Part 1: Basics of multivariate analysis

Ian Jermyn

Miscada / ASML / Epiphany

Literature

HASTIE, T., TIBSHIRANI, R., & FRIEDMAN, J. (2001) [H]
The Elements of Statistical Learning. Springer. Sec 14.3

JAMES, G., WITTEN, D., HASTIE, T. and TIBSHIRANI, R. (2013) [J]
An Introduction to Statistical Learning: With Applications in R (2nd Edition). Springer. Sec 12.4

MURPHY, K.P. (2012) [M]
Machine Learning: A Probabilistic Perspective, The MIT Press. Section 2.5, 4.1, 14.4

First: What is learning?

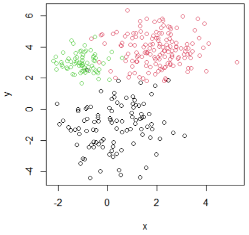
- Learning means finding probability distributions to model data Y.
- We will see many examples of this in Foundations:
  - Estimating the parameters a of a parametric model $P(Y \mid a)$.
- So we will know how to do learning.

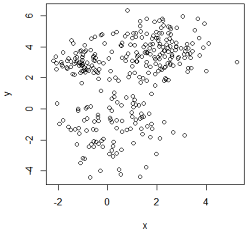
Supervised learning

- Special case of learning.
- ‘Supervised’ means that the data comes in pairs:
  - In addition to Y there are conditioning variables X.
  - The task is to model $P(Y \mid X)$.
- Essentially the same framework as ‘conditional models’ from Foundations, so we will know how to do that too.
- What about ‘unsupervised’?

Unsupervised learning

- Again, a special case of learning.
- 'Unsupervised' means one of two things:
  1. The distribution does not depend on X;
  2. We simply do not know the corresponding X values.
- Case 1 seems straightforward: there is no X!
- Case 2 seems impossible: X could be anything!
- Both these things are true, so why do we have this submodule?!





Why we have this submodule

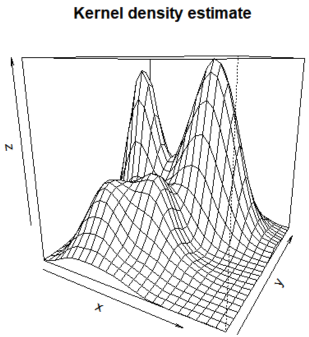
Case 1:

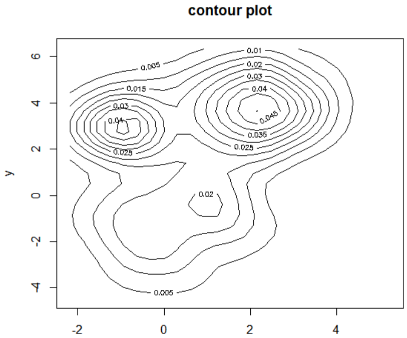
- To go beyond the simple parametric models we have been looking at:
  - Mixture models: combining many simple parametric models.
  - Nonparametric models: effectively an ‘infinite’ number of parameters.
  - Dimensionality reduction: models concentrated on lower-dimensional subsets and/or depending on subset of dimensions.

Case 2:

- To use these more complex models to address problems in which X is unknown. Specifically:
  - Clustering: where X is assumed to take values in a finite set.

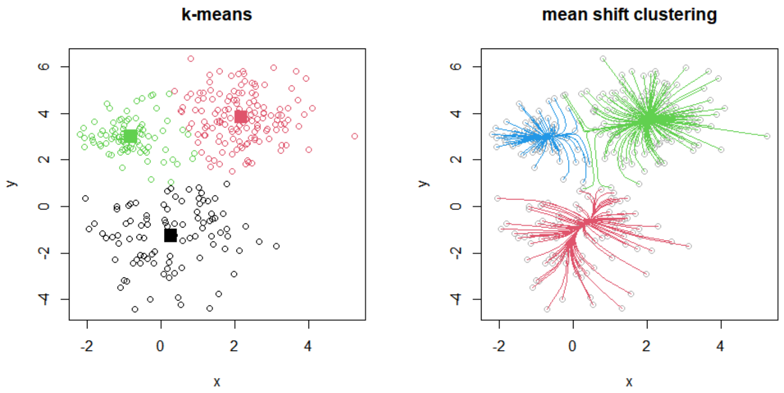
Flavours of unsupervised learning: Density estimation





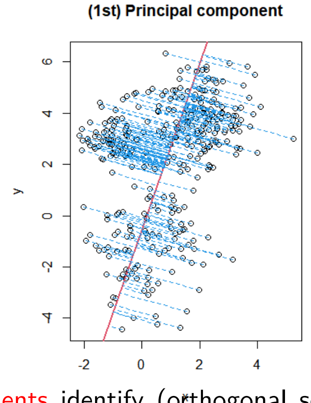
- Kernel density estimators are nonparametric: number of parameters can change with quantity of data.
- There also exist parametric density estimators, such as mixture models.

Flavours of unsupervised learning: Clustering



- Different clustering algorithms give different outcomes.
- Assessing goodness-of-fit in unsupervised learning is difficult.

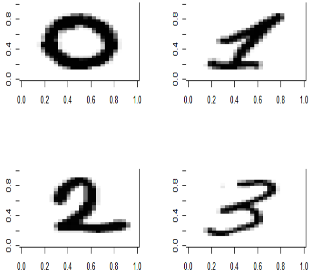
Flavours of unsupervised learning: Principal components and curves

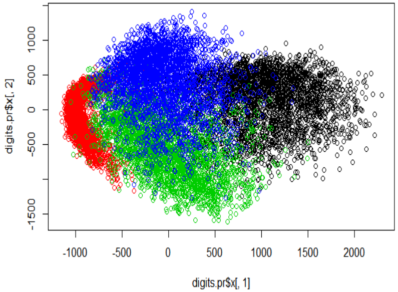


- Principal components identify (orthogonal sequences of) directions of maximum variability.
- Deep (generative) latent variable models / autoencoders.

Example application

Handwritten Letter recognition

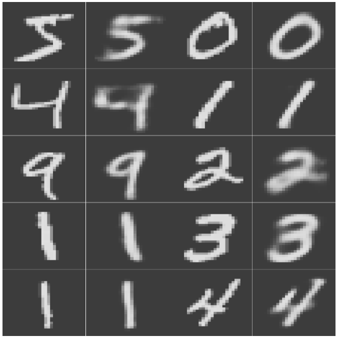


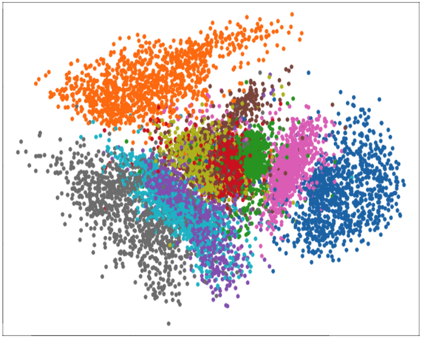


First and second principal components of 12000 handwritten digits 0,1,2, and 3.

Example application

Handwritten Letter recognition





Reconstructions and latent space of a variational autoencoder (VAE) using simple multi-layer perceptrons (MLPs).

An outline of ASML (UL)

Part 1: Basics of multivariate analysis (ca. 3 Lectures): k-means, variance matrix, multivariate normal distribution.

Part 2: Density estimation (ca. 3 Lectures): Gaussian mixtures, kernel density estimation.

Part 3: Cluster analysis (ca. 3 Lectures): Mixture- and -density based clustering; assessing goodness of fit.

Part 4: Dimensionality Reduction (ca. 3 Lectures): Principal component analysis and (deep) autoencoders.

Overview over the remainder of Part 1

We will...

- look in detail into k-means (firstly conceptually, then based on the notion of dissimilarities); as well as some variants including k-medoids;
- discuss the estimation of mean, variance, and correlation matrix of a random vector;
- recall the normal distribution, and introduce the multivariate normal distribution;
- define Mahalanobis distances (another dissimilarity) and use them to check for outliers and multivariate normality.

K-means clustering

Probably the most famous clustering technique.

Objective: For a fixed number of clusters, find a set of clusters which minimizes the sum of squared distances between data and their cluster centres (to be formalized later).

This is a special case of Gaussian mixtures, which we will study later in the course.

K-means clustering

One can show that a local minimum of this objective is achieved by the following algorithm. For a data set with n observations:

1. Select the number of classes, K, in advance.
2. Randomly select K out of the n cases as initial cluster centres.
3. Then repeat
   (i) Assign all observation to their nearest centre (producing a partition in K parts)
   (ii) Compute the mean of each partition, producing updated centres
4. ... until the centres stop moving.

Illustration: K-means

Consider again the unlabelled data set illustrated previously.

Set $K = 3$ (subjective choice by data analyst, expert knowledge,...).

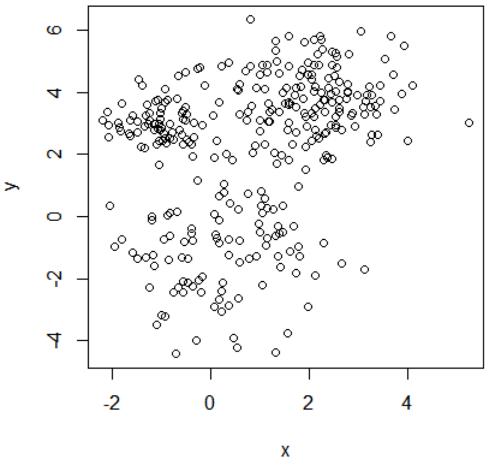
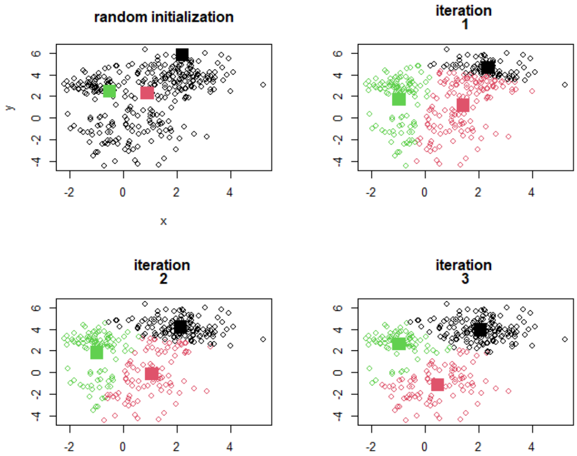


Illustration: K-means



Analysis in R: Reading in data

```
intro.dat <- read.table(
    "http://www.maths.dur.ac.uk/~dma0je/Data/intro-asml2.dat",
    header=TRUE
    )
```

```
head(intro.dat)
```

```
##            x          y
## 1 -0.4720543 -1.3495412
## 2 -2.0489395  0.3468624
## 3 -1.3263892 -1.3380228
## 4  1.7823161 -0.9631754
## 5  0.5540121  0.2210657
## 6 -0.8238523  0.1000506
```

K-means in R

```
k3 <- kmeans(intro.dat, centers=3)
k3$centers
```

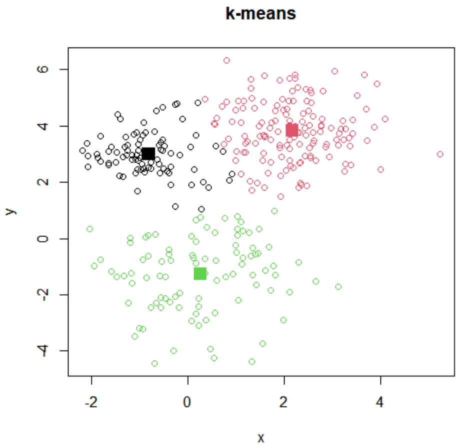
```
##            x         y
## 1  0.2621725 -1.267684
## 2 -0.8169763  2.996031
## 3  2.1752499  3.826271
```

```
k3$cluster[1:20]
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 
##  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

Plotting k-means output

```
plot(intro.dat,
   col=k3$cluster,
   main="k-means")

points(k3$centers, pch=15,
   cex=2, col=c(1,2,3))
```



Dissimilarity

To understand on a deeper level what k-means is trying to achieve, one needs to introduce the concept of a dissimilarity, $D(\boldsymbol{x}_i, \boldsymbol{x}_i')$ between two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_i' \in \mathbb{R}^p$ [H Sec 14.3.2].

Often, these dissimilarities are defined component-wise, based on dissimilarity measures between the the values of the jth variable (attribute),

$$d_j(x_{ij}, x_{i'j}) = \ell(x_{ij} - x_{i'j})$$

with a loss function $l(\cdot)$; for instance,

- $\ell(z) = z^2$ (squared error loss);
- $l(z) = |z|$ (absolute loss).

Dissimilarity

The p attribute-wise dissimilarities can be merged into a single overall measure of dissimilarity between objects i and i' via

$$D(\boldsymbol{x}_i, \boldsymbol{x}_i') = \sum_{j=1}^{p} w_j d_j(x_{ij}, x_{i'j})$$

where w_j is a set of weights.

Simple example: Set $l(z) = z^2$ and $w_j = (1, ..., 1)$. Then

$$D(\boldsymbol{x}_i, \boldsymbol{x}_i') = \sum_{i=1}^{n} (x_{ij} - x_{i'j})^2 = ||\boldsymbol{x}_i - \boldsymbol{x}_i'||^2 = (\boldsymbol{x}_i - \boldsymbol{x}_i')^T(\boldsymbol{x}_i - \boldsymbol{x}_i').$$

(squared Euclidean distance)

Dissimilarity and k-means

Note firstly that, for a collection of data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, one can describe the iterative part of the k-means algorithm entirely through two optimization steps rooted in dissimilarities [H Sec 14.3.6]:

(i) For a given set of cluster centres, say $\boldsymbol{c}_k$, $k = 1 \ldots, K$, assign each observation $\boldsymbol{x}_i$ to partition C_k, with k given by

$$k = \mathrm{argmin}_\ell D(\boldsymbol{x}_i, \boldsymbol{c}_\ell).$$

(ii) For a given partition $C_1, \ldots, C_K$, find the updated centre via

$$c_k' = \mathrm{Mean}(\boldsymbol{x}_i | \boldsymbol{x}_i \in C_k) = \mathrm{argmin}_{\boldsymbol{m}} \sum_{\boldsymbol{x}_i \in C_k} D(\boldsymbol{x}_i, \boldsymbol{m})$$

Dissimilarity and k-means

If D is the squared Euclidean distance, one can show that the k-means algorithm finds a minimum of the within-cluster sum of squares

$$\text{SS}_{\text{within}} = \sum_{k=1}^{K} \frac{1}{n_k} \sum_{(i,\ell):\, i<\ell,\, x_i, x_\ell \in C_k} D(\boldsymbol{x}_i, \boldsymbol{x}_\ell) = \sum_{k=1}^{K} \sum_{i:\boldsymbol{x}_i \in C_k} D(\boldsymbol{x}_i, \boldsymbol{c}_k)$$

where n_k is the number of observations in partition k.

This value can be compared to the total sum of squares,

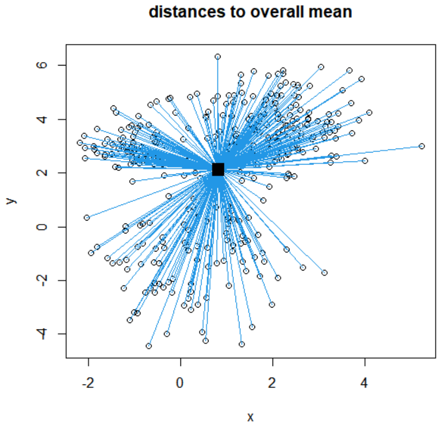
$$SS_{\text{total}} = \sum_{i=1}^{n} ||\boldsymbol{x}_i - \bar{\boldsymbol{x}}||^2 = \sum_{i=1}^{n} D(x_i, \bar{\boldsymbol{x}})$$

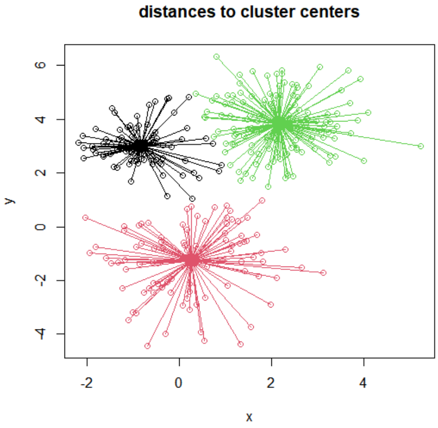
and their difference, $SS_{\text{total}} - \text{SS}_{\text{within}}$ corresponds to the between-cluster sum of squares.

Decomposing variation

```
k3$withinss

## [1] 270.39234  89.49661 258.54463

k3$tot.withinss      # SS_within

## [1] 618.4336

k3$totss             # SS_total

## [1] 2681.893

k3$betweenss         # SS_between

## [1] 2063.459
```

Decomposing variation





Sum of squares: 2681.893

Sum of squares: 618.4336

Analysis in R: Visualize projections

Total

```
m <- colMeans(intro.dat)
plot(intro.dat,
  main="distances to
        overall mean")
points(m[1], m[2], pch=15,
  cex=2, col=1)

n<-dim(intro.dat)[1]
for (j in 1:n){
  segments(intro.dat[j,1],
     intro.dat[j,2],
     m[1], m[2], col=4 )
}
```

Cluster-wise

```
plot(intro.dat, col=k3$cluster,
   main="distances to
         cluster centres")
points(k3$centers, pch=15,
  cex=2, col=c(1,2,3))

for (j in 1:n){
  segments(intro.dat[j,1],
    intro.dat[j,2],
    k3$centers[k3$cluster[j],1],
    k3$centers[k3$cluster[j],2],
    col=k3$cluster[j]  )
 }
```

k-Means, chickens, and eggs

Note that k-means is a chicken-and-egg problem....

- Determining the partitions requires knowing the means
- Finding the means requires knowing the partitions

- How to get started is a rather arbitrary decision!

k-Means: not a unique technique

There do exist many similar k-means algorithms, some of which differ in how this initialization step is handled, and some in other aspects:

The algorithm provided on slide 12 is Lloyd-Forgy's algorithm (Lloyd 1957, published 1982; independently by Forgy, 1965).

Alternative: Random partitioning: First, randomly assign a cluster label to each observation, and then find the mean of these clusters. (Excellent illustration in [J Sec 10.3.1]).

Default option in R: Hartigan-Wong algorithm. Initialization as in random partitioning, but then steps (i) and (ii) are iterated point-by-point rather than for the whole data set at once [H Sec 14.3.6.].

k-Means: not a unique technique

Whether any of these variants finds the global minimum of the total within-cluster sum of squares, will depend on the seed (Try!):

```
k31<- kmeans(intro.dat, centers=3, algorithm="Forgy")
k31$centers


##            x         y
## 1 -0.8169763  2.996031
## 2  0.2621725 -1.267684
## 3  2.1752499  3.826271


k31$tot.withinss


## [1] 618.4336
```

k-Means: not a unique technique

```
k3l$iter
```

```
## [1] 8
```

```
k3<- kmeans(intro.dat, centers=3, algorithm="Hartigan-Wong")
k3$centers
```

```
##            x         y
## 1  0.2621725 -1.267684
## 2 -0.8169763  2.996031
## 3  2.1752499  3.826271
```

```
k3$tot.withinss
```

k-Means: not a unique technique

```
## [1] 618.4336
```

```
k3$iter
```

```
## [1] 3
```

As with any algorithm that finds a local minimum, one can increase the chances of finding the global minimum for any variant by using multiple random starts; see option `nstart` in `kmeans`. [H Sec. 14.3.6]

k-Medoids

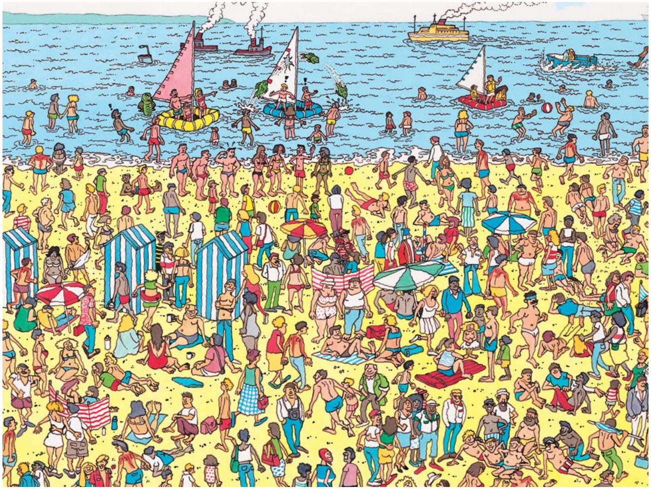
There are occasions when it is desirable that the cluster centres corresponds to actual observations ('the most central observation of the cluster').

Recall: In 'usual' k-means, in step (ii), finding the kth cluster mean is equivalent to minimizing $\sum_{\boldsymbol{x}_i \in C_k} D(\boldsymbol{x}_i, \boldsymbol{m})$ over all possible centres $\boldsymbol{m}$. k-Medoids replaces this by an optimization step which restricts the search space to the observations within the k-th cluster; i.e.

$$c_k' = \text{argmin}_{\boldsymbol{m} \in C_k} \sum_{\boldsymbol{x}_i \in C_k} D(\boldsymbol{x}_i, \boldsymbol{m}).$$

[H Sec 14.3.10, M Sec 14.4.2]

Example: Where's Wally?



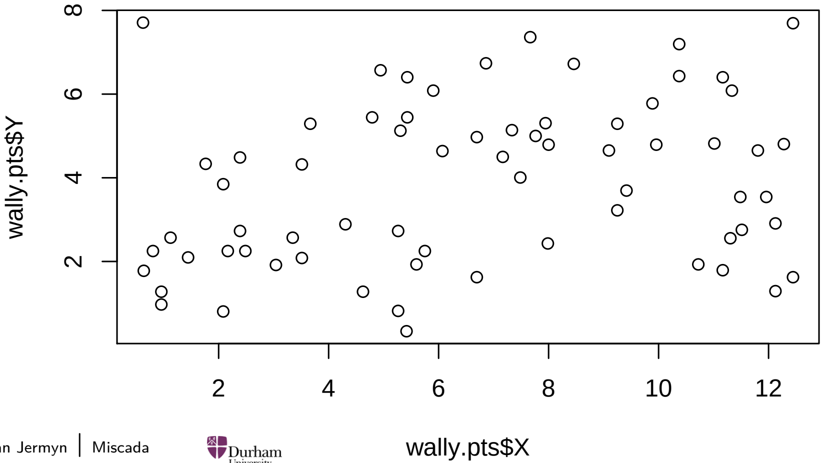
Example: Where's Wally?

Data gives the locations (X, Y coordinates) of where Wally (called Waldo in the US) was found in $n = 68$ double-paged illustrations in a total of seven 'Where's Wally' books.

```
wally.pts =
  read.csv('http://www.randalolson.com/wp-content/
            uploads/wheres-waldo-locations.csv')
```

```
plot(wally.pts$X, wally.pts$Y)
```

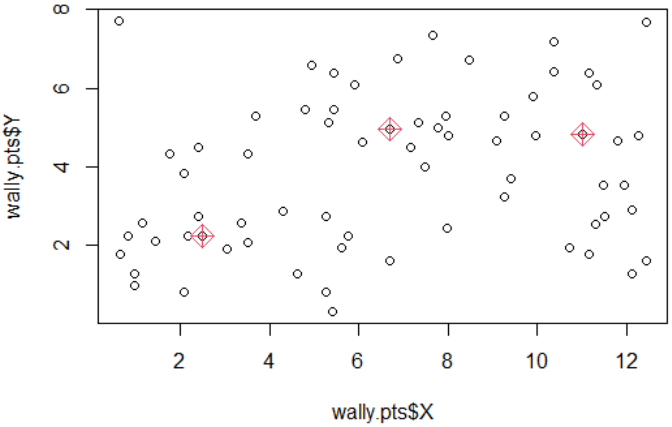
Example: Where's Wally?



Example: Where's Wally?

```r
library(clue)
wally.kmed<- kmedoids(dist(wally.pts[,c("X","Y")]), k=3)
plot(wally.pts$X, wally.pts$Y)
points(wally.pts$X[wally.kmed$medoid_ids],
   wally.pts$Y[wally.kmed$medoid_ids],
   col=2, pch=9, cex=1.8)
```

Example: Where's Wally?

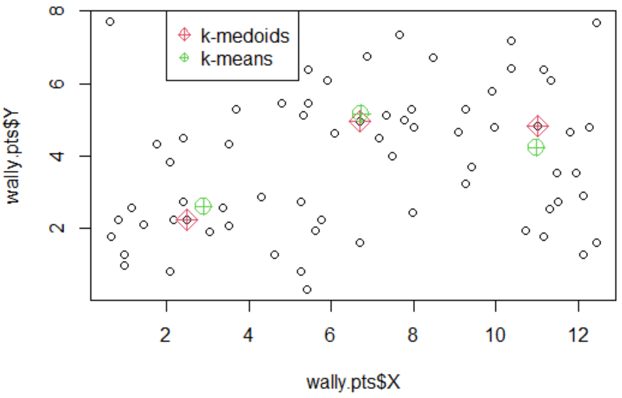


Where's Wally?

Compare to k-means:

```r
wally.kmean <- kmeans(wally.pts[,c("X","Y")],centers=3)
points(wally.kmean$centers, col=3, pch=10, cex=2)
legend(2,8, legend=c("k-medoids", "k-means"),
    pch=c(9,10), col=c(2,3)) #$
```

Where's Wally?



Data matrix

Let $\boldsymbol{X}$ denote a data matrix (or data frame) with n rows and p columns (variables, features); i.e.

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & \dots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{np} \end{pmatrix}$$

For instance, the data on slide 3 (right plot) constitute such a matrix with $n = 320$ and $p = 2$:

```
dim(intro.dat)


## [1] 320   2
```

Random vectors

We can combine the p components of each row of $\boldsymbol{X}$ into a vector $X \in \mathbb{R}^p$. The rows of $\boldsymbol{X}$ can then be thought of as realizations of a random vector, with the j-th component, X_j, being responsible for the generation of the j-th column of $\boldsymbol{X}$.

We have implicitly been using the properties of random vectors in $\mathbb{R}^2$ so far. We now make these properties more explicit and define some quantities for future use.

[M Sec 2.5]

Properties of random vectors

(a) There exists (under some mild conditions), a probability density function ('density') $f : \mathbb{R}^p \longrightarrow \mathbb{R}_0^+$, such that

$$P(X \in d\boldsymbol{x}) = d\boldsymbol{x}\, f(\boldsymbol{x}) \tag{1}$$

$$P(X \in S) = \int_S d\boldsymbol{x}\, f(\boldsymbol{x}) = \int_S f(\boldsymbol{x})\, d\boldsymbol{x} \tag{2}$$

for any (appropriately well-behaved) subset $S \subset \mathbb{R}^p$. In particular, for $S = \mathbb{R}^p$, $\int_{\mathbb{R}^p} f(\boldsymbol{x})\, d\boldsymbol{x} = 1$.

The density (f) can be estimated from data ($\boldsymbol{X}$); we deal with this problem in Part 2 of this course.

Properties of random vectors

(b) We refer to the $X_1, \ldots, X_p$ as independent if the joint density $f(\boldsymbol{x})$ can be factorized; i.e. if

$$f(x_1, \ldots, x_p) = f(x_1) \times \ldots \times f(x_p).$$

This means that the values of any random variable X_j are uninformative for the values of the other random variables.

Properties of random vectors

(c) The random vector X has a mean:

$$\begin{pmatrix} m_1 \\ \vdots \\ m_p \end{pmatrix} = \boldsymbol{m} = E(X) = \int \boldsymbol{x} f(\boldsymbol{x})\, d\boldsymbol{x} =$$

$$\int \cdots \int \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} f(x_1, \ldots, x_p) dx_1 \ldots dx_p$$

which implies $m_j = E(X_j)$, for the j-th component of $\boldsymbol{m}$.

Estimating the mean vector

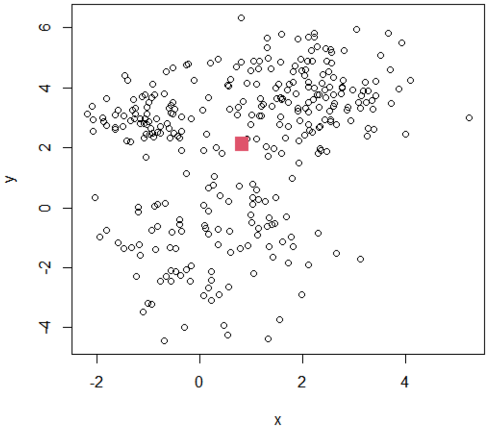
Let $\boldsymbol{x}_i^T$ denote the i-th row of $\boldsymbol{X}$, i.e., the i-th observation. Then we may estimate $\boldsymbol{m}$ by

$$\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i = \frac{1}{n}\sum_{i=1}^{n}\begin{pmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{pmatrix} = \begin{pmatrix} \frac{1}{n}\sum_{i=1}^{n} x_{i1} \\ \vdots \\ \frac{1}{n}\sum_{i=1}^{n} x_{ip} \end{pmatrix} = \begin{pmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_p \end{pmatrix}$$

One can easily show that this estimator is unbiased: if we were able to draw repeated samples of size n from X, and then compute $\bar{\boldsymbol{x}}$ each time, the average of these estimates would tend to the true value of $\boldsymbol{m}$ as the number of samples increased.

Estimating the mean vector: Example

```
m <- colMeans(intro.dat)
plot(intro.dat)
points(m[1], m[2], col=2,
  pch=15, cex=2)
```



Properties of random vectors (cont'd)

(d) The random vector also possesses a variance (also called variance matrix, covariance matrix),

$$\begin{aligned} \mathsf{Var}(X) &= E((X-\boldsymbol{m})(X-\boldsymbol{m})^T) = E(XX^T) - \boldsymbol{m}\boldsymbol{m}^T \\ &= \begin{pmatrix} \Sigma_{11} & \cdots & \Sigma_{1p} \\ \vdots & \ddots & \vdots \\ \Sigma_{p1} & \cdots & \Sigma_{pp} \end{pmatrix} = \boldsymbol{\Sigma}, \end{aligned}$$

where

$$\begin{aligned} \Sigma_{ij} &= \mathsf{Cov}(X_i, X_j) = E(X_iX_j) - E(X_i)E(X_j) \qquad i \neq j \\ \Sigma_{jj} &\equiv \sigma_j^2 = \mathsf{Var}(X_j) \end{aligned}$$

Properties of random vectors

Any variance matrix $\boldsymbol{\Sigma}$ has the following properties:

(i) $\boldsymbol{\Sigma}$ is symmetric, i.e. $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^T$

(ii) $\boldsymbol{\Sigma}$ is positive semi-definite; i.e. its eigenvalues are non-negative.

In short, we write

$$X \sim (\boldsymbol{m}, \boldsymbol{\Sigma})$$

meaning that X has some unspecified distribution with mean $\boldsymbol{m}$ and variance $\boldsymbol{\Sigma}$.

Estimating variance matrices

Firstly, recall

$$\boldsymbol{\Sigma} = \mathsf{Var}(X) = E\left((X - \boldsymbol{m})(X - \boldsymbol{m})^T\right).$$

Replacing all expectations by means, a natural candidate estimator for $\boldsymbol{\Sigma}$ is given by

$$\tilde{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\boldsymbol{x}_i^T - \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^T \in \mathbb{R}^{p\times p}.$$

In fact, this is the Maximum Likelihood estimator for $\boldsymbol{\Sigma}$ under an assumption of multivariate normality, but it is biased.

Estimating variance matrices

An unbiased estimator of $\boldsymbol{\Sigma}$ is obtained through the sample variance matrix,

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T = \frac{n}{n-1}\tilde{\boldsymbol{\Sigma}}$$

We generally prefer $\hat{\boldsymbol{\Sigma}}$ to $\tilde{\boldsymbol{\Sigma}}$. R functions `var` and `cov` (identical!) use this too.

In the special case of $p = 2$, $\hat{\boldsymbol{\Sigma}}$ can be written as

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n-1}\begin{pmatrix} \sum_{i=1}^{n}(x_{1i} - \bar{x}_1)^2 & \sum_{i=1}^{n}(x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2) \\ \sum_{i=1}^{n}(x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2) & \sum_{i=1}^{n}(x_{2i} - \bar{x}_2)^2 \end{pmatrix}$$

[M Sec 2.5.1]

Estimating variance matrices: Example

```
Sigma <- var(intro.dat)
Sigma
```

```
##          x        y
## x 2.439586 1.205168
## y 1.205168 5.967602
```

Correlation matrix

The correlation matrix is defined as

$$\boldsymbol{R} = (R_{ij})_{1 \leq i \leq p, 1 \leq j \leq p}$$

with pairwise correlation coefficients

$$R_{ij} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}\Sigma_{jj}}}$$

The matrix $\boldsymbol{R}$ is scale–invariant, and all diagonal elements are equal to 1. We will use this matrix mainly for PCA (Part 4).

We call the random variables X_i and X_j uncorrelated if $R_{ij} = 0$. Note:

- If all X_i, $i = 1, \ldots, p$ are uncorrelated then $\boldsymbol{R}$ becomes the identity matrix;
- If X_i and X_j are independent then they are uncorrelated (but the converse does not hold).

Estimating correlation matrices: Example

```
R <- cor(intro.dat)
R
```

```
##           x         y
## x 1.0000000 0.3158564
## y 0.3158564 1.0000000
```

The normal distribution

A random mechanism tends to have approximately a normal (aka Gaussian) distribution if its deviation from the average is the cumulative result of many independent influences.

Examples:

- Measurement errors;
- Minor variations in production of things (e.g. thickness of coins);
- Distribution of marks in a test (of an equally skilled cohort);
- Heights of people sampled from a population.

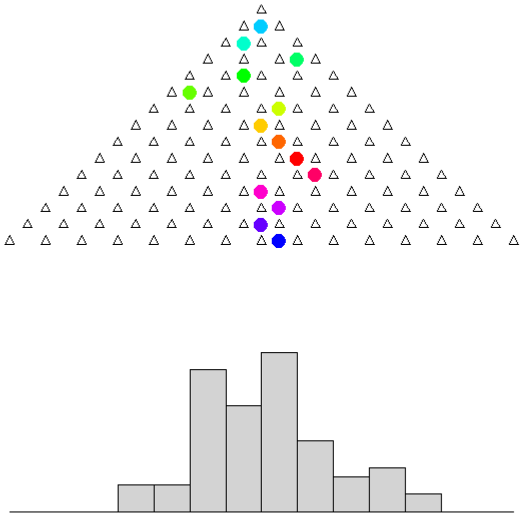
We will see this and other ways of justifying a Gaussian distribution in Foundations.

The normal distribution

A famous tool to mimic a normal distribution is the Galton Board:

```
require(animation)
set.seed(42)
balls <- 200
layers<- 15
ani.options(nmax = balls + layers - 2, 2)
galton.sim = quincunx(balls = balls,
    col.balls = rainbow(layers))
```

The normal distribution



Galton Board in progress.

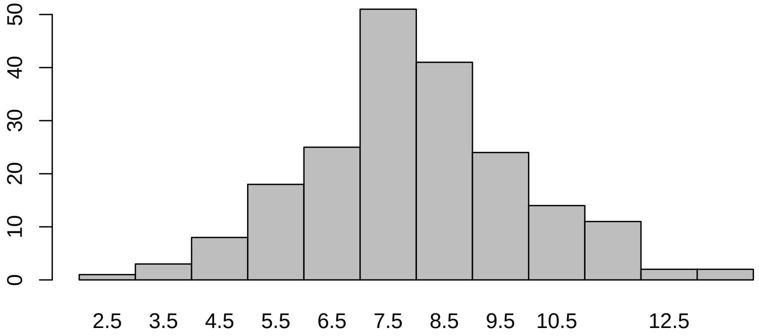
See full animation in R!

The normal distribution

Final outcome of animation:

```r
barplot(galton.sim, space = 0)
```

The normal distribution



The normal distribution

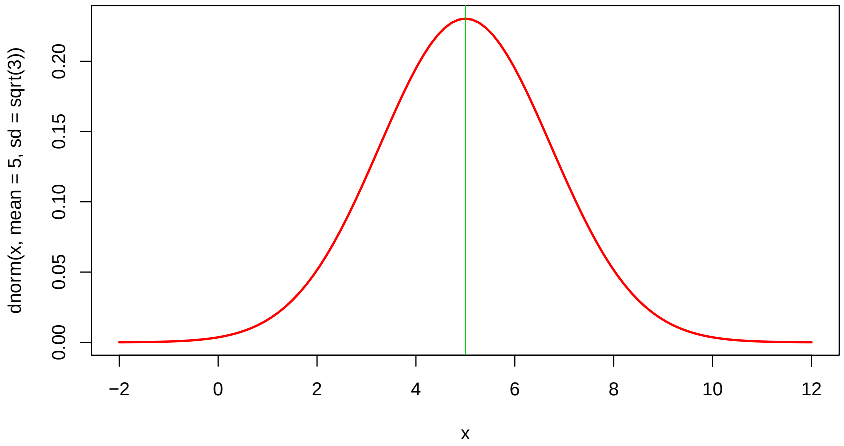
Probability density function of a (univariate) normal distribution with mean μ and variance σ^2, short $X \sim N(\mu, \sigma^2)$:

$$f(x \mid \mu, \sigma^2) = \phi(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

For instance, density function for $\mu = 5$ and $\sigma^2 = 3$:

```r
x<- seq(-2,12, length=101)
plot(x, dnorm(x,mean=5, sd=sqrt(3)), type="l", lwd=2, col=2)
abline(v=5, col=3)
```

The normal distribution



The normal distribution

“ [The normal distribution] cannot be obtained by rigorous deductions. Several of its putative proofs are awful [. . .]. Nonetheless, everyone believes it, as M. Lippmann told me one day, because experimenters imagine it to be a mathematical theorem, while mathematicians imagine it to be an experimental fact.”

— Henri Poincaré, Le calcul des Probabilités. 1896

Actually not true! Maximum entropy and the central limit theorem justify the Gaussian distribution.

Multivariate normal distribution

A random vector $X = (X_1, X_2, \ldots, X_p)^T$ is multivariate normal if (and only if) any linear combination of the random variables X_1, X_2, ..., X_p is univariate normally distributed, i.e. iff

$$a_1X_1 + a_2X_2 + \ldots + a_pX_p$$

has a univariate normal distribution for any constants a_1, a_2, ..., a_p.

- Setting $a_j = 1$, and all others $a_\ell = 0, \ell \neq j$, we see that multivariate normality of X implies univariate normality of each X_j (the reverse is not true)!
- The definition includes the limiting case that $(a_1, \ldots, a_p) \equiv \mathbf{0}$, in which case X is a p-variate point mass at 0 (a ‘Dirac delta function’, and strictly speaking no longer a density).

Multivariate normal distribution

When a positive-definite variance matrix $\boldsymbol{\Sigma}$ exists, then the density of a multivariate normal (MVN) distribution takes the form

$$\phi(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right\}, \qquad (3)$$

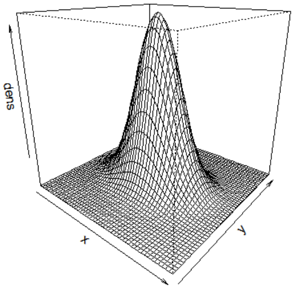
with parameters $\boldsymbol{\mu} \in \mathbb{R}^p$, $\boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$. We write then

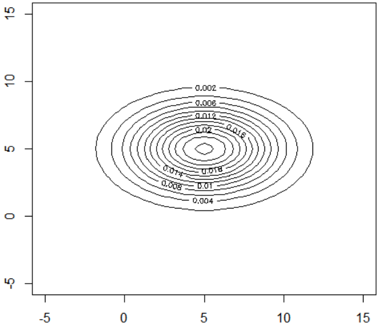
$$X \sim N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

[M Sec 2.5.2, M Sec 4.1]

Multivariate normal distribution

Example for MVN density: $\boldsymbol{\mu} = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$, $\boldsymbol{\Sigma} = \begin{pmatrix} 3^2 & 0 \\ 0 & 2^2 \end{pmatrix}$





Durham University

Multivariate normal distribution

R Code for displaying density

```
# creates an an appropriate grid
x1    <- seq(-5,15, length=51)   # 51 is an arbitrary grid size
x2    <- seq(-5,15, length=51)
dens <- matrix(0,51,51)

# defines mu and Sigma
mu <- c(5,5)
Sigma <- matrix(c(9,0,0,4), byrow=TRUE, ncol=2)

# fills grid with density values
require(mvtnorm)
for (i in 1:51){
  for (j in 1:51){
    dens[i,j] <- dmvnorm(x=c(x1[i],x2[j]), mean=mu, sigma=Sigma)
  }
}
persp(x1, x2, dens, theta=40, phi=20)  # draws the density in 3D
contour(x1, x2, dens)                  # draws contour plots in 2D
```

Multivariate normal distribution

Some observations (based on the example from the last two slides):

- From the marginalization property, we can conclude that $X_1 \sim N(5, 3^2)$ and $X_2 \sim N(5, 2^2)$;
- For the matrix $\mathbf{\Sigma}$ used in the last two slides, all entries off the diagonal are zero. Statistically, the entries off the diagonal are the covariances between the two random variables X_1 and X_2. If these are 0, then X_1 and X_2 are uncorrelated. If X is MVN with uncorrelated components X_1 and X_2, then these are also independent, i.e. $f(x_1, x_2) = f(x_1) \times f(x_2)$.

Multivariate normal distribution

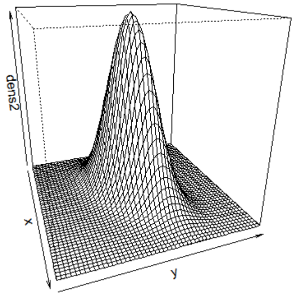
What happens if the two components are not uncorrelated? Assume now $\mathrm{Cov}(X_1, X_2) = 3$, that is $\boldsymbol{\Sigma} = \begin{pmatrix} 3^2 & 3 \\ 3 & 2^2 \end{pmatrix}$. The only actual difference to the previous code is

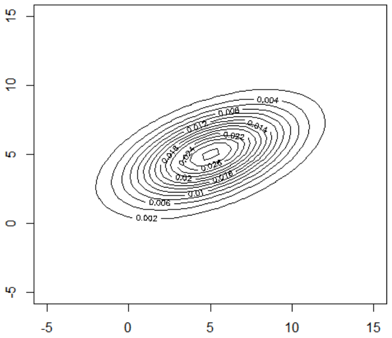
```
Sigma <- matrix(c(9,3,3,4), byrow=TRUE, ncol=2)
```

but for aesthetic reasons we slightly change the graphical parameters:

```
persp(x1, x2, dens, theta=40, phi=20)
contour(x1, x2, dens, nlevels=20 )
```

Multivariate normal distribution





Quick reminder: Variance matrix needs to be valid

Can we choose ‘anything’ for $\mathbf{\Sigma}$?

No! It needs to be symmetric and positive definite or the density cannot be normalized. Check the latter via

```
eigen(Sigma)$values

## [1] 10.405125  2.594875

#$
```

$\mathbf{\Sigma}$ is positive definite if and only if all eigenvalues are positive.

Weighted Dissimilarities

Sometimes attributes operate on very different scales (for instance, kg, meters, etc.), which could distort the dissimilarities. This can be mitigated by defining appropriate weights. Common choices are

(i) $w_j = 1/\bar{d}_j$, where $\bar{d}_j$ is the mean over all (n^2) pairwise dissimilarities over all n observations (gives all attributes equal influence) [H1 Sec 14.3.3].

(ii) $w_j = 1/\sigma_j^2$, where σ_j is the (known or estimated) standard deviation of the jth attribute (scales all attributes to unit variance).

Mahalanobis distances

The dissimilarities created in (ii) can be written as

$$D(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \sum_{j=1}^{p} \frac{1}{\sigma_j^2}(x_{ij} - x_{i'j})^2 = (\boldsymbol{x}_i - \boldsymbol{x}_{i'})^T \begin{pmatrix} \sigma_1^{-2} & & \\ & \ddots & \\ & & \sigma_p^{-2} \end{pmatrix} (\boldsymbol{x}_i - \boldsymbol{x}_{i'})$$

This can be generalized by replacing the diagonal matrix by the inverse of a full variance matrix $\boldsymbol{\Sigma}$ (again, known or estimated), yielding what is known as the (squared) Mahalanobis distance:

$$D_M(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = (\boldsymbol{x}_i - \boldsymbol{x}_{i'})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_{i'})$$

[M Sec 4.1.2]

Mahalanobis distances

Have we seen this dissimilarity before? Yes, in MVN density!

Note that we can write the $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ density function as

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}D(\boldsymbol{x}, \boldsymbol{\mu})\right\}, \tag{4}$$

therefore points with equal Mahalanobis distance to the mean lie on the same contour of the respective Gaussian distribution.

Mahalanobis distances

An interesting property of Mahalanobis distances to the mean is that, when considered as a random variable,

$$D_M(X, \boldsymbol{\mu}) \approx \chi^2(p)$$

where $\chi^2(p)$ denotes a χ^2 distribution with p degrees of freedom.

This result can be used for outlier detection and testing multivariate normality.

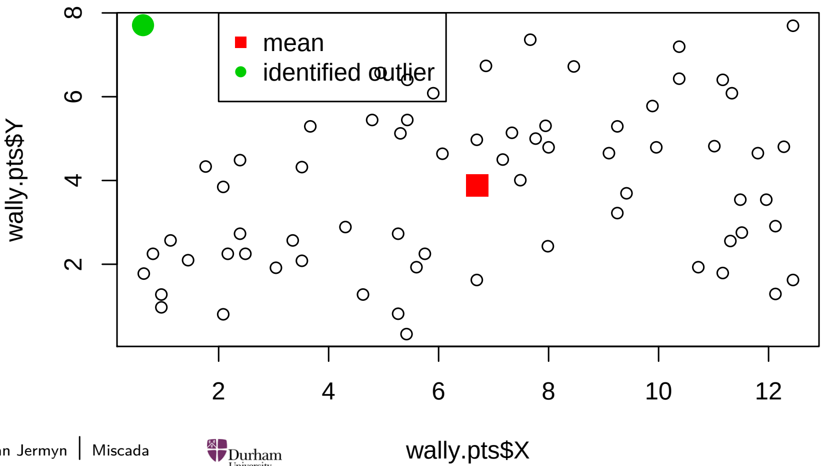
Outlier detection

For instance, for Wally's data, compute all squared Mahalanobis distances and compare with the 2.5% tail quantile of $\chi^2(2)$:

```
wally.m <- colMeans(wally.pts[,c("X","Y")])
wally.S <- var(wally.pts[,c("X","Y")])
wally.Mdist <- mahalanobis(wally.pts[,c("X","Y")],
    center=wally.m, cov=wally.S)
detect <- which(wally.Mdist>qchisq(0.975,2) )
detect


## [1] 1
```

```
plot(wally.pts$X, wally.pts$Y)
points(wally.m[1], wally.m[2], pch=15, cex=2,col=2 )
points(wally.pts$X[detect], wally.pts$Y[detect], pch=16, col=3, cex=2)
legend(2,8,pch=c(15,16), col=c(2,3),  legend=c("mean", "identified outlier"))
```

Outlier detection

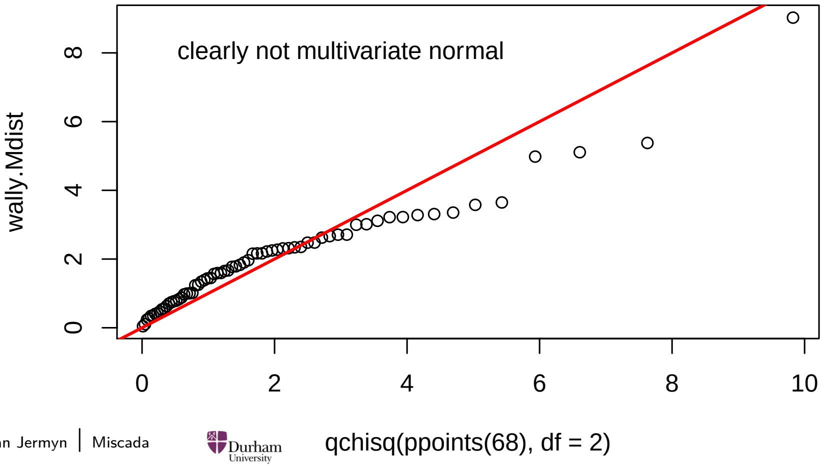


Checking multivariate normality

Since, under MVN, Mahalanobis distances to the mean follow a $\chi^2(p)$ distribution, we can use them to check a dataset for multivariate normality:

```r
qqplot(qchisq(ppoints(68), df=2),wally.Mdist)
abline(a=0,b=1, col=2, lwd=2)
text(3,8,"clearly not multivariate normal")
```

Checking multivariate normality

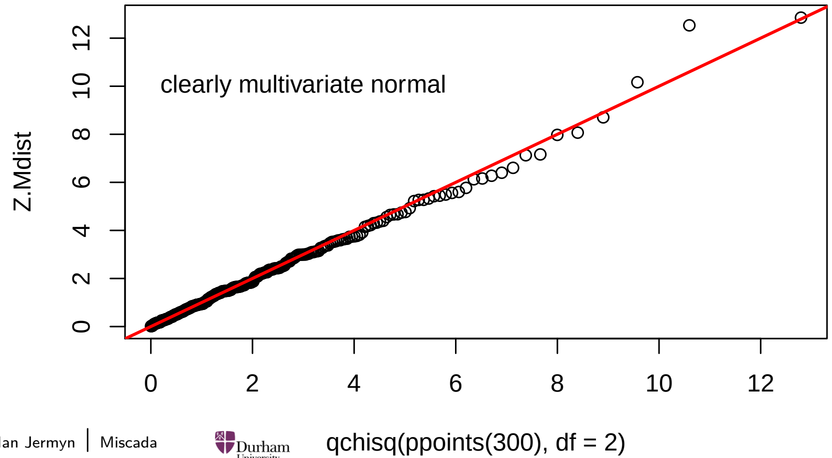


Checking multivariate normality

Compare to data generated from a MVN:

```
require(mvtnorm)
mu    <- c(5,5)
Sigma <- matrix(c(9,0,0,4), byrow=TRUE, ncol=2)
Z <- rmvnorm(300, mean=mu, sigma=Sigma)
Z.Mdist <- mahalanobis(Z, mu, Sigma)
qqplot(qchisq(ppoints(300), df=2),Z.Mdist)
abline(a=0,b=1, col=2, lwd=2)
text(3,10,"clearly multivariate normal")
```

Checking multivariate normality



Practical 1

In the first lab session (fetch **Practical 1** on Jupyter), we will:

- carry out some simple exploratory data analysis of a four-dimensional, oceanographic data set;
- apply k-means to this data set; identify the cluster centres and visualize the clusters;
- experiment with different settings of `kmeans` and different numbers of clusters;
- search for outliers and check for multivariate normality of this data set.