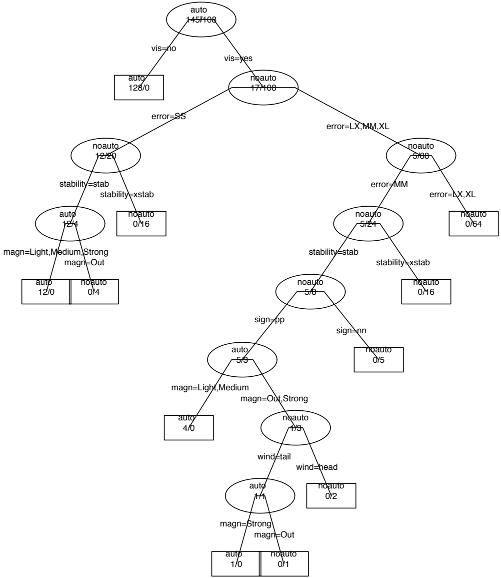
Notes

- Class on Thursday, Mar 25
- Takehome MT due Mar 25
- Trees and forests; Nearest neighbours and prototypes (Ch. 13)
- Unsupervised Learning: Cluster analysis and Self-Organizing Maps (Ch. 14)
- Netflix Prize: some details on the models and methods
- `www.fields.utoronto.ca/programs/scientific/`

A Decision Tree (Ripley, 1996)



Shuttle lander decision tree

```
> library(MASS)
> library(rpart)
> data(shuttle)
> shuttle[1:10,]
   stability error sign wind   magn vis  use
1      xstab    LX   pp head  Light  no auto
2      xstab    LX   pp head Medium  no auto
3      xstab    LX   pp head Strong  no auto
4      xstab    LX   pp tail  Light  no auto
5      xstab    LX   pp tail Medium  no auto
6      xstab    LX   pp tail Strong  no auto
7      xstab    LX   nn head  Light  no auto
8      xstab    LX   nn head Medium  no auto
9      xstab    LX   nn head Strong  no auto
10     xstab    LX   nn tail  Light  no auto
> ?shuttle
```

... shuttle lander

```
> shuttle.rp = rpart(use ~ ., data = shuttle, minbucket = 0,
+  xval = 0, maxsurrogate = 0, cp=0, subset = 1:253)
>  # from the MASS scripts; the default tree is much simpler
> post(shuttle.rp,horizontal = F, height = 10, width = 8,
+ title = "", pointsize = 8, pretty = 0)  #finally a nice look
> summary(shuttle.rp)
Call:
rpart(formula = use ~ ., data = shuttle, subset = 1:253,
minbucket = 0,   xval = 0, maxsurrogate = 0, cp = 0)
  n= 253

          CP nsplit  rel error
1 0.84259259      0 1.00000000
2 0.03703704      1 0.15740741
3 0.00925926      4 0.04629630
4 0.00462963      8 0.00925926
5 0.00000000     10 0.00000000
```

Reference: Chapter 9 of Venables & Ripley, MASS

Random Forests Ch. 15

- trees are highly interpretable, but also quite variable
- bagging (bootstrap aggregation) resamples from the data to build B trees, then averages
- if $X_1, \ldots, X_N$ independent (μ, σ^2), then $\mathrm{var}(\bar{X}) = \sigma^2/B$
- if $\mathrm{corr}(X_i, X_j) = \rho > 0$, then

$$\mathrm{var}(\bar{X}) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

- $\to \rho\sigma^2$ as $B \to \infty$; no benefit from aggregation
- $$\frac{\sigma^2}{B}\{1 + \rho(B-1)\}$$
- average many trees as in bagging, but reduce correlation using a trick: use only a random sample of m of the p input variables each time a node is split
- $m = O(\sqrt{p})$, for example, or even smaller

... random forests

588 15. Random Forests

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

... random forests

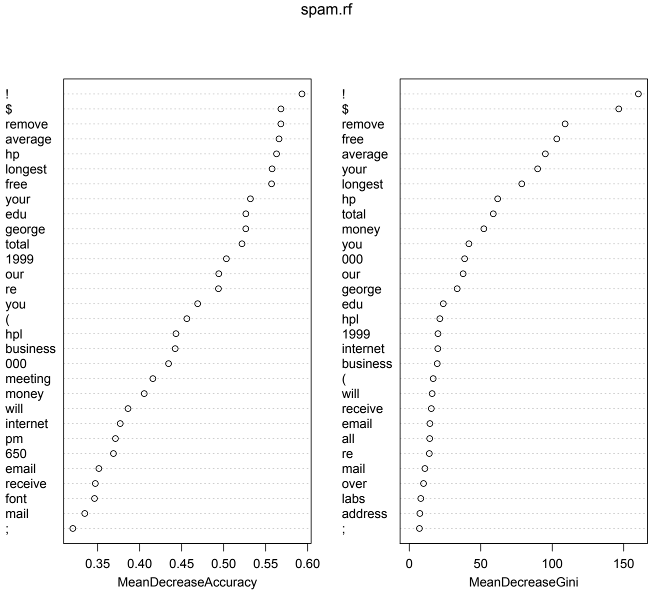
- email spam example in R
- Figures 15.1, 4, 5

```
> spam2 = spam
> names(spam2)=c(spam.names,"spam")
> spam.rf = randomForest(x=as.matrix(spam2[spamtest==0,1:57]),
            y=spam2[spamtest==0,58] , importance=T)
> varImpPlot(spam.rf)
>  table(predict(spam.rf, newdata = as.matrix(spam2[spamtest==1,])),spam2[spamtest==1,58])

        email spam
  email   908   38
  spam     33  557
> .Last.value/sum(spamtest)

           email     spam
  email 0.591146 0.024740
  spam  0.021484 0.362630
> .0247+.02148
[1] 0.04618
```

... random forests

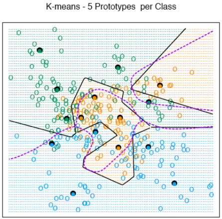


Prototype and nearest neighbour methods: Ch. 13

- model free, or “black-box” methods for classification
- related to unsupervised learning (Ch. 14)
- training data $(x_1, g_1), \ldots (x_N, g_N)$: g indicates one of K classes
- reduce $x_1, \ldots, x_N$ to a (small) number of “prototypes”
- classify new observation by the class of its closest prototype
- “close”: Euclidean distance
- need to center and scale training data x’s
- how many prototypes, and where to put them

K-means clustering

- K refers to the number of clusters!, not the number of classes: book uses R for this
- start with a set of cluster centers, for each center identify its cluster (training x's)
- compute the mean of this cluster of training points, make this the new cluster center
- usually start with R randomly selected points
- with "labelled data" (§13.2.1) apply this cluster algorithm within each of the K classes
- Figure 13.1 (top)

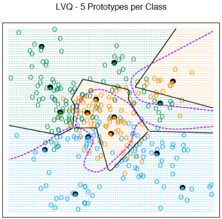


... generalizations

- learning vector quantization (§13.2.2) allows observations from other classes to influence prototypes in class k: see Algorithm 13.1
- Figure 13.1 (bottom)

Algorithm 13.1 *Learning Vector Quantization—LVQ.*

1. Choose R initial prototypes for each class: $m_1(k), m_2(k), \ldots, m_R(k)$, $k = 1, 2, \ldots, K$, for example, by sampling R training points at random from each class.
2. Sample a training point x_i randomly (with replacement), and let (j, k) index the closest prototype $m_j(k)$ to x_i.
   (a) If $g_i = k$ (i.e., they are in the same class), move the prototype towards the training point:
   $$m_j(k) \leftarrow m_j(k) + \epsilon(x_i - m_j(k)),$$
   where ϵ is the *learning rate*.
   (b) If $g_i \neq k$ (i.e., they are in different classes), move the prototype away from the training point:
   $$m_j(k) \leftarrow m_j(k) - \epsilon(x_i - m_j(k)).$$
3. Repeat step 2, decreasing the learning rate ϵ with each iteration towards zero.



... generalizations

- Gaussian mixture modelling (§13.2.3) assumes

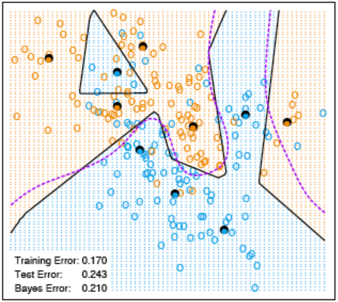
$$Pr(X \mid G = k) = \sum_{r=1}^{R} \pi_{kr} \phi(X; \mu_{kr}, \Sigma)$$

- same flavour as linear discriminant analysis
- π_{kr} are unknown mixing probabilities, to be estimated along with μ_{kr}, Σ
-

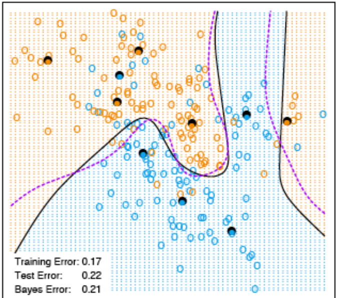
$$Pr(G = k \mid X = x) = \frac{\sum_{r=1}^{R} \pi_{kr} \phi(x; \mu_{kr}, \Sigma) \Pi_k}{\sum_{\ell=1}^{K} \sum_{r=1}^{R} \pi_{\ell r} \phi(x; \mu_{\ell r} \Sigma) \Pi_\ell} \qquad (12.60)$$

  with Π_k the prior class probabilities
- here same number of prototypes R in each class; could let this vary with class
- usually assume $\Sigma = \sigma^2 I$ scalar covariance matrix
- Figure 13.2

K-means - 5 Prototypes per Class



Gaussian Mixtures - 5 Subclasses per Class



Reminder: Bayes boundary

- $$pr(G = k \mid x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^{K} f_\ell(x)\pi_\ell}$$
- In Figures 13.1, 13.2, etc., $x = (x_1, x_2)$
- data is simulated from known f_k with known probability π_k
- $pr(G = k \mid x_0)$ can be calculated for any x_0 in R^2
- x_0 assigned to, e.g., class 2 if

$$pr(G = 2 \mid x_0) > pr(G = 1 \mid x_0), pr(G = 3 \mid x_0), \quad (2.23)$$

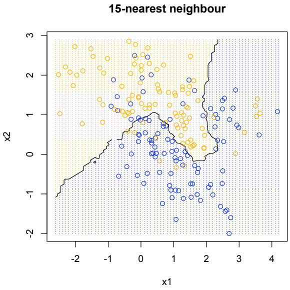
- `MASS` scripts (Ch. 12) give code for drawing a continuous boundary
- code from Jean-François for SVMs uses `expand.grid` and colors to indicate boundary
- `boundaries(y, b, n=100)`

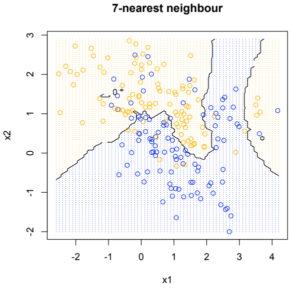
k-nearest-neighbours

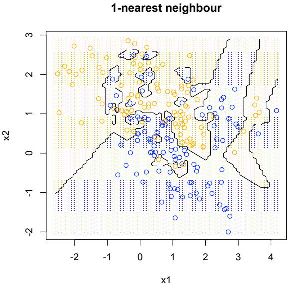
- classify new point x_0 using majority vote among k training points x that are closest to x_0
- if features are continuous, use Euclidean distance (after standardizing)
- Cover & Hart: error rate of 1-nearest neighbour asymptotically bounded above by twice Bayes rate
- asymptotic with size of training set
- can be used as a rough guide to the best possible error rate ($1/2$ the 1-nn rate) (p.468)
- LandSat data: Figure 13.5, 13.6
- refinements for improvements: tangent distance (§13.3.3), adaptive neighbourhoods (§13.4), dimension reduction (§13.5)

... *k*-nearest-neighbours

```
> data(mixture.example) # see ElemStatLearn
> x = mixture.example$x
> g = mixture.example$y
> xnew = mixture.example$xnew # gridpoints
> library(class)
> mod15 <- knn(x, xnew, g, k=15, prob=TRUE)
> prob = attr(mod15, "prob")
> prob <- ifelse( mod15=="1", prob, 1-prob)
> px1 <- mixture.example$px1
> px2 <- mixture.example$px2
> prob15 <- matrix(prob, length(px1), length(px2))
> contour(px1, px2, prob15, levels=0.5, labels="", xlab="x1",
+ ylab="x2", main = "15-nearest neighbour")
> points(x, col=ifelse(g==1, "orange", "blue"))
> points(xnew, col = ifelse(prob15 > 0.5, "orange","blue"),
+ pch=".", cex=0.8)
```


15-nearest neighbour
x1
x2


7-nearest neighbour
x1
x2


1-nearest neighbour
x1
x2

Unsupervised Learning (Ch 14)

- training sample $(x_1, \ldots, x_N)$ with p features
- no response y
- want information on the probability function (density) of $X = (X_1, \ldots, X_p)$ based on these N observations
- if $p = 1$ or 2, can use kernel density estimation as in §6.6
- we also used density estimation to construct a classifier, via Naive Bayes
- goal: subspaces of feature space (R^p) where $pr(X)$ is large: principal components, multidimensional scaling, self-organizing maps, principal curves
- search for latent variables of lower dimension
- regression with missing response variable
- goal: decide whether $pr(X)$ has small number of modes (= clusters)
- classification with missing class variable
- no loss function to ascertain/estimate how well we're doing
- best viewed as descriptive: plots important
- exploratory data analysis

Cluster Analysis (§14.3)

- discover groupings among the cases; cases within clusters should be 'close' and clusters should be 'far apart'
- Figure 14.4
- many (not all) clustering methods use as input an $N \times N$ matrix D of dissimilarities
- require $D_{ii'} > 0$, $D_{ii'} = D_{i'i}$ and $D_{ii} = 0$
- sometimes the data are collected this way (see §14.3.1)
- more often D needs to be constructed from the $N \times p$ data matrix
- often (usually) $D_{ii'} = \sum_{j=1}^{p} d_j(x_{ij}, x_{i'j})$, where $d_j(\cdot, \cdot)$ to be chosen, e.g. $(x_{ij} - x_{i'j})^2$, $|x_{ij} - x_{i'j}|$, etc.
- sometimes $D_{ii'} = \sum_{j=1}^{p} w_j d_j(x_{ij}, x_{i'j})$, with weights to be chosen
- pp 504, 505
- this can be done using `dist` or `daisy` (the latter in the R library `cluster`)

... cluster analysis

- dissimilarities for categorical features
- binary: simple matching uses

$$D_{ii'} = (\#\{(1,0) \text{ or } (0,1) \text{ pairs }\})/p$$

Jacard coefficient uses

$$D_{ii'} = (\#\{(1,0) or (0,1) \text{ pairs }\})/(\#\{(1,0),(0,1) \text{ or } (1,1) \text{ pairs }\})$$

- ordered categories – use ranks as continuous data (see eq. (14.23))
- unordered categories – create binary dummy variables and use matching

... cluster analysis

```
dist(x, method = c("euclidean", "maximum",
"manhattan", "canberra", "binary", "minkowski"))
```

where `maximum` is $\max_{1 \le j \le p}(x_{ij} - x_{i'j})$ and `binary` is Jacard coefficient.

```
daisy(x, metric=c("euclidean", "manhattan", "gower")
standardize=F, type=c("ordratio","logratio","asymm","symm")
```

(see the help files)

```
> x = matrix(rnorm(100),nrow=5)
> dim(x)
[1]  5 20
> dist(x)
         1        2        3        4
2 5.493679
3 6.360923 5.652732
4 7.439924 5.885949 7.960187
5 4.437444 3.679995 6.133873 5.936607
```

Combinatorial algorithms

suppose number of clusters K is fixed ($K < N$)
$C(i) = k$ if observation i is assigned to cluster k

$$\begin{aligned}
T &= \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N} D_{ii'} \\
&= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\left(\sum_{C(i')=k} D_{ii'} + \sum_{C(i')\neq k} D_{ii'}\right) \\
&= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} D_{ii'} + \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')\neq k} D_{ii'} \\
&= W(C) + B(C)
\end{aligned}$$

$W(C)$ is a measure of within cluster dissimilarity
$B(C)$ is a measure of between cluster dissimilarity
T is fixed given the data: minimizing $W(C)$ same as maximizing $B(C)$

K-Means clustering (§14.3.6)

- most algorithms use a 'greedy' approach by modifying a given clustering to decrease within cluster distance: analogous to forward selection in regression
- K-means clustering is (usually) based on Euclidean distance: $D_{ii'} = ||x_i - x_{i'}||^2$, so x's should be centered and scaled (and continuous)
- Use the result

$$\frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k}||x_i - x_{i'}||^2 = \sum_{k=1}^{K} N_k \sum_{C(i)=k}||x_i - \bar{x}_k||^2$$

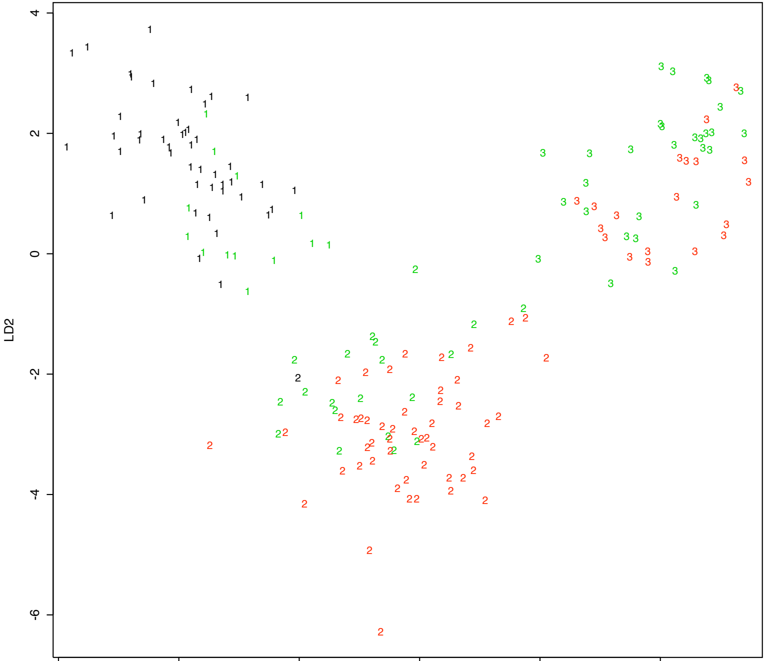
  where N_k is the number of observations in cluster k and $\bar{x}_k = (\bar{x}_{1k}, \dots, \bar{x}_{pk})$ is the mean in the kth cluster
- The algorithm starts with a current set of clusters, and computes the cluster means. Then assign observations to clusters by finding the cluster whose mean is closest. Recompute the cluster means and continue.

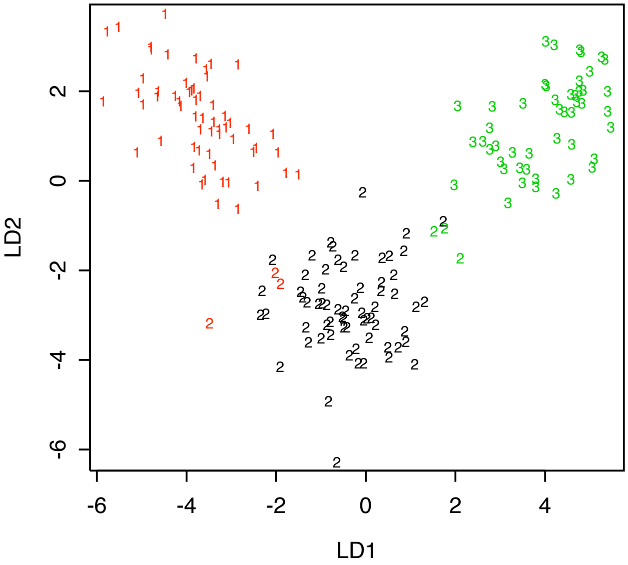
- sometimes require cluster center to be one of the data values (means that algorithm can be applied to dissimilarity matrices directly)
- choose K by possibly plotting the total within cluster dissimilarity vs. K; it is always decreasing but a 'kink' may be evident (see §14.3.11).
- hard to describe the results of partitioning methods of clustering, Figure 14.6
- Algorithm 14.1:
  - for a given cluster assignment, minimize the total cluster variance $\sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - m_k||^2$ with respect to $\{m_1, \ldots, m_K\}$; this is easily achieved by taking each m_k to be the sample mean of the kth cluster
  - For a given set of $\{m_k\}$, minimize distance by letting $C(i) = \text{argmin}_{1 \leq k \leq K} ||x_i - m_k||^2$

Example: wine data

.

- recall 3 classes, 13 feature variables
- linear discriminant analysis showed a good separation of the 3 classes
- K-means with a random choice of initial cluster
- again on standardized data


LD2
4
2
0
-2
-4
-6



LD1
LD2
-6
-4
-2
0
2
4

Partitioning methods

- K-Means – uses the original data
- uses Euclidean distance $D_{ii'} = \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2$
- requires a starting classification
- minimizes the within-cluster sum of squares
- maximizes the between-cluster sum of squares
- variables should be 'suitably scaled' (Ripley): no mention of this in HTF
- K-medioids: replace Euclidean by another dissimiilarity measure

$$D_{ii'} = \sum_{j=1}^{p} |x_{ij} - x_{i'j}| \quad \text{manhattan}$$

$$D_{ii'} = \sum_{j=1}^{p} \frac{|x_{ij} - x_{i'j}|}{|x_{ij} + x_{i'j}|} \quad \text{Canberra}$$

Dissimilarities for categorical features

- binary: simple matching uses

$$D_{ii'} = (\#\{(1,0) \text{ or } (0,1) \text{ pairs }\})/p$$

Jacard coefficient uses

$$D_{ii'} = (\#\{(1,0) or (0,1) \text{ pairs }\})/(\#\{(1,0),(0,1) \text{ or } (1,1) \text{ pairs }\})$$

- ordered categories – use ranks as continuous data (see eq. (14.23))
- unordered categories – create binary dummy variables and use matching
- mixed categories – Gower's 'general dissimilarity coefficient' – see Gordon

Constructing dissimilarity matrices

```
dist(x, method = c("euclidean", "maximum",
"manhattan", "canberra", "binary"))
```

where `maximum` is $\max_{1 \leq j \leq p}(x_{ij} - x_{i'j})$ and `binary` is Jacard coefficient.

```
daisy(x, metric=c("euclidean", "manhattan",
standardize=F, type=c("ordratio","logratio","asymm"
```

(see the help files)