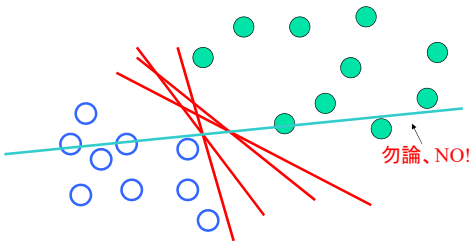
知的情報処理
11.サポートベクターマシン(補足)とブートストラップ

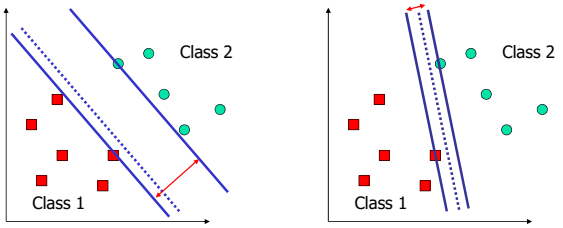
理工学部管理工学科
櫻井彰人

復習
とはいえ、線形境界候補はたくさん

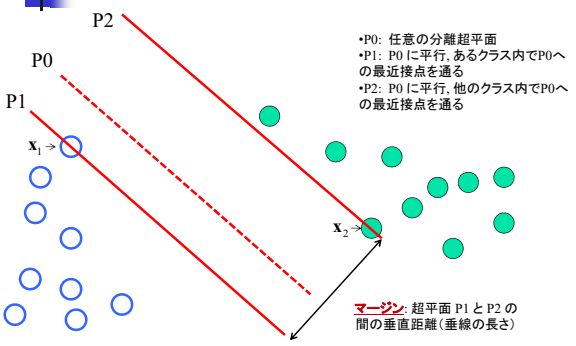


復習
「マージン」の導入

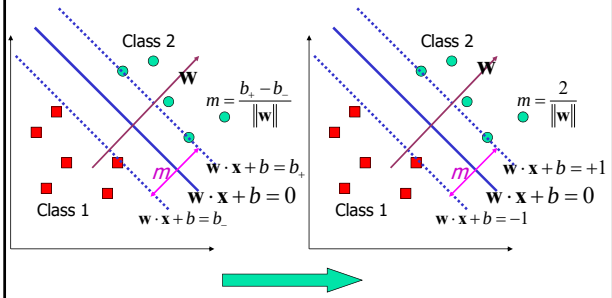
- マージン(大辞林)
  - (1)売買の差額金。利鞘(りざや)。
    (2)販売手数料。
    (3)本などの印刷部分を除く周辺の余白。
- 今回は、「余白」が近いでしょう



復習
「マージン」の導入　その2



復習
式で表すと



復習
改めて記述すると

- 訓練データを $\{x_1, \ldots, x_n\}$, 各x_i のクラスラベルを $y_i \in \{1,-1\}$ とする
- ちょっと工夫すると、決定境界は全ての訓練データを正しく分類する　⇔　$\forall i \;\; y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1$
- そうすると、欲しい決定境界は、次の制約付き最適化問題を解けば求まる

$$\text{Maximize } \frac{2}{\|\mathbf{w}\|} \quad \text{subject to } \forall i \;\; y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 \quad \Rightarrow \quad \text{Minimize } \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to } \forall i \;\; y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1$$

- この解き方は、既に知っていますよね？

復習

双対問題

- 新しい目的関数は、α_i だけに関するものである
- 双対問題である: **w**が分かれば α_i が分かる; α_i が分かれば**w**が分かる
- 元の問題は主問題と呼ばれる
- 双対問題の目的関数は、最大化する
- 従って、双対問題は:

$$\text{Maximize } W(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \qquad \sum_i \alpha_i y_i = 0$$

ラグランジュ乗数を導入したときの α_i の性質

ラグランジュ関数を b に関して微分して（0とおいて）得た条件

復習

解の性質

- α_i の多くはゼロ
  - **w** は「少数の」データ点の線形結合
  - このようなスパースな表現は、k-nn と同様に、データ圧縮とみなすことが可能である。
- **非零の** α_i に対応する$\mathbf{x}_i$ は support vectors (SV) と呼ばれる
  - 決定境界は SV のみによって決まる
  - t_j (j=1, …, s) を s 個の SV の添え字とする. そうすると

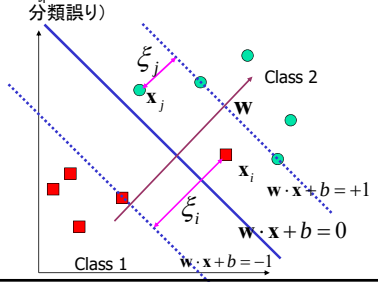
$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

- 未知のデータ **z** に対して
  - $\mathbf{w}\cdot\mathbf{z}+b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}\cdot\mathbf{z}) + b$ を計算し、結果が負ならばクラス1とし、そうでなければクラス2とする
  - 注: **w** を明示的に構成する必要はない

復習

線形分離可能でないとき

- 分類において「誤り」ξ_i を認めよう; 当該誤り値は決定関数 $\mathbf{w}^T\mathbf{x}+b$ の出力値に基づく
- ξ_i は分類を誤った事例の個数の近似となる（1より大の時、分類誤り）



復習

ソフトマージン

- $\sum_i \xi_i$ が最小化されれば, ξ_i は:

$$\begin{cases} \mathbf{w}\cdot\mathbf{x}_i + b \geq 1-\xi_i & \text{if } y_i = 1 \\ \mathbf{w}\cdot\mathbf{x}_i + b \leq -1+\xi_i & \text{if } y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

ここはスラック変数を導入していても、不等式です。サポートベクトル以外のデータ点に対しては、不等式制約だからです

  - ξ_i はスラック変数である
  - 正しく分類されていれば ξ_i=0 である
  - ξ_i は分類誤りの上界である
- 最小化するのは $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i$
  - C: 誤りとマージンのトレードオフを表すパラメータ

実際に計算するときには適宜設定する必要がある。

- 最適化問題

$$\text{Minimize } \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{subject to } \forall i \; y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1-\xi_i, \quad \xi_i \geq 0$$

復習

最適化問題

- この制約付き最適化問題の双対問題は

$$\text{Maximize } W(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \qquad \sum_i \alpha_i y_i = 0$$

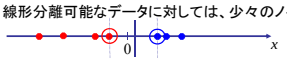
- **w** は $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- これは、線形分離可能な場合とそっくりである。違いは、各α_i に C という上限があることである
- 同様に、QP solver 用いて解く

線形分離できなくとも、非線形関数で分離できないか？

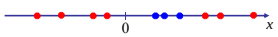
- 線形関数を非線形関数（例えば、多項式）に変換してもうまくいかない。
- ここでのアイデアは、kernel function

非線形 SVM

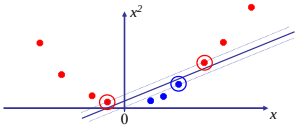
- 線形分離可能なデータに対しては、少々のノイズがあっても、うまくいく:



- しかし、データ集合が線形分離可能でなかったらどうしよう?

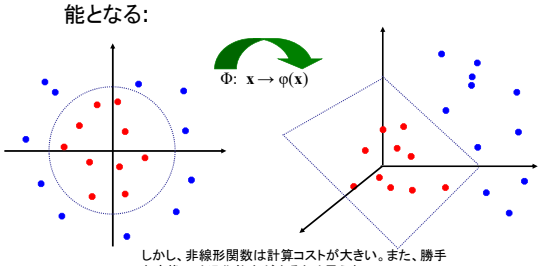


- 例えば… データをより高次元の空間に写像したらどうだろうか:



非線形 SVM: 特徴空間

- 一般的なアイデア: もともとの特徴空間は、いつでも、ある高次元特徴空間に写像すれば、線形分離可能となる:



しかし、非線形関数は計算コストが大きい。また、勝手な変換では汎化能力が出るとは思えない

高次元空間への写像: 問題点

- 計算時間:
  - データが1001個あれば、(非線形関数で)1000次元に写像すれば、必ず、線形分離できる。
  - しかし、非線形関数の計算は時間がかかるのに、データ1個につき1000回の計算(共通部分を多くして計算するにせよ)が必要では、大変な計算量となる

  ⇒ カーネル関数の利用(カーネルトリック)による、計算量の大幅な削減

- 汎化能力:
  - データが1001個あれば、(非線形関数で)1000次元に写像すれば、必ず、線形分離できる。
  - それは、すなわち、どんな教師データであっても、それを実現できるということ。すなわち、過学習!!

  ⇒ この問題の解決こそ、ラージマージン分類器の本領

高次元空間への非線形写像

- データ x から高次元空間 F への(非線形)写像 $\phi(x)$ を考える. $\phi: R^N \to F$

主問題のラグランジアンは

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \sum_i \alpha_i \left(y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1\right)$$

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \sum_i \alpha_i \left(y_i(\mathbf{w}\cdot\phi(\mathbf{x}_i) + b) - 1\right)$$

双対問題: 拘束条件付き最大化

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j$$

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0 \text{ and } \forall i\ \alpha_i \geq 0$$

カーネルトリック "Kernel Trick"

- 注目すべきは、$\Phi(x)$ は、$\Phi(x)\cdot\Phi(y)$ というように、内積でしか表れない。

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$$

- そこで、もし、$K(x,y)=\Phi(x)\cdot\Phi(y)$ となる、簡単な関数Kがあれば、計算が非常に楽になる。
  - 特に、$K(x,y)$ が $x\cdot y$ の関数であるとさらに楽になる

よく使われるカーネル関数

- 線形カーネル $K(x,y) = x^T y$
- 多項式カーネル $K(x,y) = (x^T y + 1)^p \text{ or } (x^T y)^p$
- RBFカーネル $K(x,y) = \exp\left(-\|x-y\|^2 / 2\sigma^2\right)$
- MLP $K(x,y) = \tanh\left(\beta_0 x^T y + \beta_1\right)$ ← 半正定値ではない
- 例: 2-次元ベクトル $\mathbf{x}=[x_1\ x_2]$ に対し $K(\mathbf{x}_i,\mathbf{x}_j)=(1+\mathbf{x}_i^T\mathbf{x}_j)^2$,とおく
  このとき、次の式が成立する $K(\mathbf{x}_i,\mathbf{x}_j)=\varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$:

  $K(\mathbf{x}_i,\mathbf{x}_j)=(1+\mathbf{x}_i^T\mathbf{x}_j)^2$,
  $= 1+ x_{i1}^2x_{j1}^2 + 2\,x_{i1}x_{j1}\,x_{i2}x_{j2} + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$
  $= [1\ \ x_{i1}^2\ \sqrt{2}\,x_{i1}x_{i2}\ \ x_{i2}^2\ \sqrt{2}x_{i1}\ \sqrt{2}x_{i2}]^T\ [1\ \ x_{j1}^2\ \sqrt{2}\,x_{j1}x_{j2}\ \ x_{j2}^2\ \sqrt{2}x_{j1}\ \sqrt{2}x_{j2}]$
  $= \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$
  ただし $\varphi(\mathbf{x}) = [1\ \ x_1^2\ \sqrt{2}\,x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \sqrt{2}x_2]$

SVM: 汎化能力の推定

- 汎化能力最大(新規データに対して最も正確)の分類器がほしい.
- 良い汎化性能を得るための糸口は?
  - 訓練データを大きくする
  - 訓練データに対する誤りを小さくする
  - 容量/分散 (モデル記述パラメータ数, モデルの表現能力) をおおきくする
- SVM では、これらの量に基づいて、新規データに対する誤差限界を明示的に示すことができる.

容量/分散: VC 次元

- 理論的なリスク限界:

$$R(\alpha) \le R_{emp}(\alpha) + \sqrt{\frac{h(\log(2l/h)+1) - \log(\eta/4)}{l}}$$

- Risk = 平均誤り率
- α – 当該モデル (パラメータで決まる)
- R_{emp} – 経験リスク, l – 観測数, h – VC 次元, 当該式は確率 (1-η) で正しい
  - VC (Vapnik-Chervonenkis) 次元/容量: shatterできる点の最大数
  - ある点集合がshatterできるとは、その任意のラベル付けを当該分類器が行えること.
- 重要な理論的性質. しかし, 実際にはあまり使われない

例: 超平面のVC次元

- d 次元空間に n 個の点があり、それらは、red か green とラベルが付けられていると仮定する. n を (d の関数として) どれだけ大きくとれば, red 点と green 点が線形分離でなくなる例が作れるか?
- 例, d =2 に対しては $n \ge 4$.

スケッチ: マージン最大化の理論的な正当化

- Vapnik は次のことを証明した:
  マージンが ρ 以上の線形判別器クラスの VC 次元 h は、次の上界をもつ

$$h \le \min\left\{\left\lceil \frac{D^2}{\rho^2} \right\rceil, m_0\right\} + 1$$

  ただし D は訓練事例をすべて囲い込む最小の超球の直径, そして m_0 は(事例の表現空間の)次元である.

- 直感的に, これは空間の次元 m_0 にかかわりなく, マージン ρ を最大化することにより、VC 次元を最小化することができる.
- こうして, 分類器の複雑度は、次元数に関わりなく小さく保つことができる.

Vapnik 1982: Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics) Springer-Verlag New York, Inc.

SVM の性能

- SVM は、最良の性能を持つと考える人は多い.
- 統計的な有意性は明確な場合もそうでない場合も.
- SVM と同程度の性能をもつ手法は他にもある.
- 例: regularized logistic regression (Zhang & Oles)
  - Tong Zhang, Frank J. Oles: Text Categorization Based on Regularized Linear Classification Methods. Information Retrieval 4(1): 5-31 (2001)
- 比較研究の例: Yang & Liu
  - Yiming Yang, Xin Liu: A re-examination of text categorization methods, 22nd Annual International SIGIR (1999).

SVM回帰 (Support Vector Regression)

SVM は

線形モデルを用いて分類を行う手法(の一つ):

$$y(x) = w^T \phi(x) + b$$

そのSVMを用いて回帰を行おう

ある見方:
コスト関数 ＝ 誤差関数+正則化項

線形回帰で、次の誤差関数の最小化を図る:

$$\frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2 + \frac{\lambda}{2}\|w\|^2$$

この二乗誤差部分を ε–insensitive 誤差関数で置き換える:

$$C\sum_{n=1}^{N}E_\varepsilon(y(x_n)-t_n)+\frac{1}{2}\|w\|^2$$

ε–insensitive 誤差関数の例:

$$L_\varepsilon(y)=\begin{cases}0 & \text{for} \quad |f(x)-y|<\varepsilon \\ |f(x)-y|-\varepsilon & \text{otherwise}\end{cases}$$

Gunn. Support Vector Machines for Classification and Regression

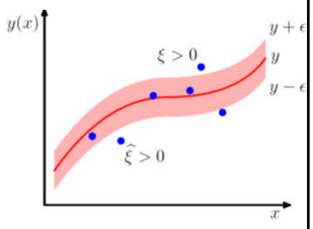
スラック変数の導入

目標値がこの ε-tube の中にあるためには:

$$y_n - \varepsilon \le t_n \le y_n + \varepsilon$$

スラック変数を導入して、ε-tube の外側に点があることを許すには:

$$t_n \le y(x_n) + \varepsilon + \xi_n$$
$$t_n \ge y(x_n) - \varepsilon - \xi_n^-$$



Support Vector Regression の最適化問題

Minimize:

$$C\sum_{n=1}^{N}(\xi_n+\xi_n^-)+\frac{1}{2}\|w\|^2$$

Subject to:

$$\xi_n \ge 0 \quad \text{かつ} \quad t_n \le y(x_n)+\varepsilon+\xi_n$$
$$\xi_n^- \ge 0 \quad\quad\quad t_n \ge y(x_n)-\varepsilon-\xi_n^-$$

ラグランジュ関数

最小化問題:

$$L = C\sum_{n=1}^{N}(\xi_n+\xi_n^-)+\frac{1}{2}\|w\|^2-\sum_{n=1}^{N}(\mu_n\xi_n+\mu_n^-\xi_n^-)-\sum_{n=1}^{N}a_n(\varepsilon+\xi_n+y_n-t_n)-\sum_{n=1}^{N}a_n^-(\varepsilon+\xi_n^--y_n+t_n)$$

$$\frac{\partial L}{\partial w}=0 \Rightarrow w=\sum_{n=1}^{N}(a_n-a_n^-)\phi(x_n)$$

$$\frac{\partial L}{\partial b}=0 \Rightarrow \sum_{n=1}^{N}(a_n-a_n^-)=0$$

$$\frac{\partial L}{\partial \xi_n}=0 \Rightarrow a_n+\mu_n=C$$

$$\frac{\partial L}{\partial \xi_n^-}=0 \Rightarrow a_n^-+\mu_n^-=C$$

双対方程式

最大化:

$$W(a,a^-)=-\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n-a_n^-)(a_m-a_m^-)k(x_n,x_m)-\varepsilon\sum_{n=1}^{N}(a_n+a_n^-)+\sum_{n=1}^{N}(a_n-a_n^-)t_n$$

$$0\le a_n\le C$$

$$0\le a_n^-\le C$$

予測:

$$y(x)=\sum_{n=1}^{N}(a_n-a_n^-)k(x,x_n)+b$$

まとめ

- サポートベクターマシン (SVM) は
  - サポートベクターに基づいて超平面を決める
    - Support vector = 判定境界付近のクリティカルな点
  - 線形 SVM は線形分類器.
  - カーネル: 高次元へ写像するが、その内積は低次元の内積で簡単に計算できる
  - リスクの上界 (リスク = テストデータでの期待誤り)
  - (邪魔な属性が多いときの) 分類器としてベスト?
    - 数1000も属性があるときは、安定的に強い
  - ポピュラー: SVMlight がきっかけ?
    - 速くて無料 (研究目的には)
    - 他にもいくつか: TinySVM, libsvm, ….
- 回帰にも使える (support vector regression)

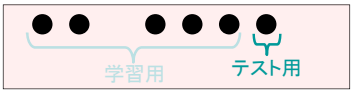
Rでは、少し面倒です。

Rにおけるクロスバリデーション

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

R で 10-fold cross validation (1)

パッケージ bootstrap などが使えるが、まずは、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。

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、まずは、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
```

bootstrap 中の crossval

R で 10-fold cross validation

パッケージ bootstrap 中の crossval を用いる
使い方が少々面倒なので、プログラム全体を記す。
iris データに rpart を minsplit=30 で行った結果である。

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

library(bootstrap)              # crossval will be used
theta.fit <- function (x,y) {
  tmp <- data.frame( x, class=y)
  return( rpart(class~., tmp, control=rpart.control(minsplit=30) ) )
}
theta.predict <- function( fit, x) {predict( fit, data.frame(x), type="class" ) }
results <- crossval( iris[,-5],iris[,5], theta.fit, theta.predict, ngroup=10)
(cm <- table( iris[,5], results$cv.fit ))
(accuracy <- sum(diag(cm))/sum(cm))
```

「5」はクラス変数のある列番号

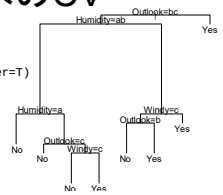
```
> (cm <- table( iris[,5], results$cv.fit ))

                1  2  3
  Iris-setosa  50  0  0
  Iris-versicolor  0 47  3
  Iris-virginica   0  6 44
> (accuracy <- sum(diag(cm))/sum(cm))
[1] 0.94
```

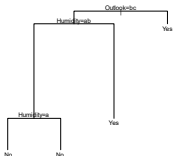
分割はランダムに行われるので、実験ごとに結果は異なっても不思議ではない。

R で試す決定木のCV

```
library(rpart)
setwd("D:/R/Sample")
playTennis <- read.csv("07PlayTennis02.csv", header=T)
(playTennis.tr <- rpart(Play~ ., playTennis,
    control=rpart.control(minsplit=1)) )
plot(playTennis.tr); text(playTennis.tr)
```




```
library(rpart)
setwd("D:/R/Sample")
playTennis <- read.csv("07PlayTennis02.csv", header=T)
(playTennis.tr <- rpart(Play~ ., playTennis,
    control=rpart.control(minsplit=3)) )
plot(playTennis.tr); text(playTennis.tr)
```



R で試す決定木のCV

下記の方法で CV ができる（ngroupがCV時の分割個数を表す）

```
library(rpart)
setwd("D:/R/Sample")
xy <- read.csv("07PlayTennis02.csv", header=T)

library(bootstrap)              # crossval を使用する
theta.fit <- function (x,y) {
  tmp <- data.frame( x, class=y)
  return( rpart(class~., tmp, control=rpart.control(minsplit=1) ) )
}
theta.predict <- function( fit, x) {predict( fit, data.frame(x), type="class" ) }
results <- crossval( xy[,-5], xy[,5], theta.fit, theta.predict, ngroup=7)
(cm <- table( xy[,5], results$cv.fit ))
(accuracy10CV <- sum(diag(cm))/sum(cm))
```

決定木の複雑さを制御するパラメータは minisplit であるので、これを変えて、CV を試みる。なお、本例では、7-fold CV とした

補足: R で試してみる決定木のCV

注: data.frame(x) は x がデーター個のときと2個以上のときとでは振る舞いが異なるため、上記のプログラムで、次のようなエラーが発生することがある。
下記の例では、ngroup=8 としたため（データ数は15なので）、テストデータ数が1個となることがあり、その時エラーが発生した。
Leave-one-out の実験を行うときは、データの作成方法を変更する必要がある。

```
> results <- crossval(xy[,-5], xy[,5], theta.fit, theta.predict, ngroup=8)
 以下にエラー eval(expr, envir, enclos) :
   オブジェクト 'Outlook' がありません
```

補足: R で試してみる決定木のCV

下記のエラーがでてしまうことがある。これは、R で factor を用いるときに発生しうる問題である

```
> playTennis <- read.csv("07PlayTennis02.csv", header=T)
> library(bootstrap)
> theta.fit <- function (x,y) {
+   tmp <- data.frame( Outlook=x[,1], Temperature=x[,2],
+           Humidity=x[,3], Windy=x[,4], class=y)
+   return( rpart(class~., tmp, control=rpart.control(minsplit=30) ) )
+ }
> theta.predict <- function( fit, x) {predict( fit, data.frame(x), type="class" ) }
> results <- crossval(playTennis[,-5],playTennis[,5], theta.fit, theta.predict, ngroup=3)
 以下にエラー model.frame.default(Terms, newdata, na.action = na.action, xlev = attr(object,  :
  factor 'Temperature' has new level(s)  Hot
```

本日の課題

- iris データを SVM で分析（Speciesを予測するように学習）してみよう。Kernel を rbfやpolyとしてみよう

```
library(e1071)
# iris を使えるようにする
#
data(iris)
# 属性名を調べる( head(iris) でデータを見る方がよい)
names(iris)
```

- 先週と同じく、文字認識データを用いてみよう。Kernel を替え、10-fold cv を行って、どのkernel がよいか確認しよう。Cの値は、default値にしておこう