Introduction to Classification

Dr. Gordon Ross

Supervised Learning

Supervised learning: given a set of (y_i, x_i) pairs where the y_i's are known, how can we build a model/algorithm for predicting future values of y given the corresponding values of x?
It is typical to distinguish between two settings:

- When the y_i variables can only take on a discrete set of categorical values, this is known as **classification**. For example, in binary classification, there are two classes and $y_i \in \{0, 1\}$.
- When the y_i variables can take on continuous values, this is known as **regression**.

Authorship attribution is a classification problem, so this course will focus only on the categorical setting.

Classification

General setup: an 'object' can belong to one of a number of categories, but we do not know to which category a particular object belongs. We collect a set of measurements (also known as features) from this object to represent the object as an x_i vector and then have to decide its most likely category.

If there are only two possible categories then this is called binary classification. If there are more than two, then it is called multi-class classification.

Classification

Classification has many other applications outside stylometry. Examples (categories indicated in red)

1. An emergency room in a hospital needs to determine whether a patient should be put into an intensive care unit or not put in. The measurements will be the results of preliminary medical tests carried out on the patient.
2. An investment firm wants to know whether a company's stock price will increase or decrease over the next year. The measurements will include features of the company such as its market capitalisation, along with general economic conditions.
3. Astronomers take an images of a distant object in the sky and want to classify it as being a star, a galaxy or a nebula
4. A music genre classification program needs to decide whether a particular song is rock, a pop or hiphop. etc

Classification - Discriminant Analysis

There are many different approaches/algorithms for performing classification, and we will discuss several throughout the course. In this lecture we focus on one called **discriminant analysis**.

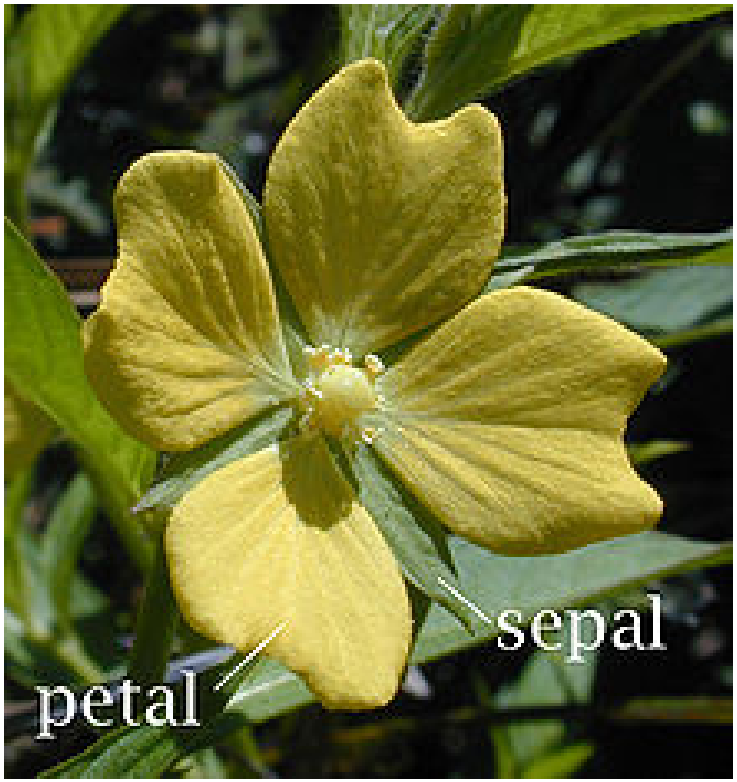
We will work through a particular example in detail, to illustrate the general concepts. This uses the classic Iris flower data set, originally introduced by the famous statistician Roland Fisher back in 1936, where the method of discriminant analysis was first presented.

Suppose that there are two different species of Iris flowers: setosa and versicolor. Given a flower for which we do not the species, we wish to predict which species it belongs to. This is hence an example of binary classification

Classification - Discriminant Analysis

The left image shows an iris setosa flower, the right shows an iris versicolor flower.

Classification - Discriminant Analysis



It is known that the different species of iris flowers vary in the lengths of their sepals and petals. Suppose that we have measured the length of the sepal of the flower.

Classification - Discriminant Analysis

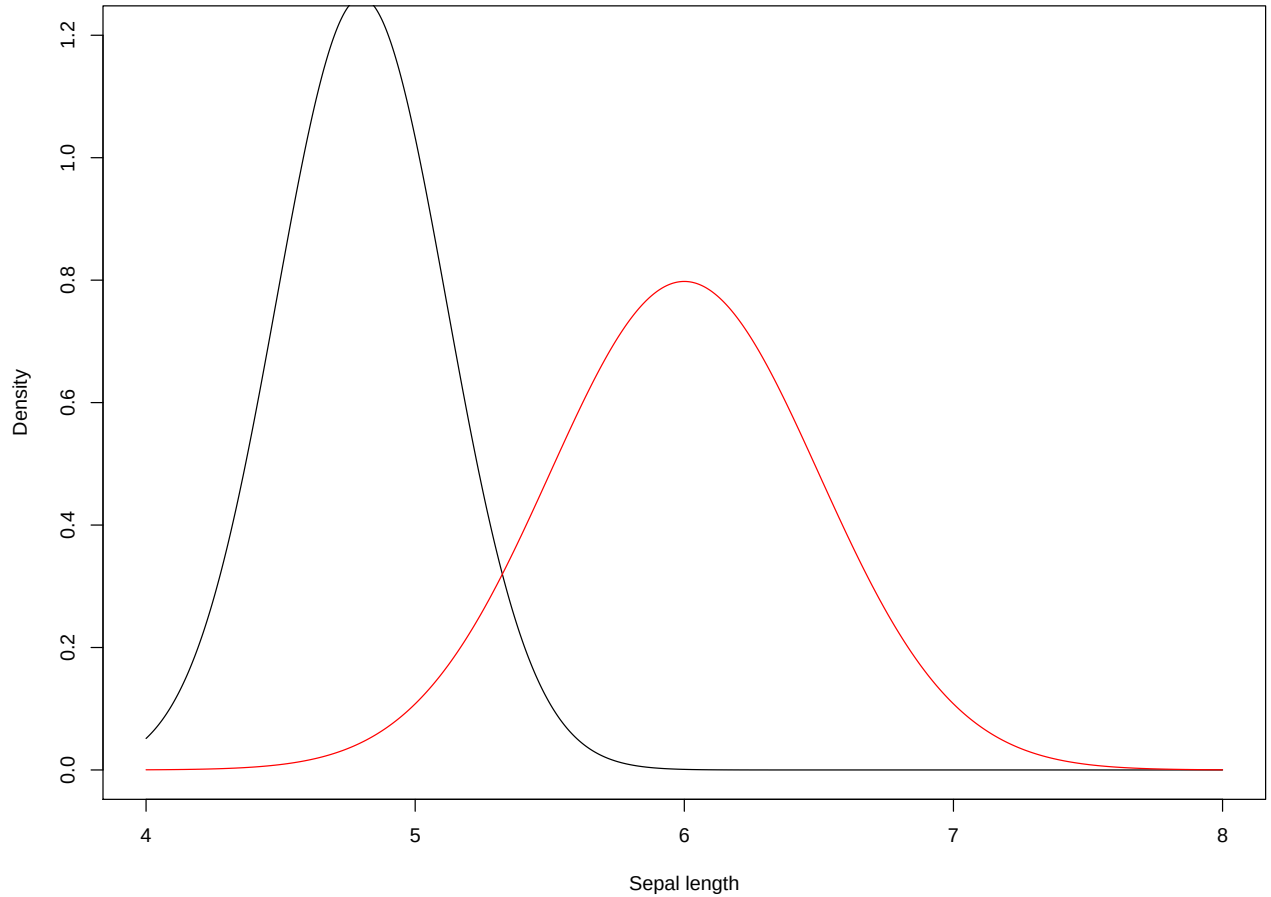
Suppose we know that the sepal length of setosa flowers measured in centimetres has a Normal distribution:

$$N(4.8, 0.1)$$

In other words, even though any two different setosa flowers will have slightly different sepal lengths, on average their length will be 5cm, with a variance of 0.1. Similarly, the sepal length of versicolor flowers is known to have a different Normal distribution

$$N(6, 0.25)$$

Classification - Discriminant Analysis



We can see that if the flower has a sepal length of (e.g.) 7 then it is probably a versicolor, while if it has a sepal length of (e.g.) 4.5 then it is probably a setosa. But how should we classify the flower if it (e.g.) has a sepal length of 5.5?

Classification - Discriminant Analysis

Lets introduce some notation. For a given flower i let x_i denote its sepal length, and y_i denote its species. There are two possible values of the species. We write $y_i = 0$ if the flower is a setosa, and $y_i = 1$ if the flower is a versicolor. We do not know y_i in general, and must predict it given the observed value of x_i.

From the previous slide, we hence know:

$$p(x_i|y_i = 0) = N(4.8, 0.1)$$

$$p(x_i|y_i = 1) = N(6, 0.25)$$

This is the likelihood function, i.e. the probability of getting a particular observed value of x_i given the (unknown) type of flower. In the context of discriminant analysis it is often referred to as the **class conditional density**, i.e. the density function of the observations for each class.

Classification - Discriminant Analysis

To keep things slightly more general, lets instead write:

$$p(x_i|y_i = 0) = N(\mu_0, \sigma_0^2)$$

$$p(x_i|y_i = 1) = N(\mu_1, \sigma_1^2)$$

where $\mu_0, \mu_1, \sigma_0^2, \sigma_1^2$ are known quantities (in this case, they have the values on the previous slide).

We will refer to x_i as the feature or measurement. y_i is called the class label or class indicator.

Note that referring to the two classes as 0 and 1 is conventional.

Classification - Discriminant Analysis

Our goal is to predict the species of the flower given its sepal length. In other words, given x_i we want to predict y_i. In discriminant analysis we do this by computing : $p(y_i|x_i)$, the posterior distribution of y_i given the observed x_i. By Bayes theorem, we have:

$$p(y_i = 0|x_i) = \frac{p(x_i|y_i = 0)p(y_i = 0)}{p(x_i)}$$

$$p(y_i = 1|x_i) = \frac{p(x_i|y_i = 1)p(y_i = 1)}{p(x_i)}$$

Classification - Discriminant Analysis

The $p(y_i = 0)$ term is the prior probability of a flower being a setosa. By 'prior', I mean what we believe about the flower before observing the sepal length (or any other measurement). If there were an equal number of setosa and versicolor flowers in the world, then this would be equal to 0.5. If 90% of iris flowers were setosa, then this would be 0.9, and so on. Lets say that:

$$p(y_i = 0) = \xi$$

$$p(y_i = 1) = 1 - \xi$$

(these must sum to 1)

In the current Iris example, lets assume it is known that $\xi = 0.5$, so both species are equally likely a priori.

Classification - Discriminant Analysis

The $p(x_i)$ term in the denominator is the unconditional probability of observing the sepal length x_i (unconditional means not depending on the particular species). It is hence an average over both species. By the theorem of total probability:

$$p(x_i) = p(x_i|y_i = 0)p(y_i = 0) + p(x_i|y_i = 1)p(y_i = 1)$$

Since this does not depend on y_i, this quantity is the same in the denominator of both $p(y_i = 0|x_i)$ and $p(y_i = 1|x_i)$ on the previous slide. As such, it can be ignored.

Classification - Discriminant Analysis

We hence have:

$$p(y_i = 0|x_i) \propto p(x_i|y_i = 0)\xi$$

$$p(y_i = 1|x_i) \propto p(x_i|y_i = 1)(1 - \xi)$$

And substituting in the likelihood terms:

$$p(y_i = 0|x_i) \propto \xi \frac{1}{\sqrt{2\xi\sigma_0^2}} e^{\left(\frac{-(x_i-\mu_0)^2}{2\sigma_0^2}\right)}$$

$$p(y_i = 1|x_i) \propto (1 - \xi) \frac{1}{\sqrt{2\xi\sigma_1^2}} e^{\left(\frac{-(x_i-\mu_1)^2}{2\sigma_1^2}\right)}$$

Classification - Discriminant Analysis

For example, suppose we take an Iris flower and measure its sepal length, and it turns out to be 5.5cm. We can compute the posteriors easily from the previous side. Rather than doing it by hand, we could also use the R programming language:

```
xi <- 0.5
mu0 <- 4.8; sigma0 <- sqrt(0.1)
mu1 <- 6; sigma1 <- sqrt(0.25)

x <- 5.5

p0 <- xi*dnorm(x,mu0,sigma0)
p1 <- (1-xi)*dnorm(x,mu1,sigma1)
```

This gives: $p0 = 0.054$, $p1 = 0.242$

Classification - Discriminant Analysis

So we have found that:

$$p(y_i = 0|x_i = 5.5) \propto 0.054$$

$$p(y_i = 1|x_i = 5.5) \propto 0.242$$

Note that we have proportionality here rather than equality, since we threw away the constant denominator term $p(x_i)$. As such, these are unnormalised probabilities and they don't sum to 1. To normalise them, we simply rescale them so they sum to 1:

$$p(y_i = 0|x_i = 5.5) = 0.054/(0.054 + 0,242) = 0.18$$

$$p(y_i = 1|x_i = 5.5) = 0.242/(0.054 + 0.242) = 0.82$$

So the probability of the flower being a setosa is 0.18.

Classification - Discriminant Analysis

In practice, we often want to make a definite prediction. Rather than give a probability, we want to actually say whether we believe the flower is setosa or versicolor. This involves creating a **decision rule**, i.e. a way to reduce the posteriors down to a single prediction. An obvious rule is to classify the flower as being a setosa if we find that it is more likely to be a setosa than a versicolor, i.e if:

$$p(y_i = 0|x_i) > p(y_i = 1|x_i)$$

and to classify it as a versicolor if

$$p(y_i = 1|x_i) > p(y_i = 0|x_i)$$

In the above example, we would classify the flower as a versicolor since $0.86 > 0.14$.

Discriminant Analysis - Estimation

This is all nice in theory, but how did we actually know the formulae for the class conditional densities functions? Previously, I simply stated that:

$$p(x_i|y_i = 0) = N(4.8, 0.1)$$

$$p(x_i|y_i = 1) = N(6, 0.25)$$

But in practice we will not know this is true. Instead, these distributions must be learned. For now, we will assume that we believe these distributions are Normal, but that their parameters must be estimated.

Discriminant Analysis - Estimation

For a general classification problem we have class conditionals:

$$p(x_i|y_i = 0) = N(\mu_0, \sigma_0^2)$$

$$p(x_i|y_i = 1) = N(\mu_1, \sigma_1^2)$$

with unknown parameters $\mu_0, \sigma_0^2, \mu_1, \sigma_1^2$.

Discriminant Analysis - Estimation

The usual situation is as follows. We have a **data set** which consists of n previous observations of the (x_i, y_i) pairs where the class labels y_i are **known** . In this example, this means we a random sample of measurements on a set iris flowers from both species, and for each flower we know its true species

We use this data to estimate the parameters of the class conditional densities. In statistics, we call this inference or parameter estimation. In machine learning, they tend to call it training or fitting a model. Once we have learned the class conditional densities, we can then predict the class labels y_i of other observations for which the true class is unknown.

We can estimate the parameters using either frequentist techniques (e.g. maximum likelihood) or a Bayesian approach. For now, lets be frequentists.

Discriminant Analysis - Estimation

Suppose that in our data set there are n^0 setosa flowers, and n^1 versicolor flowers. Let $x_1^0, x_2^0, \ldots, x_{n^0}^0$ denote the sepal lengths of the setosa flowers, and $x_1^1, x_2^1, \ldots, x_{n^1}^1$ denote the sepal lengths of the versicolor flowers. Then, the maximum likelihood estimates are the usual MLEs for a Normal distribution:

$$\hat{\mu}_0 = \frac{1}{n^0}\sum_{i=1}^{n^0} x_i^0, \quad \hat{\sigma}_0^2 = \frac{1}{n^0 - 1}\sum_{i=1}^{n^0} (\hat{\mu} - x_i^0)^2$$

$$\hat{\mu}_1 = \frac{1}{n^1}\sum_{i=1}^{n^1} x_i^1, \quad \hat{\sigma}_1^2 = \frac{1}{n^1 - 1}\sum_{i=1}^{n^1} (\hat{\mu} - x_i^1)^2$$

Discriminant Analysis - Estimation

We can do this in R. The iris dataset is included in the standard R installation, and consists of sepal length measurements of 50 setosa and 50 versicolor flowers. It can be accessed by simply typing 'iris'

```
x0 <- iris[iris$Species == 'setosa',]$Sepal.Length
x1 <- iris[iris$Species == 'versicolor',]$Sepal.Length

mu0 <- mean(x0); sigma0 <- sd(x0)
mu1 <- mean(x1); sigma1 <- sd(x1)

> c(mu0, sigma0)
[1] 5.0060000 0.3524897
> c(mu1, sigma1)
[1] 5.9360000 0.5161711
```

Discriminant Analysis - Estimation

We could also use this data set to estimate the prior probabilities $p(y_i = 0)$ and $p(y_i = 1)$. Assuming that the data set is a genuinelly random sample from the population of iris flowers, the maximum likelihood estimate of $p(y_i = 0)$ is simply the percentage of setosa flowers:

```
xi <- length(x0)/(length(x0)+length(x1))
> xi
[1] 0.5
```

In this case, there are equal numbers of both flowers.

Important: we have to be **very** careful when estimating the prior probabilities, since we are assuming that the proportion of classes in the historical data is equal to the proportion of classes in the population. In practice, this is often not the case since data collection is often biased, and we may have oversampled one class compared to another. As such, we will instead continue to assume that $p(y_i = 0) = 0.5$ rather than estimating it.

Discriminant Analysis - Estimation

Once we have estimated all the parameters, we can plug them into the previous formulae by simply replacing μ_0 with $\hat{\mu}_0$ and so on.

To classify an object, we assign to the class which gives the highest posterior probability when we substitute in these estimated values.

Performance Assessment

In practice we will probably not be classifying only a single object, we will classify multiple objects. For example, there may be several flowers which we want to classify as setosa or versicolor.

Suppose there are m objects we wish to classify. We observe their set of measurements $x_1, x_2, \ldots, x_m$ (one for each object) and wish to predict the corresponding class labels $y_1, y_2, \ldots, y_m$. In this case we **do not know** the class labels – we are trying to predict them using the model we have learned.

We will generally assume that the predictions are made independently. In this case, each y_i is predicted based on x_i independently, using the above method. This gives the set of class predictions $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m$.

The empirical error rate is then the proportion of class labels which we mispredict, i.e:

$$\frac{1}{m} \sum_{i}^{m} I(\hat{y}_i \neq y_i)$$

Performance Assessment

So, a typical classification scenario works as follows: we have a data set of observations (x_i, y_i) for which the true class labels y_i are known. We use these estimate the model (i.e. learning the parameters of the class conditionals, etc). We then use the resulting model to predict the class label of future objects, where the class labels are unknown.

Typically, we want to have some idea of how well our model will perform in practice, i.e. we want to estimate what the asymptotic error rate (from the previous slide, as $m \rightarrow \infty$) is going to be. This will be a combination of both the irreducible (Bayes) error rate, and the model error that comes from the fact that we are estimating the model parameters from the data rather than assuming that they are known exactly.

Performance Assessment

A simple approach would be to use the training set to estimate the error rate directly. In other words, once we have estimated the model parameters, we classify each object in the training set using the resulting model, and count the proportion of objects which we classify incorrectly.

Since all the true class labels in the training set are known exactly, we can do this directly. The following two slides shows R code for this, from scratch:

Performance Assessment

```
#first separate out the different classes and estimate parameteres
x0 <- iris[iris$Species == 'setosa',]$Sepal.Length
x1 <- iris[iris$Species == 'versicolor',]$Sepal.Length

mu0 <- mean(x0);
sigma0 <- sd(x0)
mu1 <- mean(x1)
sigma1 <- sd(x1)

#to make the code for classification easier
#we combine both classes together
x <- c(x0,x1)

#'truey' contains the true class labels
truey <- c(rep(0,length(x0)), rep(1,length(x1)))
```

Performance Assessment

```
#prior, this can be omitted since its the same for both classes
xi <- 0.5

#this vector stores the predictions
predictions <- length(x)

#we now classify each observation
for (i in 1:length(x)) {
  p0 <- xi * dnorm(x[i],mu0,sigma0)
  p1 <- (1-xi) * dnorm(x[i],mu1,sigma1)
  if (p0 > p1) {
    predictions[i] <- 0
  } else {
    predictions[i] <- 1
  }
}
sum(predictions!=truey)/length(truey)
[1] 0.11
```

so 89% of flowers were classified correctly and 11% were classified incorrectly

Performance Assessment

However this approach to estimating the error rare is **not recommended**. The problem is that we are using the training set both for estimating the model parameters and for estimating how well the model will perform when it comes to predicting new objects. The data is hence being used twice. This leads to overfitting – we will underestimate the true error rate since

A standard way to avoid this is to instead randomly split our data into two separate sets . The first set is used to estimate the model parameters, and the second set is used to estimate the error rate.

The first is known as the **training set** while the latter is the **test set**. Since the test set is not used to train the model (i.e. learn the parameters), it can be considered as 'new' data. As such, the error rate on the test set should be an unbiased estimate of the true error rate.

Performance Assessment

Typically the test set is taken to be between 10% and 50% of the data, and we randomly select the observations to belong to it.

The larger the test set, the better our estimate will be of the prediction error, since we have more observations with which to estimate it. However a large test set leads to a smaller training set, which means there will be fewer observations with which to learn the model parameters. As such, there is an inherent trade-off.

In a future lecture we will discuss cross-validation which mitigates this problem to some degree.

Performance Assessment

In our Iris example we will choose the test set to contain 1/3 (33%) of the observations. We hence randomly assign 1/3 of the observations to the test set and the remainder to the training set. In R we can do this using the **sample** function:

```
x0 <- iris[iris$Species == 'setosa',]$Sepal.Length
x1 <- iris[iris$Species == 'versicolor',]$Sepal.Length
x <- c(x0,x1)
y <- c(rep(0,length(x0)), rep(1,length(x1)))

n <- length(x)
```

Performance Assessment

```
#randomly choose the training set. We do it by selecting indexes so
#that we assign paired vales of x and y to both sets. The set.seed
#function initialises the random number generator so that we get
#the same random numbers every time we run the code (so the
# results can be replicated)

set.seed(1)
trainindices <- sample(1:n, round(0.66*n), replace=FALSE)
trainx <- x[trainindices]
trainy <- y[trainindices]
testx <- x[-trainindices]
testy <- y[-trainindices]
```

Performance Assessment

We can then predict the class labels for the test set and compute the error rate

```
#first estimate the parameters. The 'which' function can ue
#used to select the elements of a vector that fulfill a certai
#conditon -- here it is used to extract the class 0 and class
#observations, based on the value of y (i.e. the class)
mu0 <- mean(trainx[trainy==0])
sigma0 <- sd(trainx[trainy==0])
mu1 <- mean(trainx[trainy==1])
sigma1 <- sd(trainx[trainy==1])
xi <- 0.5
```

Performance Assessment

```
predictions <- numeric(length(testx))
for (i in 1:length(testx)) {
  p0 <- xi * dnorm(testx[i],mu0,sigma0)
  p1 <- (1-xi) * dnorm(testx[i],mu1,sigma1)
  if (p0 > p1) {
    predictions[i] <- 0
  } else {
    predictions[i] <- 1
  }
}
sum(predictions!=testy)/length(testy)
```

The error rate here is 0.176 so around 17.6% of observations are misclassified.

Performance Assessment

Two points to note:

1. Since the assignment of observations to the training and test sets is random, we will get a slightly different test set error rate for each random assignment. If we ran the above code (without set.seed(1)) then we would get a different error rate each time. This is unavoidable due to randomness. This will be important in our later discussion of cross-validation,
2. The Bayes error is the minimum possible error rate. However for any finite sample, the empirical error rate might be lower than this value (perhaps we get lucky on the test set and classify everything correctly)

Discriminant Analysis - Multiple Classes

In the previous example we assumed that there were only two classes, i.e. that y_i only had two possible values. But the above framework extends naturally to the case where there are more than two classes. Suppose that there are C classes, i.e. that y_i can take values in the set $(1, 2, \ldots, C)$.

The analysis proceeds very similar to before. For each possible class c, we compute $p(y_i = c|x_i)$, and then assign to the class for which this quantity is maximised.

i.e. we assign observation x_i to class r where

$$r = \arg\max_r p(y_i = c|x_i), \quad c \in \{1, 2, \ldots, C\}$$

Discriminant Analysis - Multiple Classes

Example: The full Iris dataset actually contains 3 species: setosa, versicolor, and virginica. As before, we write $y_i = 1$ if the flower is a setosa, and $y_i = 2$ if the flower is a versicolor. But we now also write $y_i = 3$ if the flower is a virginica. Our model is:

$$p(x_i|y_i = 1) = N(\mu_1, \sigma_1^2)$$

$$p(x_i|y_i = 2) = N(\mu_2, \sigma_2^2)$$

$$p(x_i|y_i = 3) = N(\mu_3, \sigma_3^2)$$

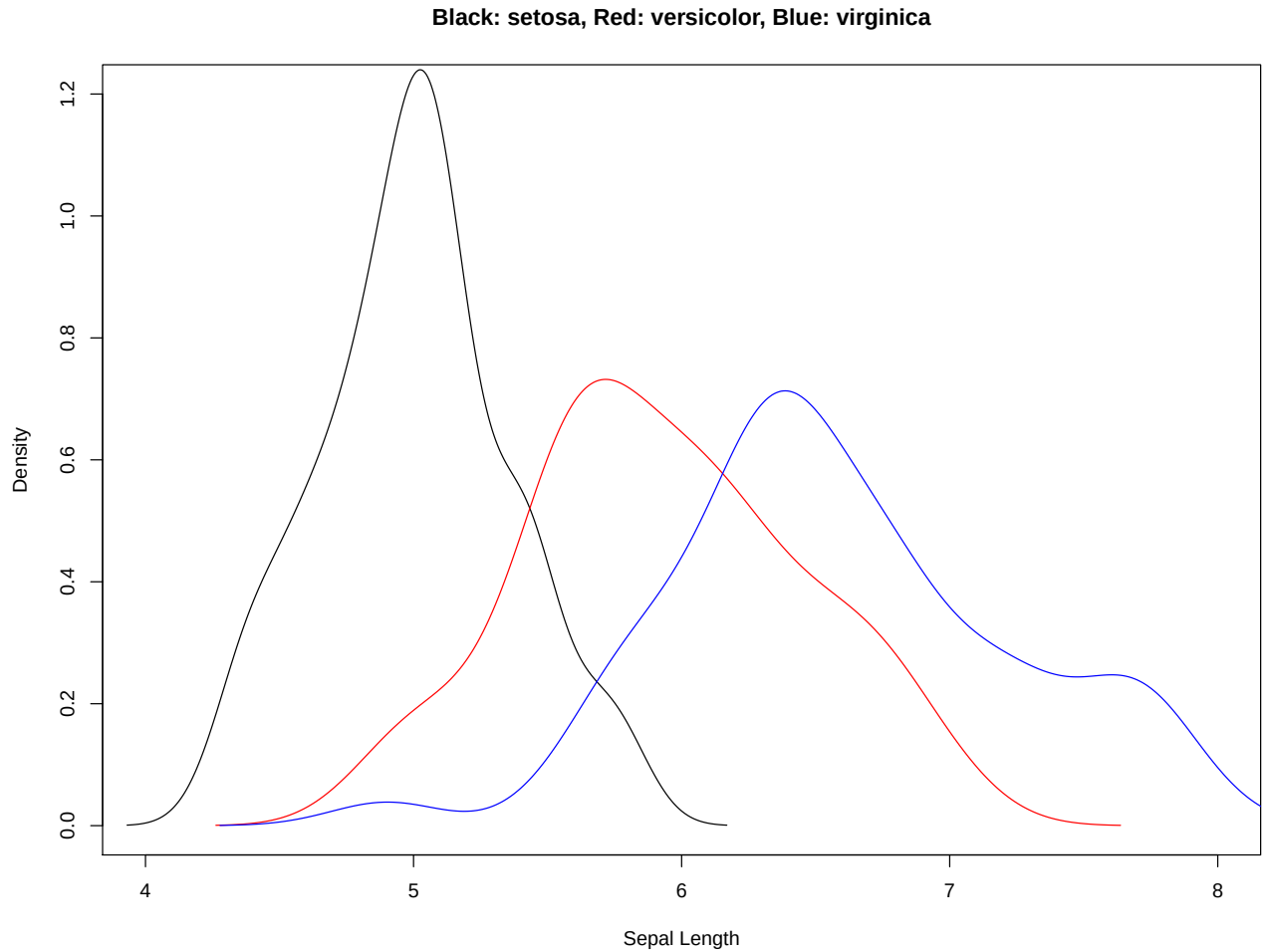
where the parameters must be estimated from the data, as before.

Discriminant Analysis - Multiple Classes

For a general classification problem with C classes we write:

$$p(x_i|y_i = c) = N(\mu_c, \sigma_c^2), \quad C \in (1, 2, \ldots, C)$$

Empirical sepal length distribution of the 3 Iris classes



Discriminant Analysis - Multiple Classes

As before, for each possible class:

$$p(y_i = c|x_i) \propto p(y_i = c)p(x_i|y_i = c)$$

In the absence of strong prior information we will again often take the class priors to be equal, i.e. $p(y_i = c) = 1/3$ for all c.

Although we could compute an explicit decision boundary as in the binary classification case, the algebra is slightly awkward, so we will instead just compute $p(y_i = c|x_i)$ explicitly for each class and then assign to the class for which this is maximised

Lets see how to do this in R...

Discriminant Analysis - Multiple Classes

First load the data and create training/test sets as before (note I am renumbering the classes to be 1,2,3 rather than 0,1, so that setosa is now class 1)

```
x1 <- iris[iris$Species == 'setosa',]$Sepal.Length
x2 <- iris[iris$Species == 'versicolor',]$Sepal.Length
x3 <- iris[iris$Species == 'virginica',]$Sepal.Length

x <- c(x1,x2,x3)
y <- c(rep(1,length(x1)), rep(2,length(x2)), rep(3,length(x3)))

n <- length(x)
set.seed(1)
trainindices <- sample(1:n, round(0.66*n), replace=FALSE)
trainx <- x[trainindices]
trainy <- y[trainindices]
testx <- x[-trainindices]
testy <- y[-trainindices]
```

Discriminant Analysis - Multiple Classes

Now estimate the model parameters using the training set:

```
xi1 <- 1/3; xi2 <- 1/3; xi3 <- 1/3 #priors

mu1 <- mean(trainx[trainy==1]); sigma1 <- sd(trainx[trainy==1])
mu2 <- mean(trainx[trainy==2]); sigma2 <- sd(trainx[trainy==2])
mu3 <- mean(trainx[trainy==3]); sigma3 <- sd(trainx[trainy==3])
```

Discriminant Analysis - Multiple Classes

Now predict on the test set:

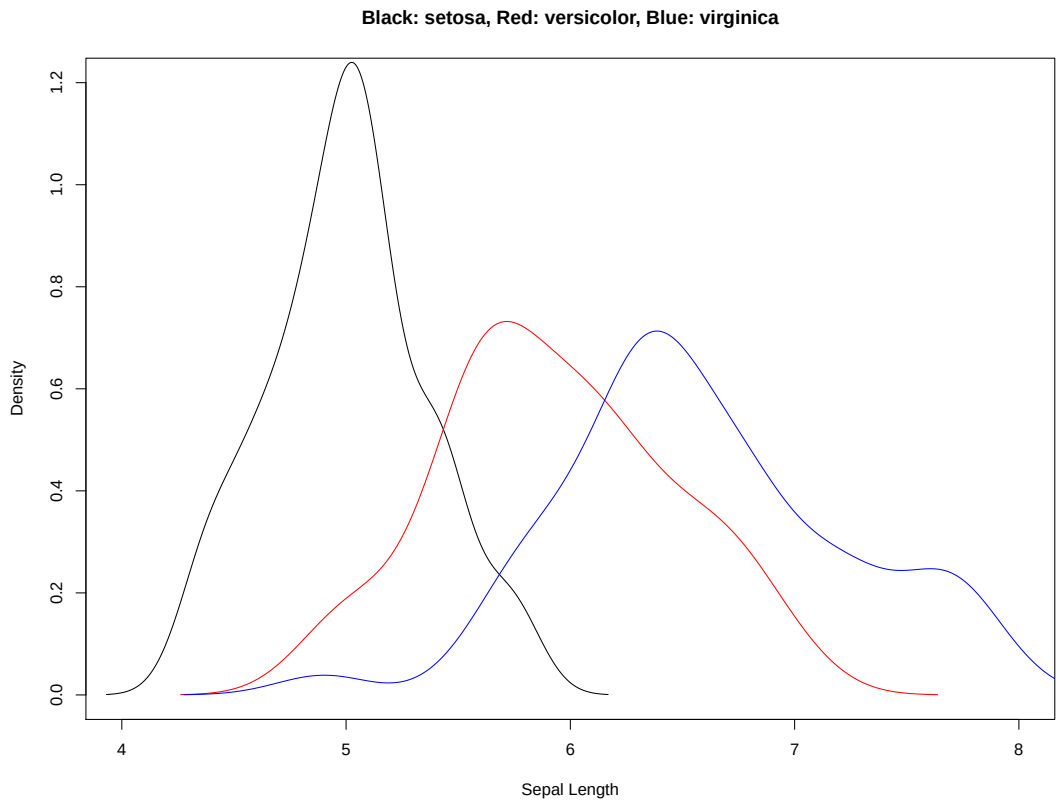
```
ypreds <- numeric(length(testx))

for (i in 1:length(testx)) {
  p1 <- xi1 * dnorm(testx[i], mu1, sigma1)
  p2 <- xi2 * dnorm(testx[i], mu2, sigma2)
  p3 <- xi3 * dnorm(testx[i], mu3, sigma3)

  if (p1 == max(p1,p2,p3)) {
    ypreds[i] <- 1
  } else if (p2 == max(p1,p2,p3)) {
    ypreds[i] <- 2
  } else {
    ypreds[i] <- 3
  }
}
```

Discriminant Analysis - Multiple Classes

The error rate here is 27.4%. This is substantially higher than in the two class case, but perhaps to be expected given how much the virginica distribution overlaps with the versicolor, making it hard to distinguish between these two classes:



Discriminant Analysis - Multiple Classes

In multiple class problems, it can often to be useful to look at the error rates for each specific class in additional to the overall error rate. This can give insight into where our classifier is doing well/badly. Perhaps there are particular classes which we are very good at correctly identifying (i.e when an object belongs to this class, we get the prediction right) and other classes where we struggle to make correct predictions.

A useful visualization tool is the **confusion matrix**. In a C class problem, the confusion matrix has C rows and C columns. Each row *c* of the matrix represents the **predictions** that we made for class *c*. Each column represents the true classes.

It is probably easier to explain this by just showing what a confusion matrix looks like....

Discriminant Analysis - Multiple Classes

We can create a confusion matrix in R by using the confusionMatric() function in the.*caret* package. This takes two arguments: the first is our predictiosn for the test set, and the second is the true test set class indicators:

```
#install.package('caret') #needed if package isnt installed

library(caret) #load the caret library
confusionMatrix(ypreds,testy)
```

Discriminant Analysis - Multiple Classes

	Reference		
Prediction	1	2	3
1	17	1	0
2	1	10	9
3	0	3	10

The first column tells us that there were 18 test set flowers which were truly class 1 (i.e. setosa). Of these, 17 were correctly predicted as being setosa, while 2 was incorrectly predicted as being versicolor.

The second column tells us that there were 14 versicolor flowers in the test set. Of these, 10 were correctly classified, while 1 was incorrectly labelled as setosa and 3 were incorrectly labelled as virginica.

Finally the third column tells us that there were 19 virginica flowers in the test set of which only 10 were correctly classified, with the other 9 being incorrectly labelled as versicolor

So we can see that the main driver of the overall error rate is that we struggle to correctly identify virginica flowers

Discriminant Analysis - Non-Gaussian Class Conditionals

Discriminant Analysis - Non-Gaussian Class Conditionals

In the previous examples, we assumed that the class conditional densities were Gaussian, i.e:

$$p(x_i|y_i = c) = N(\mu_c, \Sigma_c)$$

However in many cases this will not be a plausible assumption. For example, perhaps some of the x features are binary or categorical. Since they are not continuouis, the Gaussian assumption makes no sense.

The general procedure is nonetheless the same; we define an appropriate parametric form for the class conditionals, then compute $p(y_i|x_i)$ for each observation that we wish to classify

Discriminant Analysis - Example

Lets return to stylometry.

Given a text x_i represented as a vector of feature counts, we wish to predict the author given a set of C potential authors. This is hence a multi-class problem with C classes.

We collect a large sample of known writing for each author to define the training set. This is used to learn the C class conditional densities $p(x_i|y_i)$ and we use these to classify the disputed text.

Discriminant Analysis - Example

Suppose for each author A_c we have collected n_c words from their published writing. For well known authors, this is simply a case of accessing published electronic copies of their various books/etc and collecting the words.

We then count how many times each of the previous 70 function words occurs. This defines a 70 element vector for each author.

We also count the number of non-function words they used (i.e. any word that isnt in the above list) and put this on the end of the vector, hence making a 71-element vector in total.

Discriminant Analysis - Example

So for example, suppose we find a set of books written by an author and find they contain 100,000 words in total. The word ‘a’ occurs 730 times, the word ‘all’ occurs 203 times, and so on.

80,000 of the words were not function words

The 71-element feature vector is then

$$x_i = (730, 203, \ldots, 80000)$$

We model these vectors as a Multinomial distribution. Note that this means we are working with **counts** rather than **proportions**, i.e. the vector is **not** normalised.

Discriminant Analysis - Example

The **Multinomial** distribution is a generalisation of the Binomial distribution for modelling categorical random variables which can take on more than 2 values.

Suppose a dice has C sides, and we roll it N times. Each of these independent rolls gives a value in the set $\{1, 2, \ldots, C\}$. We then count how many times each number occurred, and collect this information into a vector $x = (x_1, \ldots, x_C)$ where x_1 is the number of times the dice rolled a '1', and so on.

The dice might be biased, so not all numbers are equally likely to occur. Define the probability vector $\theta = (\theta_1, \ldots, \theta_C)$ where θ_i is the probability of the dice rolling i.

Then, we say that x has a Multinomial distribution with parameter θ and write:

$$x \sim Multinomial(N, \theta)$$

Note that when $C = 2$ this is just the Binomial distribution.

Discriminant Analysis - Example

The Multinomial distribution has density function:

$$p(x|\theta) = \frac{n!}{x_1!x_2!\ldots x_C!}\theta_1^{x_1}\ldots\theta_C^{x_C}$$

Given a sample of observations from the Multinomial distribution, it can be shown (exercise) that the maximum likelihood estimate of θ is:

$$\hat{\theta} = (\hat{\theta_1}, \ldots, \hat{\theta_C})$$

where

$$\hat{\theta_i} = \frac{x_i}{N}$$

i.e. the empirical proportion of observations which fell into category i

Discriminant Analysis - Example

In stylometry, it is common to model texts using a bag-of-words model. Essentially this means that we treat every word the author writes as being an independent random draw from his vocabulary of words.

For a given author, c we define the 71-element parameter vector $\theta_c = (\theta_{1,c}, \theta_{2,c}, \ldots, \theta_{71,c})$ where $\theta_{1,c}$ is the probability of a random word from author c being 'a', $\theta_{2,c}$ is the probablty of a randomly selected word being 'all', and so on

Discriminant Analysis - Example

This leads to the following model: suppose we are presented with a text x_i containing N_i words that has been written by an unknown author. Here x_i is a 71-element vector containing the counts of how many of each time each of the 70 function words occurred, along with the number of non function words.

$$p(x_i|y_i = c) = Multinomial(x_i; N_i, \theta_c)$$

This is a similar model to the previous Normal discriminant setting, except the class conditional densities are now multinomial.

Discriminant Analysis - Example

Classification is done in the exact same way as before. Given a new text x_i with an unknown author and a set of C possible candidate authors, we compute

$$p(y_i = c|x_i) \propto p(x_i|y_i = c)p(y_i = c)$$

for each $c \in \{1, 2, \ldots, C\}$ and then classiy the book as being written by the author for which this is maximised.

In practice we will have to learn $p(x_i|y_i = c)$ by estimating the parameters of the Multinomial distributions from a training set (i.e. a set of books which we know were written by the author

Discriminant Analysis - Example

The authorshiptrainingset.csv file available on the course website contains the function word counts for the following 5 authors:

1. George R R Martin (all Game of Thrones novels)
2. JK Rowling (all Harry Potter novels)
3. Terry Pratchett (all published novels)
4. Robert Jordan (all Wheel of Time novels)
5. Charles Dickens (all published novels)

These were compiled by counting the function words in digital versions of their published works

The first 4 authors are all in the fantasy genre so may be ‘similar’, while Dickens is included to represent an author from a very differnet time period

Discriminant Analysis - Example

The cuckooscalling.csv file on the course website contains the function word counts for JK Rowling's 'Cuckoos Calling' novel which she wrote under a pseudonym to hide her name

We will use discriminant analysis to find which of the previous 5 authors is the most likely author of this book – i.e. can we correctly attribute it to JK Rowling

The classes in the training set are in the order of the previous slide, so Charles Dickens is class 1, George R R Martin is class 2, JK Rowling is class 3, and so on.

In R, the dmultinom() function evaluates the density of the multinomial distribution.

Discriminant Analysis - Example

```
trainset <- as.matrix(read.csv('~/authorshiptrainingset.csv',
  header=FALSE))
testset <- as.numeric(read.csv('~/cuckooscalling.csv',
  header=FALSE))

C <- dim(trainset)[1]
pis <- rep(1/C, C)

thetas <- matrix(numeric(C*dim(trainset)[2]),nrow=C)
for (i in 1:C) {
  thetas[i,] <- trainset[i,]/sum(trainset[i,])
}

posteriors <- numeric(C)
for (i in 1:C) {
  posteriors[i] <- log(pis[i]) + dmultinom(testset,
    prob=thetas[i,],log=TRUE)
}
```

Discriminant Analysis - Example

The unnormalised (log) posteriors are:

```
> posteriors
[1] -4502.421 -3890.305 -2088.049 -3766.335 -2755.329
```

So we would assign this novel to class 3, which is JK Rowling. Therefore, discriminant analysis has correctly identified the true author.

K Nearest Neighbours

An Alternative Approach - K Nearest Neighbours

The classification methods we have considered so far have been based on an explicit probability models. We choose a parametric form for the class densities $p(x_i|y_i)$ and then learn the parameters.

This is essentially how a statistician would think about classification. In classical statistics, we usually think in terms of building probability models by trying to model the data generation mechanism explicitly

As we shall see throughout this course, not all algorithms used in machine learning are based on such probability models. A simple example is the approach is taken by the K-nearest netighbours (KNN) classification algorithm.

K Nearest Neighbours

The idea behind KNN is quite simple: suppose we have a training set consisting of the pairs (x_i, y_i) where the x_i observations are K dimensional feature vectors. Suppose that we have a new observation with feature vector $\tilde{x}$ that we wish to classify.

To do this, we compute the distance between $\tilde{x}$ and every observation in the training set in order to find which one is closest to it. We then assign $\tilde{x}$ to the same class as this closest observation. Slightly more formally, define:

$$d_i. = d(x_i, \tilde{x}), \quad \text{for some distance function d}$$

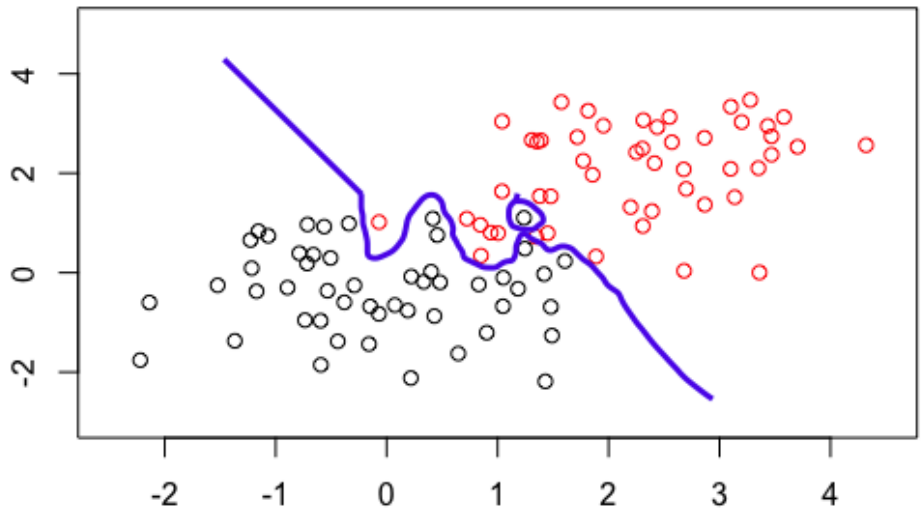
Let

$$r = \arg\min_i d_i$$

where arg min denotes the index of the minimum value. Then we predict $\tilde{y} = y_r$

An Alternative Approach - K Nearest Neighbours

This simple algorithm produces a very complex decision boundary. The below plot shows an example with two classes with two features. New oservations to the left of the blue line will be allocated to the black class, while new observations to the right will be allocated to the red class.



Note that new observations which fall close to the black observation that is 'inside' the red class territory will be assigned to the black class. This is a very non-linear boundary!

An Alternative Approach - K Nearest Neighbours

The first step to implementing the KNN algorithm is to choose an appropriate distance function. This choice will depend on the specific classification problem, but typical choices are the L_1 and L_2 norms.

Given two vectors $x = (x_1, \ldots, x_K)$ and $x' = (x'_1, \ldots, x'_K)$, we could have:

$$d_{L_1}(x, x') = \sum_{k=1}^{K} |x_k - x'_K|$$

$$d_{L_2}(x, x') = \sqrt{\sum_{k=1}^{K} (x_k - x'_K)^2}$$

K Nearest Neighbours

When working with **any** distance based algorithm, we need to make sure that all the different features are on the same scale.

Suppose for the Iris data that we had measured sepal length in centimetres, and measured petal length in millimetres. Suppose that for one flower, the sepal length and petal length were both 1cm, and for another flower they were both 2cm.

Then the 'distance between the sepal lengths would be based on the quantity $2 - 1 = 1$, while the distance between the petal lengths would be based on $20 - 10 = 10$, since we are measuring in millimetres. As such, the petal length measurement would completely dominate the clsssification

K Nearest Neighbours

To avoid this, we either standardise or normalise each feature separately to put them on the same scale. There are several ways to do this, but perhaps the most common is to rescale each individual feature to have mean 0 and standard deviation 1.

If we think in terms of the data being a matrix (with each observation being a row, and each feature a column) this means we are rescaling the columns. Let μ_k and σ_k be the sample mean and standard deviation of column k (NOT class k, we are doing this for the whole data at once, not for each class separately)

Then for a vector $x_i = (x_{i,1}, \ldots, x_{i,K})$, the standardised version is $x_i^* = (x_{i,1}^*, \ldots, x_{i,K}^*)$ where:

$$x_{i,k}^* = \frac{x_{i,k} - \mu_k}{\sigma_k}$$

K Nearest Neighbours

The KNN algorithm we have presented has simply assigned each new observation $\tilde{x}$ to the class of its nearest neighbour. But this intuitively feels like it will make a lot of errors when classifying. In the previous picture, we saw that the decision boundary is very 'wiggly' and will potentially overfit each class.

To make the KNN algorithm slightly more robust, we can modify it as follows. Rather than simply finding the training set observation which is closest to $\tilde{x}$ we can instead find the k closest observations (for example, $k = 3$) and assign it to the class which is most widely represented among them.

For example suppose we are doing a classification problem with 3 classes. We run KNN with $k = 3$. Given a new observation $\tilde{x}$ we find the 3 closest observations in the training set. Suppose their associated class labels y are $(1, 1, 2)$. Then we would assign the new observation to class 1, since this is the majority class.

K Nearest Neighbours

Increasing the value of K results in a smoother decision boundary and less overfitting. How we choose which value of K to use? We will see in the next topic, when we discuss cross-validation.

This is the previous example again, with $k = 3$

K Nearest Neighbours

We will use R to run the KNN algorithm on the Iris data. As always, we start by splitting the data into training and test sets:

```
set.seed(1)
n <- dim(iris)[1]
trainindices <- sample(1:n, round(0.66*n), replace=FALSE)
trainset <- iris[trainindices,]
testset <- iris[-trainindices,]

trainlabels <- trainset[,5] #class labels
testlabels <- testset[,5] #class labels
traindata <- trainset[,-5]
testdata <- testset[,-5]
```

K Nearest Neighbours

To do the prediction, you can use the myKNN function that I provide in the stylometryfunctions.R file. This also requires loading the 'class' library. You can do this as follows:

```
install.packages('class') #only ever need to do this once
library(class)
```

K Nearest Neighbours

Now to do the predictions:

```
#KNN wth K=1
preds <- myKNN(traindata, testdata, trainlabels, k=1)
sum(preds==testlabels)/length(preds)
[1] 0.9019608

#KNN with K=3
preds <- myKNN(traindata, testdata, trainlabels, k=3)
sum(preds==testlabels)/length(preds)
[1]0.9803922
```

(Note: this function uses the *L2* distance. Also if we have a tie, for example if $k = 4$ and the 4 closest observations have class labels $(1, 1, 2, 2)$ then one of the majority classes are chosen at random)

K Nearest Neighbours

Now lets repeat the above stylometry example. We should use $K = 1$ here since each class only has a single member (since we are combining all authors books together)

```
#first read in the data
traindata <- as.matrix(read.csv('~/authorshiptrainingset.csv',
  header=FALSE))
testdata <- as.numeric(read.csv('~/cuckooscalling.csv',
  header=FALSE))

trainlabels <- 1:5

preds <- myKNN(traindata, testdata, trainlabels, k=1)
preds
[1] 3
```

Recall that class 3 is J. K. Rowling as expected