第六章 有监督学习

本章节将涉及以下方法：

* 决策树学习（Decision tree learning）-对胸痛患者的预先医疗护理指示
* 决策树学习-基于收入的房地产价值分布
* 决策树学习-预测股票走势方向
* 朴素贝叶斯（Naive Bayes）-预测股票走势方向
* 随机森林（Random forest）-外汇交易策略
* 支持向量机（Support vector machine）-外汇交易策略
* 随机梯度下降（Stochastic gradient descent）-成年收入

简介（Introduction）

决策树学习：决策树作为分类和问题预测的常用工具，是一种递归地划分实例空间或变量集的分类器。决策树被表示为树结构，其节点分为两类：叶节点或决策节点。叶节点用于保存目标属性的值，决策节点则指定在单个属性-值（attribute-value）上实现的规则。决策节点根据输入属性-值的特定离散函数将实例空间分割成两个或多个子空间。每个测试考虑单个属性，使得实例空间根据属性的值进行分区。例如，在数值属性的情况下，条件就指的是一个范围。在决策节点上实现规则后，对应的子树就是决策结果。每个叶节点保存一个概率向量，该向量指示了目标属性具有特定值的概率。根据沿路径测试的结果，由树的根节点到叶节点游历，从而对实例进行分类。

使用决策树挖掘数据的核心需求如下：

* 属性-值描述：目标可以被描述为一个特性或属性的固定集合
* 预定义级别：待分配类别的示例必须是有监督数据
* 足够的数据：使用多个训练样本

朴素贝叶斯：朴素贝叶斯是一种有监督学习的方法。作为一种基于贝叶斯理论的线性分类器，其理论指出，某一类的特定特征的存在与其它特征的存在无关。朴素贝叶斯是一种强大且高效的算法。贝叶斯分类器可以预测类成员概率，例如给定属于特定类的元组的概率。贝叶斯将网络视为联合条件概率分布。它允许在变量子集之间定义类条件独立性。它提供了可供学习的因果关系的图形模型。

随机森林：随机森林是对数据结构提供预测的决策树的集合。它们是一种在明智的随机化中汇集多个决策树的能力的工具，以及整体学习以产生预测模型。它们为每个记录提供可变排名、缺失值、分段和报告，以确保深入的数据理解。每棵树建成后，所有的数据都沿树自上而下运行。对于每一对情况均需对邻域进行计算。如果两种情况占用同一终端节点，则其邻域增加一个。在运行结束时，将结果除以树的数量来进行归一化。近似值用于替换丢失的数据、定位异常值并助于对数据的低维度理解。训练数据作为外包数据，用于评估分类误差并计算变量的重要性。

随机森林运行在大型数据库上可以非常有效地产生准确的结果。他们可以在不需要删除操作的情况下处理多个变量，并通过估计变量的重要性来解决分类问题。在森林构建的过程中，它们产生了对泛化误差的内部无偏估计。随机森林是估计丢失数据的有效方法，特别是在大部分数据丢失时依然能保持很高的准确性。

支持向量机（SVM）：机器学习算法解决学习问题需要使用正确的特征集。SVM利用（非线性）映射函数Φ，将输入空间的数据转换为特征空间中的数据，使得问题线性可分离。SVM查找最佳分离超平面并通过Φ-1映射回输入空间。在所有可能的超平面中，我们选择距离最接近的数据点（边距）尽可能大的超平面。

决策树学习：对胸痛患者的预先医疗护理指示

预先医疗护理指示的文档解释说明了未来对患者在各种医疗状况下的健康护理，引导患者在紧急情况下或根据需要做出正确的决定，帮助患者了解其医疗护理决定和指示的性质和后果、自由且自愿地做出这些决定，并以某种方式传达决定。

就绪

我们使用心脏病患者的数据集来搭建决策树分类。

步骤1-收集和描述数据

选用的数据集Heart.csv是一种可访问的CSV格式标准数据集，存储303行数据和15种变量，其中数字类变量包括：

* Age
* Sex
* RestBP
* Chol
* Fbs
* RestECG
* MaxHR
* ExAng
* Oldpeak
* Slope
* Ca

非数字类变量包括：

* ChestPain
* Thal
* AHD

开始

以下为实现细节。

步骤2-遍历数据

首先需要加载以下软件包：

**> install.packages("tree")**

**> install.packages("caret")**

**> install.packages("e1071")**

**> library(tree)**

**> library(caret)**

**版本信息：代码测试的R语言版本为3.3.0（2016-05-03）**

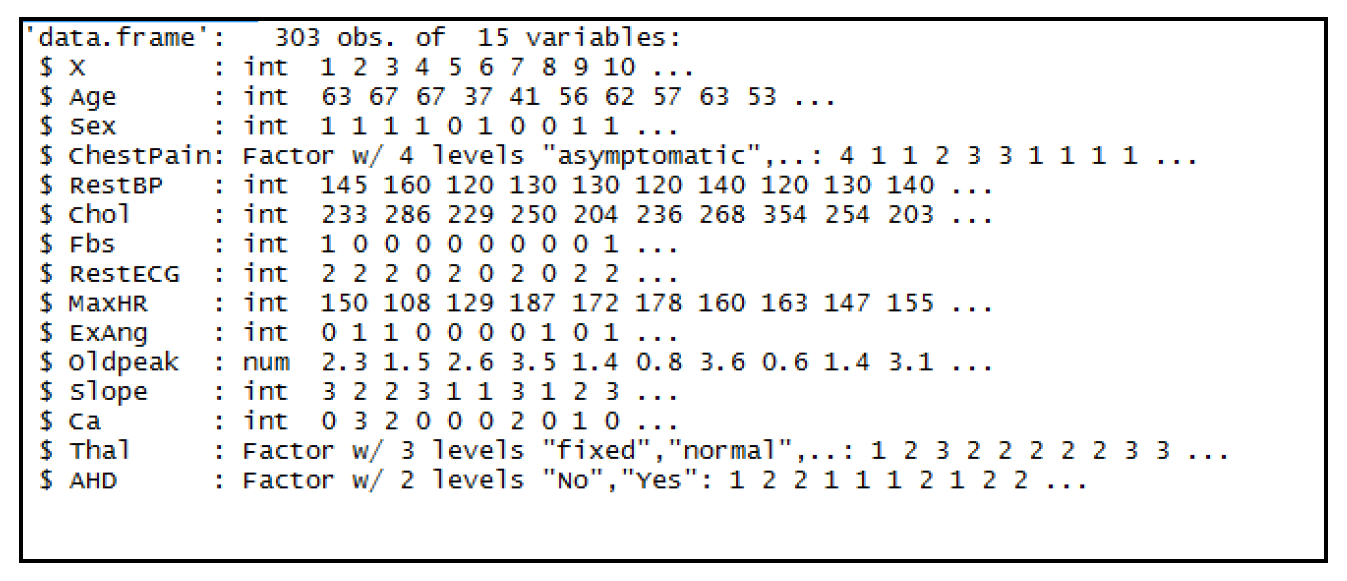
现在开始遍历数据并理解参数之间的关系。我们导入Heart.csv数据集并保存为AHD\_data数据帧：

**> AHD\_data <- read.csv("d:/Heart.csv", header = TRUE)**

**遍历AHD\_data数据帧的内部结构：**使用函数str()遍历作为R对象的AHD\_data数据帧的内部结构：

**> str(AHD\_data)**

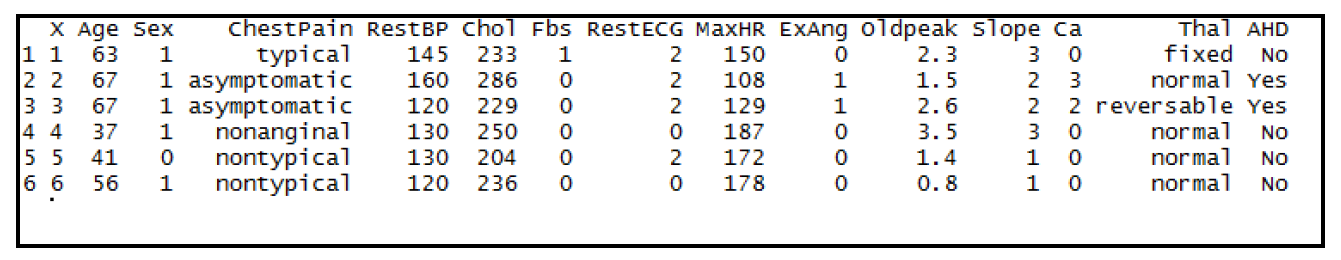
结果如下：



**输出AHD\_data数据帧：**使用函数head()返回AHD\_data数据帧的第一部分，AHD\_data作为函数的输入参数：

**> head(AHD\_data)**

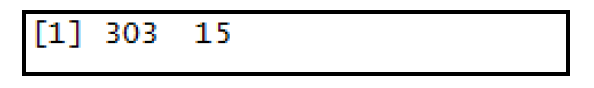
结果如下：



**遍历AHD\_data数据帧的维度**：使用dim()函数返回AHD\_data数据帧的维度，AHD\_data作为函数的输入参数。返回结果清楚的显示数据帧有303行和15列：

**> dim(AHD\_data)**

结果如下：



步骤3：准备数据

需要准备训练集数据分为两部分：一部分为训练数据，用于构建模型；另一部分为测试数据，用于测试已构建的模型。

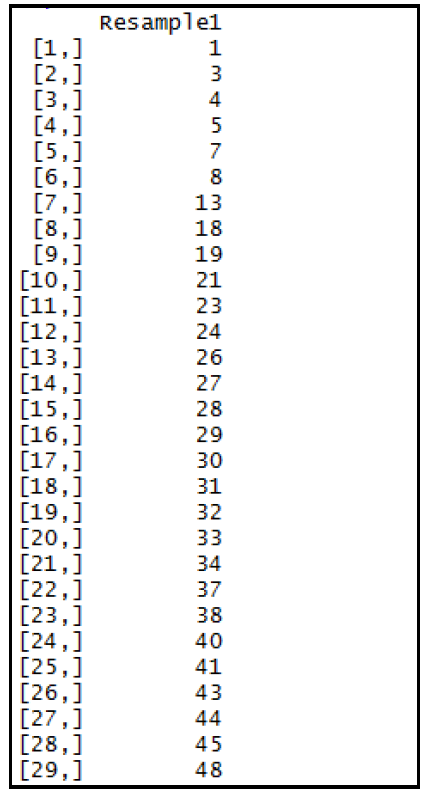
函数createDataPatition()用于数据分块，AHD\_data为函数的输入参数。函数采用随机抽样，参数p用于训练和构建模型的数据比例，本例中p=0.5，也就是说有50%的数据用于训练和构建模型。参数List=’False’避免返回结果为数据列表，而是存储在数据帧split中：

**> split <- createDataPartition(y=AHD\_data$AHD, p = 0.5, list=FALSE)**

调用数据帧split即可查看训练数据集：

**> split**

结果如下：



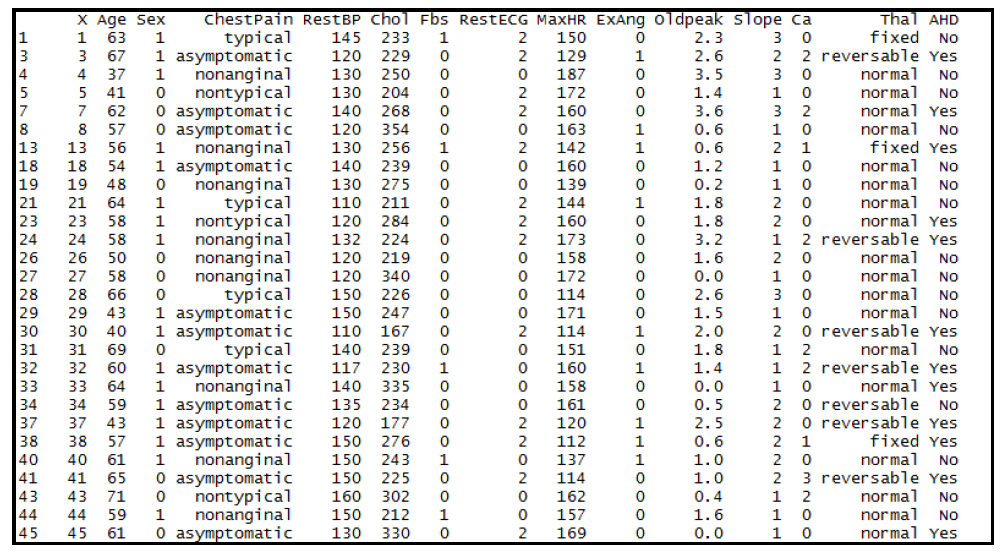
下面创建训练数据。使用split数据帧来创建训练数据并存储于train数据帧中：

**> train <- AHD\_data[split,]**

数据训练数据帧：

**> train**

结果显示如下：



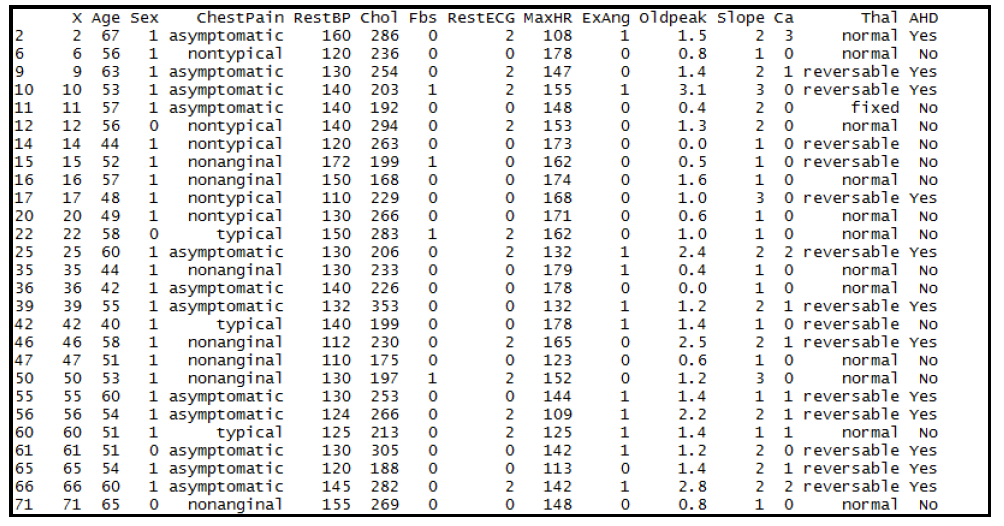
下面创建测试数据。同样利用split数据帧来建立测试数据。符号“-”代表获得所有split数据帧里面未被选用为训练数据的数据，即取训练数据的补集。使用test数据帧来存储测试数据：

> test <- AHD\_data[-split,]

输出测试数据帧：

**> test**

结果显示如下：



步骤4：训练模型

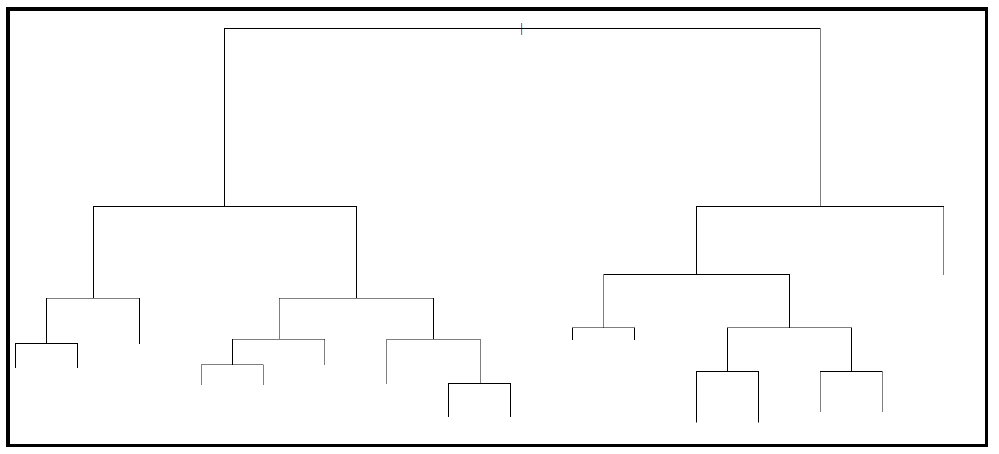
现在使用训练数据来训练模型。使用决策树来对比查看数字响应和描述符变量的关系，并将数据分为若干组，R语言中实现分类树的函数为tree()，通过二元递归分类的方法生出树。基于训练数据，使用AHD字段构建分类树，并将结果存储在trees数据帧中：

**> trees <- tree(AHD ~., train)**

使用函数plot()可以将生成的数据帧trees显示为图像，trees作为函数的输入参数：

> plot(trees)

结果显示如下：



使用函数cv.tree()，通过运行交叉验证实验来查找错误分类的数量。数据帧trees为函数的输入参数，FUN=prune.misclass代表通过递归地截取最不重要的分割来获得所提供的数据帧trees的子树的嵌套序列。输出结果存储于cv.trees数据帧中：

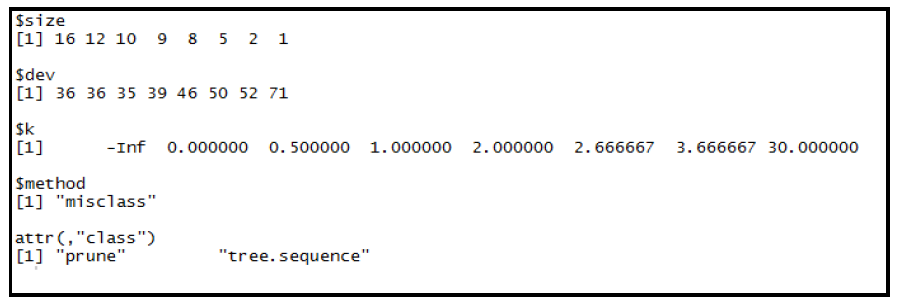
**> cv.trees <- cv.tree(trees, FUN=prune.misclass)**

输出数据帧cv.trees的结果：

**> cv.trees**

$ dev字段给出每个K的偏差。

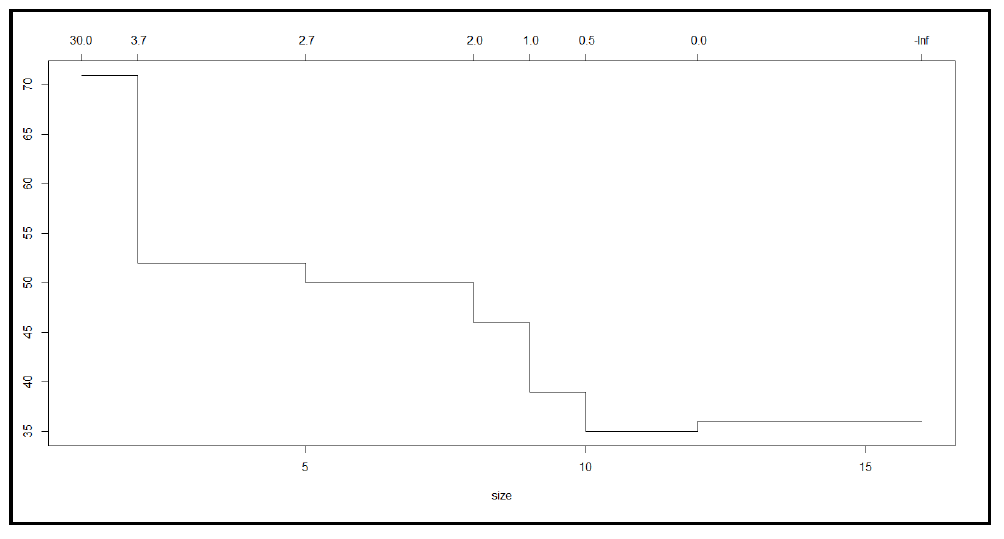
结果如下：



利用函数plot()来显示数据帧cv.trees。$ dev值位于y轴（右侧），$ k值位于顶部，$ size值位于x轴上。可以看出，当$ size = 1时，$ k = 30.000000，$ dev = 71（原文错）。使用以下命令显示数据帧cv.trees：

**> plot(cv.trees)**

结果显示如下：



步骤5-改善模型

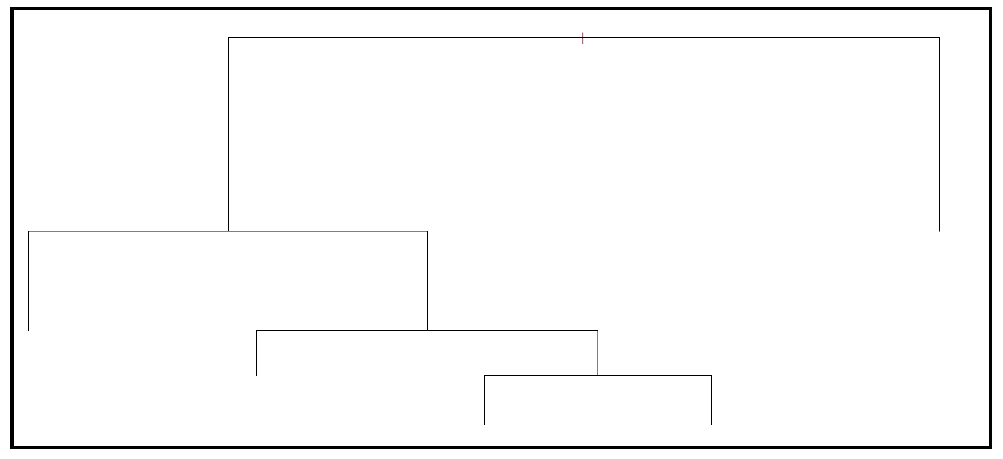
下面通过以最低偏差分裂树的方式来改进模型。调用prune.misclass()（前面提到，prune.misclass函数通过递归地截取最不重要的分割来获得提供的数据帧树的子树的嵌套序列。）函数来分割树。并将结果存储在数据帧prune.trees中。best = 4表示要返回的成本复杂度序列中特定子树的大小（例如，终端节点数）：

**> prune.trees <- prune.misclass(trees, best=4)**

使用plot()函数来显示prune.trees

**> plot(prune.trees)**

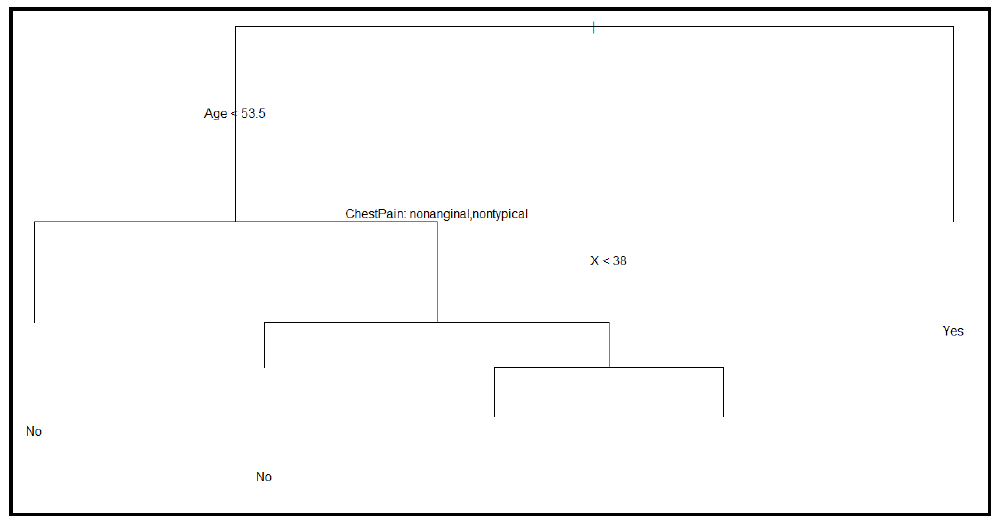
结果如下：



为prune.trees添加文本：

**> text(prune.trees, pretty=0)**

结果显示如下：



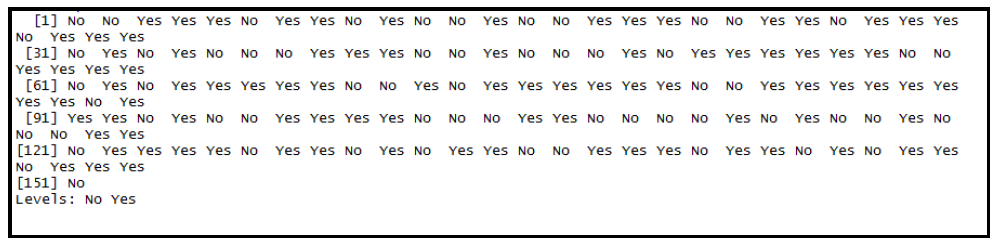
使用predict()函数用于基于线性模型对象来预测值，prune.trees作为函数的输入参数。数据对象test用于查找预测的变量。函数的输出结果将存储在数据帧tree.pred中：

**> tree.pred <- predict(prune.trees, test, type='class')**

显示变量test.pred的值：

**> tree.pred**

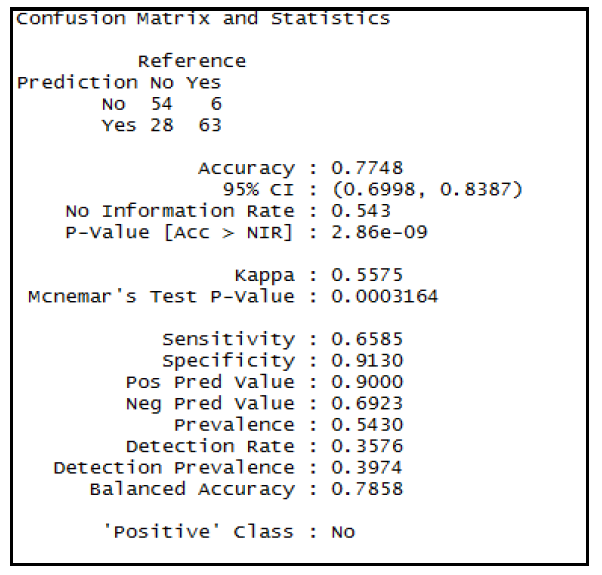
结果显示如下：



下面归纳模型的结果。函数confusionMatrix()用于计算被观察和预测的分类的交叉定标，在这里tree.pred作为被预测的分类：

**> confusionMatrix(tree.pred, test$AHD)**

结果显示如下：



决策树学习：基于收入的房地产价值分布

作为一种资产类别，收入值已经成为是房地产提供的有吸引力的长期总收益的重要组成部分。房地产投资产生的年收入回报率高于股票和50基点之后债券的2.5倍以上。房地产经常由租户支付的租金提供稳定的收入来源。

就绪

我们使用房地产的数据集来搭建决策树分类。

步骤1：收集和描述数据

使用的数据集RealEstate.txt为可访问的标准txt格式数据集，包含20640行数据和以下9种数字类变量：

* MedianHouseValue
* MedianIncome
* MedianHouseAge
* TotalRooms
* TotalBedrooms
* Population
* Households
* Latitude
* Longitude

开始

以下为实现细节。

步骤2-遍历数据

首先需要加载以下软件包：

**> install.packages("tree")**

**版本信息：代码测试的R语言版本为3.3.0（2016-05-03）**

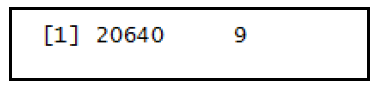
现在开始遍历数据并理解参数之间的关系。我们导入RealEstate.txt数据集并保存为realEstate数据帧：

**> realEstate <- read.table("d:/RealEstate.txt", header=TRUE)**

**遍历realEstate数据帧的维度**：使用dim()函数返回realEstate数据帧的维度，realEstate作为函数的输入参数。返回结果清楚的显示数据帧有20640行和9列：

**> dim(realEstate)**

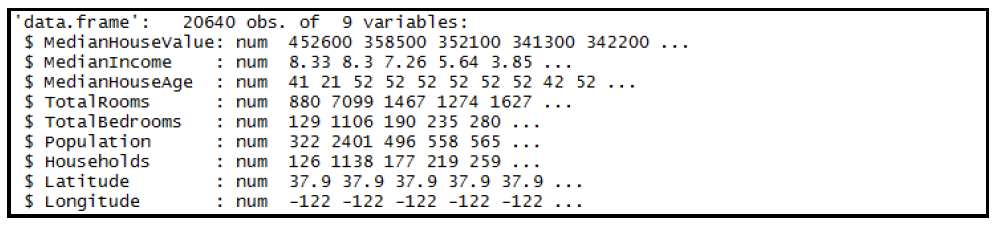
结果如下：



**遍历realEstate数据帧的内部结构：**使用函数str()遍历作为R对象的realEstate数据帧的内部结构，realEstate作为函数的输入参数：

**> str(realEstate)**

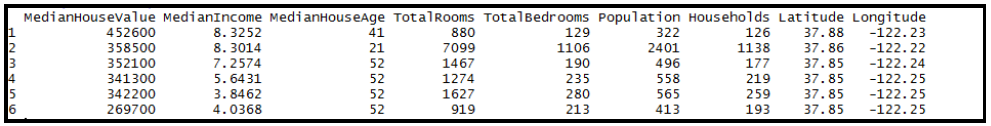
结果如下：



**输出realEstate数据帧：**使用函数head()返回realEstate数据帧的第一部分，realEstate作为函数的输入参数：

**> head(realEstate)**

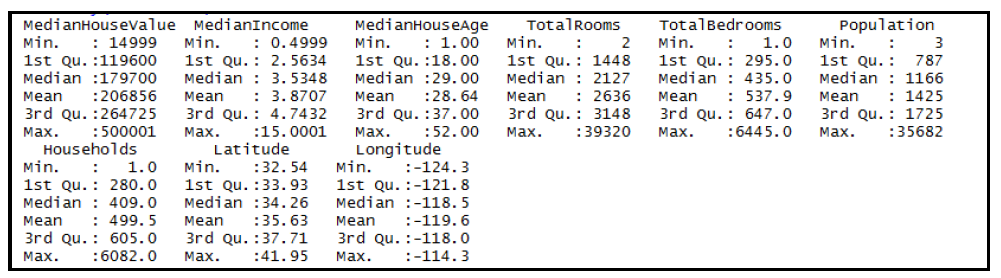
结果如下：



**输出realEstate数据帧的摘要：**summary()函数是一个多用途通用函数，提供与单个对象或数据帧相关的数据的摘要。realEstate数据帧summary()函数的输入参数：

**> summary(realEstate)**

结果如下：



步骤3：训练模型

现在开始准备数据集的模型。作为分类和预测的工具，决策树代表人类可以理解并用于知识系统（如数据库）中的规则，并通过从树的根节点开始直到到达叶节点的过程来分类实例。 节点指定对单个属性的测试，叶节点则表示目标属性的值，边则分离一个属性。

函数tree()采用二元递归分割生成树的方法来实现分类树。这些模型是计算密集型技术，因为它们根据一个或多个预测变量的关系将响应变量递归地分解为子集。

公式表达式基于变量Latitude和Longitude之和，求和结果存储在日志变量MedianHouseValue中。 data = realEstate表示优先解释公式，权重和子集的数据帧。结果存储在数据帧treeModel中：

**> treeModel <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data=realEstate)**

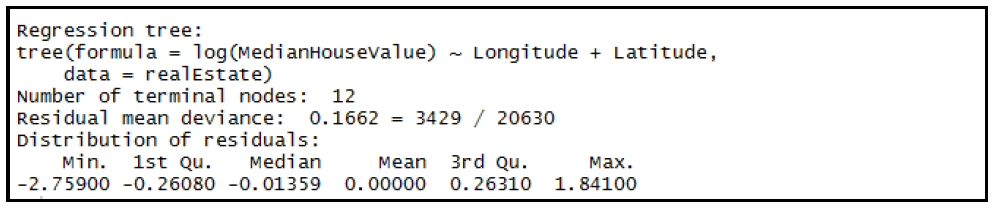
下面显示treeModel的摘要。摘要显示所使用的公式、树中的终端节点或叶节点的数量、以及残差的统计分布。

函数summary()用于显示treeModel的统计摘要。它是一个泛型，用于生成各种拟合函数的结果摘要。将需要显示摘要的数据帧treeModel作为输入参数传递。

在这里，偏差是指均方误差：

**> summary(treeModel)**

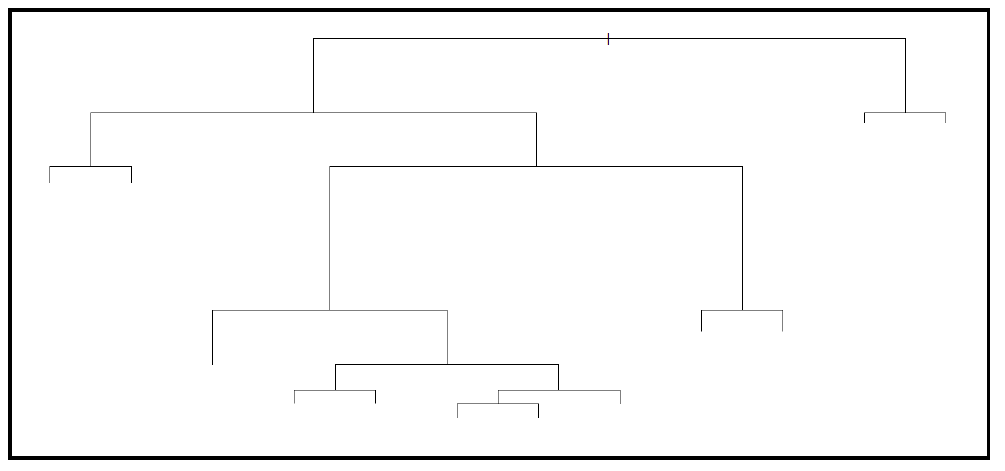
结果如下：



使用函数plot()可以将生成的数据帧treeModel显示为图像，treeModel作为函数的输入参数：

**> plot(treeModel)**

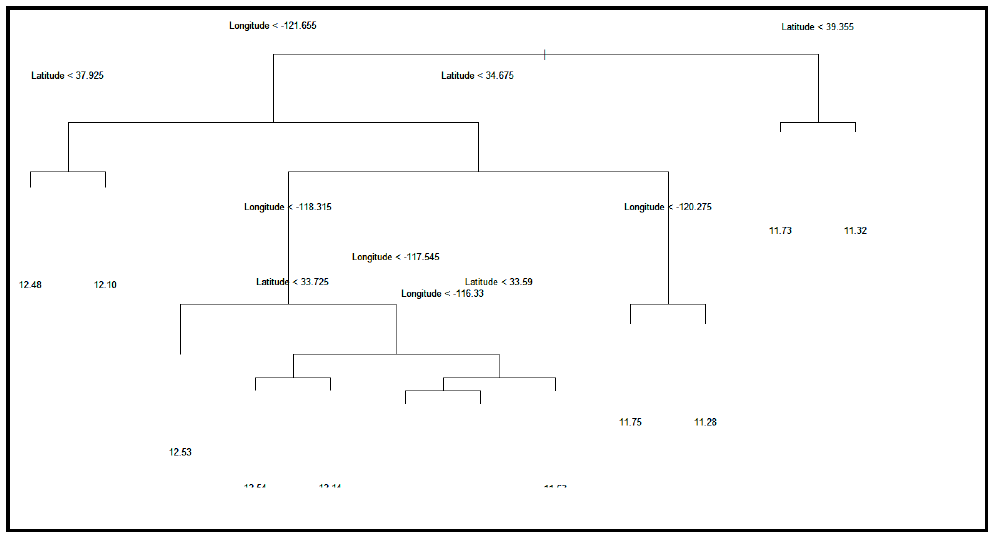
结果如下：



需要为显示为图像的treeModel数据帧的每一个节点和叶节点的值添加文本信息。使用函数text()在给定坐标处插入标签向量中给出的字符串：

**> text(treeModel, cex=.75)**

结果如下：



步骤4：对比预测结果

与反映全球价格走势的数据集对比预测结果。为了更方便的报告或比较，我们希望归纳MedianHouseValue的频率分布，最直接的方法是使用分位数。分位数是与数值排名相关分布中的一个点。分位数将划分MedianHouseValue分布，使得在分位数之下存在给定比例的观测结果。

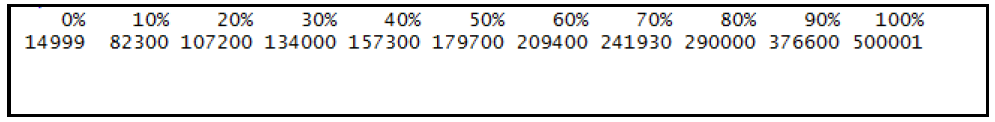
函数quantile()产生与给定概率相对应的样本分位数。realEstate$MedianHouseValue是使用样本分位数的数字向量。函数quantile()返回priceDeciles作为向量的长度：

> priceDeciles <- quantile(realEstate$MedianHouseValue, 0:10/10)

查看数据帧priceDeciles的值：

**> priceDeciles**

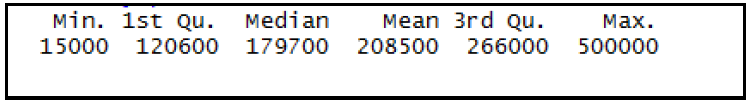
结果如下：



接下来使用summary()函数来显示priceDeciles的摘要，priceDeciles作为函数的输入参数：

**> summary(priceDeciles)**

结果如下：



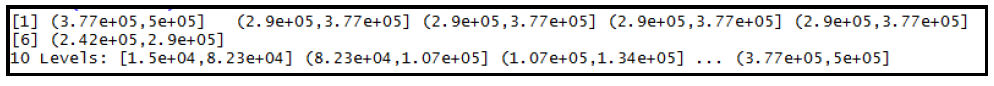
下面使用函数cut()根据所属的间隔划分间隔范围，将priceDeciles划分为两个区间。数据帧realEstate的数字向量MedianHouseValue将通过切换转换为一个因子：

**> cutPrices <- cut(realEstate$MedianHouseValue, priceDeciles, include.lowest=TRUE)**

输出数据帧cutPrices。使用函数head()返回cutPrices数据帧的第一部分，cutPrices作为函数的输入参数：

**> head(cutPrices)**

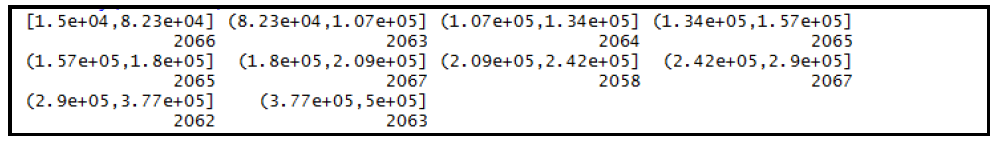
结果如下：



下面使用函数summary()来显示数据帧cutPrices的摘要，cutPrices作为函数的输入参数：

**> summary(cutPrices)**

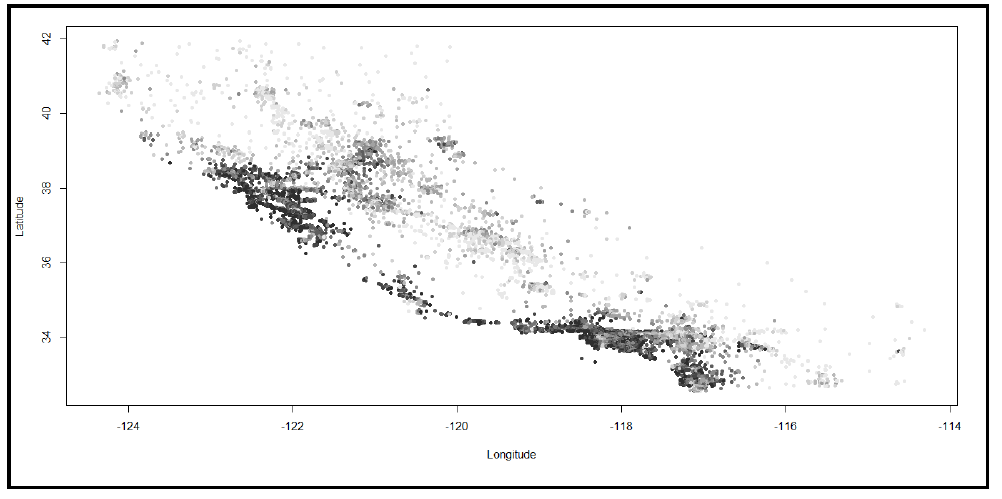
结果如下：



使用函数plot()绘制cutPrices的值，数据帧cutPrices作为函数的输入参数。realEstate数据集的Longitude变量表示图中x坐标，Latitude变量表示图中y坐标。col=gray（10:2/11）表示绘图颜色。pch=20表示在绘制符号的默认大小。xlab =“Longitude”表示x轴的标题，而ylab =“Latitude”表示y轴的标题：

> plot(realEstate$Longitude, realEstate$Latitude, col=grey(10:2/11)[cutPrices], pch=20, xlab="Longitude",ylab="Latitude")

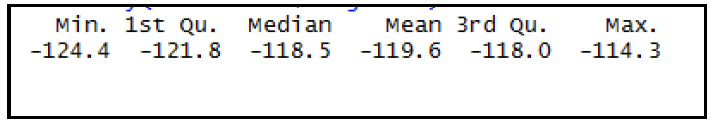
结果如下：



使用函数summary()显示Longitude的统计摘要：

> summary(realEstate$Longitude)

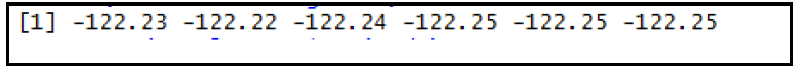
结果如下：



使用函数head()输出Longitude数据帧的第一部分：

**> head(realEstate$Longitude)**

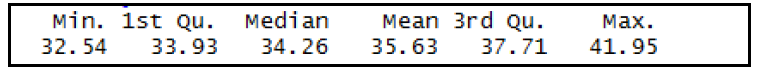
结果如下：



使用函数summary()显示Latitude的统计摘要：

**> summary(realEstate$Latitude)**

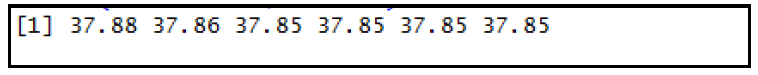
结果如下：



使用函数head()输出Latitude数据帧的第一部分：

**> head(realEstate$Latitude)**

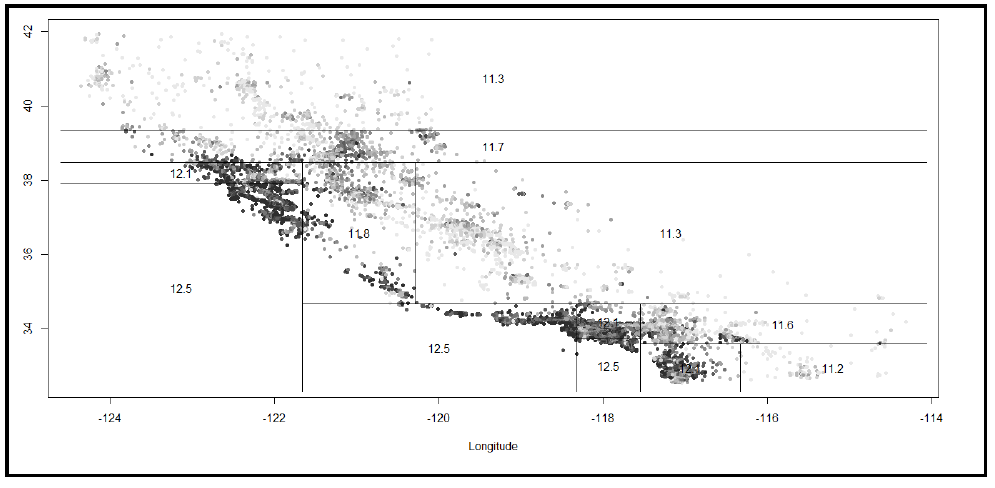
结果如下：



函数partition.tree用于划分涉及两个或多个变量的树，树对象treeModel作为函数的输入参数。ordvars = c(“Longitude”，“Latitude”)表示变量用于绘制的顺序。Longitude表示x轴，而Latitude表示y轴。add = TRUE表示添加到已有图像：

**> partition.tree(treeModel, ordvars=c("Longitude","Latitude"), add=TRUE)**

结果如下：



步骤5：改善模型

树中的叶节点数量控制了树的灵活性。叶节点数量表示树被分割成多少个单元。每个节点必须包含一定数量的点，并且添加节点必须至少减少一定量的错误。min.dev的默认值为0.01。

下面我们将min.dev的值降低至0.001。

使用函数tree()来实现分类树。公式表达式基于变量Latitude和Longitude之和，求和结果存储在日志变量MedianHouseValue中。 data = realEstate表示优先解释公式，权重和子集的数据帧。min.dev的值表示必须是要分割节点的根节点至少0.001倍的偏差。

结果存储于treeModel2数据帧中：

**> treeModel2 <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data=realEstate, mindev=0.001)**

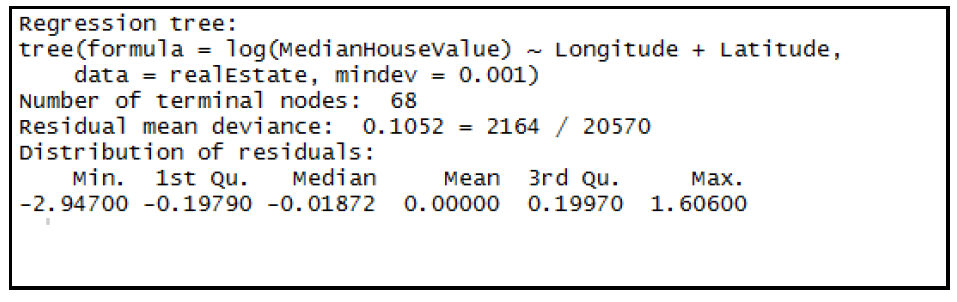
下面显示treeModel2的摘要。摘要显示所使用的公式、树中的终端节点或叶节点的数量、以及残差的统计分布。

函数summary()用于显示treeModel2的统计摘要。将需要显示摘要的数据帧treeModel2作为输入参数传递。

在这里，偏差是指均方误差：

**> summary(treeModel2)**

结果如下：

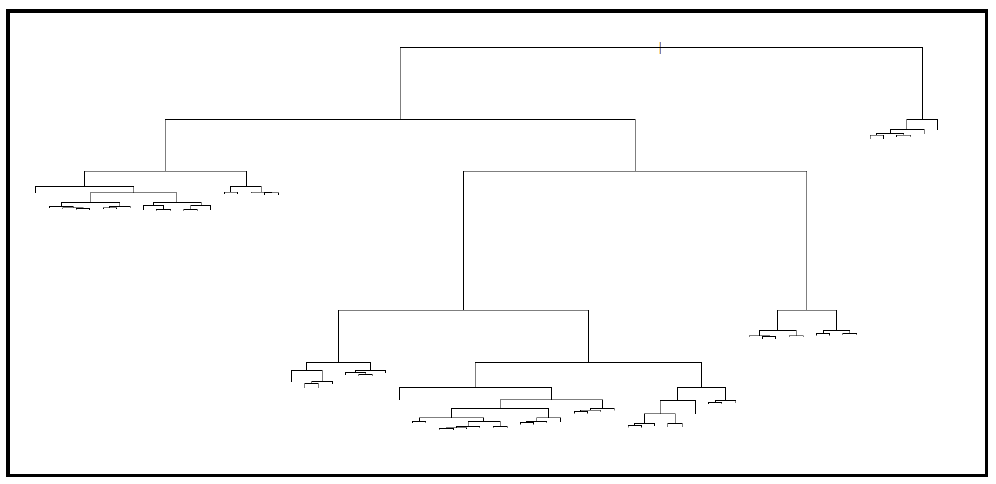


与treeModel的摘要相比，treeModel2中的叶节点值从12增加到68。偏差值从treeModel模型的0.1666变为treeModel2的0.1052。

使用通用的绘制函数plot()可以将生成的数据帧treeModel2显示为图像，treeModel2作为函数的输入参数：

**> plot(treeModel2)**

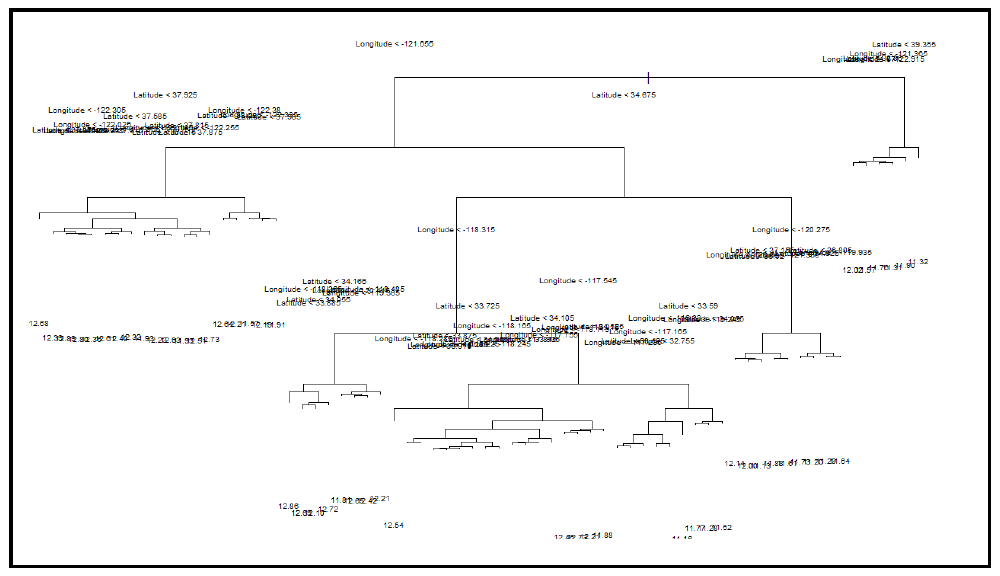
结果如下：



需要为显示为图像的数据帧treeModel2的每一个节点和叶节点的值添加文本信息。使用函数text()在给定坐标处插入标签向量中给出的字符串：

**> text(treeModel2, cex=.65)**

结果如下：



包含公式扩展中的所有变量。

函数tree()实现了分类树。公式表达式基于所有变量。

结果存储于数据帧treeModel3中：

> treeModel3 <- tree(log(MedianHouseValue) ~ ., data=realEstate)

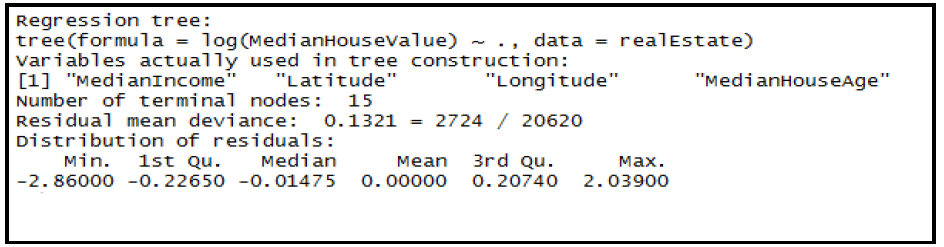
下面显示treeModel3的摘要。摘要显示所使用的公式、树中的终端节点或叶节点的数量、以及残差的统计分布。

函数summary()用于显示treeModel3的统计摘要。将需要显示摘要的数据帧treeModel3作为输入参数传递。

在这里，偏差是指均方误差：

**> summary(treeModel3)**

结果如下：

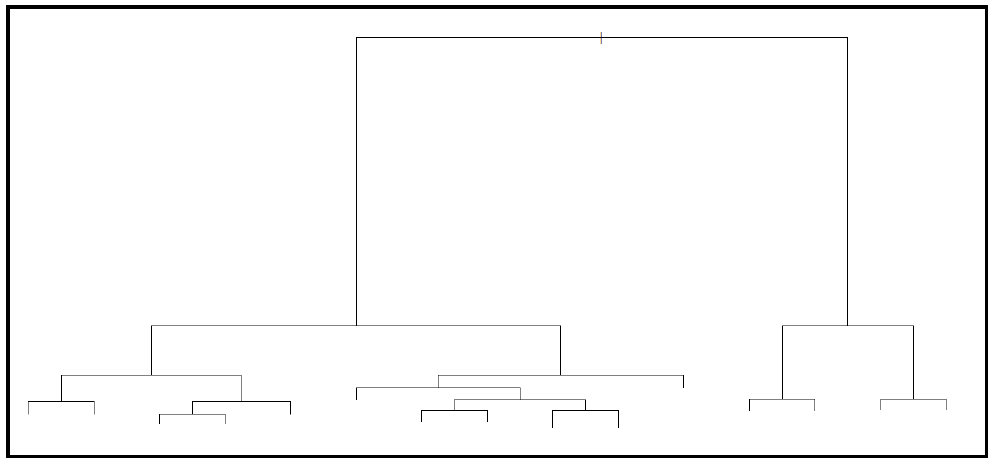


公式表达式清楚显示了realEstate数据集的所有变量。

使用通用的绘制函数plot()可以将生成的数据帧treeModel3显示为图像，treeModel3作为函数的输入参数：

**> plot(treeModel3)**

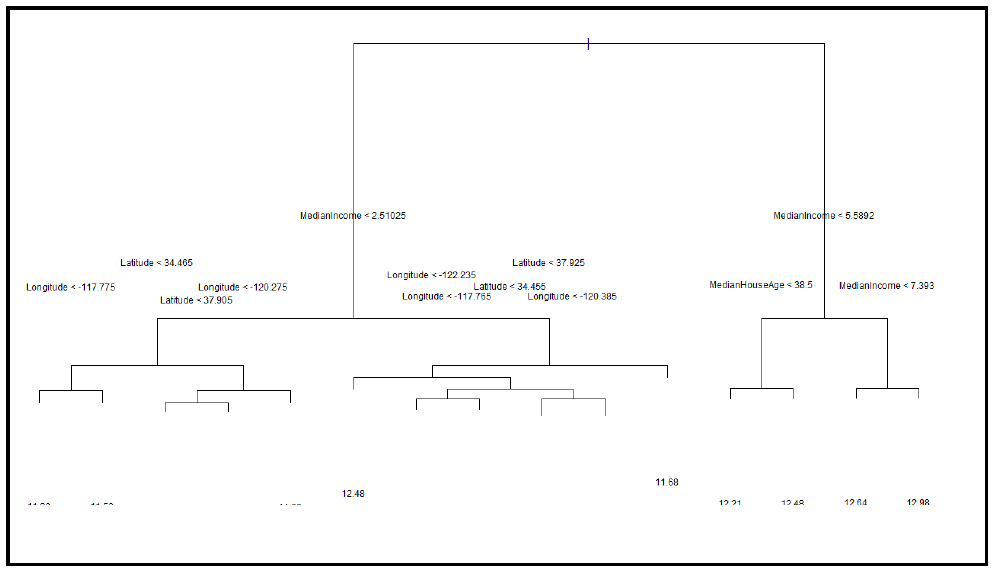
结果如下：



需要为显示为图像的数据帧treeModel3的每一个节点和叶节点的值添加文本信息。使用函数text()在给定坐标处插入标签向量中给出的字符串：

**> text(treeModel3, cex=.75)**

结果如下：



决策树学习：预测股票走势方向

股票交易是统计人员试图解决的最具挑战性的问题之一。有诸多的技术指标，例如走势方向、市场活力是否缺乏，盈利潜力波动、以及市场中监测流行度的容量措施等。这些指标可为高可靠性交易机会创造策略。发现技术指标之间的关系可能需要花费几天/周/月的时间，因此需要使用诸如决策树之类的高效省时的工具。决策树的主要优点是它是一个强大而易于解释的算法。

就绪

我们使用股票市场的数据集来搭建决策树分类。

步骤1：收集和描述数据

数据集选取美国银行在2012年1月1日至2014年1月1日之间的每日收盘价值。我们将要下载的数据集可以在以下网址免费获得：<http://yahoo.com>。

开始：

以下为实现细节。

步骤2：遍历数据

首先需要加载以下数据包：

**> install.packages("quantmod")**

**> install.packages("rpart")**

**> install.packages("rpart.plot")**

**版本信息：代码测试的R语言版本为3.3.0（2016-05-03）**

需要安装以上数据包的相应库：

**> library("quantmod")**

**> library("rpart")**

**> library("rpart.plot")**

现在开始标记所需数据时间段的起始时间和结束时间并下载数据。