使用 C4.5 算法完成分类任务

周昊一*

(南京大学 计算机科学与技术系, 南京 210093)

Using C4.5 Algorithm for Classification Task

Haoyi Zhou*

(Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract: In this paper, we introduced a classification task, and chose the decision tree algorithm to do the task. We briefly introduced the decision tree, and showed the procedure of applying ID3 and C4.5. Next we discussed the performance of the two algorithms, and finally made a conclusion that C4.5 with pruning is the best choice for this task.

Key words: data mining, classification, decision tree

摘 要: 在这篇文章中，我们首先引入了一个分类任务，并且选择使用决策树来解决这个问题。我们简要介绍了决策树算法，并且展示了使用 ID3 与 C4.5 两种算法的步骤。接下来我们讨论了这两种不同算法的性能表现，最终得出结论：带剪枝的 C4.5 算法最适合完成这个分类任务。

关键词: 数据挖掘，分类，决策树

中图法分类号: TP301 **文献标识码**: A

1 问题分析

首先简要描述一下此次需要解决的问题：

现在有 1100 个观测数据，每个观测数据有 42 个属性。其中的 700 已经根据其属性标上了标签，现在要把剩下 400 个数据也标上标签。

这样的一个问题可以看作是一个分类问题（classification）：用一个大小为 700 的训练集训练一个分类器，再用这个分类器对剩下的 400 个数据的进行分类。

2 算法分析

现在的分类算法主要有以下几种：决策树，贝叶斯分类，神经网络，支持向量机。在综合了算法的时间复杂度，正确率以及实现难度等因素后我选择使用决策树来完成本次的数据挖掘任务。

决策树使用树来存储分类信息，进行分类时类似于使用搜索树进行搜索。每个非叶节点指出一个用于分裂的属性，以及分裂点，叶节点则表示对应的类别。

* 作者简介:周昊一，学号 MF1233055。

2.1 决策树基本生成算法

决策树生成算法如下(由于本次的属性都是连续的,因此这里以及后面都不考虑离散情况):

算法: generateDecisionTree
输入:
D,用于训练的数据集
attributeList,候选属性集合
输出: 一棵决策树
方法:
(1) 创建一个节点 N;
(2) if D 中的数据都是同一类 C then
(3) 返回 N 作为叶节点,用类 C 标记;
(4) if attributeList 为空 then
(5) 返回 N 作为叶节点,标记为 D 中的多数类;
(6) 找出最好的分裂准则 splittingCriterion,包括分裂属性和分裂点;
(7) 用 splittingCriterion 标记节点 N;
(8) for splittingCriterion 的每个输出 j
(9) 设 D_j 是 D 中满足输出 j 的数据的集合;
(10) if D_j 为空 then
(11) 加一个树叶节点到 N,标记为 D 中的多数类;
(12) else 加一个由 generateDecisiontree(D_j,attributeList)返回的节点到节点 N;
(13) end for
(14) 返回 N;

算法 1.决策树基本生成算法

2.2 ID3

常用的决策树算法有 ID3,C4.5 和 CART 这几种,他们的区别都在于算法 1 中的第 5 行的分裂准则的生产，以及在剪枝的处理上。

这里我选择并实现了较为相近的 ID3 和 C4.5 这两个算法，并进行了比较。

ID3 算法在寻找分裂准则时使用的属性选择度量为信息增益（Gain）。它由如下几个公式定义:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2 p_i$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

选择具有最高增益 Gain(A)的属性 A 作为节点 N 的分裂属性。由于在这里所有的属性都是连续值的，因此还需要确定分裂点，算法如下：

算法: selectAttributeForSplitting
输入:
D,用于训练的数据集
attributeList,候选属性集合

输出: splittingCriterion,分裂准则,包含分裂属性和分裂点
方法:
(1) for attributeList 中的每个属性 A
(2) 将 D 按属性 A 排序
(3) for D 中的每个元素 D_i(i>0)
(4) 将 $D_i(A)$-$D_{i-1}(A)$作为一个分裂点 p;
(5) 计算属性 A 在点 p 分裂的 Gain(A);
(6) if Gain(A)<minGain then
(7) minGain=Gain(A);
(8) splittingAttribute=A;
(9) splittingPoint=p;
(10) end for
(11) end for
(12) 返回由 splittingAttribute 和 splittingPoint 构成的 splittingCriterion

算法 2.ID3 的分裂准则选择算法

2.3 C4.5

C4.5 是对 ID3 的改进，它采用增益率作为属性选择度量度，选择具有最大增益率的属性作为分裂属性。增益率(GainRatio)的定义如下:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \frac{|D_j|}{|D|}$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

算法: selectAttributeForSplitting
输入:
D,用于训练的数据集
attributeList,候选属性集合
输出: splittingCriterion,分裂准则,包含分裂属性和分裂点
方法:
(1) for attributeList 中的每个属性 A
(2) 将 D 按属性 A 排序
(3) for D 中的每个元素 D_i(i>0)
(4) 将 $D_i(A)$-$D_{i-1}(A)$作为一个分裂点 p;
(5) 计算属性 A 在点 p 分裂的 GainRatio(A);
(6) if GainRatio(A)<minGainRatio then
(7) minGainRatio=GainRatio(A);
(8) splittingAttribute=A;
(9) splittingPoint=p;
(10) end for
(11) end for
(12) 返回由 splittingAttribute 和 splittingPoint 构成的 splittingCriterion

算法 3.C4.5 的分裂准则选择算法

2.4 悲观剪枝

由于数据中可能存在噪声和离群点，决策树的一些分支反映的是训练数据中的异常，因此需要使用某种剪枝方法来剪去这些分支。

在 C4.5 算法中使用的是后剪枝方法中的悲观剪枝方法，它使用错误率评估对子树剪枝做出决定。在悲观剪枝中,不使用额外的剪枝集,而使用训练集.

算法中用到如下两个公式:

$$E = \sum J + L(T)/2$$

$$se(x) = \sqrt{\frac{x \times (N - x)}{N}}$$

其中，E 是估计错误数，J 在某个叶节点上分类错误的个数，L(T)表示决策树 T 中的叶节点个数，se 是标准错误，N 是训练集中元组的个数。

剪枝算法如下：

算法: prune

输入:

T,决策树

D,剪枝集,在 C4.5 中即为训练集

方法:

(1) 节点 N=root(T);
(2) 假设 N 的子树都被剪去,用子树中有最多数据被分到的类作为 N 的标记,以此计算 error=0.5+D 在这种情况下分类错误的次数;
(3) 计算 estimateError=E+se(E)
(4) if error<estimateError then
(5) 删除 N 的子树,并用子树中有最多数据被分到的类来标记 N;
(6) else
(7) 用 N 这个节点的 splittingCriterion 将 D 中的数据划分为 listOfD;
(8) for N 的每一棵子树 T_i
(9) prune(T_i,listOfD[i]);
(10) end for

算法 4.悲观剪枝算法

3 性能评估

在这一节中将用横向和纵向比较的方法分析一下两种算法的性能。

下面使用的所有测试数据都是 700 个的完整训练集，并且使用 10 叠交叉验证。

3.1 C4.5与ID3的比较

首先使用不带剪枝的 ID3 算法。

不带剪枝的 ID3 算法			
轮数	错误数	错误率	训练时间（ms）
0	2	0.028571429	818
1	1	0.014285714	816
2	5	0.071428571	741
3	1	0.014285714	828
4	1	0.014285714	819
5	2	0.028571429	826
6	0	0	827
7	0	0	823
8	5	0.071428571	763
9	2	0.028571429	839
average	1.9	0.027142857	810.0

表 1.不带剪枝的 ID3 算法

可以看到正确率已经达到了较高的水平，平均错误率只有 0.0271。

接下来再使用不带剪枝的 C4.5 算法。

不带剪枝的 C4.5 算法			
轮数	错误数	错误率	训练时间(ms)
0	1	0.014285714	1227
1	1	0.014285714	1326
2	3	0.042857143	1236
3	3	0.042857143	1138
4	1	0.014285714	1250
5	2	0.028571429	1168
6	1	0.014285714	1329
7	0	0	1328
8	3	0.042857143	1257
9	3	0.042857143	1171
averate	1.8	0.025714286	1233.0

表 2.不带剪枝的 C4.5 算法

从算法效果来说,虽然提升不是很明显，但是 C4.5 要好于 ID3 算法。首先平均错误率从 0.0271 降低到了 0.0257，其次每次错误数的方差变小，更稳定了。

从算法运行时间来说,则是 ID3 算法较优，但是仍然在一个数量级，并且都紧在 1000ms 左右，因此影响并不是很大，可以认为两种算法都相当快。

3.2 C4.5带剪枝与不带剪枝的比较

带剪枝的 C4.5 算法结果如下：

带剪枝的 C4.5 算法			
轮数	错误数	错误率	训练时间(ms)
0	2	0.028571429	1267
1	1	0.014285714	1322
2	3	0.042857143	1236
3	3	0.042857143	1144
4	1	0.014285714	1249
5	2	0.028571429	1175
6	1	0.014285714	1328
7	0	0	1334
8	2	0.028571429	1256
9	3	0.042857143	1171
average	1.8	0.025714286	1248.2

表 3.带剪枝的 C4.5 算法

与表 2 相比，发现正确率并没有提升，但是稳定性则是进一步提高了。究其原因，是无剪枝时的正确率已经非常高了，进行交叉验证时错误率已经只有 0.0257，而在用训练集作为测试集进行测试时更是能达到 0 的错误率。而在进行悲观剪枝时，是使用训练集作为剪枝集，因此过低的错误率使得错误估计几乎变的没有意义。

而从运行时间上来讲，并没有明显的边长，平均慢了 15.2ms，可以认为是没有什么代价。

因此总的来说，虽然在这里使用剪枝收效甚微，但是还是聊胜于无。

4 结论

在进行了多方面的考虑，以及实际测试后，我选择了带悲观剪枝的 C4.5 决策树算法来对本次的测试集进行分类。

References:

[1] Jiawei Han, Micheline Kamber 著范明 孟小峰 译 《数据挖掘——概念与技术(第二版)》机械工业出版社.