班　级　1407031

学　号　14070310001

西安电子科技大学

本科毕业设计论文

题　　目　机器学习中决策树及相关算法的研究及应用

学　　院　数学与统计学院

专　　业　统计学

学生姓名　尹昊宇

导师姓名　王红军 副教授

毕业设计（论文）诚信声明书

本人声明：本人所提交的毕业论文《机器学习中决策树及相关算法的研究及应用》是本人在指导教师指导下独立研究、写作的成果，论文中所引用他人的无论以何种方式发布的文字、研究成果，均在论文中加以说明；有关教师、同学和其他人员对本文的写作、修订提出过并为我在论文中加以采纳的意见、建议，均已在我的致谢辞中加以说明并深致谢意。

本论文和资料若有不实之处，本人承担一切相关责任。

论文作者：_____ (签字)　　时间：　　年　　月　　日

指导教师已阅：_____ (签字)　　时间：　　年　　月　　日

摘 要

分类与回归问题一直是数据挖掘研究领域中的核心问题，而决策树算法就是解决这两类问题的一种常用算法。该算法易于理解，可解释性强，模型准确度高，同时还可以被重复使用。随着剪枝算法的完善以及决策树算法种类的更新，使得决策树模型拥有了更好的泛化能力，也极大程度的扩展了其应用范围。

本文的工作可以归纳为如下几点：

(1) 介绍了决策树模型的基本概念，重点论述了 ID3 算法、C4.5 算法以及 CART 算法的特征选取准则以及建树流程。为解决模型泛化能力不足的问题，又分析了预剪枝技术与后剪枝技术的算法流程。

(2) 基于隐形眼镜数据以及骑行数据，实现了 ID3、C4.5、模型树、CART 以及剪枝 CART 算法。通过实验分析比较了这几种算法在模型准确率与复杂度方面的异同，同时也比较了这几种算法与回归分析和聚类分析模型之间的准确率差异。实验发现决策树模型相比于回归分析以及聚类分析拥有更高的准确率。对于分类问题，ID3 算法与 C4.5 算法之间性能差异不大，剪枝 CART 算法由于利用基尼指数选取特征来构建二叉树，并最终利用剪枝算法提高泛化能力，拥有最小的模型复杂度以及最高的准确率，其模型准确率达到了 84.7% 且模型只有 3 个叶结点。对于回归问题，模型树算法拥有最好的表现，预测值与真值之间的相关系数达到了 0.976。

(3) 将剪枝 CART 算法运用于实际数据中，设计了一种以剪枝 CART 算法为核心，结合特征处理技术以及缺失值处理技术的模型。将该模型运用于 Titanic 数据集上，对乘客生存状况进行分析，通过实验发现模型的分类准确率达到了 79.9%，拥有较好的分类效果。

关键词： **数据挖掘** **决策树** **剪枝 CART** **模型树** **缺失值处理**

ABSTRACT

Classification and regression are always the core issues in the field of data mining, and decision tree algorithm is a usual algorithm to solve these problems. The algorithm is simple, interpretable and accurate. It can also be reused. With the perfection of the pruning algorithm and the update of the type of decision tree algorithm, the decision tree model has better generalization ability and a broader range of applications.

The work of this thesis can be summed up as follows:

(1)The basic concepts of the decision tree model are introduced. The thesis focuses on feature selection criteria and tree building process of the ID3 algorithm, C4.5 algorithm and CART algorithm. In order to solve the problem of insufficient generalization ability of the model, two pruning techniques such as pre-pruning and post-pruning are analyzed.

(2)Based on lenses data and riding data, ID3, C4.5, model tree, CART and pruning CART algorithm are implemented. The similarities and differences of these algorithms in model accuracy and complexity are analyzed and compared through experiments. Also, we compare the differences between these algorithms and regression analysis and cluster analysis in model accuracy. Experiments show that the decision tree model has higher accuracy than regression analysis and cluster analysis. For classification problems, there is little difference between ID3algorithm and C4.5algorithm. The pruning CART algorithm constructs a binary tree by using gini index and it uses the pruning algorithm to improve generalization ability. Therefore, it has the lowest model complexity and the highest model accuracy. The accuracy of the model is up to 84.7% and the model has only 3 leaf nodes. For regression problems, the model tree algorithm has the best performance, and the correlation coefficient between predictive values and truth values is up to 0.976.

(3)The thesis applied the pruning CART algorithm to the actual data, and a model with pruning CART algorithm as the core, combining feature treatment technology and missing value treatment technology is designed. The model is applied to the Titanic dataset to analyze passengers' living conditions. Experiments show that the classification accuracy of the model is up to 79.9% and the model has good classification results.

Keywords: Data mining Decision tree Pruning CART Model tree Missing value treatment

目录

第一章 绪论

1.1 研究背景

近年来，随着互联网的飞速发展，我们周围的信息量呈指数级增长，数据种类繁多，数据质量也参差不齐。为了寻找数据中的模式与规律，就必须要对数据进行更高层次的处理。而分类与回归不仅是人们日常工作和生活中的常见问题，也一直是统计学习领域中的热点问题。在人类的科技发展历史中，创造出的许多算法都旨在精确而又高效的解决这两个问题。在样本量适中，数据规则的情况下，传统统计方法中的判别分析与线性回归分析等方法已经能够较好的解决分类与回归这两个问题。但在数据结构复杂，数据质量不高的情况下，一些传统统计模型由于前提假设限制过多，表达能力有限，从而稍显乏力，最终造成了数据量爆炸但模型匮乏的现象。在这种情况下，各种数据挖掘技术相继诞生，而决策树就是其中一项最为常见的方法。在众多数据挖掘算法中，决策树算法凭借其算法简单而又高效的特点，被广泛运用。

1.1.1 算法背景

数据挖掘技术旨在对数据预处理，选取数据特征，运用合适的算法来发掘数据潜在规律，从而提升人们对数据的认知程度。在如今数据爆炸的时代，人们对于数据中隐含信息的需求也日益增长，由于传统统计方法在这一领域的局限性，很大程度上促进了拥有交叉学科背景的数据挖掘技术的发展。

数据挖掘算法大致可以分为监督式学习算法，非监督式学习算法以及半监督式学习算法。而常规的分类与回归问题均可以使用监督式学习算法来解决，其中运用比较广泛的算法除了决策树算法之外，还有朴素贝叶斯模型，神经网络算法以及支持向量机等。

决策树算法首先由心理学家兼计算机科学家 E.B.Hunt 在 1962 年提出，在分类与回归问题中均有广泛运用。决策树模型是基于树结构来构建的，在面临一个决策问题时，会通过若干判断将原问题分为若干个子决策，从而得出最终决策，也就是说任意一条从根结点出发到叶结点的路径都是一种判断准则，这种决策方式也符合人类在面对决策时的处理机制[1]。决策树算法十分高效，面对大量训练数据时，无需像神经网络模型一样进行大量的迭代运算来估计参数，也无需像朴素贝叶斯模型一样在估计时需要先获取先验信息。决策树算法也十分易于维护，

构建完树模型后，面对新的样本数据，只需对树进行简单维护就可以多次使用并保证其分类精度。除此之外，决策树算法拥有强大的表达能力，算法生成的判别规则非常容易被理解。为此，决策树中 C4.5 算法以及 CART 算法还被列为数据挖掘十大算法。

在利用决策树算法处理实际问题时，由于训练数据的样本量十分大，数据结构也较为复杂，往往会导致训练得到的决策树结构十分复杂。这样的决策树虽然在训练集上的误差非常的小，但是其泛化误差却很大，使得模型的迁移性变差，从而在实际运用存在一定的局限性。但随着决策树剪枝算法的提出，这类问题也得到了较好的解决。

1.1.2 应用背景

随着互联网规模的扩大，我国网民人数也呈现爆炸增长的趋势。在 2017 年 8 月 4 日下午，中国互联网络信息中心发布了 2017 年度《中国互联网络发展状况统计报告》，该报告指出，截止至 2017 年 6 月，我国网民人数已经达到了 7.51 亿人，半年的增长率达到了 2.7%, 互联网普及率达到了 54.3%。而网民每天或多或少会在互联网上进行一些活动从而留下相关信息，这一系列庞大的数据意味着现在人们可以轻松地收集到海量数据，并通过对这些数据分类处理，来改善日常生活。

例如电子邮件凭借其正规性以及快捷性，已经成为了人们日常交流不可缺少的一项工具。对于电子邮箱使用者而言，对于收到垃圾邮件或许已经司空见惯，有时垃圾邮件的数量甚至远远多于正常邮件的数量，这或多或少对于人们的日常工作产生了影响。由于无法完全在根源上杜绝垃圾邮件，因此解决该问题的主要工作就在于防护方面。而通过对邮件行为特征提取，在利用决策树算法进行分类可以很大程度上拦截住垃圾邮件[2]。同时，随着人们的日常金融活动的增加，使得一些商业银行在保证其高效运营的同时将不可避免的面临一些信用风险。信用风险指借贷人由于自身原因无力还贷，导致银行由于无法收回贷款而产生损失。而银行通过对借贷人的经济活动进行分析，提取特征，再利用决策树模型来衡量其还贷能力可以在很大程度上降低所需面临的信用风险[3]。此外，决策树模型在事故评估方面也有所运用，即通过提取事故中的一些特征，来预测各类结果发生的可能性，从而改善基础设施，保证用户安全。可见，作为一种简单而又高效的算法，决策树模型在信息防护，风险评估以及生存分析领域都有着广泛的应用价值。

1.2 国内外研究现状

决策树是基于树结构，通过自上至下的递归方法来构造的模型，它基于样本属性来对实例进行分类，可以被看作是一系列判断规则的集合[4]。决策树模型拥有强大的表达能力以及较快的分类速度。在训练模型时，往往利用最小化经验风险的原则来构建决策树，但只考虑经验风险却不考虑模型自身复杂度的决策树往往会有过拟合的风险，因此从正则化的角度出发，决策树模型还需要进行剪枝处理。通常，决策树的构建通常可以归纳为如下 3 个步骤：根据给定的特征评分准则选择最佳的分割属性、根据分割得到的子数据集递归的生成决策树以及对决策树进行剪枝处理。

与决策树相关的算法国内外已经做了很多年研究，最初的决策树算法起源于 E.B.Hunt 在 1962 年提出的“Concept Learning System”[1]，该算法也确立了决策树分而治之的学习理念，为决策树接下来的发展奠定了理论基础。该系统示意图如下图所示：

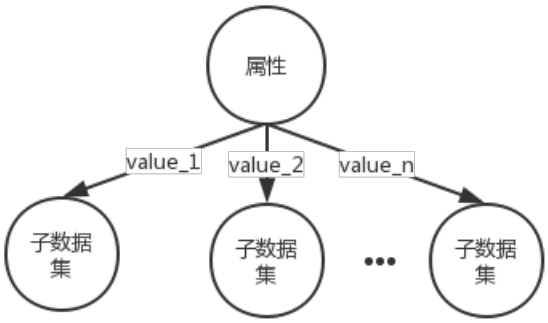

图 1.1 Concept Learning System 示意图

从图1.1可以看到，“Concept Learning System”的核心思想就是选择分裂属性来简化样本空间，从而达到将目标问题分解为若干子问题，通过求解子问题来得到最终的判断结果这一目的。

1979 年，J.ROSS Quinlan 利用“Concept Learning System”解决了一个关于判断象棋残局的课程作业，后将这些工作整理发表，ID3 算法由此诞生。之后在 1986 年，该算法又被重新发表在 Machine Learning 期刊上，引发决策树算法的研究热潮。ID3 算法的核心思想是利用信息增益作为各个特征的得分，从决策树的根结点开始，每次选择使信息增益达到最大的特征作为分支结点，利用特征的属性值将数据集划分为多个数据子集，随后基于数据子集采用递归的方式构建出整棵决策树。ID3 算法的理论较为清晰，实现方法较其他算法也比较简单。该算法可以

运用于对于大多数离散数据集的分类。ID3 算法虽然简单且容易实现，但在面对连续型数据以及属性取值较多的数据集时，存在一定的局限性。由于决策树算法本身是一个贪心算法，因此当训练数据存在噪声时，ID3 算法极易产生过拟合的问题。

为了解决 ID3 算法诸多的局限性，1992 年，J.ROSS Quinlan 在 ID3 算法上做了进一步改进，将其命名为 C4.5 算法。C4.5 算法利用信息增益比作为各个特征的得分，树的构造方式与 ID3 算法大致相同[5]，依旧是利用选出的特征将数据集划分为若干子集，再进行递归构建决策树。该算法主要运用于对离散或者连续数据集的分类，尤其是在属性值较多的数据集上，有着良好的表现，但如果单纯的只利用信息增益比来选择特征，最终的决策树模型往往又会产生特征选择偏好的问题，因此在实际使用中，人们往往利用启发式的学习方法来选择特征[4]。

为了进一步使决策树模型拥有更好的迁移性和可维护性，一些学者又给决策树增加了“增量学习”的功能，即已经训练好的决策树模型在面对新的训练样本时，不用进行重新训练，只需要对原模型进行局部调整即可。比较有代表性的算法有 1986 年 Schlimmer 和 Fisher 提出的 ID4[6] 算法，Utgoff 分别于 1989 年以及 1997 年提出的 ID5R[7] 和 ITI[8] 算法。这些算法虽然可以极大程度的减少决策树模型的训练花费和维护成本，但多次训练之后会与全样本训练的模型存在较大差异。

在 1984 年 Breiman 提出了一种新的决策树算法，CART 算法[9]，该算法是目前决策树算法中运用最为广泛的算法。该算法利用基尼系数来选择分支结点，将使基尼系数最小化的属性值作为分支结点，将数据集划分为两个子数据集，随后递归构建决策树。CART 算法构造出的决策树是一棵二叉树，较前面两种算法其树的结构最为简单，最后还可以采取剪枝的方法来降低决策树的过拟合风险。CART 算法既可以运用于离散数据的分类，也可以运用于连续数据的回归。

为了进一步降低决策树的高度，并提高其预测性能，Murthy 等人在 1993 年提出了多变量决策树算法 (OC1)[10]。OC1 算法是一种贪心算法，它首先会给每一个样本属性赋予一个初始权值，再贪心的寻找每一个样本属性的最优权值，当局部达到最优时，会再给模型加入一个随机扰动因素，从而试图去寻找一个全局最优解。在该算法下，决策树的每一个分支结点不再只选择一个最优属性，而是通过选择多个属性，并将它们线性组合来进行决策，从而简化决策树的复杂度，加快预测速度。在 1992 年，Guo 和 S.B.Gelfand 又提出了在决策树的叶子结点上嵌套多层神经网络的想法[11]，这种做法既保留了决策树模型高解释性的优点，同时也使得模型拥有了像神经网络那样强大的学习能力，但这种做法也使得决策树的训练成本大大增加。针对训练成本过大这一问题，许多学者在叶结点上嵌套了线性

分类器 (参数利用最小二乘的方法来估计) 和感知机 (一般只有两层神经元的神经网络模型) 来替代较为复杂的多层神经网络模型。

1.3 本文研究设想

本文将着重分析与研究决策树算法中的三种经典算法：ID3 算法、C4.5 算法以及 CART 算法。首先完整论述其不同的特征选择方式、对应的算法流程、决策树的剪枝方式以及模型树的概念。其次利用 Python 语言对这些算法进行实现，结合实际数据进行仿真训练，比较三种算法之间的优劣性。随后引入传统统计方法：回归分析与判别分析 (主要针对离散数据集)，并用同样地数据集进行训练，比较决策树算法与传统统计模型之间的性能差异。最后基于上述算法，构建一个可以训练各类数据集的系统化模型，并解决一个实际问题。主要分为如下几个步骤：数据预处理、模型选择、决策树构建以及决策树剪枝。数据预处理可以将数据规整化，例如填补缺失值，将离散数据连续化等。在决策树构建时，将会给决策树增加缺失值处理的功能，即当数据集存在少量缺失值时，不用通过预先的缺失值处理，也可以利用原始数据集进行训练，从而最大程度上利用了数据信息。最后的剪枝处理利用验证数据集，在训练数据集存在噪声时，能够最大程度上的增加决策树的泛化能力。

1.4 论文结构

论文的结构安排具体如下：

第一章绪论，主要介绍了决策树算法的算法背景、应用背景、运用的领域以及国内外的研究现状，也简单介绍了本文的研究设想与大致框架。

第二章论述了三种经典决策树算法的相关内容，包括特征选择方式，决策树的构建方式以及各类剪枝技术。最后对这三种算法进行总结。

第三章利用实际数据集对三种决策树算法进行仿真训练，比较其优劣性，同时引入模型树的概念来提高在复杂数据集下的预测精度。最后引入传统统计模型，在同一数据集下比较决策树模型与传统统计模型之间的性能差异。

第四章设计了一个系统化的模型，使模型可以处理数据质量不高甚至是含有较多缺失值的数据。再针对从 Kaggle 网站下载的实际数据，解决实际问题并测试模型性能。

第五章为总结与展望，将对论文所做工作进行总结，说明论文所得结果以及研究意义。同时指出论文的不足之处，并在此基础上提出对应的改进想法。

第二章 决策树算法理论

2.1　算法简介

决策树模型将通过一组包含多个属性以及一个数据标签的数据集来构建。在构建好的决策树模型中，每一个分支结点可以看作是一个数据属性，每一个叶结点可以看作是一个数据标签 (最终的预测结果)，每一条从根结点出发到叶结点的路径都可以看作是一条分类规则。因此决策树模型的学习本质是在训练数据集中归纳出一条与训练数据集没有矛盾或者是产生矛盾最小的决策树。

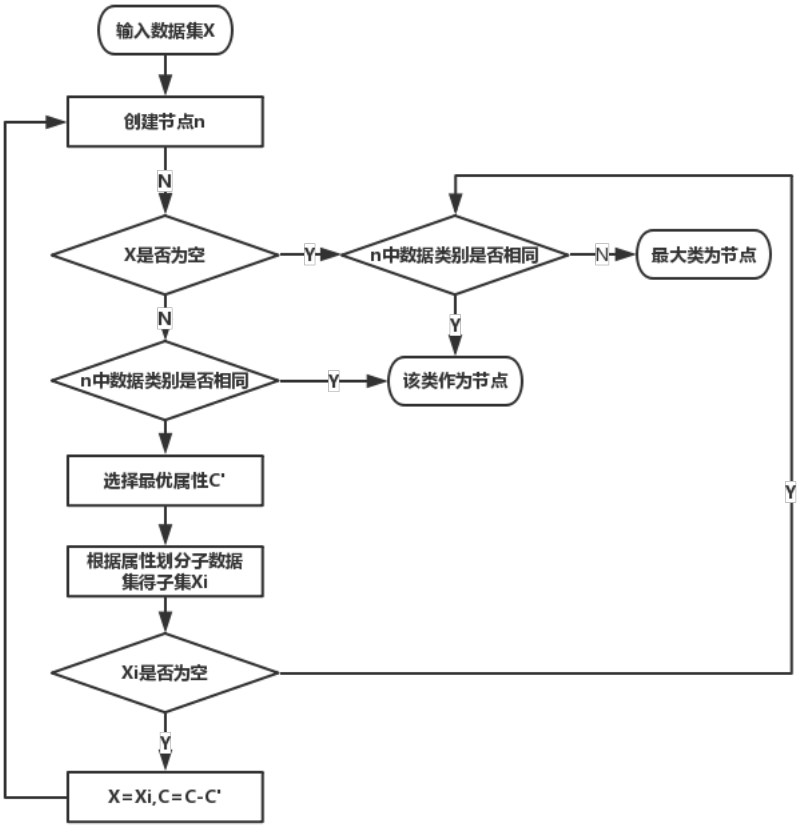

图 2.1　决策树构建流程图

决策树模型在构建时，每次都会以某种方式来选择一个最佳属性作为分类结点，如果在某一个结点上属性值都被用完或者该结点上的所有样本都属于同一类，则将该结点作为决策树的叶子结点。由于决策树模型是一种贪心算法，为了提高其泛化能力，往往会将数据集分为训练集与验证集两部分，前者用于训练，后者用于剪枝处理。特征的选择方式与剪枝处理将在本章之后小节中详细叙述。决策

树的构建流程图如2.1所示。

2.2 经典决策树算法

决策树算法从最早被提出，至今已经扩展成许多性能功能不一的算法。本节将主要介绍 ID3 算法、C4.5 算法以及 CART 算法。重点介绍其特征选择方式，构建流程以及剪枝方法。

2.2.1 ID3 算法

ID3 算法定义了同类算法的基本流程。ID3 算法的核心在于利用信息熵的概念来进行特征的选取。信息熵是对信息的一个量化指标，反映了信息源的不确定度，即信息源包含信息的多少。信息熵越大，信息源的不确定性就越大，信息熵越小，信息源的不确定性就越小，当信息熵等于 0 时，意味着信息源是完全确定的。接下来给出信息熵的定义。

设离散型随机变量 X 取值是有限的，分别为 $x_1, x_2, ..., x_n$，X 所对应的概率分布为:

$$P(X = x_i) = p_i \tag{2-1}$$

则随机变量 X 的信息熵可以由下式来定义:

$$E(X) = -\sum_{i=1}^{n} p_i log_2 p_i \tag{2-2}$$

特别地，当 $p = 0$ 时，有 $0log_2 0 = 0$。

条件信息熵 $E(Y|X)$ 表明在随机变量 X 给定的情况下，随机变量 Y 的不确定性。条件信息熵可以定义为在随机变量 X 给定的情况下，Y 的条件概率分布对应的信息熵再对随机变量 X 求数学期望，下面给出具体表达式。

现设有一二维随机变量 (X, Y)，对应的联合概率分布为:

$$P(X = x_i, Y = y_i) = p_{ij} \quad i = 1, 2, ..., n; j = 1, 2, ..., m \tag{2-3}$$

则条件熵 $E(Y|X)$ 为:

$$E(Y|X) = -\sum_{i=1}^{n} P(X = x_i) \sum_{j=1}^{m} P(Y = y_j|X = x_i) log_2 P(Y = y_j|X = x_i) \tag{2-4}$$

当信息熵和条件信息熵中的概率由数据样本估计得到时，对应的信息熵与条件信息熵分别被称为经验熵与经验条件信息熵。概率的估计方式常用极大似然的方式来估计。设数据集 X 总共有 N 个数据样本，有 K 个类别，每个类别出现次

数为 $n_1, n_2, ..., n_k$，对应的概率参数为 $p_1, p_2, ...p_k$。则似然函数可以写为：

$$L = \frac{m!}{n_1!n_2!...n_k!}\prod_{j=1}^{k} p_j^{n_j} \tag{2-5}$$

其中 $\sum_{j=1}^{k} p_j = 1$，$\sum_{j=1}^{k} n_j = N$，令对数似然函数对每一个概率参数求偏导可得：

$$\frac{n_1}{p_1} = \frac{n_2}{p_2} = ... = \frac{n_{k-1}}{p_{k-1}} = \frac{N - \sum_{j=1}^{k-1} n_j}{1 - \sum_{j=1}^{k-1} p_j} \tag{2-6}$$

通过求解上述方程组，可以得到对应概率的极大似然估计为 $\hat{p_i} = \frac{n_i}{N}, i = 1, 2, ..., k$。

因此在给定数据集 X 后，其对应的经验熵可以写为：

$$E(X) = -\sum_{i=1}^{k} \frac{n_i}{N} log_2 \frac{n_i}{N} \tag{2-7}$$

给定特征 C，特征 C 对于数据集 X 的经验条件信息熵 $E(X|C)$ 可以写为：

$$E(X|C) = -\sum_{j=1}^{q} \frac{c_j}{N} \sum_{i=1}^{k} \frac{c_{ji}}{c_j} log_2 \frac{c_{ji}}{c_j} \tag{2-8}$$

其中 c_j 与 c_{ji} 分别为数据集中样本属性 c 取值为 j 的样本个数以及样本属性 c 取值为 j 且属于 i 类的样本个数。

对于给定特征 C，其对训练数据集 X 的信息增益 $gain(X, C)$ 定义为训练数据集的经验熵与给定特征 C 条件下的经验条件信息熵之差，即：

$$gain(X, C) = E(X) - E(X|C) \tag{2-9}$$

信息增益就是 ID3 算法用来选择划分数据集的特征的主要准则，信息增益越大，该属性可以提供的信息量也就越大，也就拥有更高的样本辨识度，这样得到的决策树将会拥有跟高的精度。在给定训练数据集 X，特征集合 C，类别标签集 K 以及信息增益的阈值要求 ε 之后，ID3 算法的具体实现步骤如下：

(1) 若数据集 X 中的所有样本都属于同一类别 K_j，则决策树 T 只有单个结点，将类别 K_j 作为该结点的类别标签并返回决策树 T。

(2) 若属性集合 $C = \varnothing$，则决策树 T 是一个单结点树，将数据集 X 中出现最多的类别 K_j 作为该结点的类标签，返回决策树 T。

(3) 若不满足前两个步骤中的情况，计算属性集 C 中各个属性对数据集 X 的信息增益，选取其中信息增益最大的特征 C_g 作为候选特征。

(4) 若特征 C_g 的信息增益小于给定的阈值 ε，则决策树 T 为单结点树，将数据集 X 中出现最多的类别 K_j 作为该结点的类标签，返回决策树 T。

(5) 对于特征 C_g 的每一个取值 v_i，将数据集分割为若干非空的子数据集 X_i，构建出对应个数的子结点，返回决策树 T。

(6) 对于第 j 个子结点，以该结点对应的数据集 X_i 为训练集，以 $C - C_g$ 为特征集合，重复之前五个步骤，得到子决策树 T_i，返回 T_i。

2.2.2 C4.5 算法

ID3 算法由于只能处理离散数据，并没有考虑到面对连续数据的情况。C4.5 算法针对这一局限做出了相应的改进。此外，C4.5 算法在选取分裂属性时，利用了信息增益率作为衡量标准，在讨论信息增益率概念与其优势之前，先观察如下数据集:

表 2.1 多属性值数据集

属性一	属性二	类别
1	2	0
2	1	0
3	2	0
4	1	1
5	3	1
6	3	1
7	1	1
8	3	1
9	2	0

由信息增益公式可以计算得到，属性一对应的信息增益为 0.76，属性二对应的信息增益为 0.46，根据信息增益最大原则应该选取属性一作为当前节点的属性，但是仔细观察可以发现，属性一将产生 6 个分支，每个分支结点只包含一个样本，显然此时分支结点的纯度以及达到最高。选择这样的“无意义”特征会导致决策树结构过于复杂，从而不具有泛化能力。因此，除了无法处理连续数据之外，ID3 算法还存在一个很大的局限性，即特征选择时偏好于属性值较多的特征，为了解决这一问题，下面介绍信息增益率。

信息增益率定义为:

$$gainratio(X,C) = \frac{gain(X,C)}{V(C)} \tag{2-10}$$

其中 $V(C)$ 定义为

$$V(C) = -\sum_{i=1}^{m} \frac{|X^i|}{|X|} log_2 \frac{|X^i|}{|X|} \tag{2-11}$$

其中 $|X^i|$，$|X|$ 分别表示属性 C 取值为 i 的样本个数以及数据集中样本个数。

定义了信息增益率后，再回顾表2.1中的数据，可以计算得到属性一的信息增益率为 0.24，属性二的信息增益率为 0.29。可以看到在使用信息增益率选取特征时，就可以避免选择属性取值过多的特征带来的过拟合问题。但是信息增益率又带来了另一项问题，即信息增益率在选择特征时总过分偏向于属性取值较少的特征，因此在实际选择特征时，并不是单纯的用信息增益率的大小来选择特征，而是先从所有属性中挑选出信息增益高于平均值的属性，再从中选取信息增益率最大的特征作为结点特征[12]。

在之前的讨论中均是基于离散型数据来生成决策树，但是实际生活中，包含连续属性的数据也十分常见，因此考虑决策树在连续特征方面的处理是十分必要的。C4.5 算法就很好的考虑了这一问题。由于连续特征的属性取值是无穷的，因此像离散特征那样利用所有可取值对数据集进行划分的方法是无法实施的。一般情况下会使用连续值的离散化技术来解决该问题，C4.5 算法采用了较为简单且常用的二分法。

设有数据集 X 与连续属性 V，在数据集 X 中，出现了 m 个不同取值，将其按从小到大的顺序排列分别为 $v_1, v_2, ..., v_m$，一般将两个取值之间的中位数 d 作为划分点，即可以将数据集划分为 X_d^+ 与 X_d^- 两个部分，分别表示在属性 V 上取值大于 d 的数据集合与小于 d 的数据集合。因此对于连续属性 V，会产生 $m-1$ 个划分点：

$$Split = \{\frac{v_i + v_{i+1}}{2} | 1 \leq i \leq m-1\} \tag{2-12}$$

在确定划分集合后，一般选取可以使该属性信息增益达到最大的点来作为该属性最后的划分点，即：

$$\max_{d \in Split} E(X) - \sum_{\lambda \in \{+,-\}} \frac{|X_d^\lambda|}{|X|} E(X_d^\lambda) \tag{2-13}$$

在决策树构造过程中，离散属性与连续属性的不同在于，父结点已经使用过的离散特征在子结点中不允许再出现，而父结点中使用过的连续特征却允许子结点继续使用。

C4.5 算法的实施流程与 ID3 算法基本相同，因此不再赘述，主要区别在 C4.5 算法引入了离散化连续属性的方式，在处理连续属性方面有较好的效果。此外 C4.5 算法引入信息增益率的概念，解决了 ID3 算法在选取特征时偏向于属性值多的特征的局限性。

2.2.3 CART 算法

CART 算法最著名的地方在于它假设决策树是一棵二叉树，在该算法产生分支结点时，不再以特征的所有属性取值来划分数据集，而是先找出所有属性中最佳的二元划分，再以这个划分来产生新的分支。因此决策树内部结点的取值只有是与否两种情况。CART 算法不仅可以对离散数据进行分类处理，还可以对连续数据进行回归处理。

首先介绍利用 CART 算法来进行分类，生成分类树的算法。与之前两种分类算法不同的是，CART 算法利用基尼指数作为衡量指标来进行特征的选取。基尼指数是用于衡量给定数据集中纯度的指标。设有数据集 X，共包含 m 个类别，某一样本为第 k 类的概率为 p_k，则对应的概率分布的基尼指数可以由下式定义：

$$gini(X) = \sum_{i=1}^{m} p_i(1-p_i) = 1 - \sum_{i=1}^{m} p_i^2 \tag{2-14}$$

从上式可以看到，数据集 X 的基尼指数反映了在数据集 X 中的任意两个样本不属于一个类别的概率，因此基尼指数越小，数据集 X 的纯度就越高，数据集也就越有序。

对于数据集 X，若取属性 V 为分裂属性，其取值为 $v_1, v_2, ..., v_m$，则数据集 X 可以根据特征 V 的某一取值 v_i 划分成 X_1 与 X_2 两个部分，即：

$$X_1 = \{(x,y) \in X | x_V = v_i\}, X_2 = X - X_1 \tag{2-15}$$

则在特征 V 取值为 v_i 的情况下，数据集 X 的基尼指数为：

$$gini(X, V = v_i) = \frac{|X_1|}{|X|} gini(X_1) + \frac{|X_2|}{|X|} gini(X_2) \tag{2-16}$$

分类问题中的 CART 算法将基尼指数作为选择分裂特征取值的主要准则，给定训练数据集 X，特征集合 C，类别标签集 K 以及阈值要求 ε 之后，分类问题中 CART 算法的实施步骤如下：

(1) 若数据集 X 中的所有样本都属于同一类别 K_j，则决策树 T 只有单个结点，将类别 K_j 作为该结点的类别并返回决策树 T。

(2) 若属性集合 $C = \varnothing$，则决策树 T 是一个单结点树，将数据集 X 中出现最多的类别 K_j 作为该结点的类标签，返回决策树 T。

(3) 若不满足上述情况，对于训练数据集 X，计算特征集合中所有特征对于数据集的基尼指数，对于其中任意一个特征 V，对其所有可能取值 $v_1, v_2, ..., v_m$，面对 $V = v_i$ 这一条件，可以将数据集分为 X_1 与 X_2 两个部分，再计算 $V = v_i$ 条件下数据集的基尼指数。

(4) 对于所有可能的特征与所有可能的取值，计算其对应的基尼指数，选择基尼指数最小的特征以及对应的取值作为对应的最优特征与最优样本切割点。对于最优特征与最优样本切割点，将数据集划为是与否两个部分，同时在现结点上生成两个子结点，将划分的数据集分配给对应的子结点。

(5) 递归调用上述四个步骤，生成 CART 分类树。

介绍完 CART 算法在分类问题中的应用之后，再考虑其在回归问题中的运用。一棵回归树的本质是将特征空间划分为若干个单元，而每一个单元均有一个对应的输出值。不妨设特征空间被划分成了 N 个单元，分别为 $Q_1, Q_2, ...Q_N$，每个单元对应的固定输出为 $\omega_1, \omega_2, ..., \omega_N$，则回归树模型可以由下式表示：

$$f(x) = \sum_{i=1}^{N} \omega_i I(x \in Q_i) \tag{2-17}$$

其中 I 为符号函数：

$$I(x \in Q_i) = \begin{cases} 0 & x \notin Q_i \\ 1 & x \in Q_i \end{cases} \tag{2-18}$$

当特征空间的划分给定时，一般利用最小化平方误差 ($\sum_{i \in Q_j}(y_i - f(x_i))^2$) 的准则来计算各个区域上的输出值，从而令：

$$\frac{\partial \sum_{x_i \in Q_j}(y_i - f(x_i))^2}{\partial \omega_j} = 0 \tag{2-19}$$

计算可得在该准则下，每个区域预测值的估计为该区域样本值得平均：

$$\hat{\omega_j} = mean(y_i | x_i \in Q_j) \tag{2-20}$$

在对于特征空间划分的问题上，与对于连续型数据的处理方法类似，先选取第 j 个特征的属性值 v_i 作为切分值，可以将特征空间切分两个部分：

$$Q_1(j, v_i) = \{x | x^j \leq v_i\}, Q_2(j, v_i) = \{x | x^j > v_i\} \tag{2-21}$$

由之前的讨论可知，在特征与切分值给定的情况下，最优的回归估计值是该区域样本取值的均值，因此需要解决上述问题，只需要遍历所有特征以及可以取到的值，寻找最小解，即解决如下问题：

$$\min_{j, v_i}[\min_{\omega_1} \sum_{x_i \in Q_1} (y_i - \omega_1)^2 + \min_{\omega_2} \sum_{x_i \in Q_2} (y_i - \omega_2)^2] \tag{2-22}$$

在确定预测方式以及空间划分方式之后，给定数据集 X 与精度要求 ε，回归 CART 算法具体步骤如下：

(1) 变量所有特征，对于给定的特征遍历所有可能切分值，寻找可以使的式2-22达到最小的值来确定最优划分变量以及最优划分点。

(2) 对于选定的最优划分变量与最优划分值，利用式2-20来确定在对应区域上的固定输出值。

(3) 每一次划分将会将数据集分为两个部分并将一个完整的特征空间划分为两个子区域，将划分之后的数据集分配给对应的子区域，重复前两个步骤，直至预测精度达到要求 ε。

(4) 最后将样本的特征空间划分为 N 个单元之后，得到回归决策树：

$$f(x)=\sum_{i=1}^{N}\hat{\omega}_i I(x\in Q_i) \tag{2-23}$$

2.3 决策树剪枝

决策树算法是通过自身的递归调用，从而产生决策树的，而终止条件往往是样本属于同一类或者已经用完了所有特征，这就使得决策树模型在训练集上的预测精度十分之高，但是在未知数据集上的表现不尽如人意。刚构建好的决策树往往规模十分庞大，复杂的结构使得其可读性较差，因此在保障正确率的前提下，规模较小的模型总是有较好的可读性与泛化能力，应用范围也更广。人们基于上述问题，提出了剪枝算法来简化决策树的结构，剪枝算法总体上可以分为预剪枝与后剪枝这两类。

在衡量决策树泛化能力时，人们一般会采用“留出法”的方式，即从训练样本中选取一部分数据作为验证集，其余样本作为训练集。利用决策树模型在验证集上的误差来衡量其泛化能力。

2.3.1 预剪枝

预剪枝指在决策树的构建过程中就进行剪枝处理，在构造分支结点之前先估计当前分支是否可以提高决策树的泛化能力，如果不能，就不进行划分，并将当前结点作为叶节点。预剪枝除了在每构造一个分支结点时衡量其泛化能力来完成剪枝工作，还有许多其他更为简单且效果较好的策略来完成预剪枝：

(1) 在构建决策树之前先设定树的最高高度，到达最大高度时立刻停止构建，当前结点为叶结点，对应类别为该结点上出现次数最多的类别。

(2) 给定数据集最小划分阈值，即每次划分数据集，若子数据集包含的样本个数小于阈值，就立刻停止构建。但是此方法只适用于数据集较大的情况。

(3) 给定衡量指标 (如信息增益，基尼指数) 的阈值。若遍历完所有特征，最优

的指标值依旧低于阈值，则立刻停止构建。

预剪枝操作十分简单，开销也远小于后剪枝技术，但是有些情况下浪费掉有用的样本数据导致预测精度下降，因此在精度方面，后剪枝会强于预剪枝，其过拟合风险很小。

2.3.2 后剪枝

后剪枝技术是运用较为广泛的一种剪枝技术，在决策树构建完成后 (一般是将决策树构建至最复杂的状态)，通过某种衡量标准，将其中一些子树转化为叶结点来减小树的规模，提高泛化能力。后剪枝方法相当于寻找一个最优子树 T，可以使得下式达到最小:

$$L_\alpha(T) = L(T) + \alpha|T| \tag{2-24}$$

其中 $L(T)$ 为树对训练集的预测误差，$|T|$ 为树的复杂度，一般用叶节点个数来衡量。α 为权衡预测误差与复杂度之间的参数。当 α 很小时，树会偏大，当 α 很大时，希望得到的最优子树规模就会偏小。

常用的后剪枝方法是“REP”算法[13]，对于决策树 T 中的内部结点，若将其子树剪去，该结点变为叶子结点，类别为结点样本中数量最多的类别。若剪枝后的树在验证集上的测试误差小于等于原树的误差 (当剪枝误差与原模型误差相等时，由奥卡姆剃刀原理，两个模型性能相同时应该选取更简单的模型)，就进行剪枝，否则就不进行剪枝。重复该步骤直至无法进一步降低测试误差或者没有内部结点为止。

决策树剪枝算法的种类繁多，开销和剪枝效果也有所不同，因此对于不同的决策树，应该选择合适的剪枝方法来进行处理。

2.4　本章小结

本章主要介绍了利用信息增益选择特征的 ID3 算法、利用信息增益率来进行特征选择的 C4.5 算法以及利用基尼指数来构建二叉决策树的 CART 算法，介绍了它们各自的算法流程以及特点。最后还介绍了剪枝技术在决策树模型过拟合方面的运用。

第三章 决策树实验结果对比

之前的章节介绍了三种经典决策树算法 ID3、C4.5 以及 CART 的分裂特征选取准则以及具体算法流程。本章将利用实际数据集对三种算法进行仿真训练，比较这三种算法的预测准确率以及模型复杂度，最后比较与传统统计模型之间的性能差异。同时也将引入模型树概念提升决策树在一些特殊数据集上的表现能力。

3.1 实验准备

3.1.1 数据准备

为了获取合理有效的实验结果来对算法性能进行对比分析，首先需要获取合适的训练数据集。本文选取了三组数据集作为实验数据，分别用于比较算法在分类与回归方面的性能。具体数据说明如下表:

表 3.1　数据集介绍

数据名称	数据介绍
lenses	此数据集为著名的隐形眼镜类型数据，包含 4 个特征属性，24 个样本和 3 个数据标签。
bikeSpeedVsIqtrain	此数据集为 Peter Harrington[14] 收集的骑车人骑行速度与智商数据，该数据中所有变量均为连续值。
bikeSpeedVsIqtest	该数据集结构与 bikeSpeedVsIqtrain 相同，主要用于模型测试。

这里再申明一下数据集各变量含义，隐形眼镜数据中 4 个特征分别表示年龄、近视程度、是否有散光以及患者眼泪量，数据标签 (类别) 为患者所适合的隐形眼镜类型。骑行数据集中作为自变量的是测试者的骑车平均速度，因变量是其智力水平。出于简化实验步骤的角度考虑，本实验选取的三组实验数据集均不含有缺失值，隐形眼镜数据为离散型数据而骑行数据为连续型数据。

本文将利用隐形眼镜数据集来比较三种经典决策树算法与传统统计模型 (离散数据上的判别分析) 之间的性能差异。由于该数据集没有额外的测试数据，因此本文使用 4 折交叉验证的方式，将数据集平均分为 4 份，即每次选取其中 18 组样本作为训练集，6 组样本作为验证集来计算模型精度，最终取多次实验精度的平均值作为最终的模型精度 (将模型在验证集上的预测准确率作为精度)。同时，本文将利用 bikeSpeedVsIqtrain 与 bikeSpeedVsIqtest 数据集来比较普通决策树模型 (CART 回归模型)、模型树模型以及传统统计模型 (回归分析) 之间的差异，这里将

模型预测值与实际值之间的相关系数作为模型的精度。

3.1.2 模型准备

本实验将会使用到的模型有 ID3 算法、C4.5 算法、CART 算法、离散数据上的回归分析模型、模型树模型以及线性回归模型。前三种算法在第二章已经详细叙述，这里利用 Python 语言对算法进行仿真实验，不再赘述，现主要介绍后三种模型。

对于在离散数据上的判别分析模型，设数据集中因变量 (类别) 为 y，自变量为 $x_1, x_2, ...x_p$, 其中自变量 x_j 所取水平为 $a_{j1}, a_{j2}, ..., a_{jm_j}$。首先将所有自变量均转化为哑变量，定义 $\delta_i(j,k)$ 为:

$$\delta_i(j,k) = \begin{cases} 1, & \text{当第 } i \text{ 组实验变量 } x_j \text{ 取水平 } a_{jk} \\ 0, & \text{其余情况} \end{cases} \tag{3-1}$$

离散数据中的判别分析模型即在线性假设的前提下，建立数据类别 y 与自变量水平取值 $\delta_i(j,k)$ 之间的线性关系:

$$y_i = \sum_{j=1}^{p}\sum_{k=1}^{m_j}\delta_i(j,k)\beta_{jk} + \varepsilon_i \tag{3-2}$$

其中 $i = 1, 2, ..., n$ 代表了实验编号，β_{jk} 为第 j 个自变量的第 k 个水平取值对应的系数，为待估计的常数，ε_i 为依赖于第 i 次实验的随机误差。

本文利用最小二乘估计的方式来估计带估系数，令残差平方和为下式:

$$R = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\sum_{k=1}^{m_j}\delta_i(j,k)\beta_{jk})^2 \tag{3-3}$$

最小二乘估计希望选取那些可以使得 R 达到最小的系数，由极值原理:

$$\frac{\partial R}{\partial \beta_{hl}} = -2\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\sum_{k=1}^{m_j}\delta_i(j,k)\beta_{jk})\delta_i(h,l) \tag{3-4}$$

其中 $h = 1, 2, ..., p$，$l = 1, 2, ..., m_h$，由上式可以整理得到 $\sum_{j=1}^{p} m_j$ 元的线性方程组。

求解得到的线性方程组，不妨设解为 $b_{jk}(j = 1, 2, ..., p,\ k = 1, 2, ..., m_j)$，则最后估计值为:

$$\hat{y_i} = \sum_{j=1}^{p}\sum_{k=1}^{m_j}\delta_i(j,k)b_{jk} \tag{3-5}$$

在求得因变量的估计值之后，还需要估计其所属类别。由于因变量的估计值显然并不一定是一个整数，因此无法直接判断样本所属于哪一类。现不妨设因变

量 y 可以所属 s 个类别 $(C_1, C_2, ..., C_s)$，在一个设计合理的实验中，样本中各种类别都应该有所出现，因此本位利用多次实验中各个类别的出现次数来确定测试样例所属类别。

设总共有 n 个实验样本，每一个 C_t 所包含的样本个数为 $n_t(t = 1, 2, ..., s)$，$\sum_{t=1}^{s} n_t = n$，令:

$$l_t = \frac{tn_t + (t+1)n_{t+1}}{n_{t+1} + n_t} \tag{3-6}$$

则可以得到最终的判别准则[15], 当 $y < l_1$ 时，认为测试样本属于 C_1 类别，当 $l_{j-1} \leq y < l_j$ 时，则认为测试样本属于 C_j 类，$j = 2, ..., s-1$，当 $y \geq l_s$ 时，则认为测试样本属于 C_s 类。

下面介绍之前引入的模型树概念。模型树与之前把叶结点设置为常数值得简单决策树不同，它将叶结点全部嵌套了一个数学模型来获得更高的精度，而其构建流程则与一般决策树完全一致。本文将模型树中叶结点的模型全部设置为线性回归模型，这样子得到的决策树模型相当于是一个分段线性函数，当样本中因变量取值不集中时，该模型会具有更好的解释能力。

最后，对于连续样本数据的回归分析方法，即假设样本中的因变量与自变量有线性关系 $Y = X\beta + \epsilon$，再采用最小二乘方法得到如下估计式:

$$\hat{\beta} = (X'X)^{-1}X'Y \tag{3-7}$$

由于其推导过程较为简单，这里不再赘述。

3.2　实验结果分析

3.2.1 离散数据集上的模型结果对比

通过将 ID3 算法运用于已经处理好的隐形眼镜数据集上，训练得到的决策树结构如图3.1所示。

图3.1中决策树的每一个分支结点表示一个特征，每一个分支 (箭头) 上的值表示属性值，叶结点上的值表示最终类别。

将 C4.5 算法运用在相同的测试数据集上 (依旧是隐形眼镜数据集)，训练得到的决策树如图3.2所示。

对比图3.1与图3.2可以发现，两棵树除了结点排列顺序之外，树的基本结构以及分裂特征的选取完全相同。实际上这种情况是合理的，因为测试数据集各个特征的属性的水平分量个数相对平均，从而导致 ID3 算法与 C4.5 算法的特征选择偏好不会对最终结果产生明显差异。同时，这一现象也反映了 C4.5 算法利用信息增益率来选择特征的方法对决策树结构的改变在一般情况下是有限的。

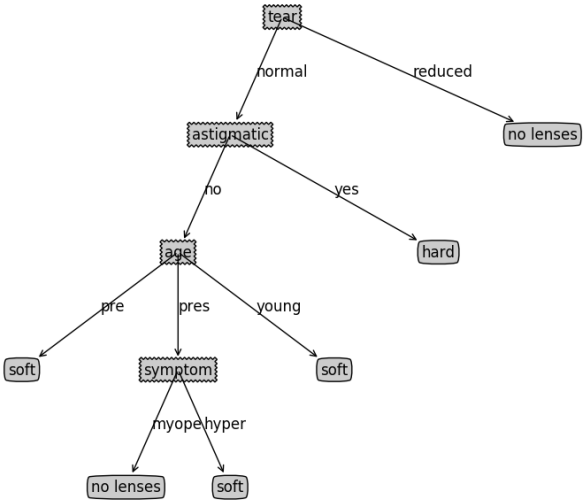

图 3.1　ID3 决策树

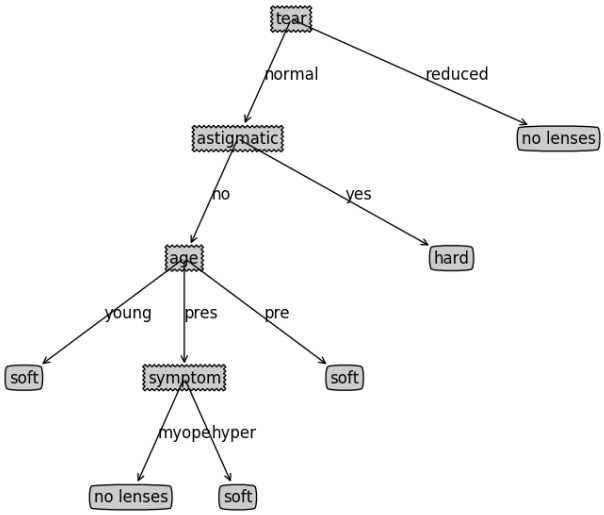

图 3.2　C4.5 决策树

最后将 CART 算法运用在测试数据集上，得到的决策树如图3.3所示。

图3.3中的分支节点表示了分裂特征以及分裂特征的取值 (用下划线分割)，箭头上为“yes”的分支表明该分支所对应的特征取值为分支结点中的取值，箭头上为“no”的分支表示其他情况。相比于图3.1与图3.2，图3.3中的决策树结构有了明显的简化：CART 决策树是一棵二叉树而非多叉树，CART 决策树的叶结点个数也少于前两棵决策树 (可以通过更少的规则来完成对所有情况的判断)。同样地，我们也可以发现，CART 算法即便使用了基尼指数作为特征选取准则，其特征的选取依旧与前两种算法相同。

图3.4是对 CART 算法采用剪枝算法后所得到的决策树。

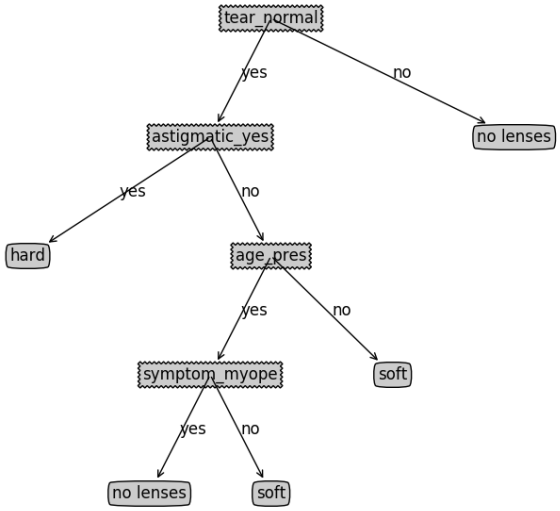

图 3.3　CART 决策树

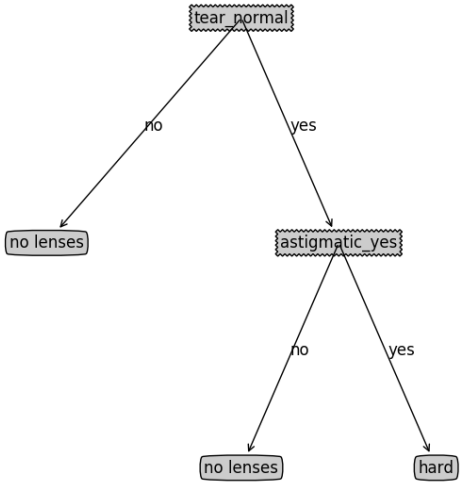

图 3.4　经剪枝的 CART 决策树

图3.4中各个结点的含义与图3.3中相同，对比这两张图可以看出，经过剪枝算法后，决策树的结构有了显著简化：树高从之前的 4 变成了 2，分支节点也只有两个。综合上述四张图可以发现，无论采用哪种准则来选取特征，在一般情况下 (数据特征的水平分量个数大体均衡)，其对最终决策树结构的影响都是有限的。而相比于特征选取准则，剪枝算法可以在很大程度上简化决策树模型的结构。

在了解决策树的大致结构后[①]，再将 ID3、C4.5、CART、剪枝 CART 算法以及判别分析方法运用于隐形眼镜数据集上 (其中剪枝 CART 算法在使用 18 个样本训练模型时，还会从 18 个样本中再额外留出 6 个样本用于计算泛化误差)，最终

[①]本节中所有决策树均是由 4 折交叉验证中的一组训练数据绘制得到

得到的五种算法的分类正确率如表3.2所示：

表 3.2 五种算法分类正确率对比

算法	正确率	平均叶结点数
ID3	76.4%	6
C4.5	77.8%	6
CART	79.2%	5
剪枝 CART	84.7%	3
判别分析	66.7%	-

可以发现，判别分析模型相比于其他几种决策树算法，其预测准确率最低。这可能是分类变量与各自变量水平分量之间的关系不符合线性关系所造成的，这也体现了该模型假设过多所带来的局限性。在 ID3 算法、C4.5 算法以及 CART 算法之中，CART 算法的准确率最高，但三种算法准确率之间的总体差距并不大，均没有高于 80%。经过剪枝后的 CART 算法预测正确率则有了较为明显的提升，达到了 84.7%。可见单纯的改变特征选择方式确实可以在一定程度上略微提升模型性能，但并不能带来什么本质改变，而剪枝算法却对模型的泛化性能的影响相当显著。同时，有研究表明特征的选取仅在 2% 的情况下会使结果有所不同[16]，但当数据存在噪声时，剪枝可以将决策树的泛化性能最高提高 25%[17]。这也与本文的结论一致。

除模型正确率之外，决策树的叶结点数也是衡量模型好坏的重要指标。叶结点的个数意味着决策树模型所产生的规则数，虽然规则越多，分类精度越高，但是这也会使得模型的泛化能力以及决策效率下降。因此一棵好的决策树应该使用尽可能少的规则来达到较高的准确率。从表3.2可以发现，由于 CART 算法是构建二叉树的缘故，其叶结点数较其他两种算法更少。而经过剪枝的 CART 算法更是构建出了我们所希望的少规则，高精度的决策树。

3.2.2 连续数据集上的模型结果对比

将 CART 回归算法、模型树算法 (模型树的建树方法依旧使用 CART 算法) 以及回归分析模型运用于 bikeSpeedVsIqtrain 数据集上，再运用 bikeSpeedVsIqtest 数据集来估计模型精度。各模型精度 (利用估计值与真实值之间的相关系数来衡量) 如表3.3所示。

从表3.3可以看出回归分析模型的精度是三种方法中最低的，模型树则表现的最好。为了更直观的解释这一结果，本文将训练数据以及测试数据绘制成如图3.5所示的散点图。

表 3.3　三种算法回归正确率对比

算法	相关系数
CART 回归	0.964
模型树	0.976
回归分析	0.943

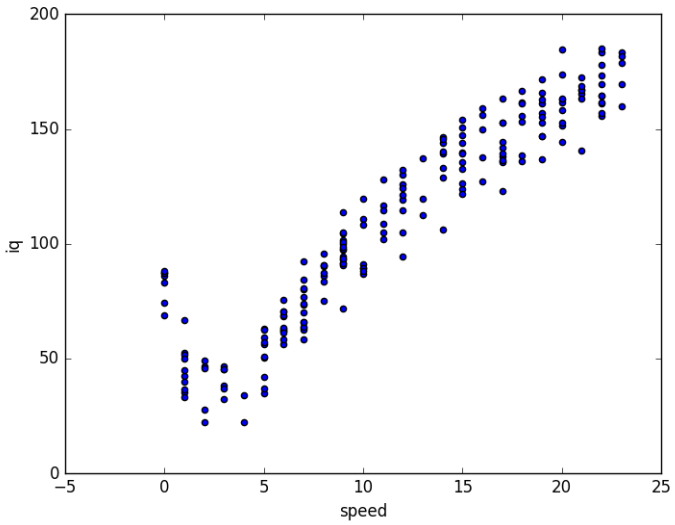

图 3.5　实验数据散点图

从图3.5中可以看到因变量与自变量之间的关系并不是一条直线，而是由两条直线构成的折线，这也解释了简单的回归分析模型无法得到较好结果的原因。而 CART 算法中用一组常数来预测结果也并不是最合适的，因为从局部来看，因变量和自变量之间呈现出一种线性关系。模型树算法却通过将回归分析模型与 CART 算法结合来避免上述这些局限性，这也反映出模型树算法拥有更好的灵活性和更优的可解释性。事实上，模型树的性能很大程度上依赖于叶结点的模型选择，不合适的叶结点模型可能会带来巨大的训练开销以及无法获得较好的精度。

3.3　本章小结

本章主要对 ID3 算法、C4.5 算法、CART 算法以及 CART 剪枝算法进行了实验分析，总结了其性能差异，同时也将这几种方法与回归分析及判别分析进行了对比。可以发现 CART 算法在保证结构简单的情况下也可以获得较好的精度，且剪枝算法可以使模型的泛化能力有较为明显的提升。

第四章 基于决策树算法的系统模型构建与分析

由于 CART 模型的复杂度较 ID3 算法以及 C4.5 算法更小，且可以同时处理连续数据以及离散数据，因此本章将基于剪枝 CART 算法构建一个功能较为完善的模型以便处理数据质量一般的数据集，最终运用实验来证明模型拥有较好的分类效果。

4.1　模型设计

由于实际情况下的数据集数据质量参差不齐，因此在将其用于模型训练之前往往还需要进行一些数据预处理工作，例如数据中心化处理、特征有效信息提取、构建新特征、构建新的特征水平、样本缺失值的填补等。这些预处理工作的质量对于最后分类结果的好坏也有着重要影响。图4.1展示了整体模型构建的相关步骤:

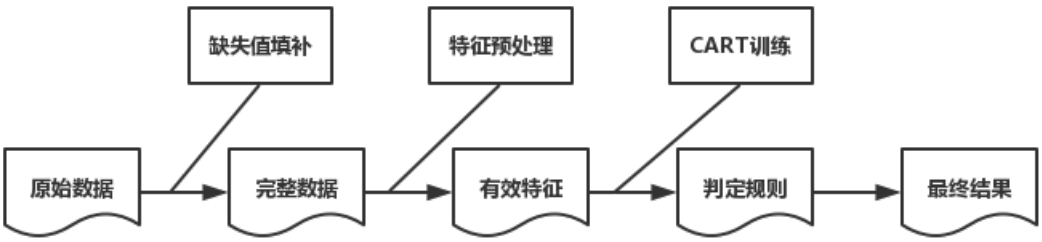

图 4.1　模型构建方案

分类模型 (剪枝 CART 算法) 的构建流程之前的章节已经有了详细介绍，这里不再赘述。而特征预处理的方法往往依赖于特征本身，没有固定的方案，因此下面着重介绍缺失值的处理方式。

当数据包含缺失值时，最直观的方法是填补完缺失的数据后再用于模型的训练。缺失值的处理方式大致可以分为三类:

第一类缺失值处理方法有均值填补 (运用所属特征的均值作为填补值)、众数填补 (运用所属特征的众数作为填补值)、前后数据填充、直接将缺失值单独作为一个水平分量等。但是上述这些方法只适用于缺失值数量非常少的情况，因为这样做相当于人为地给数据集增加了噪声，最终导致预测结果存在偏差。

第二类做法是利用算法进行拟合填充，即利用其他变量 (非缺失) 作为自变量，含有缺失值的变量作为因变量，以此来预测出缺失值。常见的做法有利用随机森

林来填补缺失值[18]、多重插补[19]、支持向量机[20] 及 BP 神经网络[21] 等。但此类做法也有一个较大的缺陷，即如果其他变量和缺失变量无关，则预测得到的缺失值将没有实际意义，如果预测结果非常的准确，则又说明这个变量可以被其余自变量解释，是没有必要加入模型的。一般情况下，该类做法的效果要优于第一类做法。

第三类做法是将变量映射到更高维的空间中，例如一个包含缺失值的性别变量可以投影为 3 个变量，分别为是否为男，是否为女，性别属性是否为缺失值。这种做法完整保留了所有原始数据信息且不用考虑缺失值，但这种做法会导致运算负担过重且只适用于样本数量非常大的情况。

最后需要注意的是，当某个特征含有大量缺失值时，可以考虑将该变量分为两类，一类是非缺失的，另一类是缺失的。但通常情况下，出于降低数据噪声的角度考虑，会将该变量直接舍去。

事实上，除去上述缺失值的填补方式之外，决策树模型自身也有处理包含缺失值数据的功能。设有训练数据集 X 以及属性 C，$\tilde{X}$ 表示数据集 X 中在属性 C 上未缺失的样本，决策树模型将根据 $\tilde{X}$ 来判断属性 C 的优劣。假定属性 C 一共有 m 个取值，分别为 $c_1, c_2, ..., c_m$，$\tilde{X}_j$ 表示数据集 $\tilde{X}$ 在属性 C 上取值为 c_j 的样本，$\tilde{X}_k$ 表示 $\tilde{X}$ 中属于第 k 类的样本 $(j = 1, 2, ..., m; k = 1, 2, ..., h)$，根据上述定义，可以得到下式:

$$\tilde{X} = \bigcup_{k=1}^{h} \tilde{X}_k = \bigcup_{j=1}^{m} \tilde{X}_j \tag{4-1}$$

若再为数据集 X 中每一个样本 s 赋予一个权重 ω_s, 则对于属性 C，不含缺失值的样本数所占比例为:

$$\lambda = \frac{\sum_{s \in \tilde{X}} \omega_s}{\sum_{s \in X} \omega_s} \tag{4-2}$$

不包含缺失值的样本中属于第 k 类的样本占所有不包含缺失值样本的比例为:

$$\tilde{p}_k = \frac{\sum_{s \in \tilde{X}_k} \omega_s}{\sum_{s \in \tilde{X}} \omega_s} \tag{4-3}$$

不包含缺失值的样本中在属性 C 上取值为 a_j 的样本占所有不含缺失值样本的比例为:

$$\tilde{\rho}_j = \frac{\sum_{s \in \tilde{X}_j} \omega_s}{\sum_{s \in \tilde{X}} \omega_s} \tag{4-4}$$

根据定义，显然有:

$$\sum_{j=1}^{m} \tilde{\rho}_j = \sum_{k=1}^{h} \tilde{p}_k = 1 \tag{4-5}$$

基于上述定义，可以得到在样本含有缺失值的情况下，属性值 C 取值为 c_j 时，数据集 X 的基尼指数可以推广为：

$$gini(X, C = c_j) = \lambda(\tilde{\rho}_j gini(\tilde{X}_1) + (1 - \tilde{\rho}_j) gini(\tilde{X}_2)) \tag{4-6}$$

其中 $\tilde{X}_1$ 与 $\tilde{X}_2$ 分别为不包含缺失值的数据样本中在属性 C 上取值为 c_j 的样本子集和不为 c_j 的样本子集。根据式2-14可以得到：

$$gini(\tilde{X}) = 1 - \sum_{k=1}^{h} \tilde{p}_k^2 \tag{4-7}$$

基于上述讨论可以看出，即使数据集含有缺失值，CART 模型也可以很方便的确定分支结点。在分支结点确定之后，若样本 s 在属性 C 上的取值未缺失，则直接将其划分入对应的子结点，且该样本的权重不变，依旧为 ω_s。若样本 s 在属性 C 上的取值是缺失的，则将该样本按照一定的概率分布划分入不同的子结点，即将该样本划分入所有的子结点，且将该样本的权重调整为 $\tilde{\rho}_j \omega_s$。

4.2　实验分析

本文采用最简单的精确度来衡量模型的优劣，即分类正确的样本占所有待预测样本的比例。本文中所使用的数据为下载自 Kaggle[①]的 Titanic 数据，旨在运用剪枝 CART 模型预测乘客是否生还。

Titanic 数据集包含 891 条训练数据，共有 10 个特征，分别为舱位等级、姓名、年龄、性别、兄弟姐妹个数、父母与孩子个数、船票信息、票价、客舱以及登船港口。其中船票信息相对于每个乘客而言是唯一的，因此该特征对于最终结果而言意义不大，故舍去。其中姓名特征的命名格式为“人名 + 称呼 + 姓氏”，例如：“Johnson, Mrs. Oscar W”。因此这里考虑将称呼提取出来，作为一个新的训练特征来使用。对于兄弟姐妹个数以及父母与孩子个数，本文做了如下汇总：

表4.1中 SibSip 表示兄弟姐妹个数，Parch 表示父母个数，-表示没有对应的水平取值，幸存率偏差为 SibSip 与 Parch 取值相同时所对应的乘客的幸存率 (存活人数除以总人数) 之差。可以看到在前三个水平，两个特征之间的幸存率偏差十分小，而样本在 SibSip 特征上取值为前三个水平的样本数占总样本数的 94.5%，样本在 Parch 特征上取值为前三个水平的样本数占总样本数的 98.3%。因此不难发现，SibSip 与 Parch 这两个特征对于乘客最终能否幸存的影响大致相同，出于简化模型的角度考虑，合并这两个特征，将这两个特征的取值之和作为新的特征。

[①]https://www.kaggle.com

表 4.1 特征汇总

幸存率偏差	SibSip	Parch
0.0017	0	0
-0.0150	1	1
-0.0360	2	2
-0.3571	3	3
0.1667	4	4
-0.2000	5	5
-	-	6
-	7	-
-	8	-

在缺失值处理方面，一共有三个属性还有缺失值，分别为年龄，客舱以及登船港口。其中对于年龄特征，包含缺失值的样本个数占总样本数的 19.9%，本文利用 R 语言中的 mice 包对该特征用多重插补的方式进行填补缺失值。客舱特征包含的缺失值是三个特征中最多的，缺失率达到了 77.1%，由于该特征的缺失值过多，因此本文将该特征做舍去处理。最后的登船港口特征，只有两个样本包含缺失值，出于降低数据噪声的角度考虑，本文不对这两处缺失值做任何的填补处理，直接利用上一小节介绍的决策树处理缺失值的方式来对其进行训练。

基于上述讨论，将 891 条训练数据输入最终的模型中，用训练完毕的模型去预测 419 条测试数据，最终的模型准确率为 79.9%。该模型准确率相比于第三章中理想情况下 (输入的数据集质量极佳) 剪枝 CART 算法 84.7% 的准确率，已经十分接近，且超越了其余三种算法的准确率。这也说明了该模型拥有良好的迁移性、鲁棒性以及在实际分类问题中运用的可行性。

4.3 本章小结

本章介绍了在面对实际数据时，一些相关的数据预处理方法以及决策树算法的一些扩展功能，给出了面对缺失值时的处理方案，完成了整体模型的设计。通过实验证实了模型的合理性，可以对数据质量一般的数据进行有效处理。

第五章 总结与展望

当今社会，信息化水平日益上涨，人们的周围充斥了各式各样的数据。数据挖掘作为一项新兴技术，通过与传统统计模型互补，极大程度的促进了人们对大量数据中所蕴藏的知识的认知程度。决策树算法作为数据挖掘技术中一项经典算法，凭借其简单高效，易于理解以及解释性强等优势，被广泛运用于各个领域。随着决策树算法的种类不断更新以及与其结合的模型不断增加，使得决策树算法变得更为智能，也扩展了其应用范围。本文主要针对决策树算法中的三种经典算法展开了研究，并将其运用于 Titanic 数据集上，做了对应的生存分析。具体而言，本文的工作成果主要可以归纳为以下三个部分：

(1) 本文总结了三种经典决策树算法，重点介绍了 ID3 算法，C4.5 算法以及 CART 算法的特征选取方式以及建树流程。同时也介绍了剪枝算法，良好的剪枝可以极大程度降低决策树过拟合的风险。

(2) 利用实验数据集对 ID3 算法，C4.5 算法以及 CART 算法进行仿真训练，比较了三种算法在分类正确率以及模型复杂度方面的差异。分析了三种算法与判别分析以及 CART 回归算法与回归分析，模型树算法之间的准确率差异。

(3) 基于剪枝 CART 算法，介绍了一些数据特征的处理方案以及缺失值处理方案，针对 Titanic 数据集，展示了决策树算法在解决实际问题时的完整操作流程，也证实了决策树算法在处理实际数据时有着较好的效果。

本文在完成上述工作的同时，也存在着诸多不足，在今后的学习工作中，将会从如下几个角度进行改进：

(1) 本文介绍的简化决策树的方法比较单一，只考虑了利用剪枝算法来降低决策树高度以及利用 CART 算法来构建二叉树。事实上，还可以建立多变量决策树，使得每一个分支结点是多个属性的组合而非单一属性，以此来获得更小的模型复杂度以及时间开销。

(2) 在对决策树算法进行实验分析时，针对回归与分类问题，本文只分别选取了一组实验数据集。为了使研究更具有普遍意义，可以选取更多的实验数据集来测试算法在不同数据集的表现能力，从而更客观的评价其性能。

(3) 虽然决策树算法自身拥有特征选择功能，但原始数据中大量的无关特征依旧会影响模型精度，而本文在利用剪枝 CART 算法处理 Titanic 数据集时，没有事先对数据特征进行筛选，即过滤去对幸存结果影响不显著的特征。今后在运用该模型解决实际问题时，应先进行特征工程处理。

致谢

至此，四年的西电本科生活就要结束了。在即将离别母校之际，内心情感十分复杂，既有对未来生活的憧憬，也有着对校园生活的怀念。回顾在西电的 4 年大学生活，一切都历历在目，这个校园承载着我 4 年的人生印记。在此论文完成之际，我想要对在我大学生活期间所有帮助过我的人致以最诚挚的感谢。

首先我要感谢我的毕业设计指导老师王红军。王老师学识渊博，循循善诱。每当我在实验中遇到难题，王老师总会耐心的为我解答疑惑。在论文撰写过程中，王老师不厌其烦的指导我论文的整体框架以及书写步骤。无论平时工作有多么忙碌，在约定和导师见面的时间，王老师总是会按时赶到并给予我悉心的指导。王老师严谨的学术作风，兢兢业业的工作态度无时不刻的影响着我，在王老师的教诲之下，才有了我今天的成长。

同时我要感谢我的父母，在我遇到困难时，你们无私的付出给予了我很大的帮助，你们对我的支持和期盼让我不再迷茫，让我有了今天的成就。

感谢数学与统计学院的每一位老师，你们给予我在学习和生活中的无私帮助，使我受益匪浅。

其次，我要感谢与我共同生活了四年的舍友，与你们的相处让我的生活不再枯燥乏味，让我以积极乐观的态度去面对每一天的大学生活。正是和你们一起营造的良好的学习氛围，才可以让我专注于论文的每一个细节。感谢你们给我带来的这一切。

最后感谢百忙之中阅读本文的各位老师，若有不足，望各位老师对我的拙作给予斧正，我一定会悉心听取，认真改善。

参考文献

[1] 周志华. 机器学习. 北京: 清华大学出版社,2016 年

[2] 于振灏. 基于模糊理论的决策树算法的研究及应用. 中国地质大学 (北京)2017 届博士学位论文. 2017 年

[3] 赵静娴. 基于决策树的信用风险评估方法研究. 天津大学 2009 届博士学位论文. 2009 年

[4] 李航. 统计学习方法. 北京: 清华大学出版社,2012 年

[5] Salzberg S L. C4.5:Programs for Machine Learning by J. Ross Quinlan. San Francisco:Morgan Kaufmann Publishers,1994.235-240

[6] Schlimmer J C,Fisher D H. A Case Study of Incremental Concept Induction.National Conference on Artificial Intelligence. 1986,8, 1.496-501

[7] Utgoff P E. Incremental Induction of Decision Trees.Machine Learning. 1997, 4(2).161-186

[8] Utgoff P E. Decision Tree Induction Based on Efficient Tree Restructuring.Machine Learning. 1997, 29(1).5-44

[9] Breiman L I,Friedman J H and Olshen R A. Classification and Regression Trees (CART).Encyclopedia of Ecology. 1984, 40(3).582-588

[10] Murthy,Sreerama and Kasif. OC1: randomized induction of oblique decision trees.National Conference on Artificial Intelligence Washington. 1993, 322-327

[11] Guo H,Gelfand S B. Classification trees with neural network feature extraction.IEEE Transactions on Neural Networks. 1992, 3(6).923-933

[12] Quinlan J R. C4.5:programs for machine learning.Machine Learning Proceedings. 1994, 70-77

[13] Fürnkranz,Johannes and Widmer. Incremental Reduced Error Pruning.Machine Learning Proceedings. 1994, 70-77

[14] Peter Harrington. 机器学习实战. 北京: 人民邮电出版社,2013 年

[15] 孙文爽, 盛玲玲. 伪变量的回归分析与判别分析. 云南大学学报: 自然科学版. 1980 年, 16-24

[16] Raileanu,Laura Elena and Stoffel. Theoretical Comparison between the Gini Index and Information Gain Criteria.Annals of Mathematics & Artificial Intelligence. 2004, 41(1).77-93

[17] Mingers J. An empirical comparison of selection measures for decision-tree induction.Machine Learning. 1989, 3(4).319-342

[18] 张晓琴, 程誉莹. 基于随机森林模型的成分数据缺失值填补法. 应用概率统计. 2017 年, 33(1).102-110

[19] Rubin D B. Multiple imputation for nonresponse in surveys.Journal of Marketing Research. 1987, 137(1).180-180

[20] 赵磊, 李国, 马现峰. 基于支持向量机的缺失数据补齐方法. 计算机工程与应用. 2006年, 42(36).207-208

[21] 张宏亭, 李学仁, 孔韬. BP 神经网络在缺失数据估计中的应用. 计算机工程与设计. 2007年, 28(14).3457-3459

附录 A ID3 算法

```
from math import log #导入科学计算库
import numpy as np  #导入矩阵计算库
#dataset用list嵌套的方式输入
#属性标签要和数据集的每一个数据始终一一对应
def classent(dataset):   #定义一个求信息熵的函数
    ent = 0.0
    samplenum = len(dataset)
    label = {}
    for i in dataset:
        temp = i[-1]
        if temp not in label.keys():
            label[temp] = 0
        label[temp] += 1
    for key in label:  #计算信息熵
        prob = float(label[key])/samplenum
        ent += -prob*log(prob,2)
    return ent
def splitdataset(dataset,axis,value):
#对于给定情况下数据集分割
    data = []    #给定某一个属性特征，对于一个属性值上进行划分
    for i in dataset:
        if i[axis] == value:
#使用过的数据属性，在子数据集中就不再出现了
            datatemp = i[:axis]+i[axis+1:]
            data.append(datatemp)
    return data
def choose(dataset):  #根据信息增益选择特征
    numfeatures = len(dataset[0])-1
    baseent = classent(dataset)
    bestgain = 0.0
```

```
    bestfeature = -1
    for i in range(numfeatures):
        feature = [example[i] for example in dataset]
#计算出所有属性分量
        uniquefeature = set(feature)
        newent = 0.0
        for value in uniquefeature:
#给定分裂属性之后将原数据集划分为若干子数据集
            subdataset = splitdataset(dataset,i,value)
            prob = len(subdataset)/float(len(dataset))
#计算某一个属性的信息增益率
            newent += prob*classent(subdataset)
        tempgain = baseent-newent
        if (tempgain>bestgain):
#如果信息增益率大于初始增益，就将该属性作为待选属性
            bestgain = tempgain
            bestfeature = i
    return bestfeature
def majorclass(classlist):#一个分支有多个类，投票来分类
    classcount = {}
    for vote in classlist:  #classlist为只一个分支上所有样本的所属类别
        if vote not in classcount.keys():
            classcount[vote] = 0
            classcount[vote] += 1
#对类别数量进行排序
    sortclass = sorted(classcount.items(),
key = lambda x:x[1],reverse = True)
    return sortclass[0][0]
#利用递归的方式来构造决策树
#不考虑剪枝情况下只有当某个分支只有一种类别或者遍历完所有属性才会递归结束
def createtree(dataset,labels,temp,tempmax = 0):
    classlist = [example[-1] for example in dataset]
    if dataset == []:
```

```
            return tempmax
        if classlist.count(classlist[0]) == len(classlist):
#全部属于一类
            return classlist[0]
        if len(dataset[0]) == 1:    #遍历完了所有特征
            return majorclass(classlist)
        bestfeature = choose(dataset)
        bestfeaturelabel = labels[bestfeature]
        tree = {bestfeaturelabel:{}}
        del(labels[bestfeature])  #选择过一次的特征后代就不会再用
        featurevalues = set(temp[:,bestfeature].T.tolist()[0])
        temp=np.delete(temp,bestfeature,axis=1)
        for value in featurevalues:
            sublabels = labels[:]
            tree[bestfeaturelabel][value] = createtree(splitdataset
(dataset,bestfeature,value),
sublabels,temp,majorclass(classlist))
        return tree
def getnumleaf(tree): #计算叶子个数
        numleaf = 0
        firststr = list(tree.keys())[0]
        secondict = tree[firststr]
        for key in secondict.keys():
            if type(secondict[key]).__name__ == 'dict':
#如果该结点不是叶子结点，就进行递归计算
                numleaf += getnumleaf(secondict[key])
            else:
#如果该结点是叶子结点，则叶子数量加一
                numleaf += 1
        return numleaf
def getdepth(tree): #计算树高
        maxdepth = 0
        firststr = list(tree.keys())[0]
```

```
    secondict = tree[firststr]
    for key in secondict.keys():
        if type(secondict[key]).__name__ == 'dict':
            tempdepth = 1 + getdepth(secondict[key])
#如果该结点不是叶子结点，则递归计算树高
        else:
#如果该结点是叶子结点，高度计为1
            tempdepth = 1
        if tempdepth>maxdepth:
#选择最高的高度作为树的最终高度
            maxdepth = tempdepth
    return maxdepth
#定义分类函数
def classify(tree,labels,test):
    firststr = list(tree.keys())[0]
    secondist = tree[firststr]
    featureindex = labels.index(firststr)
    for key in secondist.keys():
        if test[featureindex] == key:
#将数据划入对应的属性分支
            if type(secondist[key]).__name__ == 'dict':
#如果该结点不是叶子结点，则递归调用分类函数
                classlabel = classify(secondist[key],labels,test)
            else:
#如果该结点是叶子结点，则返回该结点的类别作为最终函数值
                classlabel = secondist[key]
    return classlabel
def storetree(tree,filename):
    import pickle  #使用该模块序列化对象
    fw = open(filename,'wb')  #pickle默认存储方式为二进制模式，故使用wb+
    pickle.dump(tree,fw)   #将tree序列化写入到路径中
    fw.close()
def gettree(filename):
```

```
    import pickle
    fr = open(filename,'rb') #同样地，由于二进制模式，使用rb
    tree = pickle.load(fr)  #反序列化，将其解析为一个python对象
    fr.close()
    return tree
#-------------------------测试样例 ----------------
fr = open('F:/机器学习实战/machinelearninginaction/Ch03/lenses.txt'
dataset = [x.strip().split('\t') for x in fr.readlines()]
traindataset = dataset[0:6]+dataset[8:14]+dataset[16:22]
globaldata = np.mat(traindataset)
testdataset = dataset[6:8]+dataset[14:16]+dataset[22:24]
featurelabels = ['age','symptom','astigmatic','tear']
finaltree = createtree(traindataset,featurelabels,globaldata)
#构造决策树
featurelabels = ['age','symptom','astigmatic','tear']
storetree(finaltree,'F:/毕业设计/code/ID3tree.txt') #存储树
predict = []
initial = [i[-1] for i in testdataset]
#将分类正确的样本个数站总样本数的百分比作为预测精度
for i in testdataset:
    predict.append(classify(finaltree,featurelabels,i))
pcorrect = sum([1 if predict[i]==initial[i] else 0
for i in range(len(predict))])/len(predict)
#-------------------------测试样例 ----------------

#-------------------------交叉验证 ----------------
crossans = []
#进行4折交叉验证
for j in range(24):
    if (j+18)<24:
        crosstraindata = dataset[j:j+18]
        crosstestdata = dataset[j+18:24]+dataset[0:j]
    else:
```

```
        crosstraindata = dataset[j:24]+dataset[0:j-6]
        crosstestdata = dataset[j-6:j]
    globaldata = np.mat(crosstraindata)
    crossfeaturelabels = ['age','symptom','astigmatic','tear']
    crossfinaltree = createtree(crosstraindata,crossfeaturelabels,
globaldata)  #构造决策树
    crossfeaturelabels = ['age','symptom','astigmatic','tear']
    crosspredict = []
    crossinitial = [i[-1] for i in crosstestdata]
#用过一次的属性特征将不会再在子树中出现
    for i in crosstestdata:
        crosspredict.append(classify(crossfinaltree,
crossfeaturelabels,i))
    crosspcorrect = sum([1 if crosspredict[i]==crossinitial[i]
else 0 for i in range(len(crosspredict))])/len(crosspredict)
    crossans.append(crosspcorrect)
print(sum(crossans)/len(crossans))
```

附录 B 剪枝 CART 算法

```
from math import log #导入科学计算库
import numpy as np #导入矩阵计算库
import re  #导入正则表达式库
filename =
'F:/机器学习实战/machinelearninginaction/Ch03/lenses.txt'
def loaddata(filename): #导入数据函数
    dataset = []
    fr = open(filename,'r')
    for line in fr.readlines():
        curline = line.strip().split('\t')
#将每一行数据按照空格分隔
        dataset.append(curline)
    return dataset
def splitdataset(dataset,axis,value):
#将数据集切分为两个部分，构建二叉树
    data1 = []
    data2 = []
    for i in dataset:
        datatemp = i[:axis] + i[axis+1:]
        if i[axis] == value:
            data1.append(datatemp)
        else:
            data2.append(datatemp)
    return data1,data2
def classgini(dataset):#计算类别的基尼指数
    s = 0.0
    samplenum = len(dataset)
    label = {}
    for i in dataset:
        temp = i[-1]
```

```
        if temp not in label.keys():  #不在就创建一个键
            label[temp] = 0
        label[temp] += 1  #在对应键的数目上加1
    for key in label.keys():
        prob = float(label[key])/samplenum
#将整数转化为浮点数保证结果为浮点数
        s += prob*prob
    gini = 1-s
    return gini
def choose(dataset):
    bestgini = np.inf #初始基尼指数为无穷大
    bestvalue = 0
    bestfeature = 0
    numfeatures = len(dataset[0])-1
    for i in range(numfeatures):
        feature = [example[i] for example in dataset]
        uniquefeature = set(feature)
#计算出样本中的所有属性取值
        for j in uniquefeature:
            data1,data2 = splitdataset(dataset,i,j)
#将数据集划分为两个子数据集
            newgini = (len(data1)/len(dataset))*
classgini(data1)+(len(data2)/len(dataset))*classgini(data2)
            if newgini<bestgini:
#如果当前基尼指数优于最佳基尼指数，则更换属性
                bestvalue = j
                bestfeature = i
                bestgini = newgini
    return bestfeature,bestvalue
def majorclass(classlist):
#选择出现次数最多的特征
    classcount = {}
    for vote in classlist:
```

```
        if vote not in classcount.keys():
            classcount[vote] = 0
            classcount[vote] += 1
    sortclass = sorted(classcount.items(),
key = lambda x:x[1],reverse = True)
#对出现的特征数目进行排序
    return sortclass[0][0]
def createtree(dataset,labels):
    classlist = [example[-1] for example in dataset]
#列出每个样本所属类别
    if classlist.count(classlist[0]) == len(classlist):
#如果当前样本中就只有一个类别，则最总类别就是该类别
        return classlist[0]
    if len(dataset[0]) == 1:  #所有特征已经遍历完，取多数为类
        return majorclass(classlist)
    bestfeature,bestvalue = choose(dataset)
    bestfeaturelabel = labels[bestfeature]
    tree = {(bestfeaturelabel+'_'+str(bestvalue)):{}}
    del(labels[bestfeature]) #用过一次的特征就删去
    data1,data2 = splitdataset(dataset,bestfeature,bestvalue)
    sublabels = labels[:]
#复制标签属性，不会改变原来标签集中的值
    tree[(bestfeaturelabel+'_'+str(bestvalue))]['yes'] =
createtree(data1,sublabels)
#构建决策树，左子树
    sublabels = labels[:]
    tree[(bestfeaturelabel+'_'+str(bestvalue))]['no'] =
createtree(data2,sublabels)
#构建决策树，右子树
    return tree
def classify(tree,labels,test):
#定义分类函数
    firststr = list(tree.keys())[0]
```

```
    secondist = tree[firststr]
    featureindex = labels.index(re.findall('([a-z]*)_',
firststr)[0])
    if test[featureindex] == re.findall('_([a-z]*)',firststr)[0]:
        if type(secondist['yes']).__name__ == 'dict':
            classlabel = classify(secondist['yes'],labels,test)
#如果该结点不是叶子结点，则递归调用分类函数
        else:
            classlabel = secondist['yes']
    else:
        if type(secondist['no']).__name__ == 'dict':
            classlabel = classify(secondist['no'],labels,test)
        else:
            classlabel = secondist['no']
    return classlabel
#------------------------------测试样例 -----------
dataset = loaddata(filename)
traindataset = dataset[0:6]+dataset[8:14]+dataset[16:22]
testdataset = dataset[6:8]+dataset[14:16]+dataset[22:24]
featurelabels = ['age','symptom','astigmatic','tear']
finaltree = createtree(traindataset,featurelabels)
featurelabels = ['age','symptom','astigmatic','tear']
def pre(finaltree,testdataset):
    predict = []
    initial = [i[-1] for i in testdataset]
    for i in testdataset:
        predict.append(classify(finaltree,featurelabels,i))
    pcorrect = sum([1 if predict[i]==initial[i] else
0 for i in range(len(predict))])/len(predict)
    return pcorrect
p = pre(finaltree,testdataset) #预测精度
#------------------------------测试样例 ------------
#------------------------------剪枝 -----------------
```

```
def istree(obj): #判断目标是否为树
    return (type(obj).__name__ == 'dict')
def prune(tree,testdata,traindata,labels):
    if len(testdata) == 0:
        return tree
#如果没有测试数据，则返回原树，这里默认原决策树未过拟合
        #return majorclass([example[-1] for example in traindata])
#返回最大类为最终值，这里默认原决策树过拟合
    firststr = list(tree.keys())[0]
    feature = re.findall('([a-z]*)_',firststr)[0]
    value = re.findall('_([a-z]*)',firststr)[0]
    if (istree(tree[firststr]['yes'])) or (istree(tree[firststr]['n
#如果左右子树有一颗不是叶子结点
        testdata1,testdata2 = splitdataset(testdata,
labels.index(feature),value)
        traindata1,traindata2 = splitdataset(traindata,
labels.index(feature),value)
        del(labels[labels.index(feature)])
    if (istree(tree[firststr]['yes'])):
#左子树不是叶子结点的情况
        sublabels = labels[:]
        tree[firststr]['yes'] = prune(tree[firststr]['yes'],
testdata1,traindata1,sublabels)
    if (istree(tree[firststr]['no'])):
#右子树不是叶子结点的情况
        sublabels = labels[:]
        tree[firststr]['no'] = prune(tree[firststr]['no'],
testdata2,traindata2,sublabels)
    if not istree(tree[firststr]['yes']) and
not istree(tree[firststr]['no']):
#寻找目标样本对应的所属分支，准备剪枝(都是叶子结点)
        errornomerge = pre(tree,testdata)
        temp = [i[-1] for i in traindata]
```

```
        templabel = majorclass(temp)
        errormerge = sum([1 if i[-1] == templabel else
0 for i in testdata])/len(testdata)
        if errormerge>errornomerge:
#剪枝后精度跟高，就剪枝
            print('merge')
            return templabel
        else:
            return tree
    else:
#若只有一个结点，返回该树
        return tree
#prunetree = prune(finaltree,testdataset,
traindataset,featurelabels)
#-----------------------------剪枝-------------
#-----------------------------交叉验证-----------
crossans = []
crossans2 = []
#4折交叉验证
for j in range(24):
    if (j+18)<24:
#构造验证集合和训练集合
        crosstraindata = dataset[j:j+18]
        crosstestdata = dataset[j+18:24]+dataset[0:j]
    else:
        crosstraindata = dataset[j:24]+dataset[0:j-6]
        crosstestdata = dataset[j-6:j]
    crossfeaturelabels = ['age','symptom','astigmatic','tear']
    crossfinaltree = createtree(crosstraindata,crossfeaturelabels)
#构造决策树
    crossfeaturelabels = ['age','symptom','astigmatic','tear']
    crossans.append(pre(crossfinaltree,crosstestdata))
#对CART决策树进行剪枝处理
```

```
    crossprunetree = prune(crossfinaltree,crosstestdata,
crosstraindata,crossfeaturelabels)
#计算精度
    crossans2.append(pre(crossprunetree,crosstestdata))
#输出估计精度
print(sum(crossans)/len(crossans))
print(sum(crossans2)/len(crossans2))
```