

这些传统算法的思想后续在深度学习应用到推荐系统中都有很多相关的模型，比如 facebook 将协同过滤和深度学习结合的 DLRM，以及后面的 DeepFM，Wide&Deep，Deep&Cross，以及将 Facebook 将 GBDT+LR 结合的方法一度成为业内都使用的模型。所以了解这些算法对于提升自己时很有帮助的。

最后如果有错误希望大家指出，希望和大家一起交流，一起进步！

技术交流群邀请函



△长按添加小助手

扫描二维码添加小助手微信

请备注：姓名-学校/公司-研究方向

(如：小张-哈工大-对话系统)

关于我们

MLNLP(机器学习算法与自然语言处理) 社区是由国内外自然语言处理学者联合构建的民间学术社区，目前已经发展为国内外最大的自然语言处理社区之一，汇聚超过50w订阅者，旗下包括万人顶会交流群、AI臻选汇、AI英才汇以及AI学术汇等知名品牌，旨在促进机器学习，自然语言处理学术界、产业界和广大爱好者之间的进步。

社区可以为相关从业者的深造、就业及研究等方面提供开放交流平台。欢迎大家关注和加入我们。



机器学习算法与自然语言处理

关注AI前沿技术，助力AI学者进步

370篇原创内容

公众号

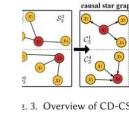
[Read more](#)

People who liked this content also liked

Sam Altman突遭起底「生活奢靡」！戴340万名表，开上亿豪车，买价值6亿豪宅
机器学习算法与自然语言处理



论文日报 | 通过因果星图发现因果关系
伯乐智荐



服务运营 | 年终回顾：服务运营为您服务
运筹OR帷幄



