

2020

Machine Learning

Fourth Assignment

姓名	班级	学号
王成航	统计 71	2176122248
张申铎	统计 71	2176112379
王泽昊	统计 71	2176112782

初涉集成学习

目录

1	前言	1
2	集成学习介绍	1
2.1	集成学习的方法	1
3	模型介绍	3
3.1	AdaBoost 步骤	3
3.2	模型训练	4
4	训练结果	4
4.1	初次结果	4
4.1.1	决策树弱学习机	4
4.2	适当的改进	5
4.2.1	学习机数量提升	5
4.2.2	支撑向量机弱学习机	6

1 前言

集成学习是指使用多种兼容的学习算法或模型来执行单个任务的技术,目的是为了得到最佳的预测表现. 集成学习的主要方法可归类为三大类: 堆叠 (Stacking)、提升 (Boosting) 和装袋. 其中最流行的方法包括随机森林、梯度提升、AdaBoost、梯度提升决策树 (GBDT) 和 XGBoost.

1979 年, Dasarathy 提出集成系统 (ensemble system) 的思想, 并使用线性分类器和最近邻居 (NN) 分类器组成的复合系统作为其组成部分的例子来说明这些概念. 1988 年 Kearns 和 1989 年 Valiant 分别提出的弱学习的概念, 引发了关于 “能否用一组弱学习器创建一个强学习器” 这一问题的讨论. 1990 年 Schapire 对这一问题给出了肯定的回答, 并开发出 boosting. 并在几年之后和 Freund 提出了 AdaBoost, 该算法被广泛使用, 他们还凭借 Adaboost 获得了计算机领域富有盛名的哥德尔奖. 1996 年, Breiman 开发出 Bagging 预测器, 并对其原理和训练进行了详细描述.

近年来, boosting 算法在数据科学或机器学习竞赛中得到了广泛的普及, 这些比赛的大多数获胜者都使用 boosting 算法来实现高精度. 这些数据科学竞赛为学习、探索和为各种商业和政府问题的解决方案提供了全球性的平台. Boosting 算法结合了多个低精度模型来创建一个高精度模型. 它可以用于各种领域, 例如信贷、保险、市场营销和销售.

2 集成学习介绍

集成是一个组合模型, 它结合了一系列性能低下的分类器, 目的是创建一个性能更好的分类器. 集成的方法提供比单个或基本分类器更高的准确性, 也可以说集成学习方法是将几种机器学习方法组合到单个预测模型中以提高性能的元算法. 集成方法可以使用 bagging 方法减少方差, 使用 boosting 方法减少偏差, 或者使用 stacking 方法来改善预测.

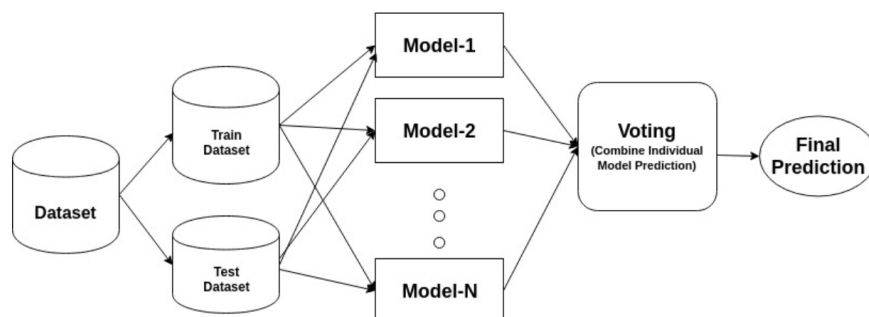


图 1: 三种机器学习方法.

2.1 集成学习的方法

一般在集成学习中, 我们会用到三种方法.

Bagging 算法

Bagging 算法代表着引导聚集 (Bootstrap aggregating), 它为了减少估计的方差组合了多个学习机. 例如, 随机森林训练了 M 个决策树, 那便可以在数据的不同随机子集上训练 M 个不同的树, 并对最终预测进行投票. Bagging 算法也可与其他分类、回归算法结合, 提高其准确率、稳定性的同时, 通过降低结果的方差, 避免过拟合的发生.

Boosting 算法

Boosting 算法是一种能够将弱学习器转化为强学习器, 从而提升各种学习算法的方法. 理论上, boosting 可以显著减小弱学习器的偏差, 这些弱学习器的效果只是稍微优于随机猜测, 比如小决策树——数据加权模型. 然后 Boosting 在运行时将更多的权重赋值给早期训练中错误最多的数据集, 通过结合加权多数投票 (分类) 或加权求和 (回归) 以产生最终预测. Boosting 的每个模型都是单独运行, 每个模型决定下一个模型要关注的特征, 最后在不偏向任何模型的前提下聚合输出结果.

Stacking 算法

Stacking 算法是一种集成学习技术, 用于最小化一个或多个泛化器的泛化误差率的方法. 它通过推导泛化器相对于所提供的学习集的偏差来发挥其作用. 这个推导的过程包括: 在第二层中将第一层的原始泛化器对部分学习集的猜测进行泛化, 以及尝试对学习集的剩余部分进行猜测, 并且输出正确的结果. 当与多个泛化器一起使用时, 堆叠泛化可以被看作是一个交叉验证的复杂版本, 利用交叉验证更为复杂的策略来组合各个泛化器. 当与单个泛化器一起使用时, 堆叠泛化是一种用于估计 (然后纠正) 泛化器的错误的方法, 该泛化器已经在特定学习集上进行了训练并被询问了特定问题.

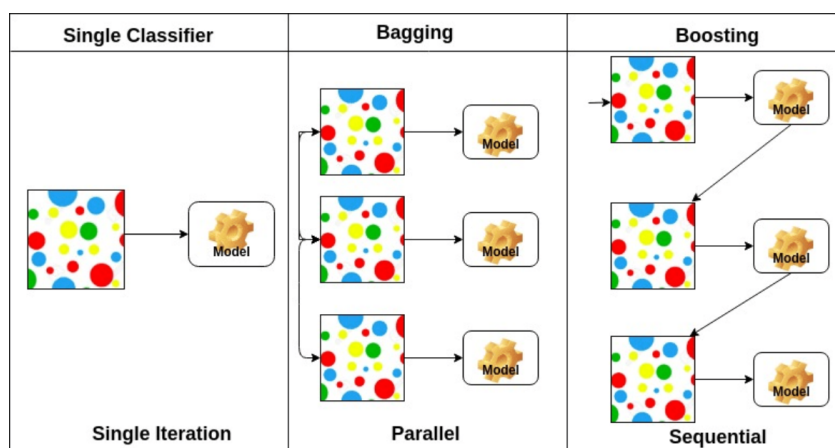


图 2: 基于 Stacking 算法的分类.

以基学习器的分布为基础, stacking 方法可以分为两组: 在平行集成方法中, 基学习器平行的产生, 例如随机森林. 在顺序集成方法中, 基学习器是顺序生成的, 例如 AdaBoost;

以基学习器的类型也可以分为两组：同构集成方法在每次迭代中使用相同类型的基学习器，异构集成方法在每次迭代中使用不同类型的基学习器。

3 模型介绍

AdaBoost 是一种迭代集成方法。AdaBoost 分类器通过组合多个性能不佳的分类器来构建一个强分类器。AdaBoost 背后的基本概念是设置分类器的权重，并在每次迭代中训练数据样本，从而确保对异常观测值的准确预测。任何机器学习算法在接受了训练集上的权重后，都可以被当作基学习器。AdaBoost 应该满足两个条件：

1. 分类器应在各种加权的训练示例上进行交互训练。
2. 在每次迭代中，它都试图通过最小化训练误差为这些示例提供出色的拟合。

3.1 AdaBoost 步骤

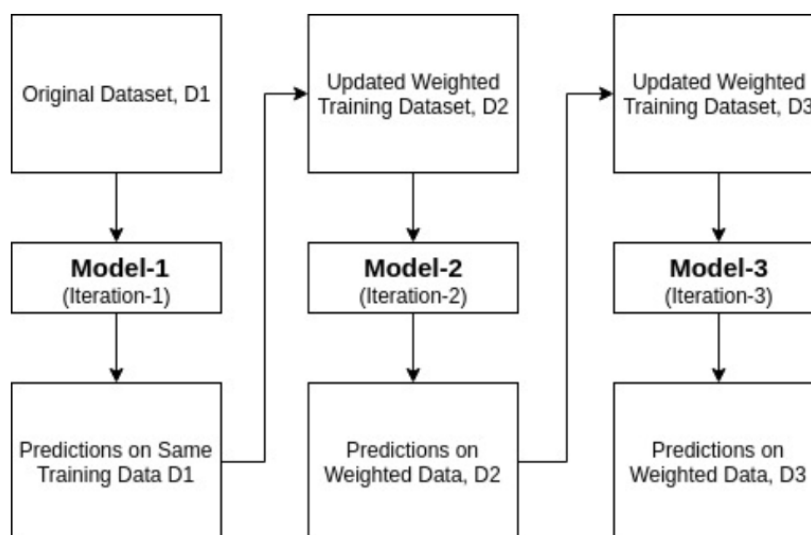


图 3: AdaBoost 步骤示意图。

1. Adaboost 先随机选择一个训练子集。
2. 基于上次训练的准确预测来选择训练集来迭代训练 AdaBoost 模型。
3. 将较高的权重分配给错误的分类观测值，以便在下一次迭代中这些观测值将获得较高的被分类概率。
4. 根据分类器的准确性在每次迭代中将权重分配给经过训练的分类器。分类器越准确，权重越高。
5. 反复进行此过程，直到完整的训练数据正确无误或达到指定的最大估计量为止。
6. 最后，让所有构建的所有学习算法进行“投票”，获取分类结果。

3.2 模型训练

我们使用了 sklearn 中 ensemble 类里的 AdaBoostClassifier 类. 数据集选用了 UCI 数据集中的 IRIS 数据集. IRIS 数据集是一个花卉分类数据集, 其中给出了标记好的三种花卉 (Iris Setosa-山鸢尾, Iris Versicolour-杂色鸢尾, 以及 Iris Virginica-维吉尼亚鸢尾, 每类包含 50 个样本), 以及其所对应的四个特征 (花萼长度, 花萼宽度, 花瓣长度, 花瓣宽度). 我们的任务是学习一个函数, 使得其通过输入这三种花其中的某一种的一个植株的这四个参数, 能够判断出这个植株是哪一种花. 这是一个典型的分类问题, IRIS 数据集也是应用广泛的数据分类数据集. 我们首先将数据集分为训练集与测试集两部分进行训练.

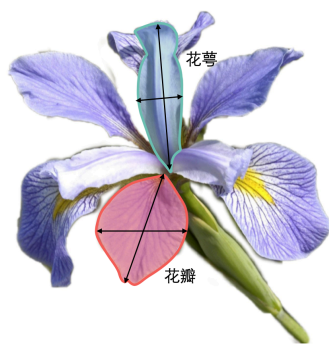


图 4: 弗吉尼亚鸢尾的花萼与花瓣示意图

4 训练结果

4.1 初次结果

4.1.1 决策树弱学习机

我们的模型采用了 50 个决策树作为弱学习机, 采用了默认的学习率 Sklearn 的 AdaboostClassifier 能够全自动地帮我们完成优化损失函数. 我们在测试集上获得了 87.6% 的准确率. 这并不是一个多么令人可喜的结果. 我们希望了解我们的模型哪里出了问题. 于是我们尝试了可视化模型.

我们尝试可视化了我们模型的分类边界. 不过由于四维的数据可视化起来比较困难. 我们又只在其中的两个特征 (两个长度) 上使用相同的方法与参数训练出来了一个学习机, 我们觉得这样的简化版学习机可以很好地解释学习机的限制以及学习的过程中发生了什么. 而通过可视化我们的确发现了我们模型中致命的问题.

从图上也可以看出这两个特征可以很好的将 A 类山鸢尾与 B 类杂色鸢尾、C 类弗吉尼亚鸢尾分开, 但是对于 B、C 类之间的分类做的很差. 除了这两个特征我们需要更多的信息去区分 B、C 类. 我们推测这个原因并不是我们的学习机的问题, 通过另外的两维特征我们应该可以在一定程度上解决这个问题. 因为其更像是某个高维空间的散点在这个低维空间上的投影.

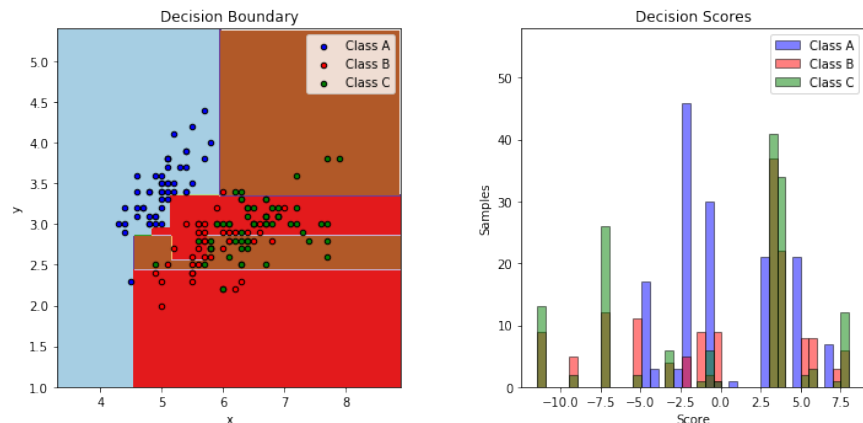


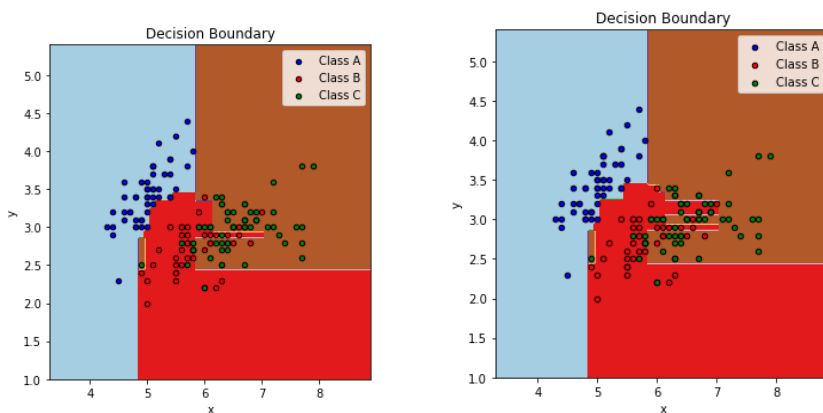
图 5: 对于采用决策树作为弱学习机的分类器在样本的两个特征上面训练出来的分类边界的可视化, 可以看到其保持了原弱学习机的分类特点, 可以说是一个超级纵横学习机。

另外一个问题是我们发现我们的弱学习机的集成依旧保持着弱学习机本来的分类边界的特点, 是一个超级纵横学习机, 本质上没有突破纵横学习机的限制. 分类的边界属于横平竖直. 但是从图上我们也可以很清楚的看出这样的样本分类并不是这样横平竖直的学习机能够很好表示的.

4.2 适当的改进

4.2.1 学习机数量提升

一个最简单的改进的思路就是去增加弱学习机的数量, 通过更大的学习规模去对函数做更好的拟合. 那么我们在增加学习机的规模的时候并没有任何的效果上的提升. 甚至有所下降. 通过简化版学习机的分类边界可视化, 我们不难观察到这样的增加弱学习机的数量并没有对我们的分类边界产生任何实质性的影响. 甚至都没有增加分类边界的曲折程度, 只是边的位置有了一定调整. 所以我们基本可以认为此时模型的限制基本完全来源于弱学习机的分类特点.



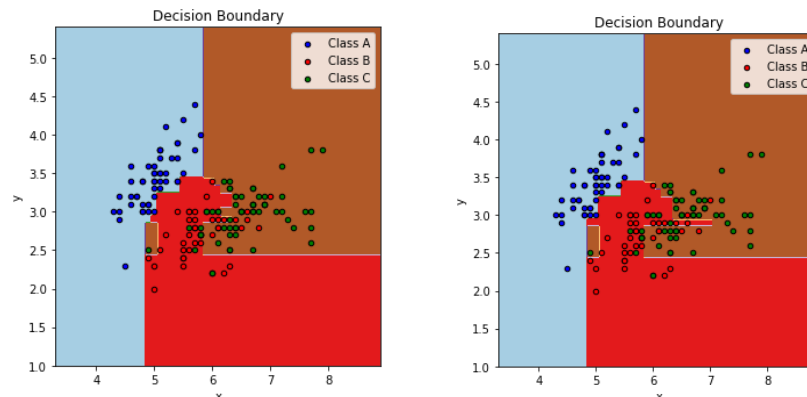


图 6: 我们的弱学习机数量从 50 增加到了 100、150、200. 可以看到其并没有任何的明显的质变, 分类边界并没有变细腻, 只是边界的位置换了地方.

4.2.2 支撑向量机弱学习机

根据我们上面的分析, 结合 Boosting 过程的特性, 另外一个显然的想法就是尝试不同的弱学习机去提升我们的分类准确率. 我们又尝试了支撑向量机作为我们的弱学习机去进行 Adaboosting 学习. `sklearn.svm` 类中的 `svc` 类给我们提供这样现成的弱学习机. 在支撑向量机里, 我们采用了线性的核函数. 通过使用相同的参数, 我们发现这样的弱学习机的选取也可以极大程度上的影响我们的分类准确率. 我们的分类准确率从 87.6% 提升到了 94.1%. 几乎正确分对了绝大多数的测试集里的数据.

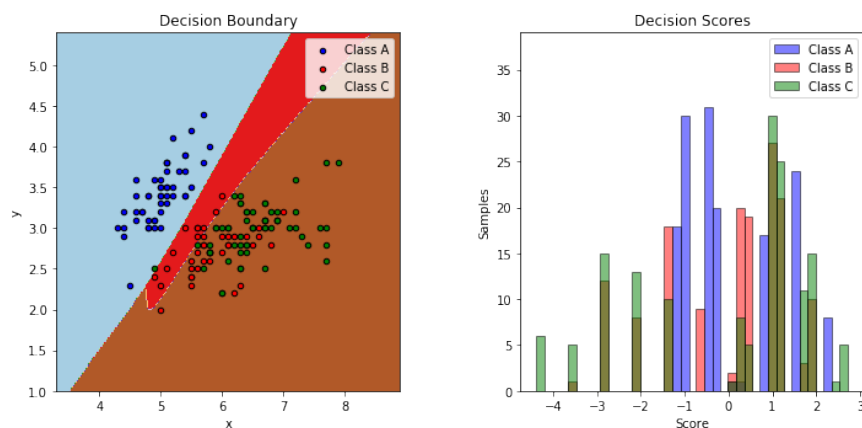


图 7: 采用了 `svm` 作为弱学习机的 `adaboost` 算法. 能够明显地以更简单的方式更好地捕捉类与类之间的边界.

我们依旧对于两个特征又训练了一个模型用于可视化, 采用了同样的可视化方案, 我们发现了采用 `svm` 作为弱学习机的 `adaboost` 学习机与原来的采用决策树的有了质的改变.

并且可以从图上观察出我们的分类器确实能够相比采用决策树时更好地捕捉到类边界. 相比于决策树的边界而言, `SVM` 作为弱学习机习得的边界不再是一条直线, 故显著增加了分类准确性. 我们还可以观察到在 A 类与 B、C 类的分离时几乎没有错分, 但是我们与上次一样依旧需要更多的信息去区分 B、C 类. 并且但我们在此时提高弱学习机数量的时候, 我们的分类边界也有了明显的变化.

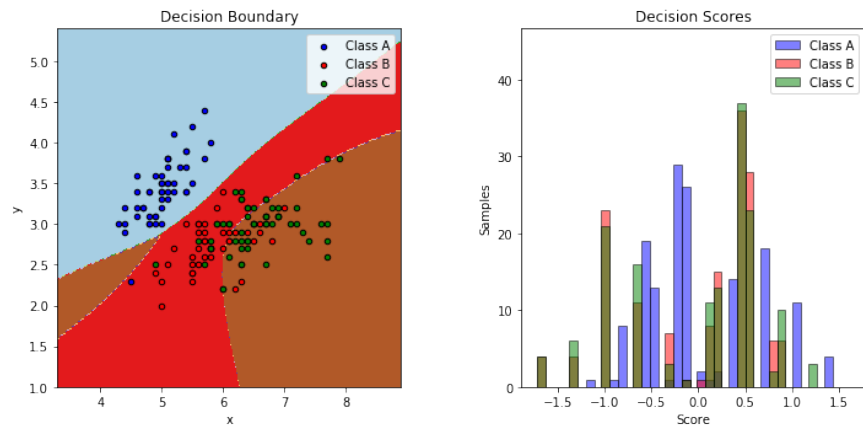


图 8: 弱 SVM 分类器的数量从 50 增加到 100, 我们的二维特征上训练出来的学习机的分类性能得到了显著的改善, 能够进一步捕捉到数据的边界特征.

其能够进一步地捕捉到类与类之间的边界. 我们在全样本四个特征上面的学习机的准确率也提升了两个百分点左右达到了 95.9%. 但是当我们再一次提高我们的弱学习机数量时, 我们就再也得不到任何的性能提升了, 二维特征训练出的学习机可视化的结果也保持了基本不变.

参考文献

- [1] 周志华. 机器学习. 清华大学出版社, 2016.
- [2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.
- [3] Iris data set - uci machine learning repository. <https://archive.ics.uci.edu/ml/datasets/Iris>, 1988.
- [4] 集成学习. <https://www.jiqizhixin.com/graph/technologies/29722de0-8501-4b01-9b73-189141b9eefd>.
- [5] Adaboost classifier in python. <https://www.datacamp.com/community/tutorials/adaboost-classifier-python>.