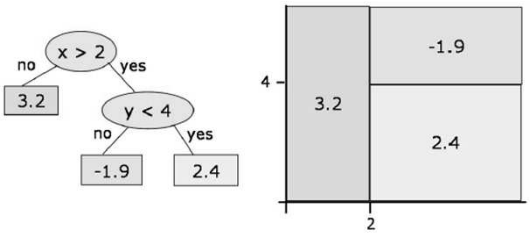
決定木　その5
回帰木

慶應義塾大学理工学部
櫻井彰人

回帰木 Regression Trees

- 決定木と同じ, 但し 葉において、実数値定数を出力する.



葉における値

- 今いる葉ノードには複数個のデータがあると仮定しよう。なおかつ、何らかの理由により、このノードはこれ以上分割しないものとする.
- 離散値の場合（これまでの場合）, 葉における値（出力値）は、その葉における多数派の値としていた.
- 数値属性の場合, 妥当な値は平均値であろう.
- 従って, もし葉ノードにおける出力値として平均値を用いるならば, (これからノードを分割して子供が葉ノードになろうというときには）枝分かれして作られる新たな葉ノードにおいて、データのもつ値が、当該葉ノード内の値の平均値よりあんまり離れていない方がよかろう.
- 統計学には、数値の集合がどのくらい分散しているかを表す尺度がある
  - (言い換えれば, 個々の数値が平均値からどれだけ離れているか);
  - ご存じの分散である.

平均: 念のため（分散の説明のため）

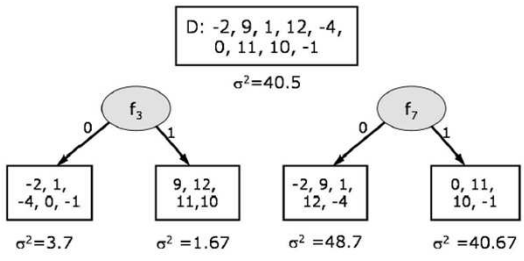
- 観測した数値データのばらつきを均した値。たくさんの意味がある
- 今回、興味があるのは、母平均と標本平均.
  - 母平均は、確率変数の1次モーメント。すなわち、
    $$\mu = \int_{-\infty}^{\infty} z\, f(z)\, dz$$
  - z_1 から z_m の標本平均値:
    $$\hat{\mu} = \frac{1}{m}\sum_{k=1}^{m} z_k$$
  - 標本平均は母平均の不偏推定量である。すなわち、観測を無限に繰り返せば、標本平均は母平均に収束する。
- 母平均は存在するとは限らない

分散: 念のため

- 観測数値データがどれだけ散らばっているかの指標・尺度.
- 母分散、標本分散、不偏分散が代表的。データは z_1 から z_m とする:
  - 母分散: 確率変数の2次の中心化モーメント
    $$\sigma^2 = \int_{-\infty}^{\infty} (z-\mu)^2 f(z)\, dz \qquad \mu = \int_{-\infty}^{\infty} z\, f(z)\, dz$$
  - 標本分散
    $$s^2 = \frac{1}{m}\sum_{k=1}^{m} (z_k - \hat{\mu})^2 \qquad \hat{\mu} = \frac{1}{m}\sum_{k=1}^{m} z_k$$
  - 不偏分散
    $$\hat{\sigma}^2 = \frac{1}{m-1}\sum_{k=1}^{m} (z_k - \hat{\mu})^2 \qquad \hat{\mu} = \frac{1}{m}\sum_{k=1}^{m} z_k$$
- 注意
  - 母分散は存在するとは限らない
  - 不偏分散は、母分散の不偏推定量である。すなわち、何度も測定を（今の場合は、測定して計算を）繰り返したときに、当該値が真の分散値に収束するようなものである。
  - 標本分散は、不偏推定量ではない。
  - 標準偏差は、分散の平方根。標準偏差は、偏差を計測・比較するときの標準値（多分）

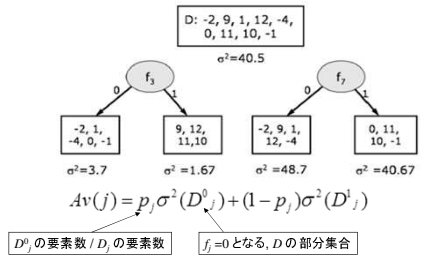
ノード分割（データの分割）

- 分割のよさを計る尺度として分散の平均値を用いることにする.
- （出力たるべき）y 値のみを記す.



ノード分割

- 離散値の場合と同様に、分散の平均値を計算するとき、各新ノードに渡されるデータ数で重み付けた、重み付き平均値を用いることにする（そうでなければならない）.

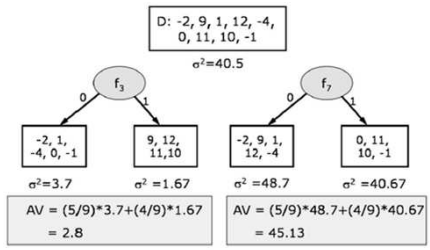


$$Av(j) = p_j \sigma^2(D^0_j) + (1-p_j)\sigma^2(D^1_j)$$

D^0_j の要素数 / D_j の要素数　　f_j =0 となる, D の部分集合

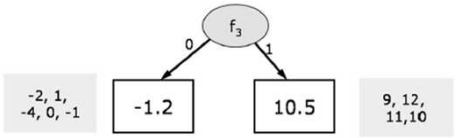
ノード分割

- 簡単に計算できることだが、属性3を用いた分割の場合の分散平均値は、属性7を用いた場合のそれより、非常に小さい。従って、属性3を選ぶことになる.



停止条件の例

- 分散が十分小さくなったらば、停止する（それ以上、ノード分割はしない）
- この場合、予め考えておいたように、葉の出力値はその平均値とする.



付録

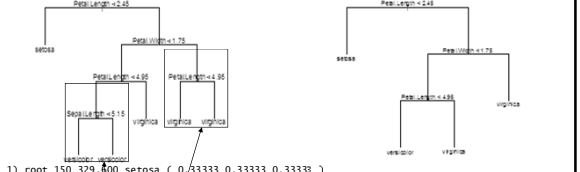
Rにおける決定木

- R には、決定木関連のパッケージとして、tree、rpart、及び rpart を多変量回帰木 (multivariate regression trees)に拡張させた mvpart がある。

分類木の例（tree）

```
data(iris)
(iris.tr<-tree(Species~.,data=iris))
plot(iris.tr,type="u"); text(iris.tr)
```

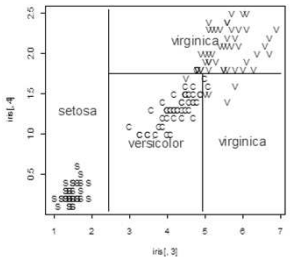
```
(iris.tr1<-snip.tree(iris.tr,nodes=c(12,7)))
plot(iris.tr1,type="u");text(iris.tr1)
```




```
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 ) *
```

分類木の例（tree）

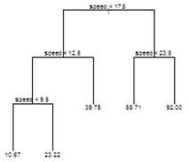
```
library(tree)
iris.label<-c("S", "C", "V")[iris[, 5]]
plot(iris[,3],iris[,4],type="n")
text(iris[,3],iris[,4],labels=iris.label)
partition.tree(iris.tr1,add=T,col=2,cex=1.5)
```



回帰木の例（tree）

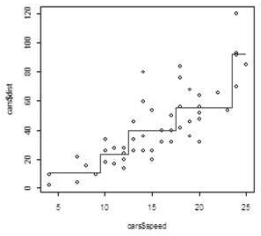
```
> Library(tree)
> data(cars)
> cars.tr<-tree(dist~speed,data=cars)
> print(cars.tr)
node), split, n, deviance, yval
      * denotes terminal node

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

```
Library(tree)
data(cars)
cars.tr<-tree(dist~speed,data=cars)
print(cars.tr)
plot(cars.tr,type="u")
text(cars.tr)
plot(cars.tr,type="u")
text(cars.tr)
```



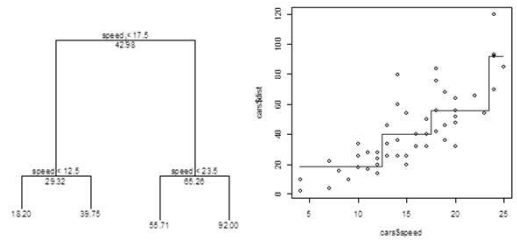
回帰木の例（tree）

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



回帰木の例（tree）

```
(cars.tr1<-prune.tree(cars.tr,best=4))
plot(cars.tr1); text(cars.tr1,all=T)

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



では、別のデータで

- 例によって、テニスのデータを用いてみよう
- このデータの特徴は、すべての属性が離散値であること

Outlook	Temp.	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

説明が必要です。なお、漢字もOK

library(tree) としたあと

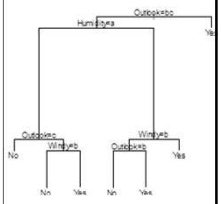
```
> setwd("D:/R/Sample")
> playTennis <- read.csv("playTennis.csv", header=T)
> (playTennis.tr<-tree(Play~.,data=playTennis))
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

1) root 14 18.250 Yes ( 0.3571 0.6429 )
  2) Humidity: High 7  9.561 No ( 0.5714 0.4286 ) *
  3) Humidity: Normal 7  5.742 Yes ( 0.1429 0.8571 ) *
> plot(playTennis.tr); text(playTennis.tr)
```

これは失敗と言っていいでしょう。なぜこうなってしまったのでしょうか？
それは、枝分かれするときの条件が厳しく（つまり、枝分かれしないよう）なっているからです。
それ（つまり、制御の仕方）を調べてみましょう。
?tree
として下さい。"tree"の説明書が得られます。しかし、木を生成するときの制御の仕方についての記述は見つかりません。こういうときは、controlというキーワードを探してみます。下の方に
control.treeという文言があります。ここをクリックするか
?control.tree
としてみてください。tree.control(nobs, mincut = 5, minsize = 10, mindev = 0.01)が制御方法であり、default値であることが分ります。多少試行錯誤すると、今回は、mincut = 1, minsize = 2が最小値、つまり、最も木が発達しやすいパラメータであることが分ります。そこで、
tree.control(length(playTennis[,1]), mincut = 1, minsize = 2)
としてみますが、結果は変わりません。
理由は分りません。
やむをえず、別のライブラリを使うことにします。

```
> library(rpart)
> setwd("D:/R/Sample")
> playTennis <- read.csv("PlayTennis.csv", header=T)
> (playTennis.tr <- rpart(Play~ ., playTennis ) )
n= 14

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 14 5 Yes (0.3571429 0.6428571) *
> plot(playTennis.tr); text(playTennis.tr)
 以下にエラー plot.rpart(playTennis.tr) : fit is not a tree, just a root
```

これはもっと悪い。枝分かれせず、根のみとなってしまった。
先ほどと同様に
?rpart
としてみましょう。今度は引数に control というものがあります。下の例題を見ると、rpart.control を使えばよいことが分ります。rpart.control をクリックするか ?rpart.control としてみましょう。Minsplit を小さくすれば良さそうなことが想像できます。試してみましょう。

```
> library(rpart)
> setwd("D:/R/Sample")
> playTennis <- read.csv("PlayTennis.csv", header=T)
> (playTennis.tr <- rpart(Play~ ., playTennis,
+    control=rpart.control(minsplit=1)) )
n= 14

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 14 5 Yes (0.3571429 0.6428571)
   2) Outlook=Rainy,Sunny 10 5 No (0.5000000 0.5000000)
     4) Humidity=High 5 1 No (0.8000000 0.2000000)
       8) Outlook=Sunny 3 0 No (1.0000000 0.0000000) *
       9) Outlook=Rainy 2 1 No (0.5000000 0.5000000)
        18) Windy=True 1 0 No (1.0000000 0.0000000) *
        19) Windy=False 1 0 Yes (0.0000000 1.0000000) *
     5) Humidity=Normal 5 1 Yes (0.2000000 0.8000000)
      10) Windy=True 2 1 No (0.5000000 0.5000000)
        20) Outlook=Rainy 1 0 No (1.0000000 0.0000000) *
        21) Outlook=Sunny 1 0 Yes (0.0000000 1.0000000) *
      11) Windy=False 3 0 Yes (0.0000000 1.0000000) *
   3) Outlook=Overcast 4 0 Yes (0.0000000 1.0000000) *
> plot(playTennis.tr); text(playTennis.tr)
```



今度はうまく行ったようである。では、未知データがどう分類されるか見てみよう。
"predict" について rpart の説明書中には記述がない。
こういったときは、?predict.rpart としてみる（つまり、クラス rpart のメソッド predict）。
パッケージ e1071の naiveBayes とは異なり、次のように簡単にテストできる。

```
PlayTennisTest02 <- read.csv("PlayTennisTest02.csv",header=TRUE)
predict(playTennis.tr, PlayTennisTest02
```

```
> playTennisTest02 <- read.csv("PlayTennisTest02.csv",header=TRUE)
> predict(playTennis.tr, playTennisTest02)
     No Yes
[1,]  1   0
[2,]  0   1
> playTennisTest02
  Outlook Temp. Humidity Windy Play
1   Sunny  Cool     High  True   No
2   Rainy  Mild   Normal False  Yes
```

結果は勿論、想定通り。なお、パラメータに type があり、確率値の出力が可能なように書かれているが、どうもそうではない。

```
> predict(playTennis.tr, PlayTennisTest02, type="prob")
     No Yes
[1,]  1   0
[2,]  0   1
> predict(playTennis.tr, PlayTennisTest02, type="matrix")
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    0    1    0
[2,]    2    0    3    0    1
```

気がついたかもしれませんが、tree も rpart も 2分木しか作りません。
その点では、weka の J48 の方がよくできています。