8. 決定木に関する演習

決定木の学習を通して、
過学習を理解しよう

本項の内容・目標

- 決定木を作ってみて、理解する。
  - 何をいまさら、かもしれませんが、
  - 一度、ゆっくり、演習をしてみよう、というわけです
- Cross validation を行ってみる
- 過学習について、過学習を起こさせてみて、理解する

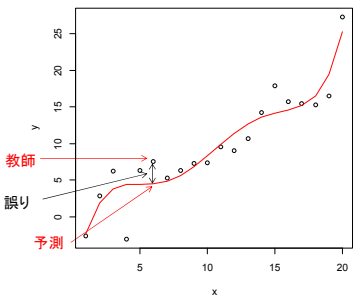
目次

- 誤差の話
- 木の作り方
- Cross validation
- 過学習の例

まずは、誤差の話

- 学習誤り・訓練誤り・学習誤差・訓練誤差
  - 学習したが、誤りが残る場合、その量
  - ただし、その「量」の測り方には、いろいろある
  - 今回は、基礎的なものを用いる
  - 回帰の場合：絶対値誤差、二乗誤差
  - 分類の場合：0-1誤差
  - 誤差といわず、ロスということもある

学習誤差：回帰の場合



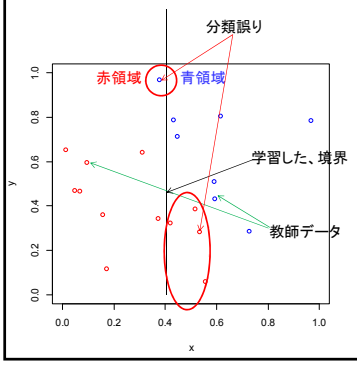
測り方

絶対値誤差
$|$ 教師値 $-$ 予測値 $|$

二乗誤差
$($ 教師値 $-$ 予測値 $)^2$

それぞれの値の平均値を用いる

学習誤差：分類の場合



測り方（数え方）

confusion matrix

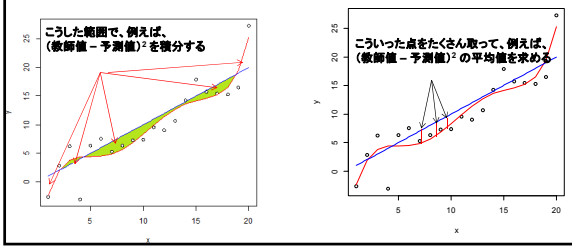
	予測の赤	予測の青
真の赤	8	4
真の青	1	7

精度= (8+7)/(8+7+4+1)

テスト誤差：回帰の場合

1. 真の回帰曲線を知っている場合

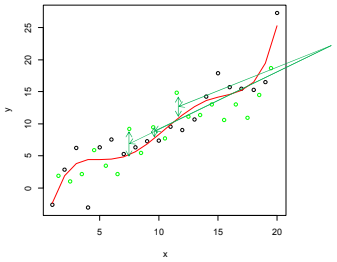
理論的に計算できるなら、勿論、計算する。
計算できないなら、サンプル点をとって近似的に求める



テスト誤差：回帰の場合

2. 真の回帰曲線を知らない場合

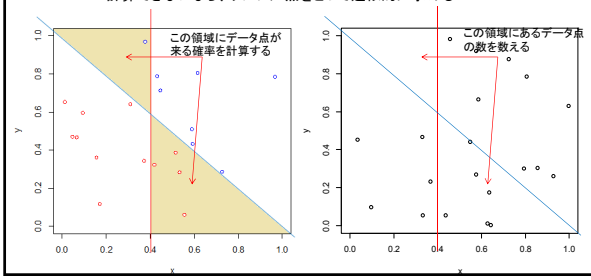
学習用のデータを、学習に用いる部分とテストに用いる部分とに分ける。
そして、テスト用部分で、誤差を計算する



テスト誤差: 分類の場合

1. 真のデータ分布を知っている場合

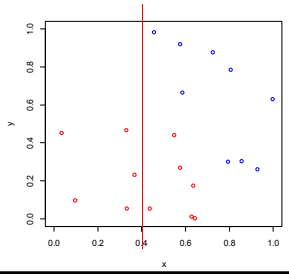
理論的に計算できるなら、勿論、計算する。
計算できないなら、サンプル点をとって近似的に求める



テスト誤差: 分類の場合

2. 真の分類境界を知らない場合

学習用のデータを、学習に用いる部分とテストに用いる部分とに分ける。
そして、テスト用部分で、誤差を計算する



confusion matrix

	予測の赤	予測の青
真の赤	5	6
真の青	0	9

精度= (5+9)/(5+9+6+0)

汎化誤差

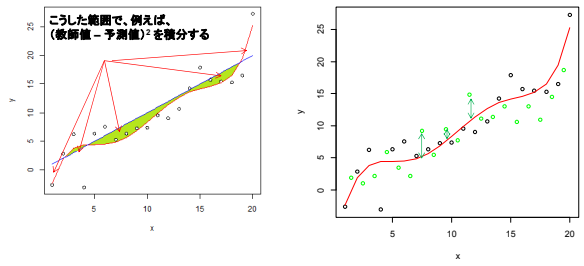
汎化誤差とは、真の回帰曲線・分類境界に対する誤差です。
したがって、前のスライドの「テスト誤差」で「真の回帰曲線・分類境界が分かっている場合の、テスト誤差になります。

問題解決、ではありません。
実データにおいては、真の回帰曲線・分類境界が分かっていることはほとんどないからです。

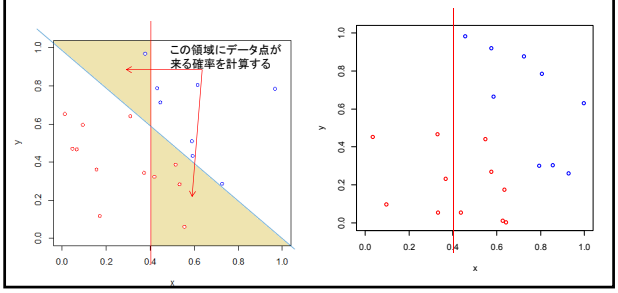
そこで、通常は、
汎化誤差 ≒ テスト誤差（真の回帰曲線・分類境界が分かっていない場合）
と考えるわけですが、それでいいのでしょうか？

その推定精度はどのくらいなのでしょうか？
また、テストデータ数はどのくらい用意すればよいのでしょうか？

汎化誤差とテスト誤差



汎化誤差とテスト誤差



汎化誤差の推定

前のスライドに書きましたように、
1. テストデータを用いて、テスト誤差を計測し、
2. それを汎化誤差の推定値にする
という方針をとります。

汎化誤差を良く推定するためには、テストデータが多い方がよい。
しかし、学習をよくするには、学習データが多い方がよい。
trade-off です。

この trade-off を緩和する一つの方法に cross-validation があります

目次

- 誤差の話
- 決定木の作り方
- Cross validation
- 過学習の例

決定木の構築

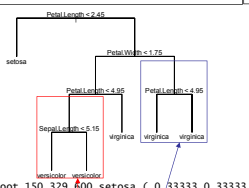
- R には、決定木関連のパッケージとして、tree、rpart、及び rpart を多変量回帰木 (multivariate regression trees)に拡張させた mvpart がある。

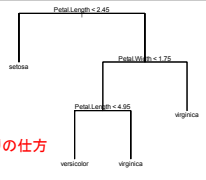
tree ライブラリの使い方

分類木の例（tree）

```
library(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)
```





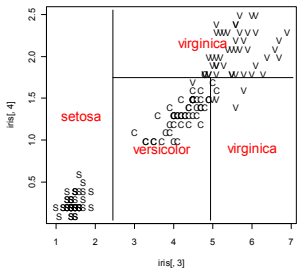
枝刈りの仕方

```
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)
```

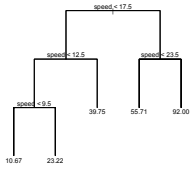
決定木がどう領域を分割して判断しているかが分かる図。treeパッケージでしかできない



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)
>
```

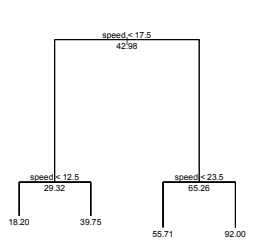
```
library(tree)
data(cars)
cars.tr<-tree(dist~speed,data=cars)
print(cars.tr)
plot(cars.tr,type="u")
text(cars.tr)
plot(cars.tr,type="u")
text(cars.tr)
```

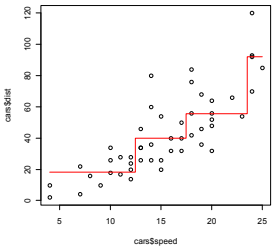


回帰木の例（tree）

```
(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)
```

グラフのようにあらわしたもの





では、別のデータで

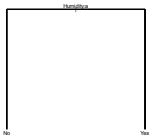
- 例によって、テニスのデータを用いてみよう
- このデータの特徴は、すべての属性が離散値であること

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("08PlayTennis.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)
```

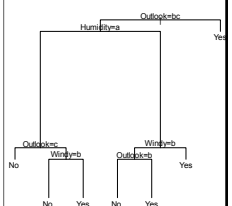


これは失敗と言っていいでしょう。なぜこうなってしまったのでしょうか？
それは、枝分かれするときの条件が厳しく（つまり、枝分かれしないよう）なっているからです。
それ（つまり、制御の仕方）を調べてみましょう。
?tree
として下さい。"tree" の説明書が得られます。しかし、木を生成するときの制御の仕方についての記述は見つかりません。こういうときは、control というキーワードを探してみます。下の方に control.tree という文言があります。ここをクリックするか
?control.tree
としてみてください。tree.control(nobs, mincut = 5, minsize = 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("08PlayTennis.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("08PlayTennis.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("08PlayTennisTest02.csv",header=TRUE)
predict(playTennis.tr, PlayTennisTest02)
```

```
> playTennisTest02 <- read.csv("08PlayTennisTest02.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。

今日の課題1

- Naïve Bayes のときの「今日の課題」とデータは同じです。
- rpart を用いて、下図左の訓練データが与えられたとき、下図右のテストデータの属性「スキー」の値を推定せよ。
- Rを使ってください。応答変数の属性名は スキー になります。

雪	天気	シーズン	体調	スキー
ベタ	霧	ロー	回復	no
新雪	晴	ロー	回復	yes
新雪	霧	ロー	回復	yes
ざらめ	霧	ロー	怪我	no
新雪	晴	ロー	怪我	no
ベタ	晴	ロー	回復	yes
新雪	霧	ロー	回復	yes
ベタ	晴	半ば	回復	yes
新雪	晴	ハイ	回復	yes
新雪	風	ロー	回復	yes
ざらめ	霧	半ば	回復	no
新雪	風	ロー	回復	yes
新雪	晴	半ば	回復	yes
ざらめ	風	ハイ	疲労	no

雪	天気	シーズン	体調	スキー
ベタ	風	半ば	疲労	?

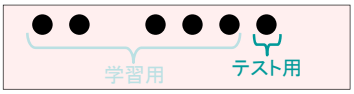
目次

- 誤差の話
- 木の作り方
- Cross validation
- 過学習の例

再掲

k 重クロスバリデーション
k-fold cross validation

訓練データをk 群に分け、$(k-1)$ 群で学習し、残りで予測誤差を計測する。これを全ての k 種類の組み合わせに対して行なう



万能ではないが、多くの場合に結構うまくいく

予測誤差の計測値を、ここでは、汎化誤差と呼ぶことにする

COM実験で行ったように、
Weka: デフォールトが 10-fold CV

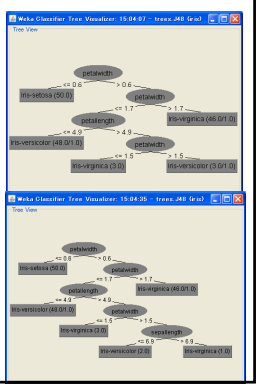
あやめのデータで試してみよう

最小データ数(minNumObj)が2のとき:

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 49  1  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  2 48 |  c = Iris-virginica
```

最小データ数(minNumObj)が1のとき:

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 49  1  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```



R で 10-fold cross validation (1)

パッケージ bootstrap などが使えるが、今回は、自分で作る
iris データに rpart を minsplit=30 で行った結果である。

プログラムファイルを参照のこと

```
setwd("D:/R/Sample")
iris <- read.csv("07iris.csv", header=T)

library(rpart)
n <- nrow( iris ); K <- 10  # number of samples and folds

size <- n %/% K     # size of bins is "size" or "size"+1)
rnk <- rank( runif( n ) )   # this gives a permutation of 1 to n
block <- ( rnk - 1 ) %/% size + 1   # converts rnk to group numbers

all.pred <- NULL
for ( k in 1:K ) {
  iris.tr <- rpart( class ~ . , iris[ block != k, ],
                    method="class", control=rpart.control(minsplit=30) )
  pred <- predict( iris.tr, iris[ block==k, ], type="class" )
  all.pred <- rbind( all.pred,
            data.frame( id=(1:n)[ block==k ], pred=pred) )
}

all.pred.sorted <- all.pred[ sort.int( all.pred$id, index.return=T )$ix, ]
( all.cm <- table( iris$class, all.pred.sorted$pred ) )
( error <- 1 - sum(diag(all.cm))/sum(all.cm) )
```

分割はランダムなので、実行ごとに結果は異なる

次のスライド参照. 第kグループ以外のサンプルを用いて学習する

第kグループに対し、予測する

予測結果を、サンプル番号ともに蓄積していく

R で 10-fold cross validation (2)

パッケージ bootstrap などが使えるが、今回は、自分で作る
iris データに rpart を minsplit=30 で行った結果である。

```
> ( all.cm <- table( iris$class, all.pred.sorted$pred ) )

                 Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa             50              0              0
  Iris-versicolor          0             46              4
  Iris-virginica           0              7             43
> ( error <- 1 - sum(diag(all.cm))/sum(all.cm) )
[1] 0.07333333
```

分割はランダムに行われるので、実験ごとに結果は異なっても不思議ではない。

参考

```
> block
  [1] 10  4  1  1  3  9  3 10  2  5  2  3  8  1  5  1  6  1 10  3  7  3 10 10 10
 [26]  1  7  5 10  4  7  3  2  2  4  9  5  8  9  5  9  4  8  4 10  3  1  9 10 10
 [51]  6  7  8  7  8  4  2  2  4  3  2  5  8  7  6  7  5  5  3  4  3  6  9  6  1
 [76]  6  5  5  6 10  9  9  3  8  4  1  1  8  4  4  3  1 10  2  7  6  6  4  9  2
[101]  4  5  5  4  7  3 10  1  8  2  7  4  1  8  6  7  6  8  2  9  6  5  6  2  7
[126]  3  8  2  6  1  9  8  8  5  9  7  9  9  8  9  7  5  2  6  3  7  2 10  1 10
```

`iris[ block!=1 ]` は、上記の1以外の場所のみ `iris` 要素を取り出したもの

なお、全データで学習した結果の学習誤差は次のようにして求めることができる。

```
iris.tr <- rpart( class ~ . , data=iris, method="class" )
pred <- predict( iris.tr, newdata=iris,
                 type="class", control=rpart.control(minsplit=30) )
cm <- table( iris$class, pred)
print( cm )
err.iris.tr <- 1 - sum(diag(cm))/sum(cm)
print( err.iris.tr )
```

```
> iris.tr <- rpart( class ~ . , data=iris, method="class" )
> pred <- predict( iris.tr, newdata=iris,
+                  type="class", control=rpart.control(minsplit=30) )
> cm <- table( iris$class, pred)
> print( cm )
                 pred
                  Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              50              0              0
  Iris-versicolor           0             49              1
  Iris-virginica            0              5             45
> err.iris.tr <- 1 - sum(diag(cm))/sum(cm)
> print( err.iris.tr )
[1] 0.04
```

R で試してみる決定木のCV: 実験

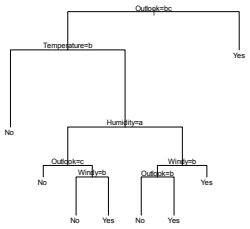
データは iris
複数回（下記の例では10回）10-fold CV を行ったのは、データの組合わせ方で結果が異なるからである。

minsplit=1, K=10 のときのある結果（誤り率）	minsplit=5, K=10 のときのある結果（誤り率）	minsplit=50, K=10 のときのある結果（誤り率）
0.06666667	0.06666667	0.06666667
0.06666667	0.00000000	0.00000000
0.06666667	0.06666667	0.06666667
0.00000000	0.00000000	0.00000000
0.06666667	0.06666667	0.13333333
0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000
0.26666667	0.26666667	0.26666667
0.13333333	0.06666667	0.13333333
0.06666667	0.06666667	0.06666667
平均 0.0733	平均 0.060	平均 0.0733

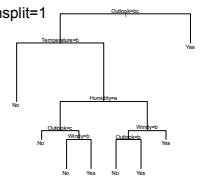
R で試してみる決定木のCV: 最適木？

全データを用いて作成した木達
（データは、 07PlayTennis02.csv ）

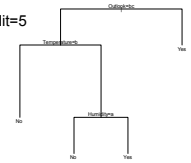
minsplit=2



minsplit=1

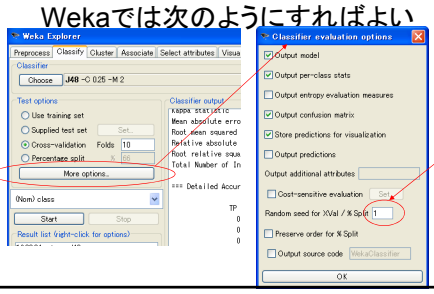


minsplit=5



Weka で CV を繰り返すには

- データ分割の仕方（乱数で決められている）を変えるには、乱数のシードを変えればよい。
  Wekaでは次のようにすればよい



これを変える

本日の課題2

- 08heart.csv を対象として、10-fold cv を行って下さい。応答変数は disease です。
  - なお、08heart.csv は、
    http://archive.ics.uci.edu/ml/datasets/Heart+Disease
    に基づいています

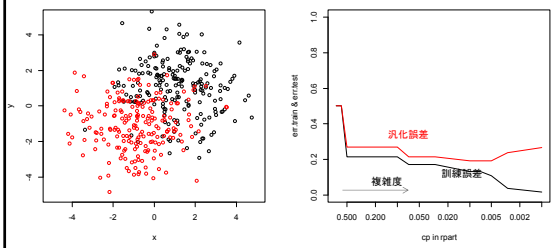
結果の例

```
> print( as.numeric( all.err ) )
 [1] 0.2962963 0.2222222 0.1481481 0.1481481 0.1851852 0.1851852
 [7] 0.2962963 0.3333333 0.1111111 0.1851852
>
> ave.err <- t( all.err ) %*% as.numeric( table(block) ) / n
> print( as.numeric( ave.err ) )
[1] 0.2111111
>
> all.pred.sorted <- all.pred[ sort.int( all.pred$id, index.return=T )$ix, ]
> ( all.cm <- table( heart$disease, all.pred.sorted$pred ) )

           absence presence
  absence      129       21
  presence      36       84
```

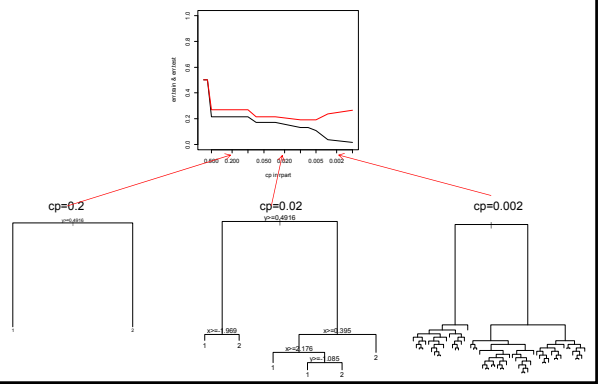
目次

- 誤差の話
- 木の作り方
- Cross validation
- 過学習の例

では、過学習の数値例を



実際の決定木



データの生成

```
library(rpart)

n.train <- 200
n.test <- 200

sd <- 1.5

train1 <- data.frame(
    x=rnorm(n.train, mean=1,sd=sd), y=rnorm(n.train, mean=1,sd=sd),
    c=rep(as.factor(1),n.train)  )
train2 <- data.frame(
    x=rnorm(n.train, mean=-1,sd=sd), y=rnorm(n.train, mean=-1,sd=sd),
    c=rep(as.factor(2),n.train)  )
train <- rbind(train1, train2)

test1 <- data.frame(
    x=rnorm(n.test, mean=1,sd=sd), y=rnorm(n.test, mean=1,sd=sd),
    c=rep(as.factor(1),n.test)  )
test2 <- data.frame(
    x=rnorm(n.test, mean=-1,sd=sd), y=rnorm(n.test, mean=-1,sd=sd),
    c=rep(as.factor(2),n.test)  )
test <- rbind(test1, test2)

plot(   subset(train, c==1)[,1:2], xlim=c(-5,5), ylim=c(-5,5)  )
points( subset(train, c==2)[,1:2], col="red", xlab="", ylab="")
```

set.seed(　　)

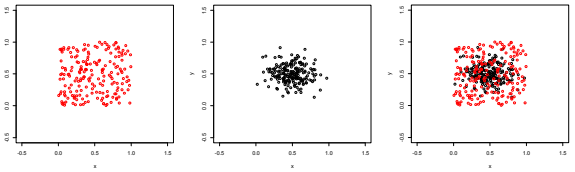
正規乱数は rnorm(個数, mean=平均, sd=標準偏差)
一様乱数は runif(個数, min=最小値, max=最大値)

測定

```
res <- c()
for (cp in
c(0.7,0.6,0.5,0.4,0.3,0.2,0.1,0.07,0.05,0.03,0.01,0.007,0.005,0.003,0.001))
{
  t <- rpart(c~ .,train,  control=rpart.control(minsplit=3, cp=cp))
  tbl.train <- table( train$c, predict(t, train, type="class") )
  err.train <- 1 - sum(diag(tbl.train))/sum(tbl.train)
  tbl.test <- table( test$c, predict(t, test, type="class") )
  err.test <- 1 - sum(diag(tbl.test))/sum(tbl.test)
  print( c(cp, err.train, err.test) )
  res <- rbind(res, c(cp, err.train, err.test) )
}

dev.new()

plot(res[,1:2],log="x", type="l",xlim=c(max(res[,1]),min(res[,1])),
  ylim=c(0,1.0), xlab="cp in rpart",ylab="err.train & err.test")
points(res[,c(1,3)],type="l",col="red",xlab="",ylab="")
```

本日の課題3

- 「では、過学習の数値例を」では、2つの正規分布を2つのクラスに割り当てました。この課題では、一つの一様分布（正方形の中の一様分布）と一つの正規分布（その正方形の中心に平均値があり、分散は適度に小さい正規分布）とをそれぞれのクラスとすることを考えましょう。
  この2つの分布に対して、過学習が発生するかをRで実験してみて下さい。



本日の課題3α

分布を変えると、過学習の起こりやすさが変わるかどうか
みてみましょう。

本日の課題2は、
正規分布の平均と分散を、様々に、変えて、過学習が起こるかどうか調べて下さい
とします。

本日の課題3: 補足