Machine Learning 2019

袁欣

2019 年 3 月 29 日

1 决策树

1.1 西瓜数据

从 csv 文件中读取西瓜数据，并进行展示。

- 代码如下：

```
wmda <- read.csv(file = "西瓜数据2.0.csv")
wmda[, 8] <- as.factor(wmda[, 8])
```

- 数据展示：

```
knitr::kable(wmda)
```

编号	色泽	根蒂	敲声	纹理	脐部	触感	label
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	1
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	1
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	1
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	1
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	1
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	1
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	1
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	1
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0
10	青绿	硬挺	清脆	清晰	平坦	软粘	0
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0

编号	色泽	根蒂	敲声	纹理	脐部	触感	label
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0

1.2 决策树简介

1.2.1 基本概念

决策树（decision tree）是一种基本的分类与回归方法。这里主要讨论分类树。决策树模型呈树形结构，在分类问题中，表示基于特征对实例进行分类的过程。它可以认为是 if-then 规则的集合，也可以认为是定义在特征空间与类空间上的条件概率分布。其主要优点是模型具有可读性，分类速度快。学习时，利用训练数据，根据损失函数最小化的原则建立决策树模型。预测时，对新的数据，利用决策树模型进行分类。

决策树学习通常包括 3 个步骤：特征选择、决策树的生成和决策树的修剪。这些决策树学习的思想主要来源于 Quinlan 在 1986 年提出的 ID3 算法和 1993 年提出的 C4.5 算法，以及由 Breiman 等人在 1984 年提出的 CART 算法。

1.2.2 划分选择

- 信息增益

“信息熵”（information entropy）是度量样本集合纯度常用的一种指标。假定当前样本集合 D 中第 k 类样本所占的比例为 $p_k(k=1,2,\ldots,y)$，则 D 的信息熵定义为

$$Ent(D)=-\sum_{i=1}^{y}p_k log_2 p_k$$

$Ent(D)$ 的值越小，则 D 的纯度越高。

信息增益是指离散属性 a 对原始样本 D 进行分类，从而使信息熵下降的值。可以用如下公式表示：

$$Gain(D,a)=Ent(D)-\sum_{v=1}^{V}\frac{D^v}{D}Ent(D^v)$$

其中 V 表示 a 共有 V 种属性，$\frac{D^v}{D}$ 对不同的属性进行了加权。一般而言，信息增益越大意味着使用属性 a 进行划分所获得的“纯度提升”越大。因此，我们可以用信息增益进行决策树的划分。

- 编程实现

```
TreeGenerate <- function(da, method = "ID3"){

  # Input
  #    da : 1(ID), 1:n-1(attribute), n(label)
  # Output
  #    tree
```

```r
CalEnt <- function(pk){
  Entd <- -(pk * log2(pk))
  Entd[is.na(Entd)] <- 0
  Entd <- sum(Entd)
  return(Entd)
}

tree <- list()
# compute boot node Entd
pvec <- as.numeric(table(da[, ncol(da)])) / nrow(da)
Entd <- CalEnt(pvec)
tree[[1]] <- list(a = "boot", aname = "boot", boot = da[, 1], Entd = Entd)
a <- 2 # count tree
dalist <- list(da = list(da = da), count = 1)
treelast <- list()
treenew <- list()
# Generate tree
while(length(dalist$da) > 0){
  Retlist <- list()
  k <- 1  # count dalistnew
  dalistnew <- list()
  for(t in 1:length(dalist$da)){
    #
    count <- dalist$count
    dat <- dalist$da[[t]]
    # compute boot node Entd
    pvec <- as.numeric(table(dat[, ncol(dat)])) / nrow(dat)
    Entd <- CalEnt(pvec)
    # choose attribute
    AEnt <- rep(0, length = ncol(dat) - 2)
    IV <- AEnt
    Gini <- rep(0, length = ncol(dat) - 2)
    ARetlist <- list()
    for(i in 2:(ncol(dat)-1)){
      avec <- levels(as.factor(dat[, i]))
      aretlist <- list()
      for(j in 1:length(avec)){
        aj.boot <- dat[which(dat[, i] == avec[j]), 1]
        pvec <- table(dat[which(dat[, 1] %in% aj.boot), ncol(dat)])
```

```
      pvec <- as.numeric(pvec) / length(aj.boot)
      AEnt[i-1] <- AEnt[i-1] + CalEnt(pvec) * length(aj.boot) / nrow(dat)
      IV[i-1] <- IV[i-1] - length(aj.boot) / nrow(dat) *
        log2(length(aj.boot) / nrow(dat))
      Gini[i-1] <- Gini[i-1] + (1-sum(pvec^2)) * length(aj.boot) / nrow(dat)
      aretlist[[j]] <- list(a = colnames(dat)[i], aname = avec[j],
                            boot = aj.boot, Entd = CalEnt(pvec))
    }
    ARetlist[[i-1]] <- aretlist
  }
  # good attribute
  ########## ID 3 ##############
  if(method == "ID3"){
    ret <- ARetlist[[which(AEnt == min(AEnt))[1]]]
  }
  ######### C4.5 ############
  if(method == "C4.5"){
    ind <- which(AEnt <= mean(AEnt))
    #
    Gr <- (1 - AEnt[ind]) / IV[ind]
    ret <- ARetlist[[ind[which(Gr == max(Gr))[1]]]]
  }
  ######## Gini ############
  if(method == "Gini"){
    ret <- ARetlist[[which(Gini == min(Gini))[1]]]
  }

  #
  retlist <- list()

  #           # print Tree last
  if(count == 1){
    print(paste("||", "Boot", " {" , paste(da[, 1], collapse = ",") ,"} ",
                "Ent = ", round(tree[[1]]$Entd, 3), sep = ""))
  }else{
    print(paste(paste(rep("..", count-1), collapse = "."), count-1, ")",
                tail(treelast[[t]]$a, 1), "=", tail(treelast[[t]]$aname, 1),
                " {" , paste(treelast[[t]]$boot, collapse = ",") ,"} ",
                sep = ""))
```

```
    }

    for(i in 1 : length(ret)){
      # Get the previous level of classification
      # use treelast
      if(count == 1){
        treelast <- tree
      }
      classe <- as.character(da[which(da[, 1] %in% ret[[i]]$boot), ncol(da)])
      classe <- table(as.numeric(classe))

      if(ret[[i]]$Entd == 0 | length(ret[[i]]$boot) == 0){
         # print Tree
         print(paste(paste(rep("..", count), collapse = "."), count, ")",
                     ret[[i]]$a, "=", ret[[i]]$aname,
                     " {", paste(ret[[i]]$boot, collapse = ","), "} ",
                     ifelse(names(which(classe == max(classe)))[1] == 1,
                            "Good", "Bad"), sep = ""))
         #
        retlist[[i]] <- list(a = c(treelast[[t]]$a, ret[[i]]$a),
                             aname = c(treelast[[t]]$aname, ret[[i]]$aname),
                             boot = ret[[i]]$boot, Entd = ret[[i]]$Entd,
                             class = names(which(classe == max(classe)))[1])
       }

      if(ret[[i]]$Entd != 0 & length(ret[[i]]$boot) != 0){
        dalistnew[[k]] <- dat[which(dat[, 1] %in% ret[[i]]$boot),
                              -which(colnames(dat) == ret[[i]]$a)]
        # truenew
        treenew[[k]] <- list(a = c(treelast[[t]]$a, ret[[i]]$a),
                             aname = c(treelast[[t]]$aname, ret[[i]]$aname),
                             boot = ret[[i]]$boot, Entd = ret[[i]]$Entd,
                             class = names(which(classe == max(classe)))[1])
        k <- k + 1
      }
    }
    Retlist[[t]] <- retlist
  } # end for (dalist)
  count <- count + 1
```

```
    tree[[count]] <- Retlist
    dalistnew <- list(da = dalistnew, count = count)
    dalist <- dalistnew
    treelast <- treenew
  } # end while
  return(tree)
}
Tree <- TreeGenerate(wmda, method = "ID3")
```

```
## [1] "||Boot {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17} Ent = 0.998"
## [1] "..1)纹理=模糊 {11,12,16} Bad"
## [1] "..1)纹理=清晰 {1,2,3,4,5,6,8,10,15} "
## [1] ".....2)根蒂=蜷缩 {1,2,3,4,5} Good"
## [1] ".....2)根蒂=硬挺 {10} Bad"
## [1] "..1)纹理=稍糊 {7,9,13,14,17} "
## [1] ".....2)触感=软粘 {7} Good"
## [1] ".....2)触感=硬滑 {9,13,14,17} Bad"
## [1] ".....2)根蒂=稍蜷 {6,8,15} "
## [1] ".........3)色泽=青绿 {6} Good"
## [1] ".........3)色泽=乌黑 {8,15} "
## [1] "...........4)触感=软粘 {15} Bad"
## [1] "...........4)触感=硬滑 {8} Good"
```

- 增益率

信息增益准则对可取值数目较多的属性有所偏好，为减少这种偏好可能带来的不利影响，Quinlan 在 1993 年提出了 C4.5 算法，使用“增益率”（gain ratio）来选择最优划分属性。增益率的定义为：

$$Gain\_ratio(D,a) = \frac{Gain(D,a)}{IV(a)}$$

其中

$$IV(a) = -\sum_{v=1}^{V} \frac{D^v}{D} log_2 \frac{D^v}{D}$$

属性 a 的可能取值数目越多，则 $IV(a)$ 的值通常会越大。增益率准则对可能取值数目较少的属性有所偏好。C4.5 算法并不是直接选择增益率最大的候选划分属性，而是使用了一个启发式规则：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

- 编程实现

生成树 `TreeGenerate()` 函数中已经包含 C4.5 算法，仅需将 `method` 参数改为“C4.5”即可。

```
Tree <- TreeGenerate(wmda, method = "C4.5")
```

```
## [1] "||Boot {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17} Ent = 0.998"
## [1] "..1)纹理=模糊 {11,12,16} Bad"
## [1] "..1)纹理=清晰 {1,2,3,4,5,6,8,10,15} "
## [1] ".....2)触感=硬滑 {1,2,3,4,5,8} Good"
## [1] "..1)纹理=稍糊 {7,9,13,14,17} "
## [1] ".....2)触感=软粘 {7} Good"
## [1] ".....2)触感=硬滑 {9,13,14,17} Bad"
## [1] ".....2)触感=软粘 {6,10,15} "
## [1] ".........3)色泽=乌黑 {15} Bad"
## [1] ".........3)色泽=青绿 {6,10} "
## [1] "...........4)根蒂=稍蜷 {6} Good"
## [1] "...........4)根蒂=硬挺 {10} Bad"
```

- 基尼系数

CART 决策树 [Breimaan et al. 1984] 使用“基尼系数”（Gini index）来选择划分属性。数据集 D 的纯度可以用基尼值来度量：

$$Gini(D) = \sum_{k=1}^{y} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{y} p_k^2$$

基尼系数越小则数据集的纯度越高。属性 a 的基尼系数定义为：

$$Gini_index(D, a) = \sum_{v=1}^{V} \frac{D^v}{D} Gini(D^v)$$

- 编程实现

```
Tree <- TreeGenerate(wmda, method = "Gini")
```

```
## [1] "||Boot {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17} Ent = 0.998"
## [1] "..1)纹理=模糊 {11,12,16} Bad"
## [1] "..1)纹理=清晰 {1,2,3,4,5,6,8,10,15} "
## [1] ".....2)根蒂=蜷缩 {1,2,3,4,5} Good"
## [1] ".....2)根蒂=硬挺 {10} Bad"
## [1] "..1)纹理=稍糊 {7,9,13,14,17} "
## [1] ".....2)触感=软粘 {7} Good"
## [1] ".....2)触感=硬滑 {9,13,14,17} Bad"
## [1] ".....2)根蒂=稍蜷 {6,8,15} "
## [1] ".........3)色泽=青绿 {6} Good"
## [1] ".........3)色泽=乌黑 {8,15} "
```

```
## [1] ".....4)触感=软粘 {15} Bad"
## [1] ".....4)触感=硬滑 {8} Good"
```

通过对比发现，ID3 与 CART 算法得到了相同得决策树。C4.5 算法得到的决策树在第二层中选择了较少类别的属性（触感）。

1.2.3 剪枝处理

决策树剪枝的基本策略有“预剪枝”（prepruning）和“后剪枝”（postpruning）[Quinlan,1993]。

- 预剪枝

预剪枝是指在决策树生成过程中，对每个结点在划分前先进行估计，若当前节点的划分不能带来决策树泛化能力提升，则停止划分并将当前结点标记为叶节点。（需要划分训练集与测试集，根据准确率进行性能评估）

- 代码实现

首先划分训练集与测试集：

```
wmda.train <- wmda[c(1,2,3,6,7,10,14:17), c(1,6,2,3,4,5,7,8)]
wmda.test <- wmda[c(4,5,8,9,11:13),  c(1,6,2,3,4,5,7,8)]
```

没有预剪枝的决策树为：

```
Tree.train <- TreeGenerate(wmda.train, method = "ID3")
```

```
## [1] "||Boot {1,2,3,6,7,10,14,15,16,17} Ent = 1"
## [1] "..1)脐部=平坦 {10,16} Bad"
## [1] "..1)脐部=凹陷 {1,2,3,14} "
## [1] ".....2)色泽=浅白 {14} Bad"
## [1] ".....2)色泽=青绿 {1} Good"
## [1] ".....2)色泽=乌黑 {2,3} Good"
## [1] "..1)脐部=稍凹 {6,7,15,17} "
## [1] ".....2)根蒂=蜷缩 {17} Bad"
## [1] ".....2)根蒂=稍蜷 {6,7,15} "
## [1] ".....3)色泽=青绿 {6} Good"
## [1] ".....3)色泽=乌黑 {7,15} "
## [1] ".....4)纹理=清晰 {15} Bad"
## [1] ".....4)纹理=稍糊 {7} Good"
```

剪枝后的树为：

```
Tree <- TreeGenerate2(wmda.train, wmda.test, method = "ID3")
```

```
## [1] "||Boot {1,2,3,6,7,10,14,15,16,17} Ent = 1"
## [1] "..1)脐部=平坦 {10,16} Bad"
## [1] "..1)脐部=凹陷 {1,2,3,14} "
## [1] "Class:Good"
## [1] "No increase in accuracy, terminate classification"
## [1] "..1)脐部=稍凹 {6,7,15,17} "
## [1] "Class:Bad"
## [1] "No increase in accuracy, terminate classification"
```

- 后剪枝

先生成整棵树，通过判断剪枝后的精度是否提高进行剪枝处理。后剪枝决策树通常比预剪枝决策树保留了更多的分支，一般情况下，后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝决策树。

1.2.4 连续值处理

C4.5 算法中采用二分法（bi-partition）对连续属性进行处理。给定样本集 D 和连续属性 a，假定 a 在 D 上出现了 n 个不同得取值，将这些值排序后按照如下规则进行划分，选取最优划分点进行样本集合得划分。

$$T_a = \{\frac{a^i + a^{i+1}}{2} | 1 \leqslant i \leqslant n-1\}$$

- 编程实现

```
wmda <- read.csv(file = "西瓜数据3.0.csv")
wmda[, 10] <- as.factor(wmda[, 10])
knitr::kable(head(wmda, 2))
```

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	label
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	1
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	1

```
Tree <- TreeGenerate3(wmda)
```

```
## [1] "||Boot {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17} Ent = 0.998"
## [1] "..1)纹理=模糊 {11,12,16} Bad"
## [1] "..1)纹理=清晰 {1,2,3,4,5,6,8,10,15} "
## [1] ".....2)密度=<= 0.3815 {10,15} Bad"
## [1] ".....2)密度=> 0.3815 {1,2,3,4,5,6,8} Good"
## [1] "..1)纹理=稍糊 {7,9,13,14,17} "
```

```
## [1] ".....2)触感=软粘 {7} Good"
## [1] ".....2)触感=硬滑 {9,13,14,17} Bad"
```

1.2.5 多变量决策树

多变量决策树把每个属性视为坐标空间中的一个坐标轴，则 d 个属性描述的样本对应了 d 维空间的一个数据点，对样本分类则意味着在这个坐标空间中寻找不同样本之间的分类边界。具体理论不再详细阐述。