

Rと樹木モデル(2)

1. 樹木モデル関数 rpart と mvpart

関数 `rpart`(recursive partitioning and regression trees)は1997年 Terry Therneau と Beth Atkinson により実装され、関数 `mvpart` は `rpart` のモデルの当てはめとプロットのための包括関数で、オーストラリア海洋研究所の Glenn De'ath より実装された。

関数 `rpart` の書き方を次に示す。より多くの引数やデフォルトの設定などは `help(rpart)` で確認することができる。

```
rpart(formula, method, ...)
```

`formula` の書き方は、`tree` と同じである。引数 `method` は、表1の中からひとつを選択することができる。引数 `method` が指定されていない場合は、`formula` に指定されている応答変数(被説明変数)のデータ型から判断する。

表1 引数 `method` のオプション

| 応答変数 <code>y</code> | 引数の書き式 |
|-------------------------|-------------------------------|
| <code>y</code> が一般の量的変数 | <code>method="anova"</code> |
| <code>y</code> がポアソン分布 | <code>method="poisson"</code> |
| <code>y</code> が質的変数 | <code>method="class"</code> |
| <code>y</code> が行列 | <code>method="mrt"</code> |
| <code>y</code> が距離 | <code>method="dist"</code> |
| <code>y</code> が生存データ | <code>method="exp"</code> |

関数 `rpart` を使用するためには `rpart`、あるいは `mvpart` を読み込まなければならない (`library(rpart)`、`library(mvpart)`)。

関数オブジェクト `rpart` のクラスに属する主なメソッド関数を次の表に示す。

表2 `rpart` クラスの主な関数

| 関数名 | 主な機能 |
|------------------------------|--------------|
| <code>plot.rpart</code> | 樹木の図示 |
| <code>text.rpart</code> | 文字列の操作 |
| <code>plotcp</code> | 交差確認の結果の図示 |
| <code>predict</code> | モデルによる当てはめ |
| <code>print.rpart</code> | 樹木の出力 |
| <code>printcp</code> | 複雑さのパラメータの出力 |
| <code>prune.rpart</code> | 樹木の剪定を行う |
| <code>residuals.rpart</code> | 残差を返す |
| <code>rpart.control</code> | 樹木をコントロール |
| <code>summary.rpart</code> | 要約を返す |

2. 分類木

関数 `rpart` では、分岐の基準としては Gini 係数

$$\text{Gini Index} = 1 - \sum_k p_{ik}^2$$

あるいは情報量エントロピー

$$\text{Entropy} = - \sum_k p_{ik} \log(p_{ik})$$

が用いられている。式の中の p_{ik} はノード i 内のクラス k のデータである。

デフォルトでは Gini 係数が設定されている。分岐基準エントロピーの指定は、引数 `split="information"` を用いる。

(1) 樹木の作成

関数 `rpart` を用いた樹木の作成について例を用いて説明する。ここでも `iris` のデータを用いることにする。

```
>library(mvpart) #あるいは library(rpart)
> iris.rp<-rpart(Species~.,data=iris)
```

上記のコマンドの実行で、デフォルトに設定された条件の下で `rpart` の分類木が作成される。作成された樹木の結果は、次のように関数 `print` で返すことができる。関数の `print` に用いた `digit` は返す値の小数点以下の桁数を指定する引数である。返される結果は `digit` の数値より一桁多い。

```
>print(iris.rp,digit=2)
n= 150
node), split, n, loss, yval, (yprob)
* denotes terminal node
 1) root 150 100 setosa (0.333 0.333 0.333)
 2) Petal.Length< 2.5 50 0 setosa (1.000 0.000 0.000) *
 3) Petal.Length>=2.5 100 50 versicolor (0.000 0.500 0.500)
 6) Petal.Width< 1.8 54 5 versicolor (0.000 0.907 0.093)
 12) Petal.Length< 5 48 1 versicolor (0.000 0.979 0.021) *
 13) Petal.Length>=5 6 2 virginica (0.000 0.333 0.667) *
 7) Petal.Width>=1.8 46 1 virginica (0.000 0.022 0.978) *
```

この結果の樹木のグラフは、次のように関数 `plot` と `text` を用いて作成することができる。

```
> plot(iris.rp,uniform=T,branch=0.6,margin=0.05)
> text(iris.rp,all=T,use.n=T)
```

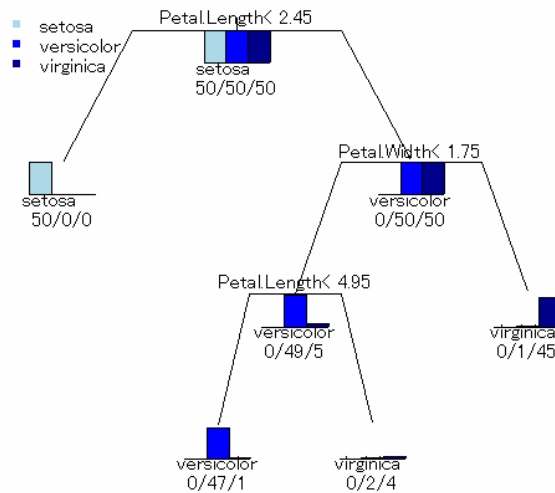


図1 rpartによるirisの分類木

関数 `plot.rpart` には樹木のデザインのための多くの引数が用意されている。

上記のコマンドに用いた `uniform` はノード間の間隔に関する引数である。デフォルトは `uniform=FALSE` で、木のノードの間隔は分類のエラーの数に比例するように作成する。引数を `uniform=TRUE` にすると、ノードの間隔は等間隔となる。

引数 `branch` は枝の角度を調整する。指定する値は 0 から 1 までである。0 の場合の角度が最も大きく、1 の場合は垂直となる。デフォルトは `branch=1` になっている。

引数 `margin` は図と 4 辺との間の余白(マージン)を調整する。引数 `margin` の値が大きいほど図が小さくなり、余白が大きい。

関数 `text` にも複数の引数が用意されている。上記のコマンドで用いた引数 `use.n` は、各ノードに含まれる個体の数を表示するか、しないかを指定する。デフォルトには `use.n=FALSE` になっている。引数が `use.n=TRUE` である場合は、ノードに含まれる数を表示する。

(2) 樹木の剪定

関数 `rpart` は樹木を成長させると同時に、交差確認法の結果も計算する。関数 `printcp` でその結果を返す。デフォルトの `n` 重交差確認の `n` は `xval=10` に設定されている。

関数 `printcp` は、樹木の剪定のための複雑さのパラメータ(`cp`)を返す。この結果は、用いたデータセットをランダムに分割して交差確認を行った結果であるので、同じデータを用いて同じ条件のもとで `rpart` を繰り返しても、まったく同じの結果になるとは限らない。

```
>printcp(iris.cp)
```

```

Classification tree:
rpart(formula = Species ~ ., data = iris)

Variables actually used in tree construction:
[1] Petal.Length Petal.Width

Root node error: 100/150 = 0.66667

n= 150

  CP nsplit rel error xerror  xstd
1 0.50     0    1.00  1.16 0.051277
2 0.44     1    0.50  0.70 0.061101
3 0.02     2    0.06  0.07 0.025833
4 0.01     3    0.04  0.07 0.025833

```

通常、樹木のサイズは **xerror** の最小値からその標準偏差 1 倍の範囲内の最大の **xerror** 値を選ぶ。これを **Min+1SE** 方法と呼ぶことにする。

上記に返された最終行の **xerro**、**xstd** はそれぞれ 0.07、0.0258 であり、**Min+1SE=0.07+0.258=0.0958** となる。この値は、第 2 行の **xerror=0.7** より小さく、第 3 行の **xerror=0.07** より大きいので、第 3 行の **cp(complexity parameter)=0.02** がひとつの目安となる。

関数 **rpart** では、複雑度 **cp** で樹木をコントロールすることができる。**cp** の値が小さいほど樹木が複雑になる。デフォルトは **cp=0.01** になっている。

関数 **plotcp** は、樹木の選定に必要となる交差確認に関する情報のプロットを返す。

```
>plotcp(iris.rp)
```

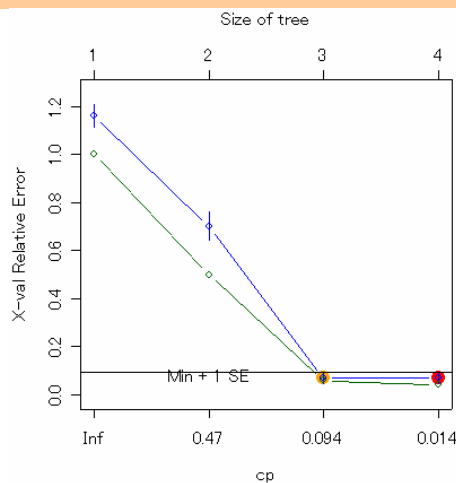


図 2 iris.rp の plotcp のプロット

グラフの下部の横軸が **cp** で、上部の横軸が樹木の葉の数で示すサイズである。図の中の水平直線 **Min+1SE** は、**xerror** の最小値に 1 倍の標準偏差を足した値である。その直線の下方に最も近い点をオレンジ色で、**xerror** の最小値の点を赤い色で示している。オレンジ色の点が樹木を剪定する目安である。

図 2 では **cp=0.094**、樹木のサイズが 3 に対応するところにオレンジ色の点が位置している。

この結果と **plotcp** での剪定の目安 **cp=0.02** が一致していないことに疑問があるに違いない。これは、**printcp** では分岐の数と **cp** の対応関係で、**plotcp** は樹木のサイズと **cp** の対応関係であるからである。樹木の剪定にはどの結果を用いても良い。

図 3 に関数 **prune** を用いて **cp=0.02** で剪定した **rpart** の樹木を示す。

```
>iris.rp1<-prune(iris.rp,cp=0.02)
> plot(iris.rp1,uniform=T,branch=0.6)
> text(iris.rp1,use.n=T)
```

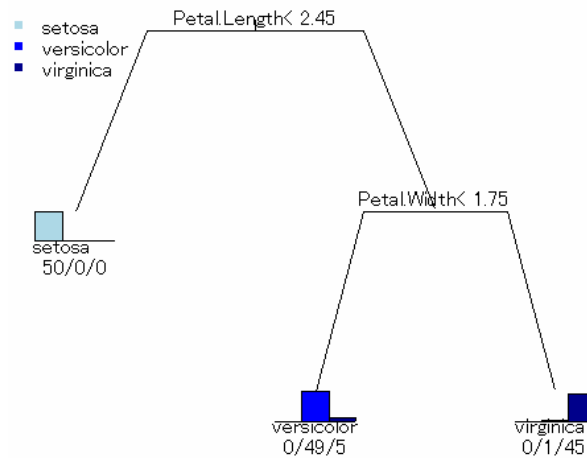


図 3 cp=0.02 で剪定した iris の分類木

(3) 樹木モデルによる予測

作成した樹木のモデルを用いて未知のデータについて予測・判別を行うためには、モデルの作成に用いていないデータが必要である。そこでここでも関数 `tree` を説明するときと同じく `iris` データを奇数行と偶数行に分けて学習用と予測のテスト用にする。

```
>even.n<-2*(1:75)-1
>train.data<-iris[even.n,]
>test.data<-iris[-even.n,-5]
>iris.lab<-factor(c(rep("S",25),rep("C",25),rep("V",25)))
>train.data[,5]<-iris.lab
>iris.rp2<-rpart(Species~.,train.data)
>plotcp(iris.rp2)
```

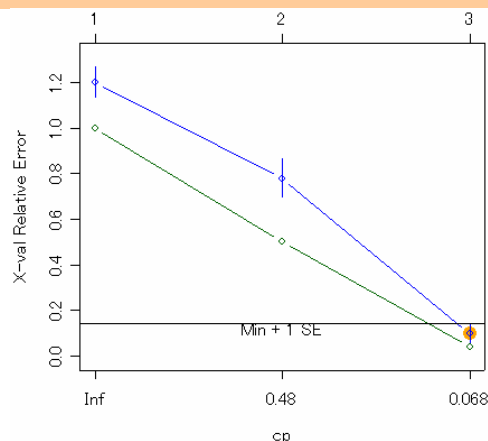


図 4 iris の奇数行の `printcp` プロット

上記の結果から、さらに剪定する必要がないと判断し、作成したモデルを用いて予測・判別を行う。

```
>iris.rp2r<-predict(iris.rp2,test.data,type="class")
> table(iris.lab,iris.rp2r)
iris.rp2r
```

```
iris.lab C S V
      C 24 0 1
      S  0 25 0
      V  3 0 22
```

(4) 樹木の剪定とコントロール

関数 `rpart` が返す樹木は関数 `rpart.control` によりコントロールされている。その引数の説明は、`help(rpart.control)` で確認できる。その主な引数の設定を説明するため、`rpart.control` のデフォルトの設定を次に示す。

```
rpart.control(minsplit=5, minbucket= round(minsplit/3), cp=0.01, maxcompete=4,
maxsurrogate=5, usesurrogate=2, xval=10,
surrogatestyle=0, maxdepth=30, ...)
```

引数 `minsplit` は分岐をする際に、ノードに含まれる最少の個体の数をコントロールする。デフォルトは 5 に設定されている。値が小さいほど大きな木が作成される。

引数 `minbucket` は葉における最少の個体数をコントロールする。デフォルトは `minsplit` の三分の一になっている。よって、`minsplit` と `minbucket` の中でひとつのみが指定されているときには、自動的に $3 * \text{minbucket} = \text{minsplit}$ によって計算される。

引数 `cp` (complexity parameter) はすでに説明したように、複雑さのパラメータである。明らかに価値のない木が成長し過ぎることを制御するためのパラメータである。デフォルトは 0.01 になっている。値が小さいほど樹木が茂る。

`maxcompete`、`maxsurrogate`、`usesurrogate`、`surrogatestyle` は分岐の候補、欠損値に関して設定を行う引数である。これらの結果は、`summary` から確認できる。

引数 `xval` は n 重交差確認の n を設定する。デフォルトは 10 になっている。通常 3~10 の値を指定する。

引数 `maxdepth` は、樹木の最大の深さをコントロールする。根の深さは 0 とカウントされる。デフォルトは 30 になっているが、32 ビットマシンではこれ以上設定することは意味がないであろう。

関数 `rpart.control` の引数は、関数 `rpart` で直接引数として用いることができる。

3. 回帰木

回帰木で用いる分岐基準は、実測値 y_k とこのセルの平均値 μ_k との差の 2 乗の和である。

$$D = \sum_k (y_k - \mu_k)^2$$

ここでも実データを用いて回帰木モデルの作成などを説明する。関数 `tree` の説明と同じくデータ `cars` を用いることにする。

データ `cars` は 50 個体、2 変数のデータフレームである。変数は、車の速度 (`speed`) とブレー

キを掛けた後車が停車するまでの距離(dist)の計測値である。ここでは速度(speed)を説明変数、距離(dist)を被説明変数とする。

(1) 回帰木の作成

関数 `rpart` による回帰木の作成は、関数 `tree` の書き式と同じである。

```
>(cars.rp<-rpart(dist~speed,data=cars))
```

<紙面の都合で返される結果は省略する>

返される結果から、回帰木は6の葉を持っていることが分かる。

(2) 回帰木の剪定

まず関数 `plotcp` を用いて成長させた回帰木について剪定の必要性について考察する。

```
> plotcp(cars.rp)
```

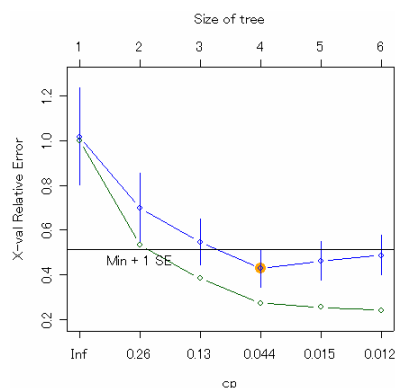


図5 回帰木の `plotcp` プロット

図5は成長させた回帰木を4つの葉に剪定する必要があることを示唆している。

次のように関数 `prune` を用いることにより4つの葉に剪定される。

```
> cars.rp1<-prune(cars.rp,cp=0.044)
```

```
>plot(cars.rp1,uniform=T,margin=0.05)
```

```
>text(cars.rp2,use.n=T)
```

<紙面の都合で返される結果は省略する>

この回帰木は関数 `tree` を用いて作成した結果と同じである。

通常、作成したモデルを用いて予測を行うときには、モデル作成に使用していない説明変数のデータを用いて予測値を求める。関数 `rpart` では、`predict` を用いる。その書き式は分類木の場合と同じであるのでここでは省略する。残差は関数 `residuals` で返される。

4. 多変量回帰木

多変量回帰(multivariate regression)は、被説明変数が複数である回帰である。

ここでは、パッケージ `mvpart` に同梱されているデータ `spider` を用いて多変量回帰木を説明する。

データ `spider` は 12 種類の蜘蛛狩りの分布と砂丘周辺の環境の特徴に関する 28 行 18 列のデータフレームである。

データの初めの 12 列は 12 種類の蜘蛛狩りに関する数値データの分布で、それに続く 6 列は砂丘周辺の環境の特徴に関するデータである。

ここでは砂丘周辺の環境の特徴に関する 6 列のデータを説明変数とし、12 種類の蜘蛛狩りの多変量回帰木を作成する例を示す。関数 `rpart` を用いて多変量回帰木を作成することもできるが、関数 `mvpart` を用いた方がもっと便利である。

関数 `rpart`、`mvpart` を用いて多変量回帰木を成長させる際には、用いた多変量の応答変数(被説明変数)をマトリックス形式にする必要がある。関数 `mvpart` には `rpart` より豊富な引数が用意されている。

その全てを紹介する紙面がないので、主な引数の使用方法を例示する。

関数 `mvpart` では、直接交差確認法の情報に基づいて剪定を行った樹木のグラフを返すように設定されている。もちろん、その設定は後術の引数を用いて替えることができる。

次にデータ `spider` を用いた多変量回帰木を作成する。

```
>data(spider)
>spider[1,13:18]
  water sand moss reft twigs herbs
1     9    0    1    1    9    5
>fit.mv<-mvpart(as.matrix(spider[,1:12])~water+twigs+reft+herbs+moss+sand,spider)
```

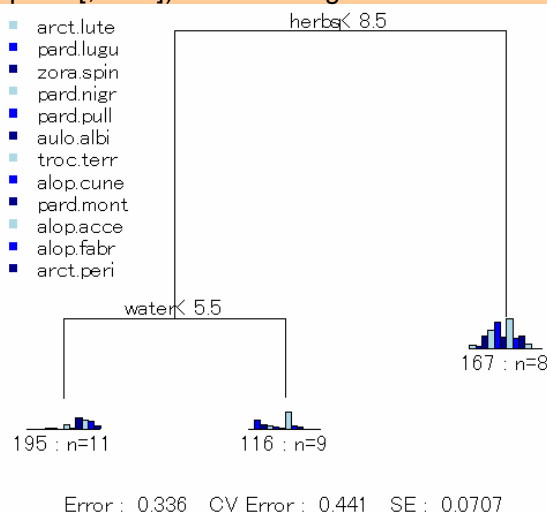


図 6 spider の多変量回帰木

どのような木を返すかは次の引数で指定することができる。

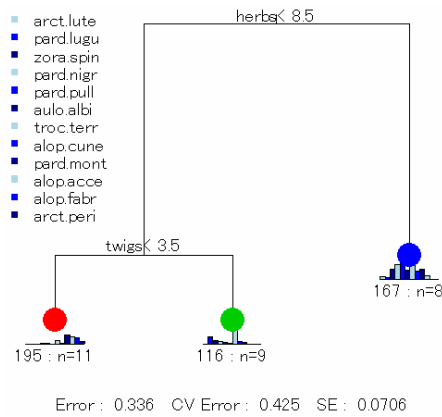
```
xv = c("lse", "min", "pick", "none")
```

引数 `xv = "lse"` は Min+1SE の方法による木を返し、`xv = "min"` は最も当てはめの良い木を返す。
`xv = "pick"` は `rpart` における `printcp` と同じのグラフと木を返し、`xv = "none"` は交差確認を行わない木を返す。デフォルトでは `xv = "lse"` になっている。

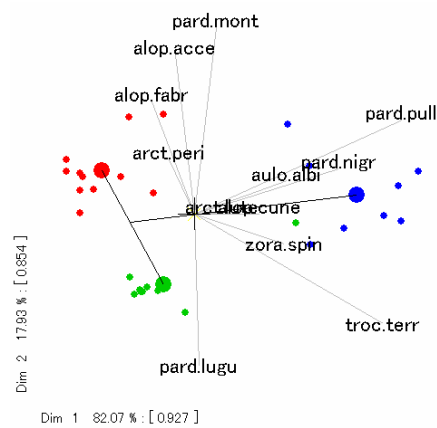
回帰木のルールを返すには関数 `print` を用いる。

引数 `pca` は、樹木モデルと被説明変数の主成分分析の結果をリンクをコントロールする。デフォルトは `pca = FALS` になっている。次に `pca = TRUE` にした結果を示す。木のグラフが返されたら、グラフの画面でマウスを右クリックすると被説明変数の主成分分析のグラフが返される。

```
>mvpart(as.matrix(spider[,1:12])~water+twigs+reft+herbs+moss+sand,spider,pca=T)
```



(a)



(b)

図7 データ `spider` の多変量回帰木と主成分の散布図とのリンク

主成分分析のグラフは、関数 `rpart.pac` を用いて作成することもできる。また、パッケージ `mvpart` の中には 9 種類の距離を求める関数 `gdiss`、古典的多次元尺度法と回帰木とのリンク関数 `cmds.diss` などが用意され、これらを用いた多変量回帰分析を行うこともできる。