《机器学习》第一次作业

1. 试析随机森林为何⽐决策树Bagging 集成的训练速度更快。

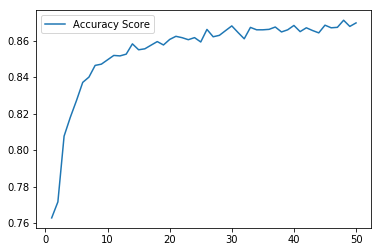
Bagging 通过有放回地抽样来增加训练集的多样性，从而提升模型的泛化性能。而随机森林则在以决策树为基学习器的Bagging集成的基础上，进一步在决策树的训练过程中引入了随机属性选择，即在训练过程中，对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含k个属性的集合，然后再从这个子集中选择一个最优的属性进行划分。对于决策树Bagging来说，它需要对结点的所有属性进行考察，而随机森林只需要考察一个属性子集，因此随机森林的训练速度比决策树Bagging集成的训练速度更快，并且由于随机森林基学习器的多样性不仅来自样本扰动，还来自属性扰动，因此泛化性能较Bagging集成会更强一些。

针对这一问题，我们通过代码实现来进行验证。当随机森林和决策树Bagging集成的决策树数量为10且并行度为8时，可以看到随机森林的训练和预测速度要明显快于决策树Bagging集成，但是随机森林的预测的正确率并没有决策树Bagging集成高，这可能是由于数据集的特征较少，随机森林的泛化性上的优势没有得以体现。

表1 随机森林和决策树Bagging在Chess(King-Rook vs. King)上的表现

|  |  |  |
| --- | --- | --- |
|  | 正确率 | 训练和预测时间 |
| 随机森林 | 77.67% | 325ms |
| 决策树Bagging | 85.26% | 3.99s |

接下来我们将迭代次数从1遍历至50，探究其训练时间和正确率的随着迭代次数的变化趋势。可以看到，两个算法随着迭代次数的增加，其正确率都逐渐上升，在30棵树左右趋于平缓，而训练和测试所用的时间则与迭代次数成正比，即迭代次数越多，训练时间越长。



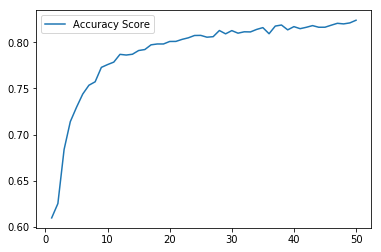


图1(a) 随机森林正确率曲线 图1(b) 决策树Bagging正确率曲线

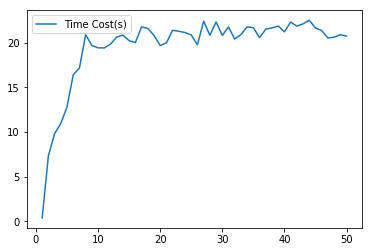
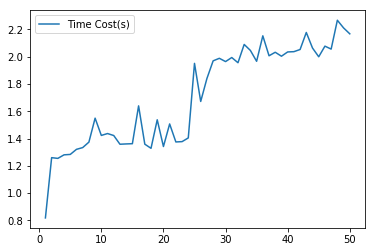


图1 (c) 随机森林训练和预测时间曲线 图1(d) 决策树Bagging集成训练和预测时

间曲线

可以看到，对于同一数据集来说，当迭代次数一定时，随机森林的训练速度是明显快于决策树Bagging集成的。

1. MultiBoosting 算法将AdaBoost 作为Bagging 的基学习器，Iterative Bagging 算法则是将Bagging 作为AdaBoost 的基学习器。试比较二者的优缺点。
2. MultiBoost

MultiBoost算法的主要思想是将AdaBoost作为Bagging的基学习器，AdaBoost能够有效降低偏差，而Bagging则能够更有效地降低方差，二者结合的MultiBoost就能有效地同时降低偏差和方差。

论文中的MultiBoost是基于AdaBoost的变种AdaBoost.M1以及Bagging的变种Wagging来实现的。AdaBoost.M1在传统AdaBoost方法上进行了改进，一是避免了权重下溢，二是在错误率大于0.5时，算法会继续训练基学习器而不是提早停止，能够有效提升预测正确率。相比于Bagging在抽样时所有样本都是相同权重，Wagging在抽样时给予样本集的样本不同权重。原版Wagging用的是高斯噪声，但是这会导致0权重的出现，因此论文中使用了泊松分布来解决这个问题。至少在迭代次数较少时，基于Wagging的MultiBoost比基于Bagging的MultiBoosting算法表现更好。

结合了AdaBoost.M1和Wagging的MultiBoost在更多情况下要比AdaBoost、Wagging和决策树Bagging的表现都要好，更加有效地降低误差和方差。然而其训练时间和预测时间也会大大增加。

下面是实验证明，基于论文我们实现了MutilBoost的分类算法，并在Chess (King-Rook vs. King)数据集上与AdaBoost和Bagging进行对比。可以看到，在同样以不限制决策树深度的决策树分类器为基学习器、迭代次数为100时，MultiBoost的准确率要稍高于AdaBoost和Bagging，而前者的训练和预测时间则明显长于后两者，这也可能是由于我们算法实现的效率过低导致的。

表2 MultiBoost、AdaBoost和Bagging在Chess(King-Rook vs. King)上的表现

|  |  |  |
| --- | --- | --- |
|  | 正确率 | 训练和测试时间 |
| MultiBoost | 87.09% | 6.86s |
| AdaBoost | 84.42% | 120ms |
| Bagging | 86.89% | 2.67s |

为了评判MultiBoost在降低方差和偏差上的效果，我们参考论文的附录B (A bias/variance estimation method)实现了分类算法的方差和偏差标准。简单来说，通过10轮三折交叉验证，然后求错误率的方差和平均值，即

这样就得到三个算法在Chess数据集上的方差和偏差表现。

表3 MultiBoost、AdaBoost和Bagging在Chess(King-Rook vs. King)的偏差和方差

|  |  |  |
| --- | --- | --- |
|  | 偏差(Bias) | 方差(Variance) |
| MultiBoost | 0.1517 | 2.2278×10-6 |
| AdaBoost | 0.1871 | 5.3467×10-6 |
| Bagging | 0.1537 | 5.3104×10-6 |

可以看到，当以不限制决策树深度的决策树分类器为基学习器、迭代次数为100时，MultiBoost在Chess数据集上能够比AdaBoost和Bagging算法更有效地降低预测结果的偏差和方差。

1. Iterated Bagging

Iterative Bagging算法对于Bias-Variance的影响：

泛化误差的构成分析 ：噪音，偏置和方差。其中偏差和方差是可预测的部分。噪音期望值为0，不可预测，表达了在当前任务上任何学习算法所能达到的期望泛化误差的下界。

1. 迭代减少偏差的时候，每次迭代都利用bagging把方差消除一定程度。bagging是通过多个不同训练集上的学习机的预测值的平均得到最终预测值，使得数据扰动产生的影响小。
2. 无偏置残差。被选中作为训练样本的分类器

如果样本是分类器训练样本，那个该分类器对于此样本的预测是有偏置的。

我们只能用分类器来预测训练集之外的样本。由此得到无偏置残差。因为样本被学习机拟合过，有偏置的残差往往是一个乐观结果，缺少客观性。

1. 迭代逼近真实值。每次迭代的目标值都是上一轮的无偏置残差。通过拟合残差来训练下一个分类器。

所有分类器的和就是最接近真实值的最终分类器。

1. 停止条件。当前阶段均方误差是之前最好的阶段的均方误差值的1.1倍时，不再迭代。

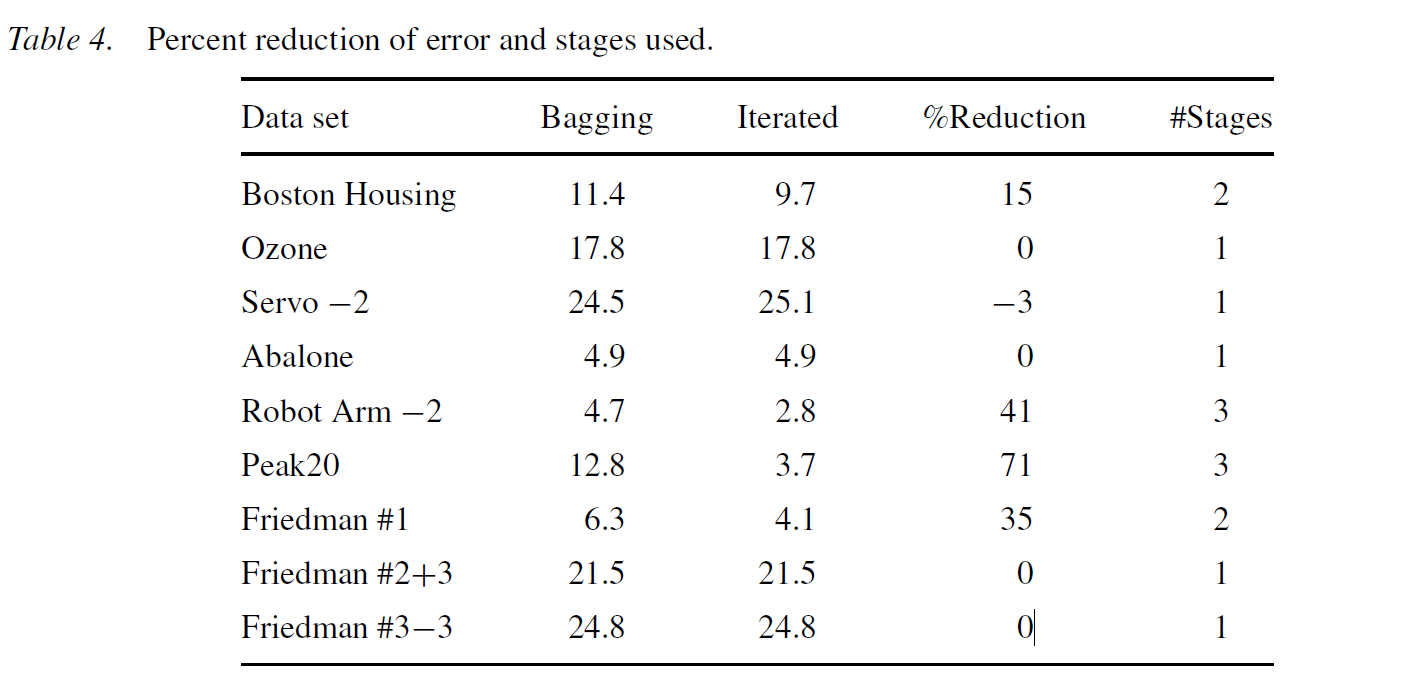
测试结果：

数据集：UCI Physicochemical Properties of Protein Tertiary Structure Data Set（CASP）

表4 迭代bagging和普通bagging在数据集CASP中的结果

|  |  |  |
| --- | --- | --- |
|  | 均方误差（error） | 可释方差（explained variation） |
| 普通bagging | 12.827350459 | 0.65646158672 |
| 迭代bagging | 12.7744147051 | 0.657880000121 |

结果：在第二轮迭代后停止，并且第二轮被舍去。因为迭代轮数为1，所以迭代bagging和普通bagging在此数据集上没有明显区别。这一点在论文中的实验也有体现：



1. 对比

对于MultiBoosting和Iterated Bagging来说，两者的优缺点对比如下表所示：

表5 MultiBoost和Iterated Bagging优缺点

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| MultiBoost | 结合了AdaBoost能够有效降低偏差以及Bagging能够有效降低方差的特点，相比于前两者MultiBoost能够更好地同时降低偏差和方差，提升预测的准确率 | 训练和预测的时间比AdaBoost和Bagging要长 |
| Iterated Bagging | 迭代bagging利用逼近真实值的方式，产生多阶段的学习机组，求和得到误差最小的总函数。它的迭代次数越多，误差越小，有效降低误差。 | 相对于普通bagging算法，它牺牲了方差，所以方差会变大，相当于单分类器和bagging的折中。  迭代轮数对于减小误差至关重要，但是迭代轮数少，甚至一部分是一轮迭代，发挥不了算法的减小误差的效果。 |

Adaboost可以同时减少模型的偏置和方差，主要是偏置。

Bagging主要减少模型的方差。

MultiBoosting和Iterated Bagging，两者出发点都是把Adaboost和Bagging的优点结合起来，同时有效减少模型的偏差和方差。但是两种算法的构造方式有着很大区别。最终的表现，即优缺点产生很大差异。两者优缺点在表5中已经总结，当选择其中某个算法的时候，可以对优缺点和业务需求进行匹配，根据实际需要选择相应的算法。