

# Rと樹木モデル(1)

## 1. 樹木モデルとは

樹木モデル(tree-based model)は、非線形回帰分析、判別分析のひとつの方法で、回帰の問題では回帰木(regression tree)、分類の問題では分類木(classification tree)あるいは決定木(decision tree)と呼ばれている。

樹木モデルは、説明変数の値を分岐させ、それらを組み合わせて、判別・予測のモデルを構築する。分析の結果は IF-THEN のような簡潔なルールを生成させ、またそのルールを樹木構造で図示することができるため、理解しやすいことから、その応用が急速に広がっている。

図1に示す散布図の回帰モデルの構築を考えてみよう。このような単回帰の場合は、従来の回帰では直線、あるいは曲線でモデルを構築するが、回帰木の場合は、図2に示すような軸(変数)と平行する直線を組み合わせた階段型折れ線でモデルを構築する。図3が図2の樹木構造の図示である。

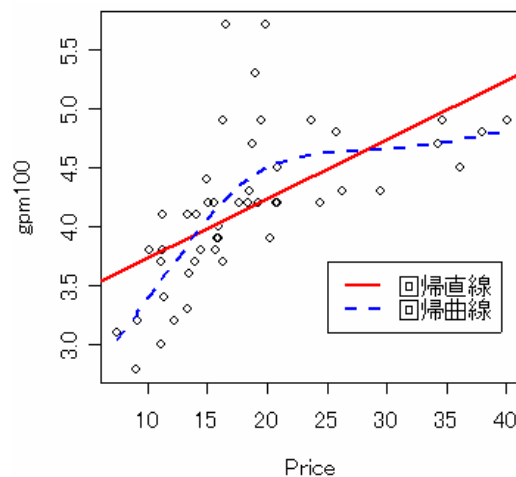


図1 回帰直線と回帰曲線

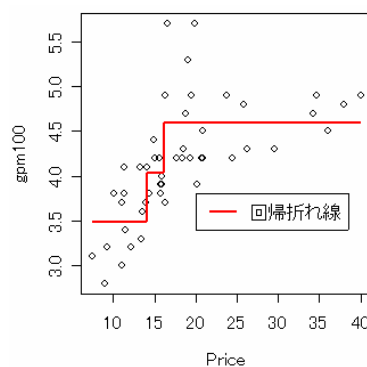


図2 樹木モデルによる回帰折れ線

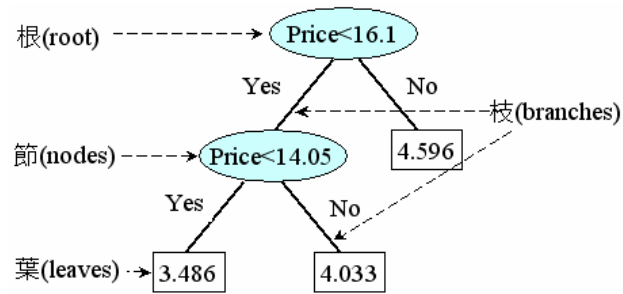


図 3 回帰木

図 3 の木は次に示す IF-THEN ルールの記述と等価である。

- 1) root
- 2) IF {Price >= 16.1} THEN {4.596}
- 3) IF {Price < 16.1} THEN{
  - 4) IF{Price < 14.05} THEN {3.486}
  - 5) IF{Price >= 14.05} THEN {4.033}

樹木モデルに関する研究は、1960 年代初期までさかのぼる。今日、広く用いられているのは CHAID、CART、C4.5/C5.0/See5 をベースとしたアルゴリズムである。

CHAID(Chi-squared automatic interaction detection)は J. A. Hartigan によって 1975 年に発表された、上記の 3 種類の中で最も古いアルゴリズムである。CHAID は 1963 年 J.A.Morgan らによって提案された AID(automatic interaction detection)を発展させたもので、変数の分岐基準としてカイ 2 乗統計量や F 検定統計量など統計学で多く用いられている統計量が用いられている。

CHAID はデータ解析の専用ソフト SAS と SPSS の統計パッケージに採用されていることにより広く用いられている。

CART(classification and regression trees) は、カリフォルニア大学の L.Breiman、R.A. Olshen、C.J.Stone、スタンフォード大学の J.H.Friedman が 1970 年代初めごろから共同研究を始め、1980 年代初めごろに公開したアルゴリズムである。

CART は、説明変数を 2 進分岐させ、2 進木を生成する。初期の CART では、分岐の評価基準として経済学者ジニ(Gini)によって提案されているジニ係数をジニ多様性指標(Gini's diversity index)として用いたが、最近は情報利得(information gain)なども用いられている。CART は、樹木をあらかじめ何の制限もせずに生長させ、データと対話しながら樹木の剪定を行う方法を取っている。この点は、樹木が生長し過ぎないように事前に制御する CHAID と大きく異なる。

C4.5/C5.0/See5 は、オーストラリアの J. Ross Quinlan が機械学習のアプローチで 1986 年に発表した ID3(iterative dichotomiser 3)を改良・発展させたものである。

現在、J. Ross Quinlan は RULEQUEST RESEARCH というソフトウェア会社を運営しな

がらシドニーにある New South Wales 大学でコンサルティング教授としてデータマイニングの分野で活躍している。

C4.5/C5.0/See5 は、2 進木に限らないのが CART との大きな違いである。C4.5/C5.0/See5 の前身である ID3 では、分岐の評価基準として情報利得(information gain)を用いたが、C4.5/C5.0/See5 では、利得比(gain ratio)を用いている。

これらの樹形モデルのアルゴリズムはそれぞれ特徴を持っており、どのアルゴリズムが優れているかは評価し難い。

これらの樹形モデルの大きい違いは樹木の生成・生長、樹木の剪定のアルゴリズムである。

樹木の生成・生長とは、データセットから樹木の幹・枝となる説明変数を選定し、分岐基準によって分岐させ、樹木を生長させるのに関する樹木モデルの核の部分である。

樹木の剪定とは、生長し過ぎた樹木を何らかの基準に基づいて枝刈りし、当てはめの良い簡潔な樹木モデルを構築する作業である。

このようなアルゴリズムの詳細については紙面上の制約により割愛せざるを得ない。

R には、樹木モデル関数 **tree**、**rpart** がある。最近、**rpart** を多変量回帰木(Multivariate regression trees)に拡張させた **mvpart** が公開されている。これらは、CART の親族であると考えられる。

## 2. 樹木モデル関数 tree

関数 **tree** では、節の分岐の評価基準、つまりそれぞれの節で、どのような分割が最適であるかに関する評価は尤離度(deviance)とジニ(Gini)係数を用いている。デフォルトは、尤離度になっている。

関数 **tree** では、被説明変数(応答変数)が量的であるか、質的であるかによって、自動的に予測問題であるか、それとも判別問題であるかを識別して分析を行うので、通常では引数のパラメータの指定を行わなくてもよい。

関数 **tree** の書き式を次に示す。書き式の中の引数 **form** は、線形回帰分析関数 **lm** の場合と同じである。その詳細は **help(tree)** で確認できる。

```
tree(form,data,...)
```

関数オブジェクト **tree** クラスの主なメソッド関数を表 1 に示す。関数 **tree** を用いるためにはパッケージ **tree** をロード(**library(tree)**) しなければならない。

表 1 関数 tree クラスの主なメソッド

メソッド名	機能
<b>print</b>	結果を返す
<b>summary</b>	要約を返す
<b>plot.tree</b>	樹木の図を返す
<b>prune.tree</b>	剪定を行う
<b>predict.tree</b>	予測・判別を行う

<code>snip.tree</code>	樹木の剪定を行う
<code>partition.tree</code>	直線による分割図を返す
<code>cv.tree</code>	交差確認を行う

実際のデータを用いて、これらの使用例を示す。

## (1) 関数 `tree` の回帰木

ここでは R に用意されているデータ `cars` を用いることにする。データ `cars` は 50 個体(観測体)、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停車までの距離(`dist`)の計測値である。ここでは、速度(`speed`)を説明変数、ブレーキを掛けた後停車までの距離(`dist`)を被説明変数とする。

```
>cars.tr<-tree(dist~speed,data=cars)
>print(cars.tr)
node), split, n, deviance, yval
    * denotes terminal node
1) root 50 32540.0 42.98
 2) speed < 17.5 31 8307.0 29.32
   4) speed < 12.5 15 1176.0 18.20
    8) speed < 9.5 6 277.3 10.67 *
    9) speed > 9.5 9 331.6 23.22 *
   5) speed > 12.5 16 3535.0 39.75 *
 3) speed > 17.5 19 9016.0 65.26
   6) speed < 23.5 14 2847.0 55.71 *
   7) speed > 23.5 5 1318.0 92.00 *
```

これが回帰木の IF-THEN ルールである。

返された結果の毎行は次のような項目を順に並べている。

```
node), split, n, deviance, yval
```

`node)`は分岐のノード(節)の番号、`split`は分岐の条件、`n`はそのノードに含まれている個体の数、`deviance`はその節の分岐基準尤離度である。`yval`はその節の被説明変数値で、判別分析の場合はグループの名前となる。星印が付いている行は端末の節であり、葉とも呼ぶ。

回帰木のルールは次のように樹木構造で図示することができる。関数 `plot.tree` の `.tree` は省略できる。

```
> plot(cars.tr,type="u")
> text(cars.tr)
```

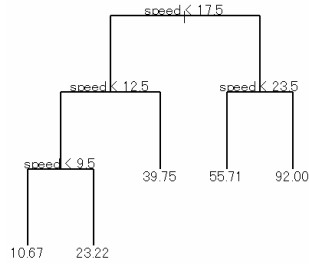


図4 データ cars の回帰木 1

また、説明変数が1つの場合は、被説明変数と説明変数の2次元の散布図に、階段型回帰折れ線を追加することができる。

```
> plot(cars$speed,cars$dist)
> partition.tree(cars.tr,add=T,col=2)
```

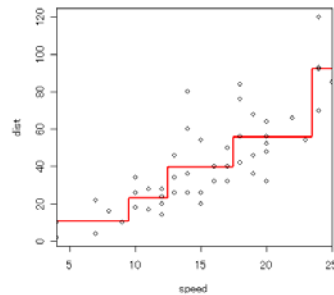


図5 データ cars の回帰木の折れ線

生成した木が冗長な場合もある。その場合は何らかの基準に基づいて剪定を行うことが必要である。剪定は、関数 `prune.tree` を次ぎのように用いることができる。例えば、葉を4つにしたいときには、引数 `best=4` にする。

```
> (cars.tr1<-prune.tree(cars.tr,best=4))
node), split, n, deviance, yval
      * denotes terminal node

1) root 50 32540 42.98
 2) speed < 17.5 31 8307 29.32
   4) speed < 12.5 15 1176 18.20 *
   5) speed > 12.5 16 3535 39.75 *
 3) speed > 17.5 19 9016 65.26
   6) speed < 23.5 14 2847 55.71 *
   7) speed > 23.5 5 1318 92.00 *

> plot(cars.tr1); text(cars.tr1,all=T)
```

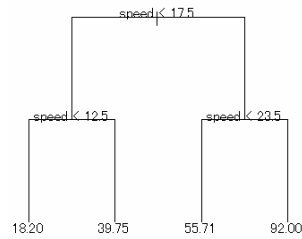


図 6 剪定を行った cars の回帰木

```
> plot(cars$speed,cars$dist)
> partition.tree(cars.tr1,add=T,col=2)
```

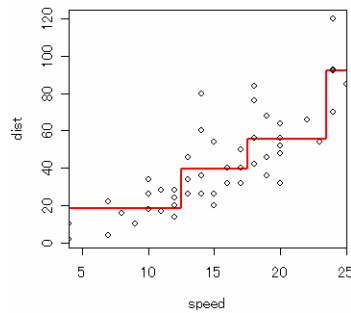


図 7 剪定を行った cars の回帰木の折れ線

## (2) 関数 tree の分類木

分類木ではデータ iris を用いることにする。

```
> (iris.tr<-tree(Species~.,data=iris))
```

```
node), split, n, deviance, yval, (yprob)
* denotes terminal node
1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
2) Petal.Length < 2.45 50 0.000 setosa ( 1.00000 0.00000 0.00000 ) *
3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
6) Petal.Width < 1.75 54 33.320 versicolor ( 0.00000 0.90741 0.09259 )
12) Petal.Length < 4.95 48 9.721 versicolor ( 0.00000 0.97917 0.02083 )
24) Sepal.Length < 5.15 5 5.004 versicolor ( 0.00000 0.80000 0.20000 ) *
25) Sepal.Length > 5.15 43 0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
13) Petal.Length > 4.95 6 7.638 virginica ( 0.00000 0.33333 0.66667 ) *
7) Petal.Width > 1.75 46 9.635 virginica ( 0.00000 0.02174 0.97826 )
14) Petal.Length < 4.95 6 5.407 virginica ( 0.00000 0.16667 0.83333 ) *
15) Petal.Length > 4.95 40 0.000 virginica ( 0.00000 0.00000 1.00000 ) *
```

```
> plot(iris.tr,type="u"); text(iris.tr)
```

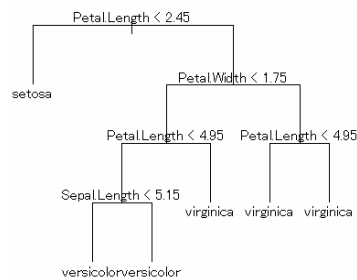


図 8 関数 tree による iris の分類木

上記の結果から分類木のノード 12, 7 以下は無意味であることがわかる。ノードを指定した木の剪定は関数 `snip.tree` を用いる。引数 `nodes` に、次ぎのようにノードの番号を指定すると、そのノードが刈り取られた結果が返される。図 9 にその分類木を示す。

```
>(iris.tr1<-snip.tree(iris.tr,nodes=c(12,7)))
```

```
node), split, n, deviance, yval, (yprob)
* denotes terminal node
1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
2) Petal.Length < 2.45 50 0.000 setosa ( 1.00000 0.00000 0.00000 ) *
3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
6) Petal.Width < 1.75 54 33.320 versicolor ( 0.00000 0.90741 0.09259 )
12) Petal.Length < 4.95 48 9.721 versicolor ( 0.00000 0.97917 0.02083 ) *
13) Petal.Length > 4.95 6 7.638 virginica ( 0.00000 0.33333 0.66667 ) *
7) Petal.Width > 1.75 46 9.635 virginica ( 0.00000 0.02174 0.97826 ) *
```

```
>plot(iris.tr1,type="u");text(iris.tr1)
```

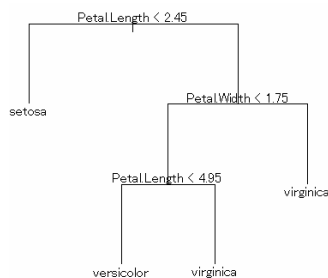


図 9 剪定された iris の分類木

図 9 から分かるように、この分類木には、2 つの変数 `Petal.Length`、`Petal.Width` のみが用いられている。このような 2 変数のみの場合は、`partition.tree` を用いて、図 10 のように、2 変数の散布図に分類の分割直線を加えることができる。

```
>iris.label<-c("S", "C", "V")[iris[, 5]]
>plot(iris[,3],iris[,4],type="n")
>text(iris[,3],iris[,4],labels=iris.label)
>partition.tree(iris.tr1,add=T,col=2,cex=1.5)
```

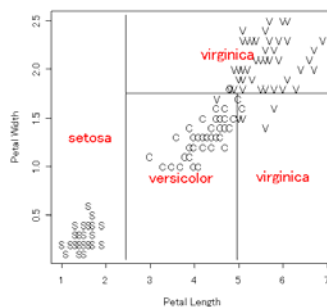


図 10 分類の分割

関数 `summary` は決定木の基本的な情報を返す。

```
> summary(iris.tr1)
Classification tree:
snip.tree(tree = iris.tr, nodes = c(7, 6))
Variables actually used in tree construction:
[1] "Petal.Length" "Petal.Width"
Number of terminal nodes: 3
Residual mean deviance: 0.2922 = 42.95 / 147
Misclassification error rate: 0.04 = 6 / 150
```

### (3) 木の剪定の目安

生長し過ぎた樹木の剪定を行う場合、どのように剪定するかが問題である。

学習データでは、木を大きく生長させることで、予測・分類の精度を上げることが可能である。ただし、木の生長が大きくなると同時に、ルールが煩雑になり、解釈が難しくなる場合も少なくない。また、生長し過ぎた結果をテストデータに当てはめたとき、適応が良くないことがよく知られている。

そこで、大きい木をどこまで剪定(枝を切る)するかを決める基準が必要である。

関数 `tree` では、幾つかの方法があるが、ここでは交差確認法による尤離度(deviance)の値を用いる方法を説明する。

関数 `tree` には、 $n$  重交差確認を行う関数 `cv.tree` がある。 $n$  重交差確認におけるデフォルトの  $n$  は 10 になっている。交差確認法による剪定のための尤離度の考察は、次のように関数 `cv.tree` を用いる。引数 `FUN=prune.tree` は最適なサイズの木を `prune.tree` によって生成する。

```
>iris.cv<-cv.tree(iris.tr,FUN=prune.tree)
>plot(iris.cv)
```

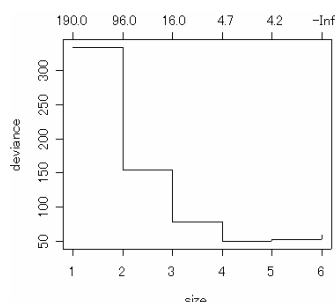


図 11 サイズ対尤離度のプロット 1

読者が同じのコマンドを実行しても図 11 と同じのグラフが得られるとは限らない。その理由は後ほど述べる。

図 11 の縦軸は尤離度、横軸は葉の数(サイズ)である。横軸のサイズ 4 のとき尤離度が最も小さい。これはサイズ 4 の木がモデルの候補として考えられることを示唆する。しかし、同じの操作を繰り返して実行すると、場合によっては尤離度が最小となるサイズが 3 になったり、5 になったりすることがある。これは交差確認を行う際、データの分割の結果が毎回異なるからである。

折衷の案として、上記のコマンドに続いて、次のように同じの作業を複数回行い、その平均値を用いることが考えられる。ここでは上記の交差確認を 10 回繰り返し、その平均値を用いることにする。10 回が不十分と考える場合は、回数をさらに増やせばよい。

```
> for(i in 2:10) iris.cv$dev<-iris.cv$dev+
cv.tree(iris.tr,FUN=prune.tree)$dev
> iris.cv$dev=iris.cv$dev/10;plot(iris.cv)
```

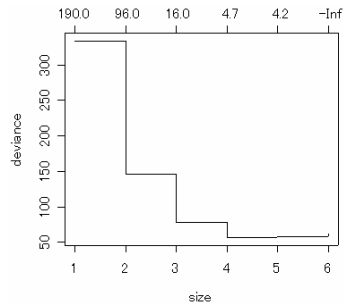


図 12 剪定サイズ対尤離度のプロット 2

交差確認を 10 回繰り返した結果、尤離度を最小とするサイズは 4 である。サイズ 4 にした木の剪定を次に示す。この結果は、図 9 と同じであることを注意して欲しい。

```
>prune.tree(iris.tr,best=4)
node), split, n, deviance, yval, (yprob)
* denotes terminal node
1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
2) Petal.Length < 2.45 50 0.000 setosa ( 1.00000 0.00000 0.00000 ) *
3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
6) Petal.Width < 1.75 54 33.320 versicolor ( 0.00000 0.90741 0.09259 )
12) Petal.Length < 4.95 48 9.721 versicolor ( 0.00000 0.97917 0.02083 ) *
13) Petal.Length > 4.95 6 7.638 virginica ( 0.00000 0.33333 0.66667 ) *
7) Petal.Width > 1.75 48 9.635 virginica ( 0.00000 0.02174 0.97826 ) *
```

このアプローチでは、データによっては計算がうまくいかない場合がある。

#### (4) 予測と判別

樹形モデルの構築の目的は、学習データ以外のデータについての予測と判別を行うことである。

学習データを用いて構築したモデルを用いて、新たなデータについての予測と判別を行う関数は **predict.tree** である。ただし、ここの `.tree` は省略することができる。その書き式は、基本的には回帰分析および判別分析と同じであるが、引数として `type = c("vector", "tree", "class", "where")` が用意され、問題に合わせて返す結果を指定することができる。

`type="vector"` を指定すると、回帰問題では予測値、分類問題では属性に関する確率が返される。`type="tree"` を指定すると、新たなデータによるノード尤離度 **deviance** とそのノードに属する個体数 **n** が返される。`type="class"` を指定すると、新たなデータの属性の判別結果が返される。`type="where"` を指定すると、各個体(ケース)が到達するノードの番号を返す。

次に、iris データの奇数行を学習データ、偶数行をテストデータとした判別の例を示す。

```
#データセットを作成する。
>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
```

```
#学習データによる分類木を構築する。
>Iris.tr<-tree(Species~.,train.data)
```

```
#分類木の剪定を行う。
>Iris.tr1<-prune.tree(Iris.tr,best=3)

# 分類木による新たなデータを分類する。
>Iris.res<-predict(Iris.tr1,test.data,type="class")
> table(Iris.lab,Iris.res)
```

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

関数 `tree` には、以上で説明・使用したメソッド以外に、`tree.control`、`tree.screens`、`tile.tree`、`text.tree`、`prune.misclass`、`na.tree.replace`、`deviance.tree` などがある。