Lecture 9: Non-linear Classification Rules

Alan Lee

Department of Statistics
STATS 784 Lecture 9

August 20, 2017

Outline

Today's agenda

In this lecture we continue our discussion of classification methods, concentrating on classification versions of the methods we covered in Lecture 4. Topics will be

- Additive models (gams)
- Neural networks
- Trees and random forests
- Support vector machines

First, we discuss two datasets.

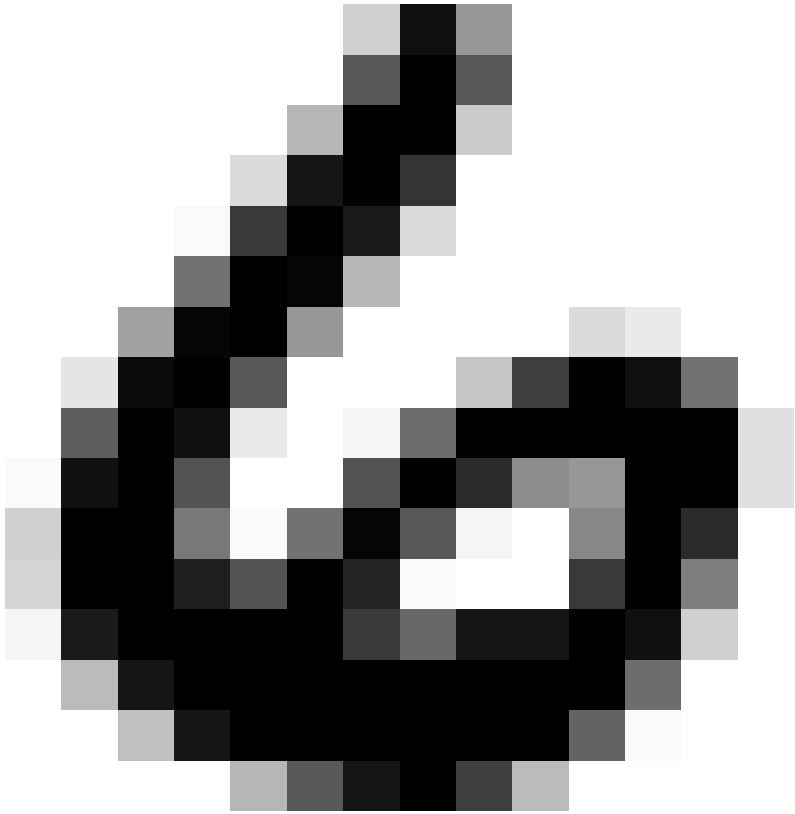
Example: Zip codes

- In the US, postal districts are labeled by 5-digit ZIP codes (ZIP=Zone Improvement Plan). Letters are supposed to have these codes as part of an address. The US Postal Service has been interested in algorithms for enabling the machine sorting of letters, including character recognition software.
- This example involves 9298 images of individual hand-written digits. Each image is represented by a $16 \times 16 = 256$ array of pixels, each with a gray-scale value, ranging from -1 (white) to 1 (black). Thus, there are 256 explanatory variables. The response is the actual digit.
- The aim is to develop a classification rule to recognise digits on the basis of the grey-scale values.
- A sample row of the data matrix and the corresponding image is shown overleaf:

Sample data

-1 -1 -1 -1 -1 -1 -1 -0.631 0.862 -0.167 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -0.992 0.297 1 0.307 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -0.41 1 0.986 -0.565 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -0.683 0.825 1 0.562 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -0.938 0.54 1 0.778 -0.715 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0.1 1 0.922 -0.439 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -0.257 0.95 1 -0.162 -1 -1 -1 -0.987 -0.714 -0.832 -1 -1 -1 -1 -1 -0.797 0.909 1 0.3 -0.961 -1 -1 -0.55 0.485 0.996 0.867 0.092 -1 -1 -1 -1 0.278 1 0.877 -0.824 -1 -0.905 0.145 0.977 1 1 1 0.99 -0.745 -1 -1 -0.95 0.847 1 0.327 -1 -1 0.355 1 0.655 -0.109 -0.185 1 0.988 -0.723 -1 -1 -0.63 1 1 0.068 -0.925 0.113 0.96 0.308 -0.884 -1 -0.075 1 0.641 -0.995 -1 -1 -0.677 1 1 0.753 0.341 1 0.707 -0.942 -1 -1 0.545 1 0.027 -1 -1 -1 -0.903 0.792 1 1 1 1 0.536 0.184 0.812 0.837 0.978 0.864 -0.63 -1 -1 -1 -1 -0.452 0.828 1 1 1 1 1 1 1 1 1 0.135 -1 -1 -1 -1 -1 -1 -0.483 0.813 1 1 1 1 1 1 0.219 -0.943 -1 -1 -1 -1 -1 -1 -1 -0.974 -0.429 0.304 0.823 1 0.482 -0.474 -0.991 -1 -1 -1 -1

The image



Example: Spam

- As you no doubt know, spam is unwanted email, the electronic equivalent of junk mail.
- Most mail programs have a way of recognizing such messages as spam and diverting them to a junk mail folder.
- This is usually done by recognizing key words and characters that are likely to occur in either genuine messages and spam, but not both. These words will be different for every user.

Example: Spam (cont)

- The data for this example consist of measurements on 4601 email messages. There are 58 variables representing the relative frequency of certain words and characters (e.g. the proportion of words in the message that are "George".
- There are additional variables representing the length of strings of consecutive capital letters, and the response variable is a binary variable (0=genuine message, 1=spam).
- The aim is to develop a classification rule for classifying an email as genuine or spam.

Reading in data

```
# read in zip code data

URL = "https://www.stat.auckland.ac.nz/~lee/760/digits.txt"
zip.df = read.table(URL, header=FALSE)
names(zip.df) = c("digit", paste("V",1:256, sep=""))

# read in spam data

URL = "https://www.stat.auckland.ac.nz/~lee/760/spam.txt"
spam.df = read.table(URL, header=FALSE)
names(spam.df) = c(paste("V",1:57, sep=""),"spam")
```

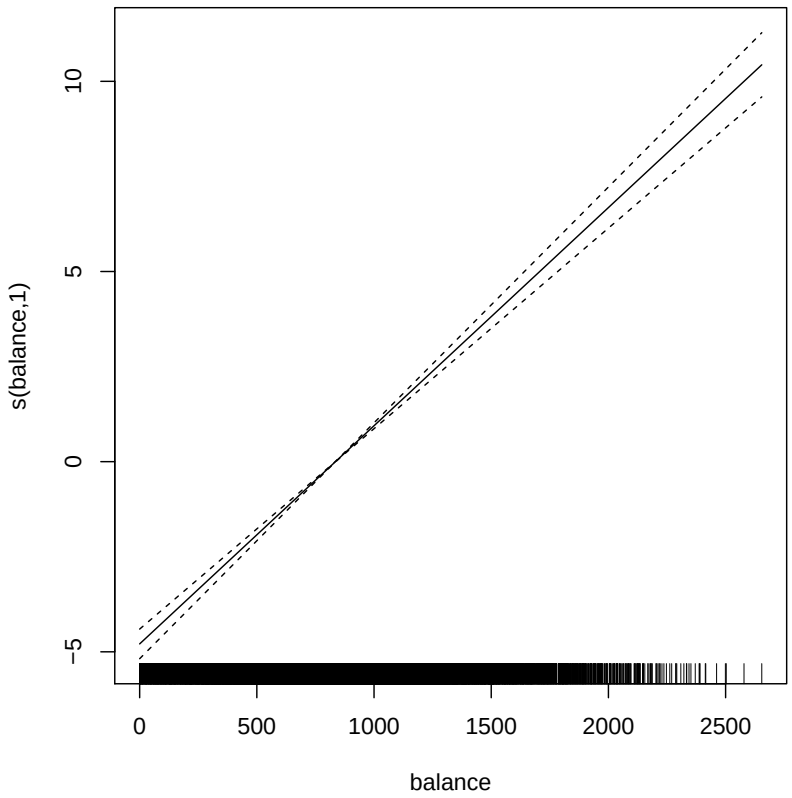
Additive models

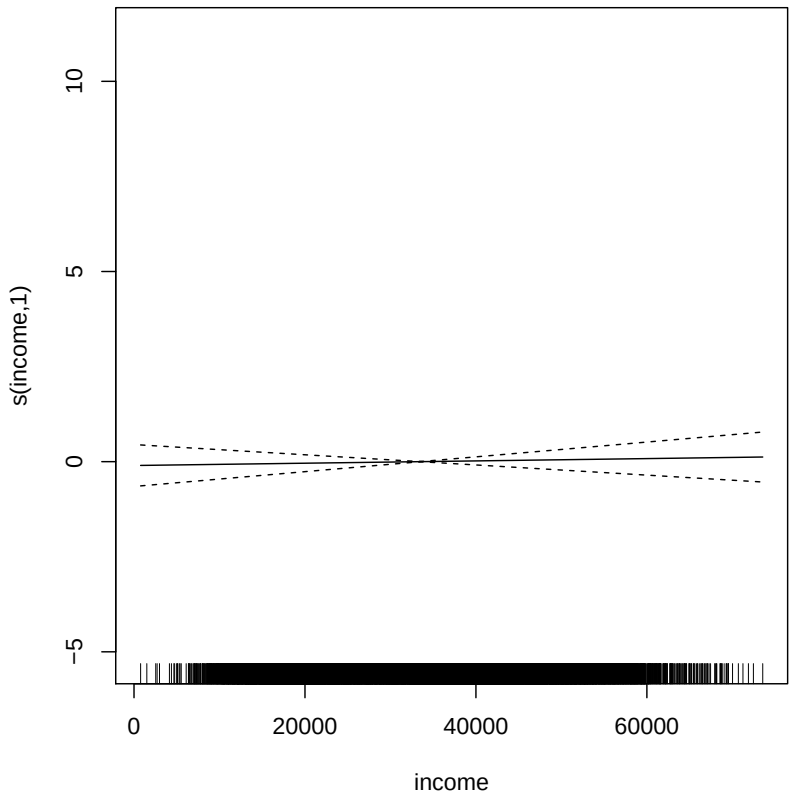
To adapt additive models to the case of logistic regression (or in fact to any other form of generalised linear model), we combine the steps in fitting an ordinary logistic (maximum likelihood implemented through iteratively reweighted least squares) with the backfitting algorithm. See ESL page 297 for details. The only change in the software is that we must signal the use of the logistic by the `family=binomial` argument (as in the `glm` function).

Example: the default data

```
library(ISLR)
library(mgcv)
data(Default)
# don't smooth student - it is a factor
default.gam = gam(default~student+s(balance)+s(income),
     data=Default, family=binomial)
par(mfrow=c(1,2))
plot(default.gam)
probs = predict(default.gam, type="response")
my.table = table(Default$default, probs>0.5)
> 1-sum(diag(my.table))/sum(my.table)
[1] 0.0268
```

Plots





No need for additive model - ordinary logistic regression will be OK

Neural nets

- There is one output node for each response category.
- At the kth output node we accumulate

$$f_k(x) = \beta_{0k} + \sum_j \beta_{kj}\sigma(\alpha_{k0} + \sum_m \alpha_{km}^T x).$$

- Outputs are

$$p_k(x) = \frac{\exp(f_k(x))}{\sum_{j=1}^K \exp(f_j(x))}$$

- Assign a case with covariate x to class having the largest value of $p_k(x)$ (equivalently the largest value of $f_k(x)$)
- Use `linout=FALSE`.

Fitting criterion

For classification the least squares loss function is not appropriate. Instead we use a loss function derived from the multinomial distribution. Suppose we have a target y having K classes. Define binary variables y_k, $k = 1, \ldots, K$ by

$$y_k = \begin{cases} 1, & y = k, \\ 0, & y \neq k. \end{cases}$$

Then we choose the weights in the NN to minimise

$$-\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik} \log p_k(x_i).$$

Note that this was the fitting criterion for the multinomial regression considered in slide 23 of lecture 8.

Why is this reasonable?

- If the the π_k are unrestricted, and $y_i \geq 0$, $\sum_i y_i = 1$ the expression

$$-\sum_{k=1}^{K} y_k \log(\pi_k)$$

takes its minimum value $\sum_{k=1}^{K} y_k \log(y_k)$ when $\pi_i = y_i$. (Assume $0 \log(0) = 0$.)

- The criterion gives the maximum likelihood estimates if the data are assumed to be multinomial.

Skip layer

- In neural networks, we can have a "skip layer" i.e linear combinations that are passed directly to the output nodes and are not transformed by the sigmoid function. In effect, we are adding another linear combination of the features to the function.
- Triggered by the argument `skip=TRUE`.
- If we set `skip=TRUE` and `size=0` we are fitting the multinomial logistic model as in slide 23 of lecture 8.

The spam data: Neural nets

Suppose data is in a data frame spam, with variables V1,...,V58. the output the variable V58, coded as 0=genuine, 1=spam. We will divide the data set into training and test sets:

```
library(nnet)
X = scale(spam.df[,-58])
y = factor(spam.df[,58])
spam = data.frame(X,y)
# pick 75% data for training set
use = sample(dim(spam)[1], 0.75*dim(spam)[1])
training = spam[use,]
test= spam[-use,]
```

The spam data: Neural nets (cont)

Fit the neural net and calculate apparent and test error:

```
fit = nnet(y~., data=training,  size = 5, maxit=1000, decay=0.1)
# traing error
mytable.training = table(training$y,predict(fit)>0.5)
> mytable.training

    FALSE TRUE
  0  2046   38
  1    39 1327
1-sum(diag(mytable.training))/sum(mytable.training)
[1] 0.02231884

# test error
mytable.test = table(test$y,predict(fit, newdata=test)>0.5)
1-sum(diag(mytable.test))/sum(mytable.test)
[1] 0.05994787
```

Classification Trees

- For classification, we use a different splitting strategy. Suppose at a node (i.e in a region) , the proportions of cases at the node falling into the different response categories are $\hat{p}_1, \ldots, \hat{p}_K$.
- We want to find splits that classify best - these are ones where one category predominates, so we want one proportion to be big and the rest small.
- Define the "node impurity" by the Gini index $1 - \sum_{k=1}^{K} \hat{p}_k^2$. This takes a minimum value of 0 when all the cases are in a single category and a maximum of $1 - 1/K$ when the numbers of cases in each category are the same.

Trees (cont)

- Thus, splits where one category dominates will have small values of the Gini index i.e. low node impurity. These are "good splits".
- We then split the cases at the node into two subgroups, in such a way to maximise the difference between the "parent" node impurity and the weighted sum of the impurities of the two " child" nodes (weighted by the numbers at each child node). Thus, we are choosing the split that gives the biggest decrease in the node impurities. (Compare with regression where we split to get the biggest reduction in the residual sum of squares.)

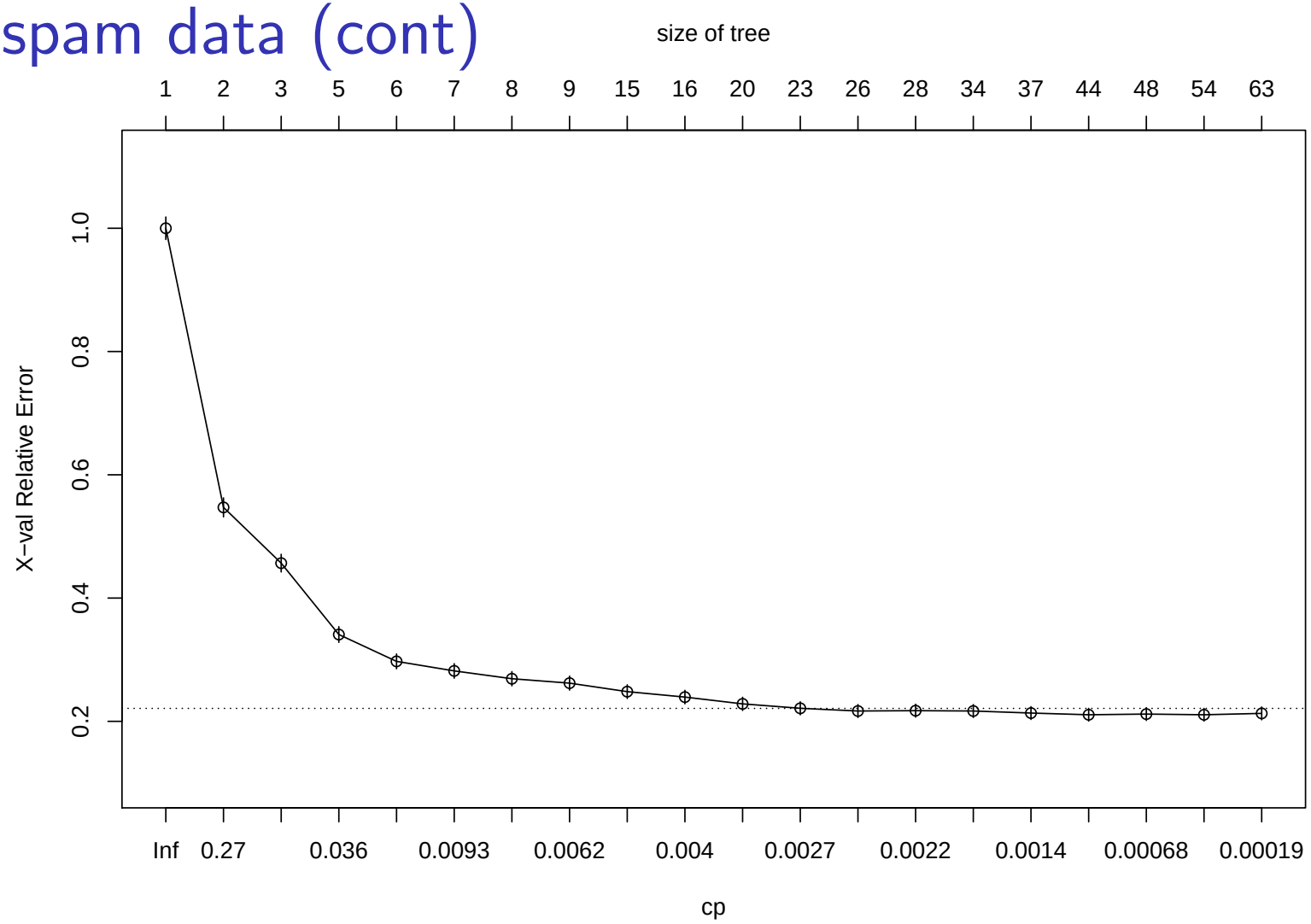
Trees (cont)

- At each terminal node, we record the class with the highest proportion of cases (the majority group)
- To predict a new case, we pass it through the tree. It winds up at node m say. We assign the new case to the "majority group" of node m.
- Note that each node of the tree corresponds to a set of probabilities (the proportions of cases falling to the various classes.)

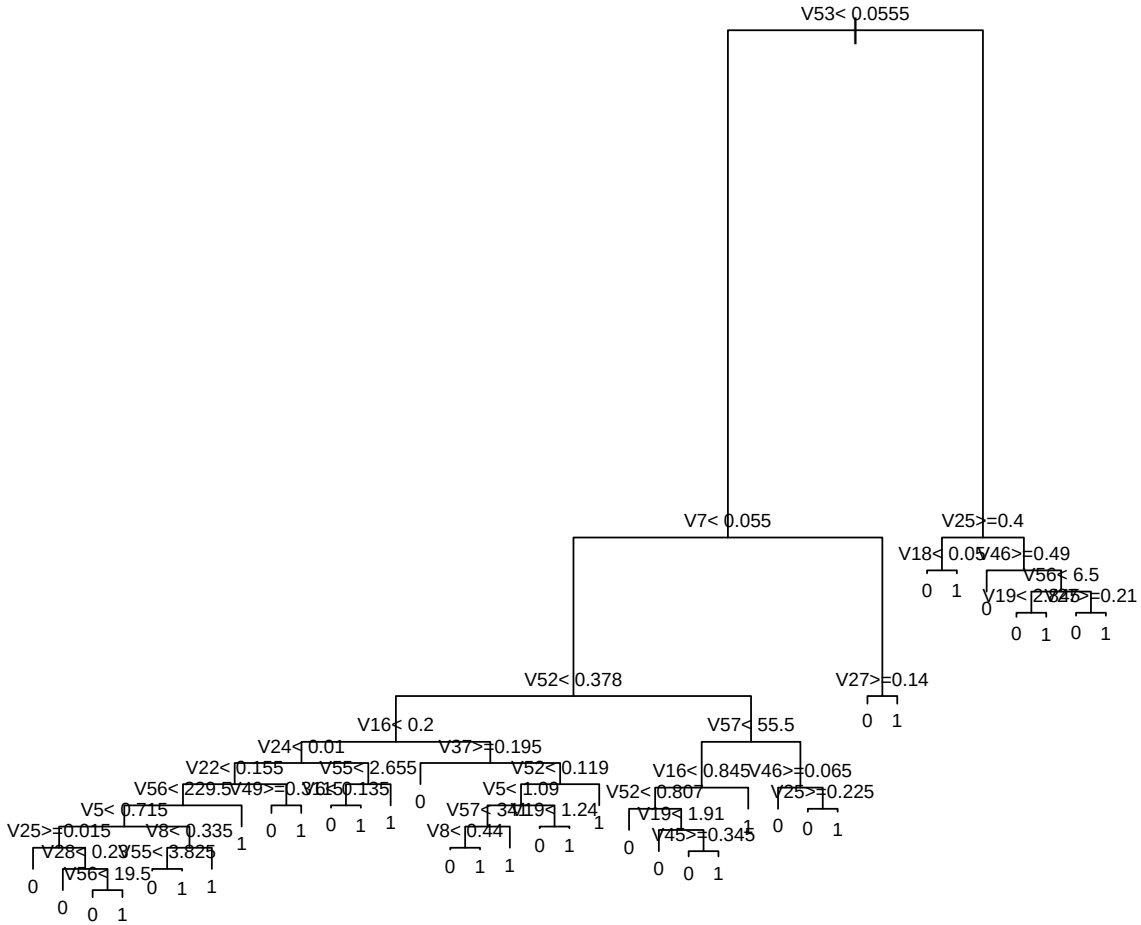
The spam data

```
# spam data: trees
library(rpart)
fit1 = rpart(spam~., data=spam.df, method = "class",
  parms = list(split="gini"), cp=0.0001)
plotcp(fit1)
fit2 = prune(fit1, cp=0.0015)
plotcp(fit2)
plot(fit2)
text(fit2, cex=0.7)
```

The spam data (cont)



The spam data (cont)



Tree: training error

```
# note that the predict function produces a matrix
# in this case
err=table(predict(fit2)[,2]>0.5, spam.df$spam)
> err

          0    1
  FALSE 2670  134
  TRUE   118 1679
> 1-sum(diag(err))/sum(err)
[1] 0.0547707
```

About the same as for NN.

Tree: CV error

The `printcp` function gives a relative (relative to the null model) cross-validated error of 0.20684. To get the actual error we have to multiply this by the error of the null model, which would always predict a message as genuine. (since genuine messages are more likely than spam, at least in the data we have.) Thus, the proportion misclassified will just be the proportion of spam messages, or $1813/4601 = 0.3940448$. The actual CV error is 0.20684* $0.3940448 = 0.08150423$. Thus, we would expect about 8% of the messages to be wrongly classified.

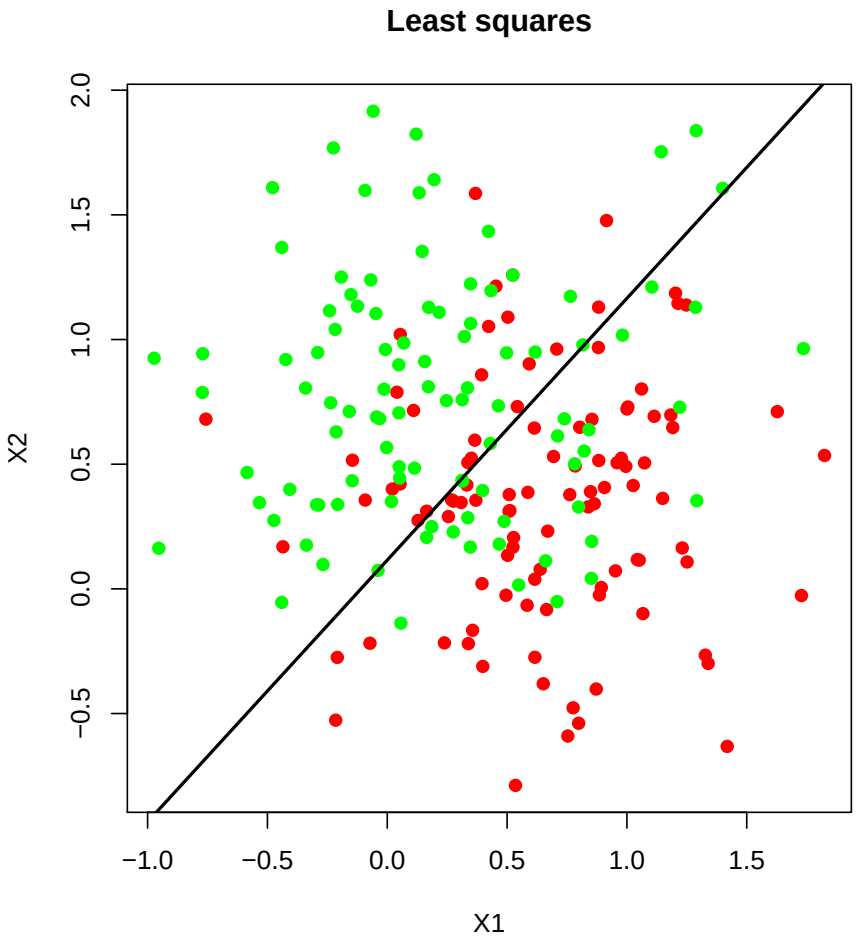
Support vector machines

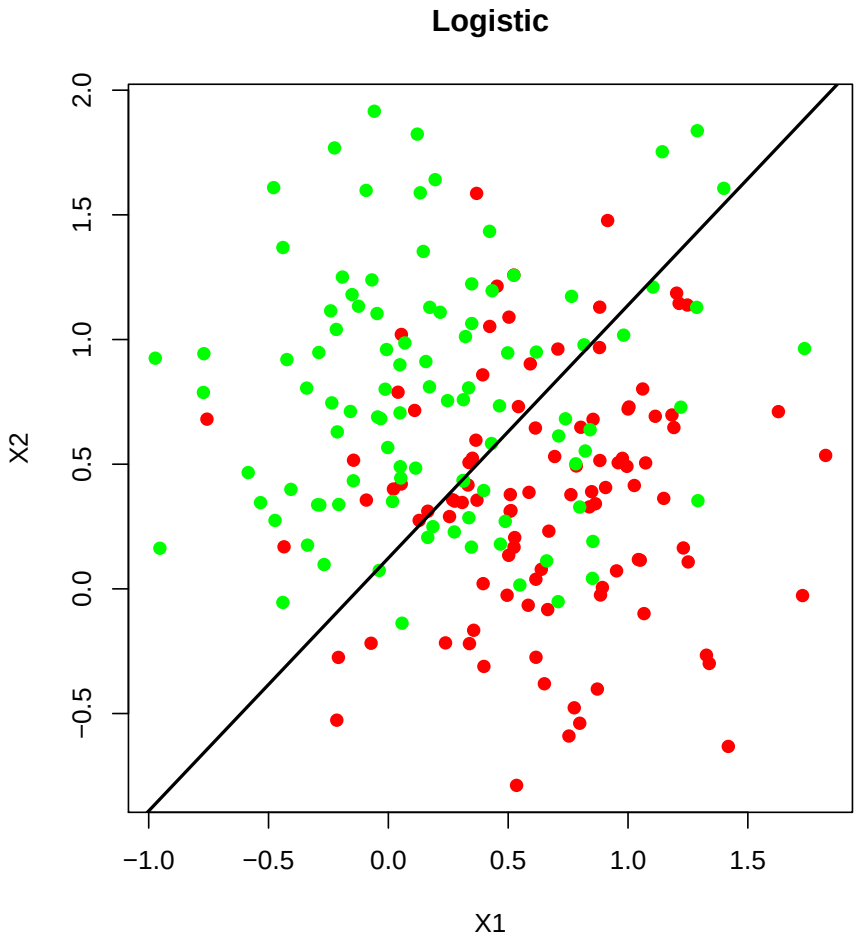
Suppose we have a a binary response Y having values ± 1 and we want to predict a future value of Y using a classifier based on features (explanatory variables) $x_1, \ldots, x_k$.
Possible approaches:

- Use linear regression. Predict $Y = 1$ if $\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k > 0$ and -1 otherwise.
- Use logistic regression, treating Y as a factor with baseline -1. Predict $Y = 1$ if $\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k > 0$ and -1 otherwise.
- Other methods ...

Often the first two are pretty similar:

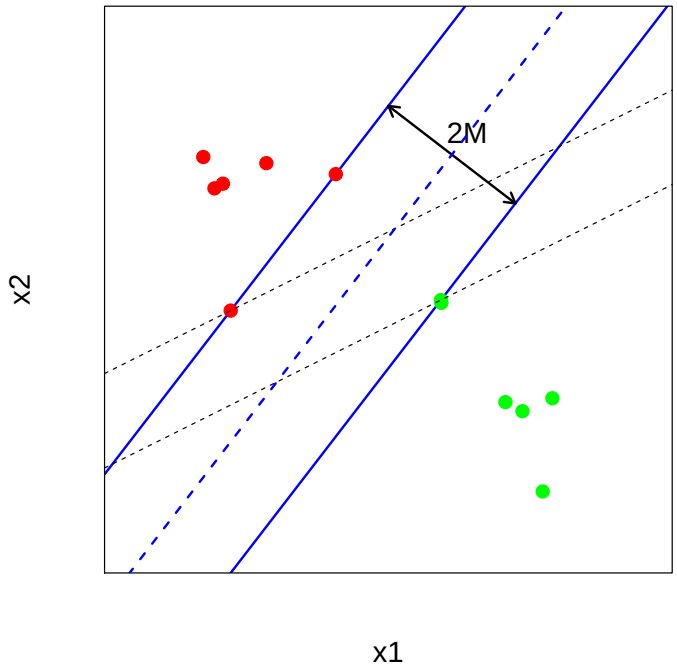
Least Squares vs Logistic





An alternative: SVM

If the points can be separated:



We choose the two parallel boundaries to maximize M (maximum separation) These in turn determine a single plane half-way in between.

Equivalent formulation

If the two sets of points can be separated, the best separating plane is the solution of an optimization problem. The equation of a plane can be written

$$b^T x + b_0 = 0,$$

for a k-vector b and a constant b_0. The equation of the best separating plane is the solution of the following constrained minimization problem:
Minimize

$$||b||^2$$

subject to

$$y_i(b^T x_i + b_0) \geq 1, i = 1, 2, \ldots n.$$

Here y_i and x_i refer to the response and the vector of covariates for the ith individual in the training set.

Another formulation

In optimisation, some problems have a dual formulation for which the solutions may be easier to find. We can then get the solution to the original problem (the primal) from the solution of the dual problem. For the SVM minimisation, the dual problem is
Minimize

$$\sum_{i=1}^{n} a_i - 0.5 \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j y_i y_j x_i^T x_j$$

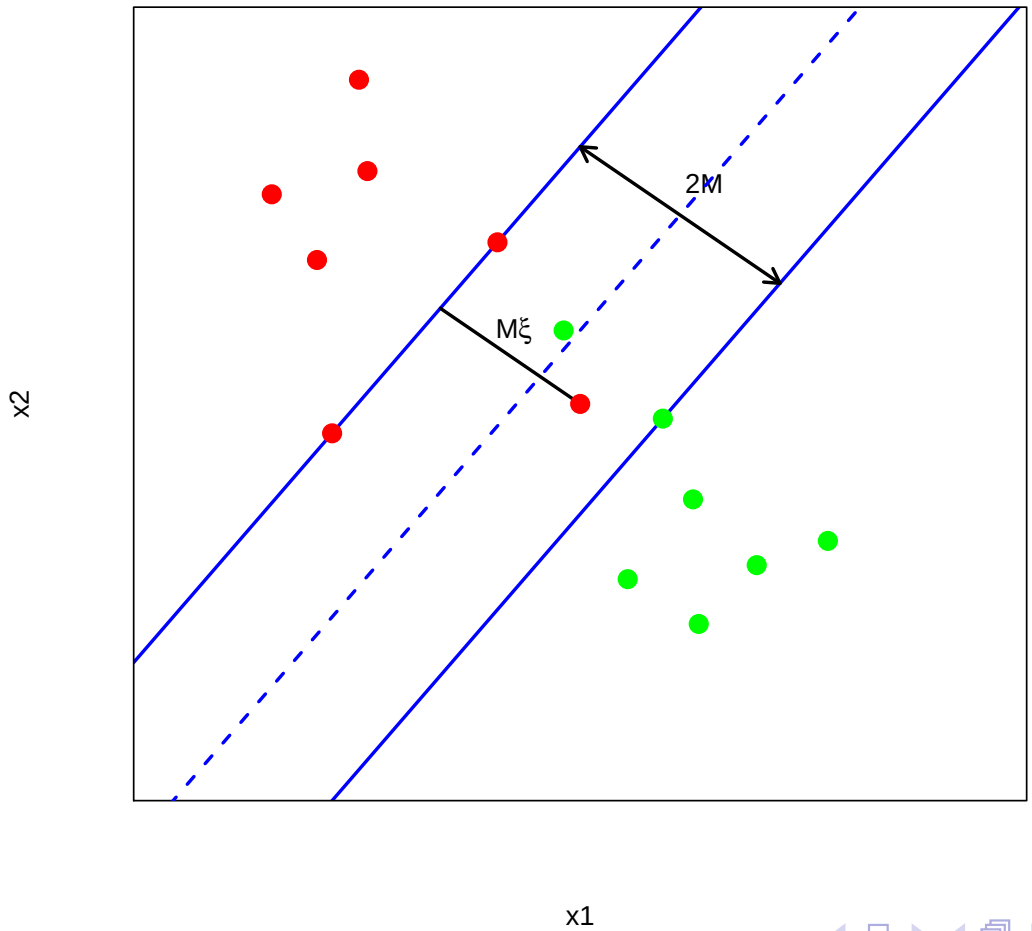
subject to $a_i \geq 0$ and $\sum_i a_i y_i = 0$.
Then the vector b solving the original problem is

$$b = \sum_{i=1}^{n} a_i y_i x_i$$

which is a linear combination of (some of) the vectors x_i. The x_i's corresponding to the non-zero a_i's are called *support vectors*.

Overlapping data

If the points cannot be separated:



Overlapping data (2)

Now we choose the two parallel boundaries to maximize the separation and minimize the sum of the ξ's:
Minimize

$$||b||^2 + C \sum_i \xi_i$$

subject to

$$y_i(b^T x_i + b_0) \geq 1 - \xi_i, i = 1, 2, \ldots n.$$

The parameter C controls the trade-off.

Dual formulation

In the overlapping case, the dual problem is
Minimize

$$\sum_{i=1}^{n} a_i - 0.5 \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j y_i y_j x_i^T x_j$$

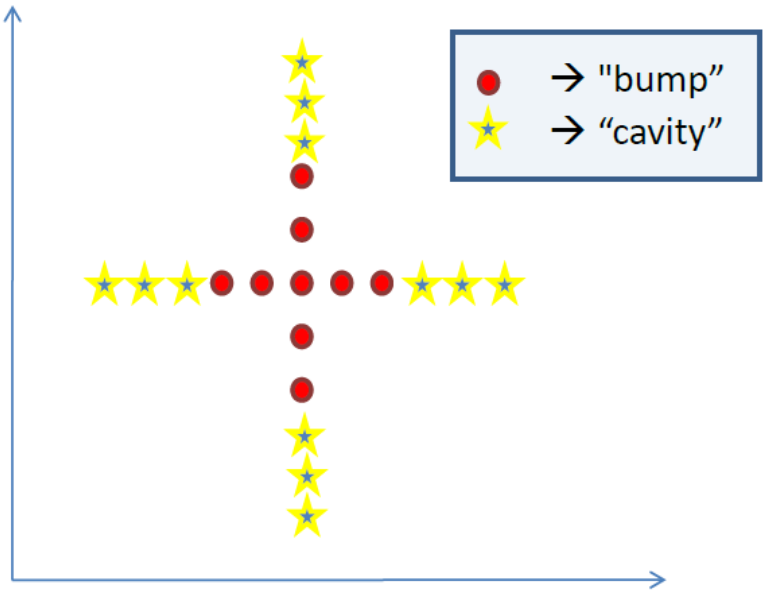
subject to $0 \leq a_i \leq C$ and $\sum_i a_i y_i = 0$.
Then as before, the vector b solving the original problem is

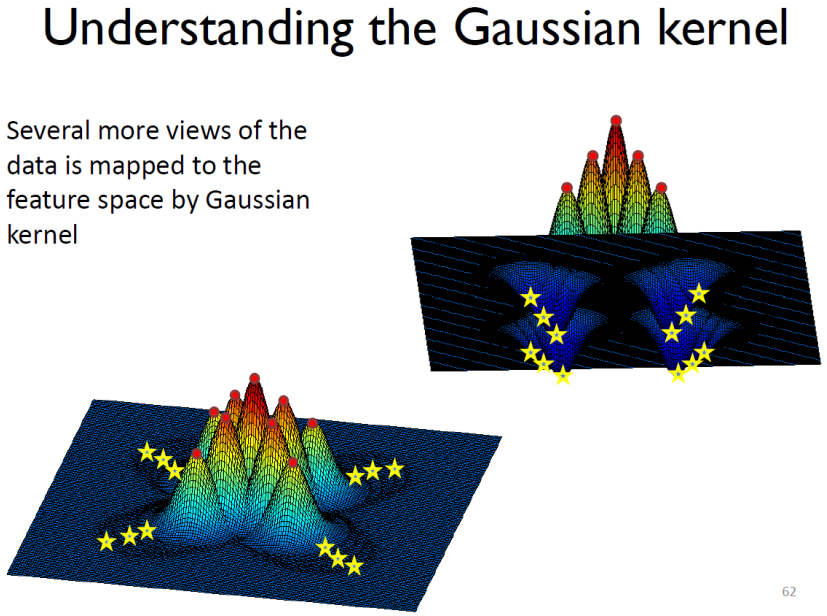
$$b = \sum_{i=1}^{n} a_i y_i x_i$$

which is a linear combination of (some of) the vectors x_i. The x_i's corresponding to the non-zero a_i's are also called *support vectors*.

Enlarging feature space

Suppose we map our data points to a higher dimension, using a mapping Φ. Even if the points overlap the linear boundary in the original space, they may not in the enlarged space.





Enlarging feature space (2)

All we need is to replace the inner products between x_i and x_j with inner products between $\Phi(x_i)$ and $\Phi(x_j)$. In fact, we assume that these are given by a "kernel" $K(x_i, x_j)$.

Examples:

Gaussian kernel: $K(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$

Laplace kernel: $K(x_i, x_j) = exp(-\frac{||x_i - x_j||}{2\sigma})$

Example: the spam data

```
# SVM example- spam data

library(kernlab)
spam.df$spam = factor(spam.df$spam)
svm.stuff1 = ksvm(spam~., data = spam.df, C = 1, cross=5)
svm.stuff2 = ksvm(spam~., data = spam.df, C = 2, cross=5)
svm.stuff5 = ksvm(spam~., data = spam.df, C = 5, cross=5)
```

Example: the spam data (cont)

```
> svm.stuff2
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 2

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0292851548010902

Number of Support Vectors : 1332

Objective Function Value : -1304.522
Training error : 0.040643
Cross validation error : 0.067812
```

Gradient boosting

For a general loss function L, we want to choose a function f from some class of functions (for example trees) to minimize

$$L(f) = \sum_{i=1}^{n} L(y_i, f(x_i)).$$

Now for the moment, just think of the numbers $f_i = f(x_i)$ as free, unrestricted variables, so we are minimizing $\sum_{i=1}^{n} L(y_i, f_i)$. A standard way of doing this is by the method of steepest descent: we start with an initial guess for the minimizing f's, and then move a short step in the direction where $L(f)$ decreases most rapidly. We keep moving in this way until the minimum is reached. The steps take the form

$$f_{new} = f_{old} - \eta \frac{\partial L}{\partial f}.$$

Gradient boosting (cont)

Of course, the f_is are not arbitrary, but are the predictions from a model. But let's choose the tree that best approximates $-\frac{\partial L}{\partial f}$ and add that on instead. This is gradient boosting.

The algorithm is described more precisely in ESL page 361. As with boosting in prediction, we add on a small multiple of the tree at each iteration. In fact, for squared error loss, the boosting procedure described earlier is equivalent to the one discussed here.

Boosting is often applied to trees as the base learner, using small trees (say between 4 and 8 terminal nodes). The R function `mboost` is used to boost classification trees.

Boosting code

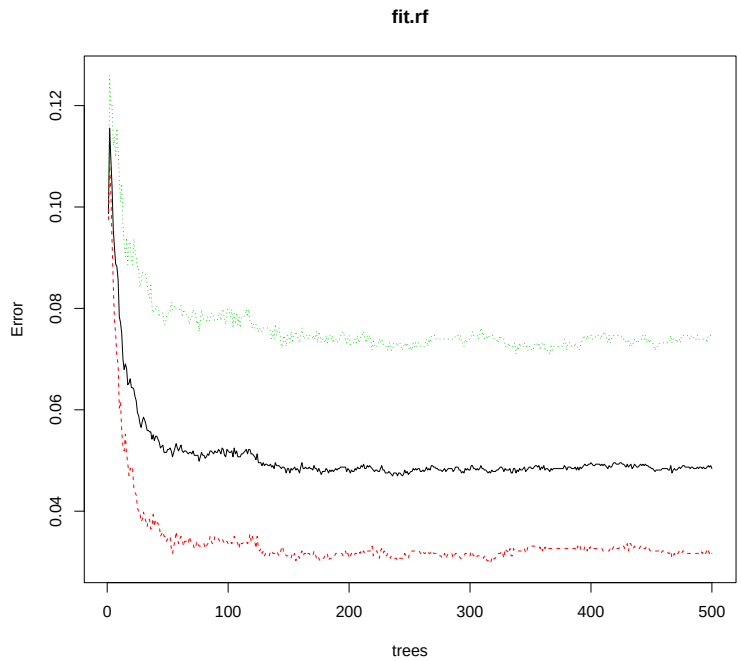
```
# boosted tree
fm =y~btree(training[,-58],
   tree_controls = party::ctree_control(maxdepth = 4))
fit.boost = mboost(fm, data = training, family=Binomial(),
   control = boost_control(mstop =200, nu = 0.1))
myrisk = cvrisk(fit.boost)

# training error
training.tab = table(training$y, predict(fit.boost)>0.5)
1-sum(diag(training.tab))/sum(training.tab)
[1] 0.06434783

# test error
test.tab = table(test$y, predict(fit.boost, newdata=test)>0.5)
1-sum(diag(test.tab))/sum(test.tab)
[1] 0.0625543
```

Random forests for classification.

- These work equally well for classification. The individual trees are grown as described on slides 19-21.
- When combining the results, instead of averaging, we use a "majority vote" rule: we classify the case into the group chosen by the majority of the trees.
- In contrast to boosting, the trees are grown quite large. (the default setting ion the software is `nodesize=1`)

Random Forests

```
fit.rf=randomForest(y~.,data=training, mtry =10, ntree=500)
plot(fit.rf)
> tail(fit.rf$err.rate, n=2)
              OOB          0          1
[499,] 0.04898551 0.03214971 0.07467057
[500,] 0.04840580 0.03166987 0.07393851
```



Comparisons: spam data.

Classification errors

	Training	Test
Linear	0.110	0.115
Logistic	0.080	0.115
Tree	0.097	0.106
Boosted tree	0.058	0.072
Neural8	0.011	0.068
SVM	0.040	0.067
Neural4	0.031	0.065
Random Forest	0.049	0.048

References

1. Efron, B. and Tibshirani, R.J. (1993). *An Introduction to the Bootstrap*. Chapman and Hall.
2. Hastie, T. and Tibshirani, R.(1986). Generalized additive models. *Statistical Science*,1,297-318.
3. Hastie, T., Tibshirani, R.J. and Friedman, J. (2009). *The Elements of Statistical Learning, 2nd Ed*. Springer.
4. Venables, W.N. and Ripley, B.D. (2002) *Modern Applied Statistics with S, 4th Ed.* Springer
5. Boosting: HTF Chapter 10.
6. Bagging: HTF Chapter 8 (section 8.7).
7. Random Forests HTF Chapter 15, see also `http://www.stat.berkeley.edu/ breiman/RandomForests/`
8. Torsten Hothorn, Peter Buehlmann, Thomas Kneib, Mattthias Schmid and Benjamin Hofner (2010), Model-based Boosting 2.0. *Journal of Machine Learning Research*, 11, 2109-2113. On the course web page.
9. Peter Buehlmann and Torsten Hothorn (2007), Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, 22(4), 477-505.