

# 极客大学机器学习训练营

## 常见机器学习模型

王然

众微科技 AI Lab 负责人

二〇二一年二月三日

1 模型评估策略

2 树模型和提升

3 参考文献

- 1 模型评估策略
  - 模型评估和建模的新方法
- 2 树模型和提升
- 3 参考文献

- 1 模型评估策略
  - 模型评估和建模的新方法
- 2 树模型和提升
- 3 参考文献

- ▶ 在我们第三章开头，我们讲过模型评估一旦出了问题之后，我们将会难以判断我们在算法中的细节是否对于整体准确性有帮助。
- ▶ 在本章开头，我们首先就模型评估的问题做一个简单的回顾。

- ▶ 在传统模型评估方法中，数据集将会被分为三个部分：训练集、验证集（开发集）和测试集；
- ▶ 训练集目的：在给定超参数的情况下，对模型参数进行估计（训练）；
- ▶ 验证集目的：选择超参数；
- ▶ 测试集目的：测试最终选择的模型的真实表现。

- ▶ 我们可以考虑  $l_1$  正则化的结果。
- ▶ 对于训练集来说，最好的结果一定是  $l_1$  正则化的惩罚系数为 0 的情况 (思考题：为什么?)；
- ▶ 所以在这种情况下，在得到了训练集上根据某个正则化惩罚系数的情况下，在验证集上可以得到关于模型泛化性的一个相对客观的评价；



- ▶ 如果我们手头有测试集的结果，则我们从技术上而言，可以用测试集替代验证集的作用；
- ▶ 但是这样做是非常危险的；
- ▶ 我们虽然没有在测试集上直接拟合模型，但是通过测试非常多的超参数我们也近似的达到了效果；
- ▶ 一个更极端的例子，加入测试集当中只有正负样本；我们第一次将预测都设定为负样本；从第二次开始测试开始，我们每一次只改变其中一个测试。只要我们有足够多次的评估，我们就可以把正确的值推算出来。
- ▶ 所以我们如果过多地在测试集上评价，则会高估测试集上面的效果。



- ▶ 如果有极多的数据，那么上述方法还一般可行；
- ▶ 但是通常情况下，我们可以得到的整体数据量是非常有限的；
- ▶ 这种情况下，如果还要用之前的方法会导致：
  - ▶ 训练集的数据量不够，导致精度不够；
  - ▶ 验证集的数据量不够，导致验证准确性不强；



- ▶ 一般来说，KFold 随机选取就可以；
- ▶ 但是在一些情况下，KFold 和 Test 集合可能即使在随机选取情况下，仍然会有很大的差别；
- ▶ 这种情况有可能有几种情况造成；
  - ▶ KFold 本身的随机性造成；
  - ▶ 训练（验证）集和测试集本身差异造成；
- ▶ 一般来说，我们希望的是，尽可能 KFold 结果和测试集保证一致；

- ▶ 一般来说，我们希望训练、验证和测试集都来自于一个分布；但是这种假设经常被打破；
- ▶ 比如说，对于时间序列来说，如果我们用 2019 年的经济数据来预测 2020 年经济数据，则大概率我们不会得到很好的结果。
- ▶ 这种问题没有通用的很好的解决方法，一般来说只有两种可能性：
  - ▶ 尽可能保证样本具有足够代表性：“北京样本收入估计全国” → “全国抽样估计全国”。
  - ▶ 尽可能保证模型的多样性：模型集成。

- ▶ 由于每个模型都具有一定的长处和短处，所以尽可能用多个模型的结果来进行预测；
- ▶ 最简单的模型集成：将一个数据级进行 KFold 之后进行直接采用预测概率的平均；
- ▶ **思考题：**传统方法建议使用 KFold 选择超参数，然后再使用同样的参数，对所有的训练集进行训练并预测 → 这样的训练有什么问题？

- ▶ 核心问题（之一）在于超参数的选择和观测数量有很大关系；
- ▶ 如果我们改变观测数量；实际上我们也改变了最佳的参数；
- ▶ 更糟糕的是，其实对于 GBDT 类模型，你不知道选取多少棵树作为最终模型；
- ▶ 此外，多模型（请注意，KFold 交叉验证在比赛中常常称之为单模型，但是实际是多个模型），往往会比单模型更稳定；

- ▶ 在（传统的）数据科学竞赛当中，通常会采用更复杂的模型平均策略；
- ▶ 很常见的一种策略是，以一个模型为基础，每次增加一个（通常与之前模型数学形式有所不同的模型）；
- ▶ 这样做的好处，可以不扔掉之前模型的效果；
- ▶ 关于模型的复杂集成，我们将会在两章后进行介绍。



- ▶ AdaBoosting(Chengsheng, Huacheng, and Bing 2017) 的核心思想是训练两个模型，得到一个模型的预测之后，对于该模型预测较差的部分我们应该对之增加权重，而已经较好的部分则不需要特别的处理；
- ▶ 这种方法（和类似衍生方法在 2010 年左右十分火热）；
- ▶ 目前该方法基本已经被 GBDT 及相关模型所取代；
- ▶ 但是，GBDT 类模型 + 神经网络 + AdaBoost 在很多实践当中效果很好；

- ▶ 三种方式：
  - ▶ 唯一的一个模型；
  - ▶ 同样的模型的 KFold（在竞赛中也叫单模型）；
  - ▶ 多个模型的复杂集成；
- ▶ 一般来说，需要考虑的是算力的要求；通常来讲，在算力可以达到的时候，尽可能不要用单模型；传统的一些人认为单模型（尤其是线性回归和逻辑回归）比复杂模型更稳定，因为表现形式简单，这主要原因是因为在实践中，往往他们对比的不是做过 KFold 之后的模型。
- ▶ 其他所谓可解释性的问题往往并不一定是真正限制模型应用的，尤其是 SHAP 值出现之后。相对来说，很多时候这里出现的是“但求无过，不求有功”的心态。

# 是否选好变量就可以用很简单的模型

- ▶ 一些所谓业务专家吹嘘自己因为懂业务，所以只要自己随便懵出来的模型，就可以比这些复杂的方法预测性更好；
- ▶ 这是不可能的；没有任何一个业务专家可以在 Kaggle 上一次就凭借自己经验，用逻辑回归得到 Kaggle 第一名；甚至我怀疑有任何业务专家能够一次性根据自己经验进入到前 90%。
- ▶ 核心问题，数据和业务真实之间的关系是非常复杂的。大部分专家的所谓经验只是对于非常小的样本、非常少的变量和非常粗略的关系，这对于挖掘出来数据真实作用其实是毫无帮助的。
- ▶ 例如违约预测问题，可以问专家：
  - ▶ 是否可以告诉我从收入区间没 50 元钱之间违约概率有什么不同；
  - ▶ 给定某用户三年内所有微信、支付宝银行卡等转账数据，是否可以告诉我哪些和收入有交叉效应；
- ▶ 结论：对于号称自己有业务专家支持的，应该引导做公平 POC。大部分专家预测结果，远远不如实习生乱做出来的模型。

- ▶ 正如前文所说：理论上，好的模型的可解释性和预测精度往往应该是相辅相成的；
- ▶ 但是实际上，对于常见的所谓可解释的模型，其结果往往是互相矛盾的；
- ▶ 原因在于常见的可解释模型，数学形式较为简单，但是带来的问题是，这些模型所得到的估计偏差较大；所以虽然可“解释”，但是解释出来的结果是错的；
- ▶ 在 SHAP 值出现之后，我们发现更复杂的模型如果提取出真实的表现，实际比逻辑回归和线性回归业务解释更直观，见[该文章](#)。

- ▶ 鲁棒性指模型在不同（来源）的数据集上表现应该尽可能一致；
- ▶ 鲁棒性从原则上来说是没有办法根本解决的；
- ▶ 但是有一些方法可以提升模型的鲁棒性：
  - ▶ 采用多种形式的模型进行平均；
  - ▶ 小心进行数据预处理（删除掉过小的类别，对一些取值范畴进行限制等）。

## 1 模型评估策略

## 2 树模型和提升

■ 原理 ■ 代码实现及重要参数

## 3 参考文献



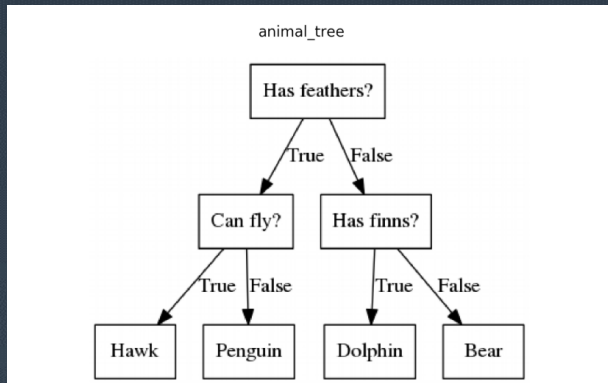
## 1 模型评估策略

## 2 树模型和提升

- 原理 ■ 代码实现及重要参数

## 3 参考文献





- ▶ 很好 (?) 的捕捉非线性效应和交叉效应;
- ▶ 可解释性 (?);
- ▶ 贪婪算法导致收敛保证和计算快捷。

- ▶ 准确率很差，并且难以提高 → 主要原因在于模型表现力不够；
- ▶ 树的结构十分随机；
- ▶ 各种调整参数的方法对于决策树用处都不大；
- ▶ 一般仅仅用于变量的离散化；实际上效果也不如其他模型（例如 LightGBM）。

- ▶ 核心思想：随机抽取部分变量和/或观测，分别拟合决策树；
- ▶ 最终预测结果由投票决定；
- ▶ 一般只支持离散或连续的预测值；
- ▶ 通常为了保证速度，采用对  $X$  分箱后再寻找合适的节点（据称可以防止过拟合？）；
- ▶ 一个重要的变种为 ExtraTrees(Geurts, Ernst, and Wehenkel 2006)。核心思路在于随机抽取分割点，然后再从分割点选取合适的。

- ▶ 优点：表现力远远强于单颗决策树；可以很容易实现并行；
- ▶ 缺点：树和树之间没有关联；一般来说对于一般的情况定义一般的损失函数不是十分容易；

- ▶ 相对于决策树来说，GBDT(Friedman 2001) 是一系列的针对一般建模情况的算法；
- ▶ 核心思想，根据上一轮的运算结果进行对模型进行补充。

1.  $F_0(\mathbf{x}) = \arg \min_{\rho} \sum_{i=1}^N L(y_i, \rho)$
2. For  $m = 1$  to  $M$  do:
3.  $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, N$
4.  $\mathbf{a}_m = \arg \min_{\mathbf{a}, \beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5.  $\rho_m = \arg \min_{\rho} \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6.  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$



- ▶ 虽然 GBDT 文章出现较早，但是实际上刚开始并没有那么流行，这多少和 GBDT 消耗算力较大（在当时看来）有关；
- ▶ GDBT 主要引入到公众视角是使用 GBDT+LR 的方式进行点击率预估的时候；
- ▶ 目前就 GBDT 基础之上，主要有四个比较著名的提升，即 XGBoost(Chen and Guestrin 2016)，LightGBM(Ke et al. 2017)，CatBoost(Prokhorenkova et al. 2017) 和 NODE(Popov, Morozov, and Babenko 2019)。对于前三者我们将会在本章进行讲解，对于 NODE 我们将会神经网络建模当中讲解。

见[XgBoost 官方文档](#)

见Ke et al. (2017)。

见Prokhorenkova et al. (2017)

## 1 模型评估策略

## 2 树模型和提升

■ 原理 ■ 代码实现及重要参数

## 3 参考文献

- ▶ XgBoost, LightGBM 和 CatBoost 在代码实现上极其相像, 所以只需要看懂一个基本就可以看懂这几个包;
- ▶ 此外, SKLearn 当中的随机森林和 ExtraTrees 的实现并不好, 不如采用 LightGBM;
- ▶ 我们以 LightGBM 为例来说明整体的方法。

## 重要的参数类：涉及到树的复杂度的

- ▶ 在 LightGBM 当中，最主要的参数为 `num_leaves`（叶子个数）；
- ▶ 除此之外，一些方法还提供树的深度作为控制；
- ▶ 一般来说用叶子个数的控制要比深度控制更细腻，也更有助于对于优质变量的选择；
- ▶ **注意：**叶子数量跟学习率高度相关；一般来说，当树更复杂的时候，学习率要尽量减小；尽可能控制在学习率可以增加较合理范畴达到最好；



## 重要的参数类：涉及到数据筛选的随机性

- ▶ 每一次随机选取多少变量或观测拟合；
- ▶ 在 LightGBM 当中，这个参数称之为 `bagging_fraction` 以及 `feature_fraction`；
- ▶ 通常来说，我会从 0.8 (?) 开始；

- ▶ DART(Vinayak and Gilad-Bachrach 2015) 是一种模仿 dropout 的方式构建的防止 (?) 过拟合的方法;
- ▶ 核心思想: 在每次进行 boosting 的时候都将前一轮的一些树随机扔掉;
- ▶ 一般来说会极大的减慢模型拟合的过程; 但是在一些情况下可以得到更好的结果。

- ▶ 类似于 LightGBM 的库有上百个不同的参数；这些参数的效果很难通过经验总结出来；
- ▶ 在一些官方网站当中，有[关于调参的建议](#)
- ▶ 在一些情况下，则只能依靠一些经验积累；注意任何人的经验积累都可能是有问题的。

- ▶ 在参数量如此之大的情况下，依靠遍历的方式去寻找最优的参数显然是有问题的；
- ▶ 一般来说，我个人会将参数寻找放到三个阶段：
  - ▶ 随机探索阶段；
  - ▶ 顺序搜索阶段；
  - ▶ 贝叶斯优化阶段；

- ▶ 在我们最开始做任何超参数选择时候，一定要考虑我们的选择是否是公平客观的；
- ▶ 如果我们对于不同的变量的选择，采用不同程度的调参（一个精调另外一个细调），则得到的结果是不公平的，这并不能告诉我们实际上衍生变量的好坏；
- ▶ 同样道理，对于不同数据集我们也不应该用不公平的比较法。

- ▶ 随机探索阶段，主要是收集尽可能多的关于模型运行的信息，并从中找到能够提示模型性质的线索；
- ▶ 核心，尽可能多记录不同的 metric；
- ▶ 可以尝试一些极端的方法；
- ▶ 可以检查变量的重要性；

- ▶ 这部分是在我们已经确定了大概的合理参数范围的情况下，开始按照参数的重要性 (?) 进行搜索；
- ▶ 例如：首先调整学习率 + 深度；再次调整 `bagging_fraction` 和 `feature_fraction`；
- ▶ 目的是为了找到一定的合理的范畴，减少之后搜索的负责度









- ▶ 整体来讲，HyperOpt 是基于贝叶斯方法的一套优化方法；
- ▶ 建议：
  - ▶ 在已经调过的最有参数周围进行调参，而未调过的尽可能选择较大的范畴；
  - ▶ 采用多次初始化比采用一次初始化跑多次要效果更好；
- ▶ HyperOpt 花费时间极长，尽量考虑是否一定需要采用这种方式。



- ▶ 通常来说，调参至少要采用 KFold 选取并在测试集上进行验证；
- ▶ 如果采用更复杂的模型集成方式，则最好在构建 pipeline 之后，每晚下班后自动调整并在第二天早上查看最终结果。

1 模型评估策略

2 树模型和提升

3 参考文献

-  Chen, Tianqi and Carlos Guestrin (2016). "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794.
-  Chengsheng, Tu, Liu Huacheng, and Xu Bing (2017). "AdaBoost typical Algorithm and its application research". In: *MATEC Web of Conferences*. Vol. 139. EDP Sciences, p. 00222.
-  Friedman, Jerome H (2001). "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics*, pp. 1189–1232.
-  Geurts, Pierre, Damien Ernst, and Louis Wehenkel (2006). "Extremely randomized trees". In: *Machine learning* 63.1, pp. 3–42.
-  Ke, Guolin et al. (2017). "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in neural information processing systems* 30, pp. 3146–3154.
-  Popov, Sergei, Stanislav Morozov, and Artem Babenko (2019). "Neural oblivious decision ensembles for deep learning on tabular data". In: *arXiv preprint arXiv:1909.06312*.

-  Prokhorenkova, Liudmila et al. (2017). “CatBoost: unbiased boosting with categorical features”. In: *arXiv preprint arXiv:1706.09516*.
-  Vinayak, Rashmi Korlakai and Ran Gilad-Bachrach (2015). “Dart: Dropouts meet multiple additive regression trees”. In: *Artificial Intelligence and Statistics*. PMLR, pp. 489–497.