NMST539

Mnohorozměrná analýza
Multivariate Analysis

Lecture notes II (predictive topics)

Ivan Mizera

Classification: generalities

Classification

Classification: we assign items (*samples*) into groups (*labels*), on the basis of observed values of various variables - classifiers (*features*)

Supervised classification: we *know* the groups where the items have to be assigned in advance, typically given a *working* (*training*) dataset on which we are able to observe those; the emphasis is then on obtaining a good *rule* (*learning*) for classification of *future* items: it is a sort of *predictive* method (aka "learning"): the rule predicts the group from the values of classifiers

Unsupervised classification: we *do not know* the groups *in advance*. Can be viewed as similar to the supervised classification setting in which the labels of group membership have been lost. The emphasis is, however, not on classifying future items, but rather on items in the working datasets

Classification: supervised

The first task that initiated the development of so-called statistical (machine) learning was classification - more precisely, what is called *supervised* classification

The adjective "supervised" means that we know not only the number of groups that we would have to classify into (like it is also somewhat true for partitioning methods in clustering), but for all the items in the working (training) dataset we also know the category where each item belongs to (its "*label*"). That usually implies that groups are reasonably well defined.

Classifying into one group would be trivial; two groups are thus a minimum. (There are quite a few methods that work predominantly only for two groups - but most of the methods can be applied also to any finite number of groups; the number of groups is typically not that large. A category "undecided" can be made an extra group, if so desired.)

Prediction and the analogy with regression

The task of supervised classification is quite analogous to that of a regression relationship - the difference is that the response variable is categorical ("which group" - label) rather than numerical. The predicting variables (covariates, predictors, features) have pretty much the same character as in regression; may be numeric or categorical or both.

Predicting the group of the the items in the working (training) dataset is of low importance in *supervised* classification - we already know the group they belong to. What is important is to obtain a *rule* that will classify future data items in similar situations (that is, the same or similar structure of predictors)

Once a rule is there, we can check how well it would predict the training data (confronting predictions to reality) - but as a rule, such checks are not very useful. What says much more about the quality of a rule is its evaluation on some the data that did not help creating it: on *validation* dataset(s).

Classification: unsupervised

In supervised classification, we know what the groups are, and know for all the items in the working (training) dataset which group they belong to

If we do not know what the groups are, then we speak about *unsupervised* classification. For those understanding supervised classification, the unsupervised one is often explained as "a supervised classification when the labels were lost"; all the predictors are still there, only the response is missing. (This explanation may be quite illuminating, but it should be remembered that it does not go all the way.)

(A situation in which we may know the groups to some extent, and only for *some* items of the working (training) dataset is called *semi-supervised* classification, and is beyond the scope of this course.)

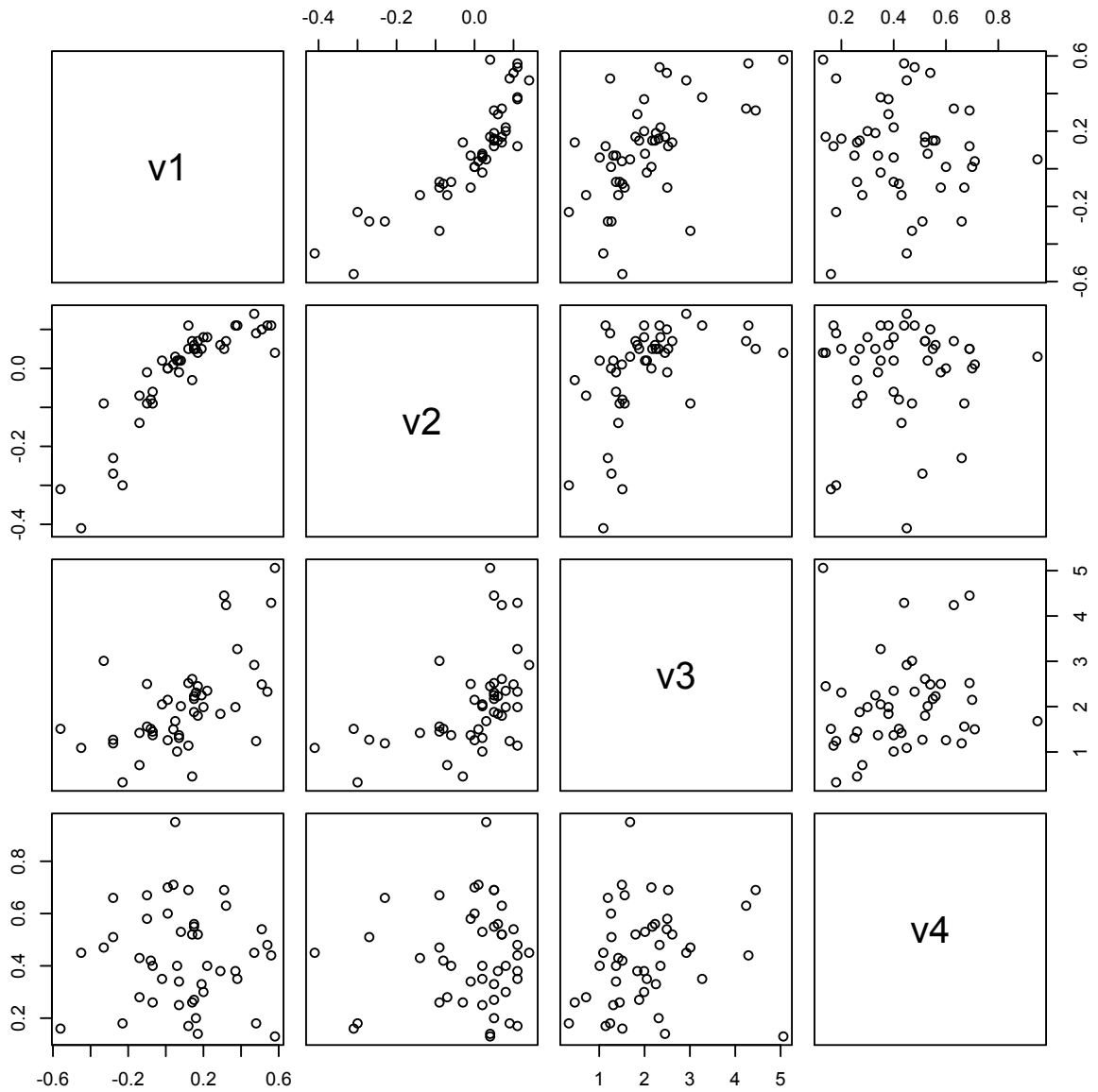
Ouverture:
cluster analysis as unsupervised classification

Clustering as unsupervised classification

Clustering, cluster analysis, is a (pretty much exhausting) instance of unsupervised classification. As a rule, we do not know what the groups are, often even not their number - but we still would like to determine them somehow

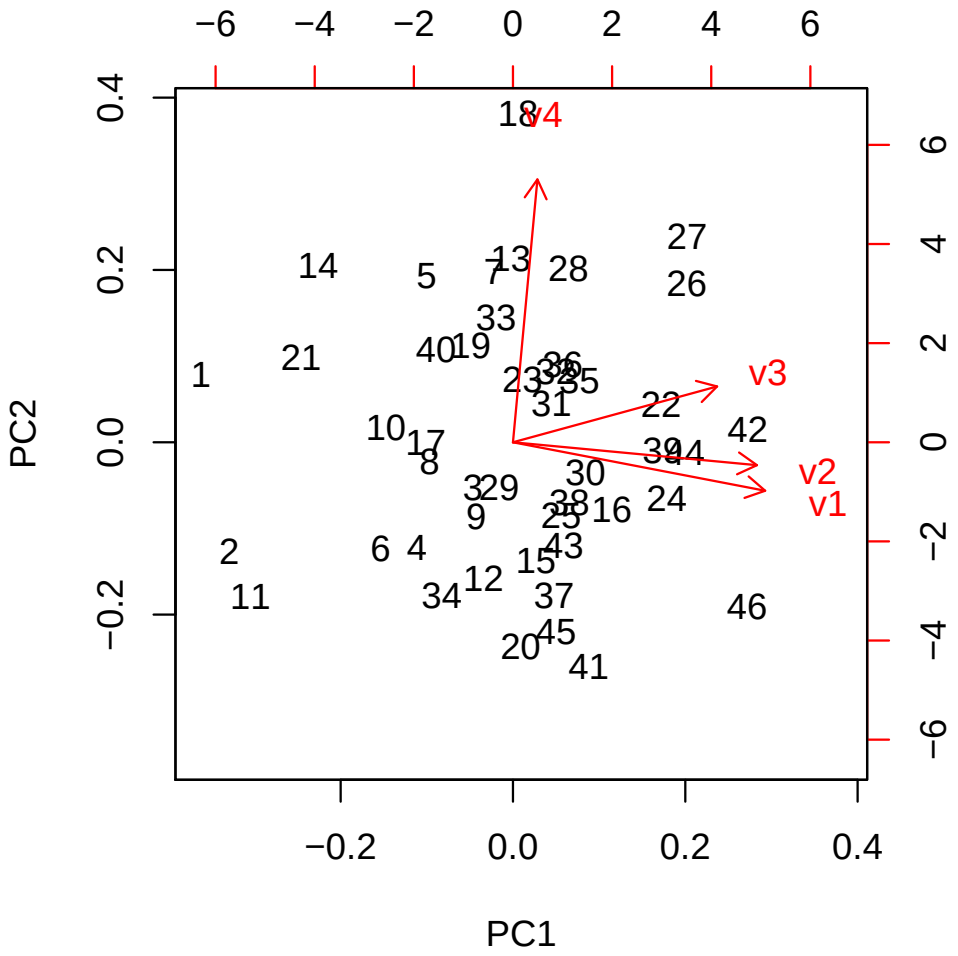
The following toy examples were created from supervised classification datasets by omitting the labels

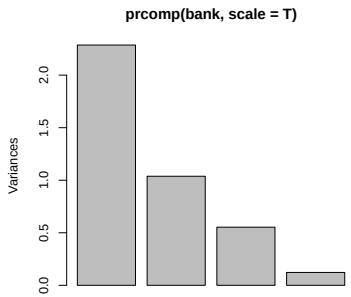
Example dataset: about bank clients




```
> pairs(Bank[,1:4])
```

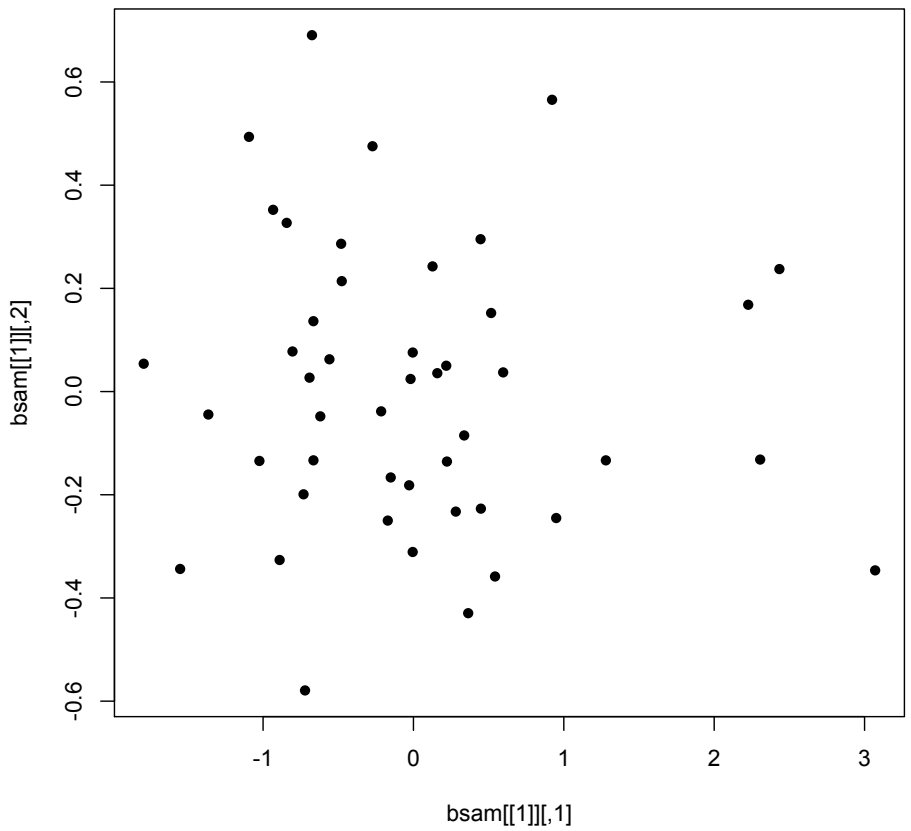
Seeing the data: principal components





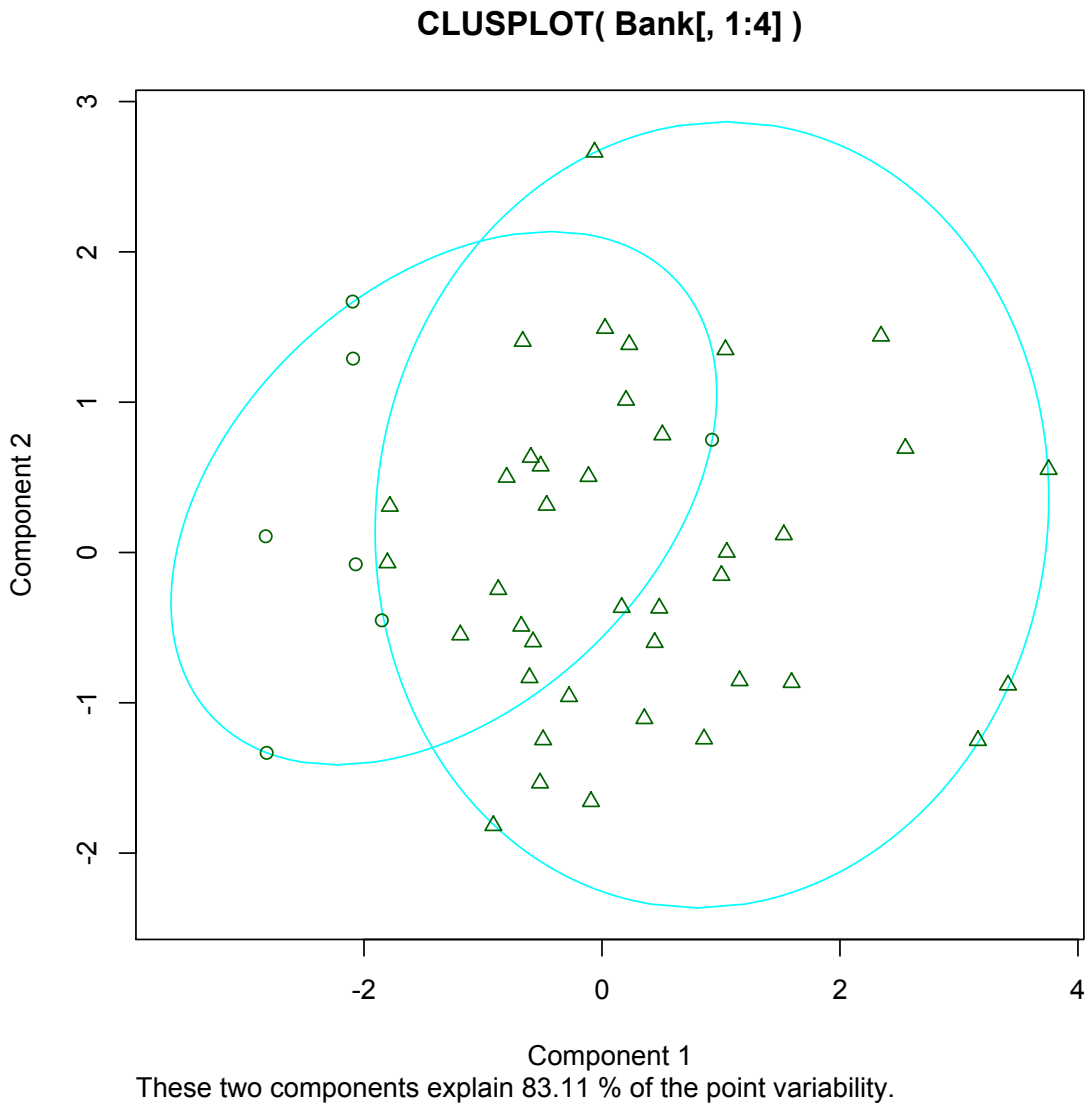

```
> biplot(prcomp(Bank[,1:4],scale=T))
> plot(prcomp(Bank[,1:4],scale=T))
```

Seeing the data: Sammon mapping



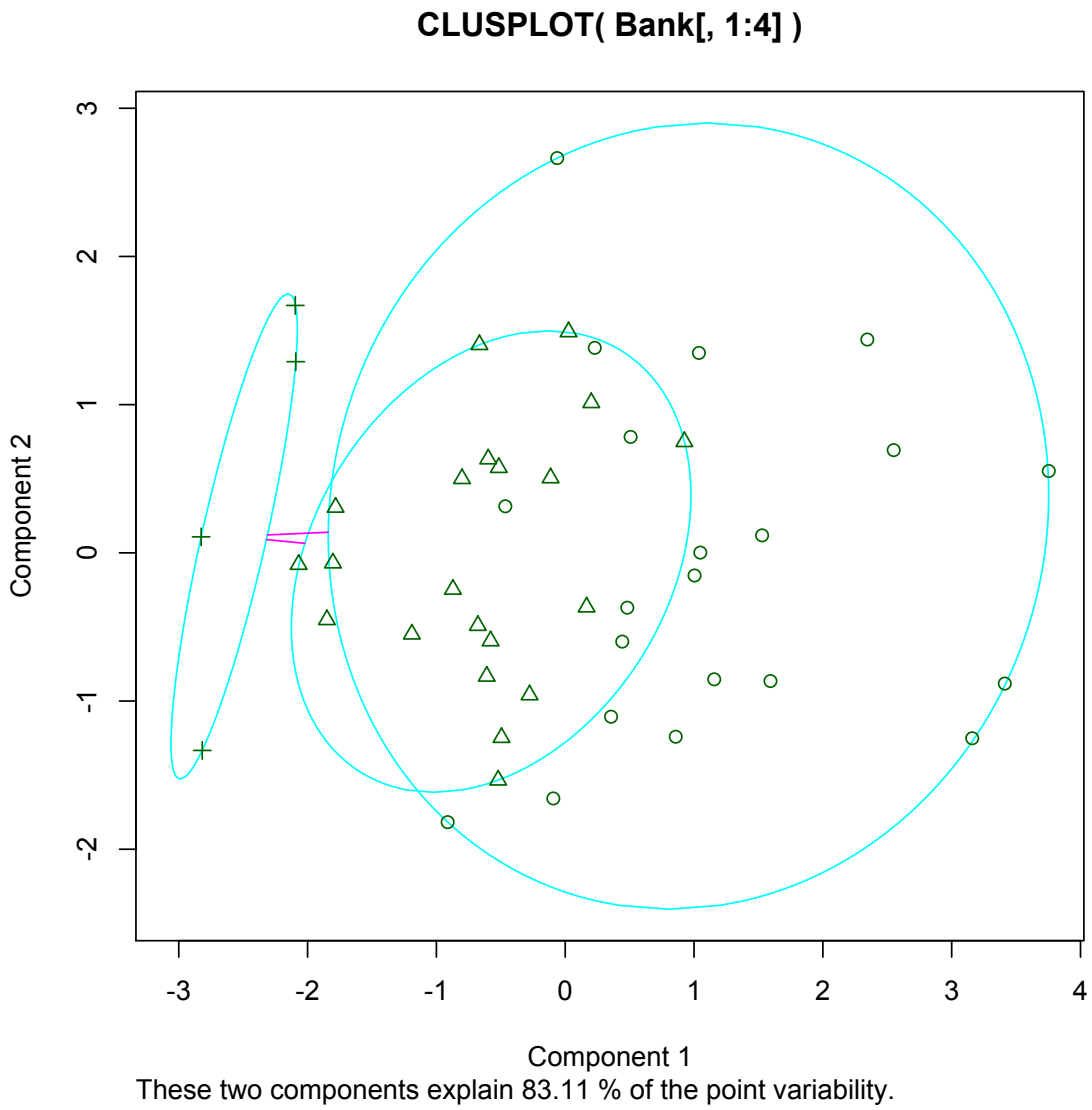

```
> library(MASS)
> bsam = sammon(dist(Bank[,1:4]))
...
> plot(bsam[[1]],pch=16)
```

Trying 2-means



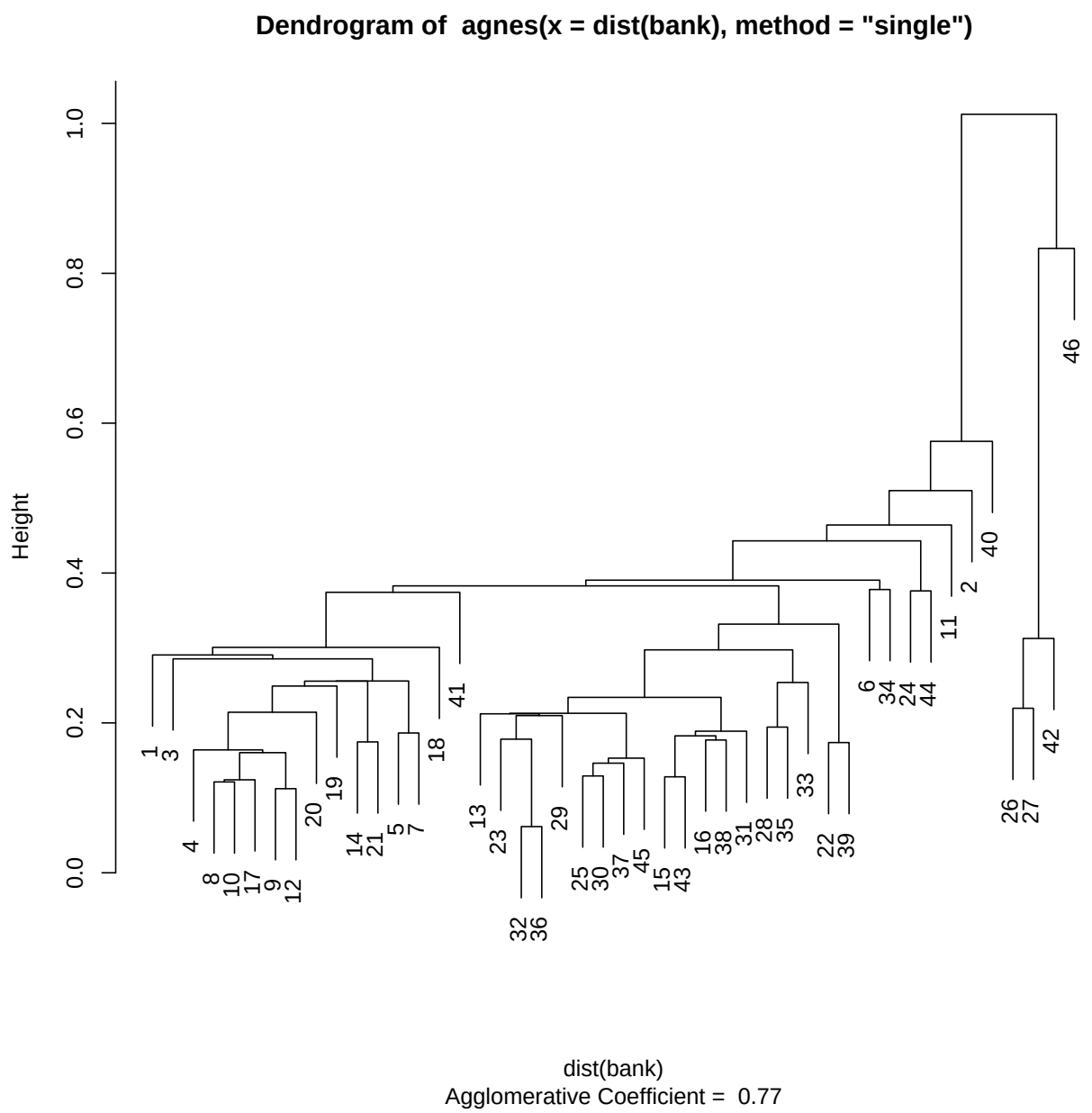

```
> library(cluster)
> clusplot(Bank[,1:4],kmeans(Bank[,1:4],2)$clust)
```

Trying 3-medoids



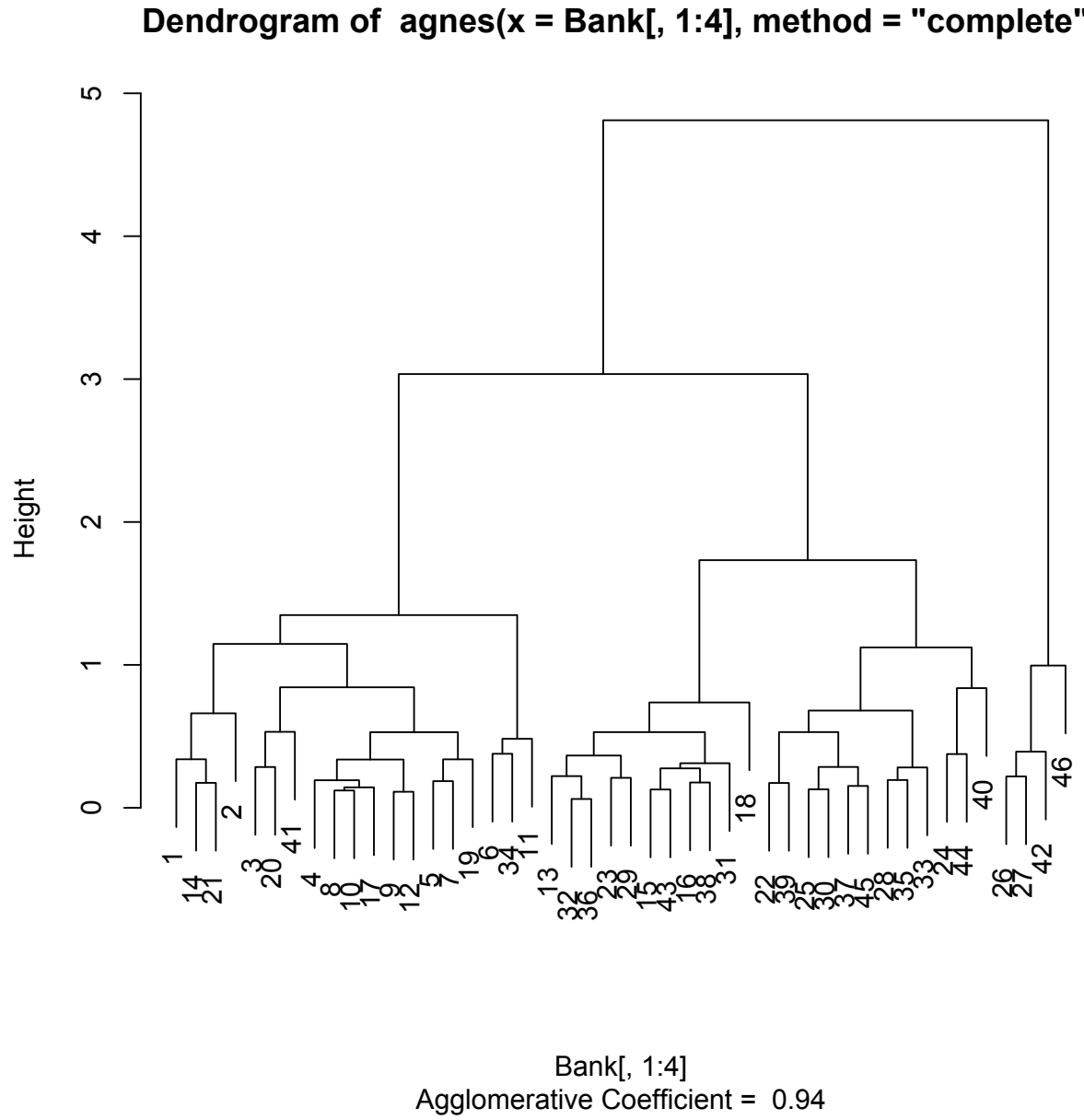

```
> library(cluster)
> clusplot(Bank[,1:4],pam(Bank[,1:4],3)$clust)
> clusplot(pam(Bank[,1:4],3))
```

Hierarchical: agglomerative, single linkage



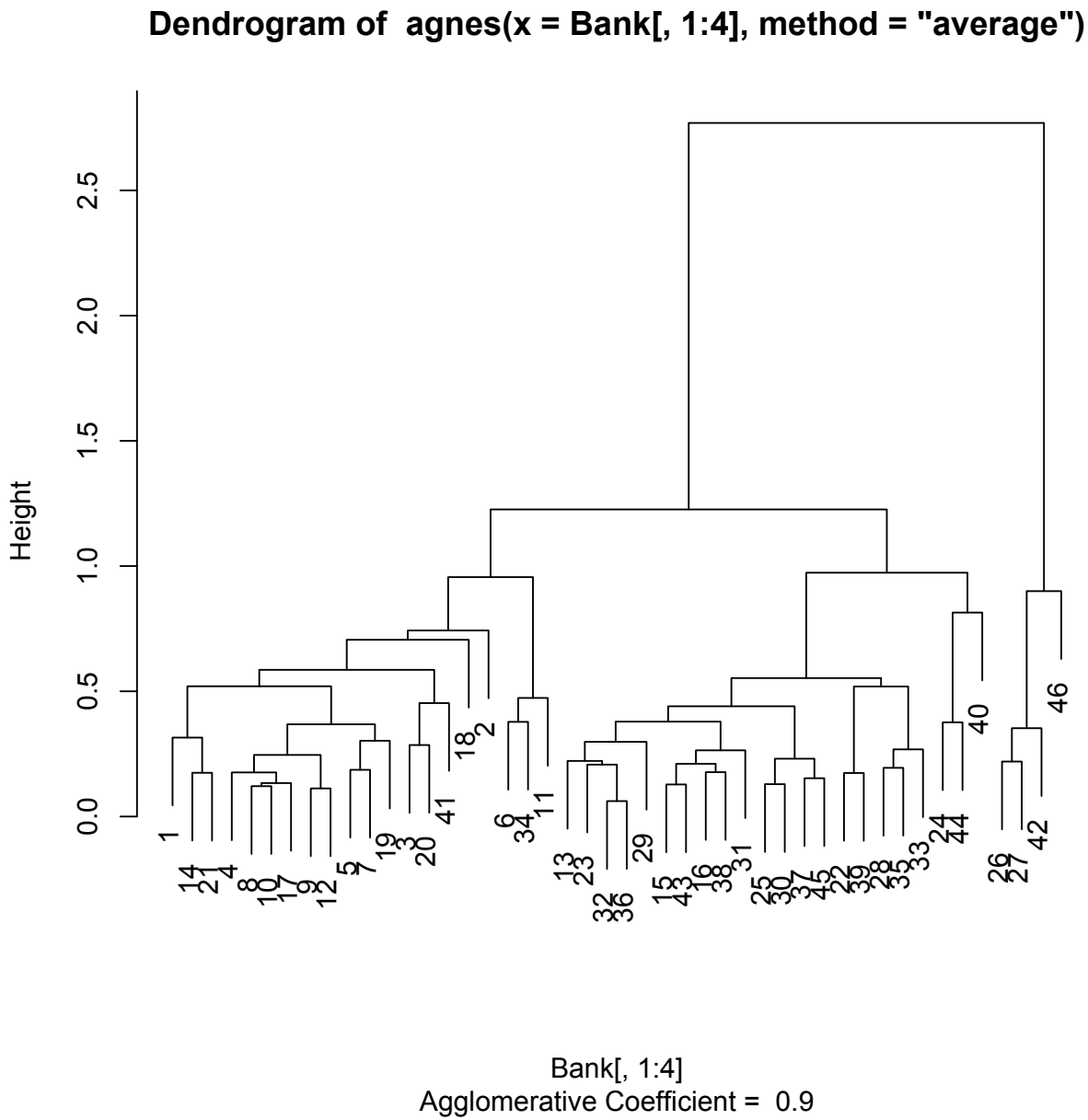

```
> plot(agnes(Bank[,1:4], method = "single"),which=2)
```

Hierarchical: agglomerative, complete linkage



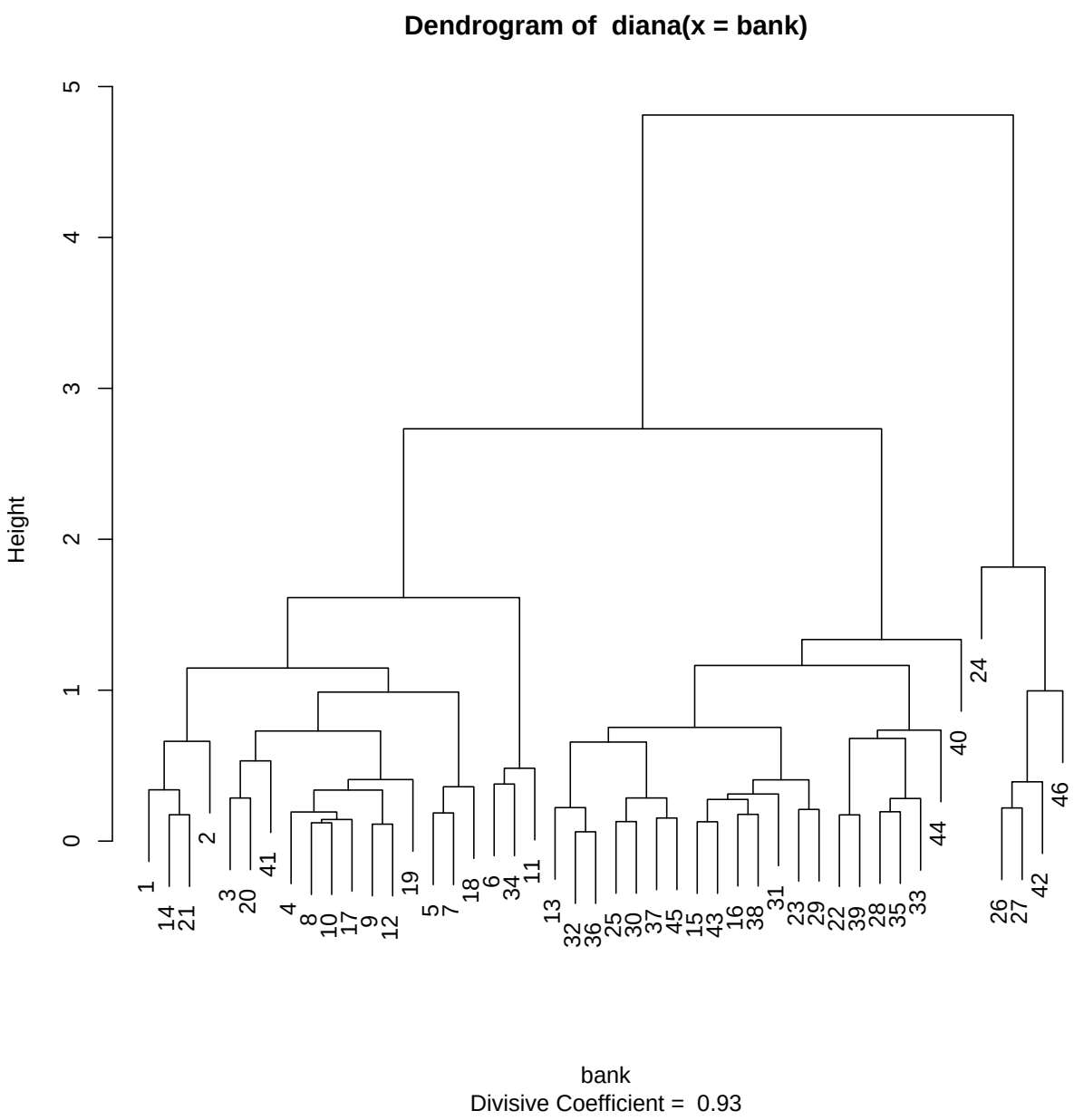

```
> pltree(agnes(Bank[,1:4], method = "complete"))
```

Hierarchical: agglomerative, average linkage




```
> pltree(agnes(dist(Bank[,1:4]), method = "average"))
```

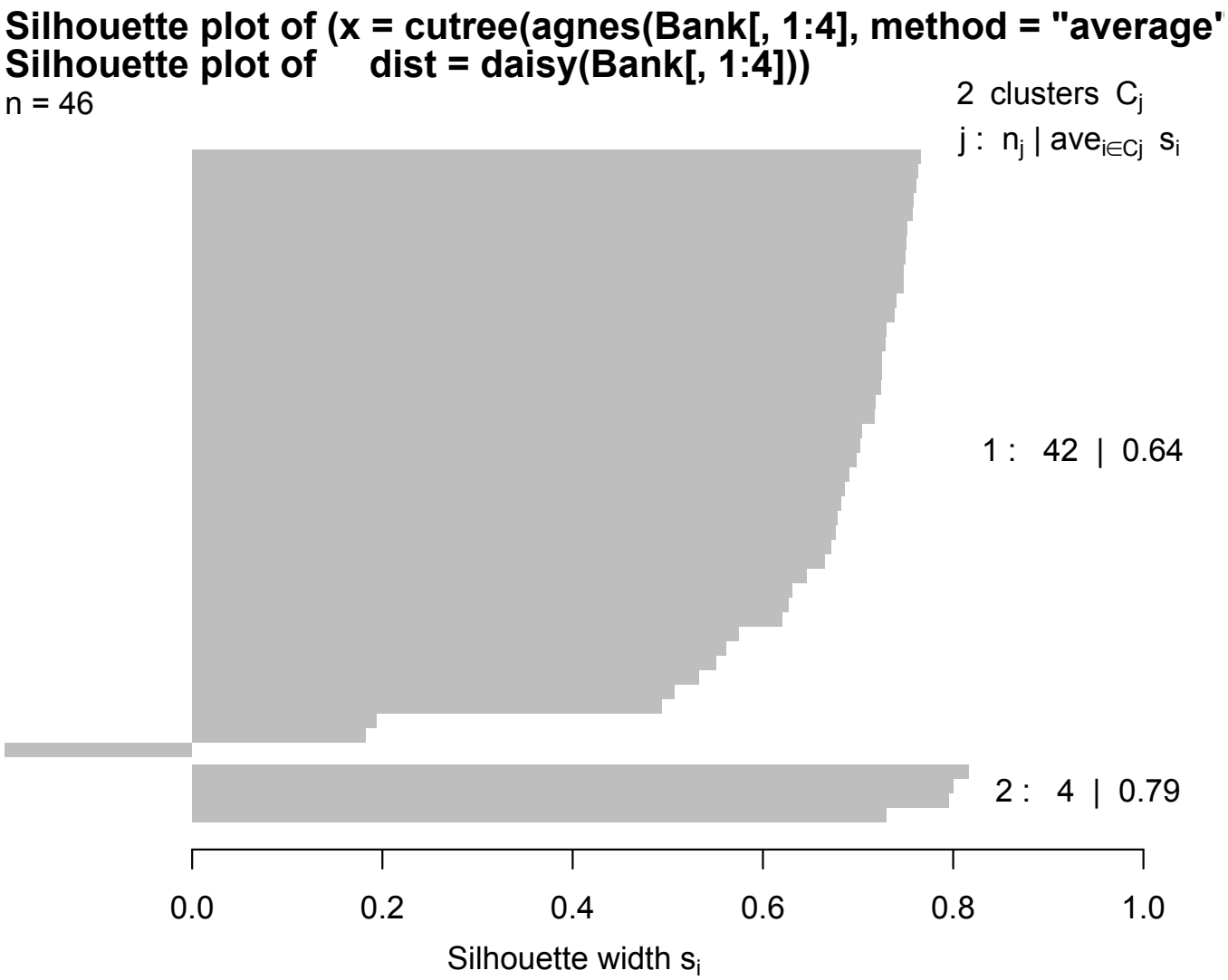
Hierarchical: divisive




```
> pltree(diana(Bank[,1:4]))
```

Silhouette for 2 clusters: agglomerative

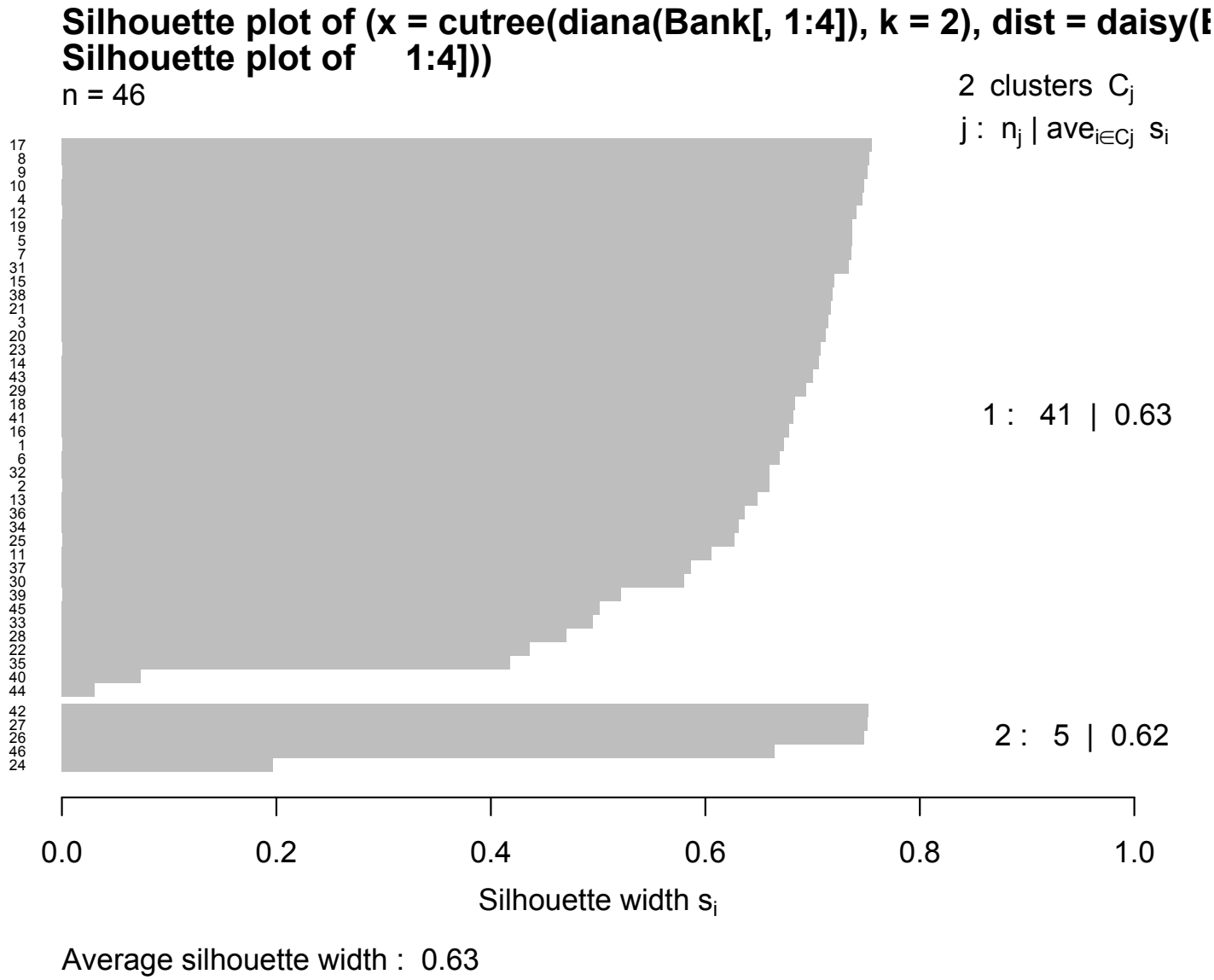
```
> plot(silhouette(cutree(agnes(Bank[,1:4],method="average"),k=2),
+ dist=daisy(Bank[,1:4])))
```



Average silhouette width : 0.65

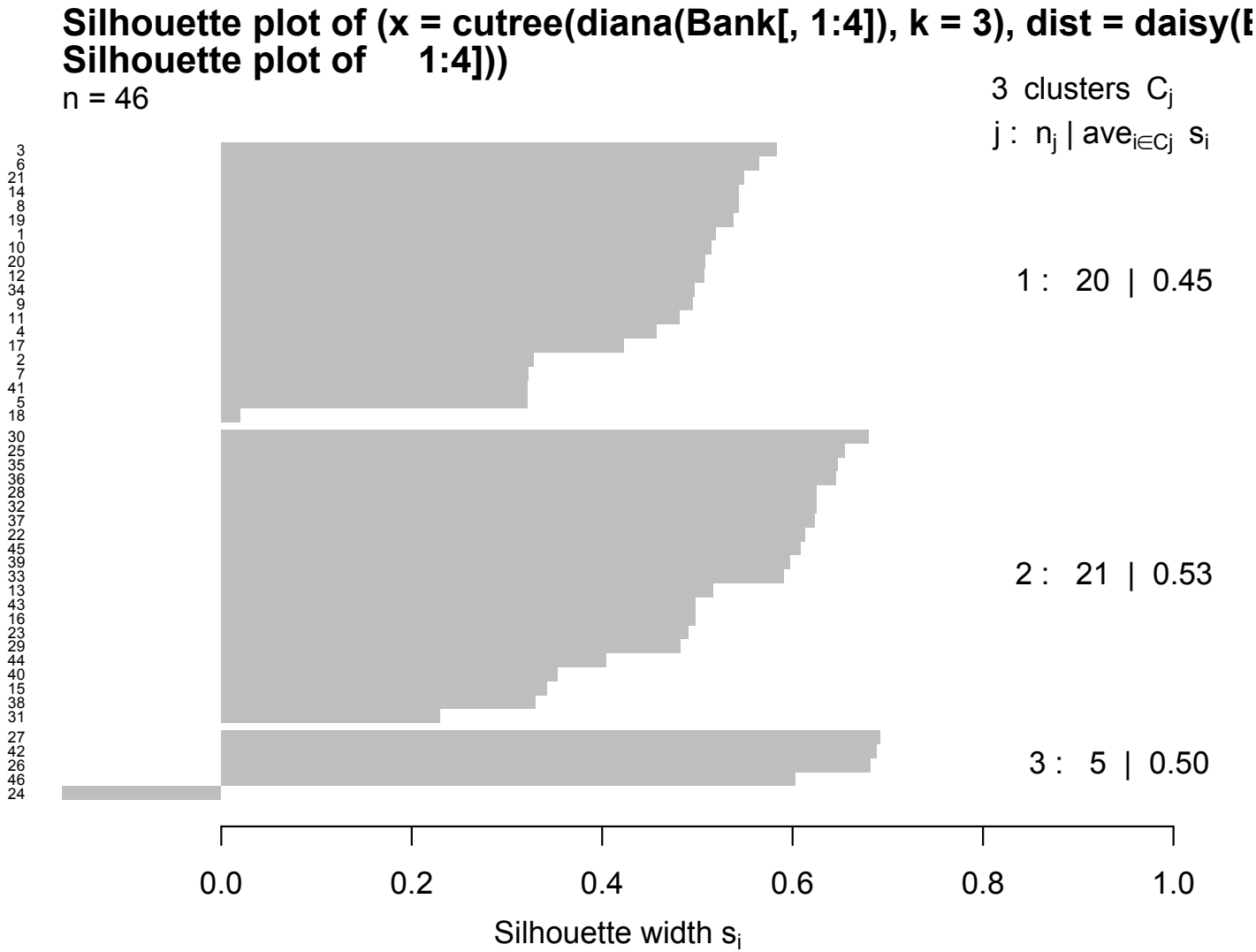
Silhouette for 2 clusters: divisive

```
> plot(silhouette(cutree(diana(Bank[,1:4]),k=2),
+ dist=daisy(Bank[,1:4])),nmax.lab=50,cex=.5)
```



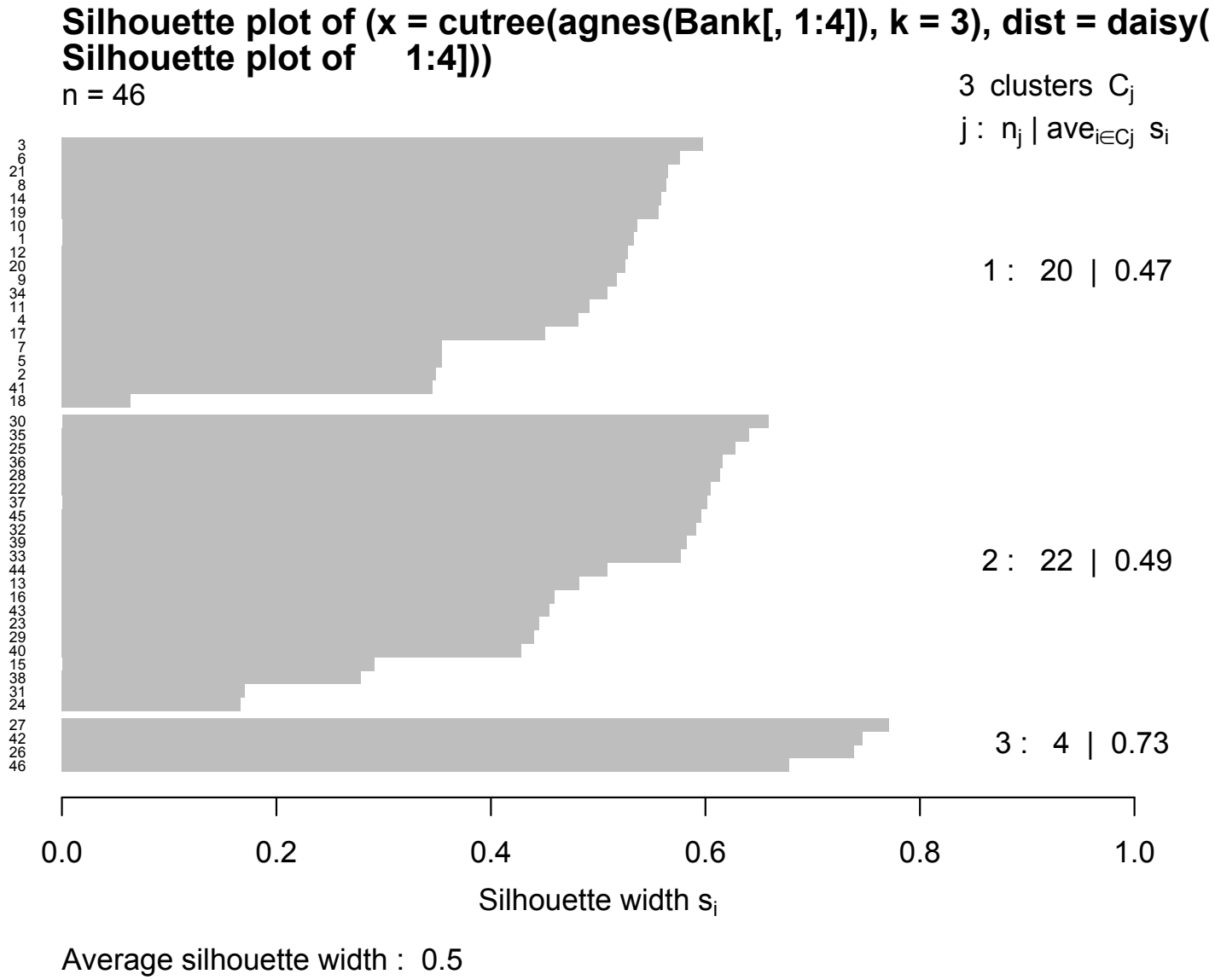
Silhouette for 3 clusters: divisive

```
> plot(silhouette(cutree(diana(Bank[,1:4]),k=3),
+ dist=daisy(Bank[,1:4])),nmax.lab=50,cex=.5)
```



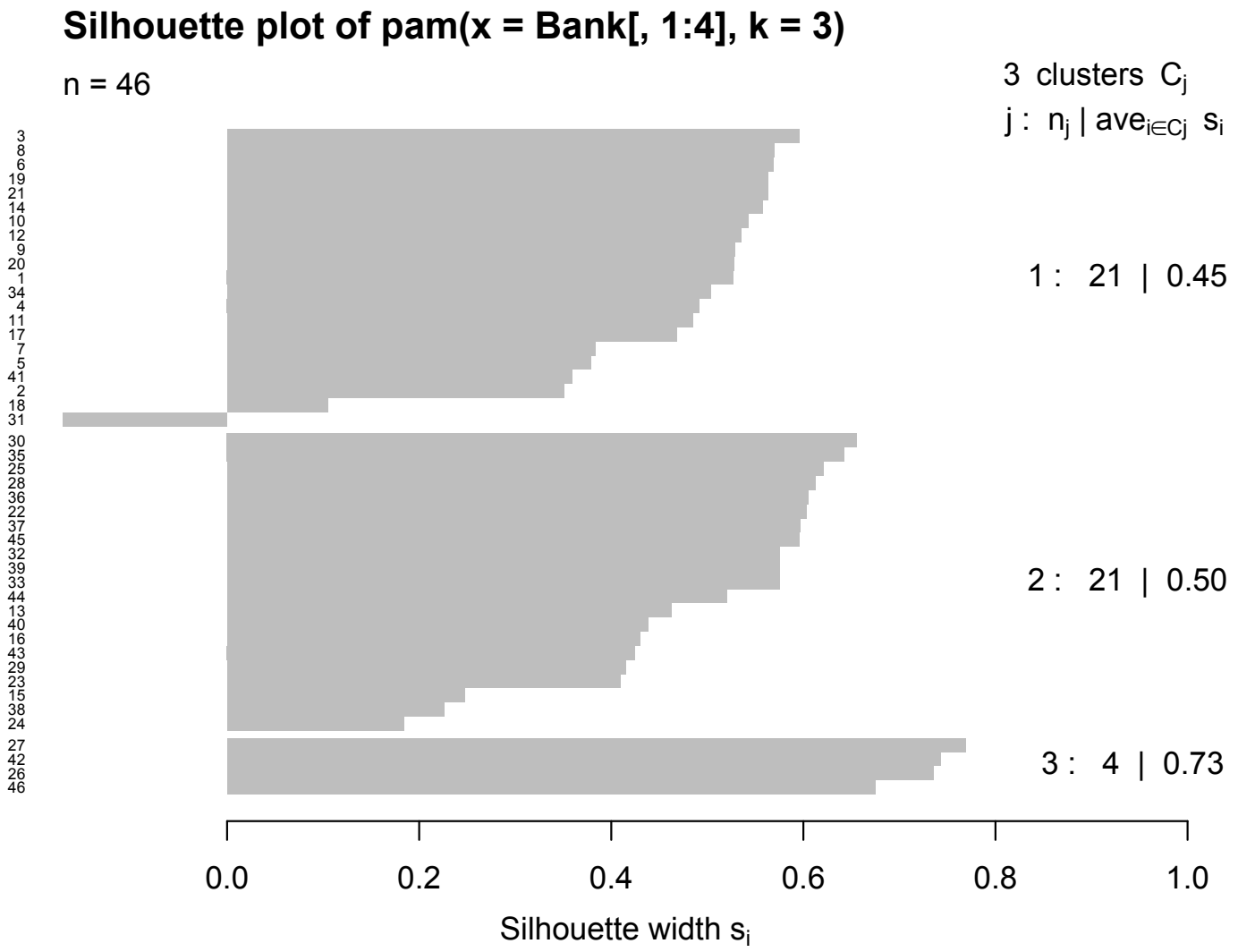
Silhouette for 3 clusters: agglomerative

```
> plot(silhouette(cutree(agnes(Bank[,1:4]),k=2),
+ dist=daisy(Bank[,1:4])),nmax.lab=50,cex=.5)
```



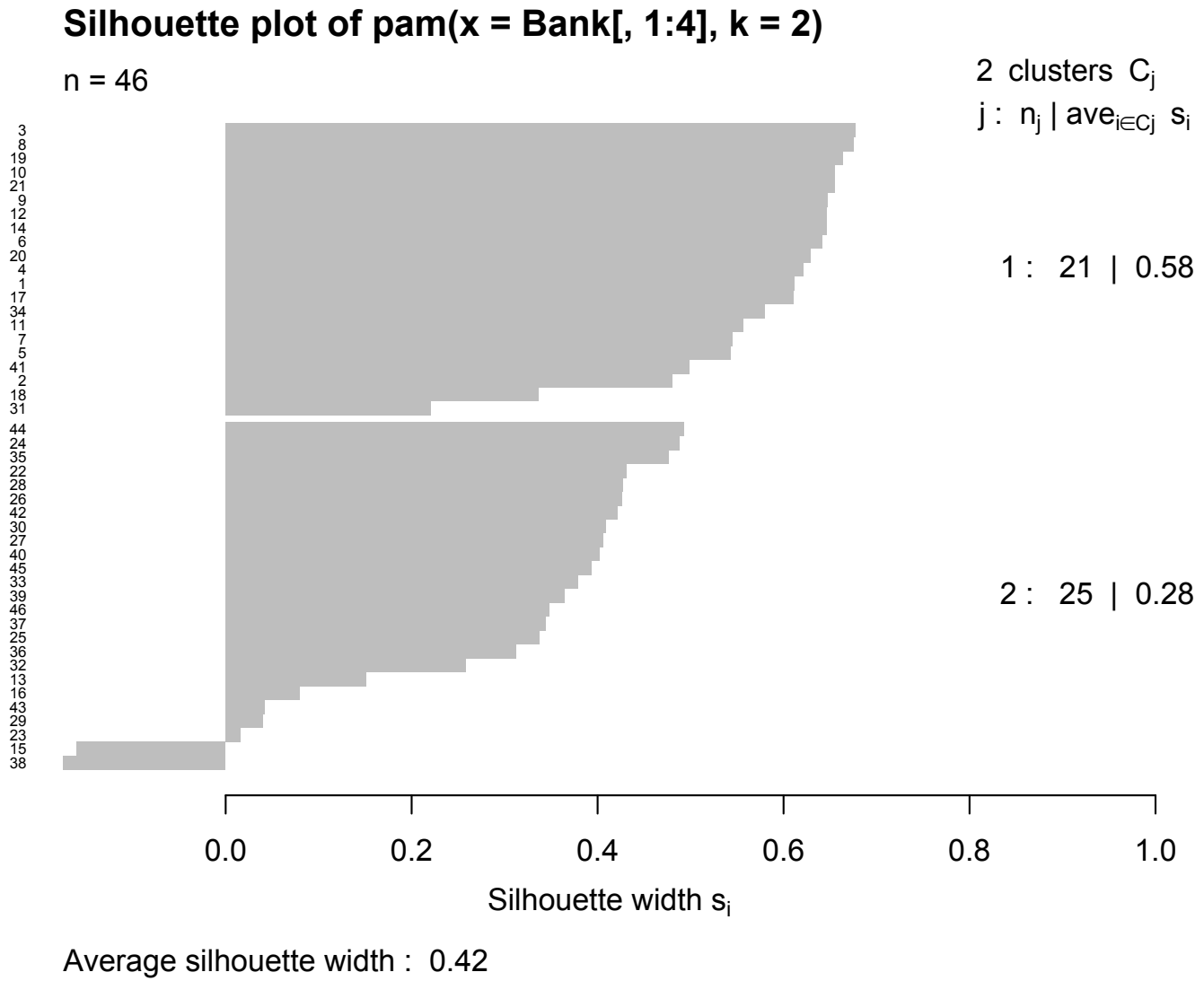
Silhouette for 3-medoids

```
> plot(silhouette(pam(Bank[,1:4],3)),nmax.lab=50,cex=0.5)
```



Silhouette for 2-medoids

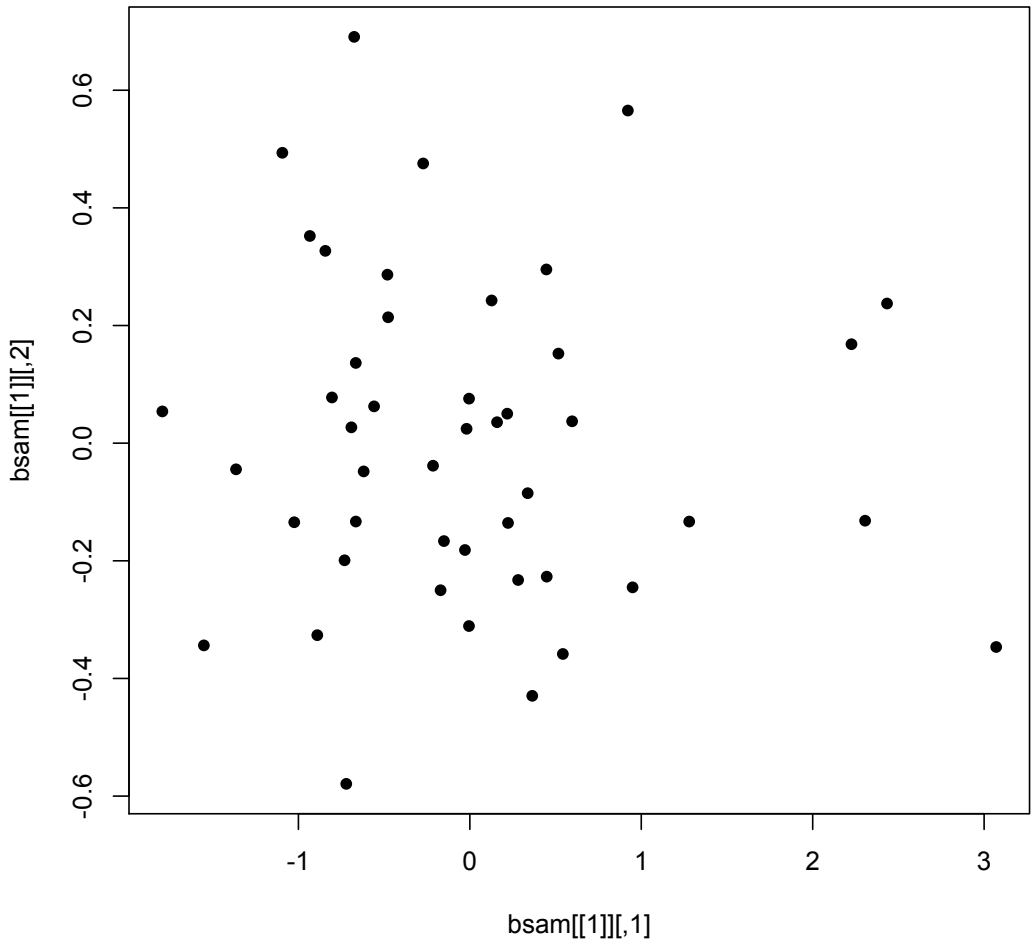
```
> plot(silhouette(pam(Bank[,1:4],2)),nmax.lab=50,cex=0.5)
```



(standardization changes it a bit, but does not really help)

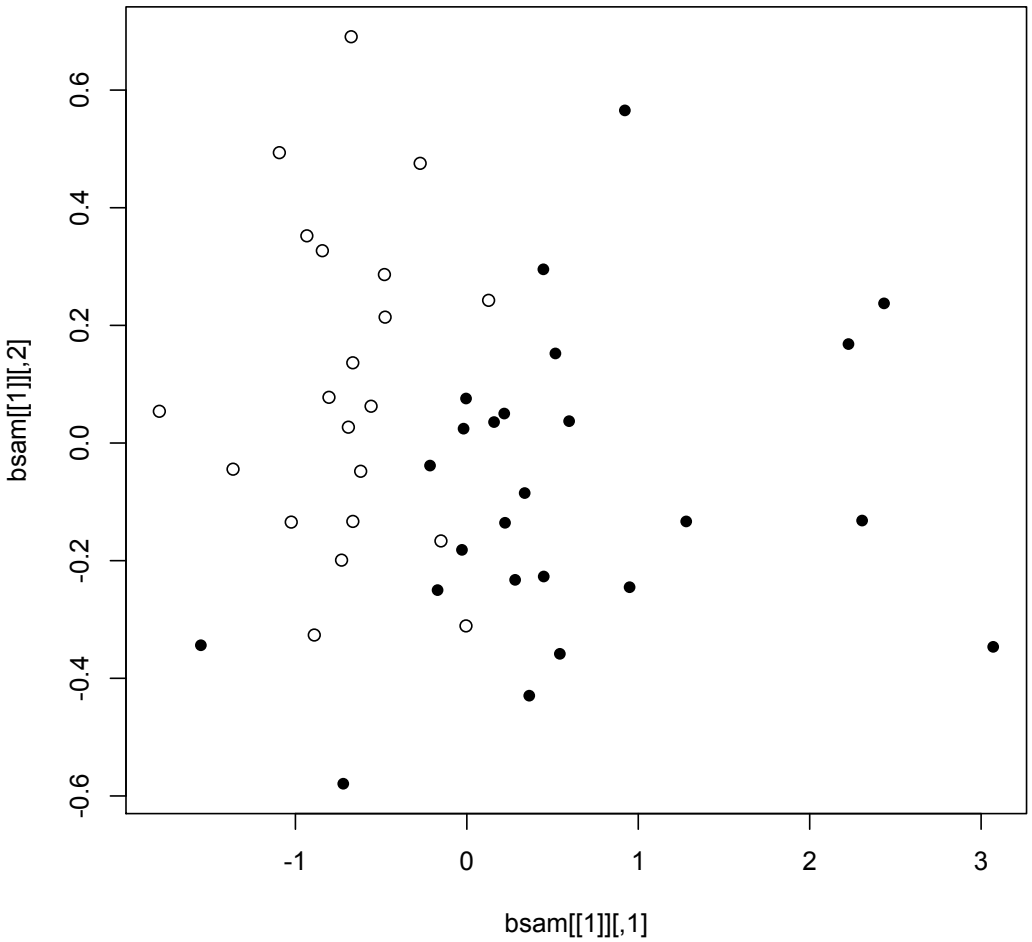
And now: bank data revealed

```
> library(MASS)
> bsam = sammon(dist(Bank0))
> plot(bsam[[1]],pch=16)
```



Now we know who bankrupted and who not

```
> plot(bsam[[1]],pch=15*Bank[,5]+1)
```



The dataset

	v1	v2	v3	v4	k
1	-0.45	-0.41	1.09	0.45	0
2	-0.56	-0.31	1.51	0.16	0
3	0.06	0.02	1.01	0.40	0
4	-0.07	-0.09	1.45	0.26	0
5	-0.10	-0.09	1.56	0.67	0
6	-0.14	-0.07	0.71	0.28	0
7	0.04	0.01	1.50	0.71	0
8	-0.07	-0.06	1.37	0.40	0
9	0.07	-0.01	1.37	0.34	0
10	-0.14	-0.14	1.42	0.43	0
11	-0.23	-0.30	0.33	0.18	0
12	0.07	0.02	1.31	0.25	0
13	0.01	0.00	2.15	0.70	0
14	-0.28	-0.23	1.19	0.66	0
15	0.15	0.05	1.88	0.27	0
16	0.37	0.11	1.99	0.38	0
17	-0.08	-0.08	1.51	0.42	0
18	0.05	0.03	1.68	0.95	0
19	0.01	0.00	1.26	0.60	0
20	0.12	0.11	1.14	0.17	0
21	-0.28	-0.27	1.27	0.51	0
22	0.51	0.10	2.49	0.54	1
23	0.08	0.02	2.01	0.53	1
24	0.38	0.11	3.27	0.35	1
25	0.19	0.05	2.25	0.33	1
26	0.32	0.07	4.24	0.63	1
27	0.31	0.05	4.45	0.69	1
28	0.12	0.05	2.52	0.69	1
29	-0.02	0.02	2.05	0.35	1
30	0.22	0.08	2.35	0.40	1
31	0.17	0.07	1.80	0.52	1
32	0.15	0.05	2.17	0.55	1
33	-0.10	-0.01	2.50	0.58	1
34	0.14	-0.03	0.46	0.26	1
35	0.14	0.07	2.61	0.52	1
36	0.15	0.06	2.23	0.56	1
37	0.16	0.05	2.31	0.20	1
38	0.29	0.06	1.84	0.38	1
39	0.54	0.11	2.33	0.48	1
40	-0.33	-0.09	3.01	0.47	1
41	0.48	0.09	1.24	0.18	1
42	0.56	0.11	4.29	0.44	1
43	0.20	0.08	1.99	0.30	1
44	0.47	0.14	2.92	0.45	1
45	0.17	0.04	2.45	0.14	1
46	0.58	0.04	5.06	0.13	1

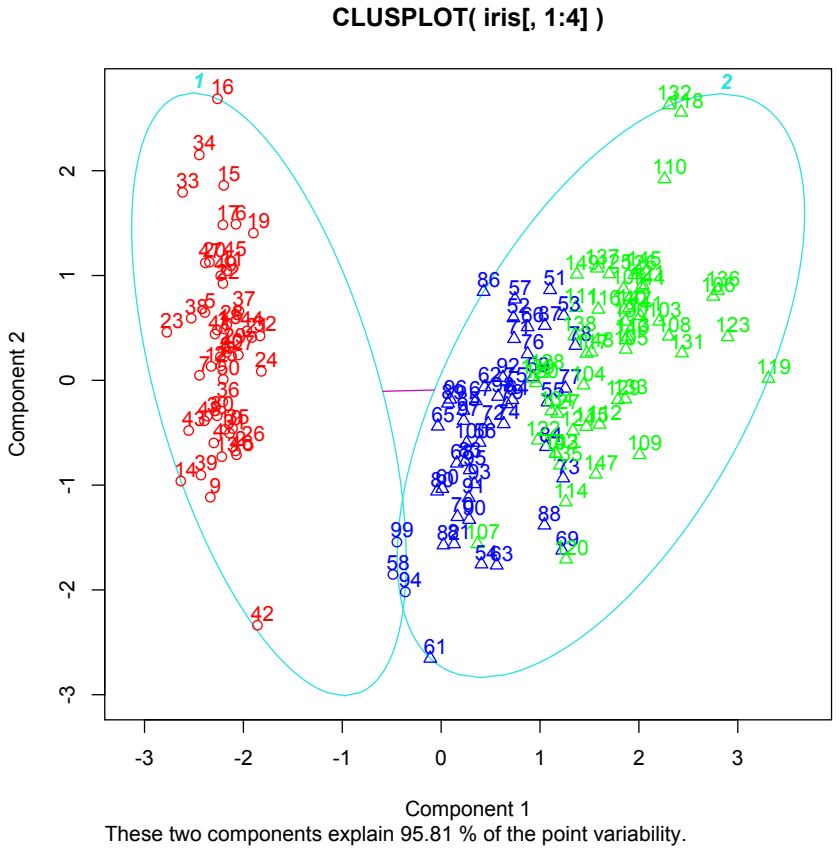
A famous dataset: Edgar Anderson's Iris Data

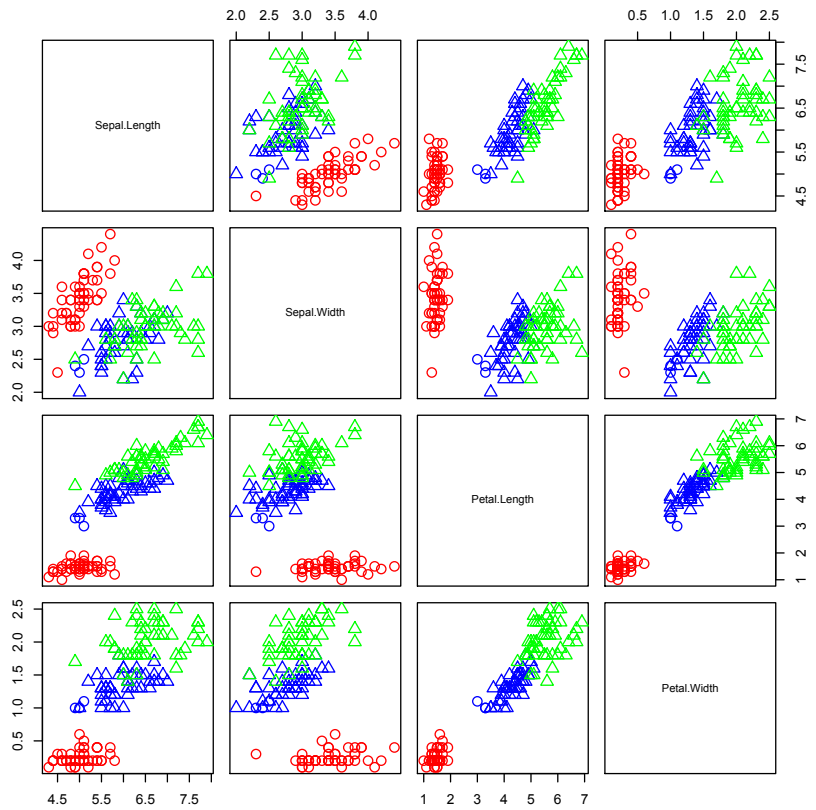
```
> ?iris
```

This famous (Fisher's or Anderson's) Iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of Iris. The species are Iris setosa, versicolor, and virginica.

Again, it is a *supervised* classification dataset: we know the categories (labels). However, we play the game again: we pretend that we do not know them, and see where clustering methods will take us

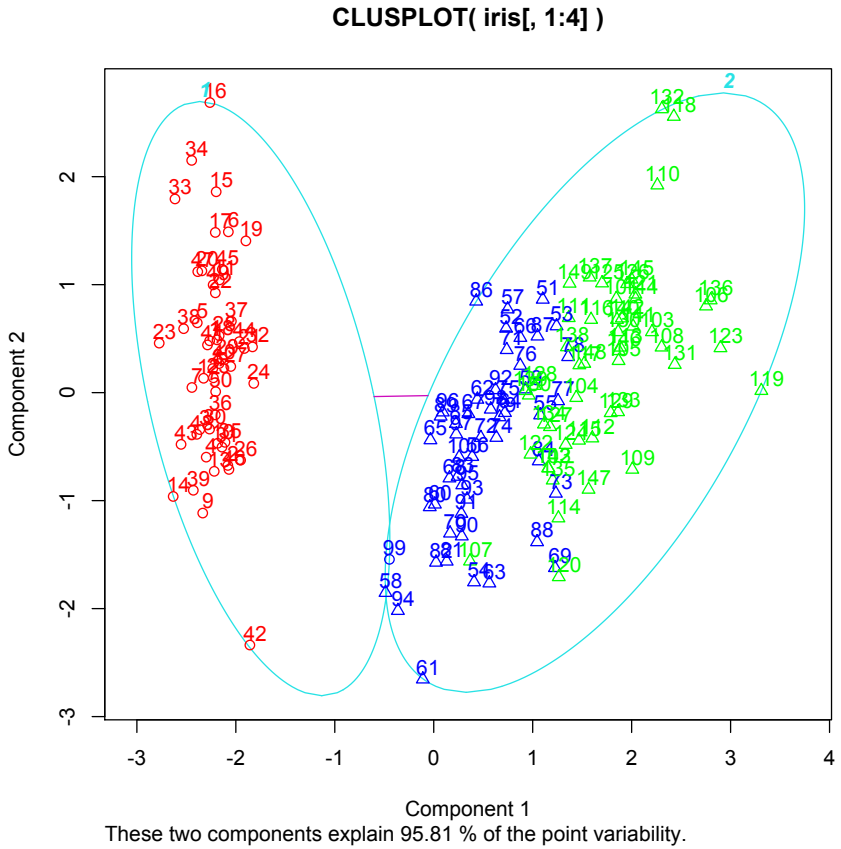
Iris: 2-means

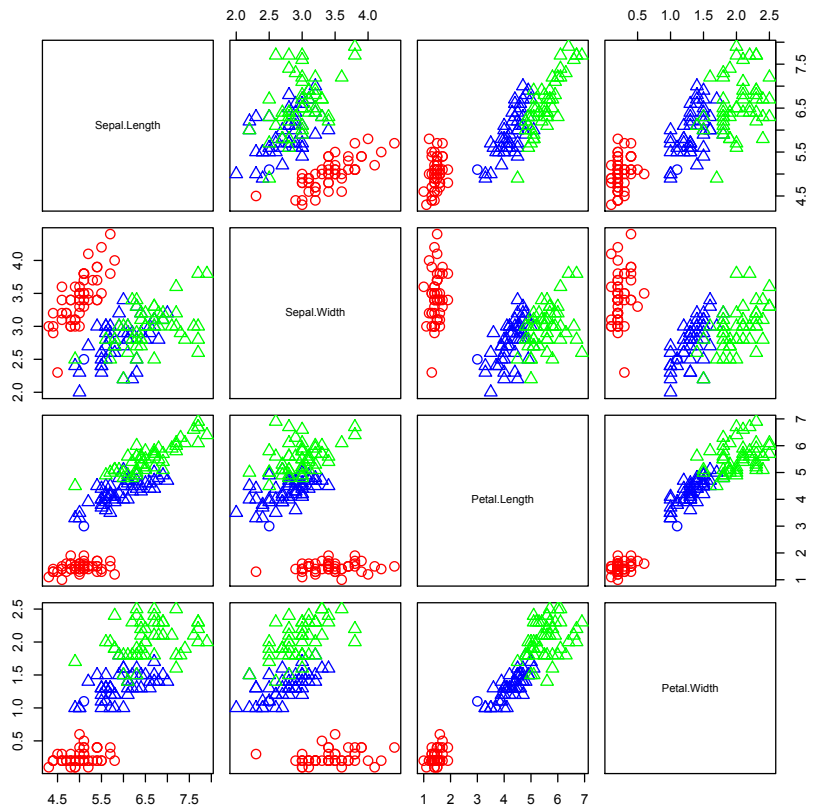





```
> clusplot(iris[,1:4],kmeans(iris[,1:4],2)$cluster,
+ diss=F,labels=2,col.p=c("red","blue","brown")[iris[,5]])
> pairs(iris[,1:4],pch=kmeans(iris[,1:4],2)$cluster,
+ col=c("red","blue","green")[iris[,5]],cex=1.5)
```

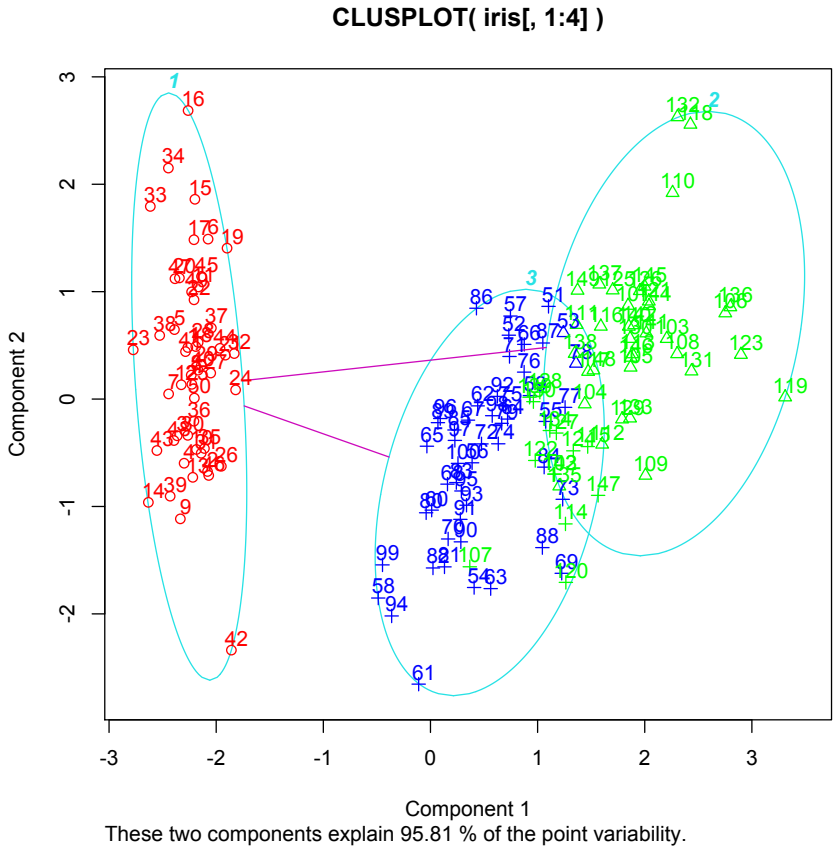
Iris: 2-medoids

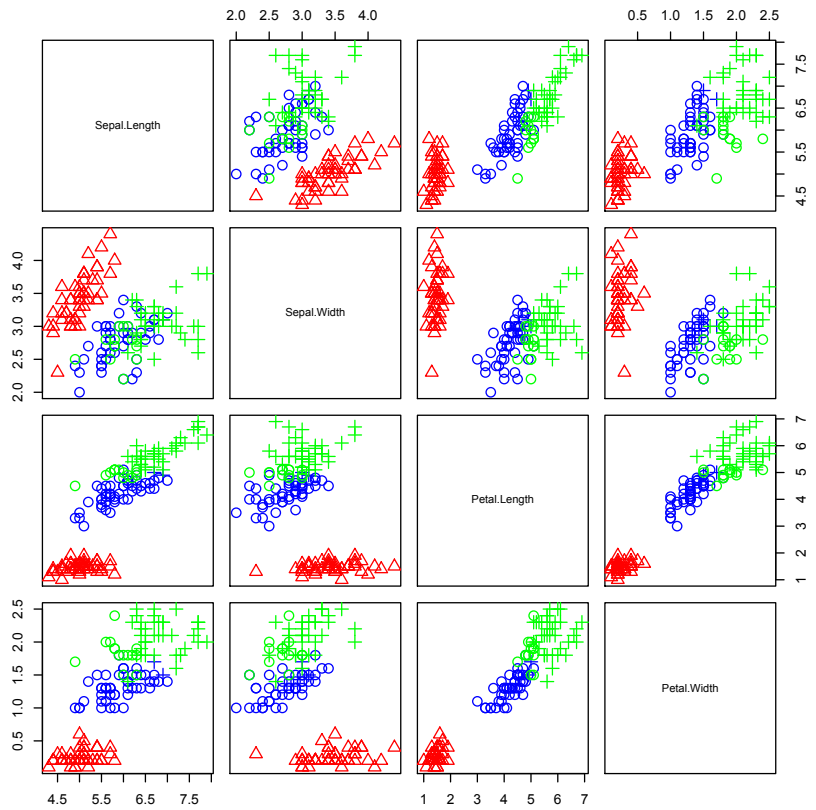





```
> clusplot(iris[,1:4],pam(iris[,1:4],2,diss=F)$cluster,
+ labels=2,col.p=c("red","blue","green")[iris[,5]])
> pairs(iris[,1:4],pch=pam(iris[,1:4],2,diss=F)$cluster,
+ col=c("red","blue","green")[iris[,5]],cex=1.5)
```

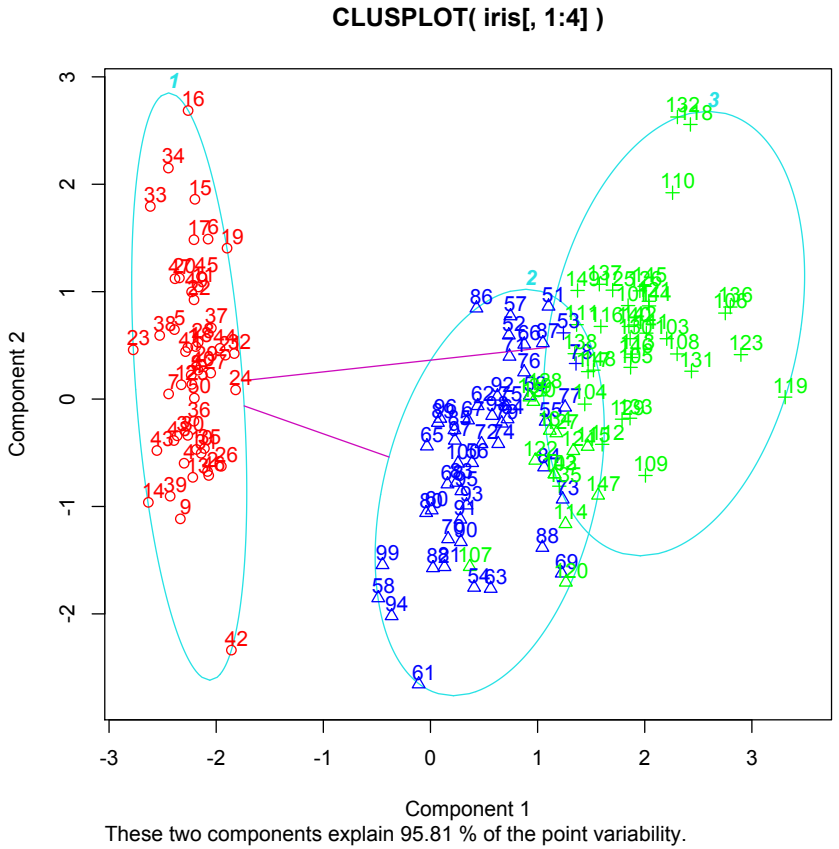
Iris: 3-means

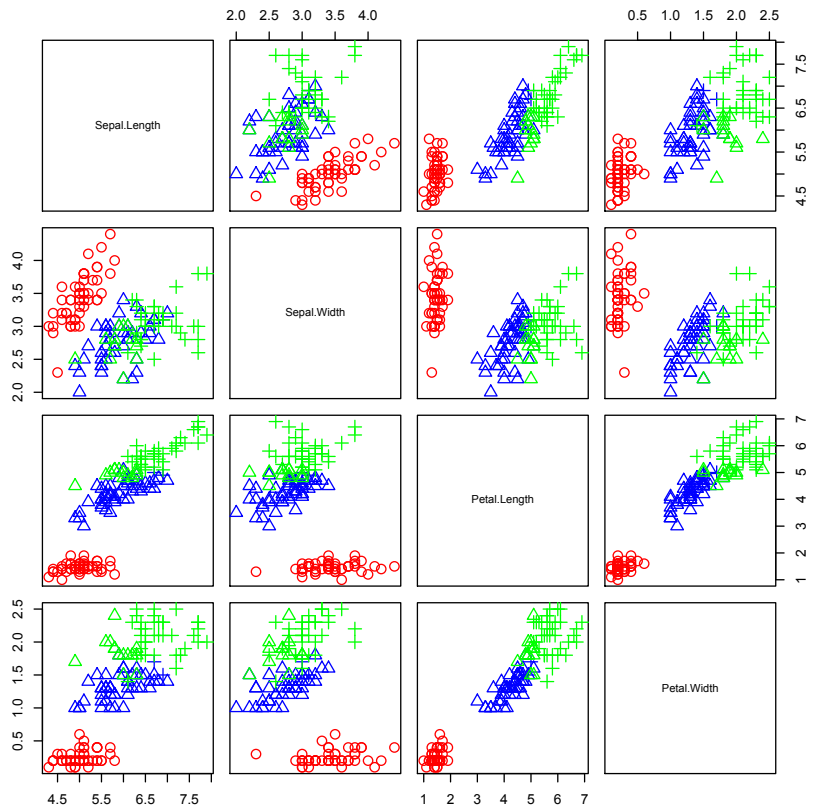





```
> clusplot(iris[,1:4],kmeans(iris[,1:4],3)$cluster,diss=F,
+ labels=2,col.p=c("red","blue","green")[iris[,5]])
> pairs(iris[,1:4],pch=kmeans(iris[,1:4],3)$cluster,
+ col=c("red","blue","green")[iris[,5]],cex=1.5)
```

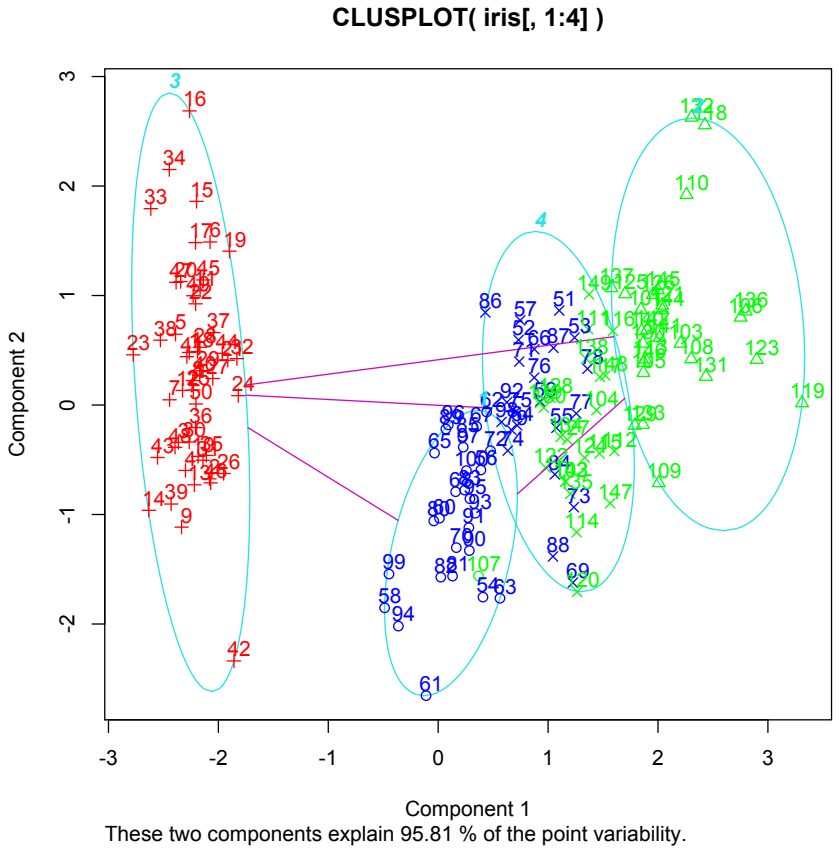
Iris: 3-medoids

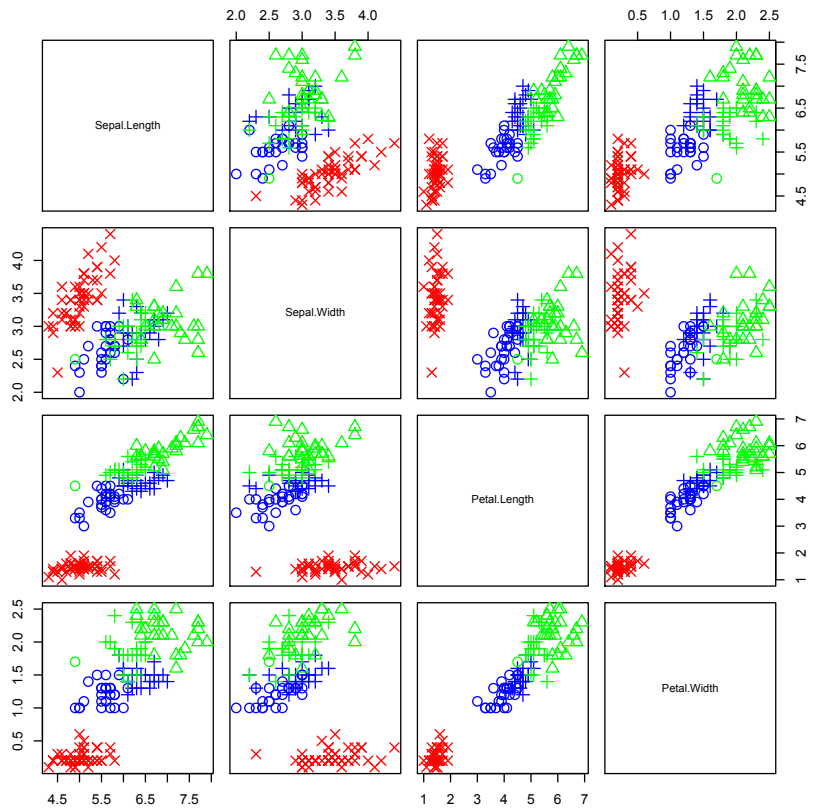





```
> clusplot(iris[,1:4],pam(iris[,1:4],3,diss=F)$cluster,
+ labels=2,col.p=c("red","blue","green")[iris[,5]])
> pairs(iris[,1:4],pch=pam(iris[,1:4],3,diss=F)$cluster,
+ col=c("red","blue","green")[iris[,5]],cex=1.5)
```

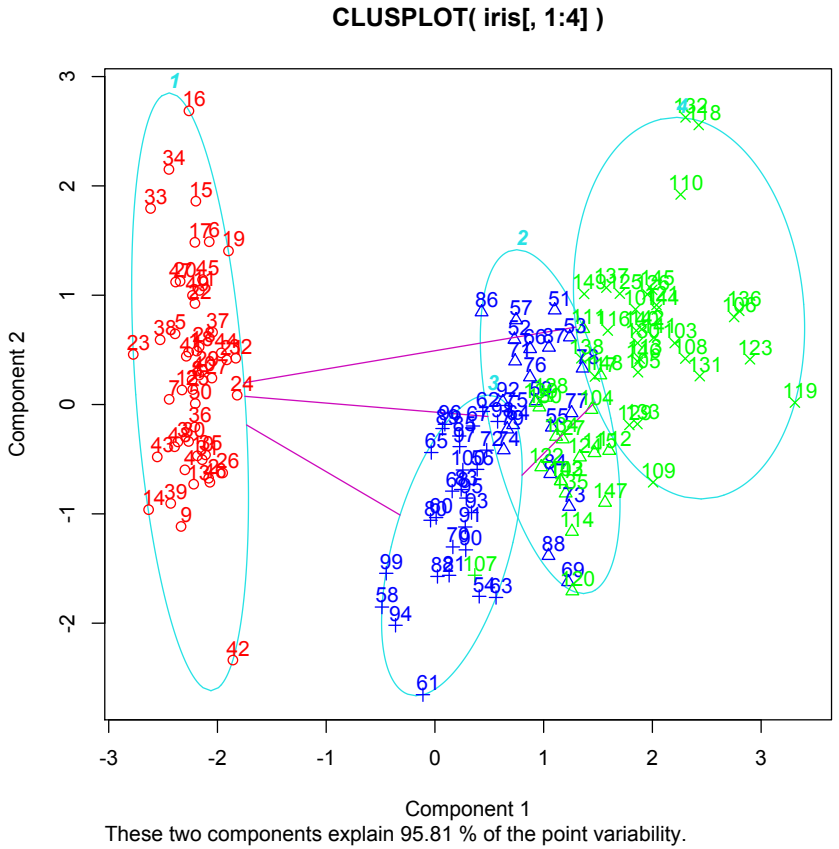
Iris: 4-means

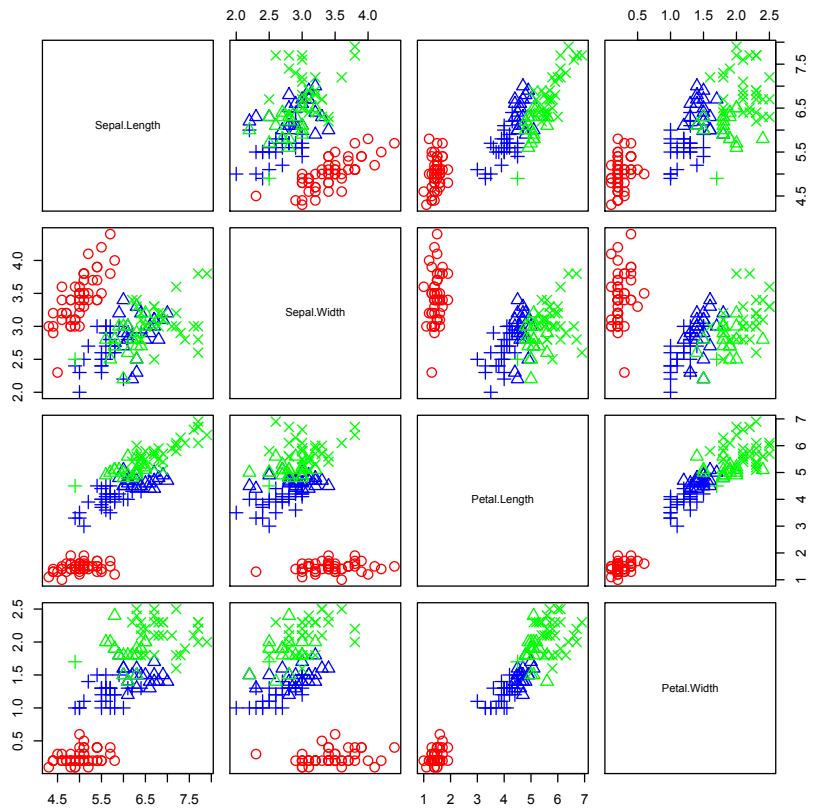





```
> clusplot(iris[,1:4],kmeans(iris[,1:4],4)$cluster,diss=F,
+ labels=2,col.p=c("red","blue","green")[iris[,5]])
> pairs(iris[,1:4],pch=kmeans(iris[,1:4],4)$cluster,
+ col=c("red","blue","green")[iris[,5]],cex=1.5)
```

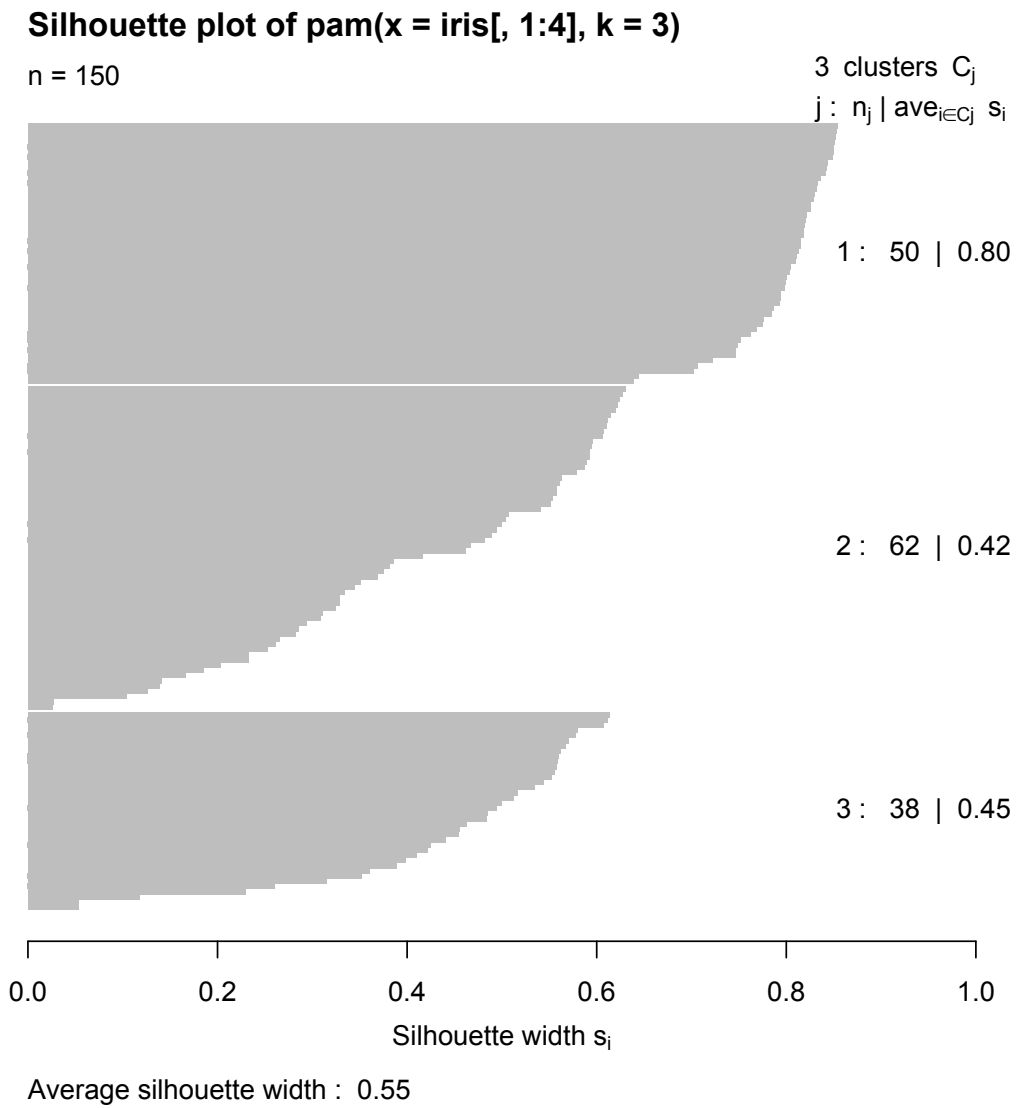
Iris: 4-medoids





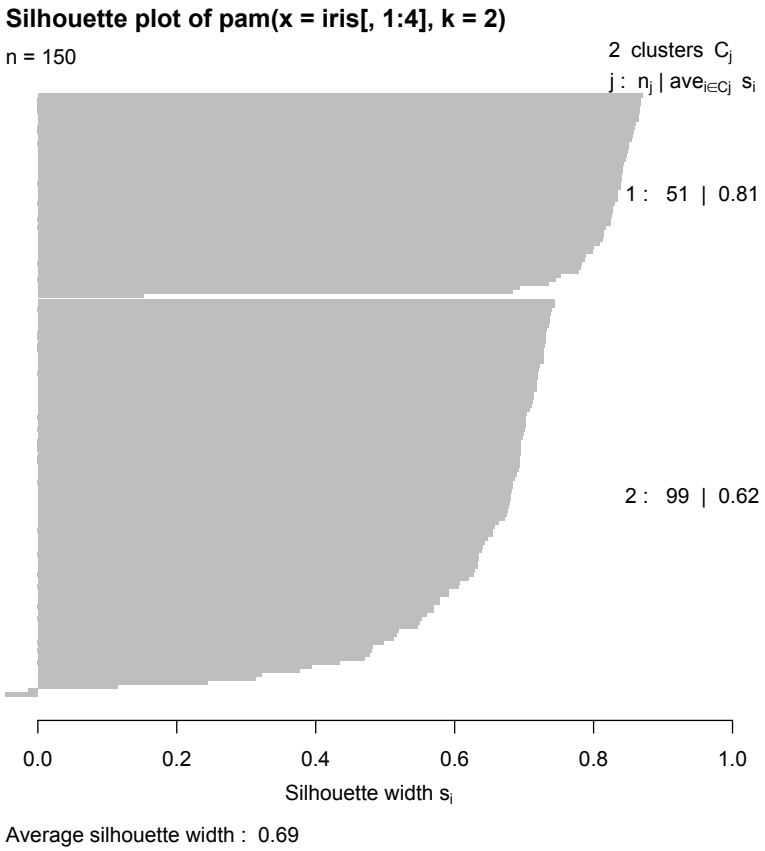

```
> clusplot(iris[,1:4],pam(iris[,1:4],4,diss=F)$cluster,
+ labels=2,col.p=c("red","blue","green")[iris[,5]])
> pairs(iris[,1:4],pch=pam(iris[,1:4],4,diss=F)$cluster,
+ col=c("red","blue","green")[iris[,5]],cex=1.5)
```

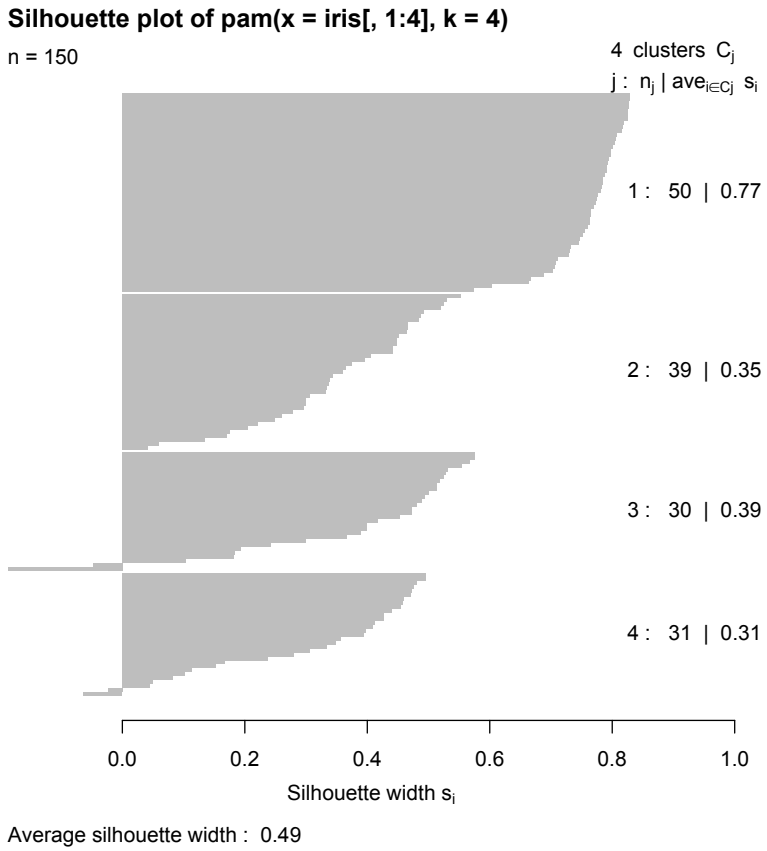
Silhouette plot




```
> plot(silhouette(pam(iris[,1:4],3)))
```

Other silhouettes






```
> plot(silhouette(pam(iris[,1:4],2)))
> plot(silhouette(pam(iris[,1:4],4)))
```

Some comparisons

```
> table(kmeans(iris[,1:4],2)$cluster,pam(iris[,1:4],2)$cluster)
    1  2
1  51  2
2   0 97
> table(kmeans(iris[,1:4],3)$cluster,pam(iris[,1:4],3)$cluster)
    1  2  3
1  50  0  0
2   0 62  0
3   0  0 38
> table(kmeans(iris[,1:4],4)$cluster,pam(iris[,1:4],4)$cluster)
    1  2  3  4
1   0  0  0 27
2  50  0  0  0
3   0 39  2  4
4   0  0 28  0
```

Note however

```
> table(kmeans(iris[,1:4],4)$cluster,pam(iris[,1:4],4)$cluster)
    1  2  3  4
1   0  7  0 31
2   0 32 30  0
3  28  0  0  0
4  22  0  0  0
> table(kmeans(iris[,1:4],4)$cluster,pam(iris[,1:4],4)$cluster)
    1  2  3  4
1  28  0  0  0
2  22  0  0  0
3   0  7  0 31
4   0 32 30  0
> table(kmeans(iris[,1:4],4)$cluster,pam(iris[,1:4],4)$cluster)
    1  2  3  4
1   0  0 28  0
2   0  0  0 27
3   0 39  2  4
4  50  0  0  0
```

And the secret unveiled

```
> table(kmeans(iris[,1:4],3)$cluster,iris[,5])
    setosa versicolor virginica
  1     50          0         0
  2      0         48        14
  3      0          2        36
> table(pam(iris[,1:4],3)$cluster,iris[,5])
    setosa versicolor virginica
  1     50          0         0
  2      0         48        14
  3      0          2        36
> table(pam(iris[,1:4],3,diss=FALSE)$cluster,iris[,5])
    setosa versicolor virginica
  1     50          0         0
  2      0         48        14
  3      0          2        36
```

(The latter is more stable, however - if you try to run each command several times, you can see it)

More of the toy data...

	income	lot	riding
1	60.0	18.4	1
2	85.5	16.8	1
3	64.8	21.6	1
4	61.5	20.8	1
5	87.0	23.6	1
6	110.1	19.2	1
7	108.0	17.6	1
8	82.8	22.4	1
9	69.0	20.0	1
10	93.0	20.8	1
11	51.0	22.0	1
12	81.0	20.0	1

	income	lot	riding
13	75.0	19.6	0
14	52.8	20.8	0
15	64.8	17.2	0
16	43.2	20.4	0
17	84.0	17.6	0
18	49.2	17.6	0
19	59.4	16.0	0
20	66.0	18.4	0
21	47.4	16.4	0
22	33.0	18.8	0
23	51.0	14.0	0
24	63.0	14.8	0

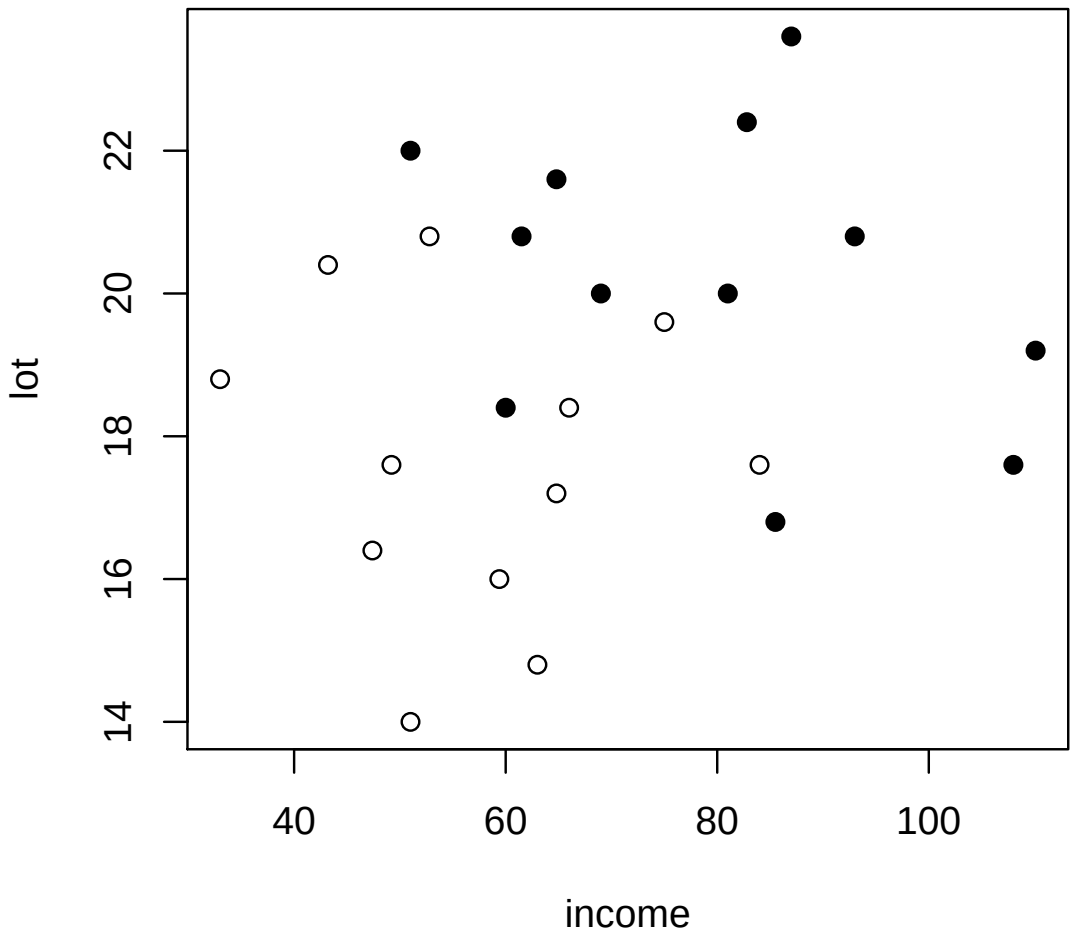
Whether a customer bought a riding-mower or not.

Depending on the income and lot size.

Important question: out of the potential *new* customers, predict (on the basis of their income and lot size) who is likely to buy a riding mower.

...nice, because can be plotted

```
> attach(Mowers)
> plot(income,lot,pch=15*riding+1)
```



Supervised classification: principles

Classification: the game field

$g = 1, \ldots, K$: these are K **classes** (*labels*) relevant to the classification task; it is assumed that each item belongs to one and only one of these K classes. The simplest and most frequented case is $K = 2$, but larger K are not excluded.

$G = 1, \ldots, \mathcal{K}$: these are $\mathcal{K}$ possible *decisions*, classes into which each item is to be classified;

$c(g|G)$: the nonnegative loss function, the **cost** of classifying an item from g-th class to G-th class

The equality of the cost $c(g|G)$ to 0 expresses that the classification from g-th class to G-th class is considered correct. In typical situations, $\mathcal{K} = K$ and the G's are in a simple one-one correspondence with the g's - that is, $c(g|G) = 0$ for $G = g$. (Decisions also of the type "undecided" and similar are somewtimes considered, in which case $\mathcal{K}$ may be different from K.) Any outcome with $c(g|G) > 0$ is referred to as a **misclassification** and $c(g|G)$ in such a case is called a **misclassification cost**. In the absence of exact knowledge of missclasification costs, it is customary to set $c(g|G) = 1$ for all g and G with $c(g|G) > 0$ - the situation referred to as *equal misclassification costs*

Classification: the game props

To evaluate the quality of different classification rules, we opt for the framework in which we specify

π_g: the **prior probability** of the g-th class (the probability that a randomly drawn item from the population is in the g-th class).

A possible absence of knowledge about π_g is often resolved by setting $\pi_g = 1/G$, the situation referred to as **equal prior probabilities**; in practical applications however, one has to be aware of the limitations of such a choice

Once $c(g|G)$ and π_g and set, then we may consider possible **classification rules**: such a rule is for each item based on the value x of **classifiers** (predictors, *features*), the variables used for classification, for this particular item.

Note: while we speak about the "value" of **x** in singular, it is typically a vector of values from some $\mathcal{X}$. Classifiers may be of different nature: some of them are numeric, other may be categorial, etc. Their nature thus determines the nature of $\mathcal{X}$

Classification: the rules

A classification rule is itself determined by

$\mathcal{R}_{\mathrm{G}}$: **classification regions**, which for $\mathrm{G} = 1, 2, \ldots, \mathcal{K}$ express how classification rule is set; for any item, if the observed value $\mathbf{x} \in \mathcal{X}$ of the classifiers falls into R_{G}, then the item is classified into the G-th class. Thus, $\mathcal{X}$ is a union of $\mathcal{R}_{\mathrm{G}}$, pairwise disjoint (or almost disjoint, in which case typically they intersect only at their common boundary; if $\mathbf{x}$ falls into this intersection, the classification is resolved somehow, possibly by a random draw)

Having various classification rules, we seek one that is in some sense optimal; the criterion for that can be expressed through

$\mathrm{P}(\mathrm{G}|\mathrm{g})$: conditional probabilities that an item from the g-th class is classified as being from the G-th class

Classification: the criterion

Once all the above are set, we can evaluate the risk, the **total expected misclassification cost**;

$$\sum_{g} \sum_{G} \pi_g P(G|g) c(g|G)$$

We seek a classification rule with the minimal total expected misclassification cost; in the situation of equal missclassification costs, when $\mathcal{K} = K$, $c(g|G) = 0$ for $G = g$, and all other $c(g|G) = 1$, the equivalent criterion to minimize is the **total misclassification probability** (**error rate**)

$$\sum_{g} \sum_{G \neq g} \pi_g P(G|g)$$

Once again, the classification setting has fixed and given π_g and $c(g|G)$; what is needed to figure out are $P(G|g)$. If we model an outcome $\mathbf{x} \in \mathcal{X}$ of classifiers as a realization of a random element $\mathbf{X}$ of $\mathcal{X}$, with a distribution dependent on g, then

$$P(G|g) = P[\mathbf{X} \in \mathcal{R}_G|g]$$

Classification: the optimal Bayes rule

It is possible to derive the optimal rule if we know, for every g,

$f_g(x)$: the density of distribution of features (for $\mathbf{x} \in \mathcal{X}$), the density of $\mathbf{X}$ given that the item belongs into the g-th group. (This density has to be understood in a very broad sense, with respect to a suitable dominating measure: it can be a density, a probability mass function, or a combination of those)

The **optimal Bayes classification rule** is:

classify into G-th class

if $\sum_g \pi_g f_g(\mathbf{x}) c(g|G)$ is the smallest among all G

In case if there is more than one smallest quantity above, we may classify into any of these - the rule remains optimal for every alternative. In theory, such a situation often occurs with probability zero; in practice, we may need to resolve the ambiguity somehow - one of the (valid!) possibilities is then a random draw

The proof

Special cases

In the situation of equal misclassification costs, when $\mathcal{K} = \mathrm{K}$, $\mathbf{c}(\mathrm{g}|\mathrm{G}) = 0$ for $\mathrm{G} = \mathrm{g}$, and otherwise $\mathbf{c}(\mathrm{g}|\mathrm{G}) = 1$, the optimal Bayes rule is equivalent to:

classify to the G-th group, $\mathrm{G} = \mathrm{g}$

if $\pi_{\mathrm{g}} \mathrm{f}_{\mathrm{g}}(\mathbf{x})$ is the largest among all g

The last condition is equivalent to looking for the largest **posterior probability**

$$\frac{\pi_{\mathrm{g}} \mathrm{f}_{\mathrm{g}}(\mathbf{x})}{\sum_{\mathrm{g}} \pi_{\mathrm{g}} \mathrm{f}_{\mathrm{g}}(\mathbf{x})} \quad \text{is the largest among all } \mathrm{g}$$

The posterior probability for the g-th class is obtained via the Bayes theorem, as the conditional probability of an item being in the g-th class given the densities $\mathrm{f}_{\mathrm{g}}(\mathbf{x})$ and the prior probabilities π_{g}

Posterior probabilities provide additional information about how "clear-cut", "unambiguous" the classification is.

The special case of two classes

The case with $K = 2$, when there are only two classes, is quite common: apart from the examples we already have, we can mention other ones: classifying email as spam and non-spam, classifying patients as healthy and ill (with specific disease), etc.

The optimal Bayes classification rule is in this case as follows:

if $\pi_2 f_2(\mathbf{x}) c(2|1) \leqslant \pi_1 f_1(\mathbf{x}) c(1|2)$ then classify to class 1

if $\pi_2 f_2(\mathbf{x}) c(2|1) \geqslant \pi_1 f_1(\mathbf{x}) c(1|2)$ then classify to class 2

As already mentioned, in the of equality we may classify into either class, and the rule in each case remains optimal

The equivalent expression (neglect possible complications with division by zero) of the rule above is:

if $\dfrac{\pi_1 f_1(\mathbf{x})}{\pi_2 f_2(\mathbf{x})} \geqslant \dfrac{c(2|1)}{c(1|2)}$ or equivalently $\dfrac{f_1(\mathbf{x})}{f_2(\mathbf{x})} \geqslant \dfrac{\pi_2 c(2|1)}{\pi_1 c(1|2)}$

then classify to class 1 (and otherwise to 2)

Classification: the real life

Thus, there would be no problem here - if everything above would be known. Well, we may set the missclassification costs; but other quantities, especially the densities $\mathrm{f_g}$, and often also π_g, are in practice not known.

What then? Dealing with π_g is not that difficult, but $\mathrm{f_g}$ are not that easy to tackle. We can estimate them somehow, and use the estimates instead. But also, what we are really after are classification regions $\mathcal{R}_\mathrm{G}$, so we can estimate those: instead of the $\mathrm{f_g}$'s, we can estimate directly the posterior probabilities. Or perhaps do yet something else...

Thus, in practice we may not be able to use the optimal Bayes rule (except perhaps in very simple situations), but its existence is important anyway: it tells us *what is the best what we can achieve*

A word about prior probabilities

Regarding the prior probabilities, π_g,

- we either know them somehow, and then we *set them up* - this is what theory supposes us to do (the result about optimal classification regions does not hold for π_g somehow estimated from data, but only for *known* π_g and *known* f_g; when we estimate those quantities, we only hope getting close to the ideal)

- or in the lack of specific knowledge about π_g, we may set them to be all equal, $\pi_g = 1/K$ (this still means that we set them up)

The other alternative is that we *estimate* π_i, which can be done either

- either from the training sample itself

- or from some other dataset

(The estimation is then typically done by taking proportions)

The special case of two classes continued

In the case when $\mathrm{K} = 2$, once we estimate all necessary quantities and end up with concrete (estimated) rule, the expression derived above suggests that it has a form

if $\quad \text{rule}(\mathbf{x}) \geqslant \text{threshold} \quad$ then classify to 1, otherwise to 2

What goes to "rule" and what to "threshold"? That depends on the method.

If we *can* incorporate the prior probabilities π_1, π_2 into the rule (and know them), then $\text{threshold} = \dfrac{\mathbf{c}(2|1)}{\mathbf{c}(1|2)}$

if the prior probabilities are not in the rule, then the threshold may be $\dfrac{\pi_2 \mathbf{c}(1|2)}{\pi_1 \mathbf{c}(2|1)}$ for equal misclassification costs $\dfrac{\pi_2}{\pi_1}$

In any case, the lower the threshold, the easier to get classified as 1 (which can be motivated, say, by the cost of missclassification from 2 to 1 being high compared to that from 2 to 1). And conversely: the higher the threshold, the harder to get classified as 1

ROC curve

(ROC - Receiving operating characteristic; from engineering)

If we single out one of the classes, we can look at how the rule performs for various thresholds - which arise from different circumstances involving misclassification costs and possibly prior probabilities as well.

If we single out class 1 - let's say, classifying person as ill (in the medical language *positive*), then low threshold means we prefer to classify person rather ill than healthy (which comes from the comparison of misclassification costs and also possibly priors)

We can summarize the behavior for various thresholds by varying those, and plot

true positive rate against false positive rate

that is

$$\frac{\#\text{ of 1's classified as 1}}{\#\text{ of all 1's}} \quad \text{against} \quad \frac{\#\text{ of 2's classified as 1}}{\#\text{ of all 2's}}$$

The overall performance of classifier is then summarized by the Area Under the (ROC) Curve (AUC)

(The example will be shown later, with discriminant analysis)

Classification: theoretical error analysis

We have now a classification rule and would like to evaluate how good it is. If we knew the densities f_g, we would use them for deriving the optimal $\mathcal{R}_\mathrm{G}$; as we do not know them, we have to take with the estimates $\widehat{\mathcal{R}}_\mathrm{G}$. However, we can still assume (or pretend) we know the the densities f_g for the purpose of evaluation. In such case, we can express the relevant probabilities $\mathrm{P}(\mathrm{G}|\mathrm{g})$ as

$$\int_{\mathcal{R}_\mathrm{G}} \mathrm{f}_\mathrm{g}(\mathbf{x})\,\mathrm{d}\mathbf{x} \qquad \text{or} \qquad \int_{\widehat{\mathcal{R}}_\mathrm{G}} \mathrm{f}_\mathrm{g}(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

We know that the classification regions $\mathcal{R}_\mathrm{G}$ are optimal, hence in terms of the total expected misclassification costs or total misclassification probabilities, we cannot do any better in terms of the total expected misclassification cost

$$\sum_\mathrm{g}\sum_\mathrm{G} \mathrm{c}(\mathrm{g}|\mathrm{G})\pi_\mathrm{g}\int_{\mathcal{R}_\mathrm{G}} \mathrm{f}_\mathrm{g}(\mathbf{x})\,\mathrm{d}\mathbf{x} \leqslant \sum_\mathrm{g}\sum_\mathrm{G} \mathrm{c}(\mathrm{g}|\mathrm{G})\pi_\mathrm{g}\int_{\widehat{\mathcal{R}}_\mathrm{G}} \mathrm{f}_\mathrm{g}(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

or, when the misclassification costs are equal, in terms of the *true* error rate

$$\sum_\mathrm{g}\sum_{\mathrm{G}\neq\mathrm{g}} \pi_\mathrm{g}\int_{\mathcal{R}_\mathrm{G}} \mathrm{f}_\mathrm{g}(\mathbf{x})\,\mathrm{d}\mathbf{x} \leqslant \sum_\mathrm{g}\sum_{\mathrm{G}\neq\mathrm{g}} \pi_\mathrm{g}\int_{\widehat{\mathcal{R}}_\mathrm{G}} \mathrm{f}_\mathrm{g}(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

Classification: practical error analysis

But still, we do not know $\mathrm{f_g}$: so the only method to figure out the true error rate is to estimate it - by repeating the classification many times and recording the results.

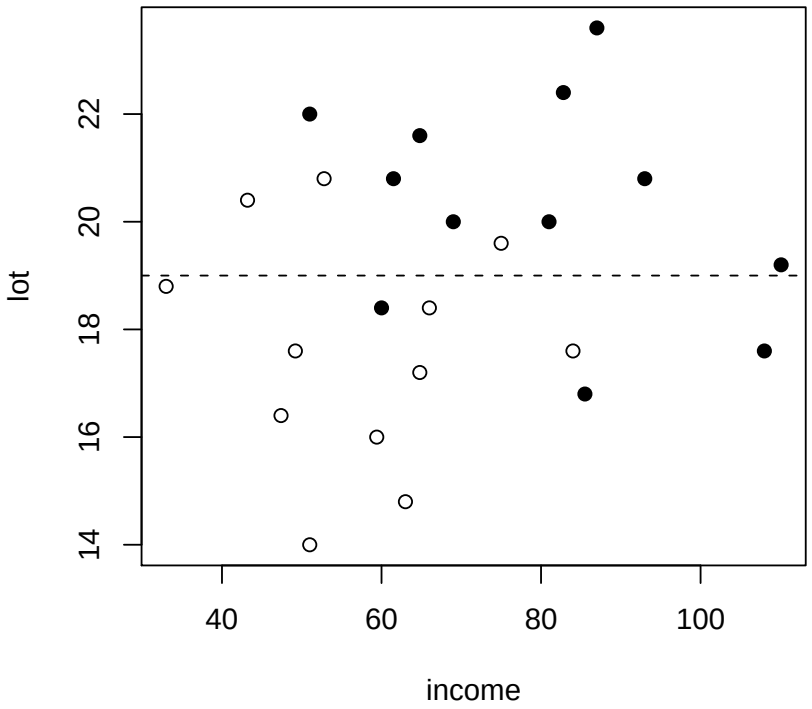
Apparent error rate: we crosstabulate the actual (known) classes of the sample and predicted (computed) classification for the same items. We obtain a $\mathrm{k} \times \mathrm{k}$ table ("*confusion matrix*") in this way: its diagonal shows the numbers of correctly classified items; all other numbers are those of incorrectly classified items. We take the sum of the latter and divide by the total number of items.

We are interested in the apparent error rate only as long as it gives a good estimate of the true error rate (probabilities), as this indicates the error rate for *future* observations - which, if low, indicates that the classification rule has good *generalization properties*.

A small example: more to come

A very crude classification: according to the lot size. If a customer has lot size large than 19, they are likely to buy a riding mower. Otherwise not. What is the apparent error rate?

(We believe in the continuity of lot size, which allows us not to worry about what happens when the size is exactly 19 - an event with zero probability. Problems of this type, while of practical importance, are here an inessential technical complication obscuring the greater picture.)




```
> cl = table(lot > 19,riding)
> cl
       riding
        0 1
  FALSE 9 3
  TRUE  3 9
> (cl[1,2]+cl[2,1])/sum(cl)
[1] 0.25
```

Data splitting

Apparent error rate calculated on the working (training) dataset is usually not very good estimate of the true error rate, as there is a possible "conflict of interest": the same items that were used for the estimation of the rule are now used for its evaluation, so the final results is likely to be biased. A better estimate is usually obtained through the apparent error rate calculated on some other data, different from those we constructed the classification rule from. In this respect, we speak about

Validation dataset(s): the dataset(s) we use for the evaluation of the performance of the classification rule

Now, if we have a lot of data items available, it is not hard to arrange for all required datasets: we split the data, use someof it to construct a classification rule, and keep aside some for the validation purpose

Note: this is an *ideal* strategy, and should be used whenever there is an abundance of data

Data reuse

The more intricate situation occurs when we do not have enough data for satisfactory splitting (the working, training/validation dataset is too small to yield stable results). Then, we would like to attempt using the working (training) dataset also for validation: estimate true error rate by the apparent error rate on the same dataset.

However, the results obtained in this way may have severe limitations; in particular, the biggest danger is that the error rate estimated on the working dataset may be too optimistic compared to the true error rate, the phenomenon wich is called **overfitting**.

While we never can avoid this completely, we can try at least partially. One of the possible strategies may be keeping the rule simple; if it cannot adapt too well to the working (training) sample, then the risk of overfitting is lesser. But the problem then may be also **underfitting**: some other rule, a bit more flexible, may performs better in terms of the true error.

Another, widely used and general strategy to reuse the same dataset and still at least partially avoid overfitting is **cross-validation**.

Cross-validation: the description

M-fold cross-validation (hold-out) proceeds as follows.

We divide data to M groups (approximately of the same size; it is also good to use some random allocation). For m from 1 to M, we

1. omit m-th group from the data;
2. estimate the classification rule from the rest of data;
3. classify the items of m-th group on the basis of this rule;
4. record the number of misclassified items.

When finished, we compute a summary apparent error rate from all M runs.

A popular choice is $\mathrm{M} = 10$, because then we have to run the procedure only ten times, which is good if it is computationally expensive. (Sometimes only $\mathrm{M} = 2$ is feasible.)

Leave-one-out cross-validation

Another popular choice is $M = n$, the number of data points; in this case we also speak about leave-one-out cross-validation. This is a favored choice if computations are inexpensive - in particular for procedures that are linear in classifiers - then we do not have to compute the classification n times actually, but there is a numerical trick avoiding that

Cross-validation may be used not only for the overall evaluation of the classification rule, but also for fine tuning (that is, determination of certain parameters) of an estimated classification procedure. We will see this later on examples

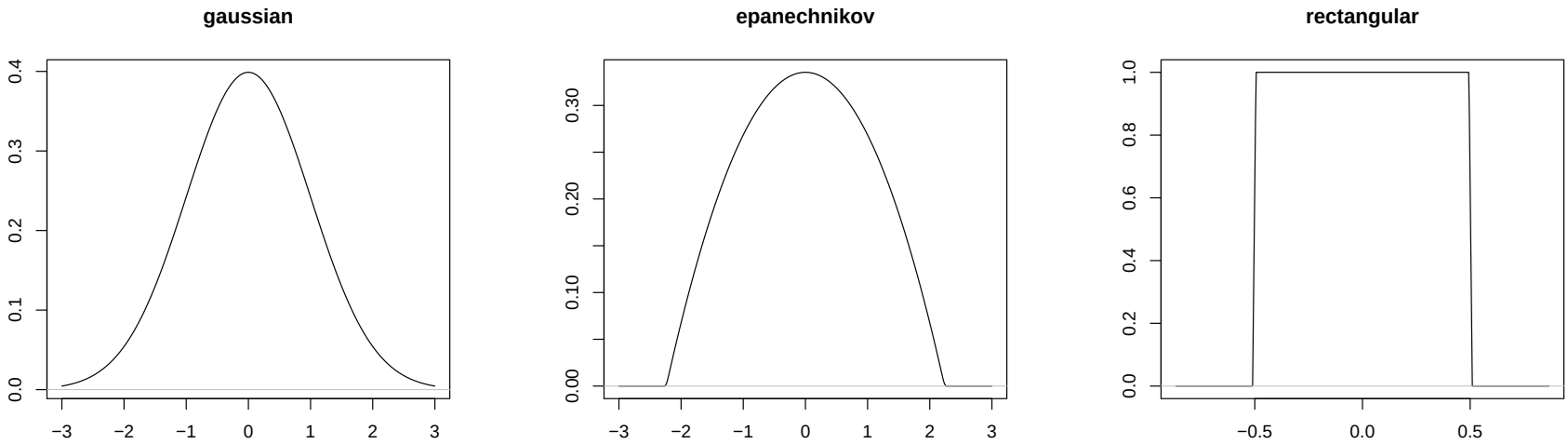
Nonparametric classification: density estimation and nearest neighbors

Recall: kernel density estimator

$$\widehat{f}(x) = \frac{1}{nb} \sum_i K\left(\frac{x_i - x}{b}\right)$$

kernel: $\int K(u)\,du = 1$ and also $K(u) \geqslant 0$

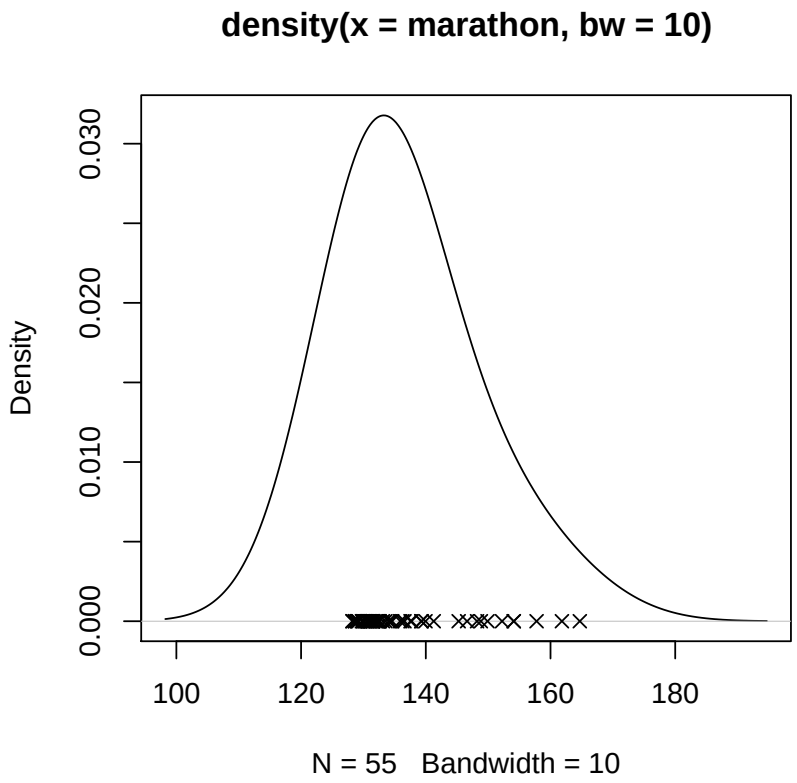
Examples: Gaussian (standard normal density), Epanechnikov, Rectangular (Parzen), and others

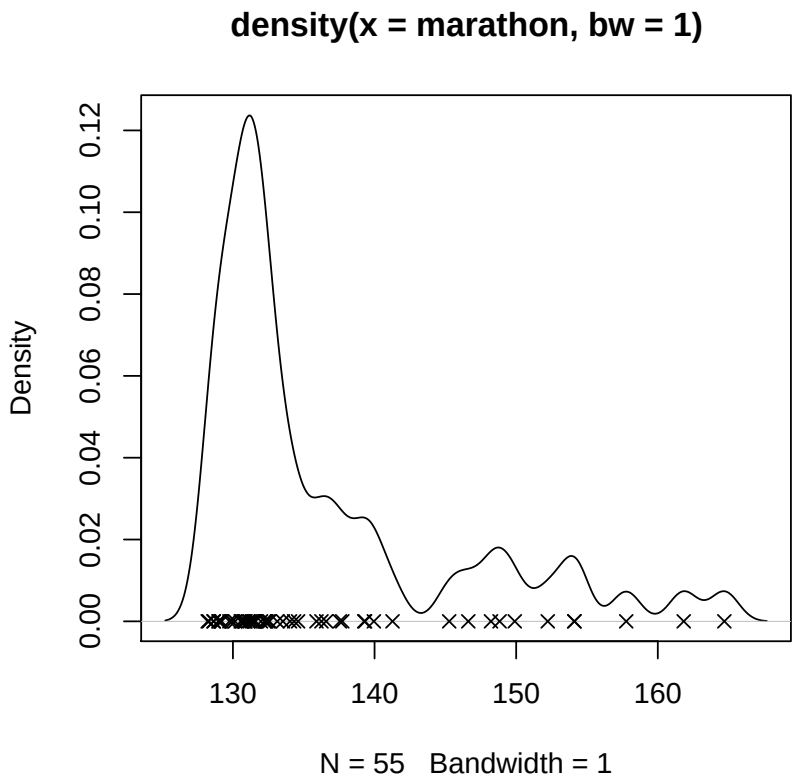


What does rectangular kernel mean? For $b = 1$, $\frac{1}{n}\sum_i K(x_i - x)$ is the relative proportion number of points falling into $[x - 1/2, x + 1/2]$; for general b, we obtain the relative proportion of points falling into $[x - b/2, x + b/2]$, divided by the length b of the interval.

Different bandwidth

The same *bandwidth* $\mathfrak{b}$ may not equally adapt to all parts of the data





Modification (inversion?) of the kernel idea

For fixed b and x, we take the relative proportion of points x_i falling into $[x - b/2, x + b/2]$ divided by the length of the interval...

Invert: for fixed x, take $k/(nb_k)$ where b_k is the length of an interval $[x - b_k/2, x + b_k/2]$ containing k *nearest neighbors*, neighboring data points x_i of x (this including x itself, if x is a datapoint, equal to some x_i).

Such modification is particularly good for classification, and leads to the

Method of k nearest neighbors

Method of k nearest neighbors:

(0. Scale all variables, so that they have unit standard deviation.)

1. Fix k

2. Given the classifiers $\mathbf{x}$ for an item to be classified, find k items in the working (training) dataset whose classifiers are closest (in some suitable, in the simplest case the Euclidean distance on $\mathbb{R}^p$) to $\mathbf{x}$. The posterior probability for the g-th class is then the proportion of elements from g-th class among the k nearest neighbors. The classification decision is (typically) done by the rule of simple majority (sometimes more elaborate voting schemes may be used)

Properties

Not that bad:

- nice interpretability ("out of k past items with closest features, ℓ belonged to category i")
- well adapts to irregular shapes ("leopard skin") of classification regions
- easily generealized to more than two classes

Not that good:

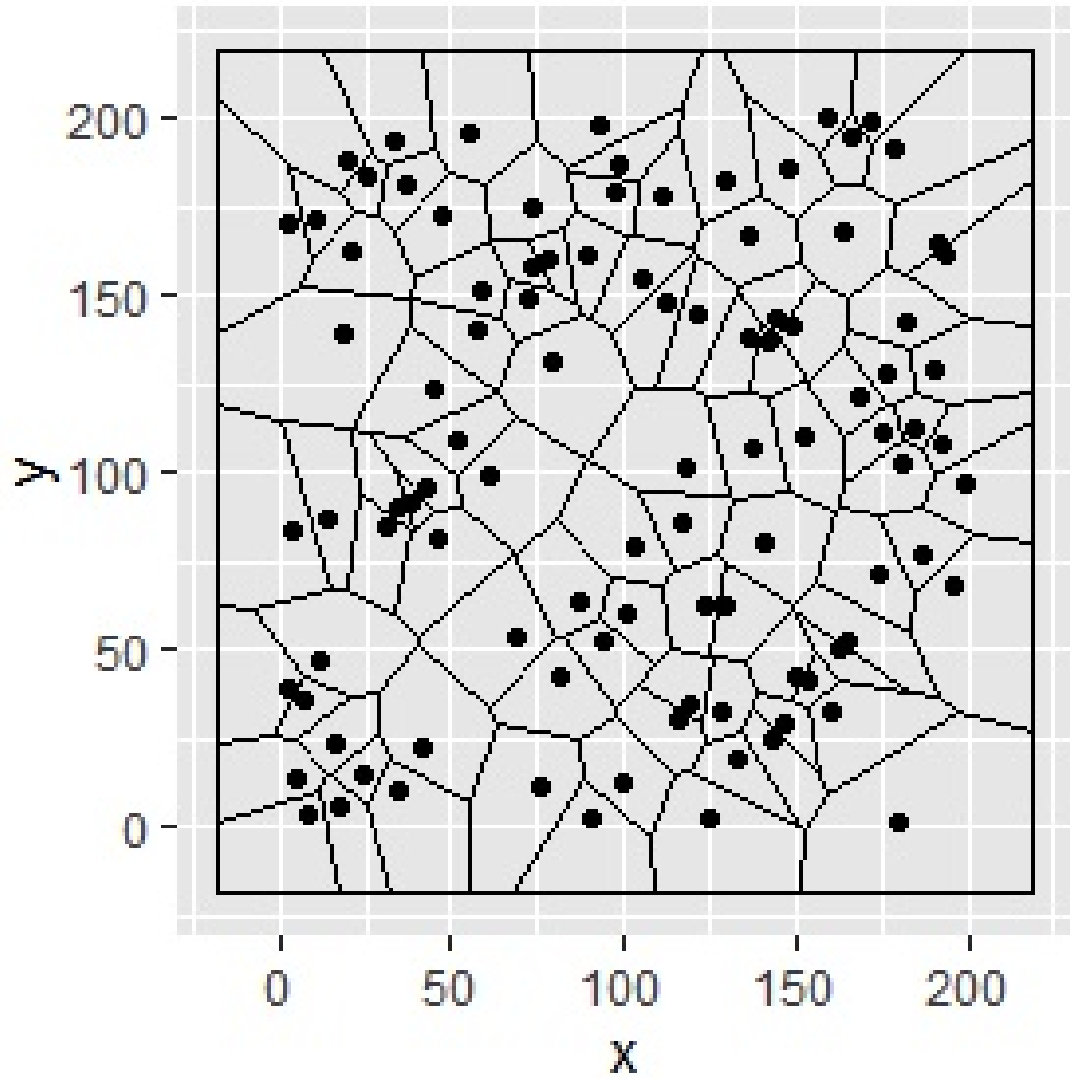
- computationally, need to store all the training set (but then, it can constantly learn)
- somewhat unstable
- not scale (affine) equivariant: scaling is usually a must
- needs to specify k

Extensions:

- treatment of priors/unequal classification costs
- "smooth" weights

Aspects of k

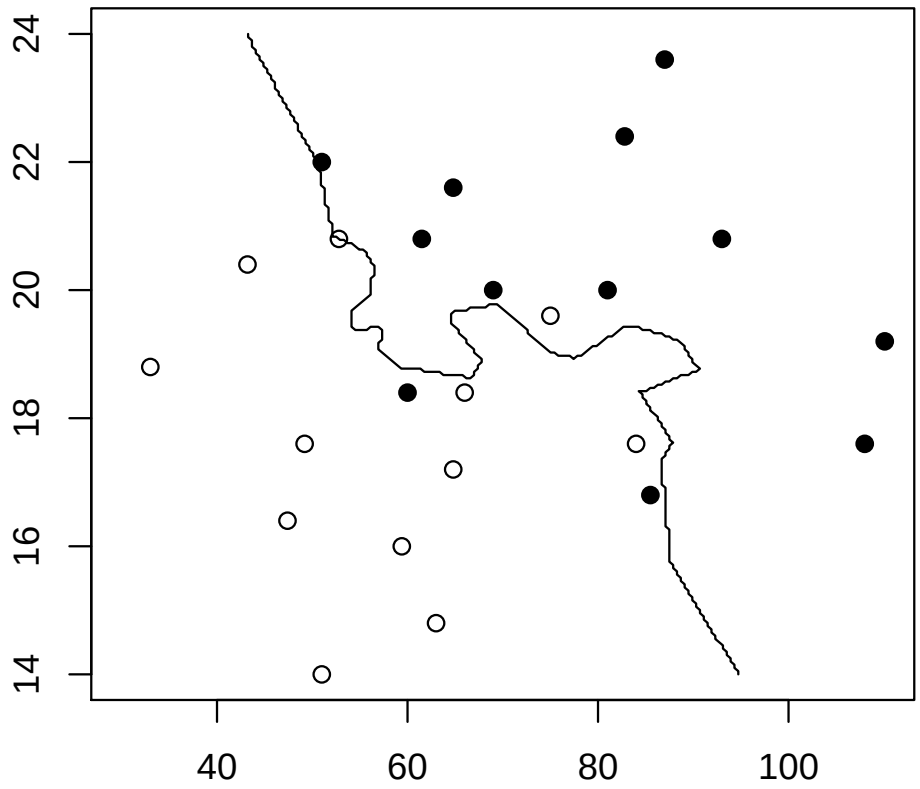
Can be used with $k = 1$; then it divides the feature space to Voronoi (Dirichlet, Thiessen) polygons.



Selection of k by (leave-one-out) cross-validation

```
> library(class)   ## note: a package is needed
> mow=scale(Mowers[,1:2])
> cltab=array(0,dim=c(2,2,10))
> clerr=rep(0,10)
> for (k in 1:10) {
+ cl=rep(0,nrow(mow))
+ for (j in 1:nrow(mow)) {
+ cl[j]=knn(mow[-j,1:2],Mowers[j,1:2],mowers[-j,3],k)
+ }
+ cltab[,,k]=table(cl,Mowers[,3])
+ clerr[k]=1-(cltab[1,1,k]+cltab[2,2,k])/sum(cltab[,,k])
+ }
> signif(clerr,3)
0.375 0.333 0.167 0.292 0.208 0.208 0.250 0.417 0.250 0.208
1     2     3     4     5     6     7     8     9     10
8 5   7 3   10 2  8 3   9 2    9 2  8 2   7 5   8 2   9 2
4 7   5 9    2 10 4 9   3 10   3 10 4 10  5 7   4 10  3 10
```

Mowers

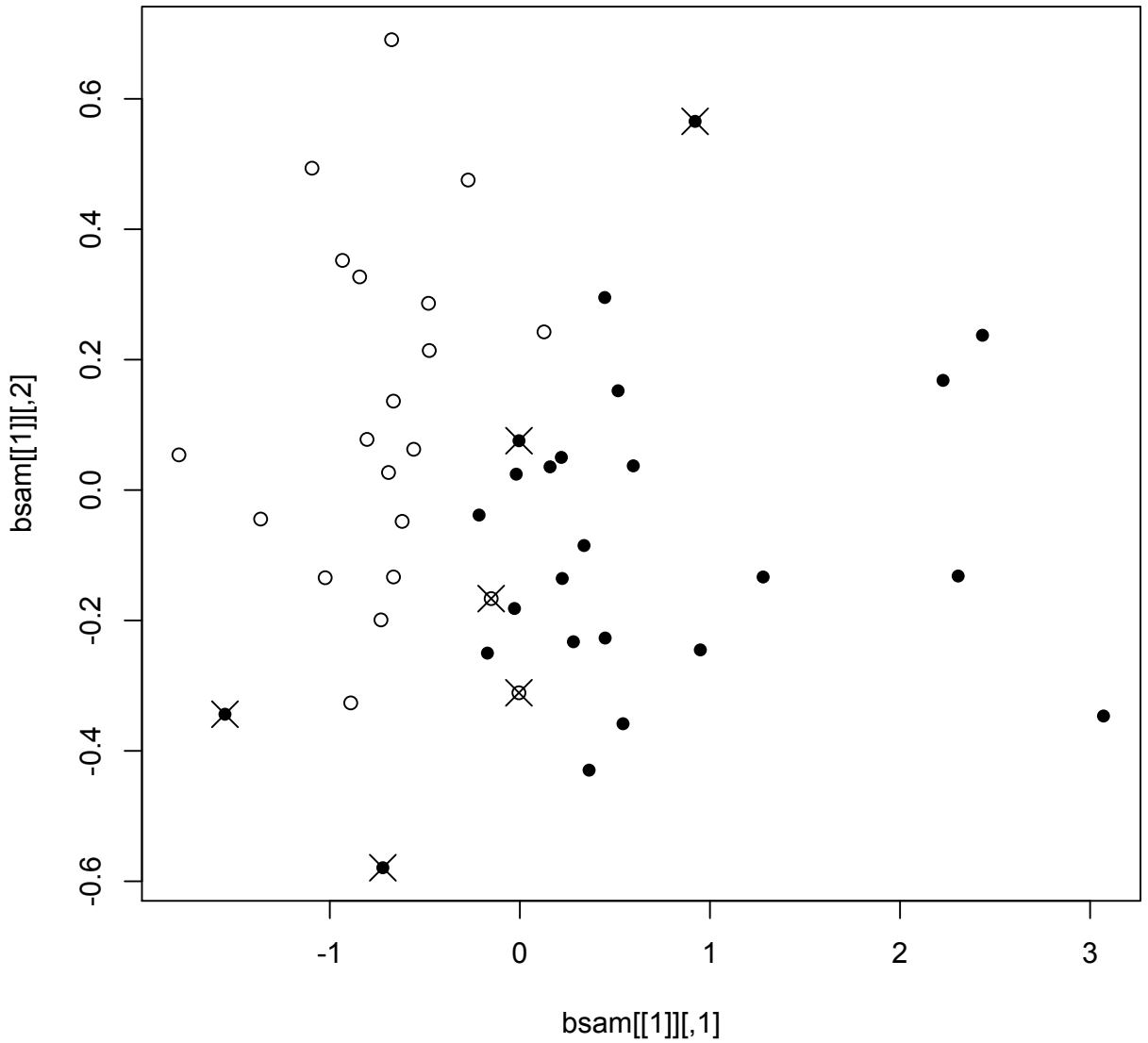



```
> table(knn(mow[,1:2],mow[,1:2],Mowers[,3],3),mowers[,3])
    0  1
  0 10  2
  1  2 10
```

Bank data: code

```
> ...
> signif(clerr,3)
0.196 0.196 0.152 0.152 0.152 0.109 0.196 0.217 0.196 0.217
> banksc = scale(Bank0)
> banknn=knn(banksc,banksc,Bank[,5],6,prob=TRUE)
> table(banknn,Bank[,5])

banknn  0  1
     0 19  4
     1  2 21
> bsam = sammon(dist(Bank[,1:4]))
> plot(bsam[[1]],pch=15*Bank[,5]+1)
> points(bsam[[1]][banknn!=Bank[,5],],pch=4,cex=2)
```

And the indicative plot of the result



Problems

```
> table(knn(banksc,banksc,Bank[,5],6),Bank[,5])
    0  1
  0 18  3
  1  3 22
> table(knn(banksc,banksc,Bank[,5],6),Bank[,5])
    0  1
  0 19  2
  1  2 23
> table(knn(banksc,banksc,Bank[,5],6),Bank[,5])
    0  1
  0 19  4
  1  2 21
> table(knn(banksc,banksc,Bank[,5],6),Bank[,5])
    0  1
  0 19  4
  1  2 21
> table(knn(banksc,banksc,Bank[,5],6),Bank[,5])
    0  1
  0 19  3
  1  2 22
```

Classification probabilities

```
> banknn$prob
NULL
> cls=signif(attr(banknn,"prob"),3)
> names(cls)=banknn
> cls
    0     0     0     0     0     0     0     0     0     0
1.000 1.000 0.833 1.000 0.833 0.833 0.667 1.000 0.833 1.000
    0     0     0     0     1     1     0     0     0     0
1.000 0.833 0.500 1.000 0.667 0.833 0.833 0.833 0.667 0.667
    0     1     1     1     1     1     1     1     0     1
1.000 0.833 0.833 0.833 0.833 1.000 1.000 0.833 0.667 0.833
    1     1     1     0     1     1     1     1     1     0
1.000 1.000 0.833 0.833 1.000 1.000 0.667 0.667 0.833 0.500
    0     1     1     1     1     1
0.500 1.000 0.833 0.833 0.667 1.000
```

Remedy?

Make all ambiguous votings classifying to 1 - and also make probs to be those for 1 (so 1-those are those for 0)

```
> knnadj = function(...) {
+ knnadj = knn(...,prob=TRUE)
+ knnadj[attr(knnadj,"prob")==0.5] <- 1
+ oldprob = attr(knnadj,"prob")
+ attr(knnadj,"prob") = (as.numeric(knnadj)-1)*oldprob +
+ (2-as.numeric(knnadj))*(1-oldprob)
+ knnadj
+ }

> bankknnn <- knnadj(banksc,banksc,Bank[,5],6)
> signif(cbind(bankknnn,1-attr(bankknnn,"prob"),
+ attr(bankknnn,"prob")),3)
```

Well...

```
   [,1]  [,2]  [,3]      [,1]  [,2]  [,3]      [,1]  [,2]  [,3]
 [1,] 0 1.000 0.000  [18,]  0 0.833 0.167  [35,]  1 0.000 1.000
 [2,] 0 1.000 0.000  [19,]  0 0.667 0.333  [36,]  1 0.000 1.000
 [3,] 0 0.833 0.167  [20,]  0 0.667 0.333  [37,]  1 0.333 0.667
 [4,] 0 1.000 0.000  [21,]  0 1.000 0.000  [38,]  1 0.333 0.667
 [5,] 0 0.833 0.167  [22,]  1 0.167 0.833  [39,]  1 0.167 0.833
 [6,] 0 0.833 0.167  [23,]  1 0.167 0.833  [40,]  1 0.500 0.500
 [7,] 0 0.667 0.333  [24,]  1 0.167 0.833  [41,]  1 0.500 0.500
 [8,] 0 1.000 0.000  [25,]  1 0.167 0.833  [42,]  1 0.000 1.000
 [9,] 0 0.833 0.167  [26,]  1 0.000 1.000  [43,]  1 0.167 0.833
[10,] 0 1.000 0.000  [27,]  1 0.000 1.000  [44,]  1 0.167 0.833
[11,] 0 1.000 0.000  [28,]  1 0.167 0.833  [45,]  1 0.333 0.667
[12,] 0 0.833 0.167  [29,]  0 0.667 0.333  [46,]  1 0.000 1.000
[13,] 1 0.500 0.500  [30,]  1 0.167 0.833
[14,] 0 1.000 0.000  [31,]  1 0.000 1.000
[15,] 1 0.333 0.667  [32,]  1 0.000 1.000
[16,] 1 0.167 0.833  [33,]  1 0.167 0.833
[17,] 0 0.833 0.167  [34,]  0 0.833 0.167
```

...and also

```
> table(knnadj(banksc,banksc,Bank[,5],6),Bank[,5])

   0  1
0 18  2
1  3 23
> table(knnadj(banksc,banksc,Bank[,5],6),Bank[,5])

   0  1
0 18  2
1  3 23
> table(knnadj(banksc,banksc,Bank[,5],6),Bank[,5])

   0  1
0 18  2
1  3 23
> table(knnadj(banksc,banksc,Bank[,5],6),Bank[,5])

   0  1
0 18  2
1  3 23
```

Classification classics: discriminant analysis

Assume that the distributions of the groups are multivariate normal

$$f(x) = \frac{1}{\sqrt{(2\pi)^p \det(\mathbf{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

We invoke the optimal rule for classifying with minimum expected cost of misclassification:

classify to G-th class when $\sum_{g \neq G} \pi_g f_g(\mathbf{x}) c(g|G)$ is the smallest

For equal misclassification costs, this rule is equivalent to:

classify to g-th class when $\pi_g f_g(\mathbf{x})$ is the largest

or equivalently

classify to g-th class when $\log(\pi_g f_g(\mathbf{x}))$ is the largest

We will evaluate those log-scores now

Classification scores, LDA and QDA

When all f_g are multivariate normal, then the (log) scores are

$$\log \pi_g - \tfrac{1}{2} \log \det(\mathbf{\Sigma}_g) - \tfrac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_g)^\top \mathbf{\Sigma}_g^{-1}(\mathbf{x} - \boldsymbol{\mu}_g)$$

Note: we are always omitting terms irrelevant to comparisons - constants present in all scores, etc. Thus, if all $\mathbf{\Sigma}_g$ are all equal to one $\mathbf{\Sigma}$, then the scores simplify to

$$\log \pi_g - \tfrac{1}{2} \boldsymbol{\mu}_g^\top \mathbf{\Sigma}^{-1} \boldsymbol{\mu}_g + \boldsymbol{\mu}_g^\top \mathbf{\Sigma}^{-1} \mathbf{x}$$

These scores are linear in $\mathbf{x}$ and thus also the boundaries of the classification regions they induce are linear; hence we speak about Linear Discriminant Analysis (LDA)

If $\mathbf{\Sigma}_g$ are considered not equal, then the boundaries are quadratic: we speak about Quadratic Discriminant Analysis (QDA)

Well: but $\boldsymbol{\mu}_g$ and $\boldsymbol{\Sigma}_g$ (or $\boldsymbol{\Sigma}$) are not known

Not known? Replace by estimates!

In the actual scores, $\boldsymbol{\mu}_g$ and $\boldsymbol{\Sigma}_g$ are replaced by their estimates - sample versions $\bar{\mathbf{x}}_g$ and $\mathbf{S}_g$

When all $\boldsymbol{\Sigma}_g$ are considered equal to a common $\boldsymbol{\Sigma}$, then $\boldsymbol{\Sigma}$ is replaced by the pooled estimate

$$\mathbf{S}_{\text{pooled}} = \frac{\sum_g (n_g - 1)\mathbf{S}_g}{\sum_g (n_g - 1)}$$

There are also other ways to deal with the $\boldsymbol{\mu}_g$ and $\boldsymbol{\Sigma}_g$, but this is the most straightforward, and often satisfactory. It is called **plug-in rule** or prediction

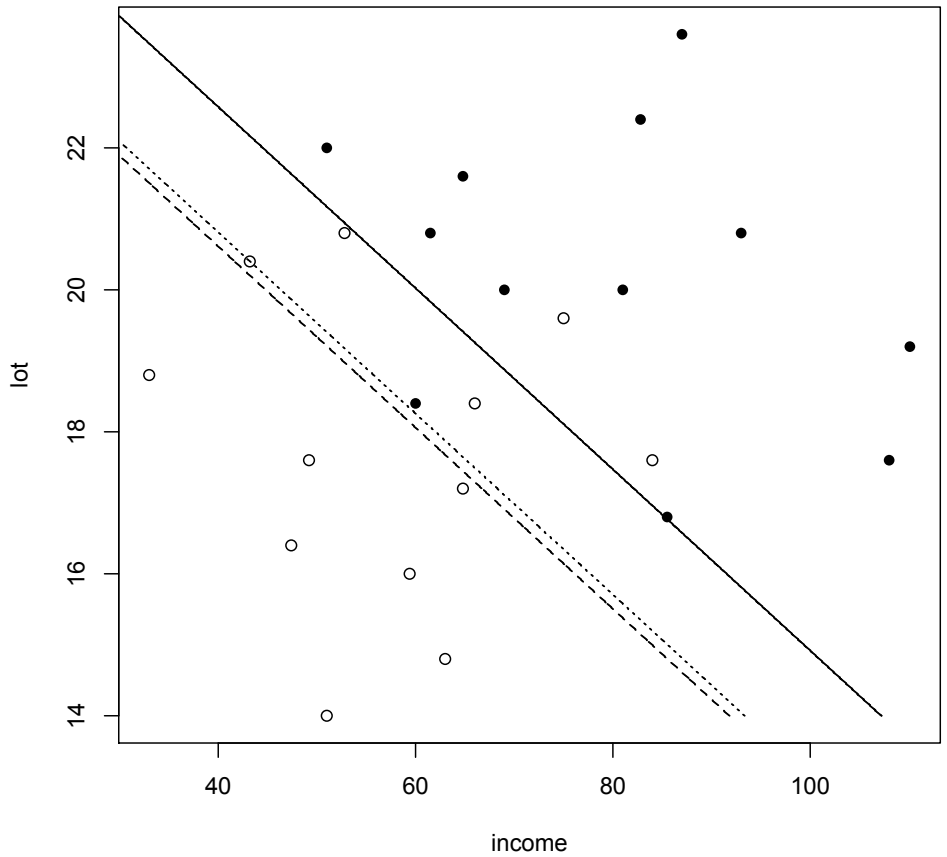
A true Bayesian would not do that, but instead use a **predictive** rule (prediction), which takes a linear combination of the normal distribution for various $\boldsymbol{\mu}_g$, each weighted by its posterior probability.

Mowers: LDA

Solid line: prior estimated from the data, thus equal to (0.5,0.5)
plug-in and predictive predictions equal

Dotted line: prior set to (0.2,0.8), plug-in prediction

Dashed line: prior set to (0.2,0.8), predictive prediction



Some code to try

```
> library(MASS)
> ldob=lda(riding~income+lot,data=Mowers)
> ldob
...
> predict(ldob)
...
> predict(ldob)$class
 [1] 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0
Levels: 0 1
> predict(ldob)$posterior
           0           1
1  0.78203155 0.217968446
2  0.49449211 0.505507885
3  0.15236751 0.847632493
4  0.31924493 0.680755073
5  0.00402325 0.995976750
...
> ?lda
> ?predict.lda
```

LDA another way: Fisher's linear discriminants

They are something like "principal components for K groups": we look for $\mathbf{a}$('s) maximizing

$$\frac{\mathbf{a}^\top \left(\sum_g (\boldsymbol{\mu}_g - \bar{\boldsymbol{\mu}})(\boldsymbol{\mu}_g - \bar{\boldsymbol{\mu}})^\top \right) \mathbf{a}}{\mathbf{a}^\top \boldsymbol{\Sigma} \mathbf{a}}$$

In the sample version:

$$\frac{\mathbf{a}^\top \left(\sum_g n_g (\bar{\mathbf{x}}_g - \bar{\mathbf{x}})(\bar{\mathbf{x}}_g - \bar{\mathbf{x}})^\top \right) \mathbf{a}}{\mathbf{a}^\top \left(\sum_g \sum_j^{n_g} (\mathbf{x}_{jg} - \bar{\mathbf{x}}_g)(\mathbf{x}_{jg} - \bar{\mathbf{x}}_g)^\top \right) \mathbf{a}} = \frac{\mathbf{a}^\top \mathbf{B} \mathbf{a}}{\mathbf{a}^\top \mathbf{W} \mathbf{a}},$$

where $\bar{\mathbf{x}}_g$ is the sample mean of the g-th class and $\bar{\mathbf{x}}$ is the sample mean of all data. Note that the maximized function is same for $c\mathbf{a}$ as for $\mathbf{a}$; hence we may set $\mathbf{a}^\top \mathbf{a} = 1$, for all $\mathbf{a}$.

Similarly to principal component analysis, we are looking for maximizing $\mathbf{a}_1$; then for maximizing $\mathbf{a}_2$ orthogonal to $\mathbf{a}_1$; and so forth. The solutions are found via eigenvalues and eigenvectors of $\mathbf{W}^{-1}\mathbf{B}$

Fisher's linear discriminants

The rank of matrix $\mathbf{W}^{-1}\mathbf{B}$ is at most $\min\{p, K-1\}$

as the rank of $\mathbf{W}^{-1}$ is p (easy)

and the rank of $\mathbf{B}$ is $K-1$ (a bit of exercise)

Fisher's linear discriminants: projections of the datapoints given by the maximizing $\mathbf{a}_j$: given that $\mathbf{a}_j^T\mathbf{a}_j = 1$, these are vectors of length n, with elements

$$\mathbf{a}_j^T\mathbf{x}_i \qquad \text{for } i = 1, \ldots, n$$

The restriction on the rank of $\mathbf{W}^{-1}\mathbf{B}$ means there are at most $\min\{p, K-1\}$ linear discriminants; recall that p is the number of classifiers and K the number of classes

And the classification rule based on them

The classification rule using *first* r linear discriminants would be:

classify to G-th class if

$$\sum_{j=1}^{r} (\mathbf{a}_j^T(\mathbf{x} - \bar{\mathbf{x}}_G))^2 \leqslant \sum_{j=1}^{r} (\mathbf{a}_j^T(\mathbf{x} - \bar{\mathbf{x}}_g))^2 \text{ for all } g \neq G$$

It is equivalent to the LDA as derived above, if:

all prior probabilities are equal

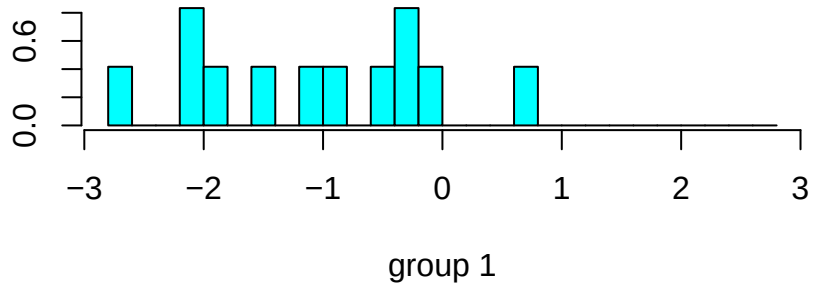
and $r = g - 1$ (all discriminants are used)

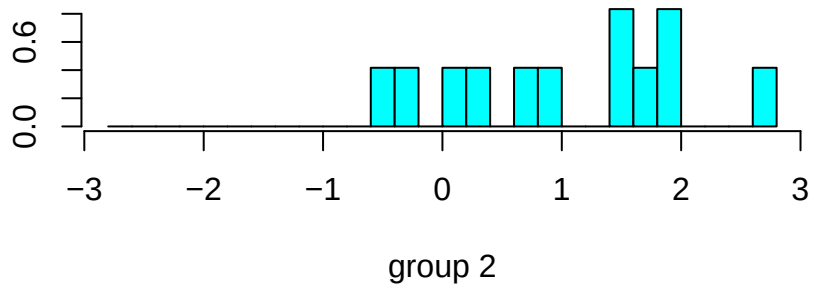
Indeed

```
> try.lda=lda(riding~income+lot,data=Mowers)
> try.lda
...
Group means:
  income      lot
0 57.400 17.63333
1 79.475 20.26667

Coefficients of linear discriminants:
              LD1
income 0.0484468
lot    0.3795228
> trycf=try.lda$scaling/sqrt(sum(try.lda$scaling^2))
> sc1=(as.matrix(Mowers[,1:2]) %*% trycf)
+ - c(try.lda$means[1,] %*% trycf)
> sc2=(as.matrix(Mowers[,1:2]) %*% trycf)
+ - c(try.lda$means[2,] %*% trycf)
> as.numeric(sc2^2 <= sc1^2)
 [1] 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0
> predict(try.lda)$class
 [1] 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0
```

Plotting

Linear discriminants (first one or two) are often plotted (like principal components)

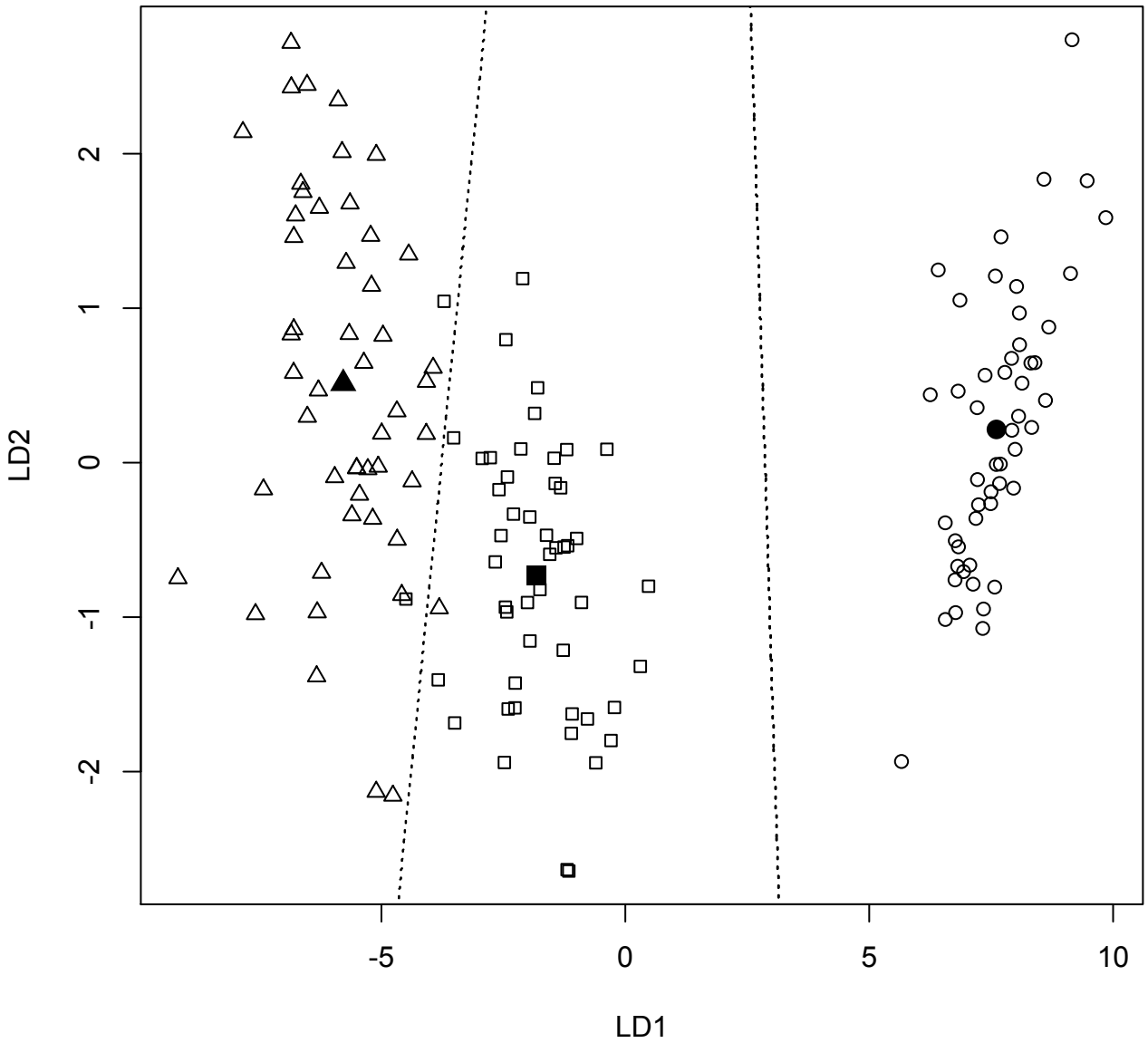




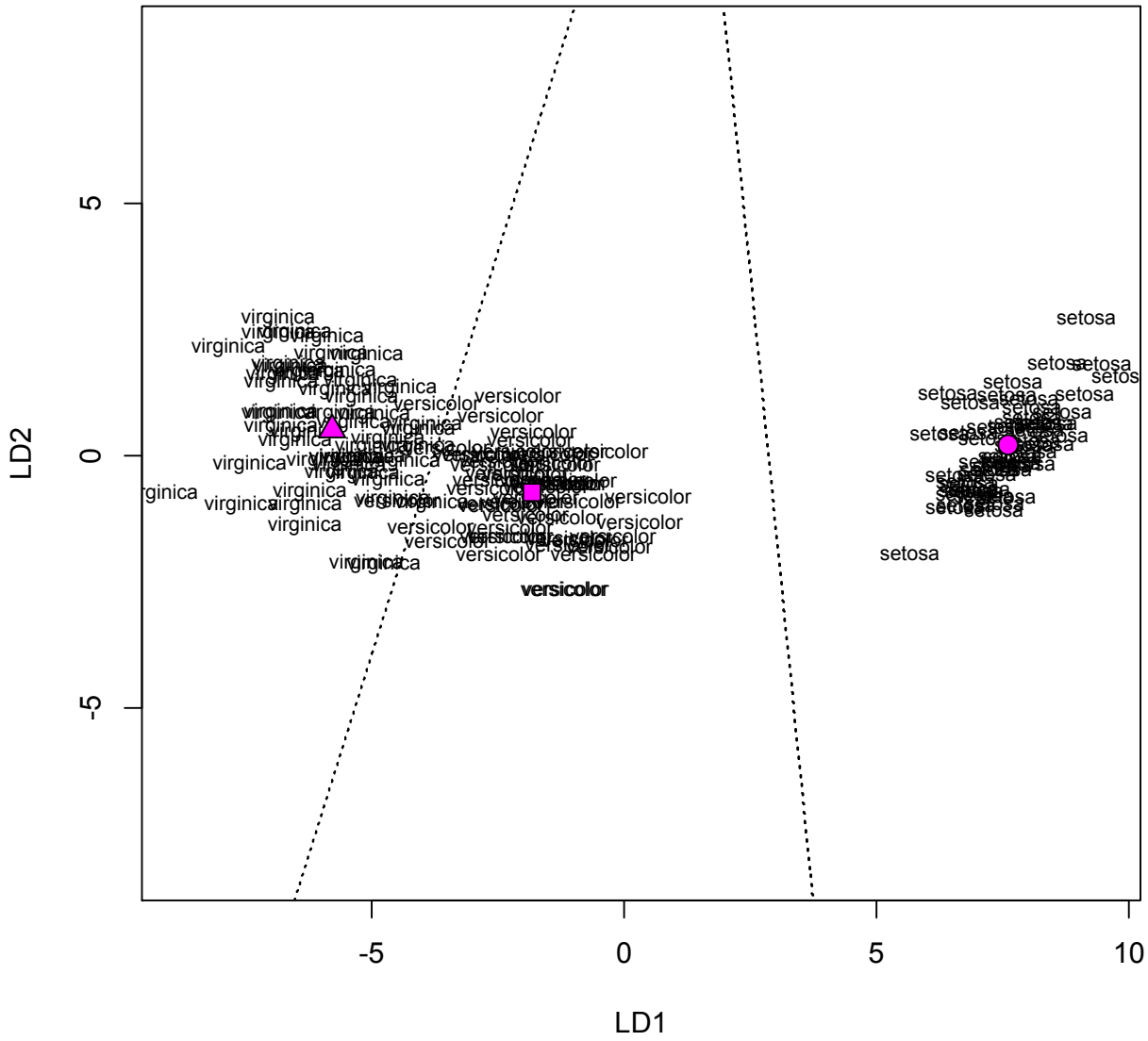
Iris data

```
> try.lda=lda(Species~.,data=iris)
> try.lda
...
Prior probabilities of groups:
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333
Group means:
           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026
Coefficients of linear discriminants:
                    LD1         LD2
Sepal.Length  0.8293776  0.02410215
Sepal.Width   1.5344731  2.16452123
Petal.Length -2.2012117 -0.93192121
Petal.Width  -2.8104603  2.83918785

Proportion of trace:
   LD1    LD2
0.9912 0.0088
```

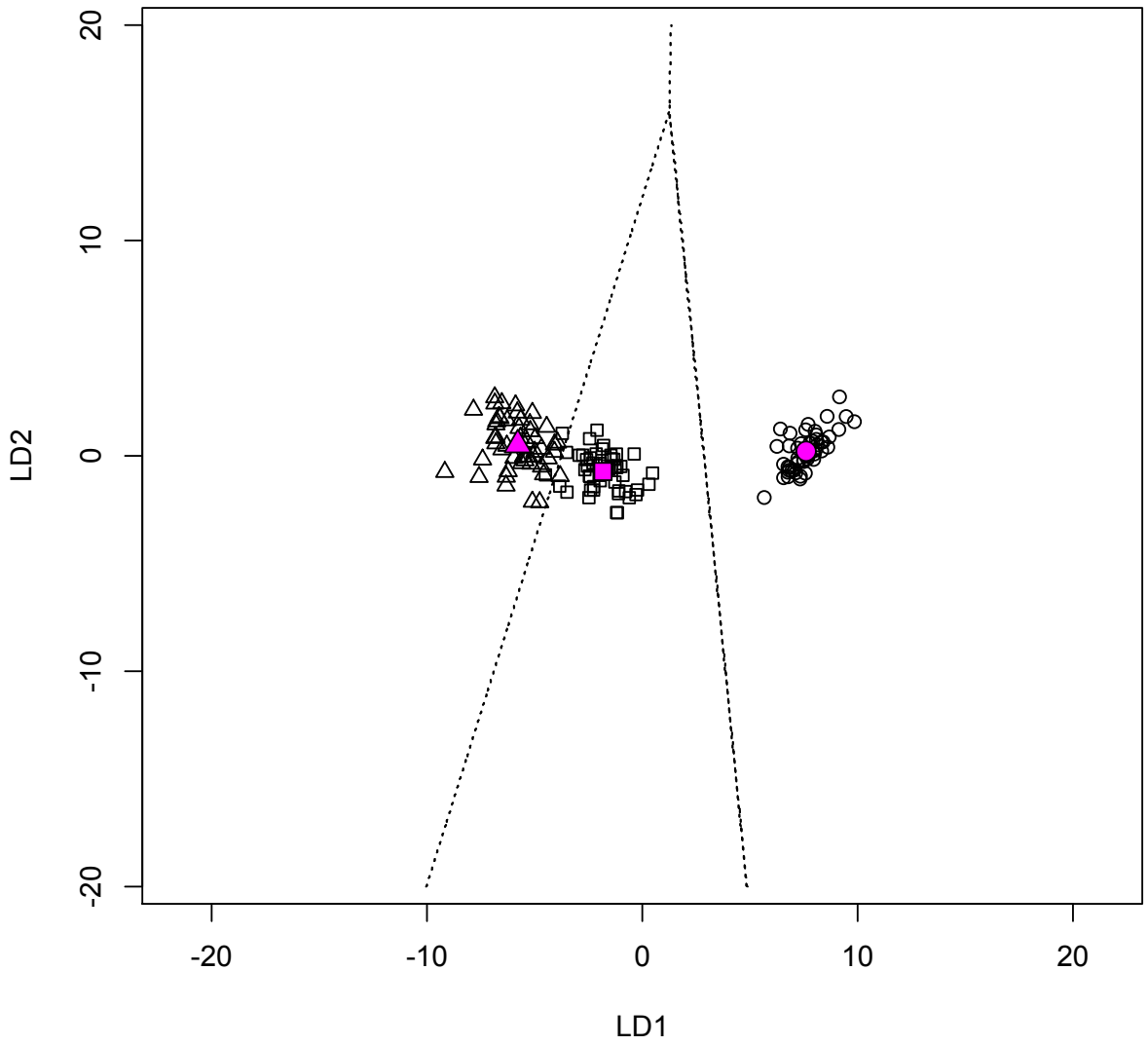
Linear discriminants for Iris data



The geometry shows correctly only on the equiscaled plot



The bigger (and still equiscaled) picture



QDA and other aspects

If we don't assume variance matrices equal, and fit them separately, then we obtain quadratic boundaries: quadratic discriminant analysis (QDA). Its results "may be strange".

We don't have to scale the variables - LDA is not vulnerable to it. In what sense: if we transform classifiers linearly, the results of LDA transform accordingly.

Other aspects:

- we can also use transformed features as classifiers

- variable selection (selection of classifiers)

Implementation: functions `lda()` and `qda()` in `library(MASS)`. They use model formula notation: response is the class indicator, predictors are classifiers.

Regression interpretation

The connection to least squares

If there are only two classes (coded -1 and 1), and *if there is the same number of items in these classes*, then, if we run a linear regression of these classes (with the codes above) on the classifiers, the intersection of the fitted plane with level 0 (now we see why we need fixed codes) gives the separating plane for the linear discriminant analysis with equal prior probabilities and misclassification costs.

Even if the number of the points in the classes is not the same: the level zero set is parallel

The connection to logistic regression

That will be thoroughly discussed later; note now that the estimates "logarithmic scores" are in fact estimates of the logs of posterior probabilities (or proportional to those)

Also, R implementation draws on regression interpretation: formula notation, response, etc.

Mowers: QDA in an analogous way as LDA

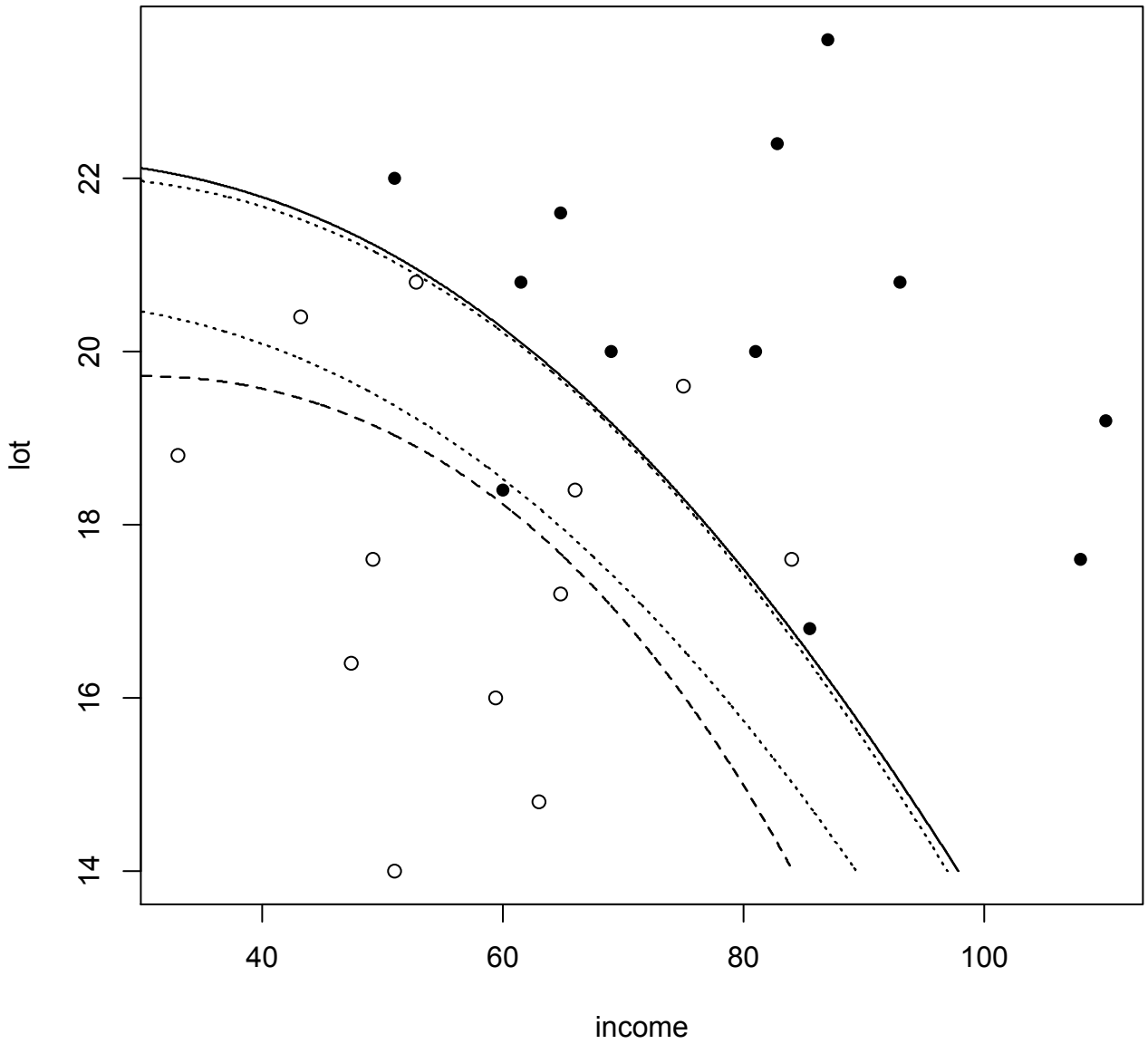
```
> library(MASS)
> attach(Mowers)
> plot(income,lot,pch=15*riding+1)
> try.qda
Call:
qda(riding ~ income + lot, data = Mowers)

Prior probabilities of groups:
  0   1
0.5 0.5

Group means:
  income      lot
0 57.400 17.63333
1 79.475 20.26667
```

Solid and dotted line: prior estimated from the data, (0.5,0.5)
plug-in and predictive predictions are different, however

Dotted line: prior set to (0.2,0.8), plug-in prediction

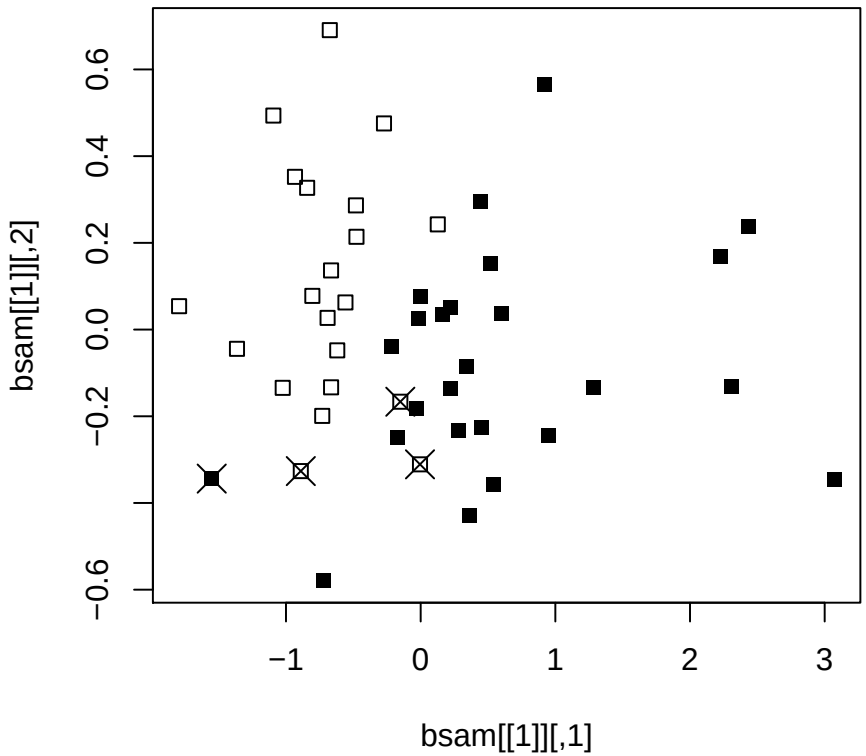
Dashed line: prior set to (0.2,0.8), predictive prediction

The QDA picture



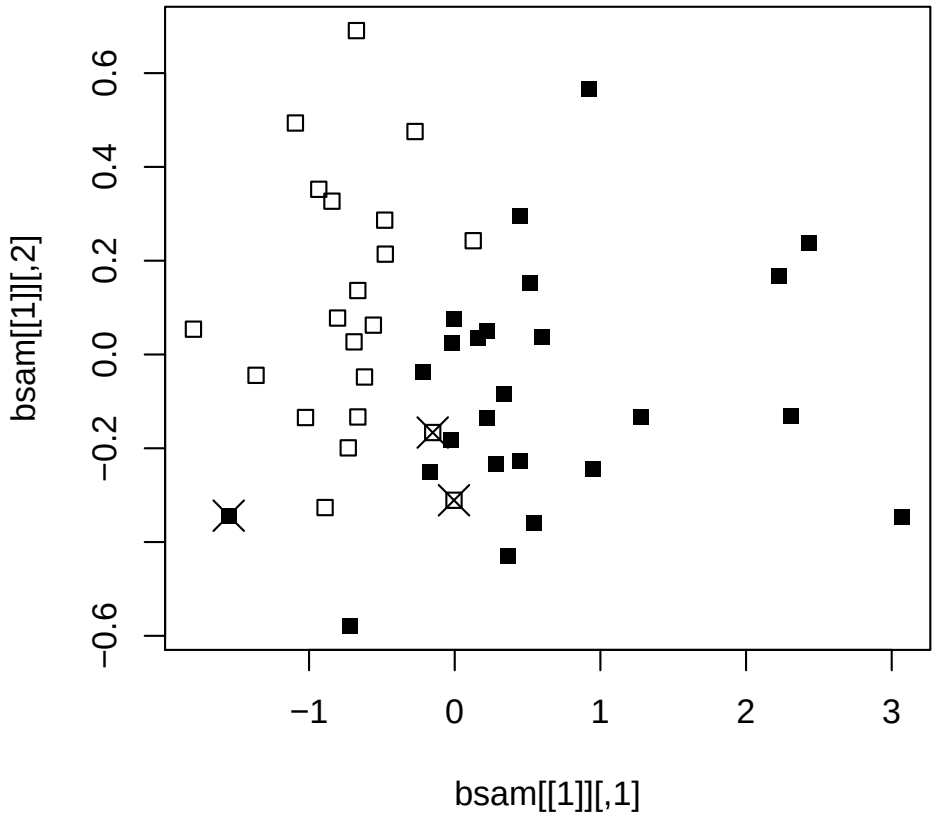
Bank data: LDA

```
> library(MASS)
> bsam=sammon(dist(Bank[,1:4]))
> plot(bsam[[1]],pch=15*Bank[,5])
> banlda=lda(k~.,data=Bank)
> wrong=Bank[,5] != predict(banlda)$class
> points(bsam[[1]][wrong,],pch=4,cex=2)
```

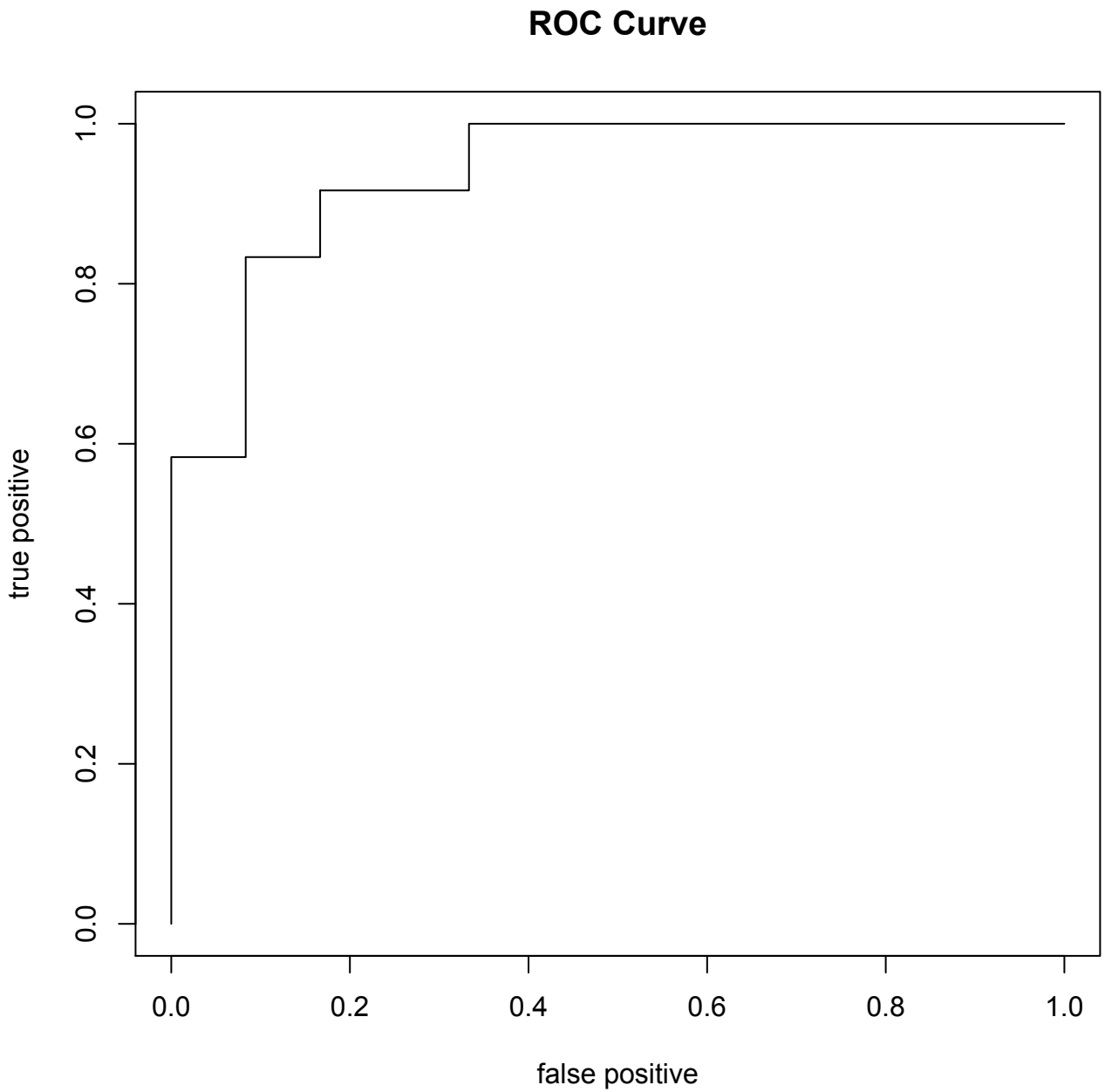


Bank data: QDA

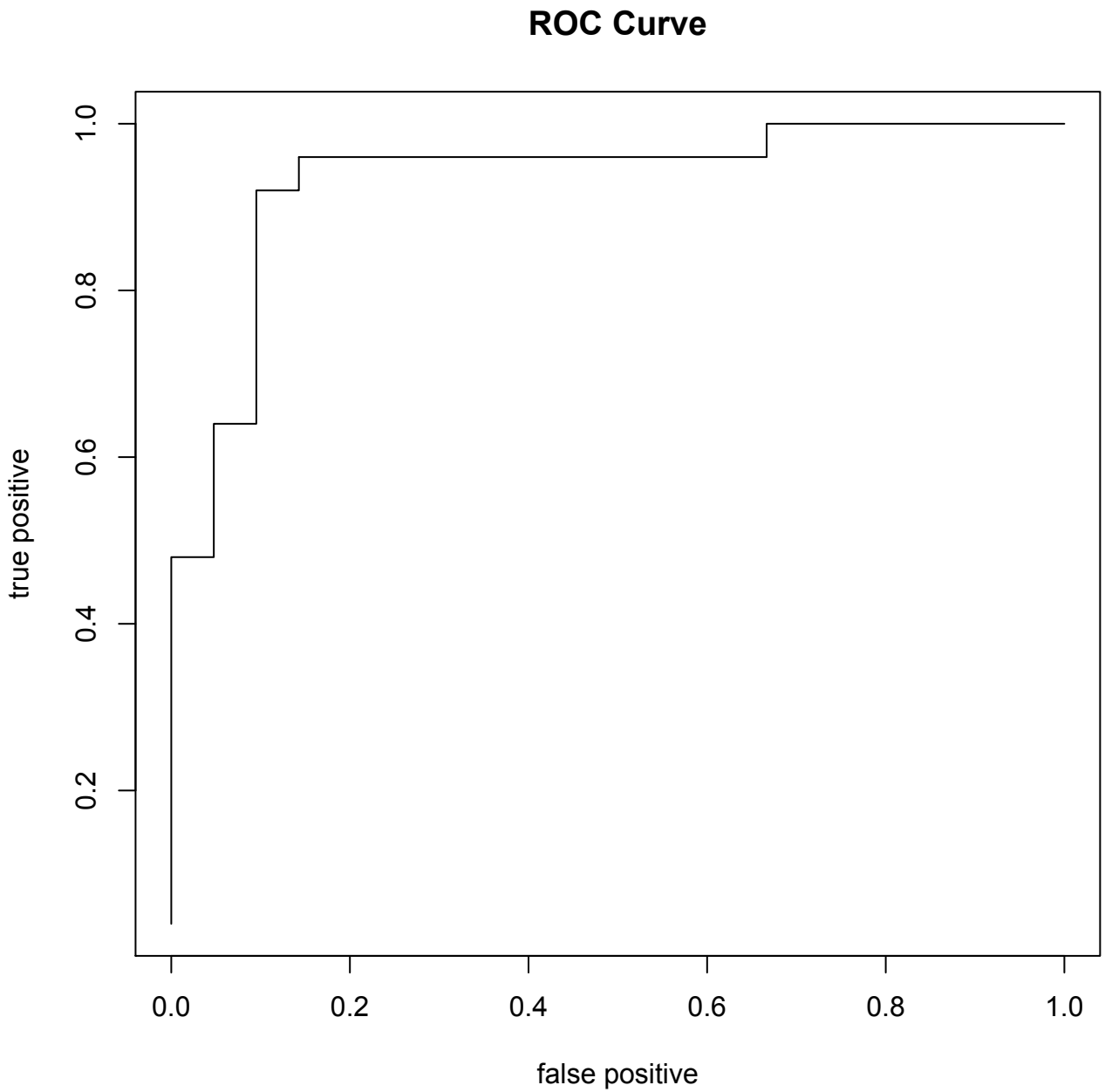
```
> bsam=sammon(dist(Bank[,1:4]))
> plot(bsam[[1]],pch=15*Bank[,5])
> banqda=qda(k~.,data=Bank)
> wrong=Bank[,5] != predict(banqda)$class
> points(bsam[[1]][wrong,],pch=4,cex=2)
```



Finally, promised ROC curves: mowers



And bank data



Classification via regression: logistic regression and relatives

Multinomial logistic regression

N observations Y_i falling into K classes

N_g of them falling in the g-th class, $N_1 + N_2 + \cdots + N_K = N$

Each observation falls into g-th class with probability $p_g(\mathbf{x})$ depending on $\mathbf{x}$

$$p_1(\mathbf{x}) + p_2(\mathbf{x}) + \cdots + p_K(\mathbf{x}) = 1$$

For the i-th observation $\mathbf{x}$ becomes $\mathbf{x}_i$; for the g-th class, the dependence on $\mathbf{x}_i$ is expressed in linear way through parameter vector $\boldsymbol{\beta}_g$. We abbreviate

$$\alpha + \beta_{1g} x_{i1} + \ldots \beta_{pg} x_{ip} \quad \text{as} \quad \alpha + \mathbf{x}_i^T \boldsymbol{\beta}_g$$

Note: while we have common α here for every g, there may be extra intercept for each g via the other covariates

Poisson model

The "symmetric" approach relaxes the requirement that all N_g sum to N, and thus may take the Poisson, loglinear model, for the counts Y_{ig}:

$$\log(E(Y_{ig})) = \alpha + \mathbf{x}_i^T \boldsymbol{\beta}_g$$

Once the counts are estimated, one finds their sum (corresponding to N) divides by it, to yield probabilities summing to 1; this determines α. The model for the probabilities is then

$$P[Y_i{=}1] = \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_1}}{\sum_{g=1}^{K} e^{\mathbf{x}_i^T \boldsymbol{\beta}_g}} \qquad P[Y_i{=}2] = \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_2}}{\sum_{g=1}^{K} e^{\mathbf{x}_i^T \boldsymbol{\beta}_g}} \qquad \ldots \qquad P[Y_i{=}K] = \frac{e^{\mathbf{x}_i^T \boldsymbol{\beta}_K}}{\sum_{g=1}^{K} e^{\mathbf{x}_i^T \boldsymbol{\beta}_g}}$$

This model is not identified: one can add any fixed constant to $\boldsymbol{\beta}_g$ and the fraction remains the same

Equivalent logistic models

To identify the model, we may set coefficients for some g to zero; this could be done for $g = K$, or, as in the R implementation, for $g = 1$. The model then becomes

$$P[Y_i{=}1] = \frac{1}{1 + \sum_{g=2}^{K} e^{x_i^T \beta_g}}$$

$$P[Y_i{=}2] = \frac{e^{x_i^T \beta_2}}{1 + \sum_{g=2}^{K} e^{x_i^T \beta_g}} \quad \ldots \quad P[Y_i{=}K] = \frac{e^{x_i^T \beta_K}}{1 + \sum_{g=2}^{K} e^{x_i^T \beta_g}}$$

which can be equivalently interpreted as $K-1$ logistic models: class 2 vs class 1, class 3 vs class 1, ..., class K vs class 1

However, the choice of $g = 1$ or $g = K$ is not substantial: the model is inherently symmetric (as is the standard binary logistic model)

The model is then estimated by (penalized) maximum likelihood

The probability predictions are the predictions of posterior probabilities - and are used accordingly to predict the class

Logistic regression (univariate)

Logistic regression is thus a special case - only customarily the classes are now coded as 0 and 1. We also change $\boldsymbol{\beta}_2$ to $\boldsymbol{\beta}$:

$$\mathrm{P}[\mathrm{Y_i}{=}0] = \frac{1}{1+\mathrm{e}^{\mathbf{x}_i^\top \boldsymbol{\beta}}} \qquad \mathrm{P}[\mathrm{Y_i}{=}1] = \frac{\mathrm{e}^{\mathbf{x}_i^\top \boldsymbol{\beta}}}{1+\mathrm{e}^{\mathbf{x}_i^\top \boldsymbol{\beta}}}$$

Knowing that $\mathrm{P}[\mathrm{Y_i}{=}0] = 1 - \mathrm{P}[\mathrm{Y_i}{=}1]$, we can focus on $\mathrm{P}[\mathrm{Y_i}{=}1]$

and use the logit transformation

$$\log\left(\frac{\mathrm{P}[\mathrm{Y_i}=1]}{1-\mathrm{P}[\mathrm{Y_i}=1]}\right) = \mathbf{x}_i^\top \boldsymbol{\beta}$$

Once $\boldsymbol{\beta}$ is estimated by $\widehat{\boldsymbol{\beta}}$, we have the predictions of posterior probabilities

$$\frac{1}{1+\mathrm{e}^{\mathbf{x}_i^\top \widehat{\boldsymbol{\beta}}}} \qquad \text{and} \qquad \frac{\mathrm{e}^{\mathbf{x}_i^\top \widehat{\boldsymbol{\beta}}}}{1+\mathrm{e}^{\mathbf{x}_i^\top \widehat{\boldsymbol{\beta}}}}$$

The connection to the LDA

In the linear discriminant analysis with two categories, the expressions for the posterior probabilities are

$$\frac{\pi_0 f_0(\mathbf{x})}{\pi_0 f_0(\mathbf{x}) + \pi_1 f_1(\mathbf{x})} \quad \text{and} \quad \frac{\pi_1 f_1(\mathbf{x})}{\pi_0 f_0(\mathbf{x}) + \pi_1 f_1(\mathbf{x})}$$

or also

$$\frac{1}{1 + \frac{\pi_1 f_1(\mathbf{x})}{\pi_0 f_0(\mathbf{x})}} \quad \text{and} \quad \frac{\frac{\pi_1 f_1(\mathbf{x})}{\pi_0 f_0(\mathbf{x})}}{1 + \frac{\pi_1 f_1(\mathbf{x})}{\pi_0 f_0(\mathbf{x})}}$$

If both f_0 and f_1 are the densities of multivariate normal distribution *with the same* $\mathbf{\Sigma}$, then the above posterior probabilities are

$$\frac{1}{1 + e^{a+b^T x}} \quad \text{and} \quad = \frac{e^{a+b^T x}}{1 + e^{a+b^T x}}$$

That indicates that both LDA and logistic regression estimate the same quantities - although they do not yield necessarily the same results. as the estimators are different)

LDA as regression

Two classes: if we code them by -1 and 1, and fit the linear regression of classes on classifiers (features),

the level zero set of the fitted hyperplane

is the LDA boundary (with equal prior probabilities and missclasification costs)

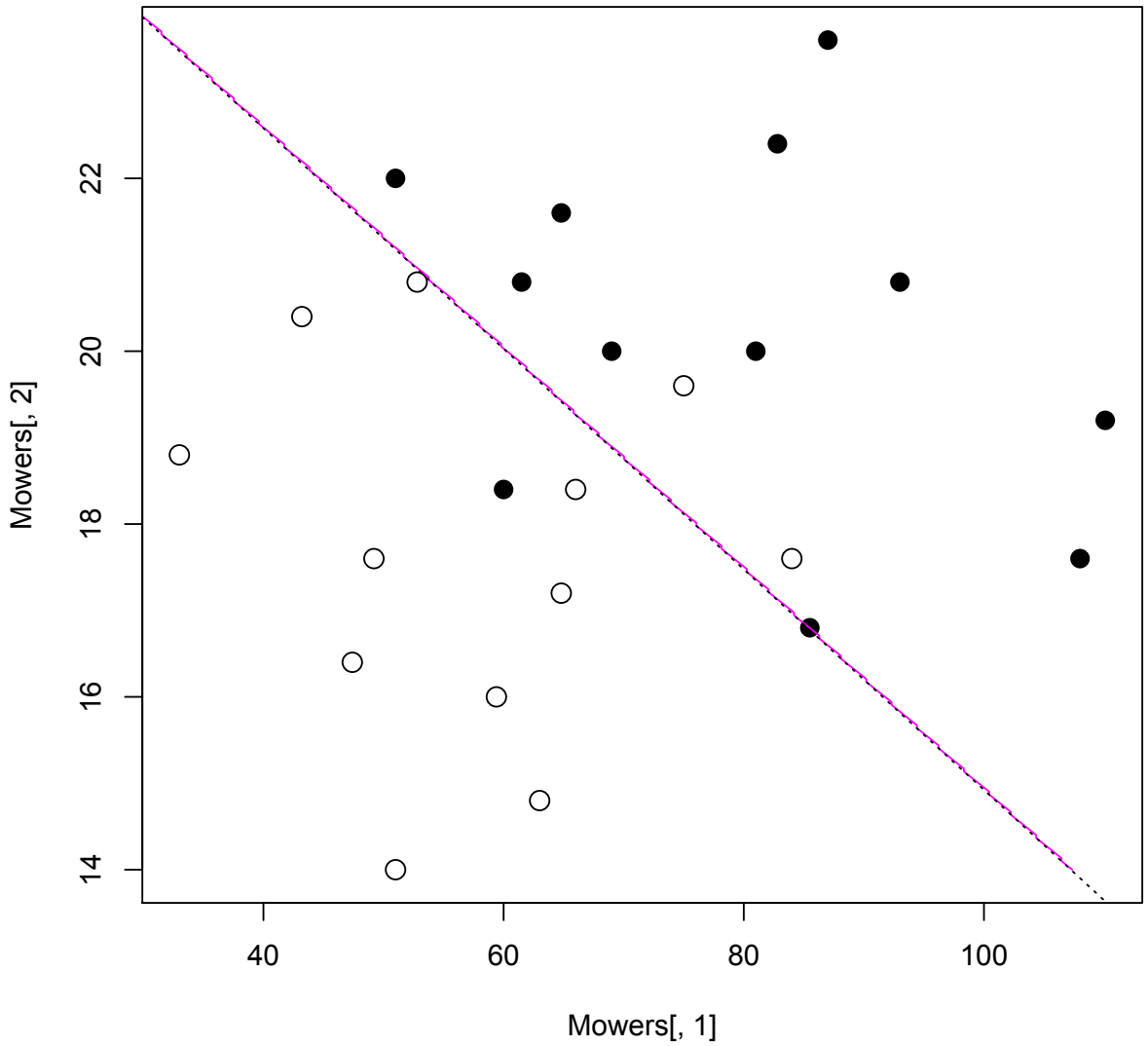
when there is the same number of points in both classes

(Even if not, the level zero set is parallel)

```
> lm(2*riding-1~income+lot,data=mowers)
Coefficients:
(Intercept)       income          lot
   -5.47100      0.02522      0.19761

> plot(mowers[,1],mowers[,2],pch=15*mowers[,3]+1)
> abline(5.471/0.19761,-0.02522/0.19761)
```

The only problem with such a regression: not too handy for predicting probabilities...

LDA as regression for mobile mowers



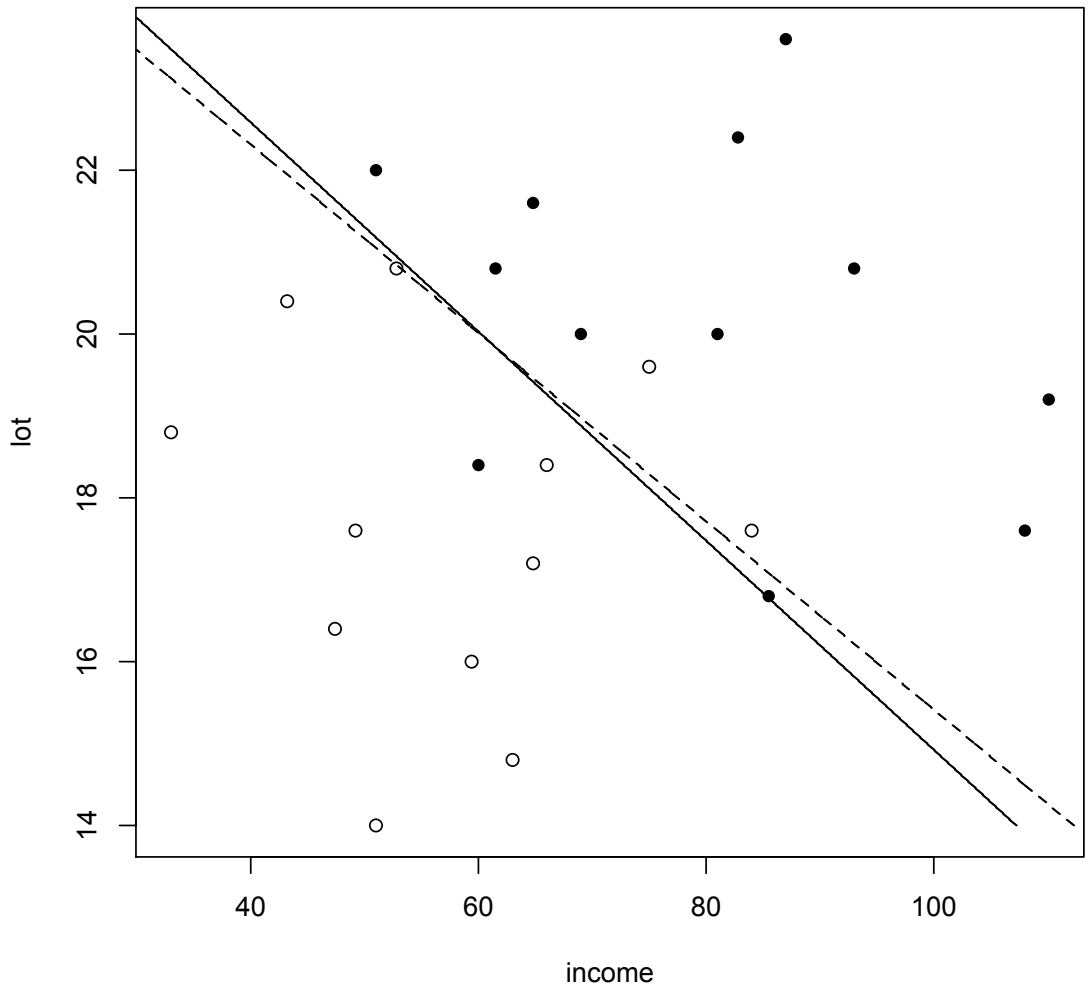
So what is the difference?

The difference is in the method: logistic regression maximizes the binomial likelihood conditionally on regressors, while LDA utilizes the covariance information in them. The results are often pretty much the same; the logistic regression is reported to be about 30% less efficient if the LDA normality assumptions hold.

On the other hand, logistic regression is more flexible, in particular with respect to incorporating qualitative classifiers, etc.

LDA and logistic regression: mobile mowers

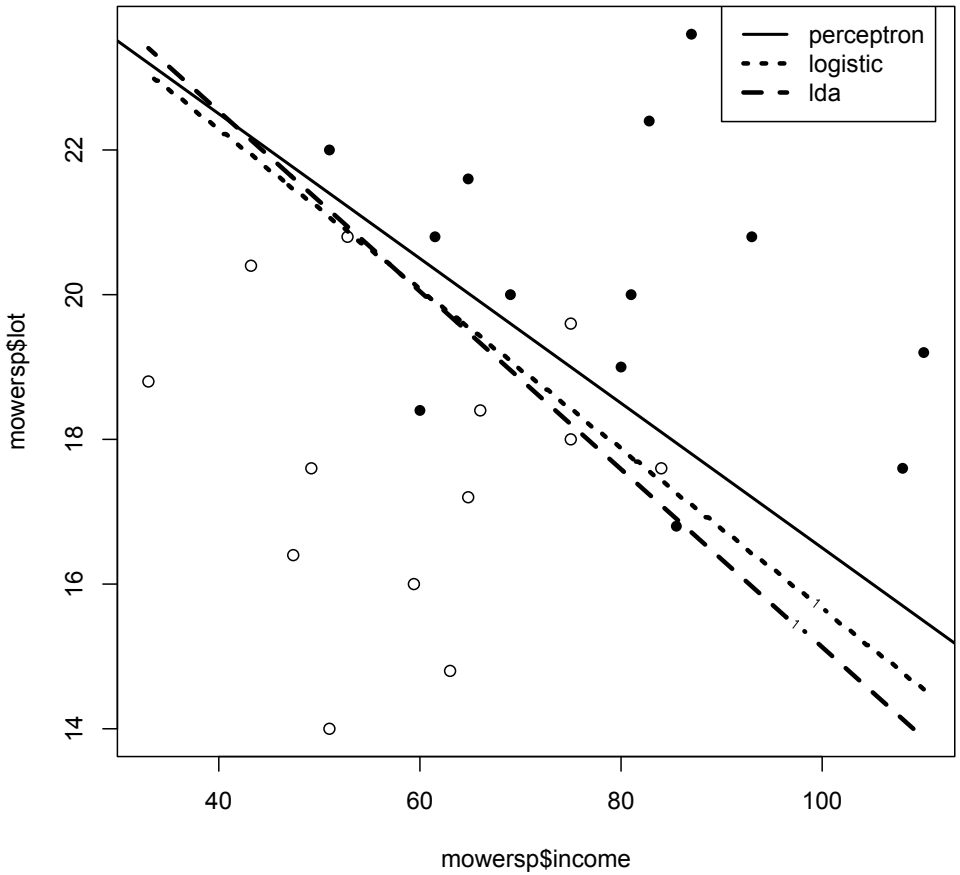
```
> mowlda=lda(riding~income+lot,data=Mowers)
> mowlog=glm(riding~income+lot,data=Mowers,family=binomial)
```



None of these has to be a "perceptron", however

Linear separation with minimal apparent error rate (*perceptron*) may be still different from both LDA and logistic regression (see augmented data below, 13 points of each category now)

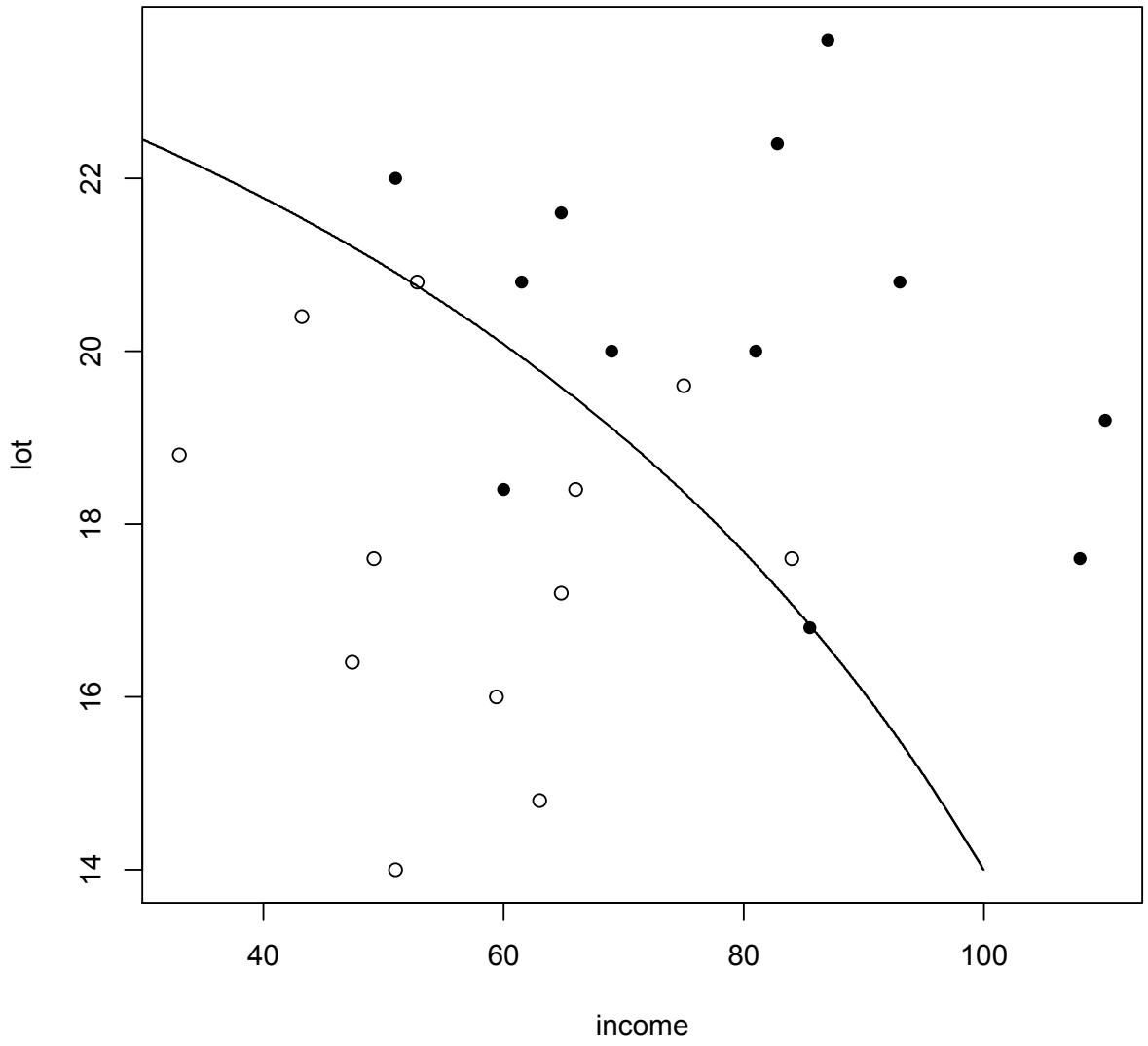
```
> mowersp <- rbind(Mowers,c(80,19,1),c(75,18,0))
```



Mobile mowers: add interaction?

```
> mowlog=glm(riding~income+lot+I(income*lot),data=Mowers,
+ family=binomial)
> summary(mowlog)
...
Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)     -37.433891  41.179585  -0.909    0.363
income            0.280810   0.574596   0.489    0.625
lot               1.560129   2.085627   0.748    0.454
I(income * lot)  -0.008915   0.029744  -0.300    0.764
```

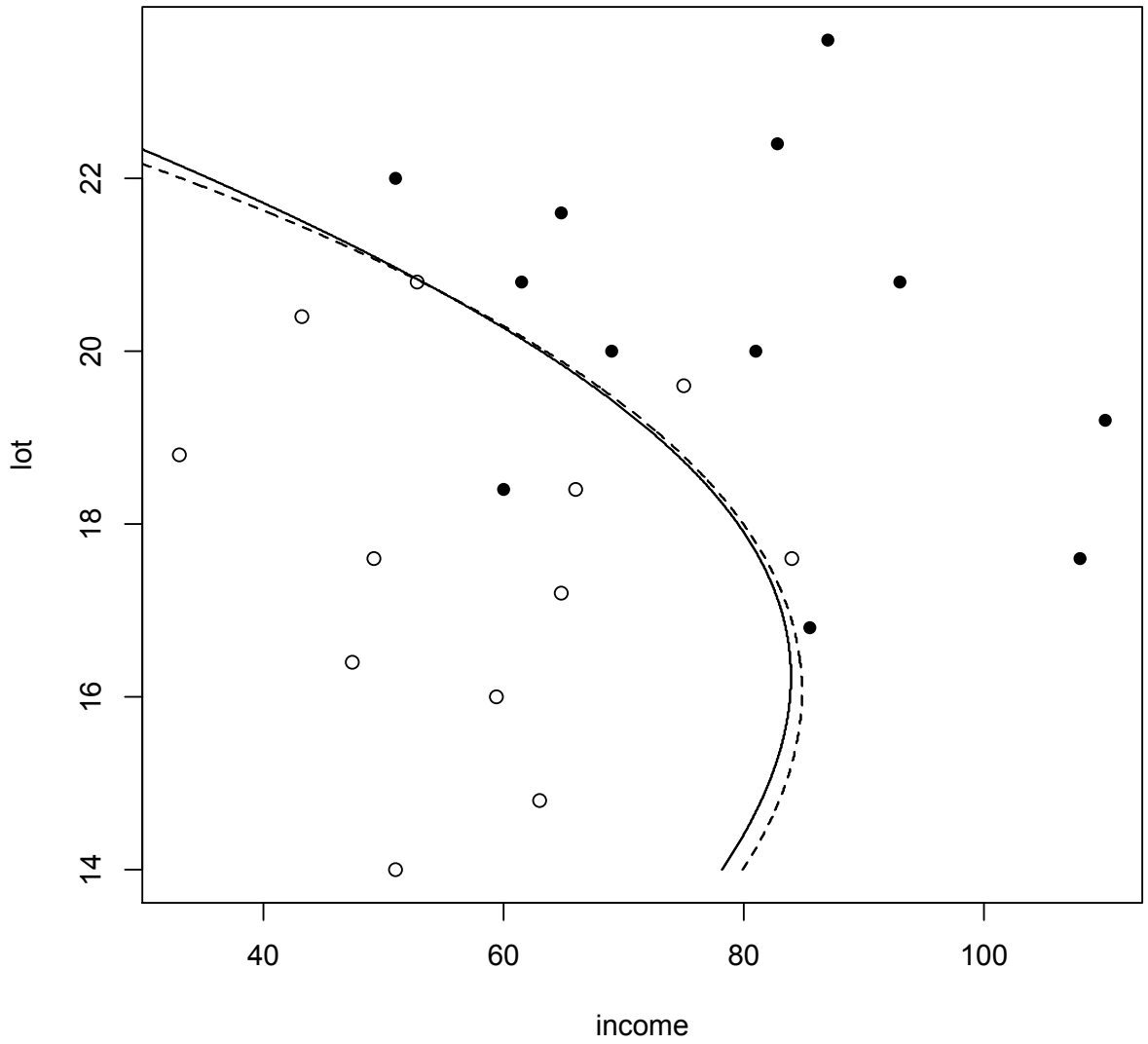
Bilinear logistic regression



Mobile mowers: add all quadratic terms?

```
> mowlog=glm(riding~income+lot+I(income*lot)+I(lot^2),
+ data=Mowers,family=binomial)
> mowlog=glm(riding~income+lot+I(income*lot)+I(lot^2)+I(income^2),
+ data=Mowers,family=binomial)
> summary(mowlog)
...
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)     20.7446485 84.6472477   0.245    0.806
income           0.3047331  0.8640822   0.353    0.724
lot             -4.7782138  8.0562174  -0.593    0.553
I(income * lot) -0.0061966  0.0336469  -0.184    0.854
I(income^2)     -0.0004715  0.0026086  -0.181    0.857
I(lot^2)         0.1632251  0.2075052   0.787    0.432
```

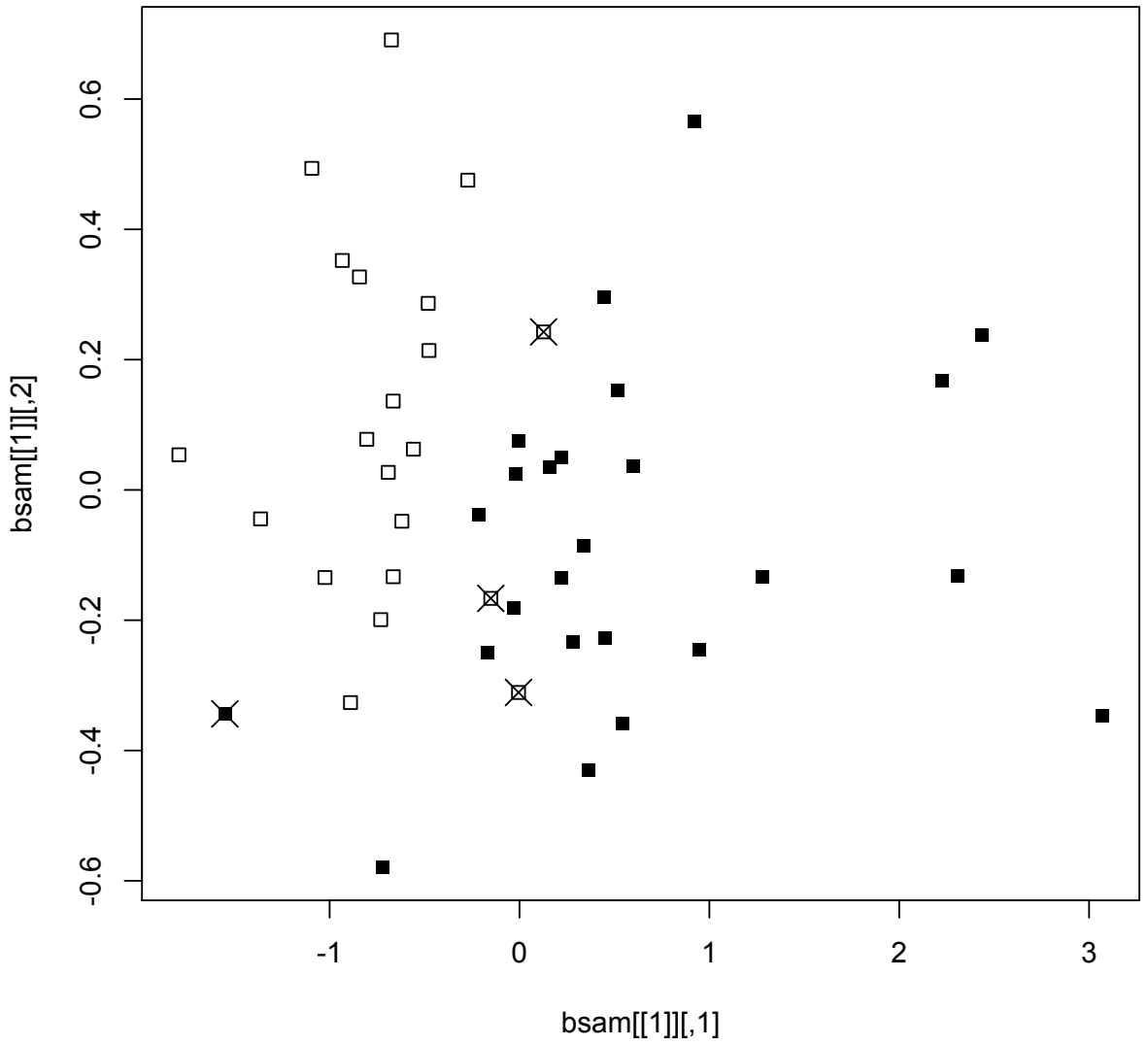
Go further?



Bank data: logistic regression

```
> library(MASS)
> bsam=sammon(dist(Bank[,1:4]))
> plot(bsam[[1]],pch=15*Bank$k)
> banlog=glm(k~.,data=Bank,family=binomial)
> wrong=(Bank$k!=(predict(banlog,type='response')>0.5))
> points(bsam[[1]][wrong,],pch=4,cex=2)
```

The result



However

```
> summary(banlog)
...
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    -5.320      2.366  -2.248  0.02459 *
v1              7.138      6.002   1.189  0.23433
v2             -3.703     13.670  -0.271  0.78647
v3              3.415      1.204   2.837  0.00455 **
v4             -2.968      3.065  -0.968  0.33286
```

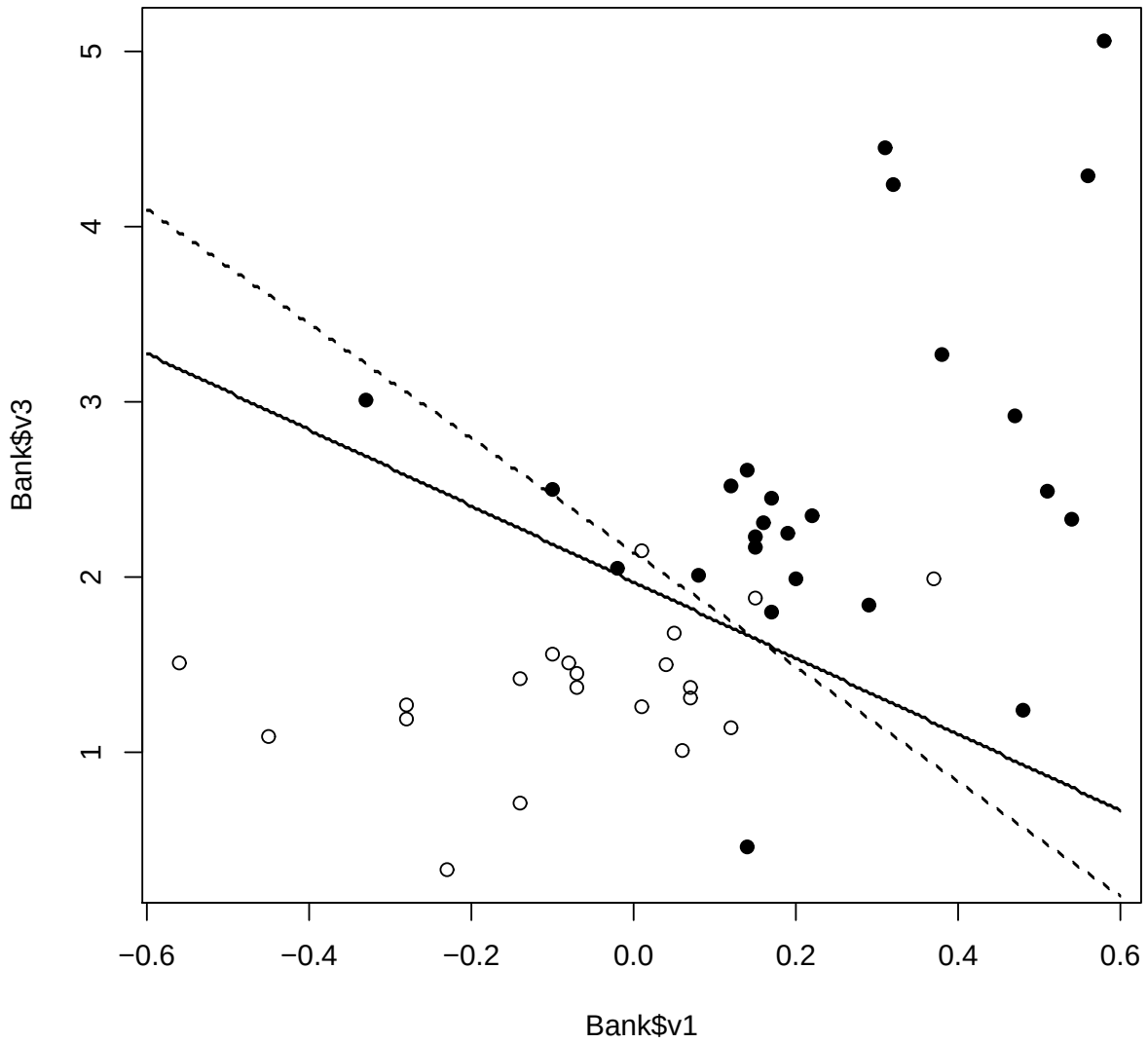
Simplify by dropping classifiers?

```
> banlog=glm(k~v3,data=Bank,family=binomial)
> table(Bank$k,(predict(banlog,type='response')>0.5))
    FALSE TRUE
  0    18    3
  1     2   23
> banlog=glm(k~v3+v1,data=Bank,family=binomial)
> table(Bank$k,(predict(banlog,type='response')>0.5))
    FALSE TRUE
  0    18    3
  1     1   24
> summary(banlog)
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -5.940      1.985  -2.992  0.00277 **
v3             3.019      1.002   3.013  0.00259 **
v1             6.556      2.905   2.257  0.02402 *
> attach(Bank)
> plot(v1,v3,pch=15*k+1)
```

Simplified prediction - and then revert to LDA?

```
> banlda=lda(k~v3+v1,data=Bank) % broken line
...
> table(k,predict(banlda)$class)
k    0  1
  0 18  3
  1  3 22
```

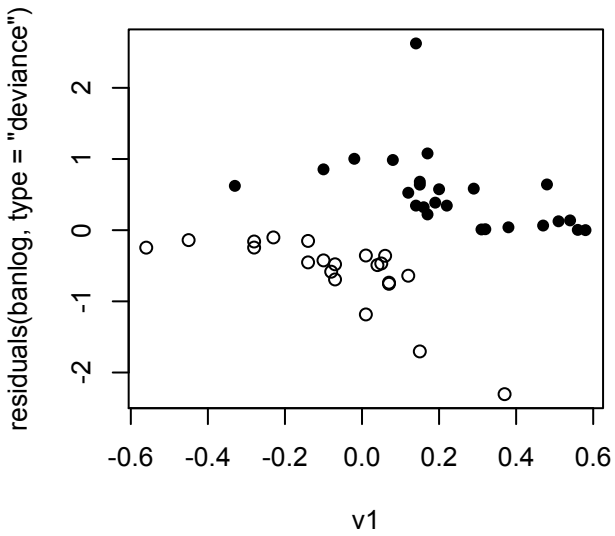
Bank data in two dimensions now

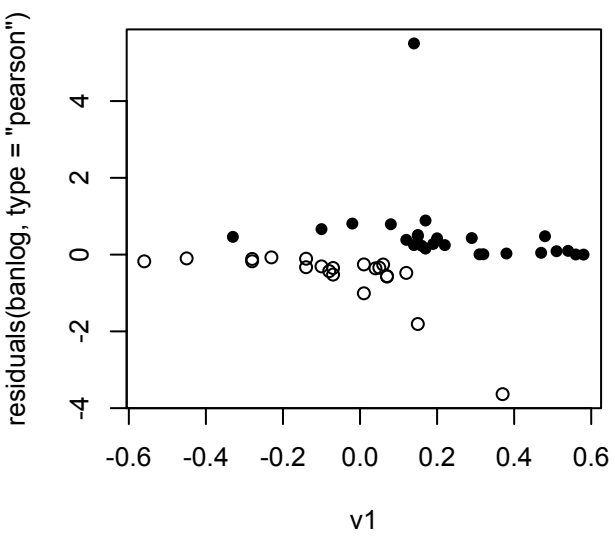


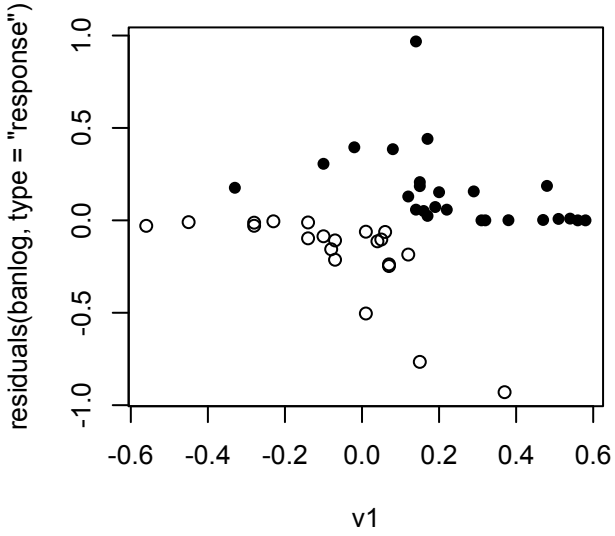
A look at residuals: code

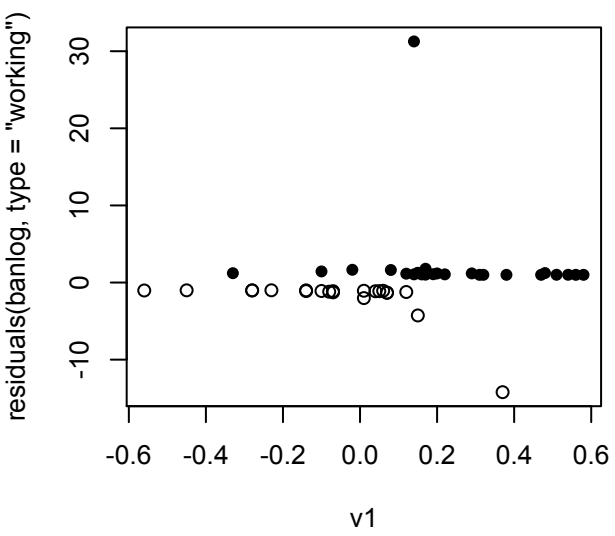
```
> par(mfrow=c(2,2))
> plot(v1,residuals(banlog,type='deviance'),pch=15*k+1)
> plot(v1,residuals(banlog,type='pearson'),pch=15*k+1)
> plot(v1,residuals(banlog,type='response'),pch=15*k+1)
> plot(v1,residuals(banlog,type='working'),pch=15*k+1)
```

A look at residuals: pictures



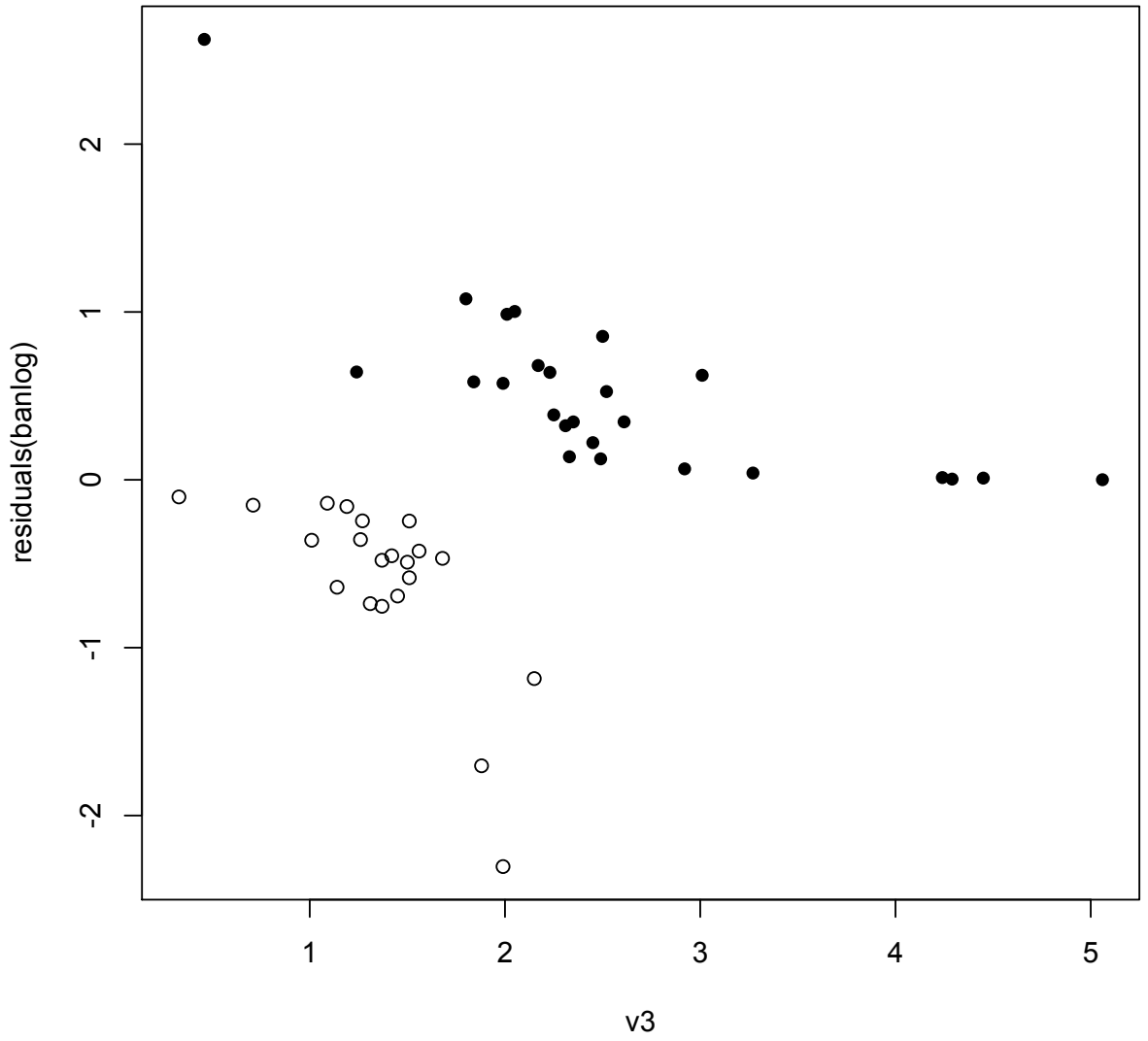






Further look at residuals

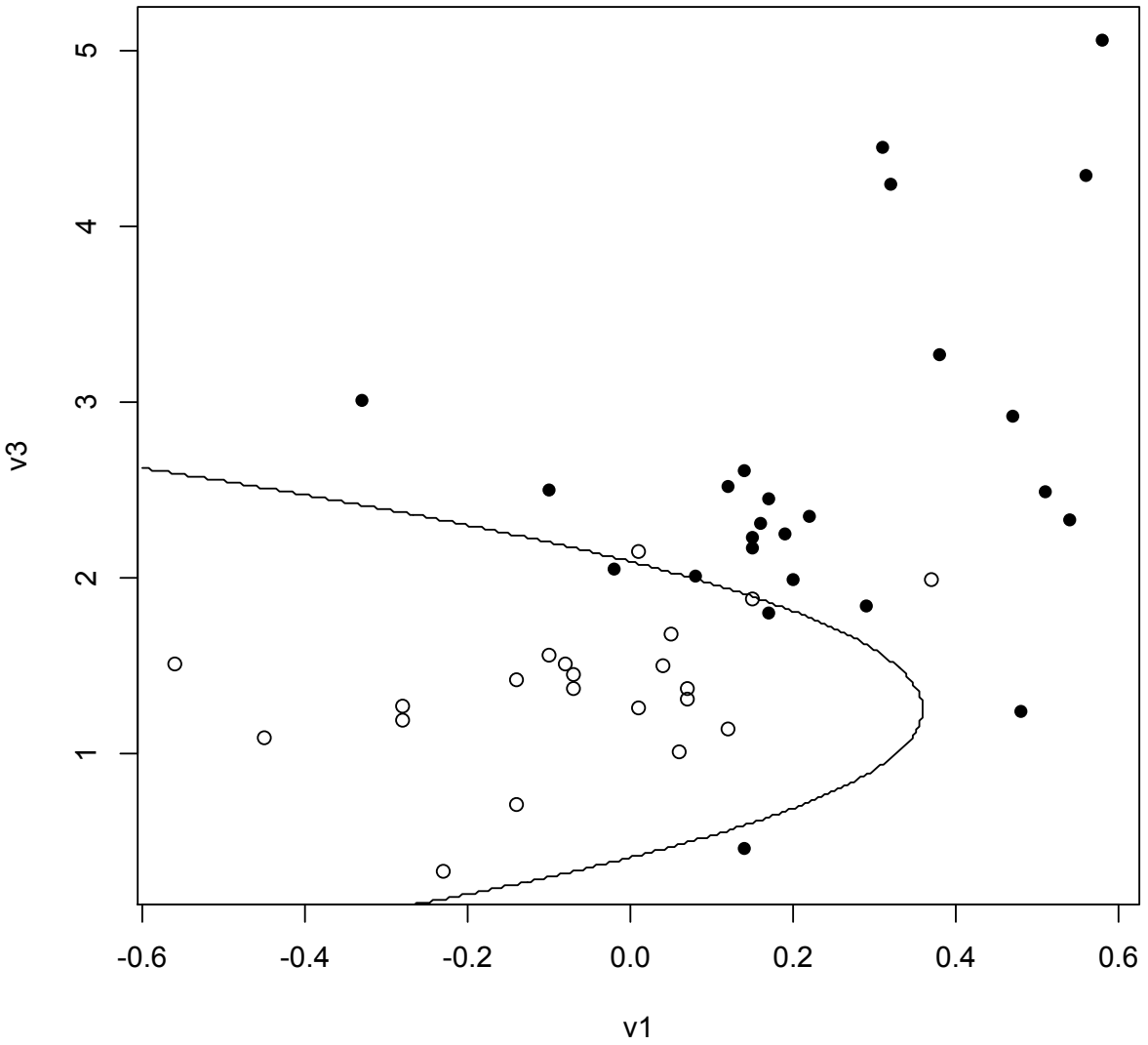
```
> plot(v3,residuals(banlog),pch=15*k+1)
```



Quadratic term in v3?

```
> banlog=glm(k~poly(v3,2)+v1,data=Bank,family=binomial)
Warning message:
fitted probabilities numerically 0 or 1 occurred in: glm.fit(x = X,
...
> table(k,predict(banlog)>0.5)
k   FALSE TRUE
  0    19    2
  1     3   22
```

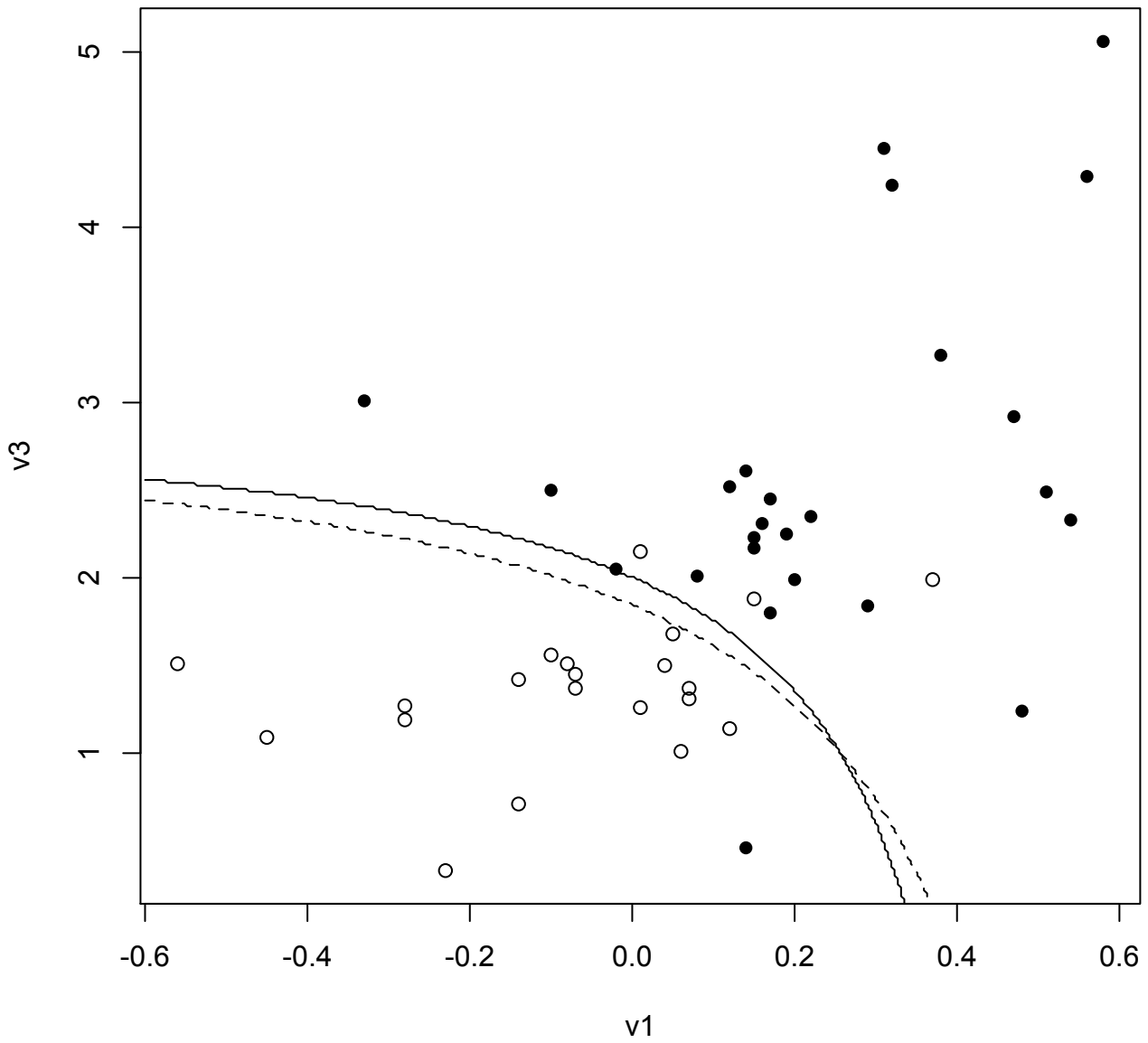
Odd?



Interaction?

```
> banlog=glm(k~v1+v3+I(v1*v3),data=Bank,family=binomial)
> table(k,predict(banlog)>0.5)
k   FALSE TRUE
  0    18    3
  1     2   23
>summary(banlog)
...
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -7.234      2.509  -2.883  0.00393 **
v1            21.141      9.201   2.298  0.02158 *
v3             3.627      1.286   2.820  0.00480 **
I(v1 * v3)    -6.953      3.476  -2.000  0.04549 *
```

The result (broken: LDA in a same way!)



More than two classes now

```
> library(nnet)
> iris.log=multinom(Species~.,data=iris)
# weights:  18 (10 variable)
initial  value 164.791843
iter  10 value 16.177348
iter  20 value 7.111438
...
iter 100 value 5.949867
final  value 5.949867
stopped after 100 iterations
```

First, we could add a bit more iterations

```
> iris.log=multinom(Species~.,data=iris,maxit=300)
# weights:  18 (10 variable)
initial  value 164.791843
iter  10 value 16.177348
iter  20 value 7.111438
...
iter 180 value 5.949393
final  value 5.949363
converged
```

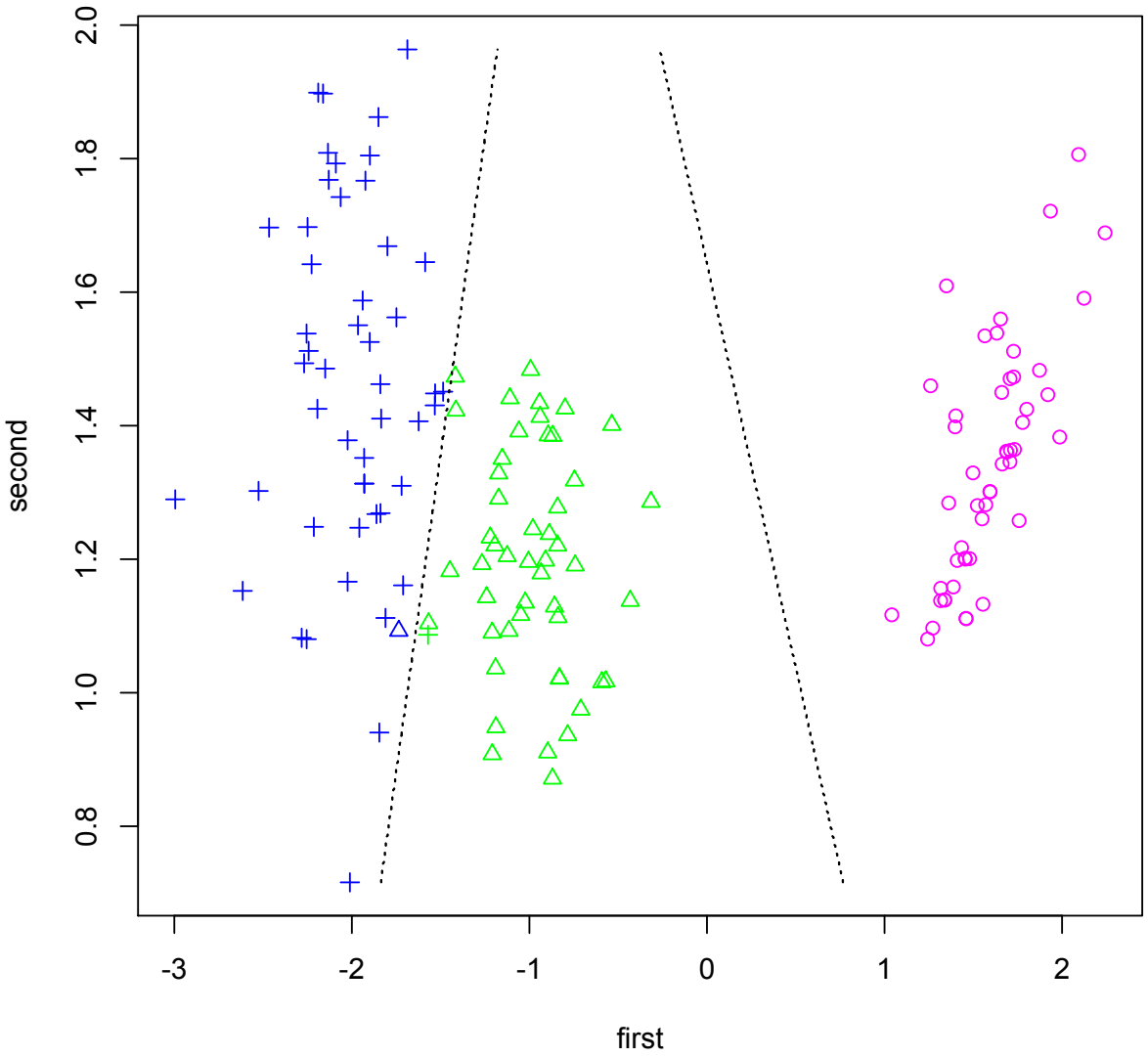
And what can be done with it

```
> iris.log
Call:
multinom(formula = Species ~ ., data = iris, maxit = 300)

Coefficients:
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
versicolor    18.40821    -6.082250   -9.396625     16.17037   -2.058115
virginica    -24.23006    -8.547304  -16.077164     25.59963   16.227474

Residual Deviance: 11.89873
AIC: 31.89873
> predict(iris.log)
  [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa
...
 [41] setosa     setosa     setosa     setosa     setosa     setosa     setosa
 [51] versicolor versicolor versicolor versicolor versicolor versicolor versicolor
...
 [91] versicolor versicolor versicolor versicolor versicolor versicolor versicolor
[101] virginica  virginica  virginica  virginica  virginica  virginica  virginica
...
[141] virginica  virginica  virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

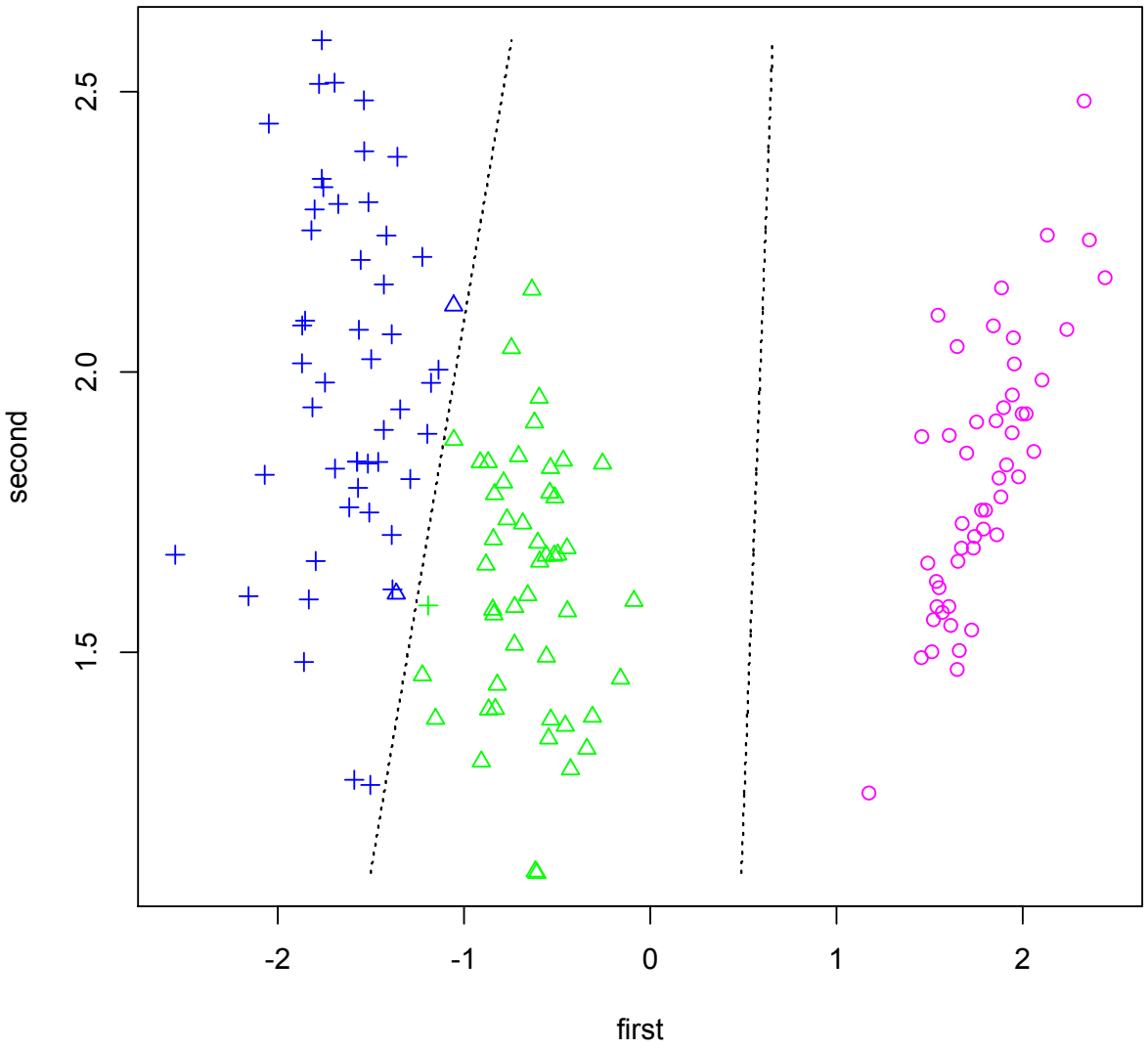
Just for comparison

```
> table(predict(iris.log),iris$Species)

            setosa versicolor virginica
  setosa        50          0         0
  versicolor     0         49         1
  virginica      0          1        49

> iris.lda = lda(Species~.,data=iris)
> table(predict(iris.lda)$class,iris$Species)

            setosa versicolor virginica
  setosa        50          0         0
  versicolor     0         48         1
  virginica      0          2        49
```

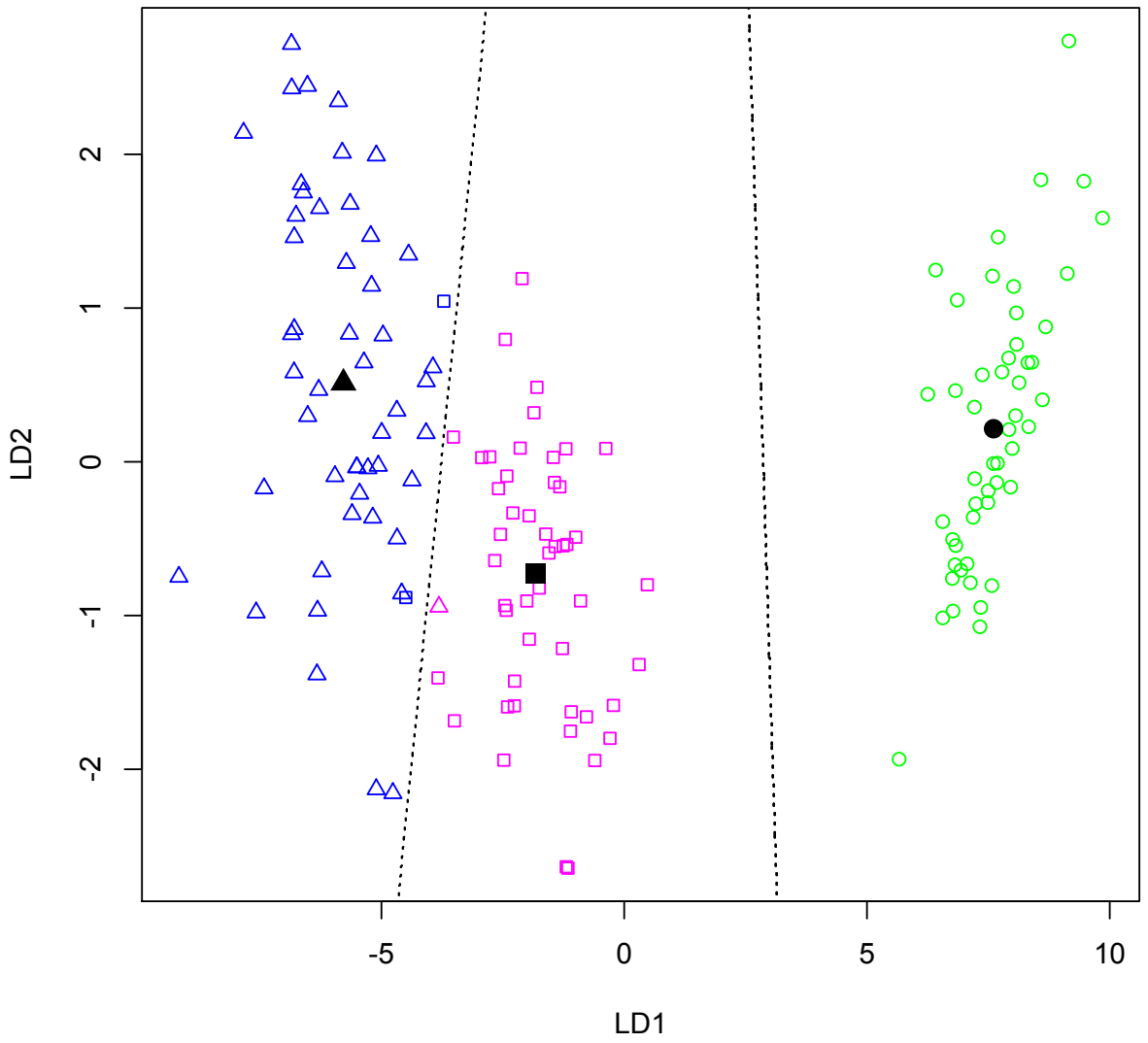
"Exact plotting" also possible here



For comparison: LDA (arbitrarily matched)



For comparison: LDA (linear discriminants)



Some (semi)final remarks

Not only logistic regression, but also LDA can be viewed as a regression method for classification.

Once the normality assumption for the classifiers does not bother us too much, our possibilities are expanded: not only we can do some model selection, but we can also consider transformed classifiers.

However, this entails various pitfalls. On one hand, would be nice to have the models flexible, so that they can adapt to various nonlinear boundaries of predicted categories. On the other hand, this bring a danger of overfitting. Would be nice to manage all that somehow

And now: yet something more

Logistic regression classifying into two classes: estimation

Likelihood: the product of terms depending on y_i

$$\frac{1}{1+e^{x_i^T\beta}} \quad \text{if } y_i = 0 \qquad \text{or} \qquad \frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}} \quad \text{if } y_i = 1$$

which is the same as

$$\frac{1}{1+e^{x_i^T\beta}} \quad \text{if } y_i = 0 \qquad \text{or} \qquad \frac{1}{1+e^{-x_i^T\beta}} \quad \text{if } y_i = 1$$

If we change coding from 0 to -1 (and keeping 1) it is always

$$\frac{1}{1+e^{-y_i x_i^T\beta}}$$

The negative loglikehood then (to be minimized in β) is

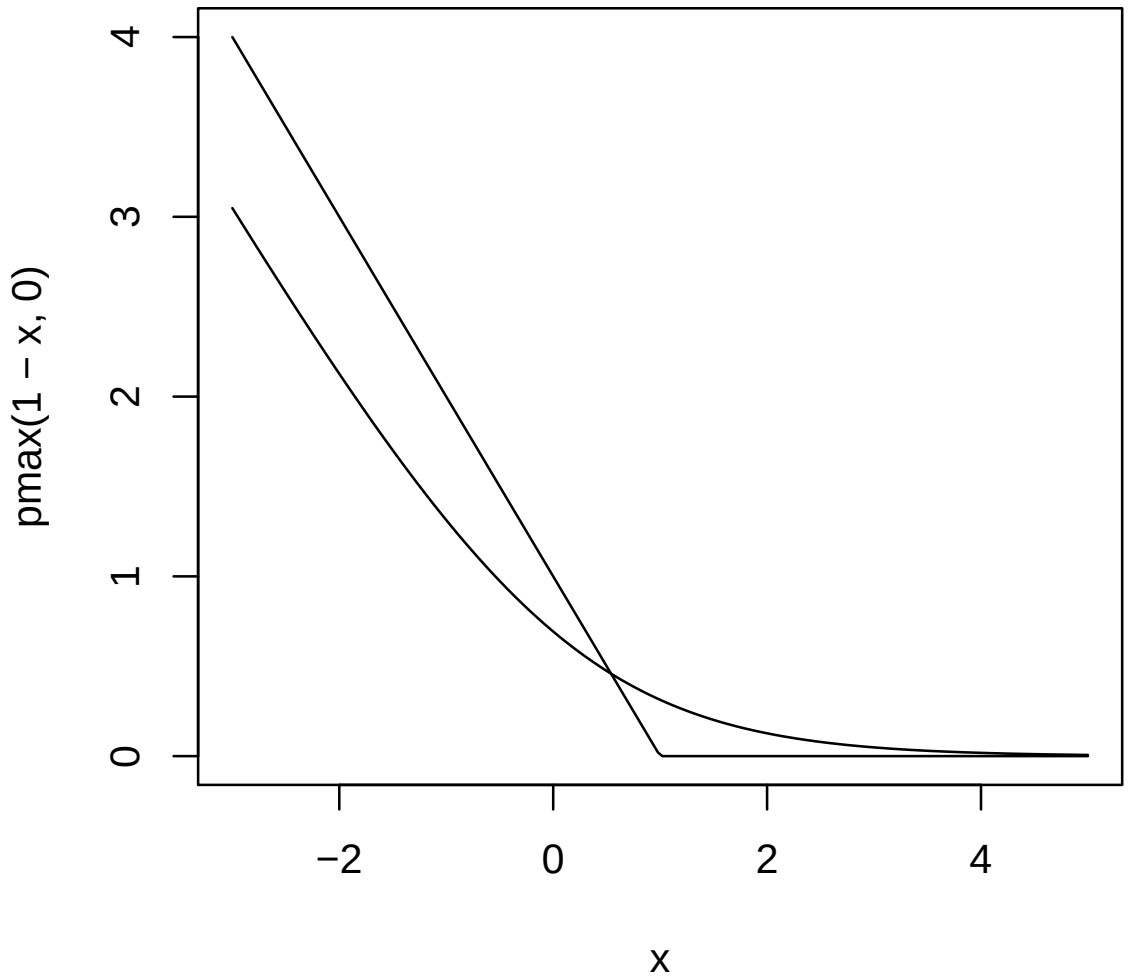
$$\sum_{i=1}^{n} \log\left(1+e^{-y_i x_i^T\beta}\right)$$

Compare to $$\sum_{i=1}^{n} (1-y_i x_i^T\beta)_+ = \max\{0, 1-y_i x_i^T\beta\}$$

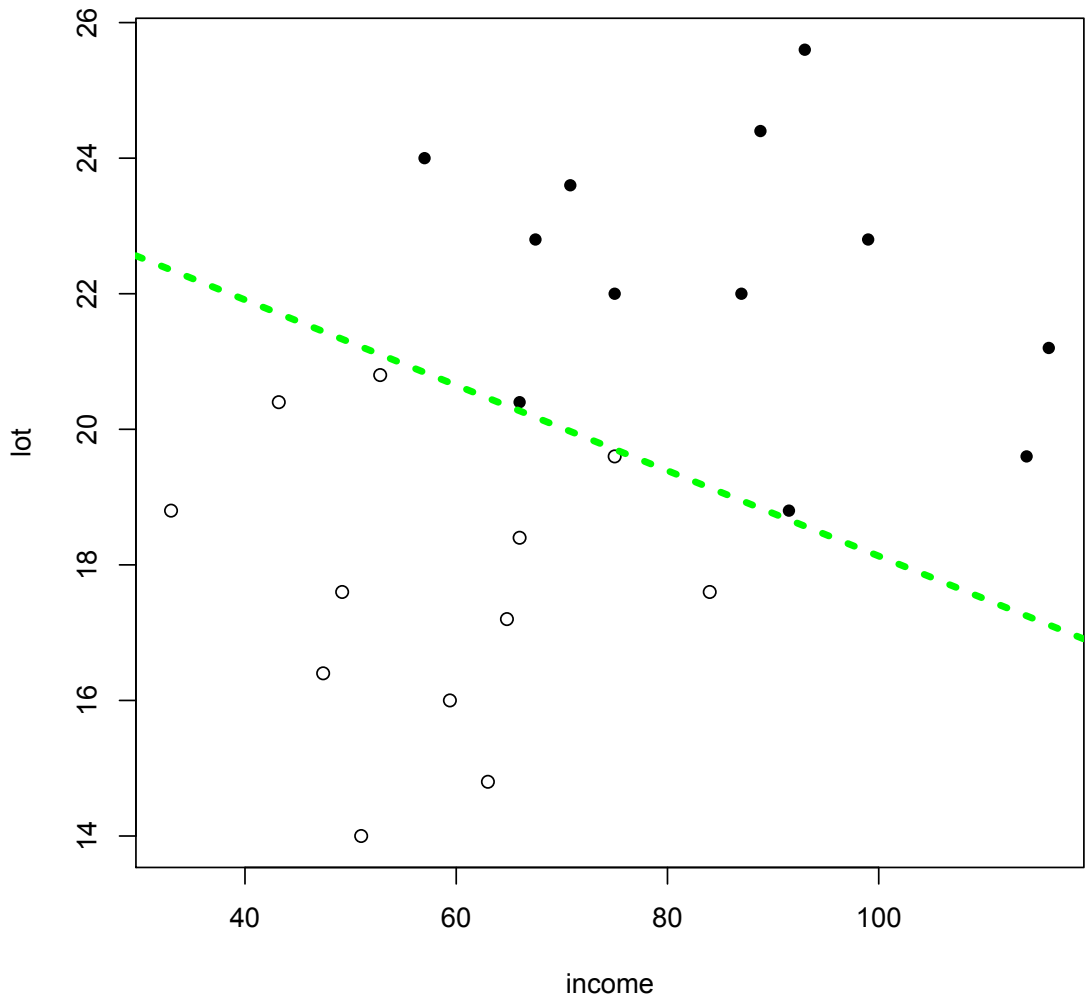
The latter corresponds to so-called **support vector machine**

Digression: support vector machine

A younger brother of logistic regression

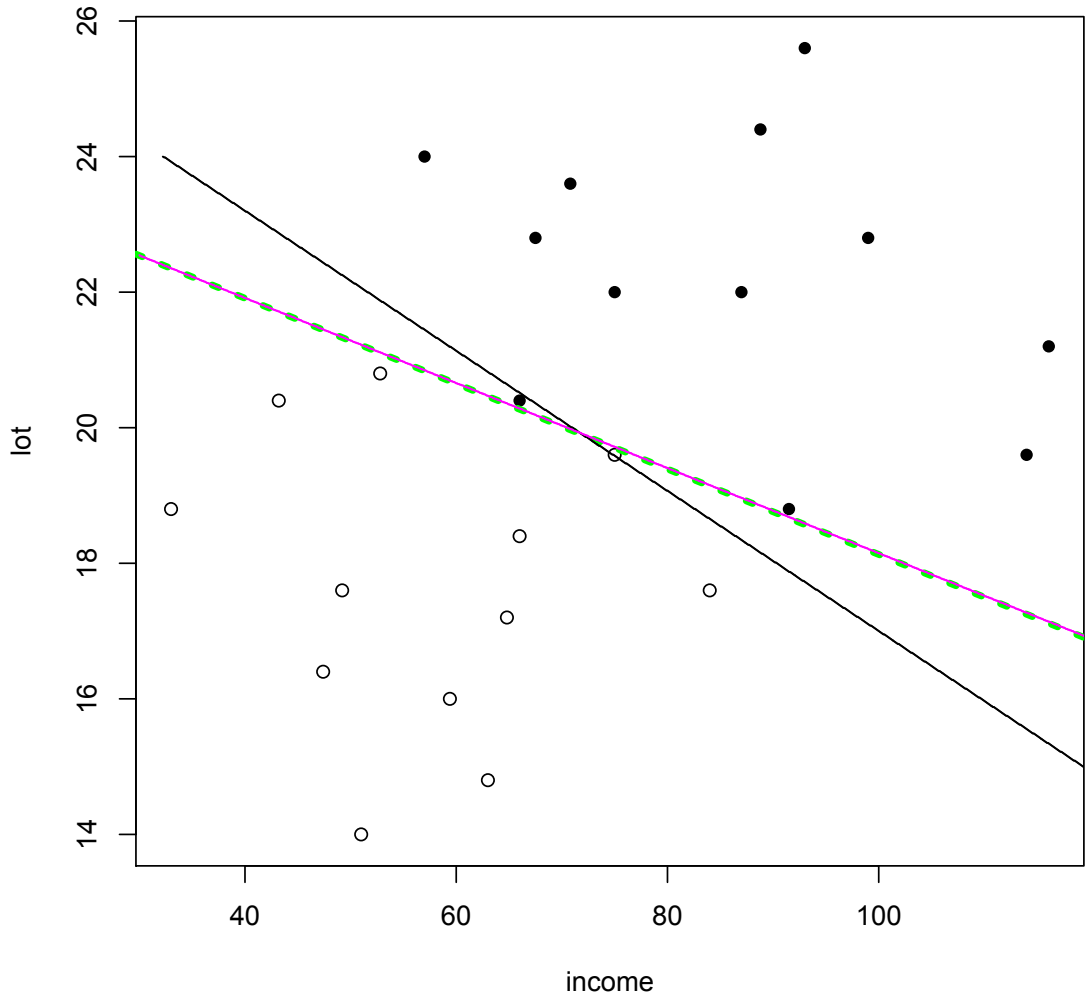


The obsession of engineering: separated data



This is not original, but slightly doctored riding mowers data; as can be seen, the clasess are now *(linearly) separated*.

The classification should be perfect then?

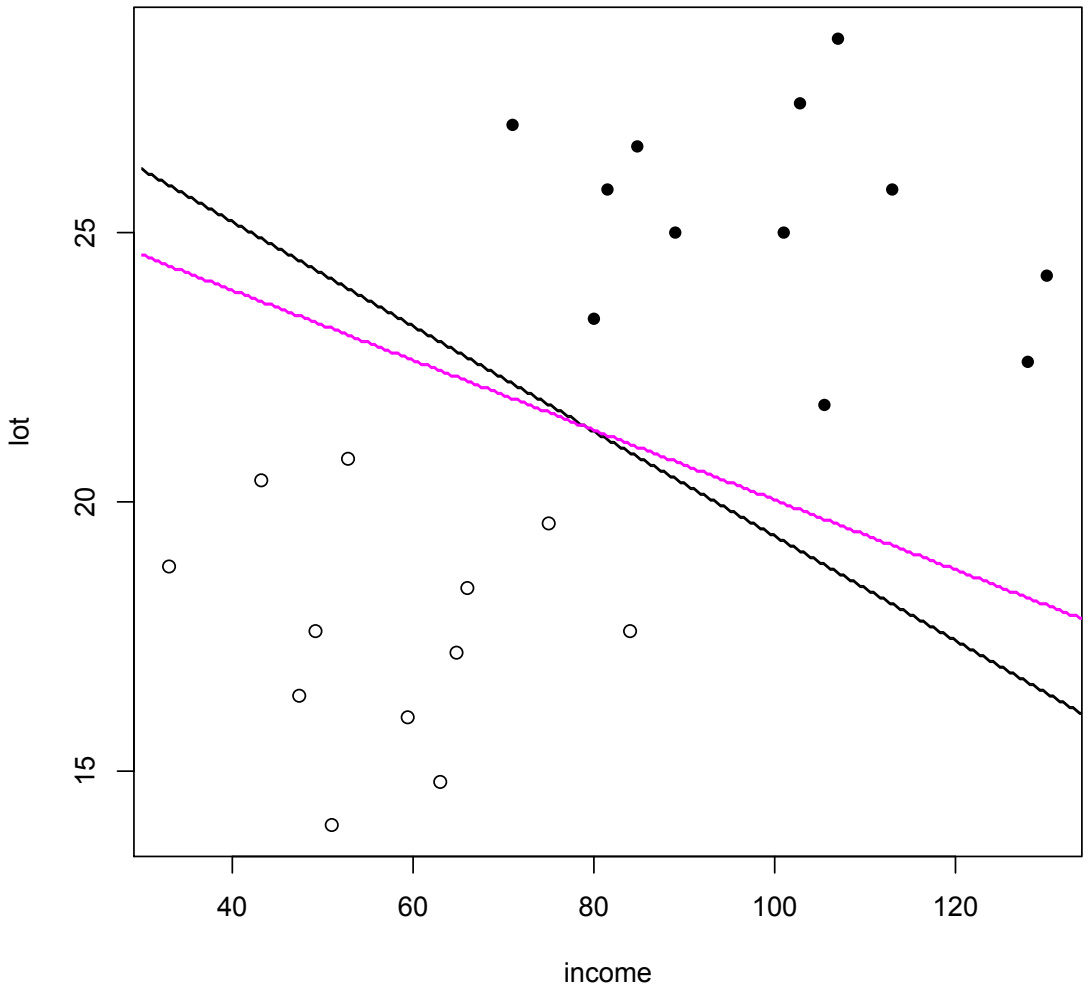


Actually, the LDA one may be not. The logistic regression may be better, but its problem is being numerically unstable for separated classes (so it is not clear what result it returns)

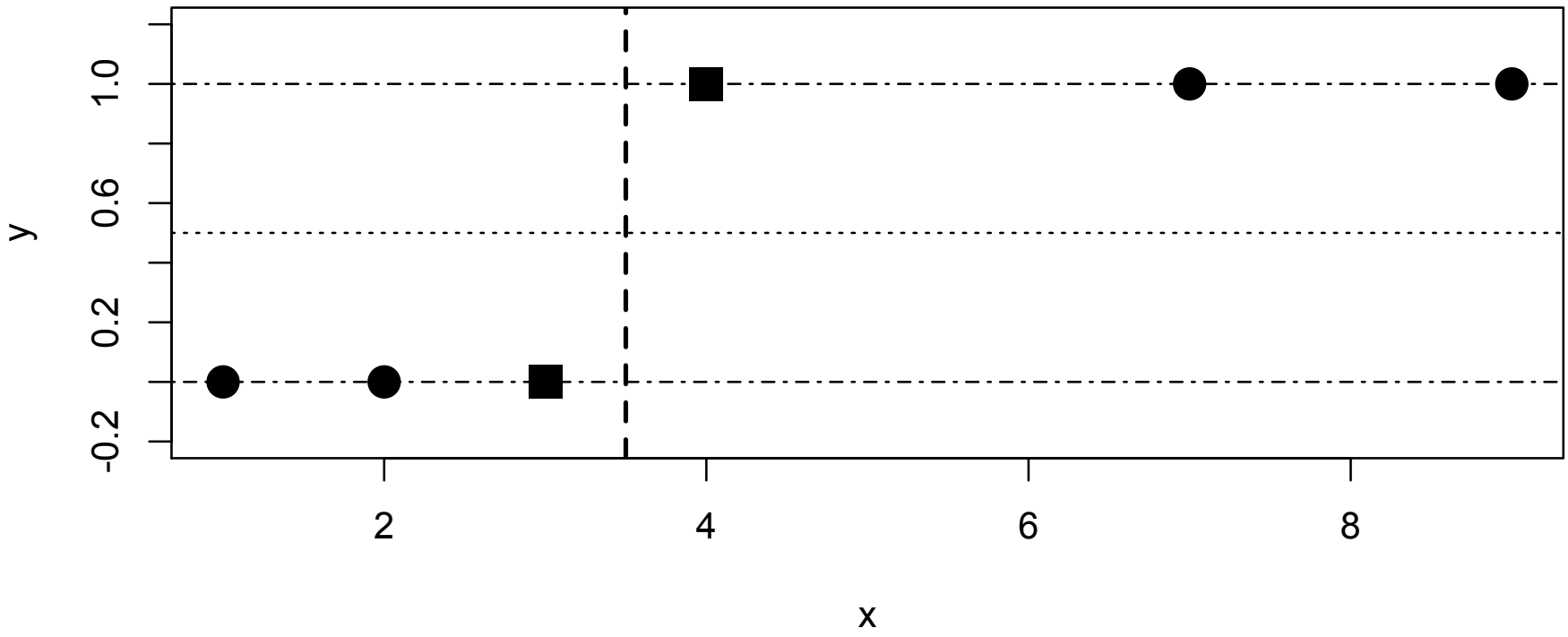
Indeed

```
> try.lda=lda(riding~income+lot,data=moo)
> table(predict(try.lda)$class,moo$riding)

    0  1
  0 11  1
  1  1 11

> try.log=glm(riding~income+lot,data=moo,family=binomial)
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
> table(predict(try.log)>0,moo$riding)

        0  1
  FALSE 12  0
  TRUE   0 12
```

Let us try even more separated data



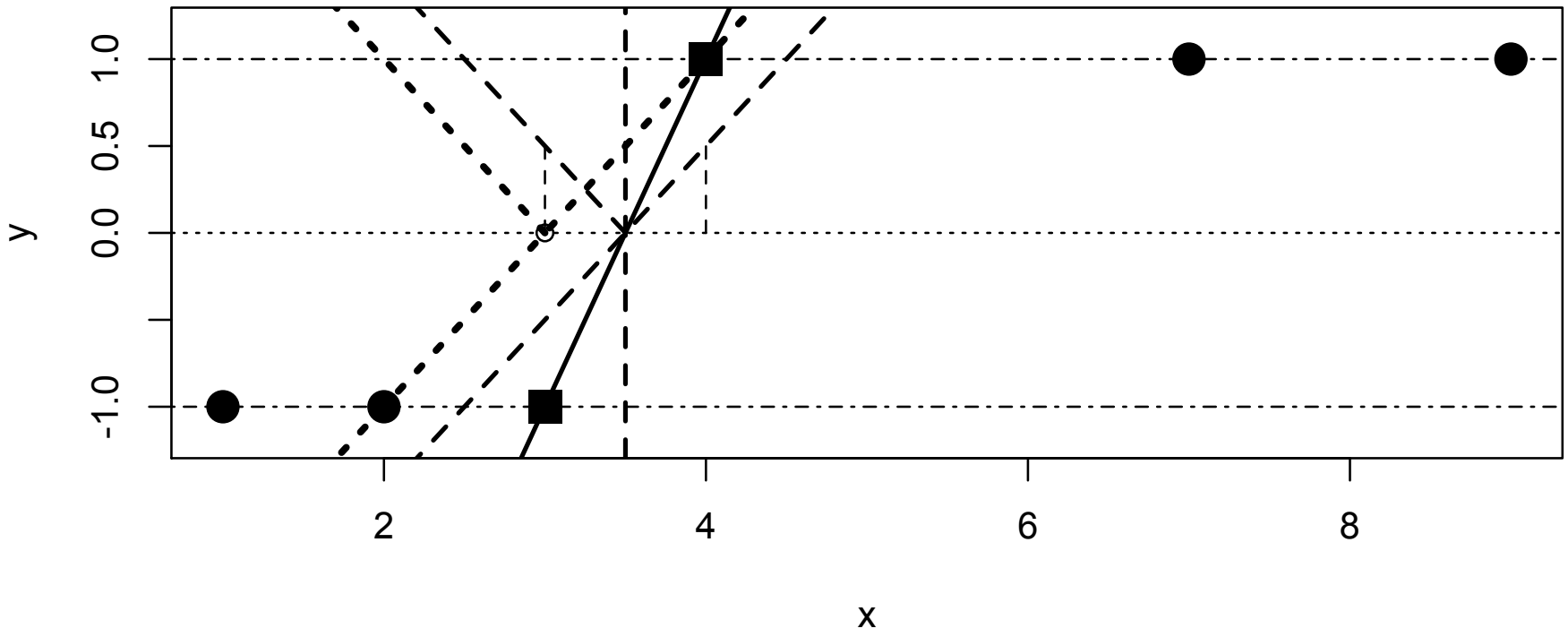
Maximum margin ("generalized portrait"), dim 1



If we are mainly thinking about configurations with separated points, it is possible to design a method that puts the divide right halfway between the groups

Handling that geometrically is tedious; it is better to do some fitting of the lines. When changing the coding to $y_i = -1$ and $y_i = 1$, the desired configuration is obtained by crossing the level $y = 0$ by the line that is the solution of

Maximum margin (dim 1) continued



$$B \to \max_{a,b}! \qquad y_i(a + bx_i) \geqslant B$$

The change of variables $\alpha = \frac{a}{B}$, $\beta = \frac{b}{B}$ makes it

$$|\beta| \to \min_{\alpha,\beta}! \qquad y_i(\alpha + \beta x_i) \geqslant 1$$

Maximum margin multidimensional

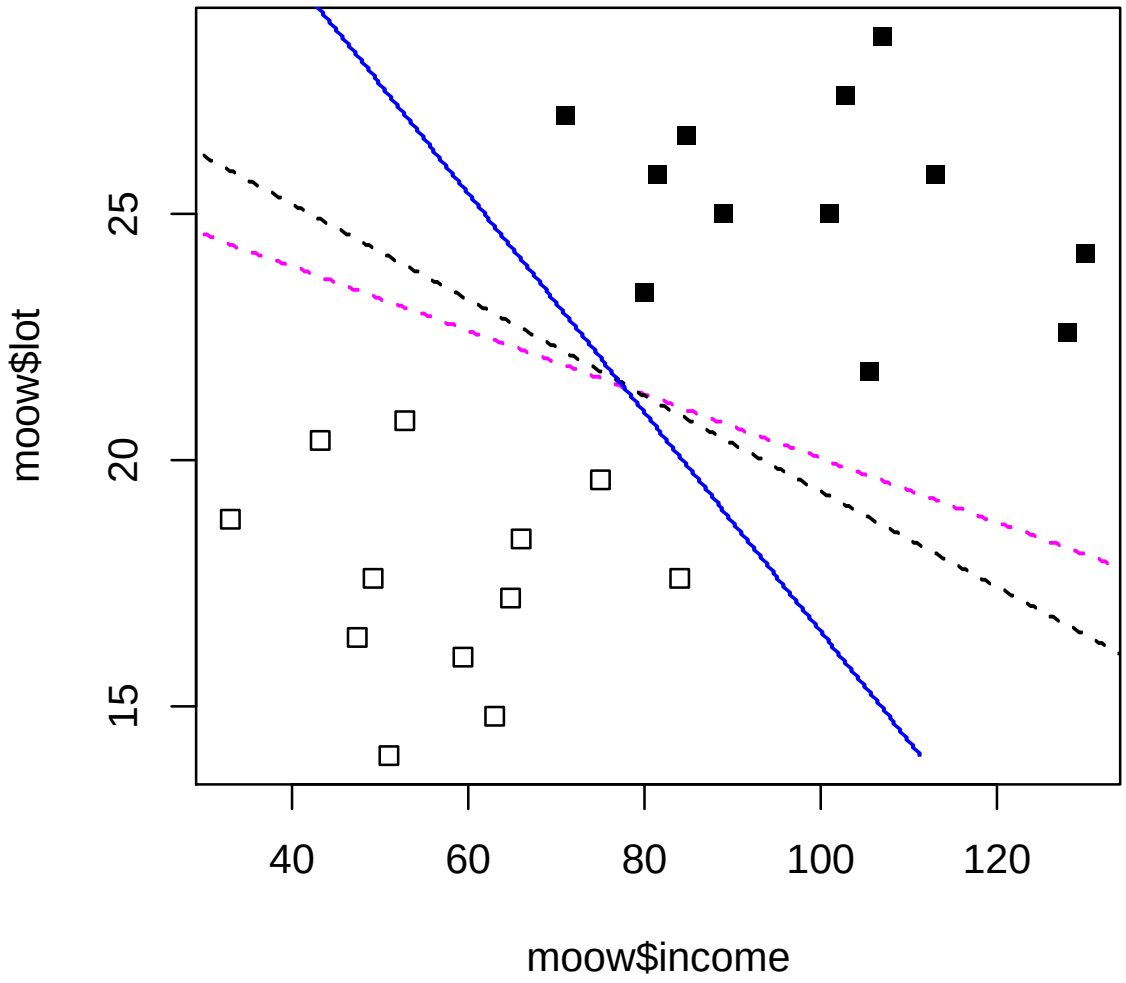
When x_i is multidimensional, it is $\mathbf{x}_i$; and $|\beta|$ also changes to $\boldsymbol{\beta}$

$$\|\beta\|^2 \leadsto \min_{\alpha,\beta}! \qquad y_i(\alpha + \mathbf{x}_i^T\beta) \geqslant 1 \qquad \text{(all } i\text{)}$$

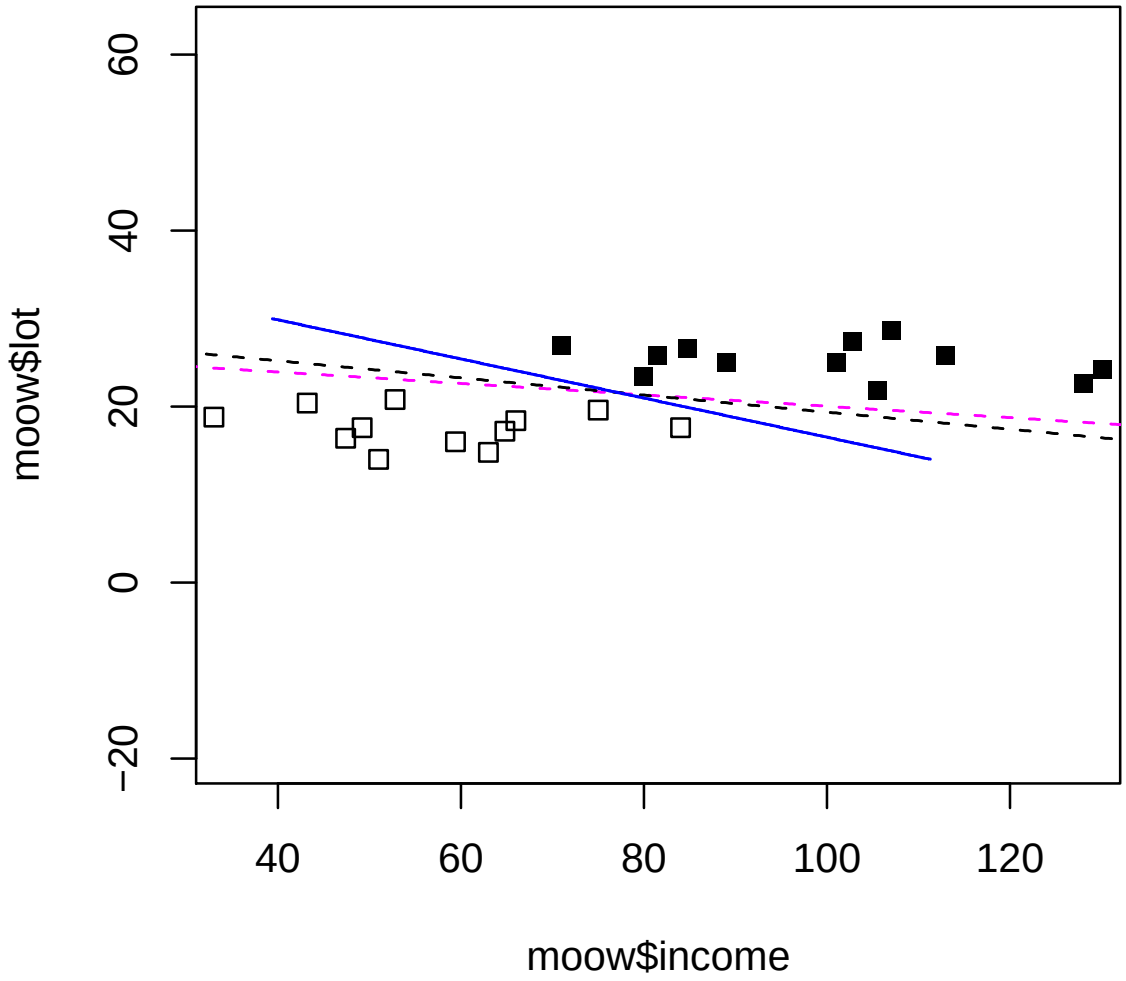
And it remains a quadratic programming problem once overlapping data configurations are accommodated (which leads already to an introduction of a tuning parameter), and then quadratic penalties to accomodate nonlinearities: kernels, etc.

The plot doesn't look like it...

(broken black: LDA; broken magenta: logistic regression)

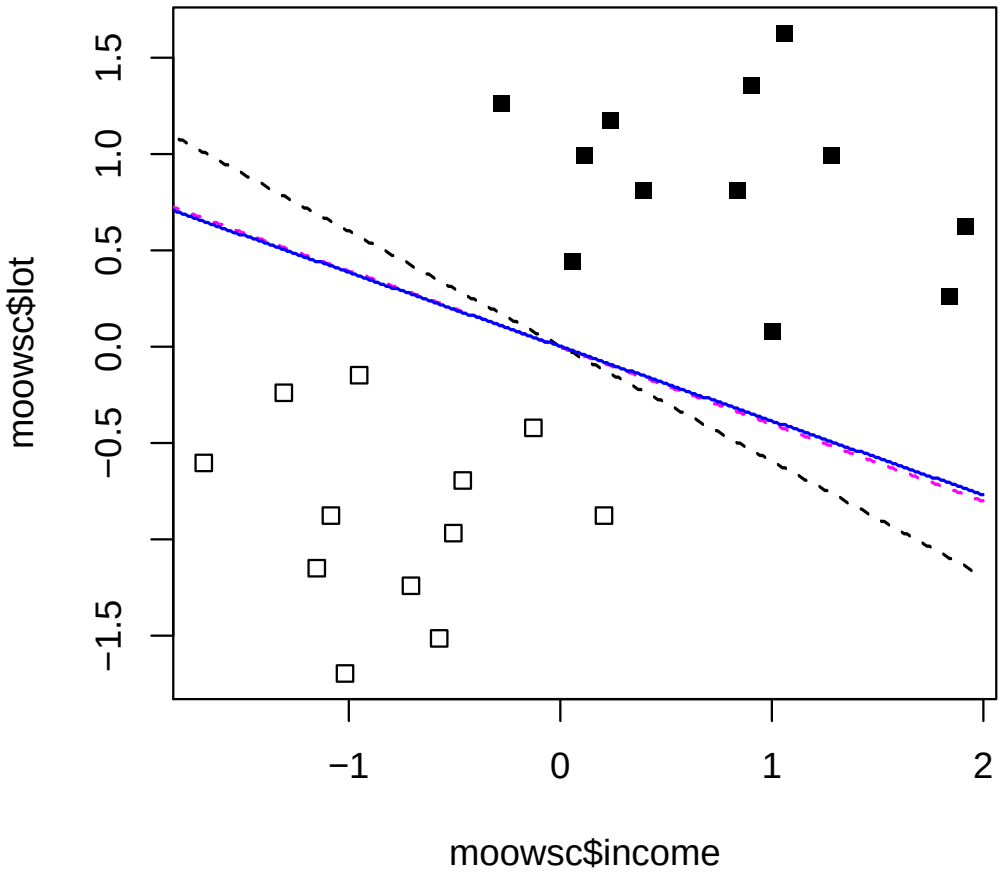


...because it's not properly scaled!



Three linear classifiers for scaled data

Maximum margin overplots logistic regression here - but this may be just a coincidence. There are similarities between the two, but in general they are different.



Scaling?

For simplicity, we consider methods only with the original classifiers, not with their transformations; then

- linear discriminant analysis does not need scaling
- logistic regression does not need scaling
- but "maximum margin method" usually does

That is one thing; another one is:

- what if the classes are not separated (and thus *overlap*)?

Then the original problem needs to be modified!

Regularized (linear) support vector machine

Modification: the original minimization problem

$$\|\boldsymbol{\beta}\|^2 \looparrowright \min_{\alpha,\boldsymbol{\beta}}! \qquad y_i(\alpha + \mathbf{x}_i^\top \boldsymbol{\beta}) \geqslant 1 \qquad (\text{all } i)$$

is modified by allowing "errors" e_i in the bounds - their sum is not to exceed the a priori chosen bound C (a tuning parameter!)

$$\|\boldsymbol{\beta}\|^2 \looparrowright \min_{\alpha,\boldsymbol{\beta}}! \qquad y_i(\alpha + \mathbf{x}_i^\top \boldsymbol{\beta}) \geqslant 1 - e_i \qquad (\text{all } i)$$

$$\text{and} \quad e_i \geqslant 0, \sum_i e_i \leqslant C$$

The latter leads to a Lagrangian formulation (with multiplier ν taking the rôle of C)

$$\|\boldsymbol{\beta}\|^2 + \nu \sum_{i=1}^{n} e_i \looparrowright \min_{\alpha,\beta}! \qquad e_i \geqslant 0 \qquad y_i(\alpha + \mathbf{x}_i^\top \boldsymbol{\beta}) \geqslant 1 - e_i \qquad (\text{all } i)$$

or

$$\frac{1}{\nu}\|\boldsymbol{\beta}\|^2 + \sum_{i=1}^{n} e_i \looparrowright \min_{\alpha,\beta}! \qquad e_i \geqslant \max\{0, 1 - y_i(\alpha + \mathbf{x}_i^\top \boldsymbol{\beta})\} \qquad (\text{all } i)$$

End of digression: regularized logistic regression

And this eventually yields, introducing $\lambda = \dfrac{1}{\nu}$

$$\sum_{i=1}^{n} (1 - y_i(\alpha + \mathbf{x}_i^T \boldsymbol{\beta}))_+ + \lambda \| \beta \|^2 \to \min_{\alpha, \boldsymbol{\beta}} !$$

Mathematical note: each C corresponds to some ν, which in turn corresponds to some λ. From the practical point of view, this means an additional requirement - for the data not necessarily separated, we need to select tuning parameter C (or ν or λ)...

How about regularizing the logistic regression instead? We would get rid of the problem when the data are separated... and retain many good properties instead!

Let us end the digression, return to logistic regression, and take

$$\sum_{i=1}^{n} \log \left(1 + e^{-y_i \mathbf{x}_i^T \boldsymbol{\beta}}\right) + \lambda \| \beta \|^2 \to \min_{\boldsymbol{\beta}} !$$

as a prescription for **regularized logistic regression**

The dual of regularized logistic regression

Differentiating the objective function in $\boldsymbol{\beta}$, we obtain

$$-\sum_{i=1}^{n} \frac{y_i e^{-y_i \mathbf{x}_i^T \boldsymbol{\beta}}}{1 + e^{-y_i \mathbf{x}_i^T \boldsymbol{\beta}}} \mathbf{x}_i + 2\lambda \boldsymbol{\beta} = 0 \qquad \text{yielding} \qquad \boldsymbol{\beta} = \frac{1}{2\lambda} \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$

where we set $\alpha_i = \dfrac{y_i e^{-y_i \mathbf{x}_i^T \boldsymbol{\beta}}}{1 + e^{-y_i \mathbf{x}_i^T \boldsymbol{\beta}}} = \dfrac{y_i}{1 + e^{y_i \mathbf{x}_i^T \boldsymbol{\beta}}}$ for $i = 1, 2, \ldots, n$

On multiplying by y_i (note that $y_i^2 = 1$ as $y_i = \pm 1$) and substituting for $\boldsymbol{\beta}$, we obtain from the latter equation

$$\alpha_i y_i e^{y_i \mathbf{x}_i^T \frac{1}{2\lambda} \sum_{j=1}^{n} \alpha_j \mathbf{x}_j} = 1 - \alpha_i y_i$$

which after taking logs and some rearrangement becomes

$$\log(\alpha_i y_i) + \frac{y_i}{2\lambda} \sum_{j=1}^{n} \alpha_j \mathbf{x}_i^T \mathbf{x}_j = \log(1 - \alpha_i y_i)$$

Taking logs is justified only for positive numbers: thus we have to add the constraint

$$0 < \alpha_i y_i < 1 \qquad \text{for all } i$$

And the dual is

On multiplying by y_i again, we obtain the equation(s)

$$y_i \log(\alpha_i y_i) - y_i \log(1 - \alpha_i y_i) + \frac{1}{2\lambda} \sum_{j=1}^{n} \alpha_j \mathbf{x}_i^\top \mathbf{x}_j = 0$$

the left-hand side of which, by straightforward verification, can be shown arising from the differentiation, in $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^\top$, of the objective function

$$\frac{1}{\lambda} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i y_i \log(\alpha_i y_i) + \sum_{i=1}^{n} (1 - \alpha_i y_i) \log(1 - \alpha_i y_i)$$

whose minimization - or more precisely, maximization of its negative - together with the constraints $0 < \alpha_i y_i < 1$ for all i, constitutes the dual problem

Putting kernel in

The quantity $\mathrm{H}(u_1, u_2, \ldots, u_n) = -\sum_{i=1}^{n} u_i \log u_i$ is called **entropy**; the dual is thus maximizing the sum of entropies

$$\mathrm{H}(\alpha_1 y_1, \alpha_2 y_2, \ldots, \alpha_n y_n) + \mathrm{H}(1 - \alpha_1 y_1, 1 - \alpha_2 y_2, \ldots, 1 - \alpha_n y_n)$$

penalized by $$-\frac{1}{\lambda} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j = -\frac{1}{\lambda} \boldsymbol{\alpha}^\top \mathbf{X}\mathbf{X}^\top \boldsymbol{\alpha}$$

The objective function is thus influenced by $\mathbf{X}$ only via inner products of lines $\mathbf{x}_i^\top \mathbf{x}_j$. This suggests an idea to introduce in it some other inner product function, to obtain corresponding inner products of potential other basis functions $\varphi(x)$. The new formulation thus minimizes the negative of the sum of the entropies above, penalized by

$$\frac{1}{\lambda} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$$

where $\mathcal{K}(\cdot, \cdot)$ is a suitable function called **kernel** (note: the word "kernel" here means something different than in the kernel density estimation)

Why we would like to do it?

Using kernels amounts to altering the space of regressors by suitable functions of those - like we did with the polynomials, when we added powers and products: but here it results in often often more interesting functions than those. Moreover, the tuning parameter λ further determines how "flexible" the fitted functional relationships are - similarly as λ controls the shape of the fitted spline in nonparametric regression via smoothing splines

To this end however, the inner product induced by a kernel must exhibit properties expected from an inner product. We require symmetry and also Cauchy-Schwarz inequality:

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \mathcal{K}(\mathbf{y}, \mathbf{x}) \quad \text{and} \quad \mathcal{K}(\mathbf{x}, \mathbf{y})^2 \leqslant \mathcal{K}(\mathbf{x}, \mathbf{x})\mathcal{K}(\mathbf{y}, \mathbf{y})$$

The optimization problem would be well-defined if the matrix with the elements $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ is nonnegative-definite. To have this property for any future $\mathbf{x}_i$ and $\mathbf{x}_j$, we usually demand that the kernel is a **Mercer kernel**: the matrix $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ is nonnegative definite for *any* collection $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ (not just the one appearing in our working dataset)

OK, but how do we recover the primal?

Altering the dual problem also alters the primal problem (which is, in fact, the dual of the dual). Once we have solved the dual problem and obtained the $\hat{\alpha}_i$'s, in the original problem, with $\mathcal{K}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$, we had

$$\hat{\boldsymbol{\beta}} = \frac{1}{2\lambda}\sum_{i=1}^{n} \hat{\alpha}_i \mathbf{x}_i$$

Now, the β_i's may be different - but in fact, we are not after them, but for classified *new* objects rather after $\mathbf{x}^T\hat{\boldsymbol{\beta}}$ - because this function determines, via inverse logit, the predicted posterior probabilities, and thence the classes. There we have

$$\hat{f}(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} = \frac{1}{2\lambda}\sum_{i=1}^{n} \alpha_i \mathbf{x}^T\mathbf{x}_i \quad \text{which is} \quad \frac{1}{2\lambda}\sum_{i=1}^{n} \alpha_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i)$$

So, let us hope that this way of recovering the prediction of the posterior probabilities - which should be then

$$\frac{1}{1 + e^{\hat{f}(\mathbf{x})}} \quad \text{and} \quad \frac{e^{\hat{f}(\mathbf{x})}}{1 + e^{\hat{f}(\mathbf{x})}}$$

would work also in the "kernelized" situation

Reproducing kernels

This recovery of $\widehat{\mathrm{f}}(\mathbf{x})$ can be done from the knowledge of the basis $\boldsymbol{\varphi}(\mathbf{x})$ induced by the kernel - but that may be tedious and/or computationally expensive (for some kernels, the dimension of the space $\boldsymbol{\varphi}(\mathbf{x})$ is even infinite)

Luckily, it can be done in much more expedient way if the kernel is so-called **reproducing kernel**; for such kernels, once once $\hat{\boldsymbol{\alpha}}$ is obtained solving the dual problem, then

$$\widehat{\mathrm{f}}(\mathbf{x}) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}} \mathcal{K}(\mathbf{x}_{\mathrm{i}}, \mathbf{x})$$

- so called **representer theorem**

The definition of reproducing kernels and representer theorem are formulated and proved in the framework of Hilbert space theory. However, for several kernels used in practice, these properties are alrady known

Examples of kernels

The one we started with ("no kernel") is the **linear kernel**

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y} \qquad \texttt{(vanilladot)}$$

The very much used one is **Gaussian** (or squared exponential) (radial basis) **kernel** - with this one comes also another tuning parameter, σ):

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \mathrm{e}^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} \qquad \texttt{(rbfdot)}$$

A somewhat similar one (also with tuning parameter σ) one is **Laplace** (radial basis) **kernel**

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \mathrm{e}^{-\frac{\|\mathbf{x}-\mathbf{y}\|}{\sigma}} \qquad \texttt{(laplacedot)}$$

Another one is **polynomial kernel** of degree d (and tuning parameters τ and σ)

$$\mathcal{K}(\mathbf{x}, \mathbf{y})(\sigma \mathbf{x}^\top \mathbf{y} + \tau)^{\mathrm{d}} \qquad \texttt{(polydot)}$$

... and there are quite a few others

(The names in brackets refer to the R package `kernlab`)

Let us go for some of those

Unfortunately, it seems there is no reliable implementation for kernelized logistic regression in R yet. The following examples have been computed via package `kernlab` using support verctor machines - where the dual instead of minimizing

$$\frac{1}{\lambda}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\mathcal{K}(\mathbf{x}_i,\mathbf{x}_j)$$
$$-H\left(\alpha_1 y_1,\ldots,\alpha_n y_n\right)-H\left(1-\alpha_1 y_1,\ldots,1-\alpha_n y_n\right)$$

minimizes

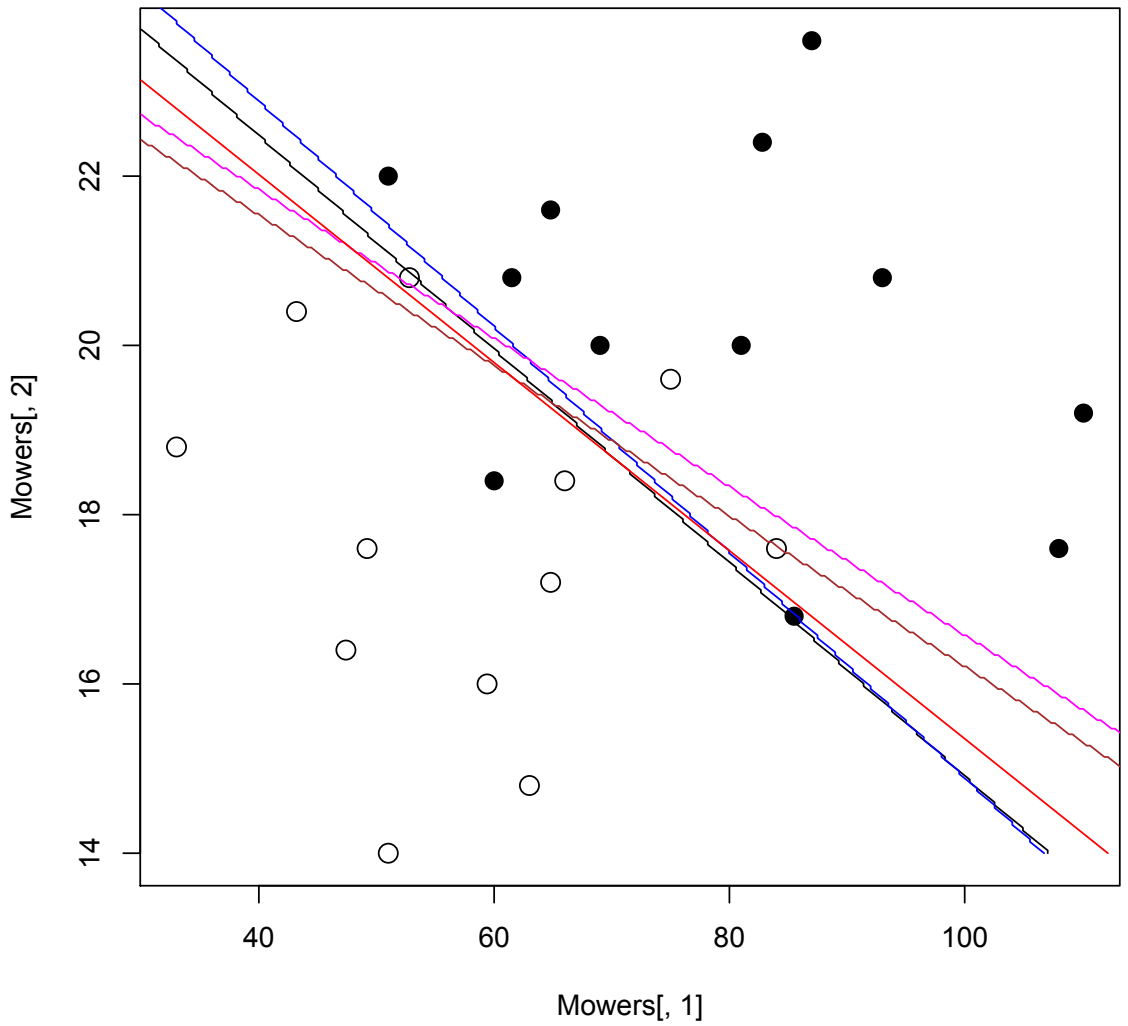
$$-\sum_{i=1}^{n}\alpha_i+\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\mathcal{K}(\mathbf{x}_i,\mathbf{x}_j)$$

under the constraints $\sum_{i=1}^{n}\alpha_i y_i=0$ and $0\leqslant\alpha_i\leqslant C$ for all i

```
> library(kernlab)
> ksvm(factor(riding)~income+lot,data=Mowers,scaled=F,
+ kernel='vanilla',C=0.01)
> ksvm(riding~income+lot,type="C-svc",data=Mowers,scaled=F,
+ kernel='vanilla',C=0.1)
```

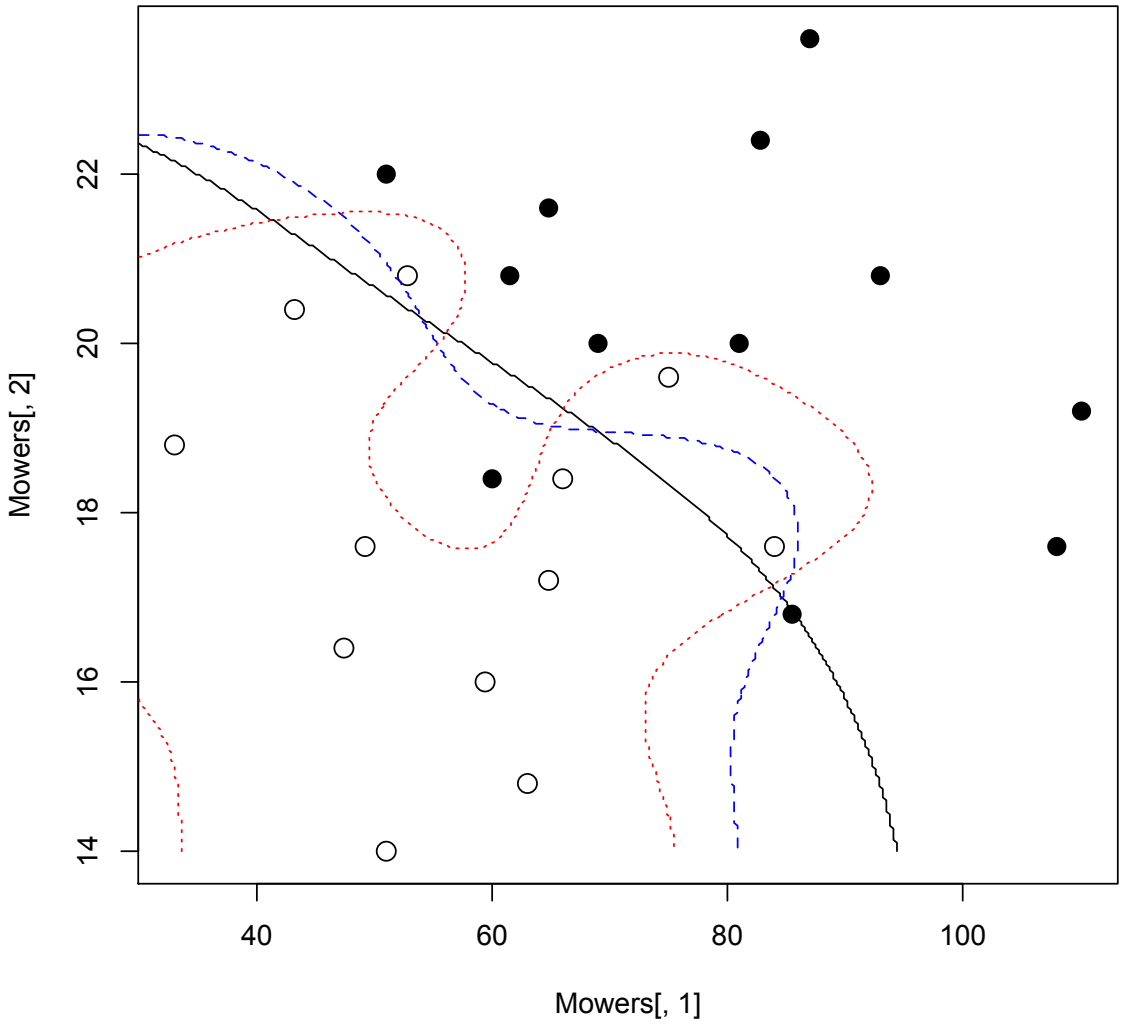
Mowers again: linear kernel, various C

$C = 0.01, 0.1, 1, 10, 100.$



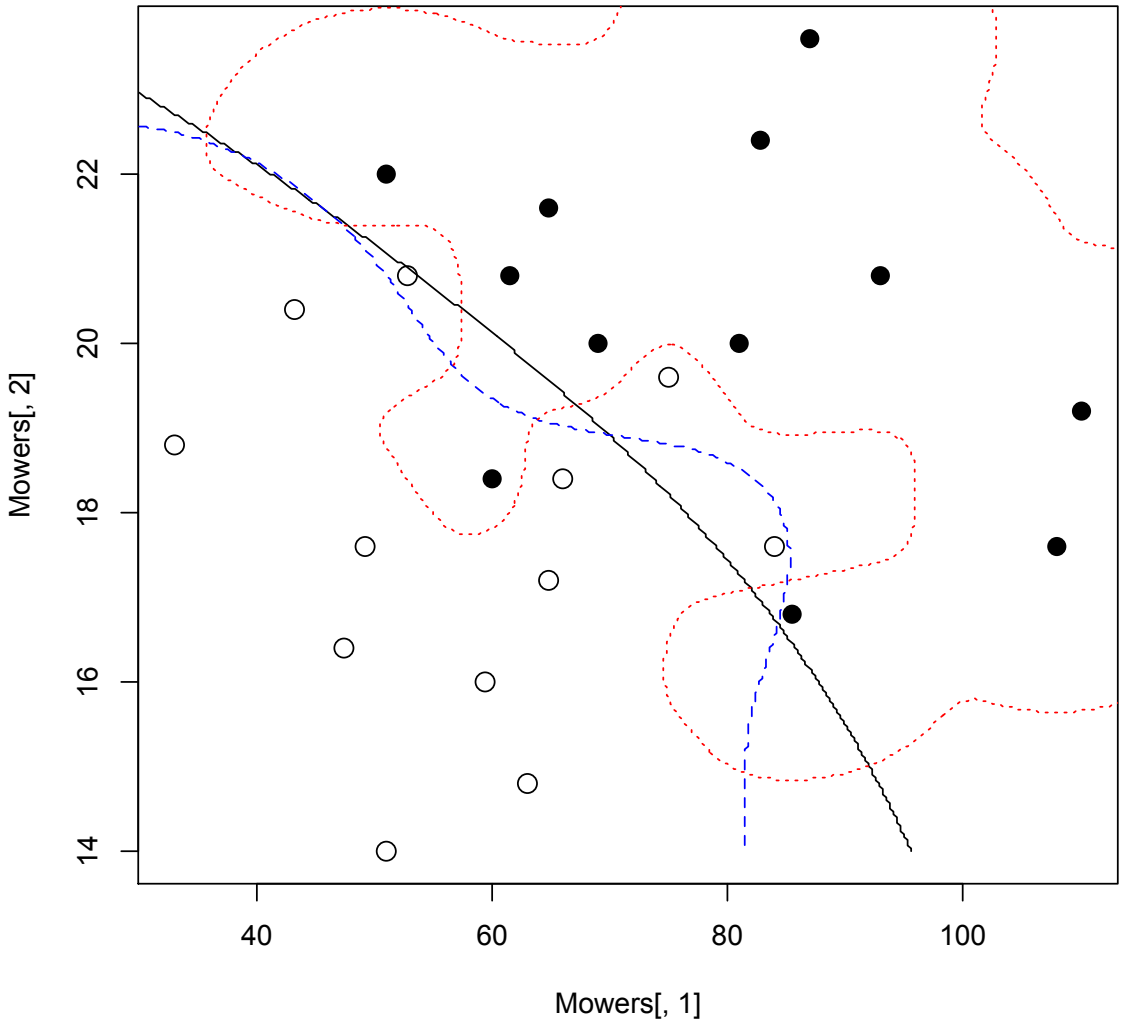
Mowers: Gaussian kernel, various C

$C = 0.001, 1, 100.$



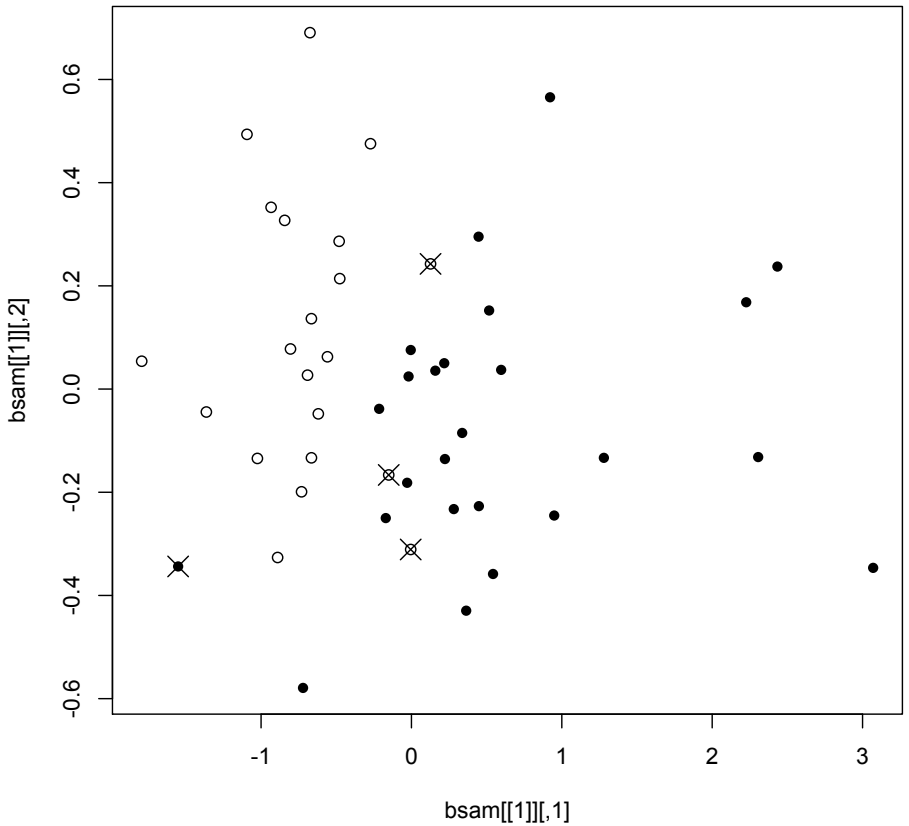
Mowers: Gaussian kernel, $C = 1$, various σ

$\sigma = 0.1, 1, 10$. A lot of room to play...



Bank data, default (Gaussian kernel, $C = 1$)

```
> bsam=sammon(dist(Bank[,1:4]))
> plot(bsam[[1]],pch=15*Bank$k+1)
> bansvm=ksvm(factor(k)~.,data=Bank)
Using automatic sigma estimation (sigest) for RBF or laplace kernel
> wrong=(Bank$k!=(predict(bansvm)))
> points(bsam[[1]][wrong,],pch=4,cex=2)
```



Classification trees:
classification à la old botanic identification

Classification trees

They use the original variables rather than their linear combinations or other transformations. A tree-like decision scheme is constructed, via the recursive splitting of the fitting data according to the values of classifiers

As a rule, only binary splits are allowed, for simplicity. Splits are chosen to optimize some splitting criterion; as the full search over all possibilities is not algorithically feasible, the method rather looks ahead only few steps – usually one (this is similar to the way how, for instance, programs playing chess are implemented; in computer science lingo such algorithms are called greedy)

The splitting is continued until the minimal prescribed size of the leaf is reached, or until the leafs are homogeneous enough (the deviance is small compared to the initial one)

After this, the trees are usually adjusted subjectively, to prevent overfitting - *pruning*; cross-validation error rate may be a useful indicator in this process.

Criteria for splitting I: impurity

Apparent error rate is again not suitable. An i-th node of tree can be seen as giving (unknown) conditional probabilities p_{ig},

estimated by $\frac{n_{ig}}{n_i} = \frac{n_{ig}}{\sum_k n_{ig}}$

The majority of implementations choose splits to maximize the average impurity decrease: the impurity of a node is quantified by some "impurity index", the most common ones are

Gini index $\quad 1 - \sum_g p_{ig}^2$

Entropy $\quad -\sum_g p_{ig} \log p_{ig}$

(convention: $0 \log 0 = 0$)

A good measure of this kind is supposed to be minimal (0), if one of the p_{ig}'s is 1 (and others 0), and maximal, if the distribution is uniform (all p_{ig} are the same)

Using entropy is motivated by information theory - and we will see that its use is equivalent to using so-called deviance

Average impurity decrease

"Impurity decrease" means the impurities of two nodes after the split are subtracted from the impurity of the node before the split; "average" means that the impurities of the subtracted notes are before that each multiplied by the relative proportion of all the objects in the node to all objects in the parent node.

That is, if node s, comprising n_s objects in total, is split into nodes t and u, with n_t and n_u objects in total respectively

(that is, $n_s = n_t + n_u$, $n_t = \sum_g n_{tg}$, $n_u = \sum_k n_{ug}$, $n_s = \sum_k n_{sg}$)

then we are maximizing

$$i_s - \frac{n_t}{n_s} i_t - \frac{n_u}{n_s} i_u$$

where i_s, i_t, i_u are the corressponding impurities; for instance, with entropy as impurity measure this is equal to

$$-\sum_g \left(\frac{n_{sg}}{n_s} \log \frac{n_{sg}}{n_s} \right) + \frac{n_t}{n_s} \sum_g \left(\frac{n_{tg}}{n_t} \log \frac{n_{tg}}{n_t} \right) + \frac{n_u}{n_s} \sum_g \left(\frac{n_{ug}}{n_u} \log \frac{n_{ug}}{n_u} \right)$$

Average impurity decrease in R

Entropy:

```
> ent = function(x) sum(-x*log(x+0.0000001))
> ent(c(1/2,1/2))-8/24*ent(c(7/8,1/8))-16/24*ent(c(5/16,11/16))
[1] 0.1534995
> ent(c(1/2,1/2))-5/24*ent(c(0/5,5/5))-19/24*ent(c(12/19,5/19))
[1] 0.1852558
```

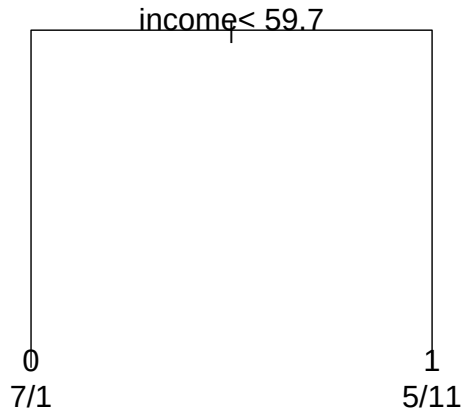
Gini:

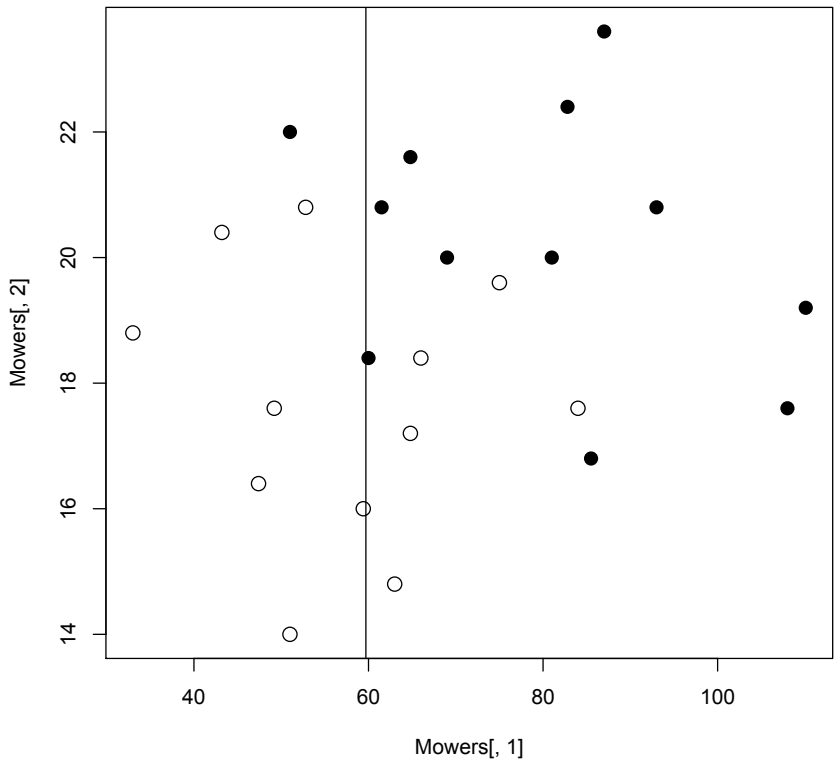
```
> gini = function(x) 1-sum(x*x)
> gini(c(1/2,1/2))-8/24*gini(c(7/8,1/8))-16/24*gini(c(5/16,11/16))
[1] 0.140625
> gini(c(1/2,1/2))-5/24*gini(c(0/5,5/5))-19/24*gini(c(12/19,5/19))
[1] 0.07894737
```

Owners of riding mowers: a coarse tree

...also called a *stump*.

```
> library(rpart)
> mowtree=rpart(factor(riding)~income+lot,data=mowers)
> plot(mowtree,margin=0.1)
> text(mowtree,use.n=TRUE,cex=1.3)
```

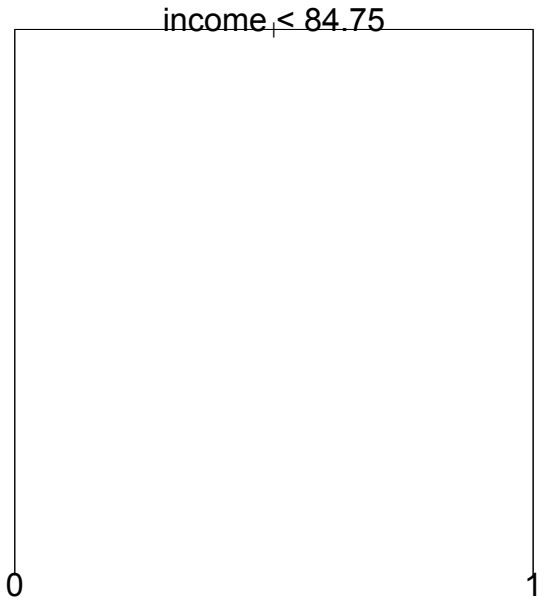


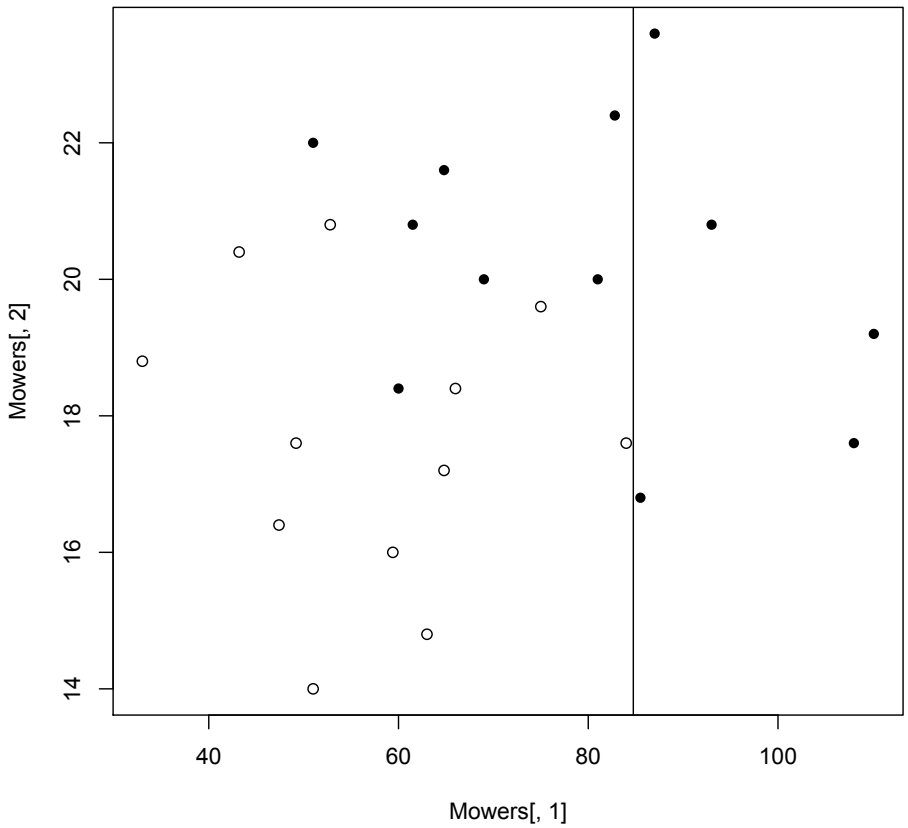


Which impurity was used?

Owners of riding mowers: another coarse tree

```
> library(tree)
> mowtree=tree(factor(riding)~income+lot,data=mowers,
+ control=tree.control(nobs=nrow(mowers),minsize=24),split="deviance")
> plot(mowtree)
> text(mowtree,cex=2)
```





Criteria for splitting II: likelihood and deviance

Thinking in likelihood terms, the likelihood of the tree is

$$\prod_i \prod_k p_{ig}^{n_{ig}}$$

where i runs over *leaves*, the terminal nodes, and k over classes

Likelihood is still maximized after taking logs

Reversing the sign results in minimization instead of maximization

This leads to the *deviance*

$$D = \sum_i D_i = \sum_i \left(-2 \sum_g n_{ig} \log p_{ig}\right)$$

which is the sum, over the leaves, of deviances

$$D_i = -2 \sum_g n_{ig} \log p_{ig}$$

(factor 2 is there just for traditional reasons; obviously it does not change anything)

The reduction in deviance

If node s is split into nodes t and u

we observe the reduction in the deviance

$$
\begin{aligned}
&D_s - D_t - D_u \\
&\quad = -2\sum_g n_{sg} \log p_{sg} + 2\sum_g n_{tg} \log p_{tg} + 2\sum_g n_{ug} \log p_{ug} \\
&\quad = -2\sum_g (n_{tg}+n_{ug}) \log p_{sg} + 2\sum_g n_{tg} \log p_{tg} + 2\sum_g n_{ug} \log p_{ug} \\
&\quad = 2\sum_k \left(n_{tg} \log \frac{p_{tg}}{p_{sg}} + n_{ug} \log \frac{p_{ug}}{p_{sg}} \right)
\end{aligned}
$$

It is equivalent to entropy as impurity

After replacing the probabilities by their estimates

$$\hat{p}_{tg} = \frac{n_{tg}}{n_t}; \quad \hat{p}_{ug} = \frac{n_{ug}}{n_u}; \quad \hat{p}_{sg} = \frac{n_{tg} + n_{ug}}{n_t + n_u} = \frac{n_{sg}}{n_s};$$

where $n_t = \sum_g n_{tg}$, $n_u = \sum_g n_{ug}$, $n_s = \sum_g n_{sg}$,

we obtain that the reduction in deviance is equal to

$$2 \sum_g \big(n_{tg} \log n_{tg} + n_{ug} \log n_{ug} - n_{sg} \log n_{sg}$$

$$+ n_s \log n_s - n_u \log n_u - n_t \log n_t \big)$$

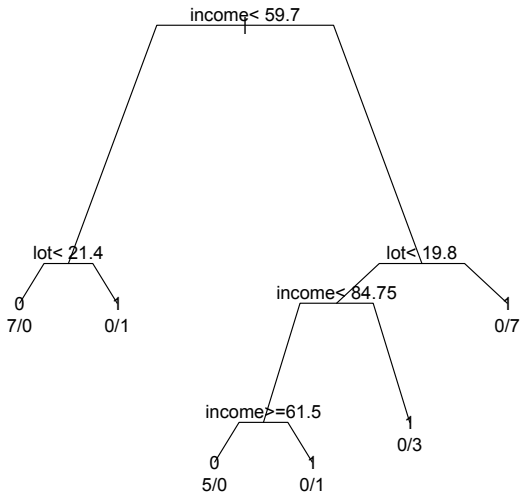
$$= 2 \sum_g \left(n_{tg} \log \frac{n_{tg}}{n_t} + n_{ug} \log \frac{n_{ug}}{n_u} - n_{sg} \log \frac{n_{sg}}{n_s} \right)$$

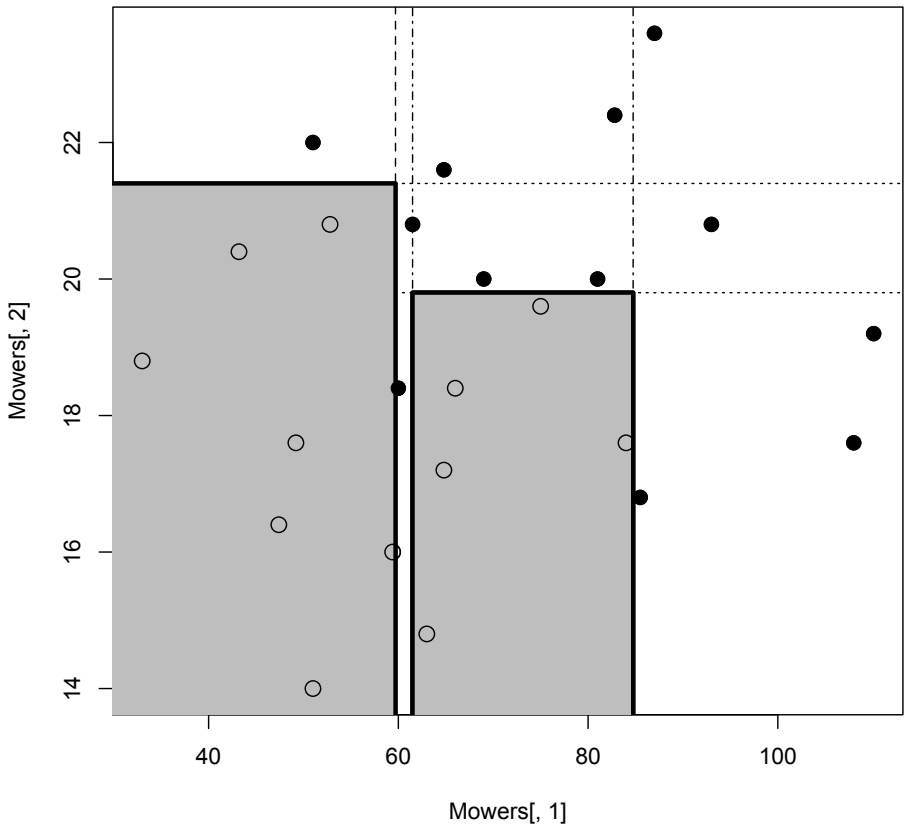
$$= 2 n_s \left(\frac{n_t}{n_s} \sum_g \left(\frac{n_{tg}}{n_t} \log \frac{n_{tg}}{n_t} \right) + \frac{n_u}{n_s} \sum_g \left(\frac{n_{ug}}{n_u} \log \frac{n_{ug}}{n_u} \right) - \sum_g \left(\frac{n_{sg}}{n_s} \log \frac{n_{sg}}{n_s} \right) \right)$$

which shows its maximization is equivalent to the maximization of the average impurity decrease for the entropy impurity.

Owners of riding mowers: fine tree

```
> mowtr=rpart(factor(riding)~income+lot,data=mowers,minsplit=1,cp=0.00001)
> plot(mowtr,branch=0.5,margin=0.1)
> text(mowtr,use.n=TRUE)
```





Criteria for pruning

The criterion for splitting, R, often leads to a maximal tree, the tree with the maximal number of splits. This is not always optimal, and in particular, it can overfit. Thus we usually reduce the tree by cutting of branches - *pruning*.

Given $\alpha \geqslant 0$, there is always a tree (with minimal size) minimizing

$R_\alpha = R + \alpha$ size

The trees minimizing this for various α are nested, so pruning can indeed lead us to one of those - if we know what α we would like.

The selection of α

One possibility is to try some version of cross-validation: 10-fold cross-validation divides data randomly to 10 equally-sized groups, evaluates the quality of prediction on each group (the rule determined from the remaining 9 groups), and then computes the aggregate error rate, which we can plot against α and see.

We would like to select α that minimizes the cross-validated error, but this often leads to the maximal tree. Then, another rule may be handy, the 1-SE rule: it selects the α which yields the largest error within 1 standard deviation of the minimal one.

In the `library(rpart)` implementation, `cp` is α divided by R_0. The dashed line on the complexity plot indicates the distance of one standard deviation from the minimal error.

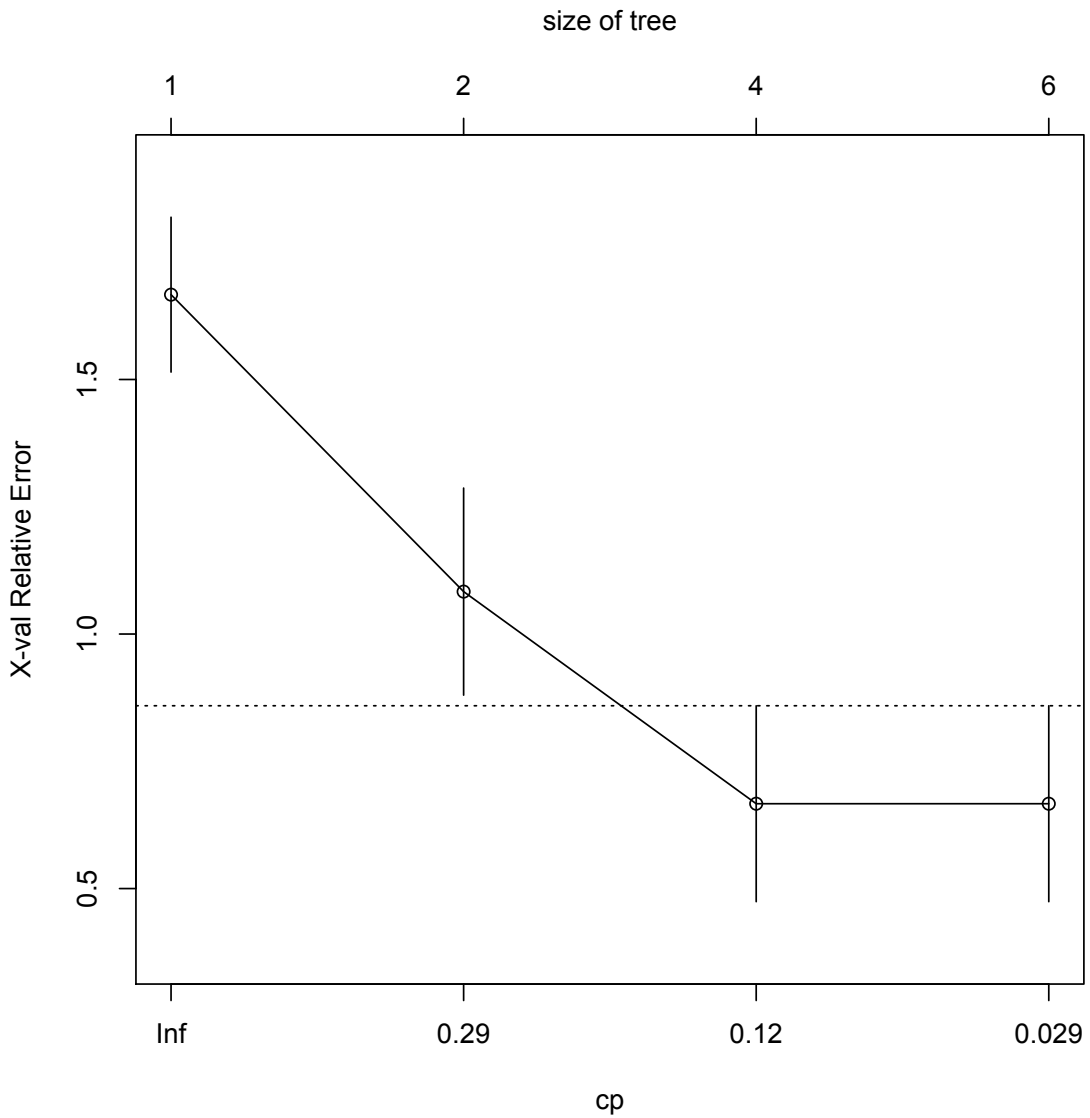
Also, in this implementation, `cp` shows somewhat differently in the printout and in the plot. While the printout shows the *minimal* `cp` for which we start to get a particular tree, the plot shows a geometric mean of two such consecutive α

Complexity printout

```
> printcp(mowtr)
...
        CP nsplit rel error  xerror    xstd
1 0.500000      0   1.00000 1.33333 0.19245
2 0.166667      1   0.50000 1.08333 0.20341
3 0.083333      3   0.16667 0.83333 0.20127
4 0.000010      5   0.00000 0.83333 0.20127
> mowtr$cptable
          CP nsplit rel error    xerror      xstd
1 0.50000000      0 1.0000000 1.3333333 0.1924501
2 0.16666667      1 0.5000000 1.0833333 0.2034141
3 0.08333333      3 0.1666667 0.8333333 0.2012691
4 0.00001000      5 0.0000000 0.8333333 0.2012691

> plotcp(mowtr)
```

The crossvalidation table was constructed already when creating the `mowtr` object - so given the randomness involved in the default 10-fold crossvalidation, to reproduce the result we need to set random seed generator before running `rpart()`

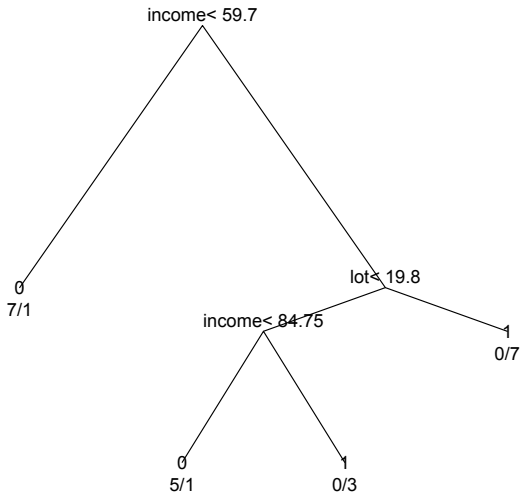
And the complexity plot

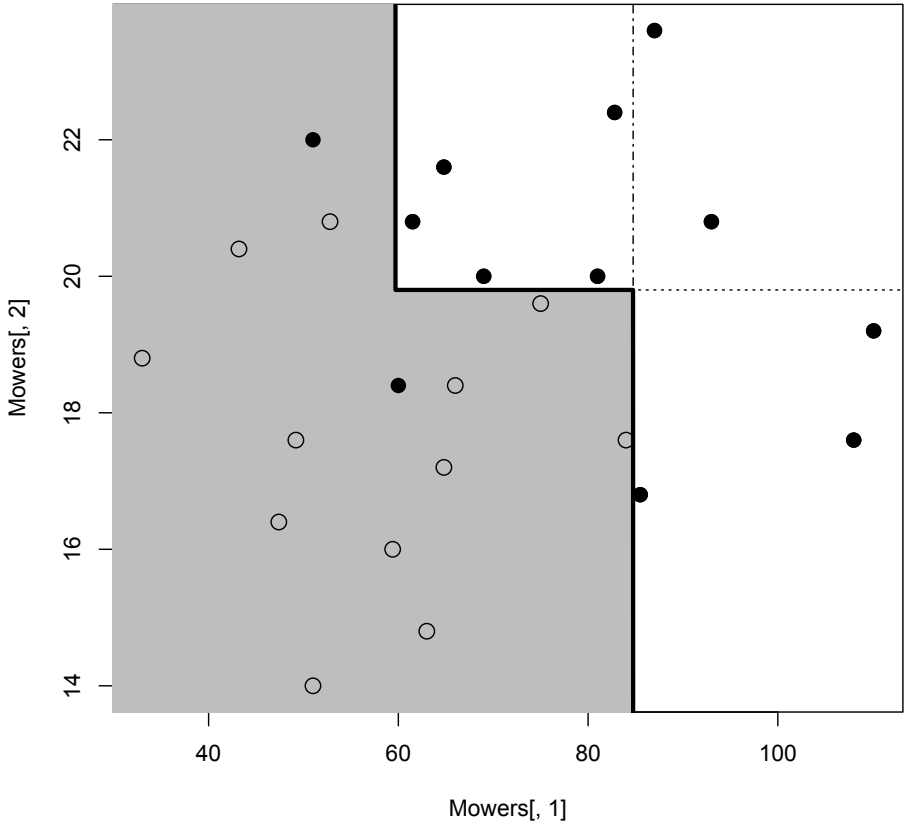


Fine aspects of cp's shown

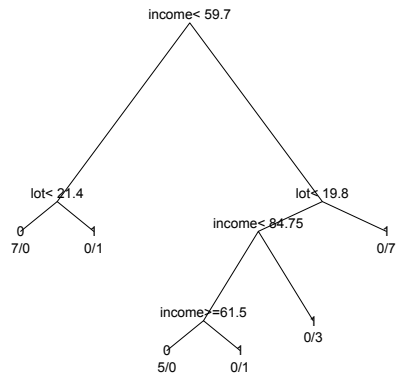
```
> cps=mowtr$cptable[,1]
> cps
         1          2          3          4
0.50000000 0.16666667 0.08333333 0.00001000
> sqrt(cps * c(Inf, cps[-length(cps)]))
           1            2            3             4
         Inf 0.2886751346 0.1178511302 0.0009128709

> mowtree=rpart(factor(riding)~income+lot,data=mowers,
+ minsplit=1,cp=0.12)
> plot(mowtree,branch=0,margin=0.1)
> text(mowtree,use.n=TRUE)
> mowtree=rpart(factor(riding)~income+lot,data=mowers,
+ minsplit=1,cp=0.08333333)
...
> mowtree=rpart(factor(riding)~income+lot,data=mowers,
+ minsplit=1,cp=0.0833333333333)
...
> mowtree=rpart(factor(riding)~income+lot,data=mowers,
+ minsplit=1,cp=0.08333333333333333)
...
```

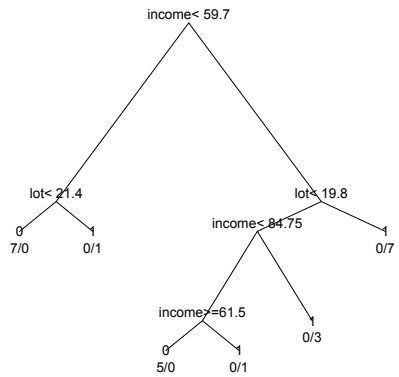
Owners of riding mowers: pruned tree

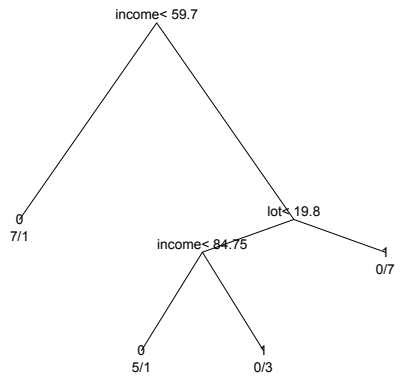




Note, however, for different cp





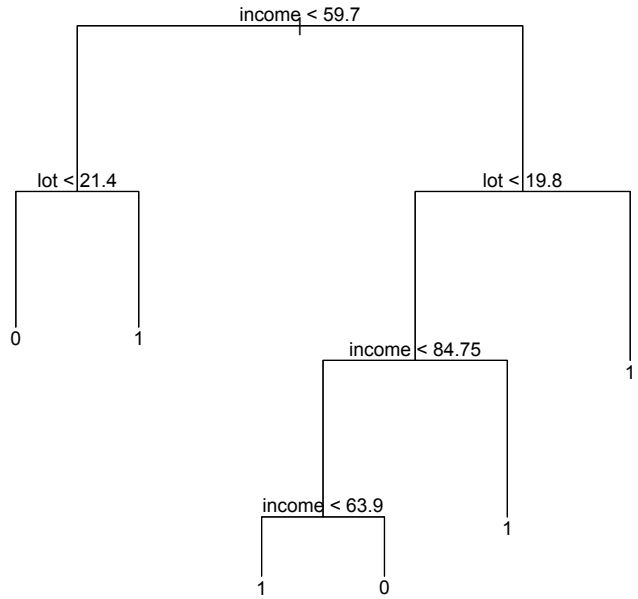


0.08333333 0.083333333333333 0.0833333333333333333

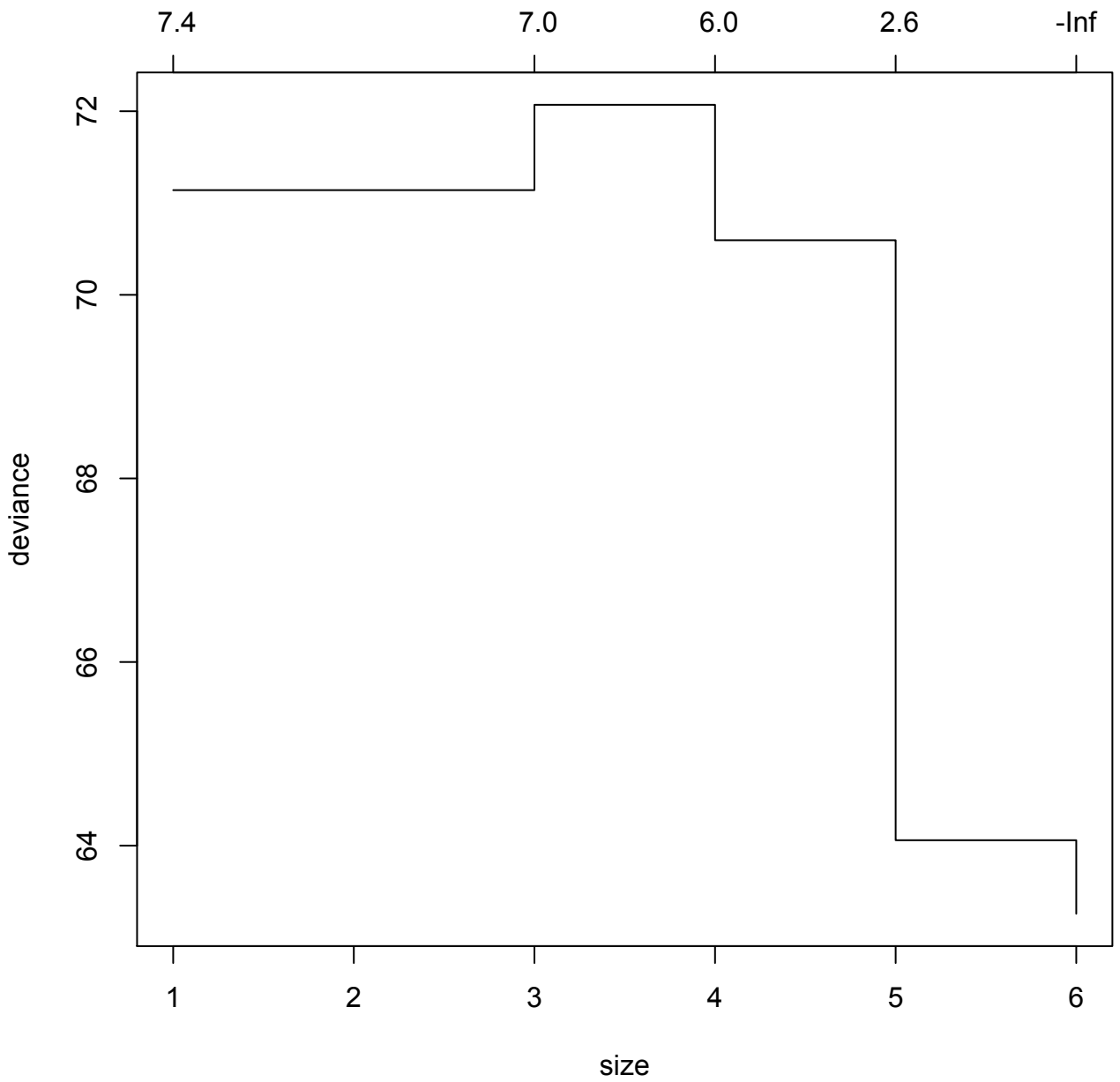
Obviously, cp from the "middle of the range" is good even rounded...

An alternative package now

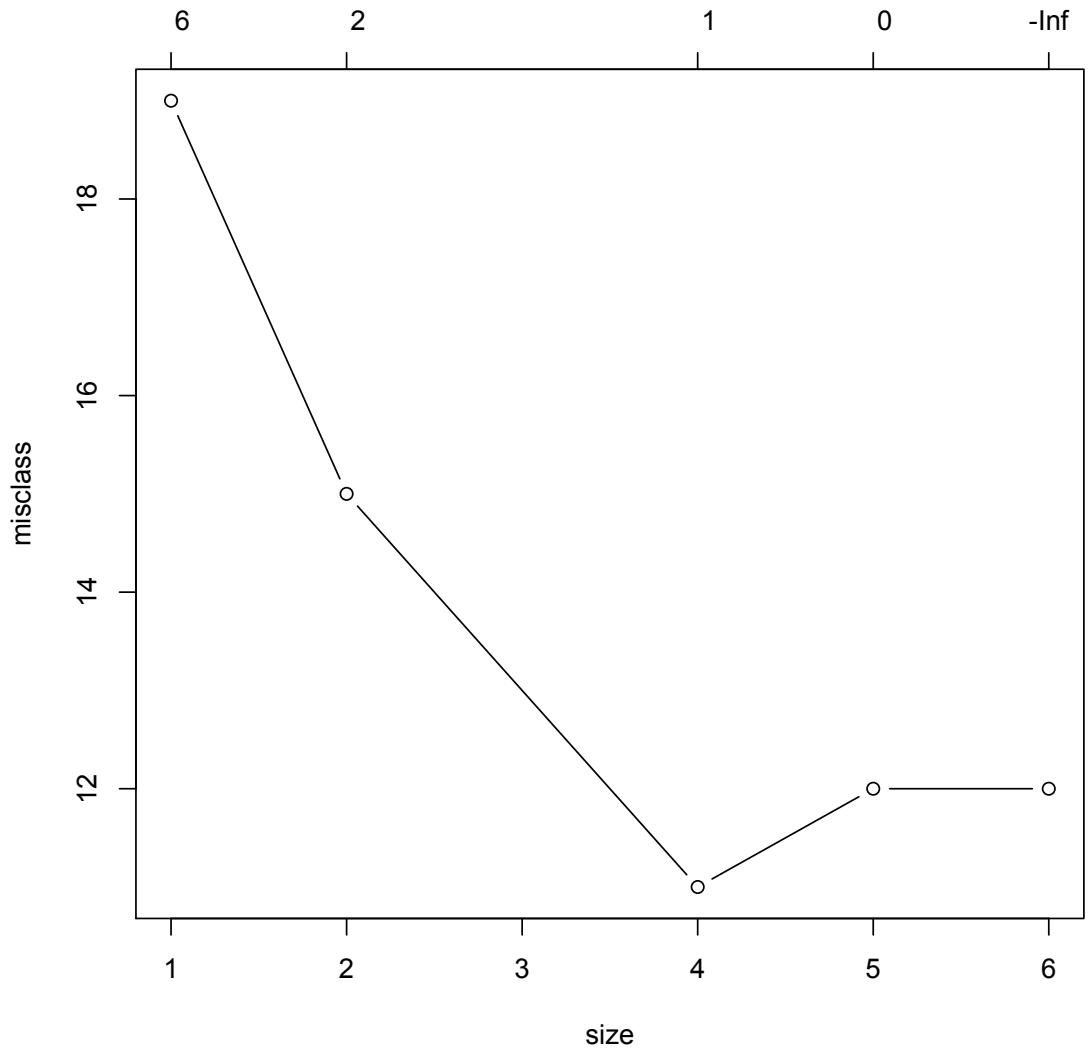
```
> mowtree=tree(factor(riding)~income+lot,data=Mowers,
+ control=tree.control(nobs=nrow(Mowers),minsize=1,mindev=0),
+ split="gini")   ## split="deviance" is actually the default
> plot(mowtree)
> text(mowtree)
> mowcv=cv.tree(mowtree)   ## randomness only here
> plot(mowcv)
```



Useful?

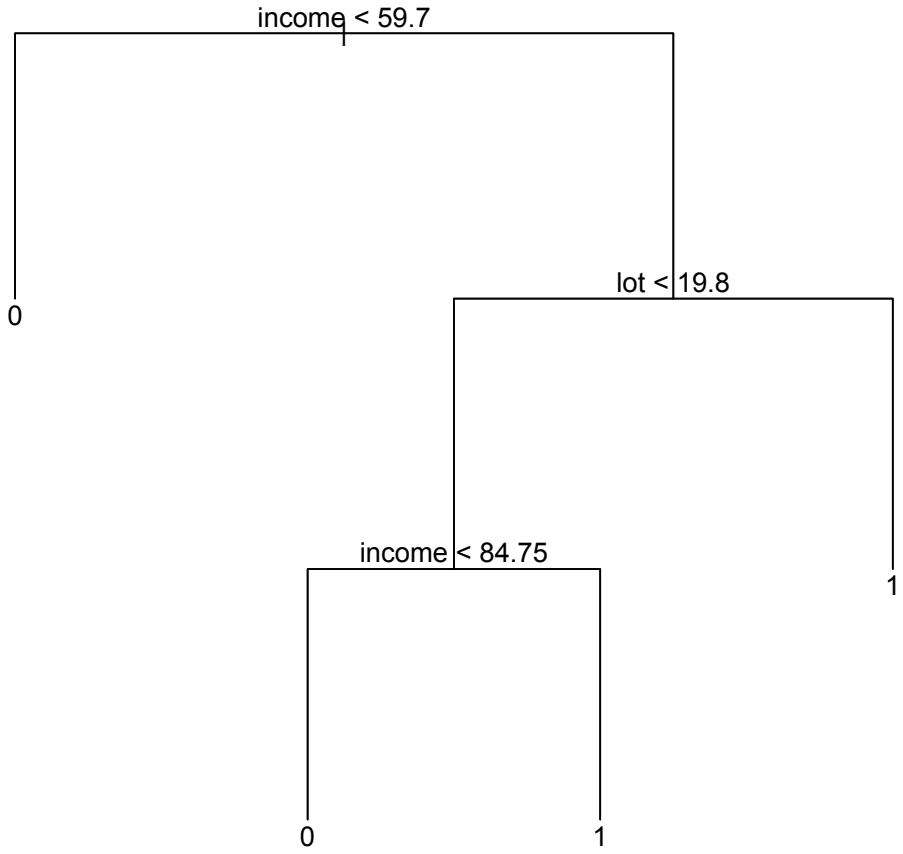


With some knowledgeable use...



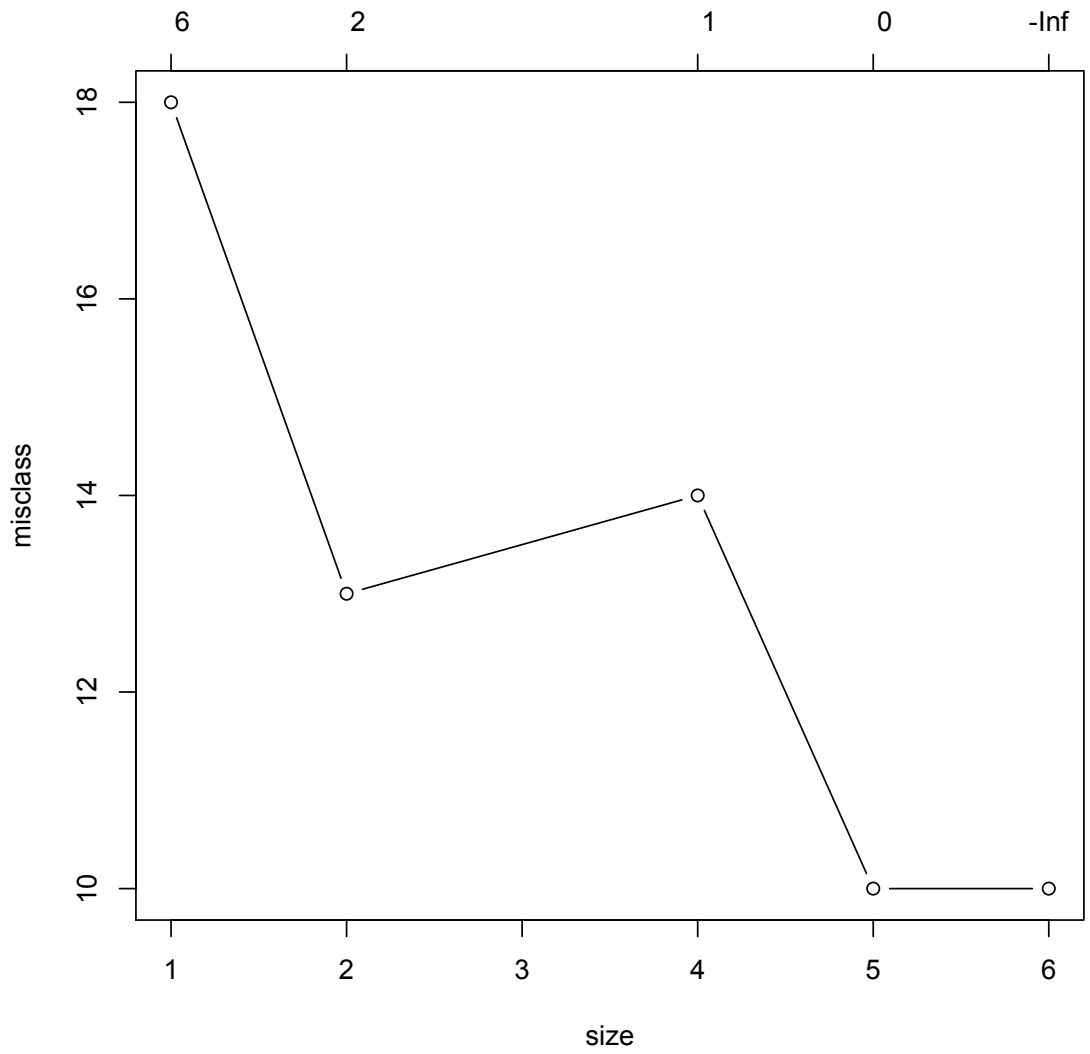

```
> mowcv=cv.tree(mowtree,FUN=prune.misclass)
> plot(mowcv,type="b")
```

... and with some luck (this would be very rare)



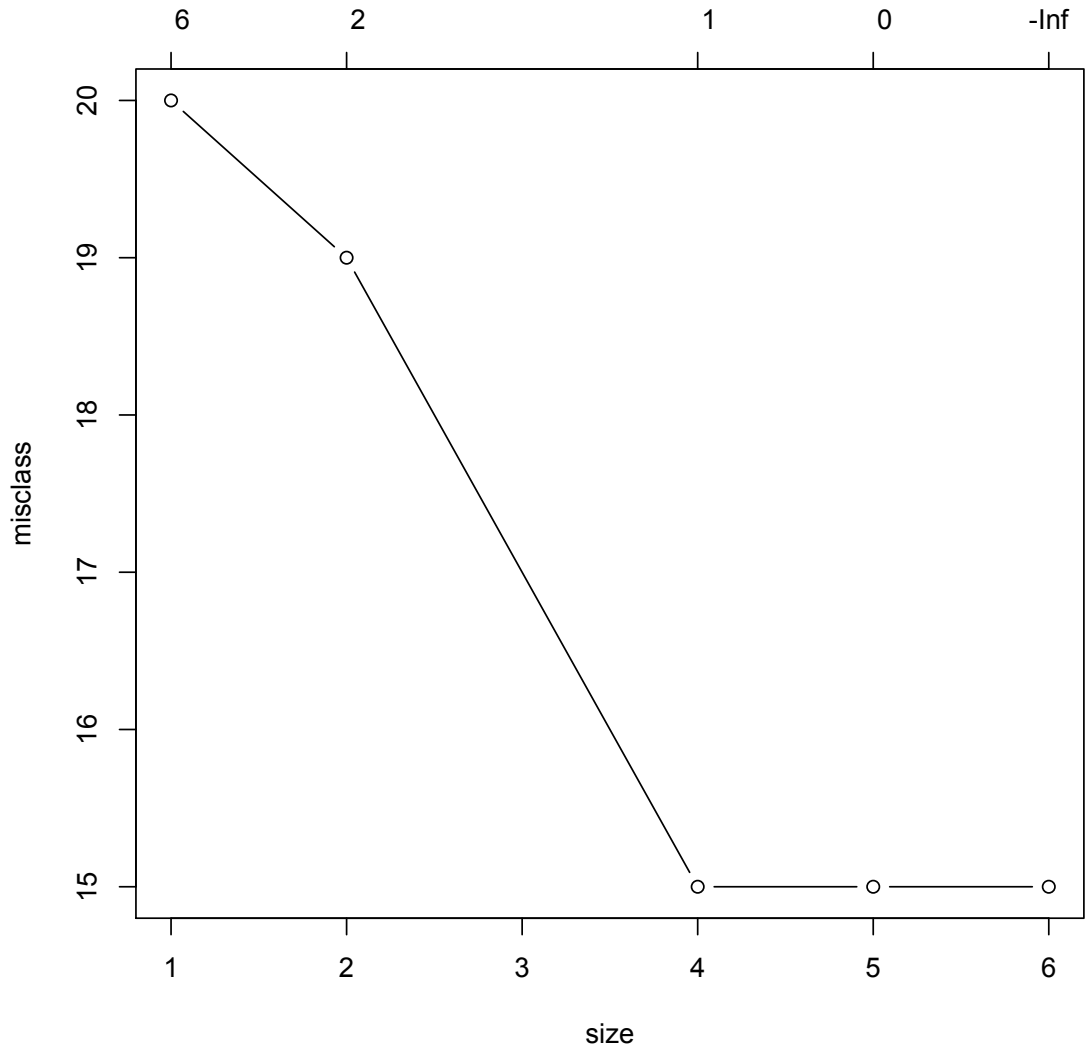

```
> plot(prune.misclass(mowtree,best=4))
> text(prune.misclass(mowtree,best=4))
```

One can fine tune here: choose K- (M−)



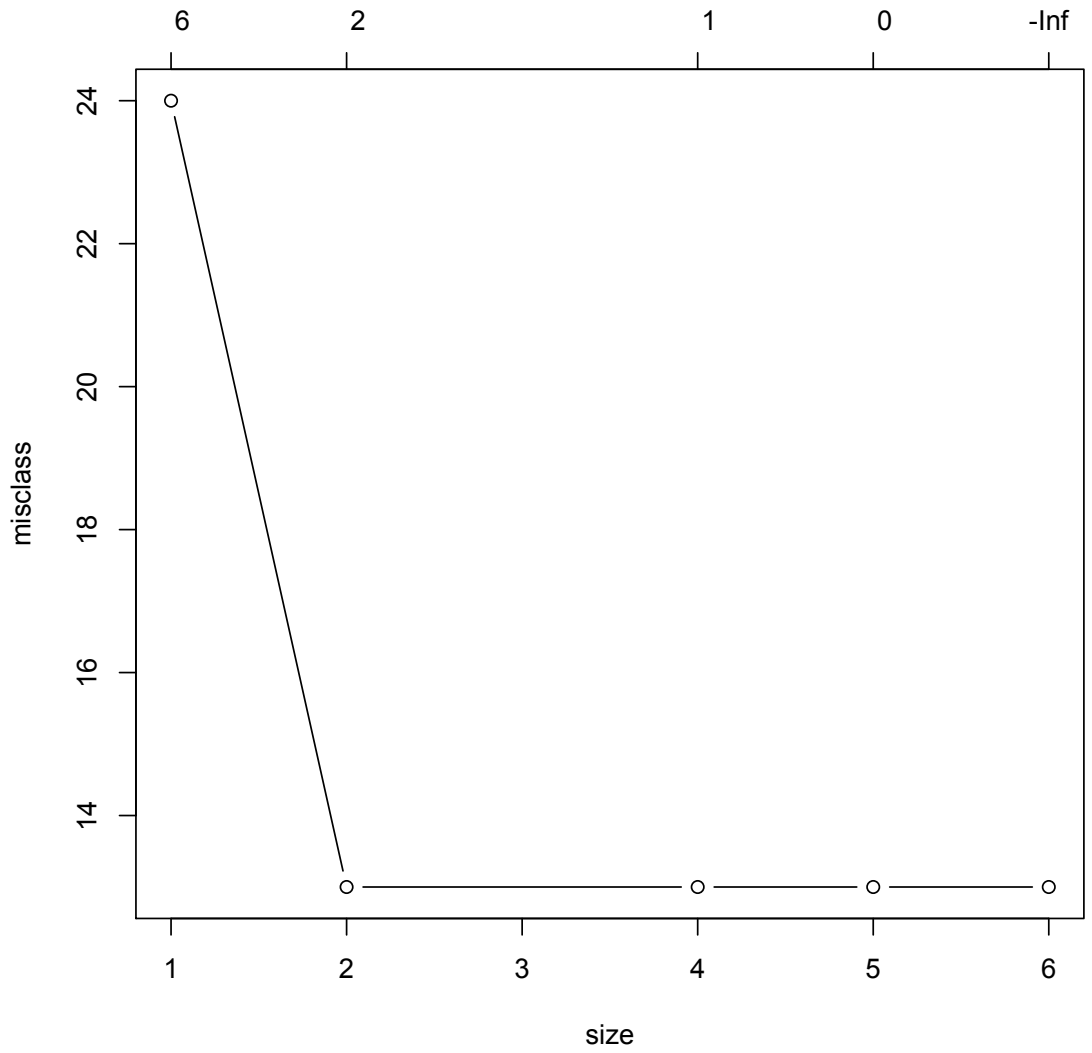

```
> mowcv=cv.tree(mowtree,FUN=prune.misclass,K=4)
> plot(mowcv,type="b")
```

And choose folds (eliminate randomness)



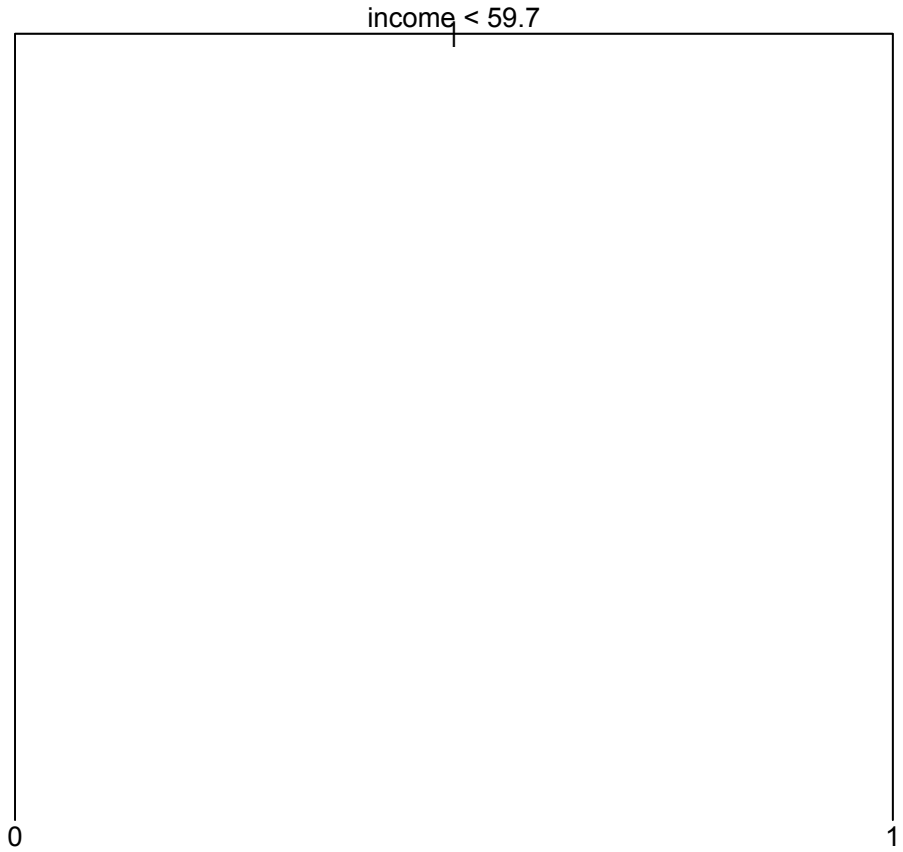

```
> mowcv=cv.tree(mowtree,rep(1:4,rep(6,4)),FUN=prune.misclass)
> ## seems like it knows K then, and ignores it even if wrong
> plot(mowcv,type="b")
```

Even leave-one-out crossvalidation possible...




```
> mowcv=cv.tree(mowtree,1:24,FUN=prune.misclass)
> plot(mowcv,type="b")
```

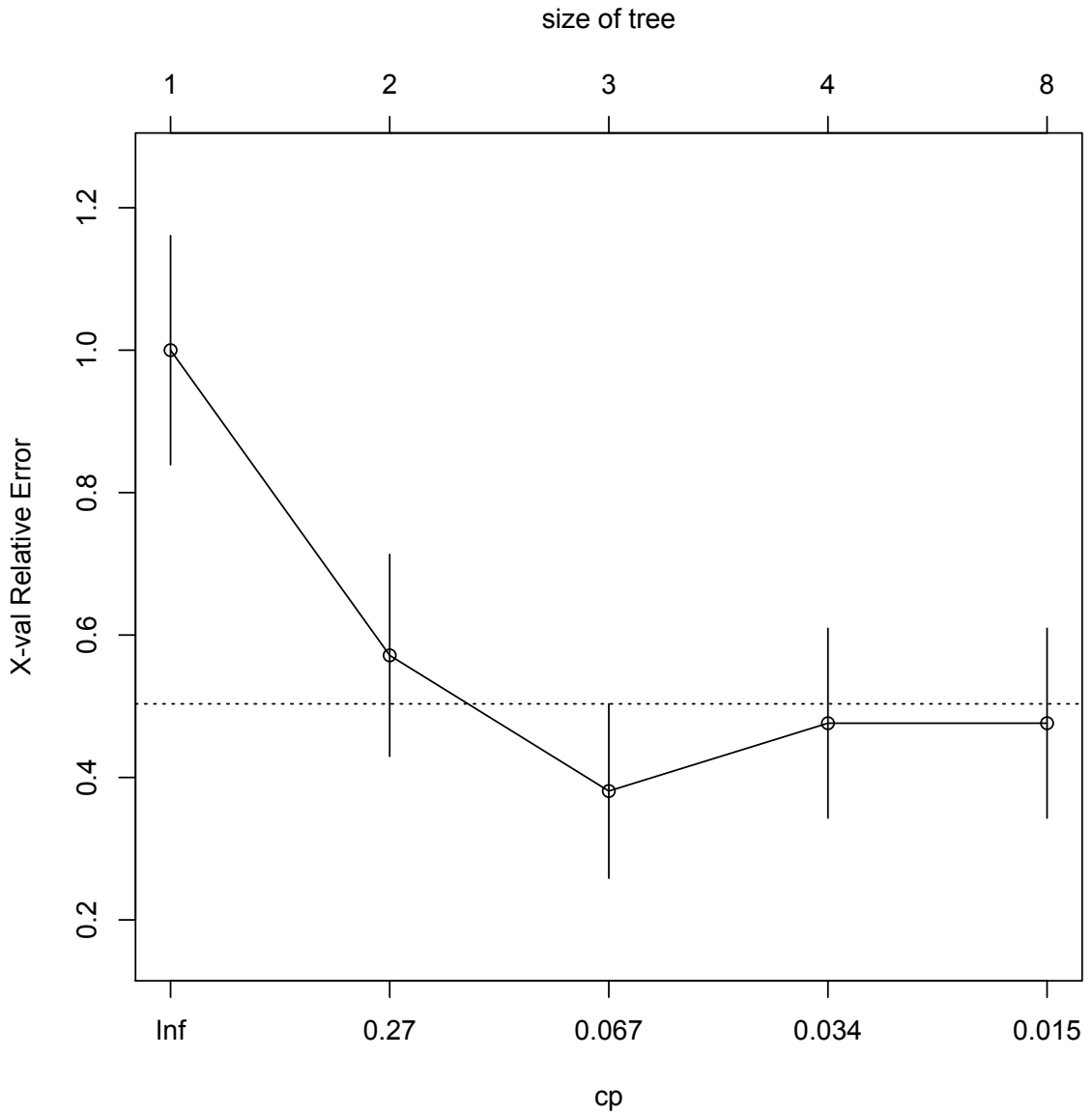
... although probably not very useful




```
> plot(prune.misclass(mowtree,best=2))
> text(prune.misclass(mowtree,best=2))
```

Bank data: complexity plot...

```
> bankfull=rpart(factor(k)~.,data=bank,minsplit=1)
> plotcp(bankfull)
```

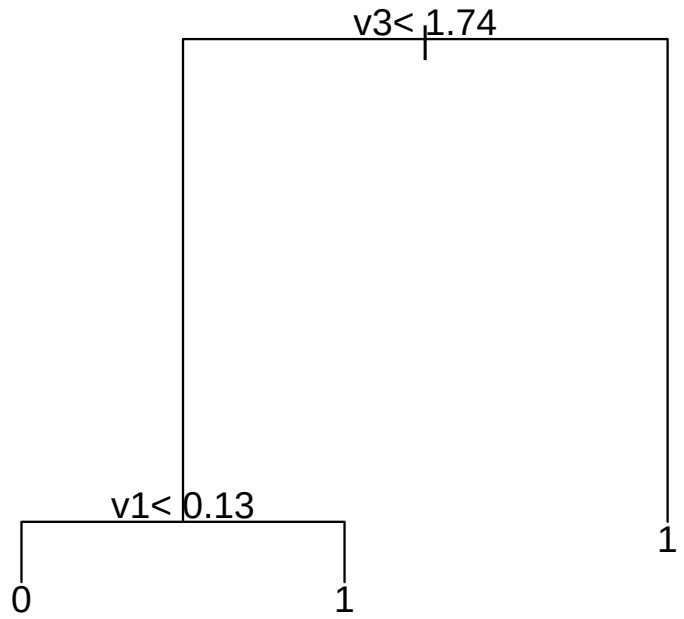


...and printout

```
> printcp(bankfull)

Classification tree:
rpart(formula = factor(k) ~ ., data = Bank, minsplit = 1)

Variables actually used in tree construction:
[1] v1 v3 v4

Root node error: 21/46 = 0.45652

n= 46

        CP nsplit rel error  xerror    xstd
1 0.761905      0  1.000000 1.00000 0.16087
2 0.095238      1  0.238095 0.57143 0.14182
3 0.047619      2  0.142857 0.38095 0.12242
4 0.023810      3  0.095238 0.47619 0.13321
5 0.010000      7  0.000000 0.47619 0.13321
```

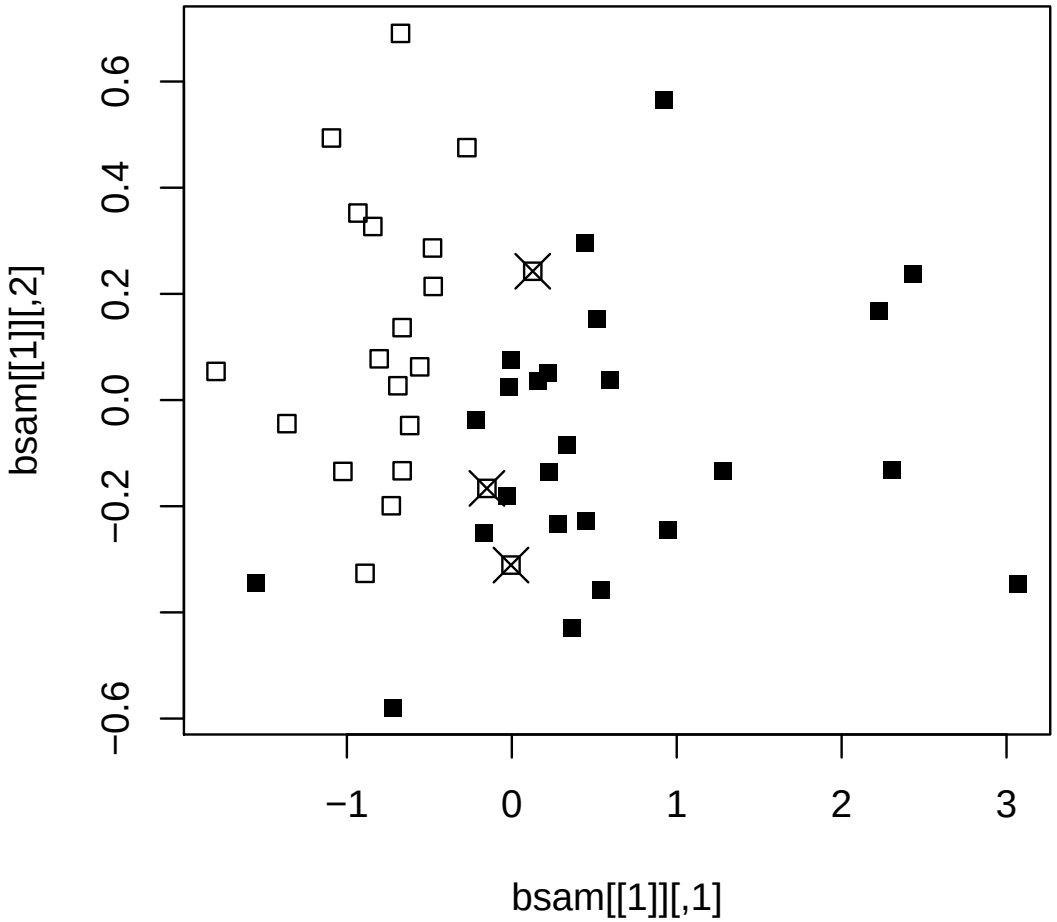
Bank data: the resulting tree

```
> banktree=prune(bankfull,cp=0.067)
> plot(banktree,margin=0.1)
> text(banktree)
```



Bank data: Sammon map and tree predictor

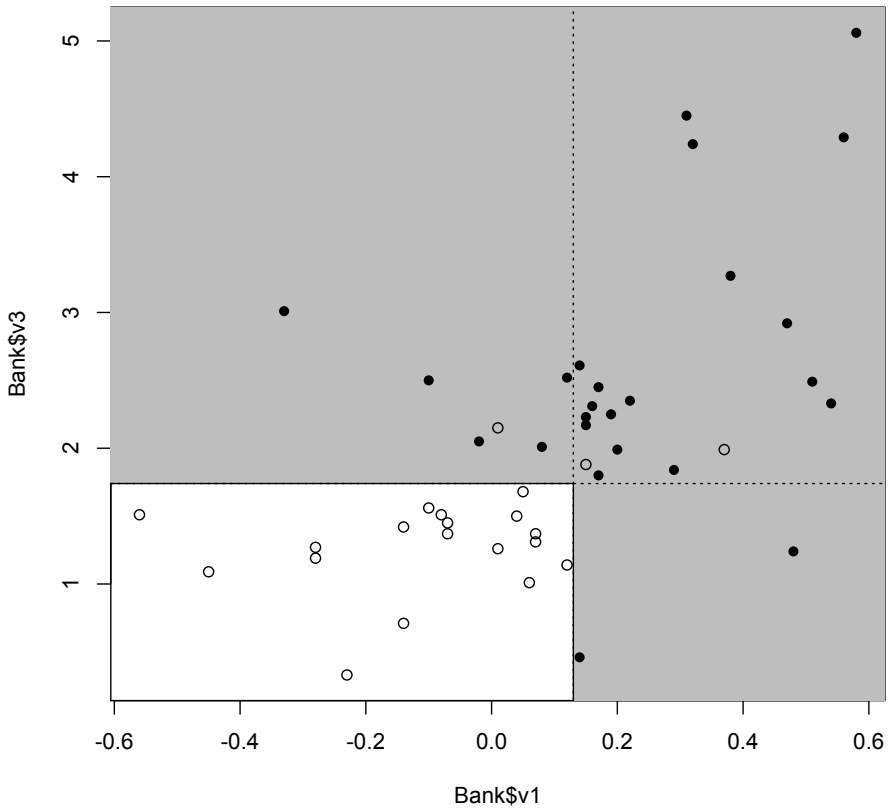
```
> plot(bsam[[1]],pch=15*bank$k)
> wrong=predict(banktree,type='class') != bank$k
> points(bsam[[1]][wrong,],pch=4,cex=2)
```



Bank data: better plot here

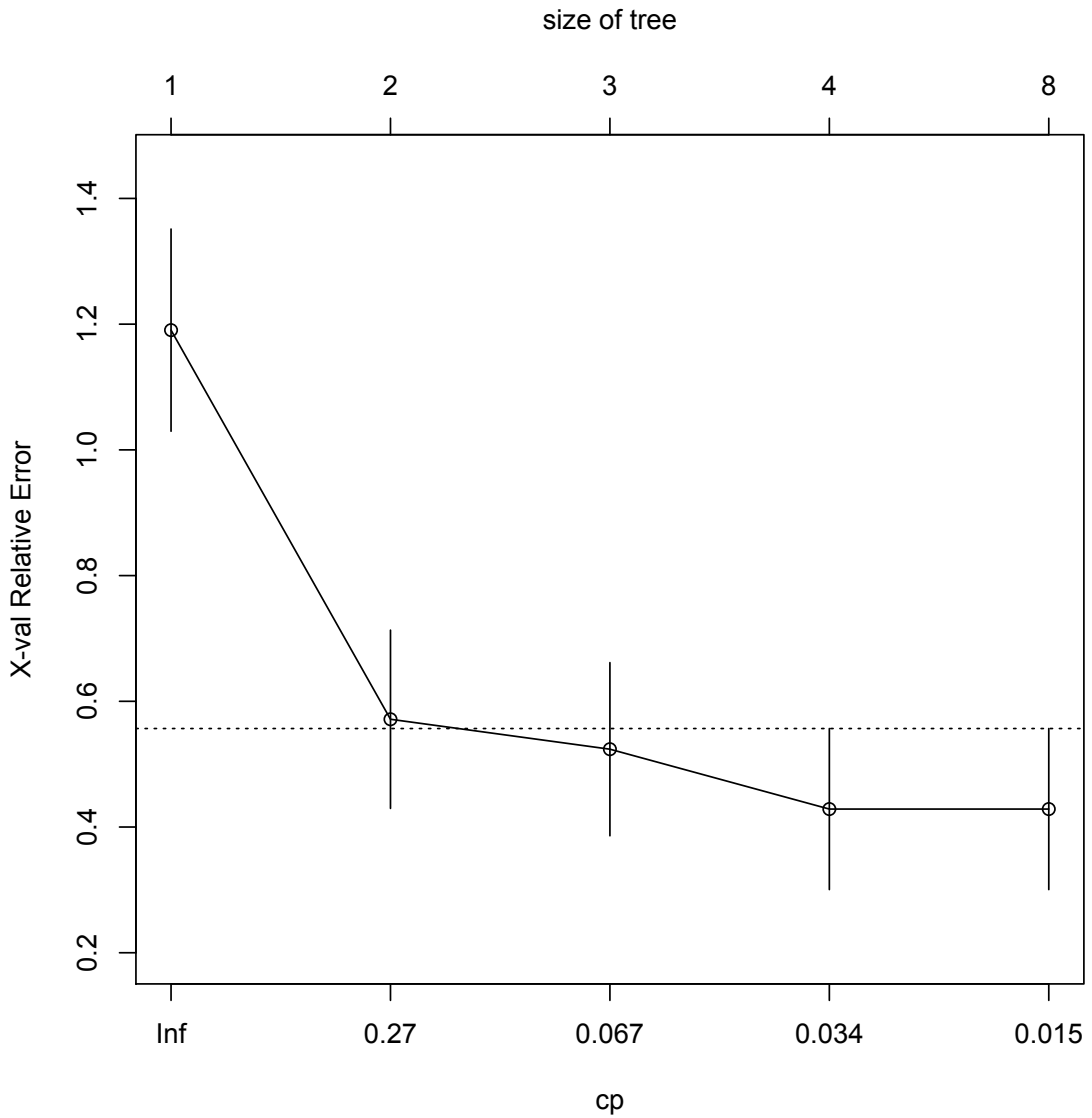
...valid in this particular situation; also some beautifications.

```
> plot(bank$v1,bank$v3,pch=15*bank$k+1,type='n')
> polygon(c(-0.7,0.13,0.13,0.7,0.7,-0.7),c(1.74,1.74,0,0,6,6),col='gray')
> points(bank$v1,bank$v3,pch=15*bank$k+1)
> abline(h=1.74,lty=3)
> abline(v=0.13,lty=3)
```



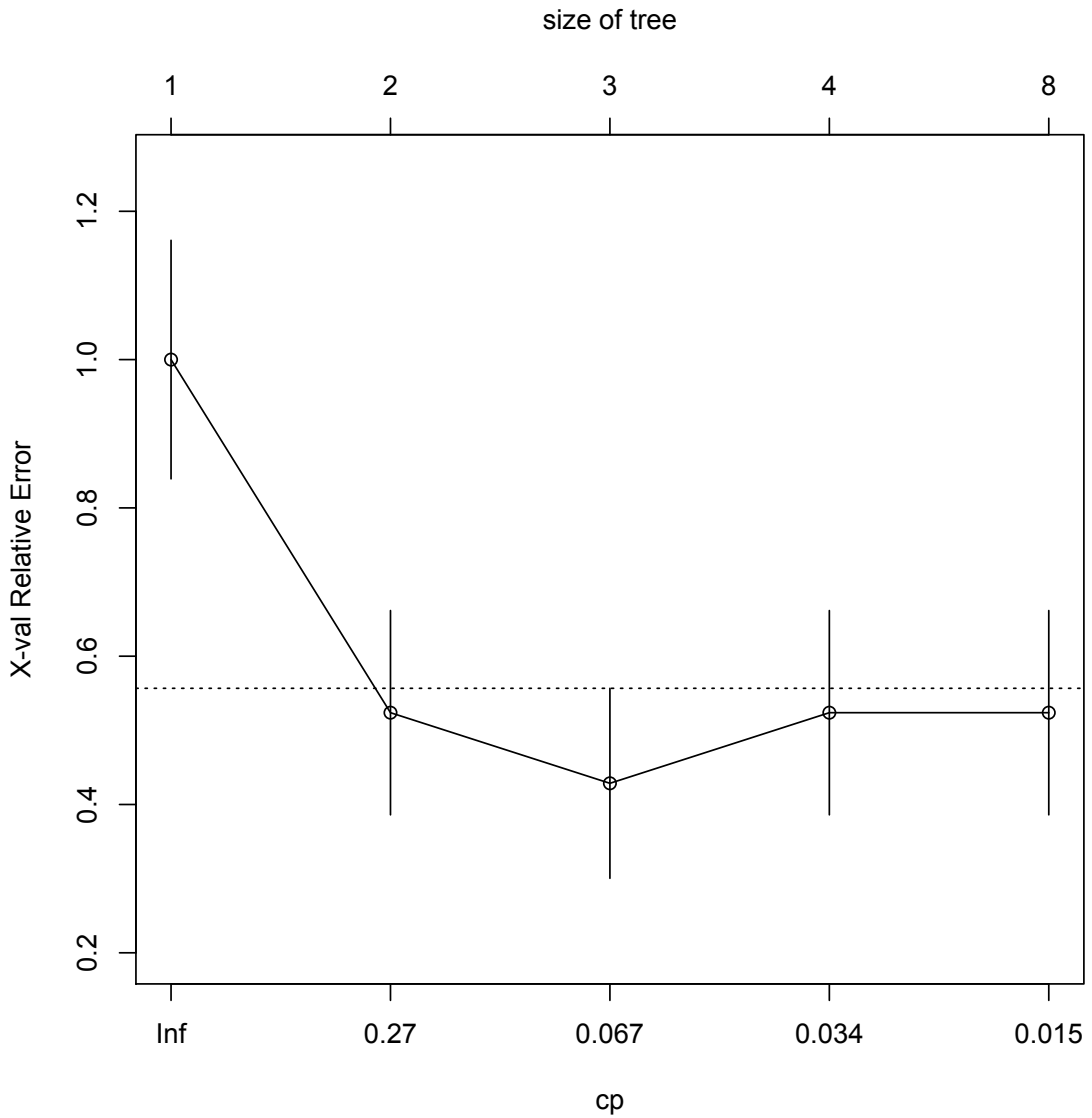
Beware, however

```
> bankfull=rpart(factor(k)~.,data=bank,minsplit=1)
> plotcp(bankfull)
```



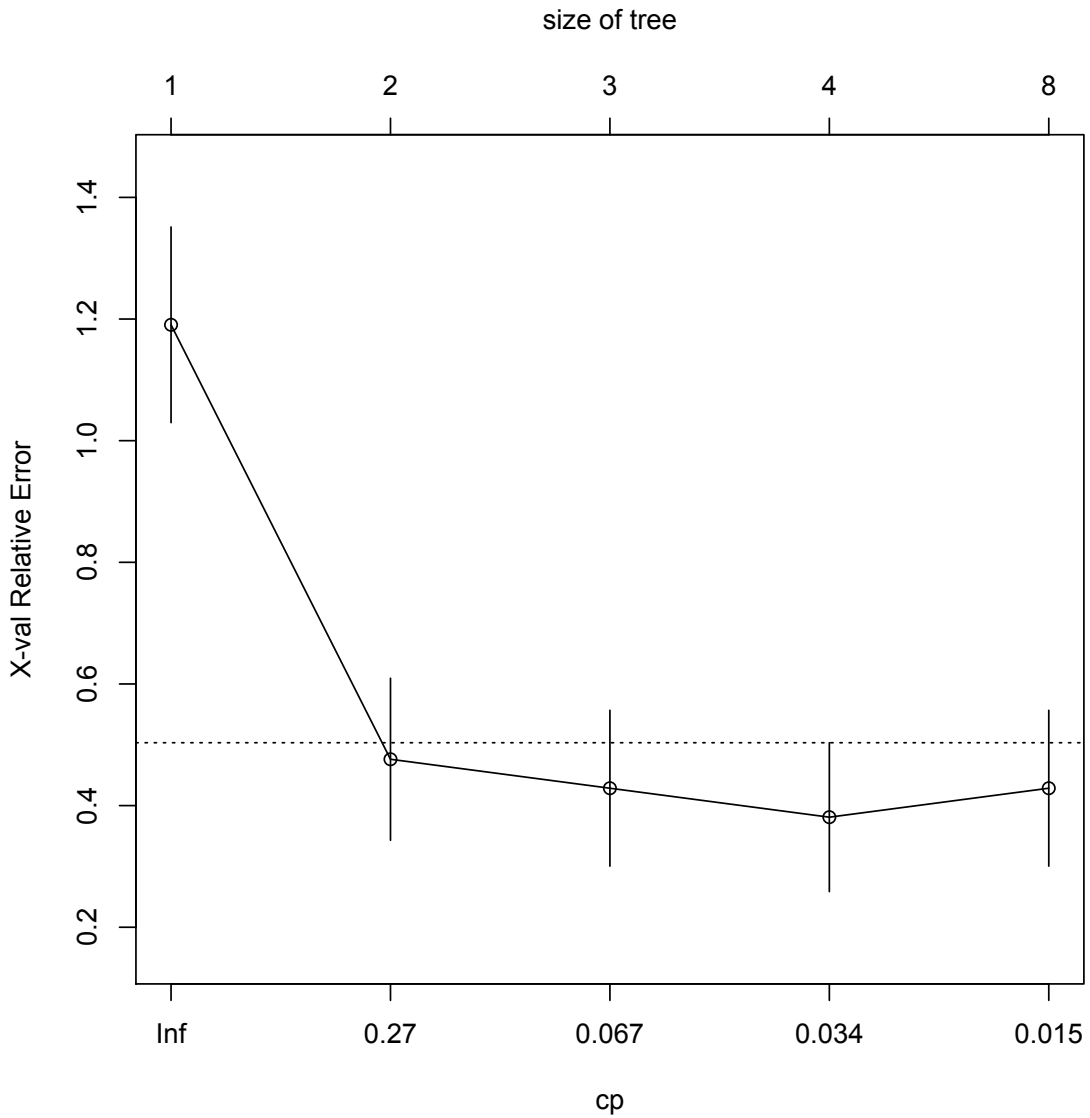
At least the final tree seems to remain...

```
> bankfull=rpart(factor(k)~.,data=bank,minsplit=1)
> plotcp(bankfull)
```



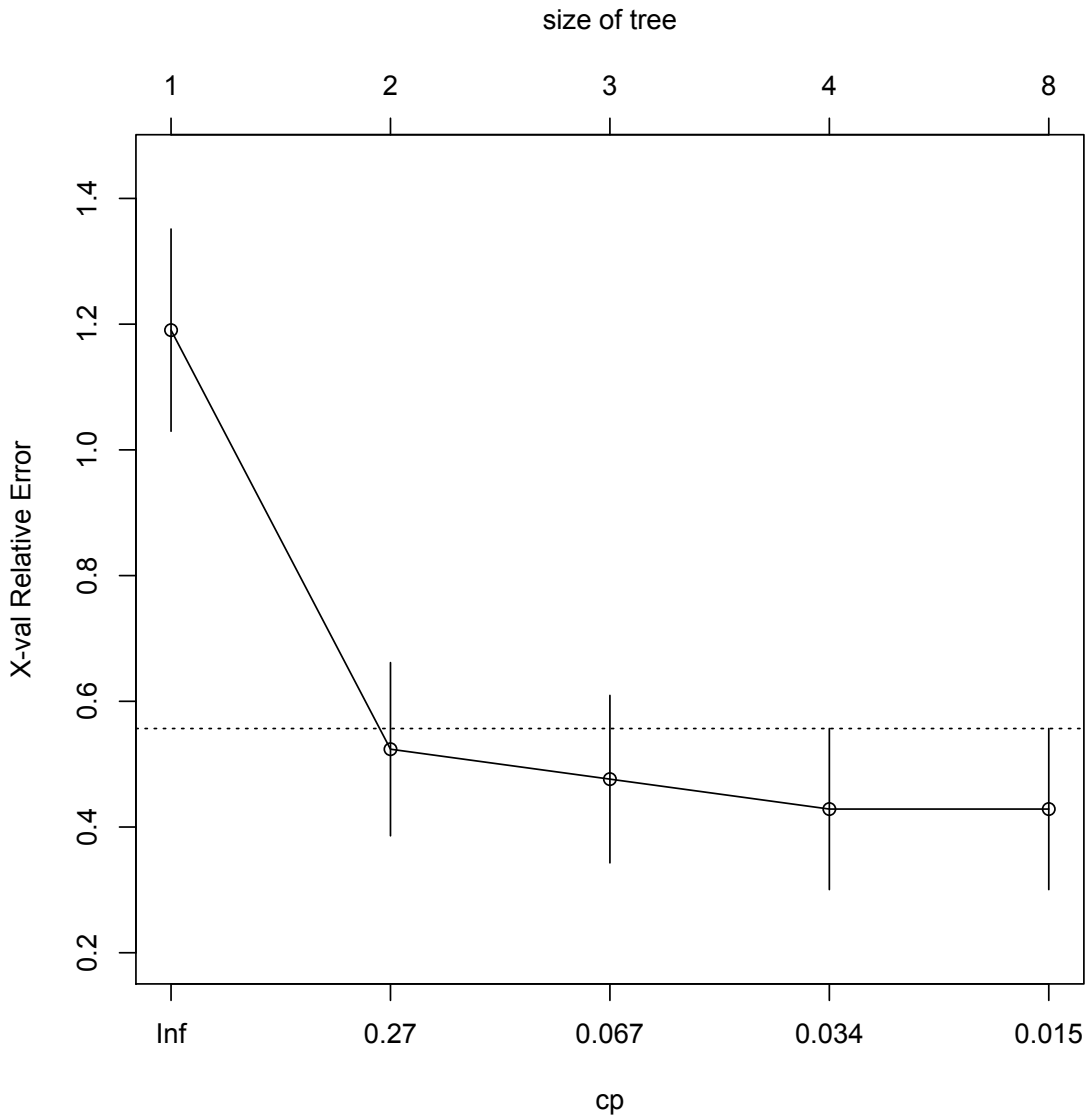
...most of the time...

```
> bankfull=rpart(factor(k)~.,data=bank,minsplit=1)
> plotcp(bankfull)
```



...maybe...

```
> bankfull=rpart(factor(k)~.,data=bank,minsplit=1)
> plotcp(bankfull)
```

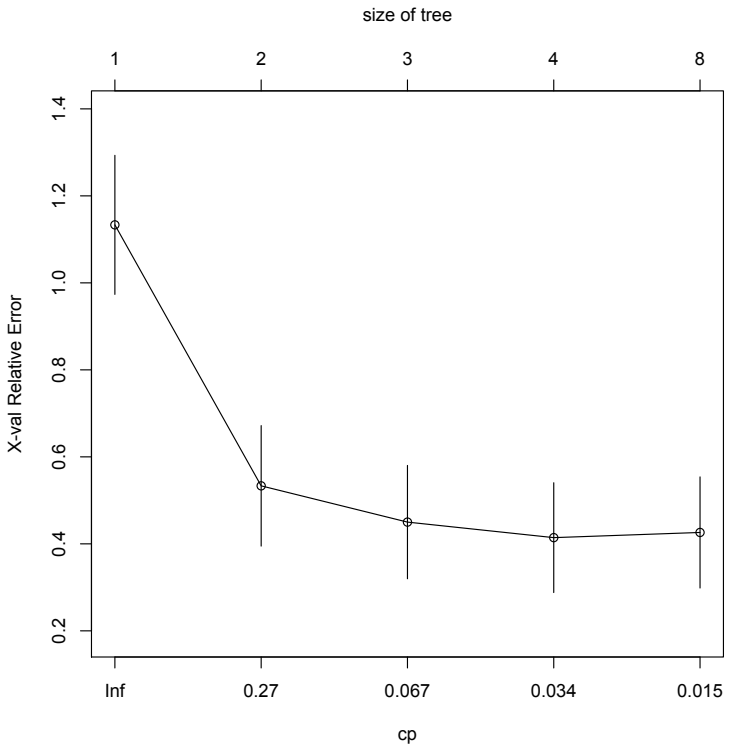


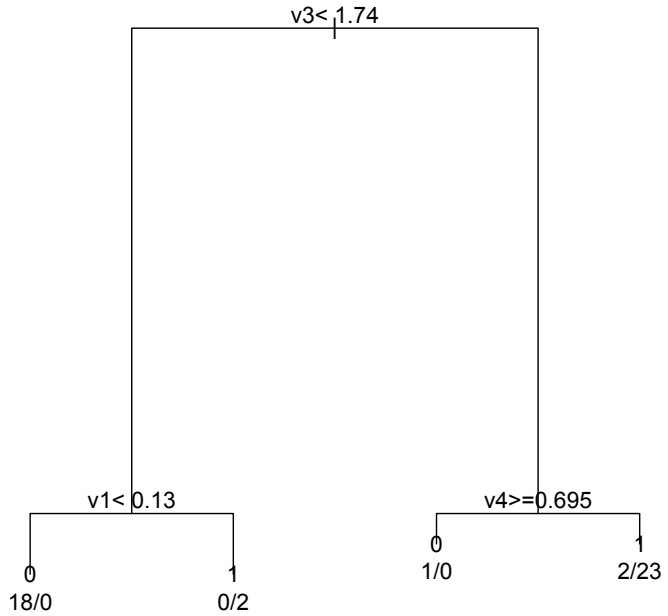
So, it depends only on chance?

Well... then perhaps this:

```
> xer=rep(0,5)
> xvr=rep(0,5)
> for (nn in (1:20)) {
+ bktr=rpart(factor(k)~.,data=Bank,minsplit=1)
+ xer=xer+bktr$cptable[,4]
+ xvr=xvr+bktr$cptable[,5]^2
+ }
> bktr$cptable[,5]=sqrt(xvr/20)
> bktr$cptable[,4]=xer/20
> plotcp(bktr,minline=FALSE)
> bktr=rpart(factor(k)~.,data=Bank,minsplit=1,cp=0.034)
> plot(bktr,margin=0.1)
> text(bktr,use.n=TRUE)
```

The result

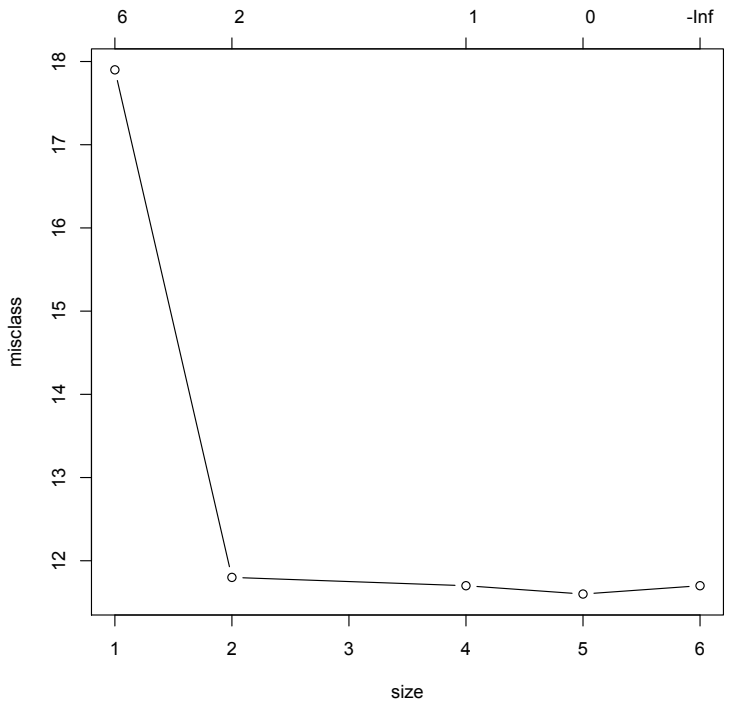

size of tree
1
2
3
4
8
X-val Relative Error
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Inf
0.27
0.067
0.034
0.015
cp

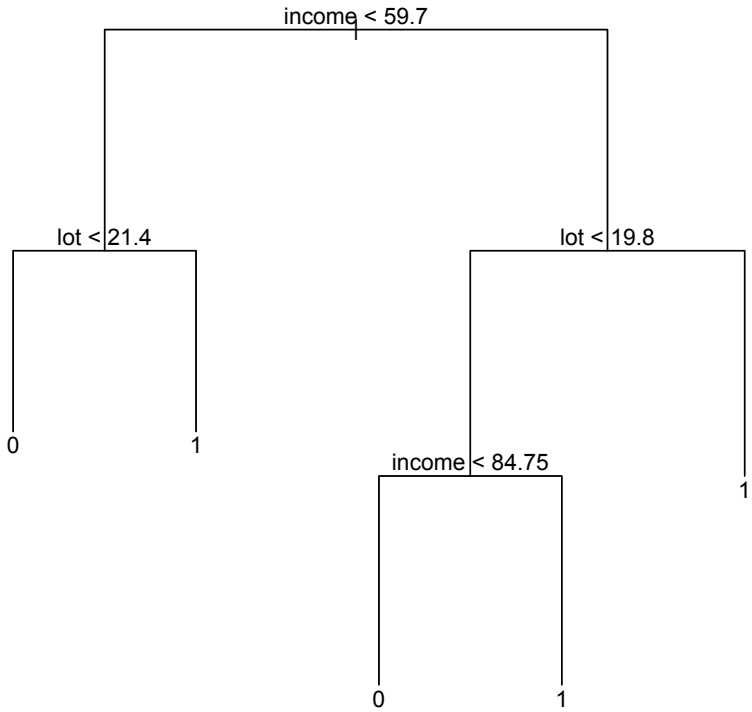

v3< 1.74
v1< 0.13
v4>=0.695
0
18/0
1
0/2
0
1/0
1
2/23

Let us try it on `cv.tree` and riding mowers

```
> xcv=rep(0,5)
> for (nn in (1:20)) {
+ mowcv=cv.tree(mowtr,FUN=prune.misclass)
+ xcv=xcv+mowcv$dev
+ }
> mowcv$dev=xcv/20
> plot(mowcv,type="b")
> plot(prune.misclass(mowtr,best=5))
> text(prune.misclass(mowtr,best=5))
```

Seems like it works





Conclusions

As far as classification trees are concerned:

- they are very interpretable
- have very good predictive power
- however, they are somewhat subjective
- not good for handling linear combinations
- but, on the other hand, they are very good for missing data

  this is in fact their very special virtue: they are capable of producing so-called surrogate splits

Appendix

Prof Brian Ripley ripley at stats.ox.ac.uk
Wed May 4 18:04:02 CEST 2005

- Previous message: [R] Difference between "tree" and "rpart"
- Next message: [R] error with the function GOHyperG from GOstats package
- **Messages sorted by:** [date] [thread] [subject] [author]

```
rpart does much more at C level, including pruning and cross-validation so
can be much faster.

It is also user-extensible.

tree was actually written to track down bugs in the then S implementation,
and so is much closer to the functionality in S.  It is not where I would
have started from.  It is really only available for R to support MASS and
PRNN (my books).

On Wed, 4 May 2005, Dr Carbon wrote:

> In the help for rpart it says, "This differs from the tree function
> mainly in its handling of surrogate variables." And it says that an
> rpart object is a superset of a tree object. Both cite Brieman et al.
> 1984. Both call external code which looks like martian poetry to me.
>
> I've seen posts in the archives where BDR, and other knowledgeable
> folks, have said that rpart() is to be prefered over tree()
>
> Is there a simple reason why? They use the same fundamental algorithm.
> Are there differences in processing time? Bells and whistles?

--
Brian D. Ripley,                  ripley at stats.ox.ac.uk
Professor of Applied Statistics,  http://www.stats.ox.ac.uk/~ripley/
University of Oxford,             Tel:  +44 1865 272861 (self)
1 South Parks Road,                     +44 1865 272866 (PA)
Oxford OX1 3TG, UK                Fax:  +44 1865 272595
```

Classification via recycling

Methods recycling other methods

An important (and perhaps the most important) class of classification methods are those that use or combine other methods: "recycling", "meta-", or "committee" methods

Prominent examples: bagging, boosting

Committee methods

We have seen that instability of algorithms can be mitigated by their averaging. This is done in a straightforward manner: the j-th classifier classifies given $\mathbf{x}$ into class y_j. "Committee" than decides by the majority vote: that class is chosen as a final results which has got maximal number of the y_j's

When there are only two classes, we can have one of them coded by 0 and another by 0; then the majority vote is about whether the average

$$\frac{1}{n}\sum_{j=1}^{J} y_j \quad \text{is} \quad \geqslant \text{ or } \leqslant \text{ than } 0.5$$

(As usual, $=$ does not matter from the theoretical point of view)

Here, "different classifiers" can be different methods, different runs of the same numerically unstable algorithm on the same data, runs of the same or different algorithms on different data - and regarding the last possibility, "different" data created by recycling the same dataset

Bootstrap AGGregation or bagging

The J "different" datasets are generated from the n original datapoints $(\mathbf{x}_i, y_i)$ by "bootstrap sampling": n new datapoints are sampled from the original ones, with replacement and the same probability of being sampled, for every $j = 1, 2, \ldots, J$.

Each of these new datasets creates a classifier that eventually classifies given $\mathbf{x}$ into class y_i. The combined classifier than takes the majority vote. (The voting proportions may serve as estimates of posterior probabilities)

In each of the samples, some of the $(\mathbf{x}_i, y_i)$ can occur once, some several times, and some (with probability about 0.37) not occur at all. The later are called "out of bag" for the j-th sample and they function as a test sample for the j-th case.

More specifically, for each $(\mathbf{x}_i, y_i)$ we take only those samples out of J where $(\mathbf{x}_i, y_i)$ is "out of bag", that is, when it is not sampled. We classify $\mathbf{x}_i$ based on the classifiers ran *only* on those samples, using the majority rule. Then we look on the average number of misclassifications over all $\mathbf{x}_I$, $i = 1, 2, \ldots, n$

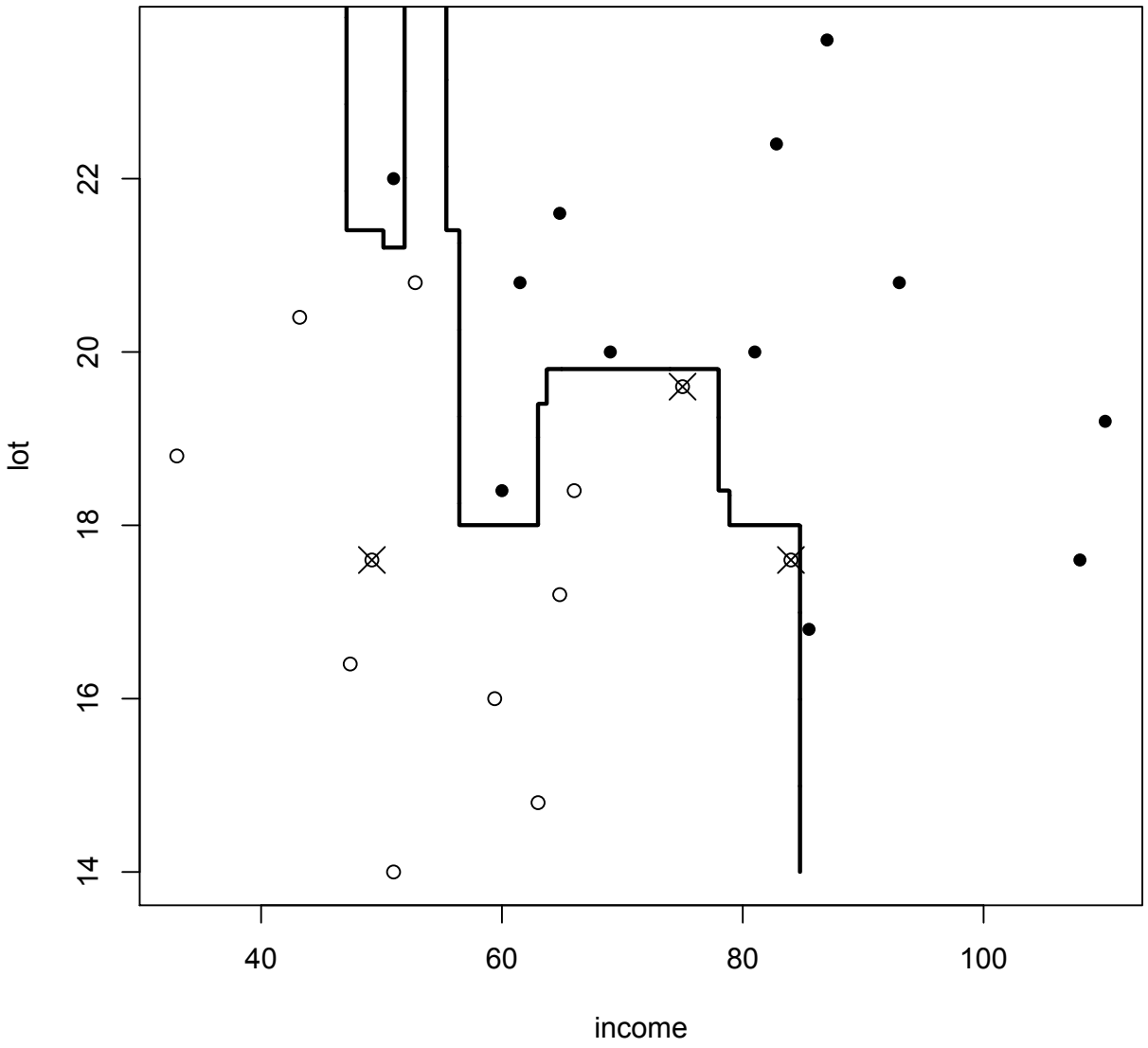
Recycling classification trees

When doing bagging with classification trees, it is not feasible to involve subjective input for all J samples; instead we construct the tree automatically - no pruning, mostly all the way down.

A further elaboration of this technology are *random forests*: randomization is applied not only in resampling, but also in the construction of the tree. Typically, at each stage of finding the split, a random sample of variables is taken and the optimal (Gini or entropy) split is determined based on those

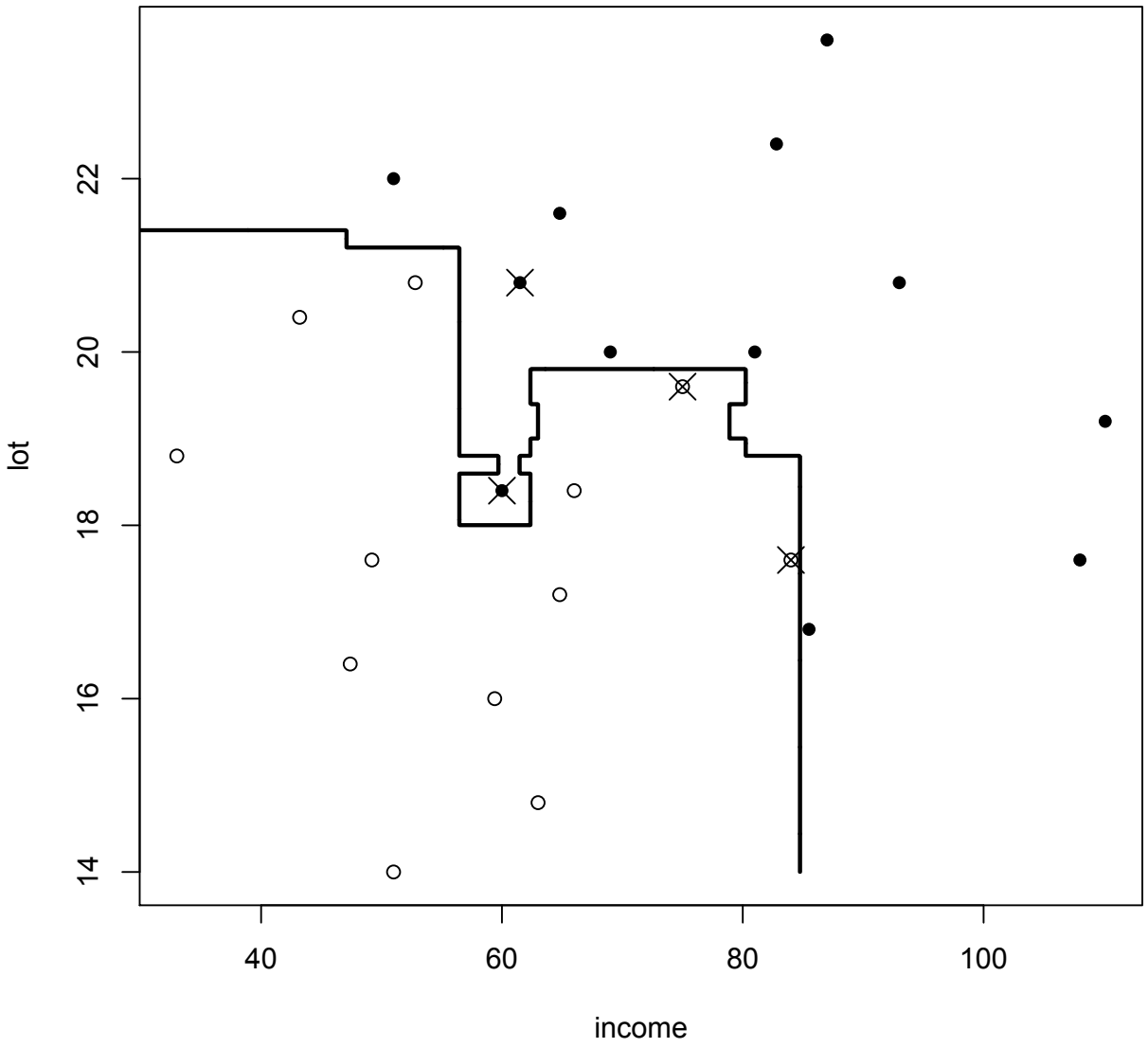
```
> library(randomForest)
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
> set.seed(321)
> mow.rf=randomForest(factor(riding)~income+lot,data=mowers)
> predict(mow.rf,newdata=mowers)
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
Levels: 0 1
> predict(mow.rf)
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 0  0  1  1  1  1  1  1  0  1  0  1  1  1  0  0  1  0  0  1  0  0  0  0
Levels: 0 1
```

Bagging (50) mowers



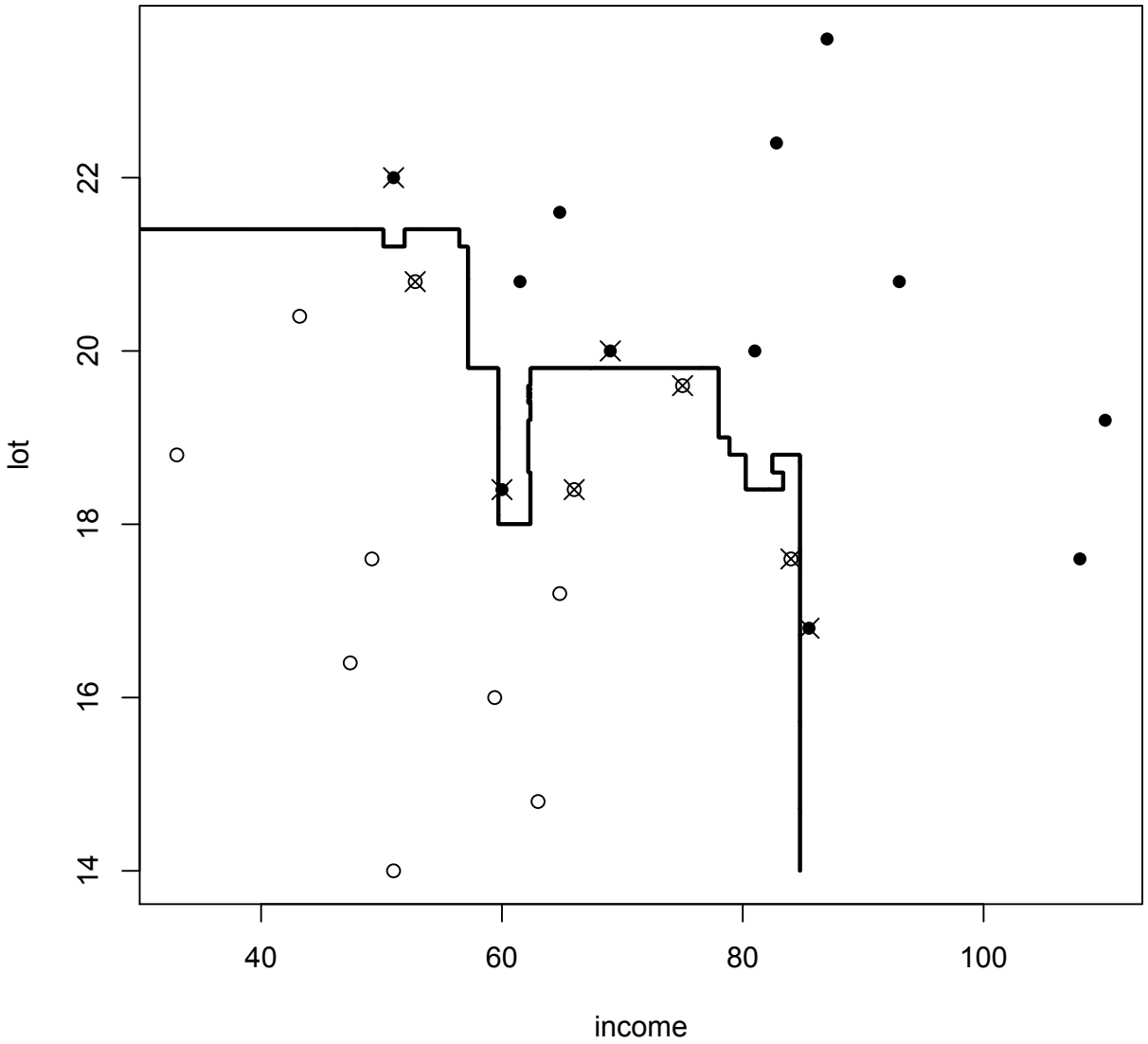
Apparent error rate is 0, but the "out-of-bag" estimate is 3/24 (X)

Bagging (50) mowers repeated



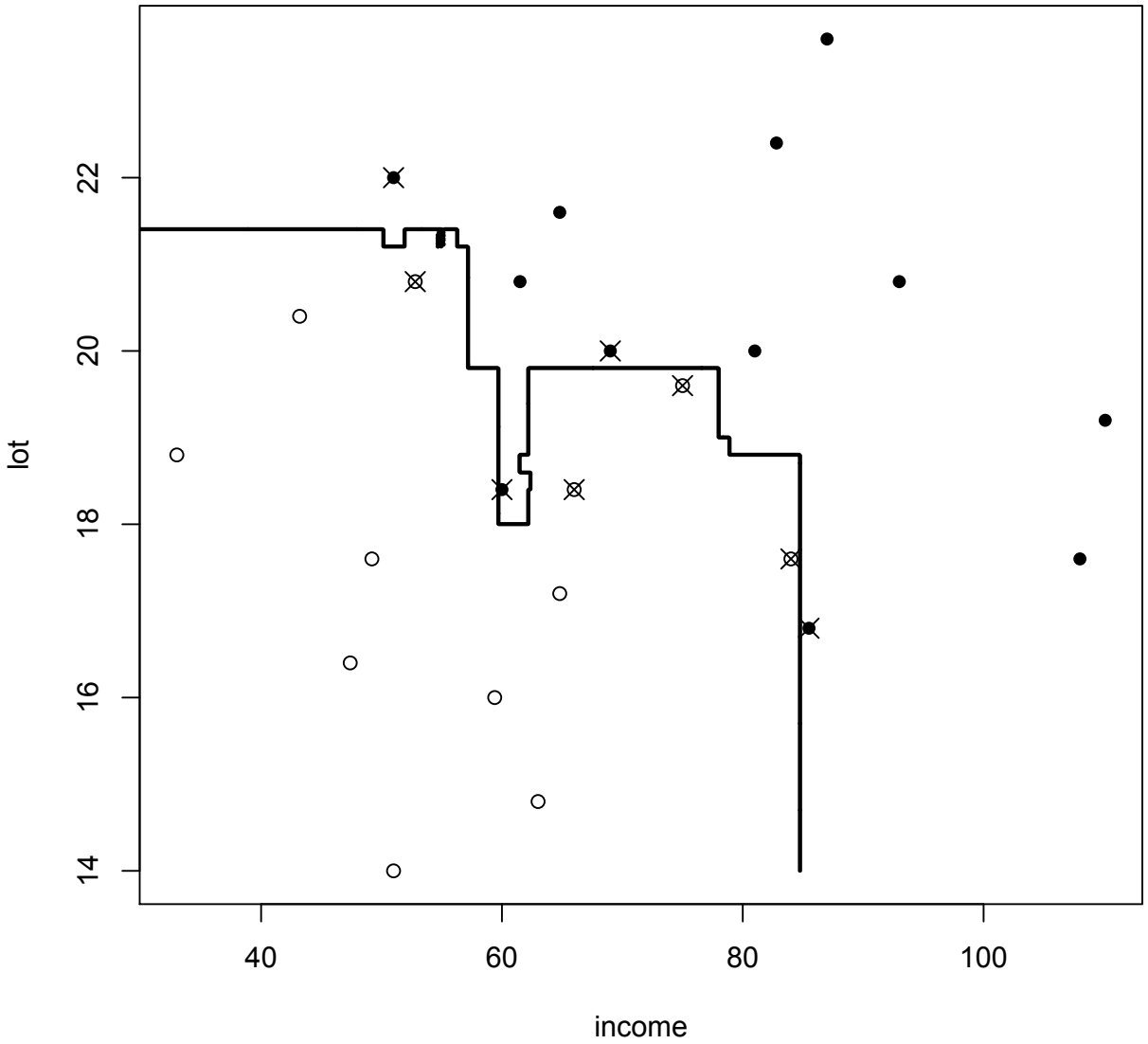
Apparent error rate again 0, the "out-of-bag" estimate is 4/24 (X)

Random forest with 500 mowers



Apparent error rate 0; "out-of-bag" misclassifications shown by X

Random forest with 500 mowers once again



Apparent error rate 0; "out-of-bag" misclassifications shown by X

Boosting

An algorithm that takes a classification method and repeatedly applies it to the reweighted data points - with the objective to get improved classification

1. Apply a classifier, store the result.
2. Reweight items: the misclassified ones receive higher weight
3. Go to point 1 and repeat.

After stopping:

4. Combine all the results obtained on the road.

AdaBoost.M1

Suppose that the two classes are coded -1 and $+1$.

1. Initialize weights to $w_i = 1/n$ (start with equal weights)

2. For $k = 1$ to K

(a) Classify training data with weights w_i; predictions are $G_k(x)$

(b) $I_i = 1$, if i-th item not classified correctly, and 0 if yes

(c) $e_k = \dfrac{\sum_{i=1}^{n} w_i I_i}{\sum_{i=1}^{n} w_i}$ (this is the quantity indicating how well the items are classified for this particular run – it depends on their weights, but otherwise it is independent of i)

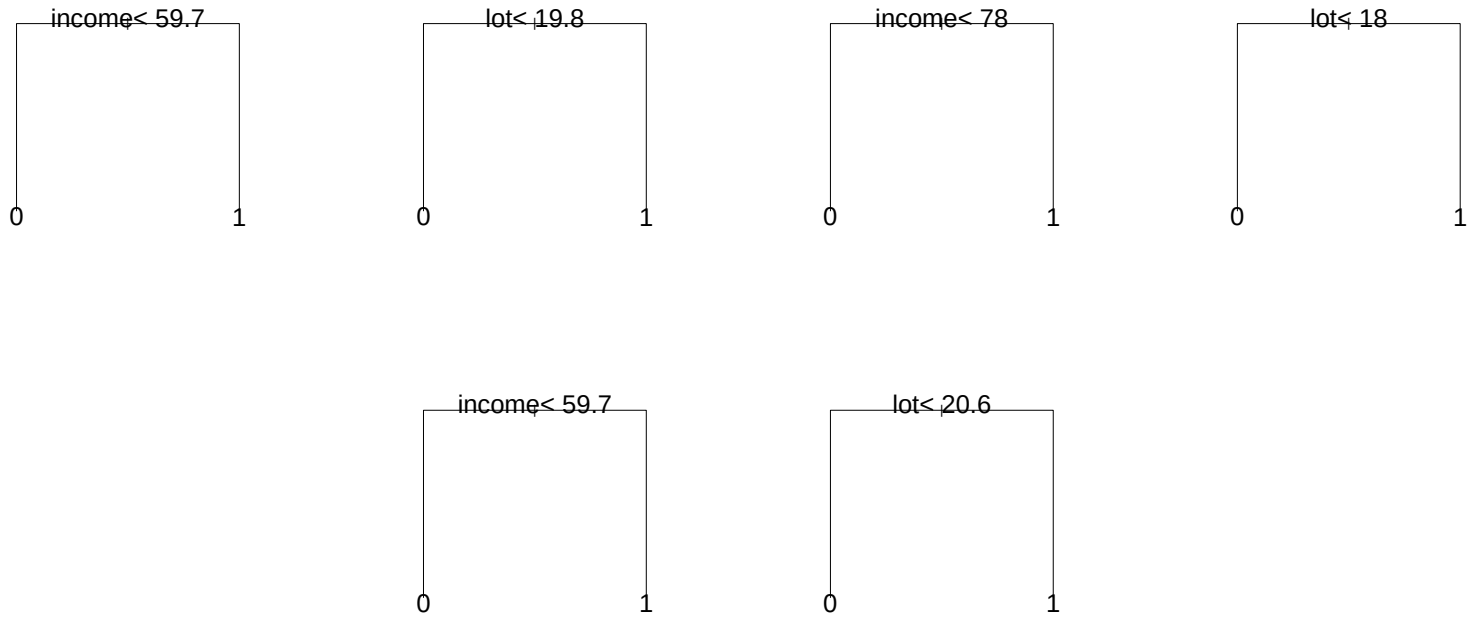
(d) let $\alpha_k = \log \dfrac{1 - e_k}{e_k}$ set the new i-th weight to be $w_i e^{\alpha_k I_i}$ that is, if the item is classified correctly no change of weight; if not, then the new weight is $\dfrac{1 - e_k}{e_k} w_i$; greater than the old one if less than the half (in total weights) items is misclassified

3. After the loop, the final prediction is: sign $\left(\sum_{k=1}^{K} \alpha_k G_k(x) \right)$

A realization for riding mowers data

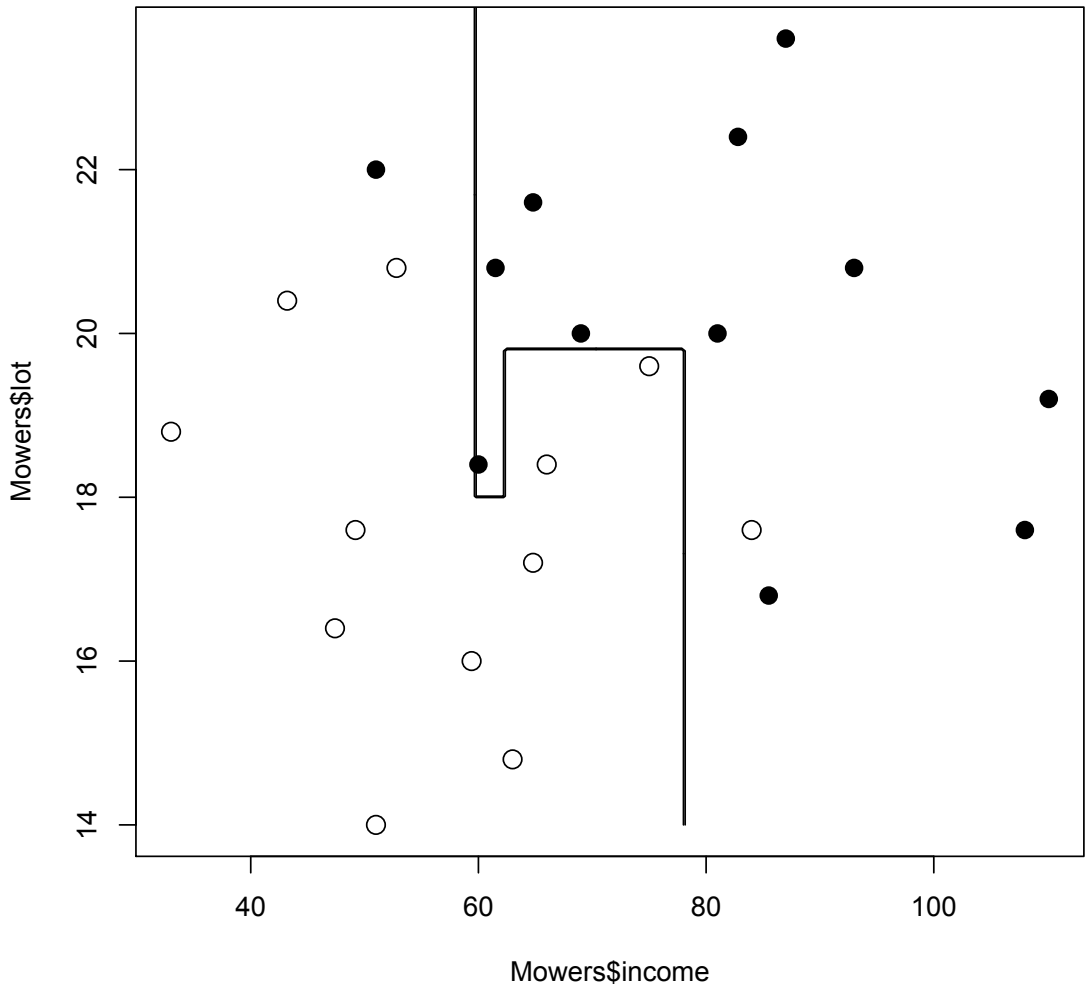
```
> boowei=rep(1/nrow(Mowers),nrow(Mowers))
> boocla=vector("list",33)
> booalp=rep(0,33)
> for (k in 1:33)
+ {
+ boocla[[k]] = rpart(factor(riding)~income+lot,
+ data=Mowers,weights=boowei,maxdepth=1)
+ boopr = predict(boocla[[k]])
+ boomis = (Mowers$riding != as.numeric(boopr[,2] > boopr[,1]))
+ booerr = crossprod(boowei,boomis)/sum(boowei)
+ booalp[k] = log((1-booerr)/booerr)
+ boowei = boowei*exp(booalp[k]*boomis)
+ print(k)
+ }
```

The classifier used: stumps



(trees with only one split)

The result



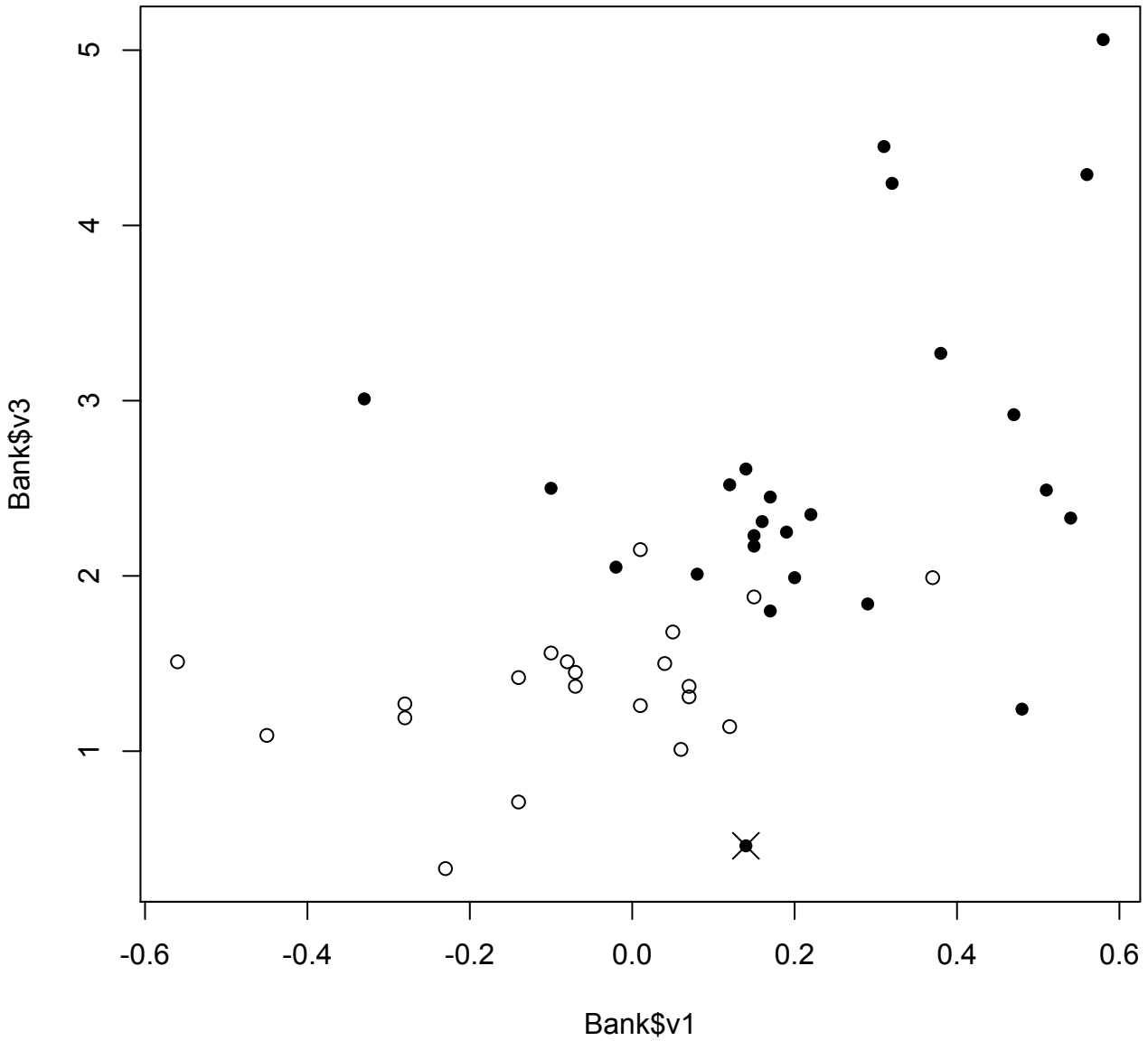
"The best off-shelf classifier (boosting with trees)"

Catch: do you know when to stop? (= hidden tuning parameter)

Finally, bank data (as usual)

```
> boowei=rep(1/nrow(Bank),nrow(Bank))
> boocla=vector("list",15)
> booalp=rep(0,15)
> for (k in 1:15)
+ {
+ boocla[[k]] = rpart(factor(k)~.,data=Bank,
+ weights=boowei,maxdepth=1)
+ boopr = predict(boocla[[k]],type='class')
+ boomis = (Bank$k != boopr)
+ booerr = crossprod(boowei,boomis)/sum(boowei)
+ booalp[k] = log((1-booerr)/booerr)
+ boowei = boowei*exp(booalp[k]*boomis)
+ }
> boocum=rep(0,nrow(Bank))
> for (k in 1:15)
+ {
+ boocum = boocum + booalp[k]*2
+ *(predict(boocla[[k]])[,2] > 0.5)-1
+ }
> wrong=(1+sign(boocum))/2 != Bank$k
> plot(Bank$v1,Bank$v3,pch=15*Bank$k+1)
> points(Bank$v1[wrong],Bank$v3[wrong],pch=4,cex=2)
```

Boosting bank data



Classification via regression revisited: neural networks

Neural networks: some principles

A somewhat actual way of specifying nonlinear models is via superposition of functions - *neural networks*. The simplest variation on this theme is the "feed-forward one hidden layer neural network"

$$f(x) = \varphi_0\left(\alpha + \sum_h \beta_h \varphi_h\left(\alpha_h + \sum_i \beta_{ih} x_i\right)\right).$$

As a rule, the "hidden layer activation functions" φ_h are logistic; the output function φ_0 is in classification logistic or treshold ($\varphi_0(x) = I(x > 0)$). (Neural networks may be used also for usual regression with quantitative response, where φ_0 may be also linear.) Estimation ("training") of α's and β's is usually done by least squares method.

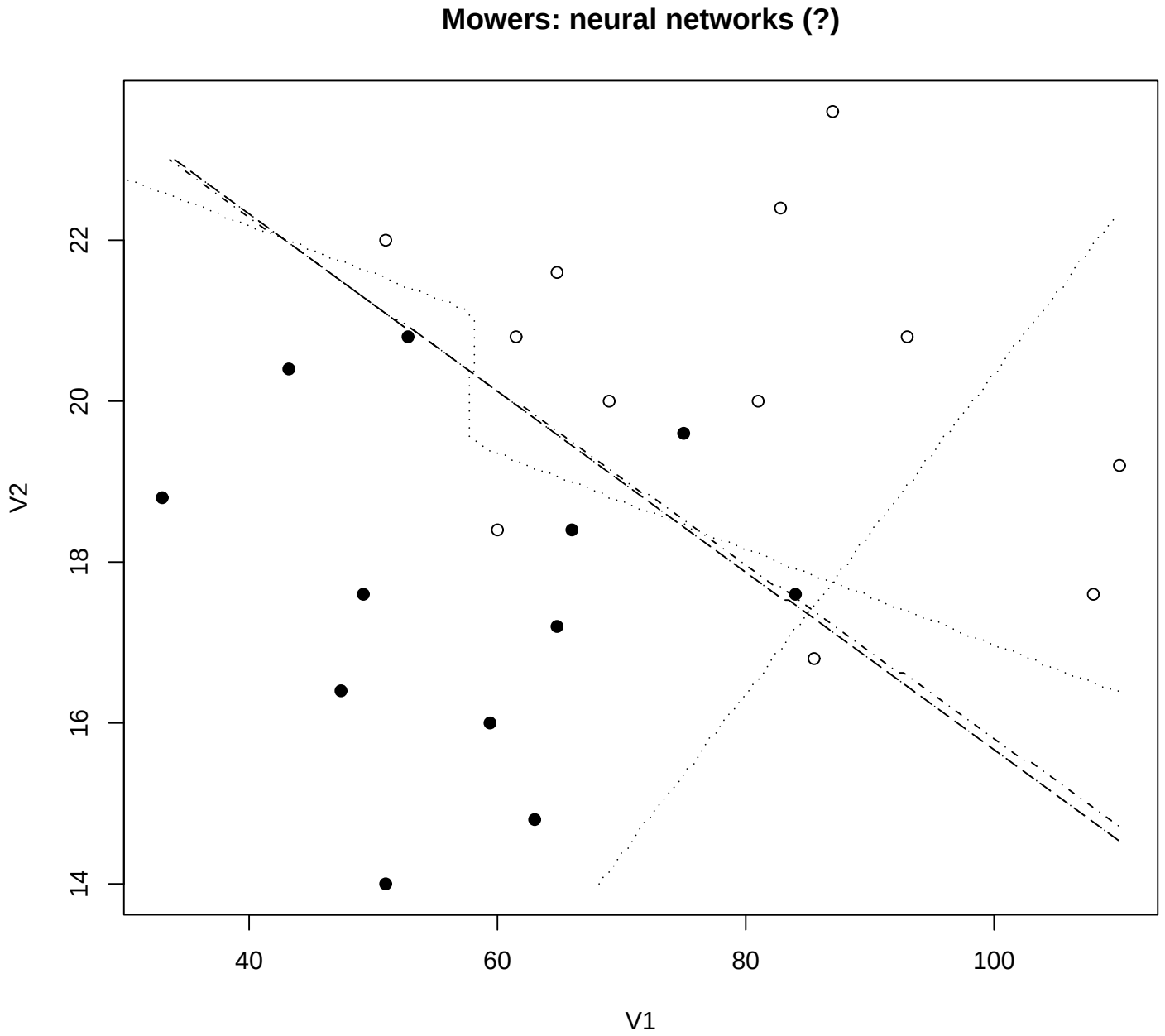
Expressed graphically

Neural networks: some details

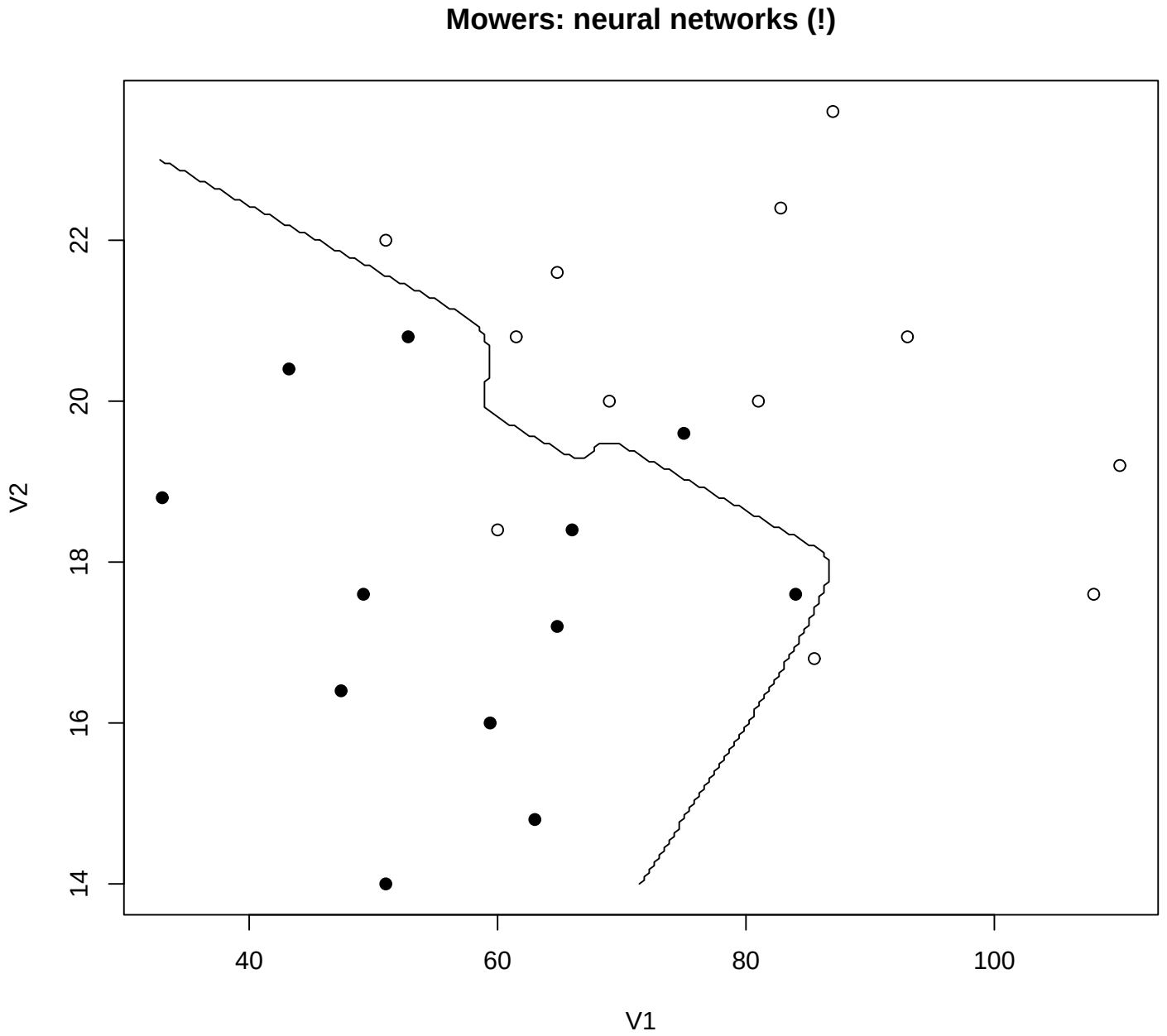
The number of layers and hidden units is chosen in advance. The number of parameters grows quickly, so the main concern is overfitting: the classification fits very well the training data, but performs poorly on the next sample. There are techniques proposed to cope with this; in particular, splitting to training and validation sample almost a must (leave-one-out cross-validation is in most cases computationally too expensive).

The algorithmic side is nontrivial; methods often lead to different local minima, so alternative predictive strategies (averaging) have to be used.

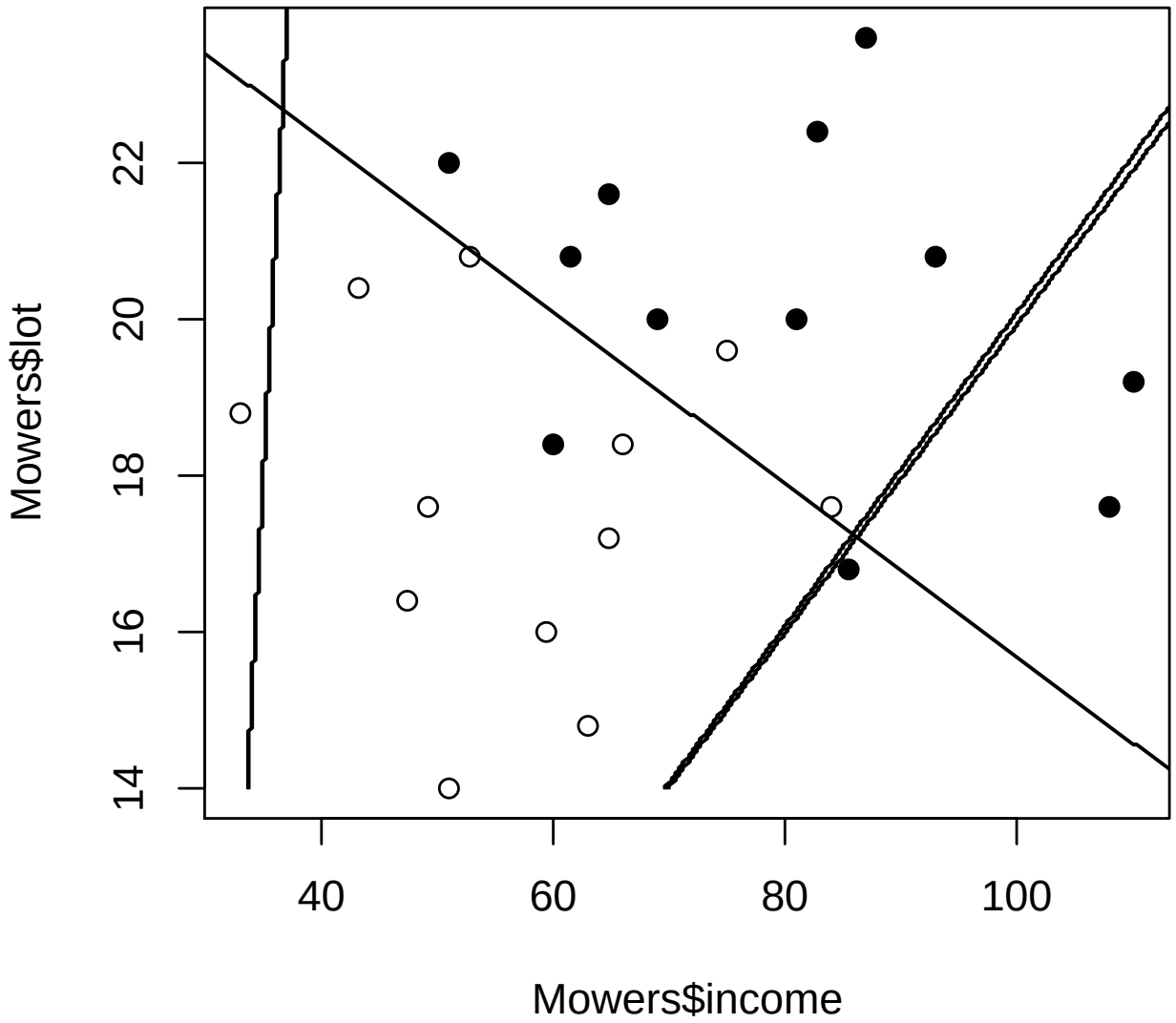
Mowers: neural networks



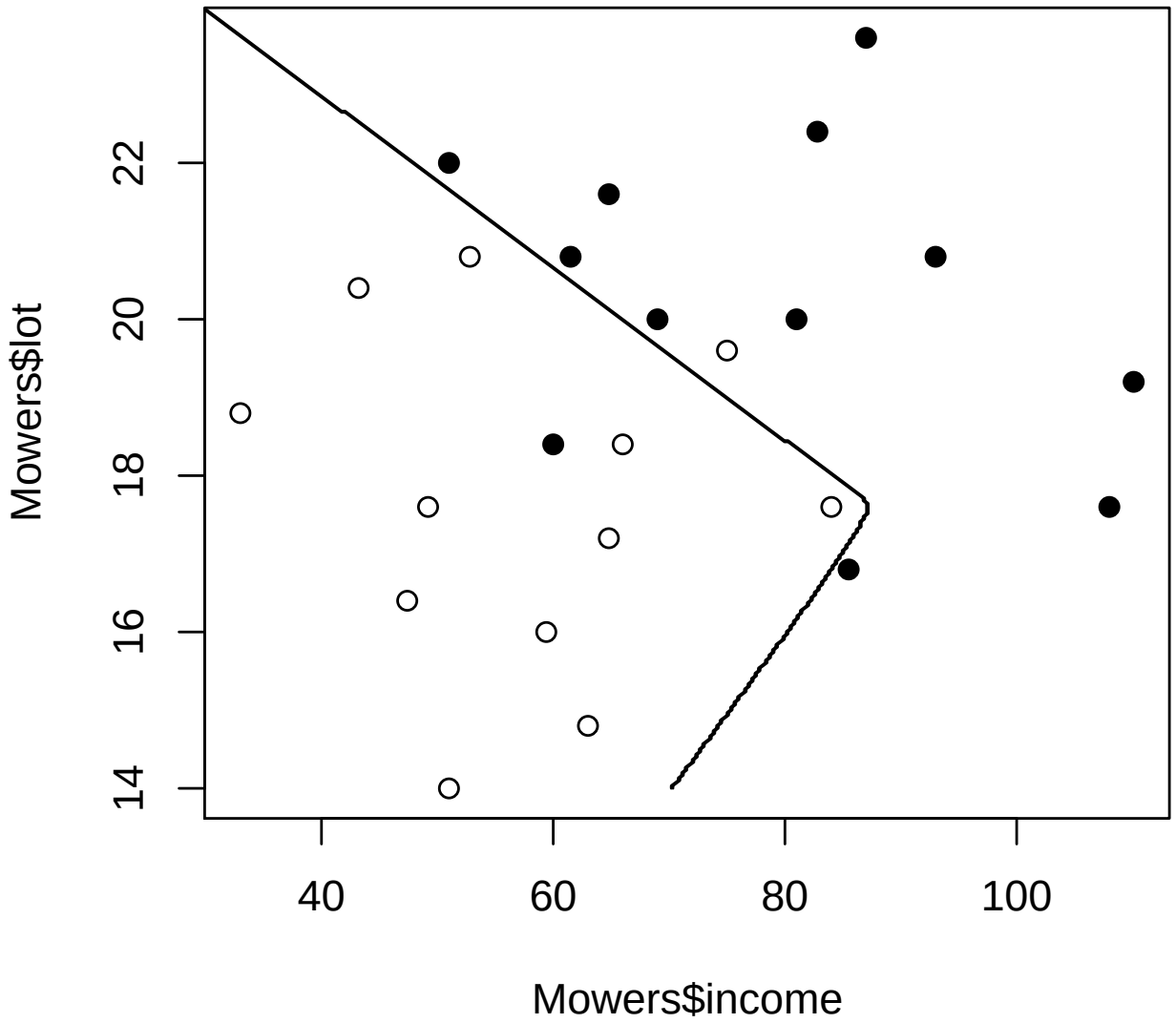
Mowers: averaged neural networks



Reproducibility?



Mowers: averaged neural networks again



Final remarks

Neural networks offer a lot of flexibility, but they are not that transparent regarding their nature.

An objective to find a classifier modeling somewhat human perception is probably lost.

What remains are serious algorithmic problems.

Nevertheless, who knows...(?)

Finale: properties of classification methods I

Predictive power: how well the method classifies?

- all methods compete for that, and new ones are introduced because they (are supposed to) perform better than old ones
- but this does not mean that the old ones are necessarily worse (LDA, for instance)

Interpretability: can we make some sense out of the rule?

- some methods are really of black box type: neural networks
- but of course, interpretability may be not the most important virtue

Number of classes capable to handle: many or only two?

- there are multiclass version of linear discriminant analysis and also of logistic regression (multinomial regression)

Properties of classification methods II

Computational difficulty: feasible also for large datasets?

- this changes with time: new algorithms, new hardware
- is there a need to tune parameters (often a hidden feature)?

Special aspects

- handling of missing data
- adaptivity to unequal priors and misclassification costs
- handling of linear combinations
- type of regions (linear boundaries? more fancy?)
- invariance with respect to transformations?
- robustness to outliers?