

# 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

二〇二一年二月六日

1 模型评估策略

2 树模型和提升

3 线性模型

4 KNN 和 t-SNE

5 模型评估指标

6 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 线性模型
- 4 KNN 和 t-SNE
- 5 模型评估指标
- 6 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 线性模型
- 4 KNN 和 t-SNE
- 5 模型评估指标
- 6 参考文献

- ▶ 在第三章开头，我们讲过模型评估一旦出了问题，我们将会难以判断算法中的细节是否对整体准确性有帮助
- ▶ 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？)
- ▶ 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也近似的达到了效果
- ▶ 一个更极端的例子：如果加入测试集当中的只有正负样本，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
- ▶ 所以我们如果过多的在测试集上评价，则会高估测试集的效果



- ▶ 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就会导致：
  - ▶ 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，k-fold 随机选取就可以
- ▶ 但是在一些情况下，k-fold 和 Test 集合在随机选取的情况下，仍然可能有很大的差别
- ▶ 这种情况有可能是几种情况造成的：
  - ▶ k-fold 本身的随机性
  - ▶ 训练（验证）集和测试集本身的差异
- ▶ 一般来说我们希望的是：尽可能保证 k-fold 结果和测试集一致

- ▶ 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，则大概率不会得到很好的结果
- ▶ 这种问题没有通用的良好解决方法，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：将一个数据集  $k$ -fold 之后直接采用预测概率的平均
- ▶ 思考题：传统方法建议使用  $k$ -fold 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，实际上也改变了最佳的参数
- ▶ 更糟糕的是，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，k-fold 交叉验证在比赛中常常被称为单模型，但是它实际上是多模型），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（通常都是数学形式不同的模型）
- ▶ 这样做的好处是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



- ▶ AdaBoost(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
  - ▶ 同样模型的 k-fold（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
- ▶ 传统人士认为单模型（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，这主要原因是在实践中，他们往往对比的不是做过 k-fold 之后的模型
- ▶ 其他所谓可解释性的问题往往不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 上一次就凭借自己的经验，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的经验进入到前 90%
- ▶ 核心问题在于，数据和业务真实之间的关系是非常复杂的。大部分专家的经验只对非常小的样本、非常少的变量和非常粗略的关系管用，但对于挖掘出来的数据的作用其实是很小的
- ▶ 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每 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效果好

- ▶ 正如前文所说：理论上好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，我们发现如果从更复杂的模型中提取出真实的表现，实际比逻辑回归和线性回归对业务的解释更直观，见[该文章](#)

- ▶ 鲁棒性是指模型在不同（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 线性模型

## 4 KNN 和 t-SNE

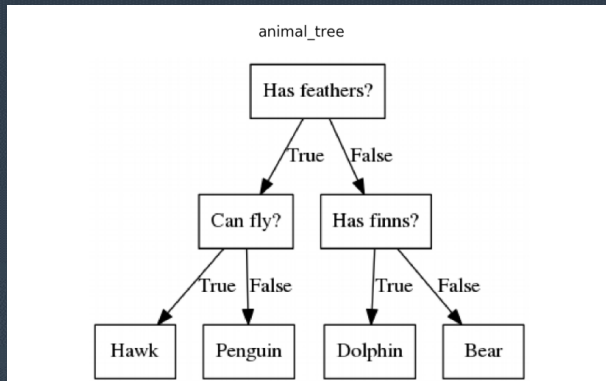
## 5 模型评估指标

## 6 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 线性模型
- 4 KNN 和 t-SNE
- 5 模型评估指标
- 6 参考文献





- ▶ 很好 (?) 的捕捉非线性效应和交叉效应
- ▶ 可解释性 (?)
- ▶ 贪婪算法使得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的用处都不大
- ▶ 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（据说可以防止过拟合？）
- ▶ 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，且可以很容易实现并行
- ▶ 缺点：树和树之间没有关联，通常根据一般情况定义一般的损失函数不是十分容易的

- ▶ 相对于决策树来说，GBDT(Friedman 2001) 是一系列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 的文章出现较早，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 引入到公众视角是从使用 GBDT+LR 的方式做点击率预估开始的
- ▶ 目前在 GBDT 的基础上，主要有四个比较著名的提升：
  - ▶ XGBoost(Chen and Guestrin 2016)
  - ▶ LightGBM(Ke et al. 2017)
  - ▶ CatBoost(Prokhorenkova et al. 2017)
  - ▶ NODE(Popov, Morozov, and Babenko 2019)
  - ▶ 对于前三者我们将会在本章讲解，NODE 将会在神经网络建模中讲解

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 线性模型

## 4 KNN 和 t-SNE

## 5 模型评估指标

## 6 参考文献

- ▶ XGBoost、LightGBM 和 CatBoost 在代码实现上极其相像，所以只需要看懂其中一个包，就可以基本看懂这几个包
- ▶ 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

- ▶ 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控制要比用深度控制更细腻，也更有助于选择优质变量
- ▶ 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来拟合
- ▶ 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 方式来构建模型以防止 (?) 过拟合的方法
- ▶ 核心思想: 在每次 boosting 的时候都将前一轮的一些树随机扔掉
- ▶ 一般来说, 这会极大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，但任何人的经验积累都是有问题的

- ▶ 当参数量非常大的时候，依靠遍历的方法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在三个阶段里：
  - ▶ 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的时候，一定要考虑选择是否公平客观
- ▶ 如果我们选择不同的变量，采用不同程度的调参（一个精调另外一个细调），那么得到的结果就会是不公平的，这并不能告诉我们衍生变量的好坏情况
- ▶ 同样道理，对于不同的数据集，我们也不应该用不公平的比较法

- ▶ 随机探索阶段：尽可能多的收集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

- ▶ 在确定了大致合理的参数范围后，我们可以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，再调整 `bagging_fraction` 和 `feature_fraction`
- ▶ 目的：为了找到一个大致合理的范畴，减少之后搜索的负责度



- ▶ 整体来说，Hyperopt 是基于贝叶斯方法的一套优化方法
- ▶ 建议：
  - ▶ 在已经调过的最优参数周围进行调参，而未调过的参数则尽可能选择较大的范畴
  - ▶ 采用多次初始化要比采用一次初始化跑多次的效果更好
- ▶ 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 k-fold 选取并在测试集上验证
- ▶ 如果要采用更复杂的模型集成方式，则最好构建 pipeline，并在每晚下班后自动调整接着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 线性模型

4 KNN 和 t-SNE

5 模型评估指标

6 参考文献

- ▶ 我们这里所说的线性模型指的是有  $x_i^t \beta$  这种形式的模型，其中  $x_i$  为第  $i$  个样本的输入，而  $\beta$  则代表估计参数
- ▶ 线性模型的表达形式要比树模型复杂的多，所以相对来说，如果线性模型能找到合适的样本，则更容易保证在较小的模型中得到更好的效果
- ▶ 反过来，由于线性模型的拟合不是贪婪的拟合，所以用线性模型处理高维的数据是很容易过拟合的
- ▶ 由于线性模型的表达式和树模型的不同，所以线性模型的特征构造有很多技巧，这部分我们会在下一章中讲
- ▶ 此外，由于线性模型对于异常值非常敏感，需要重点关注，也建议进行标准化

- ▶ 线性回归针对的是目标为连续且可以取全部实数值的情况
- ▶ 在一些情况下，可以不用这样取值，比如收入一般不会是负 100 万的
- ▶ 在这种情况下，我们要对模型做一些变化，例如取  $\log$
- ▶ 逻辑回归也是类似的情况
- ▶ 此外还有一类模型叫做 GLM，可见 Andrew NG 讲义

见 Andrew NG 讲义

1 模型评估策略

2 树模型和提升

3 线性模型

4 KNN 和 t-SNE

5 模型评估指标

6 参考文献



- ▶ KNN 的思路非常简单：假如我们需要处理的是分类问题，那么只需要找到距离最近的几个观测，然后看他们的分布情况即可
- ▶ KNN 最大的问题：计算缓慢，而且推测也很慢
- ▶ KNN 一般很少单独使用，但是 KNN 的使用技巧中有一个很有用的方法

- ▶ 两种最常见的降维方法: 主成分方法和 t-SNE(Maaten and Hinton 2008)
- ▶ 主成分法的主要目的是找到线性独立的、可以解释  $X$  变量的降维变量
- ▶ t-SNE 则更为复杂, 需要通过梯度迭代来解决, 并且效率比较低

- ▶ 在一些情况下，使用 tSNE 降维后，将 KNN 的结果（针对 tSNE）加入到其中；
- ▶ 这在模型平均中是一个很重要的方法

1 模型评估策略

2 树模型和提升

3 线性模型

4 KNN 和 t-SNE

5 模型评估指标

6 参考文献

- ▶ 即使是分类和回归问题，也有很多指标，见[sklearn 评估指标表格](#)
- ▶ 一般来说，最简单的指标最适合评估模型的效果（例如准确度）
- ▶ 更复杂的指标，容易帮我们找到模型的一些问题

- ▶ 请注意，ROC 是一个很危险的指标
- ▶ ROC 的基本思路：随着阈值的不同，正负样本区分不同，所以说我们应该考察在不同截断阈值下面的表现
- ▶ 问题：大部分模型并不能对这个违约概率进行预测，在这种情况下，可以做到 ROC 很高但是实际预测效果极差
- ▶ 一般方法：使用逻辑回归作为 Stacking 的最后一层，Stacking 策略我们最后会讲解

1 模型评估策略

2 树模型和提升

3 线性模型

4 KNN 和 t-SNE

5 模型评估指标

6 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.



Maaten, Laurens Van der and Geoffrey Hinton (2008). “Visualizing data using t-SNE.”. In: *Journal of machine learning research* 9.11.

-  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.