# Lecture 4: Classification

Alan Lee

Department of Statistics
STATS 760 Lecture 4

May 5, 2017

# Outline

# Today's agenda

In this lecture we how our machine learning techniques can be adapted for classification problems. Some of the ideas should be familiar from other courses, particularly STATS 330/762. Today we will cover

- Classification: general principles;
- Bayes classifier;
- Logistic regression;
- Cross-validation and bootstrap estimates of classification error;
- $K$-nearest neighbour classifier;
- Neural nets for classification;
- Classification trees, boosted trees, random forests;
- Support vector machines.

# Classification

- So far, we have looked at prediction problems where the response variable was continuous, and the predictions were based on fitting models by least squares.
- Now we consider the case where the response variable is categorical, indicating which one of a number of classes the observation belongs to e.g. alive/dead, win/draw/loss, 1-5 in a Likert scale in a questionnaire. Thus, the response variable is a factor (as in logistic regression).
- Given we have observed inputs $x_1, \ldots, x_k$, how should we classify an observation? For example, given a person's age, can we predict if they suffer from arthritis?
- We often refer to prediction of categorical variables as *classification*, and the prediction formulas as *classification rules*.

# Examples

Our first example is from *An Introduction to Statistical Learning*. In the package `ISLR` there is a data set `Defaults` containing data on 10,000 monthly credit card bills. The variables are

- income : annual income of card holder;
- balance : the monthly balance;
- student :student status Yes=student, No=Not a student;
- default : defaulted on payment? Yes/No

The response is default: we want to predict if a customer will default on their credit card bills, knowing the valies of the other variables.

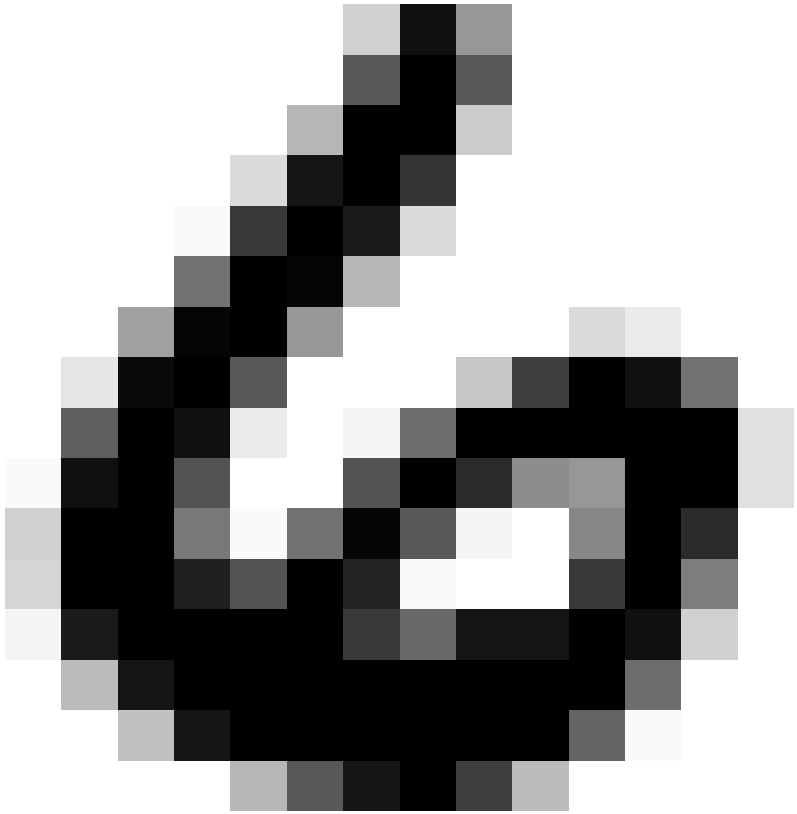
# Example: Zip codes

- 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 -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.

# Bayes classifier

Suppose for each age $x$, we know the conditional probabilities

$$\text{Prob}[\text{ has arthritis } | \text{age } = x],$$

$$\text{Prob}[\text{ does not have arthritis } | \text{age } = x].$$

Then the best classifier (one that minimises the probability of misclassification) is the one that assigns to a individual aged $x$ the class with the highest conditional probability. ( In the case of just two classes, as is the case here, this is the same as classifying an individual as having arthritis if

$$\text{Prob}[\text{ has arthritis} | \text{age } = x] > 0.5.$$

# Estimating conditional probabilities

The problem with this approach is that we don't usually know these conditional probabilities. However, if we could **estimate** them, we could use the Bayes classifier with the estimated probabilities.

We will look at several ways of doing this: including logistic regression, and $K$-nearest neighbours.
A common model for the conditional probabilities in the binary case, with response 0/1 is

$$\text{Prob}[Y = 1|x] = \frac{\exp(f(x))}{1 + \exp(f(x))}$$

where $f$ is some function eg linear (logistic regression), gam, neural net etc.

# Logistic regression

We first discuss the case where there are just two categories, which we label 0 and 1. We denote the output as $Y$, so either $Y = 0$ or $Y = 1$, and the inputs as $x$. In logistic regression, we model the probabilities as

$$p(x) = \text{Prob}[Y = 1|x] = \frac{\exp\{b_0 + b_1 x_1 + \cdots b_k x_k\}}{1 + \exp\{b_0 + b_1 x_1 + \cdots b_1 x_k\}},$$

$$\text{Prob}[Y = 0|x] = \frac{1}{1 + \exp\{b_0 + b_1 x_1 + \cdots b_1 x_1\}},$$

where we have to choose the values of $b_0, b_1, b_k$, based on a training set.

# Choosing the $b$'s

The $b$'s are chosen by the method of maximum likelihood - this is a statistical technique based on assumptions on how the data was generated. See STATS 330 for more detail. In R, this is done using the function `glm`. See the example below.
We minimise the criterion

$$-\sum_{i=1}^{n}\{y_i \log p(x_i) + (1-y_i)\log(1-p(x_i))\}).$$

# Prediction error

In classification problems, the prediction error is measured by the probability of misclassification. This can be estimated from the training set by the proportion of misclassified observations, or from the test set in the same way. We illustrate with the credit card data:

```
> library(ISLR)
> data(Default)
> default.logistic = glm(default~ student+balance+income,
+       data=Default, family=binomial)
> coefficients(default.logistic)
  (Intercept)    studentYes        balance         income
-1.086905e+01 -6.467758e-01  5.736505e-03  3.033450e-06
```

Note: The effect of being a student is to decrease the predicted log-odds by 6.467758e-01.

# Predicting

To make the predictions, we use the generic `predict` function to calculate the estimated conditional probabilities, and then make a table of predictions versus actuals. We predict a default if the estimated probability is greater than 0.5:

```
> probs = predict(default.logistic, type="response")
> my.table = table(Default$default, probs>0.5)
> my.table
      FALSE TRUE
  No   9627   40
  Yes   228  105
```

The training error is the proportion of individuals on the off-diagonals:

```
> 1-sum(diag(my.table))/sum(my.table)
[1] 0.0268
```

# Bootstrapping and cross-validation

This works exactly as before, except that instead of averaging the squared errors, we average the quantity

$$L(y, \hat{y}) = \begin{cases} 1, & y \neq \hat{y}, \\ 0, & y = \hat{y}, \end{cases}.$$

where $y$ is the actual category and $\hat{y}$ the predicted one. (For numerical outputs we use $L(y, \hat{y}) = (y - \hat{y})^2$)

# Example: the default data

```
> library(R330)
> cross.val(default.logistic,nfold=10)
Mean Specificity =  0.9958682
Mean Sensitivity =  0.3174139
Mean Correctly classified =  0.973215
> 1-0.973215
[1] 0.026785

err.boot(default.logistic,B=100)
$err
[1] 0.0268

$Err
[1] 0.026654
```

# Example: More than two output categories

Suppose that there are $J$ output categories, $C_1, C_2, \ldots, C_J$. We estimate the conditional probabilities by

$$\text{Prob}[Y \text{ in } C_j|x] = \frac{\exp\{b_{j0} + b_{j1}x_1 + \cdots b_{jk}x_k\}}{1 + \sum_{j=1}^{J-1} \exp\{b_{j0} + b_{j1}x_1 + \cdots b_{jk}x_k\}}$$

for $j = 1, 2, \ldots, (J-1)$ and

$$\text{Prob}[Y \text{ in } C_J|x] = \frac{1}{1 + \sum_{j=1}^{J-1} \exp\{b_{j0} + b_{j1}x_1 + \cdots b_{jk}x_k\}}$$

The coefficients $b_{jk}$ are found using the `nnet` function - see later.

# K-nearest neighbours

Here we estimate

$$\text{Prob}[Y \text{ in } C_j|x]$$

by the proportion of the $K$ data points closest to $x$ that are in category $C_j$.

To define "closest" we need a definition of distance - in the case of numeric variables we use Euclidean distance, in the case of categorical inputs we can use dummy variables - see STATS 330 for details.

We should standardise the variables to prevent one variable dominating the distance.

# Example

```
> input.df = Default[,-1]
> # change the varible "student" to a dummy variable
> input.df[,1] = as.numeric(input.df[,1]) - 1
> # scale the inputs
> input.df=scale(input.df)
> predictions = knn(input.df, input.df, Default$default, k=5)
> knn.table = table(Default$default,predictions)
> knn.table
     predictions
        No  Yes
  No  9614   53
  Yes  199  134
> 1-sum(diag(knn.table))/sum(knn.table)
[1] 0.0252
```

# Neural nets

- There is one output node for each response category.
- At the $k$th 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`.

# 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.)

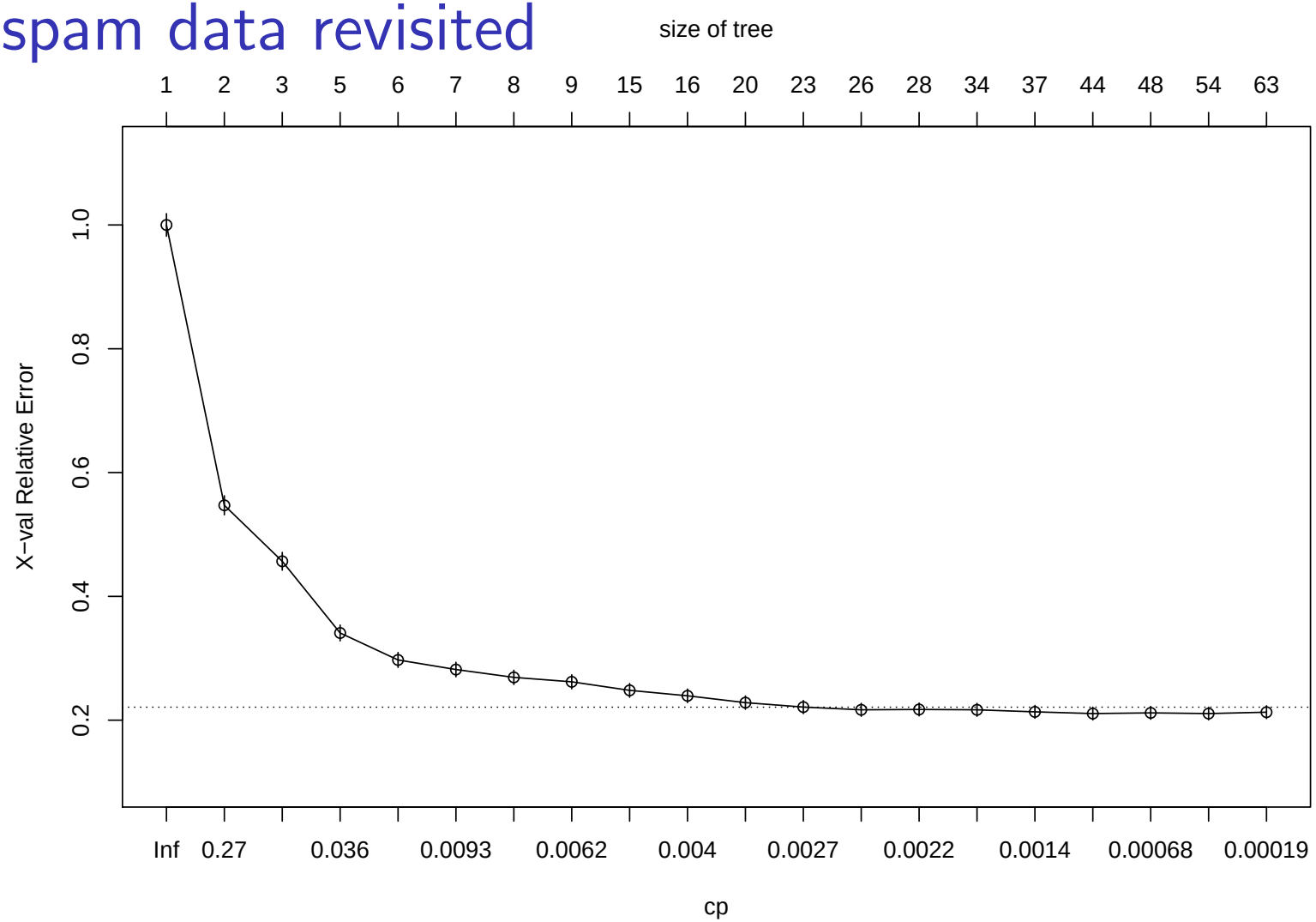
# Trees (cont)

- 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.)

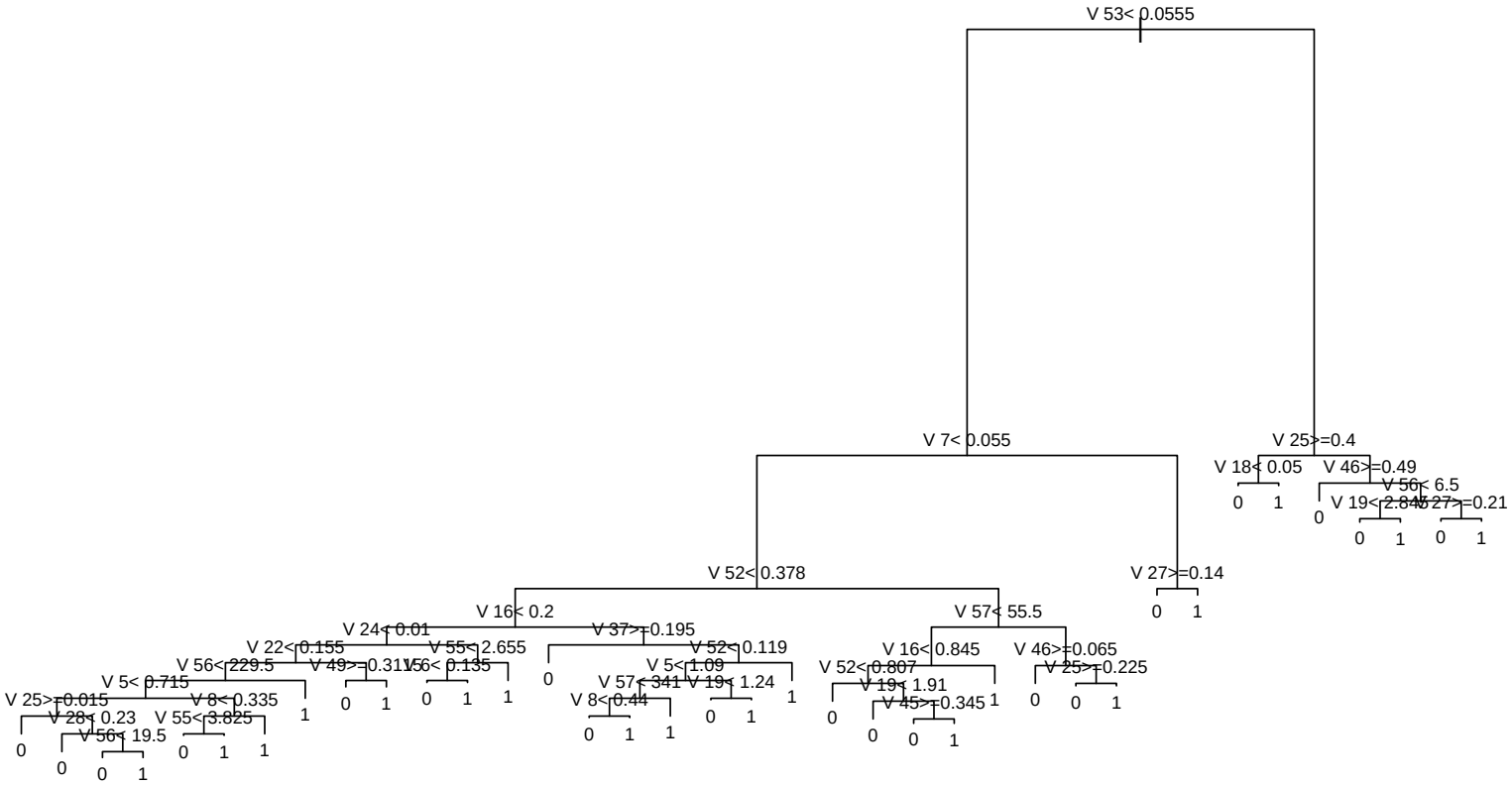
# The spam data revisited

```
# 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 revisited



# The spam data revisited



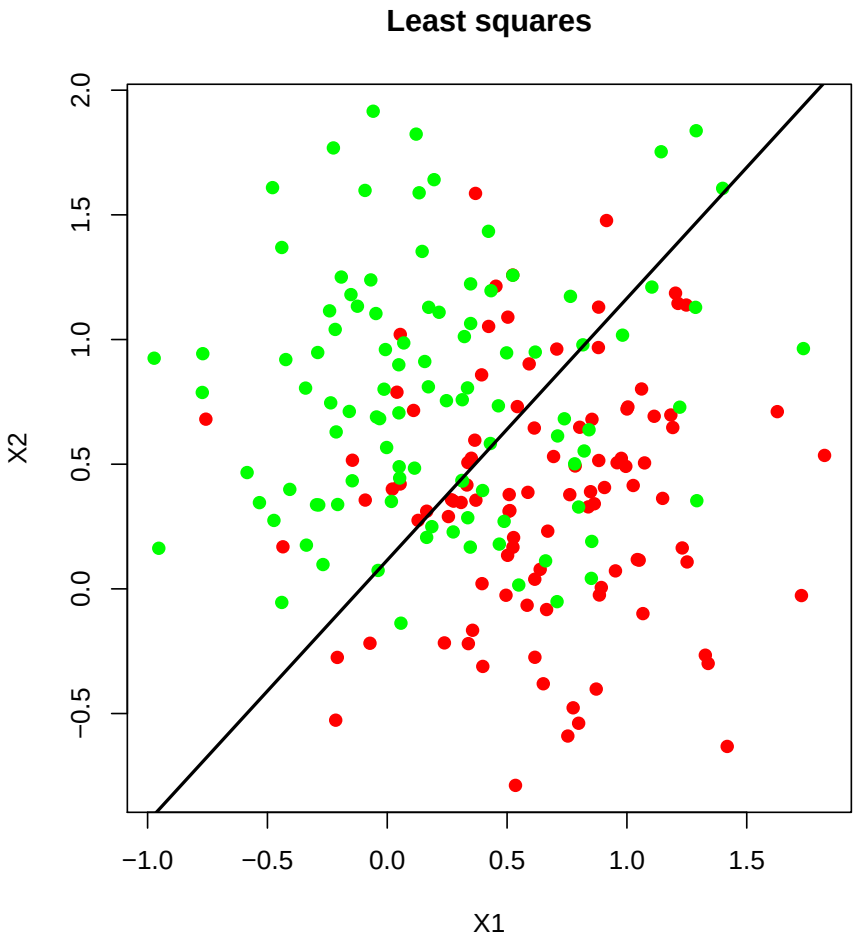
# Support vector machines

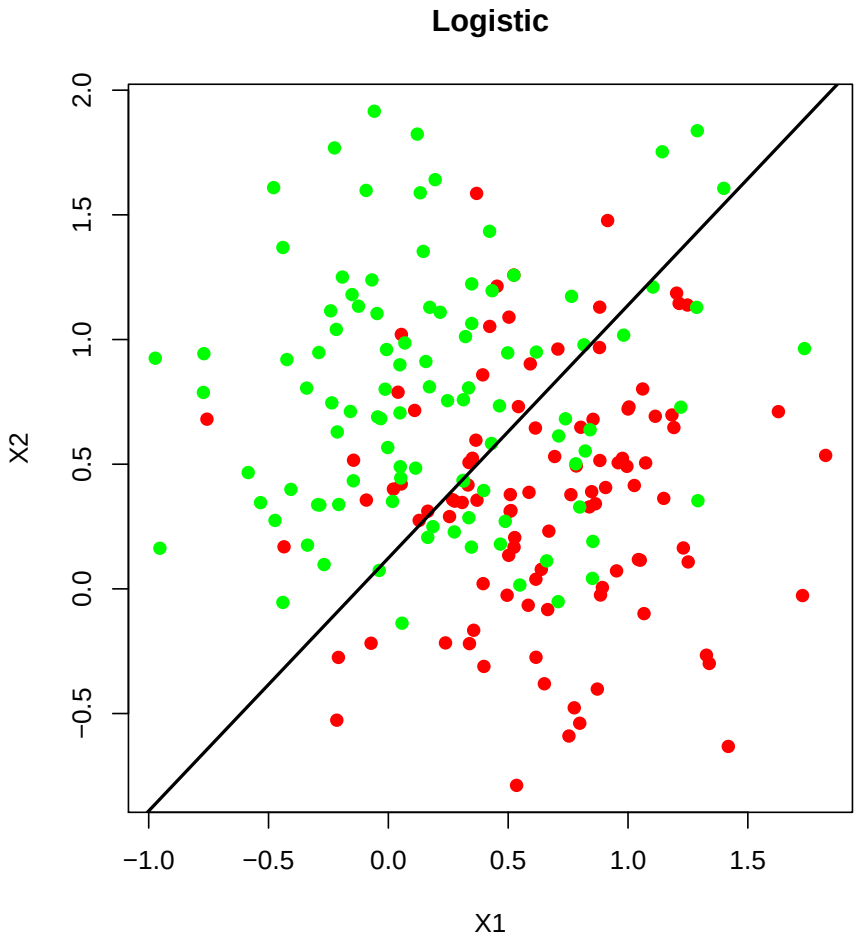
Suppose we have a a binary response $Y$ having values $\pm 1$ and we want to predict a future value of $Y$ using a classifier based on features (explanatory variables) $x_1, \ldots, x_k$.
Possible approaches:

- 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:

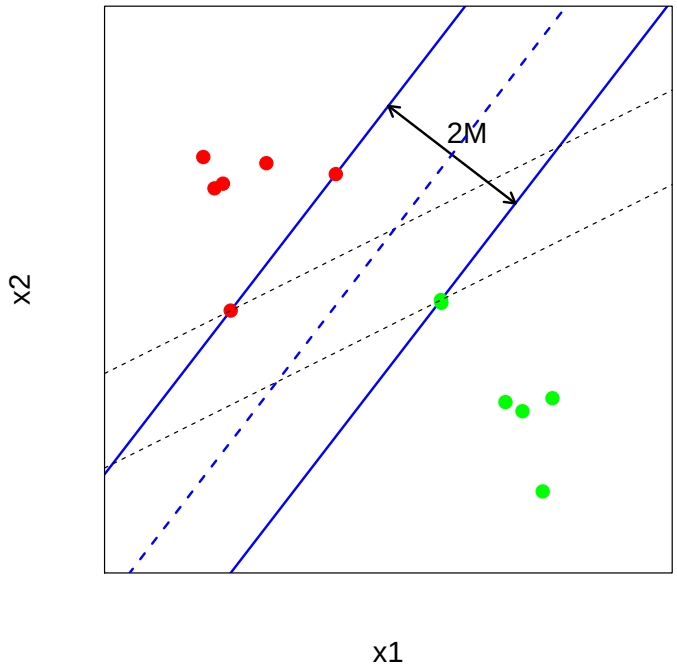
# Least Squares vs Logistic



# 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 $i$th 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

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

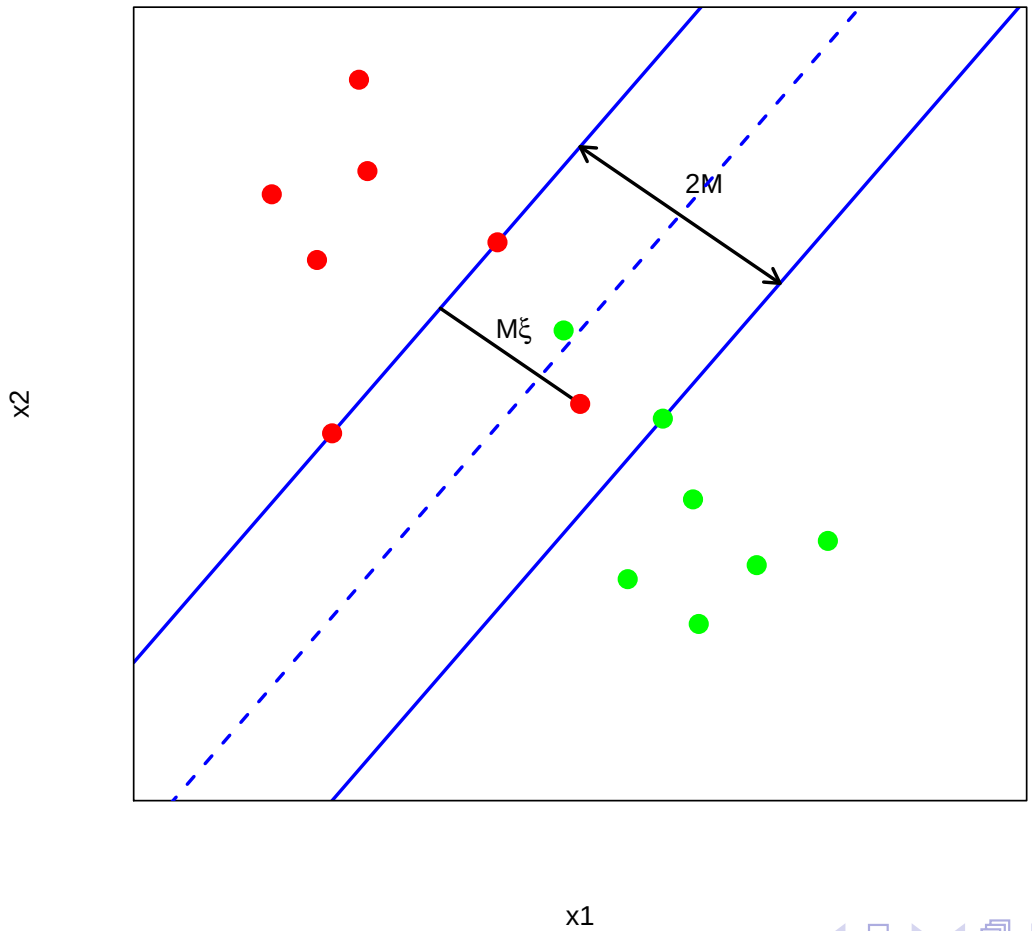
subject to $a_i \geq 0$ and $\sum_i a_i y_i = 0$.
Then the vector $b$ solving the original problem is

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

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 $\xi$'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

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

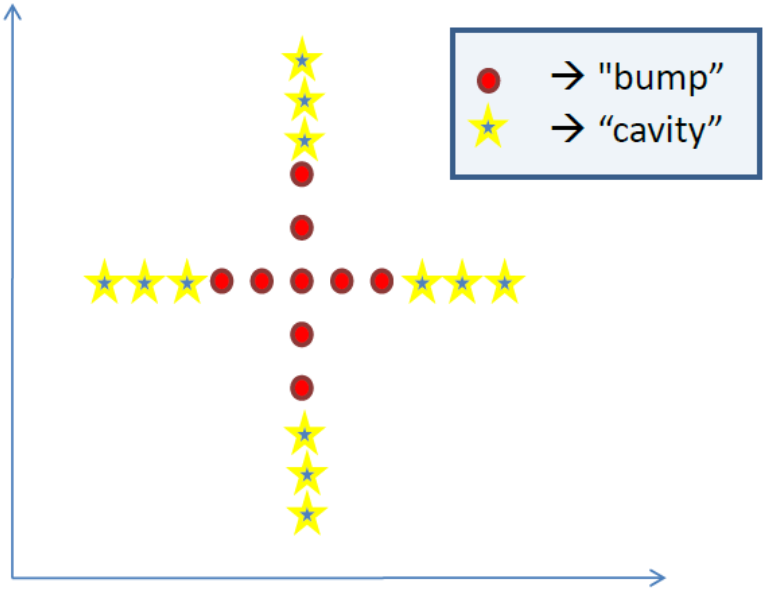
subject to $0 \leq a_i \leq C$ and $\sum_i a_i y_i = 0$.
Then as before, the vector $b$ solving the original problem is

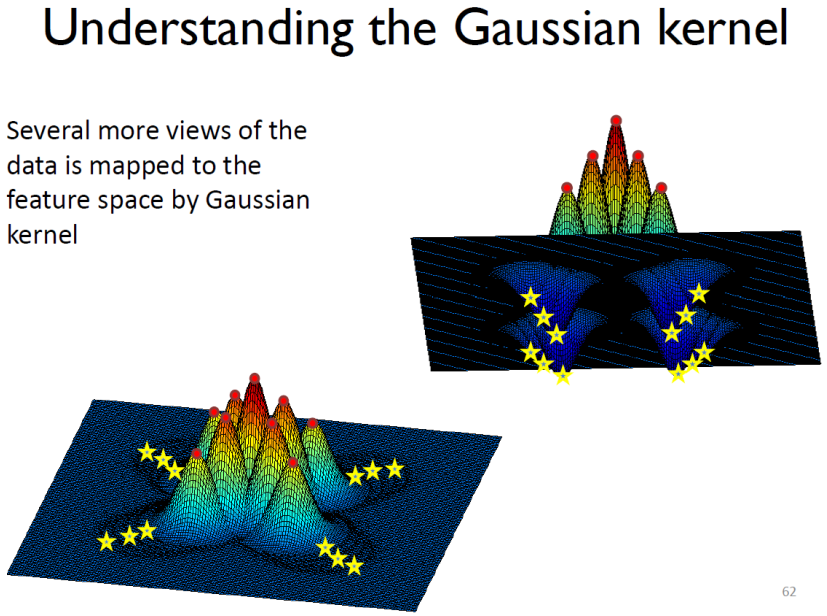
$$b = \sum_{i=1}^{n} a_i y_i x_i$$

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 $\Phi$. 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

# get data

infile = "C:\\Users\\alee044\\Documents\\
            Teaching\\760\\2013\\spam.txt"
spam = read.table(infile, header=FALSE)

library(kernlab)
spam$V58 = factor(spam$V58)
svm.stuff1.0 = ksvm(V58~., data = spam, C = 1, cross=5)
svm.stuff0.5 = ksvm(V58~., data = spam, C = 2, cross=5)
svm.stuff2.0 = ksvm(V58~., data = spam, C = 5, cross=5)
```

# Example: the spam data (2)

```
> svm.stuff2.0
Support Vector Machine object of class "ksvm"

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

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.029842359586131 4

Number of Support Vectors : 1339

Objective Function Value : -1300.194
Training error : 0.040426
Cross validation error : 0.066724
```

# Gradient boosting

A technique that can be applied to both regression and classification. See HTF sections 10.10 and 10.11. Seems to work well for classification.

# Random forests for classification.

- These work equally well for classification. The individual trees are grown as described on slides 13-15.
- 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.

# 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 |

# Code

Tree

```
myfit1 = rpart(y∼., data=training, method = "class",
parms = list(split="gini"), cp=0.01)
```

Boosted tree

```
y.t = factor(y)
training = data.frame(X, y.t)
fm =y.t∼btree(data.frame(X), tree_controls =
ctree_control(maxdepth = 5))
myfit2 = mboost(fm, data = training.t,
family=Binomial(), control = boost_control(mstop
=160, nu = 0.1))
```

Random forests

```
myfit3=randomForest(y.t∼., data=training)
```