MS1b
Statistical Machine Learning and Data Mining

Yee Whye Teh
Department of Statistics
Oxford

http://www.stats.ox.ac.uk/~teh/smldm.html

Course Information

- Course webpage:
  http://www.stats.ox.ac.uk/~teh/smldm.html
- Lecturer: Yee Whye Teh
- TA for Part C: Thibaut Lienant
- TA for MSc: Balaji Lakshminarayanan and Maria Lomeli
- Please subscribe to Google Group:
  https://groups.google.com/forum/?hl=en-GB#!forum/smldm
- Sign up for course using sign up sheets.

Course Structure

Lectures

- 1400-1500 Mondays in Math Institute L4.
- 1000-1100 Wednesdays in Math Institute L3.

Part C:

- 6 problem sheets.
- Classes: 1600-1700 Tuesdays (Weeks 3-8) in 1 SPR Seminar Room.
- Due Fridays week before classes at noon in 1 SPR.

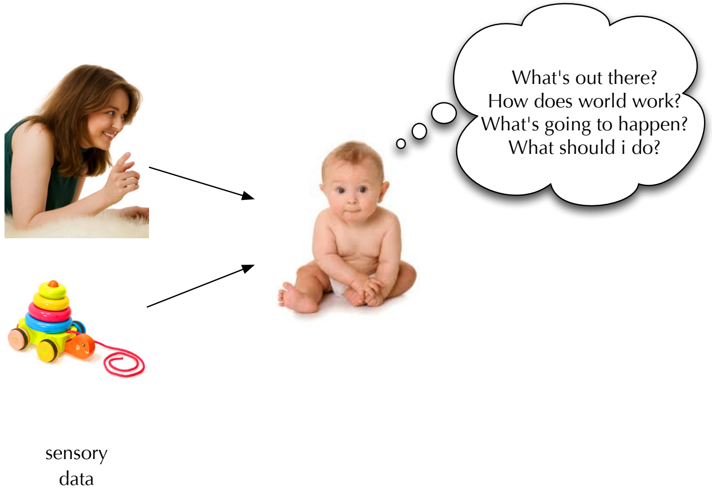
MSc:

- 4 problem sheets.
- Classes: Tuesdays (Weeks 3, 5, 7, 9) in 2 SPR Seminar Room.
- Group A: 1400-1500, Group B: 1500-1600.
- Due Fridays week before classes at noon in 1 SPR.
- Practical: Week 5 and 7 (assessed) in 1 SPR Computing Lab.
- Group A: 1400-1600, Group B: 1600-1800.

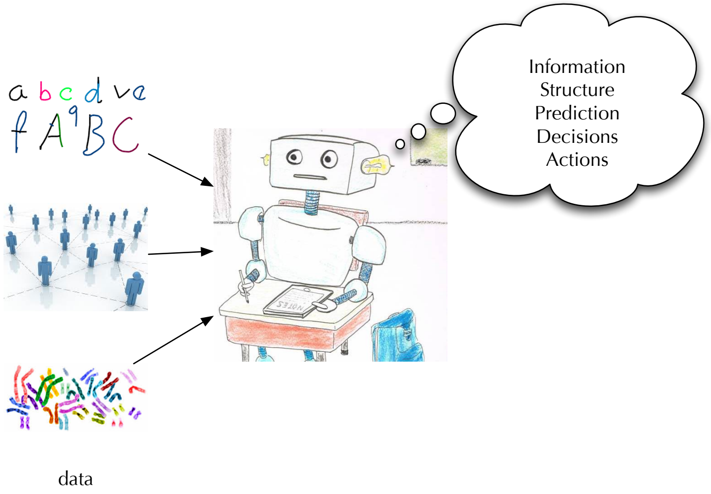
Course Aims

1. Have ability to use the relevant R packages to analyse data, interpret results, and evaluate methods.
2. Have ability to identify and use appropriate methods and models for given data and task.
3. Understand the statistical theory framing machine learning and data mining.
4. Able to construct appropriate models and derive learning algorithms for given data and task.

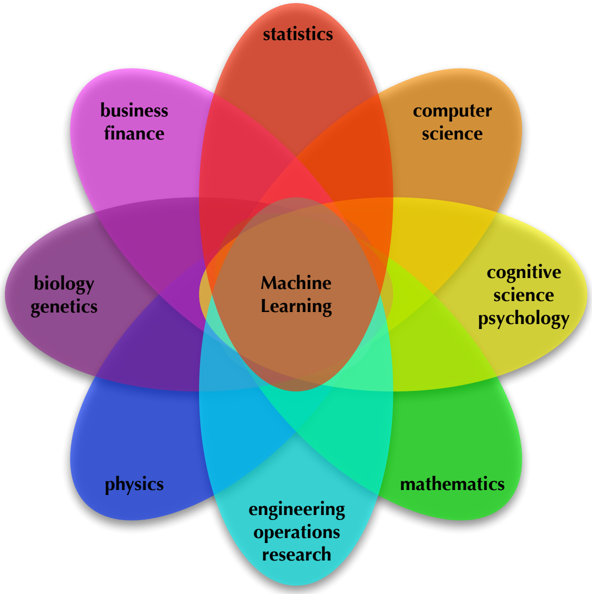
What is Machine Learning?



What is Machine Learning?



What is Machine Learning?



What is the Difference?

Traditional Problems in Applied Statistics

Well formulated question that we would like to answer.
Expensive to gathering data and/or expensive to do computation.
Create specially designed experiments to collect high quality data.

Current Situation

Information Revolution

- Improvements in computers and data storage devices.
- Powerful data capturing devices.
- Lots of data with potentially valuable information available.

What is the Difference?

Data characteristics

- Size
- Dimensionality
- Complexity
- Messy
- Secondary sources

Focus on generalization performance

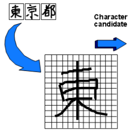
- Prediction on new data
- Action in new circumstances
- Complex models needed for good generalization.

Computational considerations

- Large scale and complex systems

Applications of Machine Learning

- Pattern Recognition



  - Sorting Cheques
  - Reading License Plates
  - Sorting Envelopes
  - Eye/ Face/ Fingerprint Recognition

Applications of Machine Learning

- Business applications
  - Help companies intelligently find information
  - Credit scoring
  - Predict which products people are going to buy
  - Recommender systems
  - Autonomous trading
- Scientific applications
  - Predict cancer occurence/type and health of patients/personalized health
  - Make sense of complex physical, biological, ecological, sociological models

---

Further Readings, News and Applications

Links are clickable in pdf. More recent news posted on course webpage.

- Leo Breiman: Statistical Modeling: The Two Cultures
- NY Times: R
- NY Times: Career in Statistics
- NY Times: Data Mining in Walmart
- NY Times: Big Data’s Impact In the World
- Economist: Data, Data Everywhere
- McKinsey: Big data: The Next Frontier for Competition
- NY Times: Scientists See Promise in Deep-Learning Programs
- New Yorker: Is “Deep Learning” a Revolution in Artificial Intelligence?

Types of Machine Learning

Unsupervised Learning

Uncover structure hidden in ‘unlabelled’ data.

- Given network of social interactions, find communities.
- Given shopping habits for people using loyalty cards: find groups of ‘similar’ shoppers.
- Given expression measurements of 1000s of genes for 1000s of patients, find groups of functionally similar genes.

Goal: Hypothesis generation, visualization.

Types of Machine Learning

Supervised Learning

A database of examples along with “labels” (task-specific).

- Given network of social interactions **along with their browsing habits**, predict what news might users find interesting.
- Given expression measurements of 1000s of genes for 1000s of patients **along with an indicator of absence or presence of a specific cancer**, predict if the cancer is present for a new patient.
- Given expression measurements of 1000s of genes for 1000s of patients **along with survival length**, predict survival time.

Goal: Prediction **on new examples**.

Types of Machine Learning

Semi-supervised Learning

A database of examples, only a small subset of which are labelled.

Multi-task Learning

A database of examples, each of which has multiple labels corresponding to different prediction tasks.

Reinforcement Learning

An agent acting in an environment, given rewards for performing appropriate actions, learns to maximize its reward.

OxWaSP

Oxford-Warwick Centre for Doctoral Training in Statistics

- Programme aims to produce EuropeÕs future research leaders in statistical methodology and computational statistics for modern applications.
- 10 fully-funded (UK, EU) students a year (1 international).
- Website for prospective students.
- **Deadline: January 24, 2014**

Exploratory Data Analysis

Notation

- Data consists of p measurements (variables/attributes) on n examples (observations/cases)
- $\mathbf{X}$ is a $n \times p$-matrix with $\mathbf{X}_{ij} :=$ the j-th measurement for the i-th example

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1j} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2j} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \dots & x_{ij} & \dots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nj} & \dots & x_{np} \end{bmatrix}$$

- Denote the ith data item by $x_i \in \mathbb{R}^p$. (This is transpose of ith row of $\mathbf{X}$)
- Assume $x_1, \dots, x_n$ are **independently and identically distributed** samples of a **random vector** X over $\mathbb{R}^p$.

Crabs Data ($n = 200$, $p = 5$)

Campbell (1974) studied rock crabs of the genus **leptograpsus**. One species, **L. variegatus**, had been split into two new species, previously grouped by colour, orange and blue. Preserved specimens lose their colour, so it was hoped that morphological differences would enable museum material to be classified.

Data are available on 50 specimens of each sex of each species, collected on sight at Fremantle, Western Australia. Each specimen has measurements on:

- the width of the frontal lobe `FL`,
- the rear width `RW`,
- the length along the carapace midline `CL`,
- the maximum width `CW` of the carapace, and
- the body depth `BD` in mm.

in addition to colour (species) and sex.

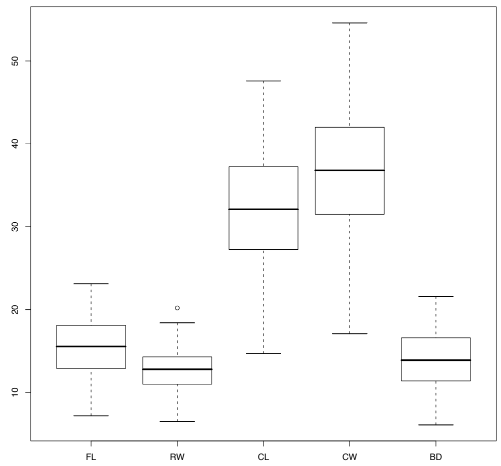
Crabs Data I

```
## load package MASS containing the data
library(MASS)
## look at data
crabs

## assign predictor and class variables
Crabs <- crabs[,4:8]
Crabs.class <- factor(paste(crabs[,1],crabs[,2],sep=""))

## various plots
boxplot(Crabs)
hist(Crabs$FL,col=’red’,breaks=20,xname=’Frontal Lobe Size (mm)’)
hist(Crabs$RW,col=’red’,breaks=20,xname=’Rear Width (mm)’)
hist(Crabs$CL,col=’red’,breaks=20,xname=’Carapace Length (mm)’)
hist(Crabs$CW,col=’red’,breaks=20,xname=’Carapace Width (mm)’)
hist(Crabs$BD,col=’red’,breaks=20,xname=’Body Depth (mm)’)
plot(Crabs,col=unclass(Crabs.class))
parcoord(Crabs)
```

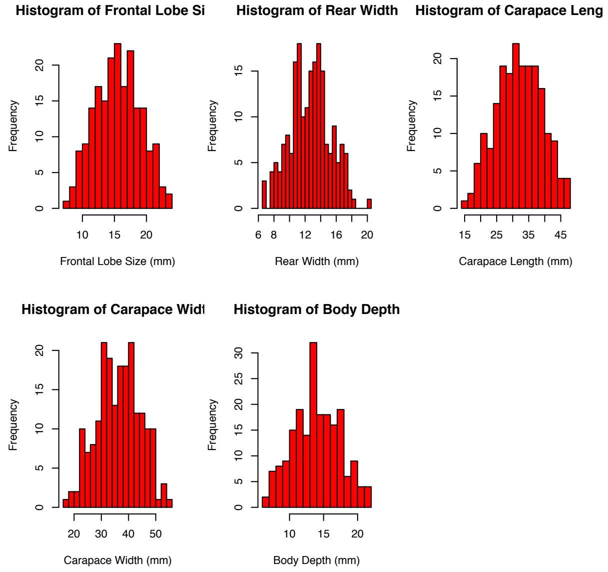
Crabs data

```
    sp sex index   FL   RW   CL   CW   BD
1    B   M     1  8.1  6.7 16.1 19.0  7.0
2    B   M     2  8.8  7.7 18.1 20.8  7.4
3    B   M     3  9.2  7.8 19.0 22.4  7.7
4    B   M     4  9.6  7.9 20.1 23.1  8.2
5    B   M     5  9.8  8.0 20.3 23.0  8.2
6    B   M     6 10.8  9.0 23.0 26.5  9.8
7    B   M     7 11.1  9.9 23.8 27.1  9.8
8    B   M     8 11.6  9.1 24.5 28.4 10.4
9    B   M     9 11.8  9.6 24.2 27.8  9.7
10   B   M    10 11.8 10.5 25.2 29.3 10.3
11   B   M    11 12.2 10.8 27.3 31.6 10.9
12   B   M    12 12.3 11.0 26.8 31.5 11.4
13   B   M    13 12.6 10.0 27.7 31.7 11.4
14   B   M    14 12.8 10.2 27.2 31.8 10.9
15   B   M    15 12.8 10.9 27.4 31.5 11.0
16   B   M    16 12.9 11.0 26.8 30.9 11.4
17   B   M    17 13.1 10.6 28.2 32.3 11.0
18   B   M    18 13.1 10.9 28.3 32.4 11.2
19   B   M    19 13.3 11.1 27.8 32.3 11.3
20   B   M    20 13.9 11.1 29.2 33.3 12.1
```

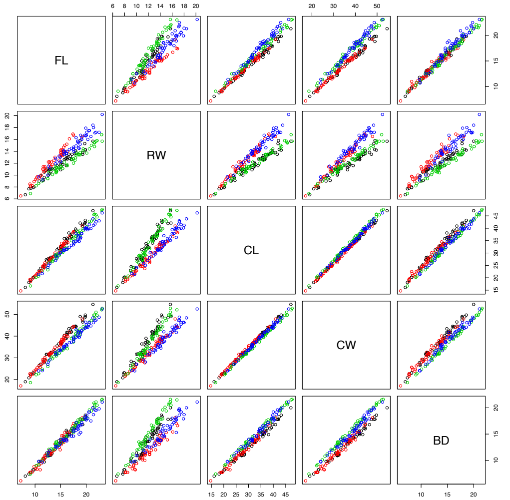
Univariate Boxplots



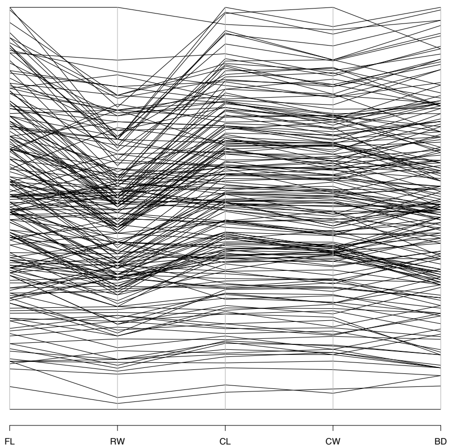
Univariate Histograms



Simple Pairwise Scatterplots



Parallel Coordinate Plots



Visualization and Dimensionality Reduction

These summary plots are helpful, but do not really help very much if the dimensionality of the data is high (a few dozen or thousands).
Visualizing higher-dimensional problems:

- We are constrained to view data in 2 or 3 dimensions
- Look for 'interesting' projections of $\mathbf{X}$ into lower dimensions
- Hope that for large p, considering only $k \ll p$ dimensions is just as informative.

Dimensionality reduction

- For each data item $x_i \in \mathbb{R}^p$, find a lower dimensional representation $z_i \in \mathbb{R}^k$ with $k \ll p$.
- Preserve as much as possible the interesting statistical properties/relationships of data items.

Principal Components Analysis (PCA)

- PCA considers interesting directions to be those with greatest **variance**.
- A **linear** dimensionality reduction technique:
- Finds an orthogonal basis $v_1, v_2, \dots, v_p$ for the data space such that
  - The first principal component (PC) v_1 is the direction of greatest variance of data.
  - The second PC v_2 is the direction orthogonal to v_1 of greatest variance, etc.
  - The subspace spanned by the first k PCs represents the 'best' k-dimensional representation of the data.
  - The k-dimensional representation of x_i is:

$$z_i = V^\top x_i = \sum_{\ell=1}^{k} v_\ell^\top x_i$$

    where $V \in \mathbb{R}^{p \times k}$.
- For simplicity, we will assume from now on that our dataset is centred, i.e. we subtract the average $\bar{x}$ from each x_i.

Principal Components Analysis (PCA)

- Our data set is an iid sample of a random vector $X = [X_1 \dots X_p]^\top$.
- For the 1^{st} PC, we seek a derived variable of the form

$$Z_1 = v_{11}X_1 + v_{12}X_2 + \cdots + v_{1p}X_p = v_1^\top X$$

where $v_1 = [v_{11}, \dots, v_{1p}]^\top \in \mathbb{R}^p$ are chosen to maximise

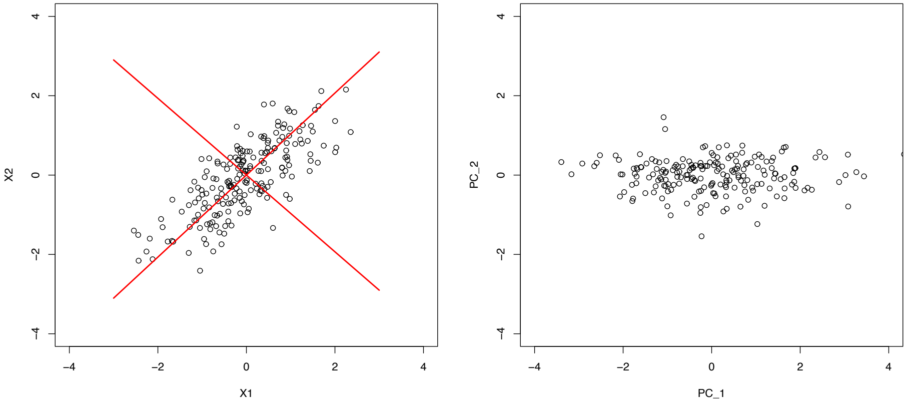
$$\mathrm{Var}(Z_1).$$

To get a well defined problem, we fix

$$v_1^\top v_1 = 1.$$

- The 2^{nd} PC is chosen to be orthogonal with the 1^{st} and is computed in a similar way. It will have the largest variance in the remaining $p-1$ dimensions, etc.

Principal Components Analysis (PCA)



Deriving the First Principal Component

- Maximise, subject to $v_1^\top v_1 = 1$:

$$\mathrm{Var}(Z_1) = \mathrm{Var}(v_1^\top X) = v_1^\top \mathrm{Cov}(X) v_1 \approx v_1^\top S v_1$$

  where $S \in \mathbb{R}^{p \times p}$ is the sample covariance matrix, i.e.

$$S = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^\top = \frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^\top = \frac{1}{n-1}\mathbf{X}^\top \mathbf{X}.$$

- Rewriting this as a constrained maximisation problem,

$$\mathcal{L}(v_1, \lambda_1) = v_1^\top S v_1 - \lambda_1 \left(v_1^\top v_1 - 1 \right).$$

- The corresponding vector of partial derivatives yields

$$\frac{\partial \mathcal{L}(v_1, \lambda_1)}{\partial v_1} = 2Sv_1 - 2\lambda_1 v_1.$$

- Setting this to zero reveals the eigenvector equation, i.e. v_1 must be an eigenvector of S and λ_1 the corresponding eigenvalue.
- Since $v_1^\top S v_1 = \lambda_1 v_1^\top v_1 = \lambda_1$, the 1^{st} PC must be the eigenvector associated with the largest eigenvalue of S.

Deriving Subsequent Principal Components

- Proceed as before but include the additional constraint that the 2^{nd} PC must be orthogonal to the 1^{st} PC:

$$\mathcal{L}(v_2, \lambda_2, \mu) = v_2^\top S v_2 - \lambda_2 \left(v_2^\top v_2 - 1 \right) - \mu \left(v_1^\top v_2 \right).$$

- Solving this shows that v_2 must be the eigenvector of S associated with the 2^{nd} largest eigenvalue, and so on
- The eigenvalue decomposition of S is given by

$$S = V \Lambda V^\top$$

  where Λ is a diagonal matrix with eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$$

  and V is a $p \times p$ orthogonal matrix whose columns are the p eigenvectors of S, i.e. the principal components $v_1, \ldots, v_p$.

Properties of the Principal Components

- PCs are **uncorrelated**

$$\mathrm{Cov}(X^\top v_i, X^\top v_j) \approx v_i^\top S v_j = 0 \text{ for } i \neq j.$$

- The **total sample variance** is given by

$$\sum_{i=1}^{p} S_{ii} = \lambda_1 + \ldots + \lambda_p,$$

  so the **proportion of total variance** explained by the k^{th} PC is

$$\frac{\lambda_k}{\lambda_1 + \lambda_2 + \ldots + \lambda_p} \qquad k = 1, 2, \ldots, p$$

- S is a real symmetric matrix, so eigenvectors (principal components) are orthogonal.
- Derived variables $Z_1, \ldots, Z_p$ have variances $\lambda_1, \ldots, \lambda_p$.

`R` code

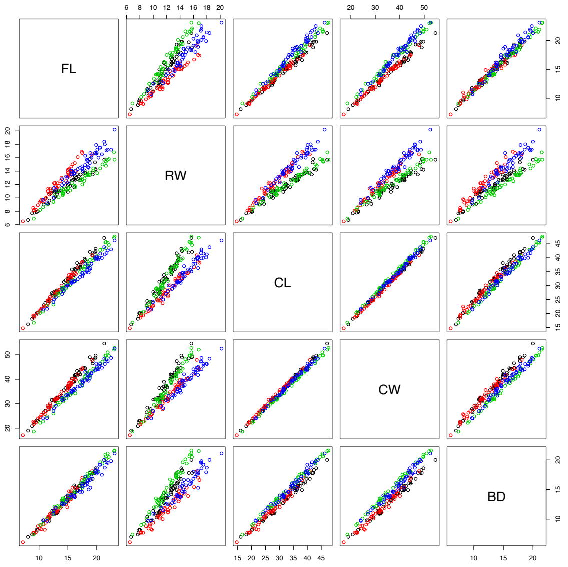
This is what we have had before:

```
library(MASS)
Crabs <- crabs[,4:8]
Crabs.class <- factor(paste(crabs[,1],crabs[,2],sep=""))
plot(Crabs,col=unclass(Crabs.class))
```

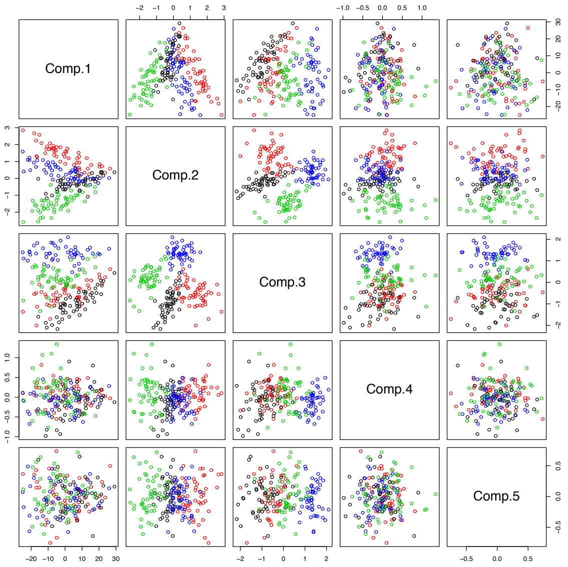
Now perform PCA with function `princomp`. (Alternatively, solve for the PCs yourself using `eigen` or `svd`).

```
Crabs.pca <- princomp(Crabs,cor=FALSE)
plot(Crabs.pca)
pairs(predict(Crabs.pca),col=unclass(Crabs.class))
```

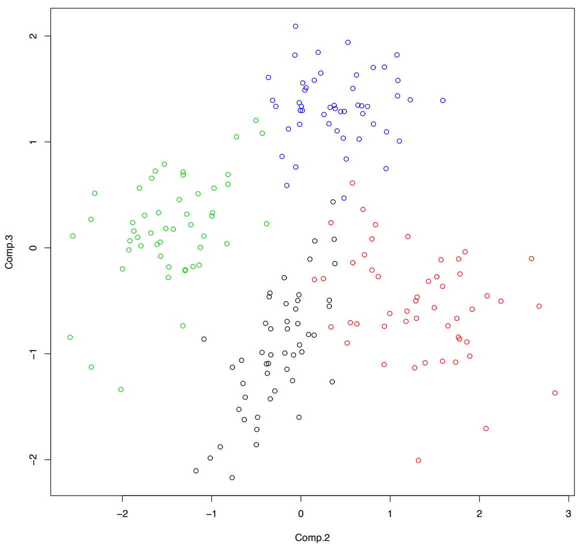
Original Crabs Data



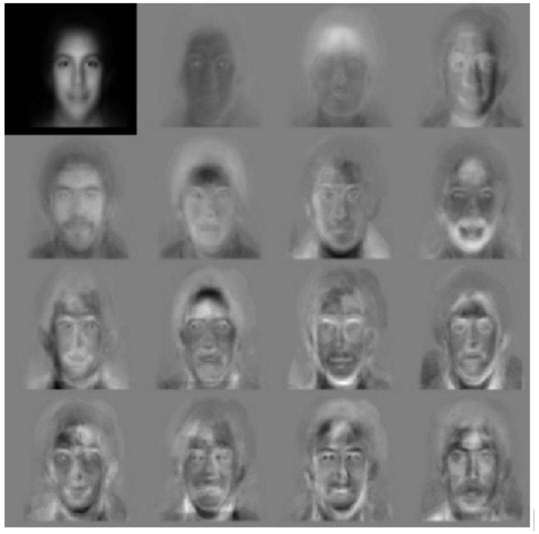
PCA of Crabs Data



PC 2 vs PC 3

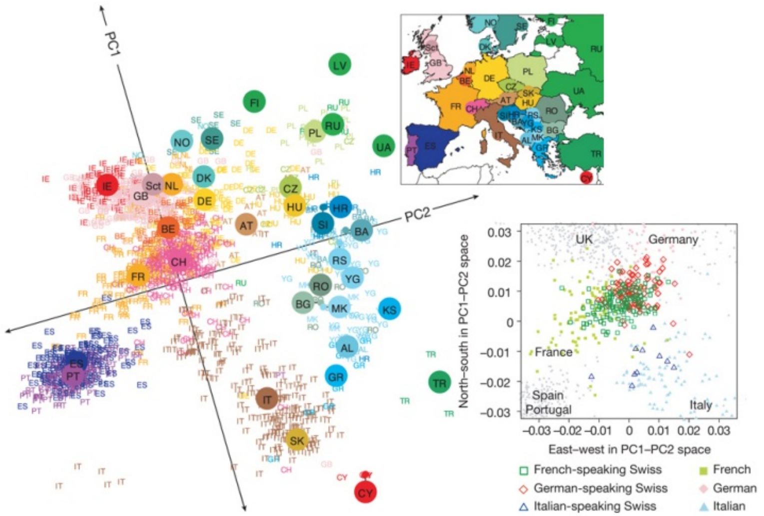


PCA on Face Images



http://vismod.media.mit.edu/vismod/demos/facerec/basic.html

PCA on European Genetic Variation



http://www.nature.com/nature/journal/v456/n7218/full/nature07331.html

---

Comments on the use of PCA

- PCA commonly used to project data X onto the first k PCs giving the k-dimensional view of the data that best preserves the first two moments.
- Although PCs are uncorrelated, scatterplots sometimes reveal structures in the data other than linear correlation.
- PCA commonly used for lossy compression of high dimensional data.
- Emphasis on variance is where the weaknesses of PCA stem from:
  - The PCs depend heavily on the units measurement. Where the data matrix contains measurements of vastly differing orders of magnitude, the PC will be greatly biased in the direction of larger measurement. It is therefore recommended to calculate PCs from $\mathrm{Corr}(X)$ instead of $\mathrm{Cov}(X)$.
  - Robustness to outliers is also an issue. Variance is affected by outliers therefore so are PCs.

Eigenvalue Decomposition (EVD)

Eigenvalue decomposition plays a significant role in PCA. PCs are eigenvectors of $S = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X}$ and PCA properties are derived from those of eigenvectors and eigenvalues.

- For any $p \times p$ **symmetric** matrix S, there exists p eigenvectors $v_1, \dots, v_p$ that are pairwise orthogonal and p associated eigenvalues $\lambda_1, \dots, \lambda_p$ which satisfy the eigenvalue equation $Sv_i = \lambda_i v_i \ \forall i$.
- S can be written as $S = V\Lambda V^\top$ where
  - $V = [v_1, \dots, v_p]$ is a $p \times p$ orthogonal matrix
  - $\Lambda = diag\{\lambda_1, \dots, \lambda_p\}$
  - If S is a real-valued matrix, then the eigenvalues are real-valued as well, $\lambda_i \in \mathbb{R} \ \forall i$
- To compute the PCA of a dataset $\mathbf{X}$, we can:
  - First estimate the covariance matrix using the sample covariance S.
  - Compute the EVD of S using the R command `eigen`.

Singular Value Decomposition (SVD)

Though the EVD does not always exist, the singular value decomposition is another matrix factorization technique that **always** exist, even for non-square matrices.

- $\mathbf{X}$ can be written as $X = UDV^\top$ where
  - U is an $n \times n$ matrix with orthogonal columns.
  - D is a $n \times p$ matrix with decreasing non-negative elements on the diagonal (the singular values) and zero off-diagonal elements.
  - V is a $p \times p$ matrix with orthogonal columns.
- SVD can be computed using very fast and numerically stable algorithms. The relevant R command is `svd`.

Some Properties of the SVD

- Let $\mathbf{X} = UDV^\top$ be the SVD of the $n \times p$ data matrix $\mathbf{X}$.
- Note that

$$(n-1)S = \mathbf{X}^\top\mathbf{X} = (UDV^\top)^\top(UDV^\top) = VD^\top U^\top UDV^\top = VD^\top DV^\top,$$

  using orthogonality ($U^\top U = I_n$) of U.
- The eigenvalues of S are thus the diagonal entries of $\frac{1}{n-1}D^2$ and the columns of the orthogonal matrix V are the eigenvectors of S.
- We also have

$$\mathbf{X}\mathbf{X}^\top = (UDV^\top)(UDV^\top)^\top = UDV^\top VD^\top U^\top = UDD^\top U^\top,$$

  using orthogonality ($V^\top V = I_p$) of V.
- SVD also gives the optimal low-rank approximations of $\mathbf{X}$:

$$\min_{\tilde{X}} \|\tilde{X} - \mathbf{X}\|^2 \qquad \text{s.t. } \tilde{X} \text{ has maximum rank } r < n, p.$$

  This problem can be solved by keeping only the r largest singular values of X, zeroing out the smaller singular values in the SVD.

Biplots

- PCA plots show the data items (as rows of $\mathbf{X}$) in the PC space.
- **Biplots** allow us to visualize the **original variables** (as columns $\mathbf{X}$) in the same plot.
- As for PCA, we would like the geometry of the plot to preserve as much of the covariance structure as possible.

Biplots

Recall that $X = [X_1, \ldots, X_p]^\top$ and $\mathbf{X} = UDV^\top$ is the SVD of the data matrix.

- The PC projection of x_i is:

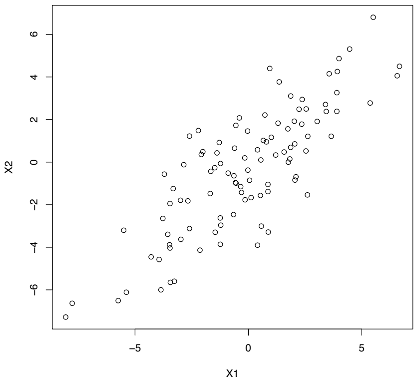
$$z_i = V^\top x_i = DU_i^\top = [D_{11}U_{i1}, \ldots, D_{kk}U_{ik}]^\top.$$

- The jth unit vector $\mathbf{e}_j \in \mathbb{R}^p$ points in the direction of X_j. Its PC projection is $V_j^\top = V^\top\mathbf{e}_j$, the jth row of V.
- The projection of the variable indicates the weighting each PC gives to the original variables.
- Dot products between the projections gives entries of the data matrix:

$$x_{ij} = \sum_{k=1}^{p} U_{ik}D_{kk}V_{jk} = \langle DU_i^\top, V_j^\top \rangle.$$

- Distance of projected points from projected variables gives original location.
- These relationships can be plotted in 2D by focussing on first two PCs.

Biplots



Biplots

- There are other projections we can consider for biplots:

$$x_{ij} = \sum_{k=1}^{p} U_{ik} D_{kk} V_{jk} = \langle DU_i^\top, V_j^\top \rangle = \langle D^{1-\alpha} U_i^\top, D^\alpha V_j^\top \rangle.$$

  where $0 \le \alpha \le 1$. The $\alpha = 1$ case has some nice properties.
- Covariance of the projected points is:

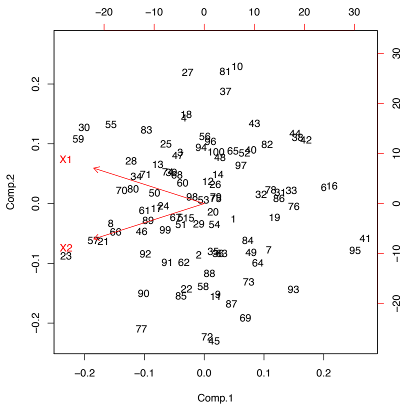
$$\frac{1}{n-1} \sum_{i=1}^{n} U_i^\top U_i = \frac{1}{n-1} I.$$

  Projected points are uncorrelated and dimensions are equi-variance.
- The covariance between X_j and X_ℓ is:

$$\mathrm{Var}(X_j X_\ell) = \frac{1}{n-1} \langle DV_j^\top, DV_\ell^\top \rangle$$

  So the angle between the projected variables gives the correlation.
- When using $k < p$ PCs, quality depends on the proportion of variance explained by the PCs.

Biplots



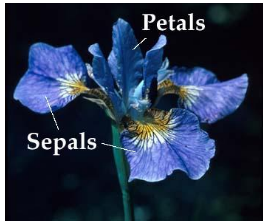
```
pc <- princomp(x)
biplot(pc,scale=0)
biplot(pc,scale=1)
```

Iris Data

50 sample from 3 species of iris: *iris setosa*, *versicolor*, and *virginica*

Each measuring the length and widths of both sepal and petals

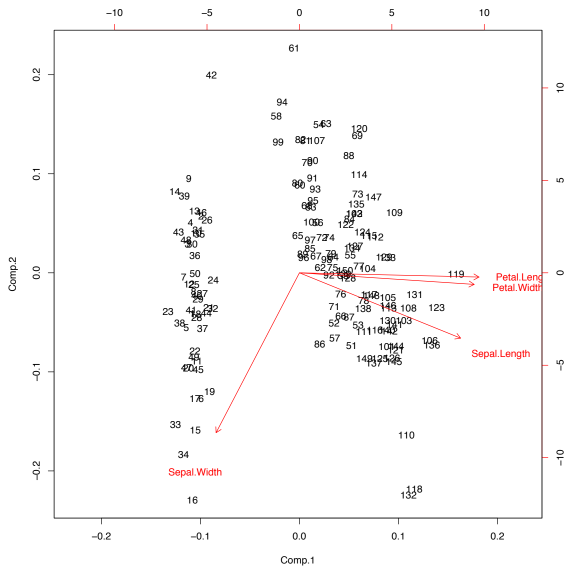
Collected by E. Anderson (1935) and analysed by R.A. Fisher (1936)



Using again function `princomp` and `biplot`.

```
iris1 <- iris
iris1 <- iris1[,-5]
biplot(princomp(iris1,cor=T))
```

Iris Data

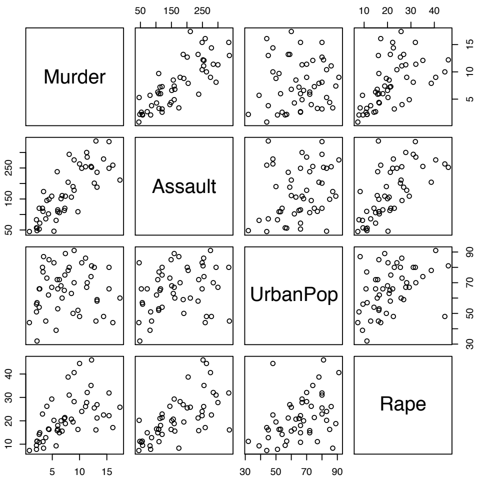


US Arrests Data

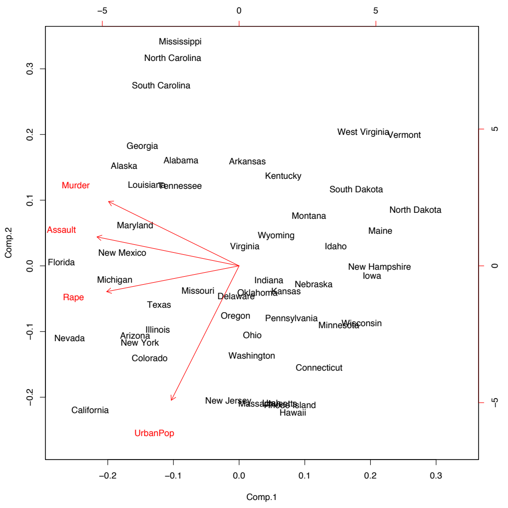
This data set contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

```
pairs(USArrests)
usarrests.pca <- princomp(USArrests,cor=T)
plot(usarrests.pca)

pairs(predict(usarrests.pca))
biplot(usarrests.pca)
```

US Arrests Data Pairs Plot



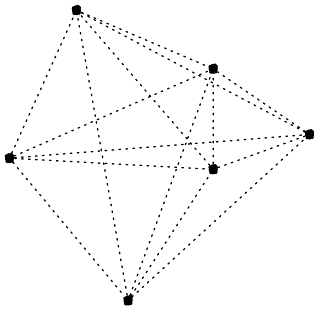
US Arrests Data Biplot



Multidimensional Scaling

Suppose there are n points $\mathbf{X}$ in $\mathbb{R}^p$, but we are only given the $n \times n$ matrix $\mathbf{D}$ of inter-point distances.

Can we reconstruct $\mathbf{X}$?



Multidimensional Scaling

Rigid transformations (translations, rotations and reflections) do not change inter-point distances so cannot recover $\mathbf{X}$ exactly. However $\mathbf{X}$ can be recovered up to these transformations!

- Let $d_{ij} = \|x_i - x_j\|_2$ be the distance between points x_i and x_j.

$$\begin{aligned} d_{ij}^2 &= \|x_i - x_j\|_2^2 \\ &= (x_i - x_j)^\top (x_i - x_j) \\ &= x_i^\top x_i + x_j^\top x_j - 2x_i^\top x_j \end{aligned}$$

- Let $\mathbf{B} = \mathbf{X}\mathbf{X}^\top$ be the $n \times n$ matrix of dot-products, $b_{ij} = x_i^\top x_j$. The above shows that $\mathbf{D}$ can be computed from $\mathbf{B}$.
- Some algebraic exercise shows that $\mathbf{B}$ can be recovered from $\mathbf{D}$ if we assume $\sum_{i=1}^n x_i = 0$.

Multidimensional Scaling

- If we knew $\mathbf{X}$, then SVD gives $\mathbf{X} = UDV^\top$. As $\mathbf{X}$ has rank $k = \min(n, p)$, we have at most k singular values in D and we can assume $U \in \mathbb{R}^{n \times k}$, $D \in \mathbb{R}^{k \times p}$ and $V \in \mathbb{R}^{p \times p}$.
- The eigendecomposition of $\mathbf{B}$ is then:

$$\mathbf{B} = \mathbf{X}\mathbf{X}^\top = UDD^\top U^\top = U\Lambda U^\top.$$

- This eigendecomposition can be obtained from $\mathbf{B}$ without knowledge of $\mathbf{X}$!
- Let $\tilde{x}_i^\top = U_i \Lambda^{\frac{1}{2}}$ be the ith row of $U\Lambda^{\frac{1}{2}}$. Pad $\tilde{x}_i$ with 0s so that it has length p.

$$\tilde{x}_i^\top \tilde{x}_j = U_i \Lambda U_j^\top = b_{ij} = x_i^\top x_j$$

  and we have found a set of vectors with dot-products given by $\mathbf{B}$.
- The vectors $\tilde{x}_i$ differs from x_i only via the orthogonal matrix V so are equivalent up to rotation and reflections.

US City Flight Distances

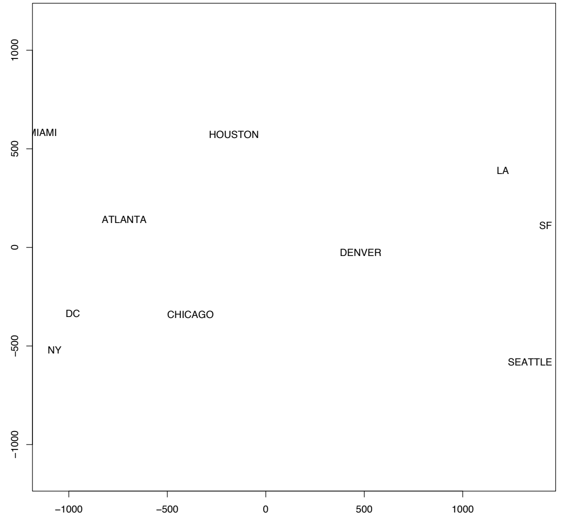
We present a table of flying mileages between 10 American cities, distances calculated from our 2-dimensional world. Using D as the starting point, metric MDS finds a configuration with the same distance matrix.

ATLA	CHIG	DENV	HOUS	LA	MIAM	NY	SF	SEAT	DC
0	587	1212	701	1936	604	748	2139	2182	543
587	0	920	940	1745	1188	713	1858	1737	597
1212	920	0	879	831	1726	1631	949	1021	1494
701	940	879	0	1374	968	1420	1645	1891	1220
1936	1745	831	1374	0	2339	2451	347	959	2300
604	1188	1726	968	2339	0	1092	2594	2734	923
748	713	1631	1420	2451	1092	0	2571	2408	205
2139	1858	949	1645	347	2594	2571	0	678	2442
2182	1737	1021	1891	959	2734	2408	678	0	2329
543	597	1494	1220	2300	923	205	2442	2329	0

US City Flight Distances

```
library(MASS)

us <- read.csv("http://www.stats.ox.ac.uk/
                      ~teh/teaching/smldm/data/uscities.csv")

## use classical MDS to find lower dimensional views of the data
## recover X in 2 dimensions

us.classical <- cmdscale(d=us,k=2)

plot(us.classical)
text(us.classical,labels=names(us))
```

US City Flight Distances



Lower-dimensional Reconstructions

In classical MDS derivation, we used all eigenvalues in the eigendecomposition of $\mathbf{B}$ to reconstruct

$$\tilde{x}_i = U_i \Lambda^{\frac{1}{2}}.$$

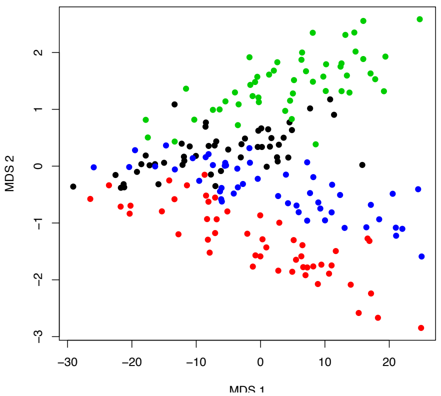
We can use only the largest $k < \min(n, p)$ eigenvalues and eigenvectors in the reconstruction, giving the ‘best’ k-dimensional view of the data.

This is analogous to PCA, where only the largest eigenvalues of $\mathbf{X}^\top \mathbf{X}$ are used, and the smallest ones effectively suppressed.

Indeed, PCA and classical MDS are duals and yield effectively the same result.

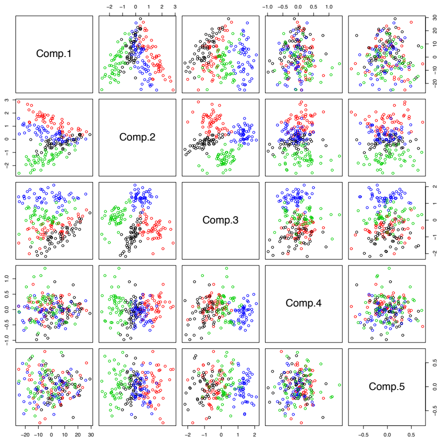
Crabs Data

```
library(MASS)
Crabs <- crabs[,4:8]
Crabs.class <- factor(paste(crabs[,1],crabs[,2],sep=""))

crabsmds <- cmdscale(d= dist(Crabs),k=2)
plot(crabsmds, pch=20, cex=2,col=unclass(Crabs.class))
```



Crabs Data

Compare with previous PCA analysis.
Classical MDS solution corresponds to the first 2 PCs.



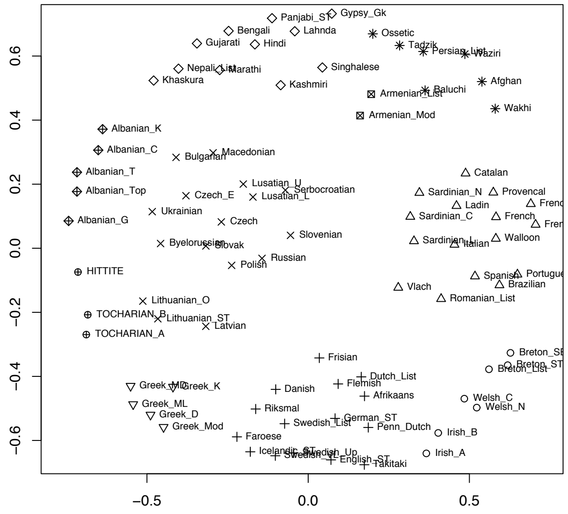
Example: Language data

Presence or absence of 2867 homologous traits in 87 Indo-European languages.

```
> X[1:15,1:16]
                  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13
Irish_A            0  0  0  0  1  0  0  0  0   0   0   0   0
Irish_B            0  0  0  0  1  0  0  0  0   0   0   0   0
Welsh_N            0  0  0  1  0  0  0  0  0   0   0   0   0
Welsh_C            0  0  0  1  0  0  0  0  0   0   0   0   0
Breton_List        0  0  0  0  1  0  0  0  0   0   0   0   0
Breton_SE          0  0  0  0  1  0  0  0  0   0   0   0   0
Breton_ST          0  0  0  0  1  0  0  0  0   0   0   0   0
Romanian_List      0  1  0  0  0  0  0  0  0   0   0   0   0
Vlach              0  1  0  0  0  0  0  0  0   0   0   0   0
Italian            0  1  0  0  0  0  0  0  0   0   0   0   0
Ladin              0  1  0  0  0  0  0  0  0   0   0   0   0
Provencal          0  1  0  0  0  0  0  0  0   0   0   0   0
French             0  1  0  0  0  0  0  0  0   0   0   0   0
Walloon            0  1  0  0  0  0  0  0  0   0   0   0   0
French_Creole_C    0  1  0  0  0  0  0  0  0   0   0   0   0
```

Example: Language data

Using MDS with non-metric scaling.



Varieties of MDS

Generally, MDS is a class of dimensionality reduction techniques which represents data points $x_1, \dots, x_n \in \mathbb{R}^p$ in a lower-dimensional space $z_1, \dots, z_n \in \mathbb{R}^k$ which tries to preserve inter-point (dis)similarities.

- It requires only the matrix $\mathbf{D}$ of pairwise dissimilarities

  $$d_{ij} = d(x_i, d_j).$$

  For example we can use Euclidean distance $d_{ij} = \|x_i - x_j\|_2$. Other dissimilarities are possible. Conversely, it can use a matrix of similarities.
- MDS finds representations $z_1, \dots, z_n \in \mathbb{R}^k$ such that

  $$d(x_i, x_j) \approx \tilde{d}_{ij} = \tilde{d}(z_i, z_j),$$

  where $\tilde{d}$ represents dissimilarity in the reduced k-dimensional space, and differences in dissimilarities are measured by a **stress function** $S(d_{ij}, \tilde{d}_{ij})$.

Varieties of MDS

Choices of (dis)similarities and stress functions lead to different objective functions and different algorithms.

- Classical - preserves similarities instead

  $$S(\mathbf{Z}) = \sum_{i \neq j} (s_{ij} - \langle z_i - \bar{z}, z_j - \bar{z} \rangle)^2$$

- Metric Shepard-Kruskal

  $$S(\mathbf{Z}) = \sum_{i \neq j} (d_{ij} - \|z_i - z_j\|_2)^2$$

- Sammon - preserves shorter distances more

  $$S(\mathbf{Z}) = \sum_{i \neq j} \frac{(d_{ij} - \|z_i - z_j\|_2)^2}{d_{ij}}$$

- Non-Metric Shepard-Kruskal - ignores actual distance values, only ranks

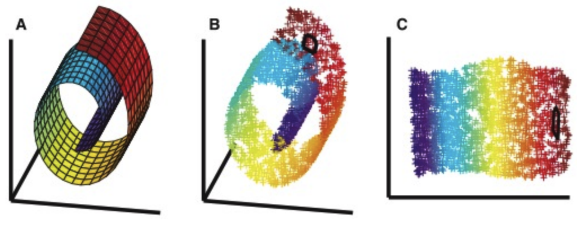
  $$S(\mathbf{Z}) = \min_{g \text{ increasing}} \sum_{i \neq j} (g(d_{ij}) - \|z_i - z_j\|_2)^2$$

Nonlinear Dimensionality Reduction

Two aims of different varieties of MDS:

- To visualize the (dis)similarities among items in a dataset, where these (dis)disimilarities may not have Euclidean geometric interpretations.
- To perform **nonlinear** dimensionality reduction.

Many high-dimensional datasets exhibit low-dimensional structure ("live on a low-dimensional menifold").

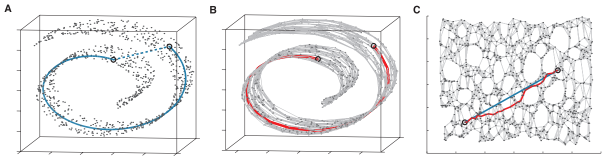


high-dim distribution   high-dim samples   estimated manifold

Isomap

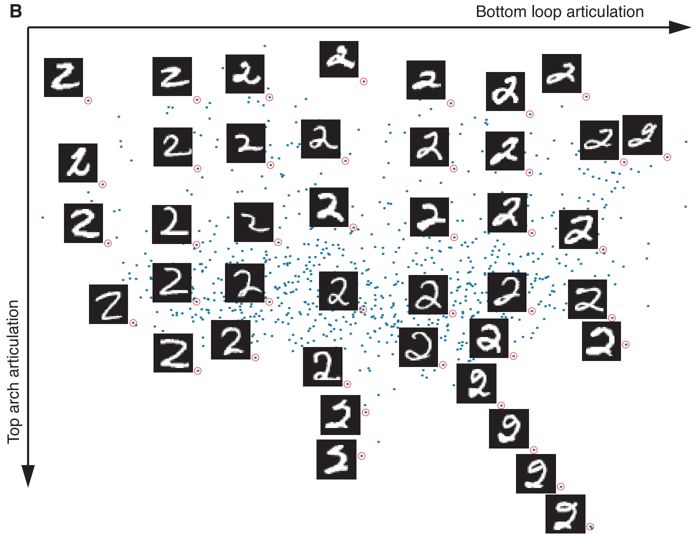
Isomap is a non-linear dimensional reduction technique based on classical MDS. Differs from other MDSs in its estimate of distances d_{ij}.

1. Calculate distances d_{ij} for $i, j = 1, \ldots, n$ between all data points, using the Euclidean distance.
2. Form a graph G with the n samples as nodes, and edges between the respective K nearest neighbours.
3. Replace distances d_{ij} by shortest-path distance on graph d^G_{ij} and perform classical MDS, using these distances.

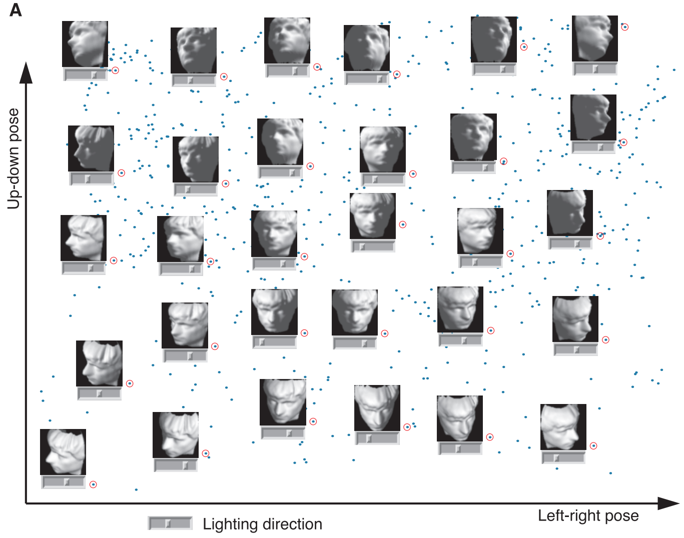


Examples from Tenenbaum et al. (2000).

Handwritten Characters



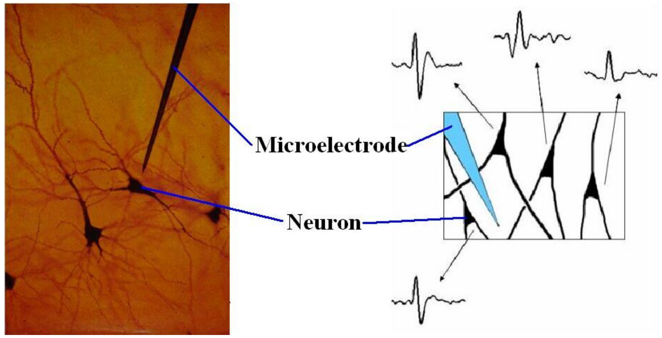
Faces



Other Nonlinear Dimensionality Reduction Techniques

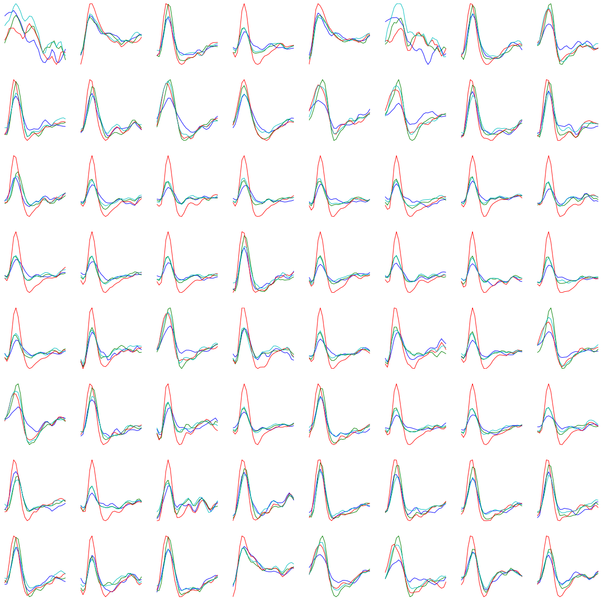
- Locally Linear Embedding.
- Laplacian Eigenmaps.
- Maximum Variance Unfolding.

Neural Electroencephalography (EEG)

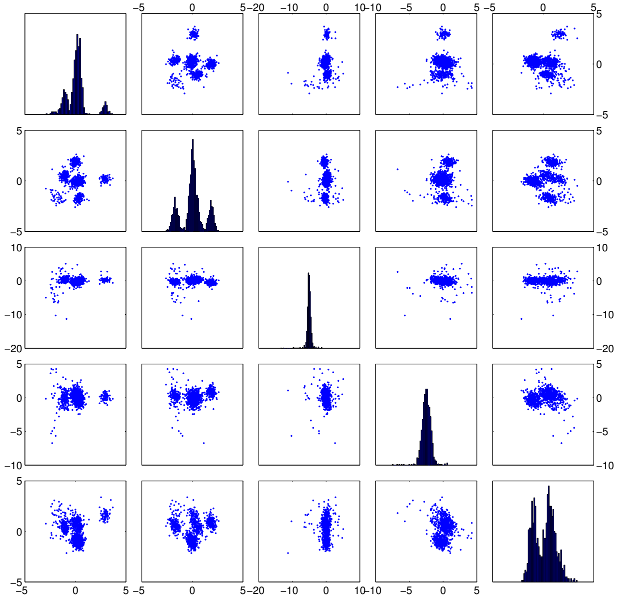


http://newton.umsl.edu/tsytsarev_files/Lecture02.htm

Neural Spike Waveforms



Pairs Plot of Principal Components



Clustering

- Many datasets consist of multiple heterogeneous subsets. Cluster analysis is a range of methods that reveal this heterogeneity by discovering clusters of similar points.
- Model-based clustering:
  - Each cluster is described using a probability model.
- Model-free clustering:
  - Defined by similarity among points within clusters (dissimilarity among points between clusters).
- Partition-based clustering methods:
  - Allocate points into K clusters.
  - The number of cluster is usually fixed beforehand or investigated for various values of K as part of the analysis.
- Hierarchy-based clustering methods:
  - Allocate points into clusters and clusters into super-clusters forming a hierarchy.
  - Typically the hierarchy forms a binary tree (a dendrogram) where each cluster has two “children”.

Hierarchical Clustering

- Hierarchically structured data can be found everywhere (measurements of different species and different individuals within species), hierarchical methods attempt to understand data by looking for clusters.
- There are two general strategies for generating hierarchical clusters. Both proceed by seeking to minimize some measure of dissimilarity.
  - Agglomerative / Bottom-Up / Merging
  - Divisive / Top-Down / Splitting

  Hierarchical clusters are generated where at each level, clusters are created by merging clusters at lower levels. This process can easily be viewed by a dendogram/tree.

Measuring Dissimilarity

To find hierarchical clusters, we need some way to measure the dissimilarity between clusters

- Given two points x_i and x_j, it is straightforward to measure their dissimilarity, say $d(x_i, x_j) = \|x_i - x_j\|_2$.
- It is unclear however how to extend this to measure dissimilarity between **clusters**, $D(C_i, C_j)$ for clusters C_i and C_j.

Many such proposals though no concensus as to which is best.

(a) **Single Linkage**

$$D(C_i, C_j) = \min_{x,y} \left(d(x, y) | x \in C_i, y \in C_j \right)$$

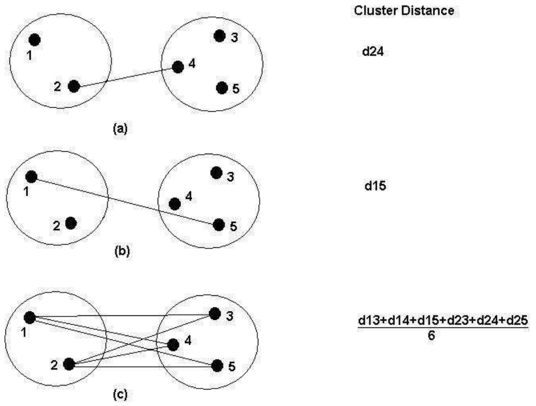
(b) **Complete Linkage**

$$D(C_i, C_j) = \max_{x,y} \left(d(x, y) | x \in C_i, y \in C_j \right)$$

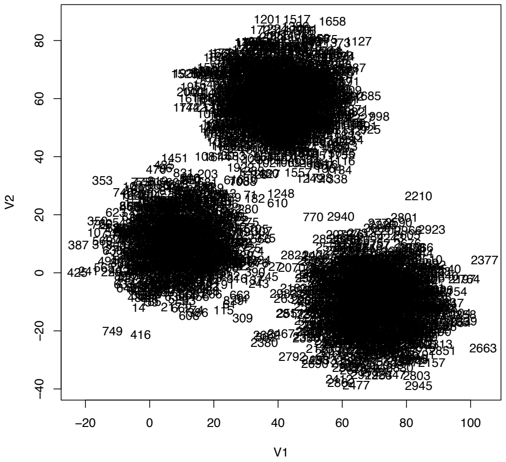
(c) **Average Linkage**

$$D(C_i, C_j) = avg_{x,y} \left(d(x, y) | x \in C_i, y \in C_j \right)$$

Measuring Dissimilarity



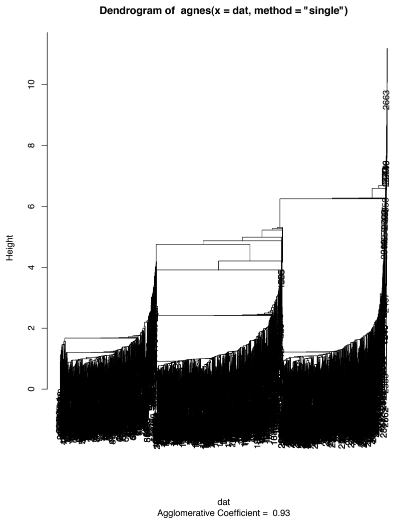
Hierarchical Clustering on Artificial Dataset



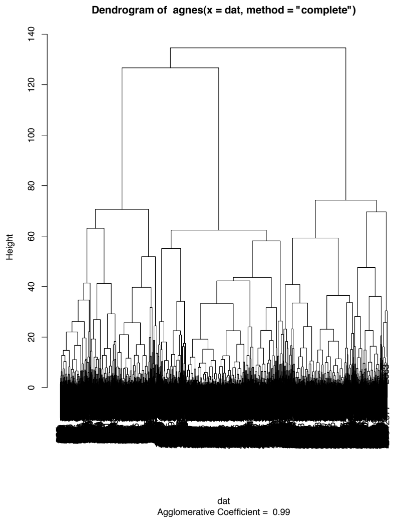
Hierarchical Clustering on Artificial Dataset

```
#start afresh
dat=xclara #3000 x 2
library(cluster)

#plot the data
plot(dat,type="n")
text(dat,labels=row.names(dat))

plot(agnes(dat,method="single"))
plot(agnes(dat,method="complete"))
plot(agnes(dat,method="average"))
```

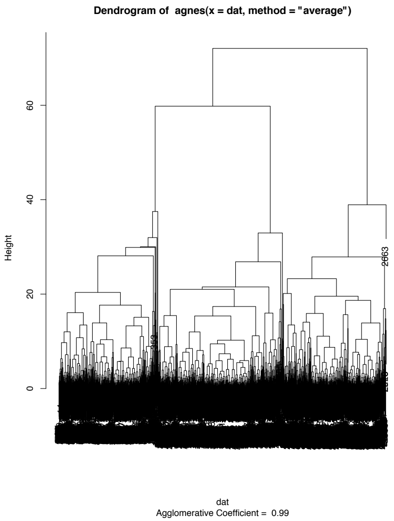
Hierarchical Clustering on Artificial Dataset



Hierarchical Clustering on Artificial Dataset



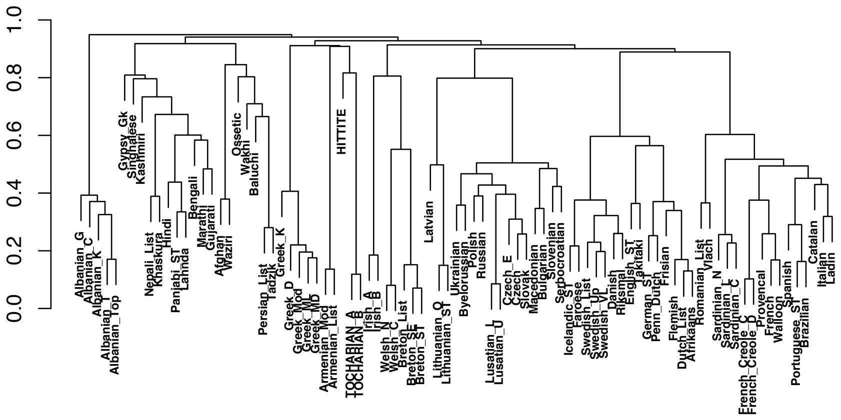
Hierarchical Clustering on Artificial Dataset



Using Dendograms

- Different ways of measuring dissimilarity result in different trees.
- Dendograms are useful for getting a feel for the structure of high-dimensional data though they don’t represent distances between observations well.
- Dendograms show hierarchical clusters with respect to increasing values of dissimilarity between clusters, cutting a dendogram horizontally at a particular height partitions the data into disjoint clusters which are represented by the vertical lines it intersects. Cutting horizontally effectively reveals the state of the clustering algorithm when the dissimilarity value between clusters is no more than the value cut at.
- Despite the simplicity of this idea and the above drawbacks, hierarchical clustering methods provide users with interpretable dendograms that allow clusters in high-dimensional data to be better understood.

Hierarchical Clustering on Indo-European Languages



K-means

Partition-based methods seek to divide data points into a pre-assigned number of clusters $C_1, \dots, C_K$ where for all $k, k' \in \{1, \dots, K\}$,

$$C_k \subset \{1, \dots, n\}, \qquad C_k \cap C_{k'} = \emptyset \;\; \forall k \neq k', \qquad \bigcup_{k=1}^{K} C_k = \{1, \dots, n\}.$$

For each cluster, represent it using a **prototype** or **cluster centre** μ_k.
We can measure the quality of a cluster with its **within-cluster deviance**

$$W(C_k, \mu_k) = \sum_{i \in C_k} \|x_i - \mu_k\|_2^2.$$

The overall quality of the clustering is given by the total within-cluster deviance:

$$W = \sum_{k=1}^{K} W(C_k, \mu_k).$$

The overall objective is to choose both the cluster centres and allocation of points to minimize the **objective function**.

K-means

$$W = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|_2^2 = \sum_{i=1}^{n} \|x_i - \mu_{c_i}\|_2^2$$

where $c_i = k$ if and only if $i \in C_k$.

- Given partition $\{C_k\}$, we can find the optimal prototypes easily by differentiating W with respect to μ_k:

$$\frac{\partial W}{\partial \mu_k} = 2 \sum_{i \in C_k} (x_i - \mu_k) = 0 \qquad \Rightarrow \mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

- Given prototypes, we can easily find the optimal partition by assigning each data point to the closest cluster prototype:

$$c_i = \operatorname*{argmin}_k \|x_i - \mu_k\|_2^2$$

But joint minimization over both is computationally difficult.

K-means

The K-means algorithm is a well-known method that **locally optimizes** the objective function W.

Iterative and alternating minimization.

1. Randomly fix K cluster centres $\mu_1, \dots, \mu_K$.
2. For each $i = 1, \dots, n$, assign each x_i to the cluster with the nearest centre,

$$c_i := \operatorname*{argmin}_k \|x_i - \mu_k\|_2^2$$

3. Set $C_k := \{i : c_i = k\}$ for each k.
4. Move cluster centres $\mu_1, \dots, \mu_K$ to the average of the new clusters:

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

5. Repeat steps 2 to 4 until there is no more changes.
6. Return the partition $\{C_1, \dots, C_K\}$ and means $\mu_1, \dots, \mu_K$ at the end.

K-means

Some notes about the K-means algorithm.

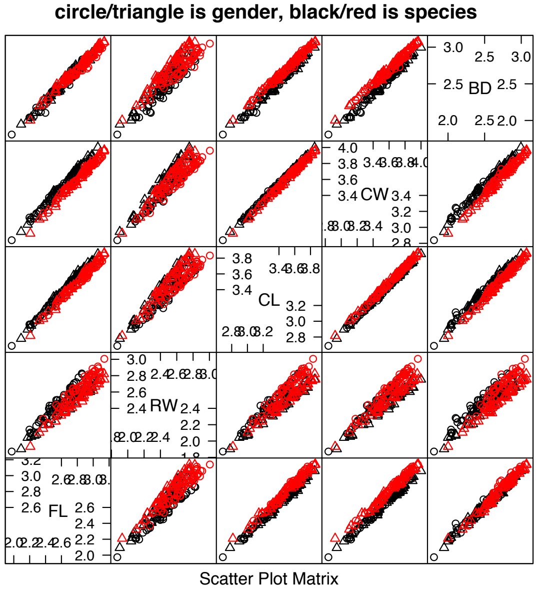
- **The algorithm stops in a finite number of iterations.** Between steps 2 and 3, W either stays constant or it decreases, this implies that we never revisit the same partition. As there are only finitely many partitions, the number of iterations cannot exceed this.
- **The K-means algorithm need not converge to global optimum.** K-means is a heuristic search algorithm so it can get stuck at suboptimal configurations. The result depends on the starting configuration. Typically perform a number of runs from different configurations, and pick best clustering.

K-means on Crabs

Looking at the Crabs data again.

```
library(MASS)
library(lattice)
data(crabs)

splom(~log(crabs[,4:8]),
    col=as.numeric(crabs[,1]),
    pch=as.numeric(crabs[,2]),
    main="circle/triangle is gender, black/red is species")
```

K-means on Crabs

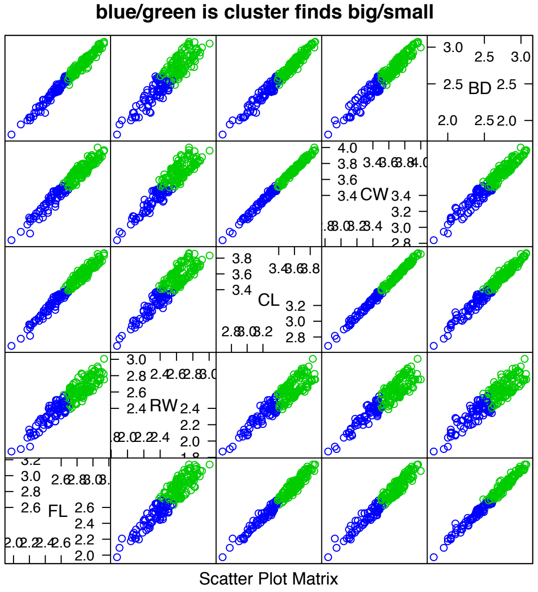


K-means on Crabs

Apply K-means with 2 clusters and plot results.

```
cl <- kmeans( log(crabs[,4:8]), 2, nstart=1, iter.max=10)

splom(~log(crabs[,4:8]),
      col=cl$cluster+2,
      main="blue/green is cluster finds big/small")
```

K-means on Crabs



K-means on Crabs

‘Whiten’ or ‘sphere’[1] the data using PCA.

```
pcp <- princomp( log(crabs[,4:8]) )
spc <- pcp$scores %*% diag(1/pcp$sdev)
splom( ~spc[,1:3],
    col=as.numeric(crabs[,1]),
    pch=as.numeric(crabs[,2]),
    main="circle/triangle is gender, black/red is species")
```

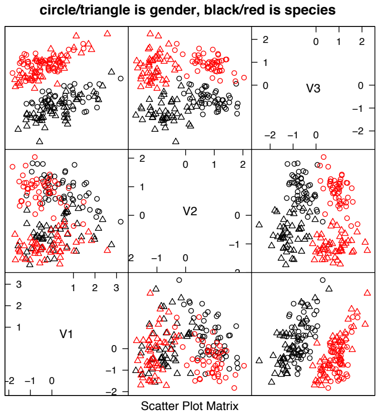
And apply K-means again.

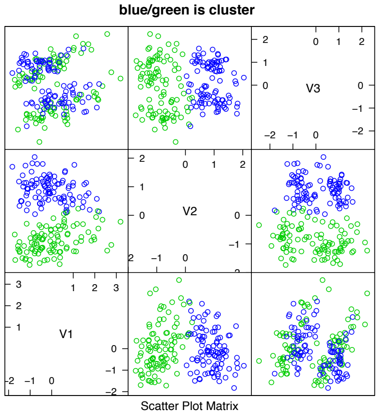
```
cl <- kmeans(spc, 2, nstart=1, iter.max=20)
splom( ~spc[,1:3],
      col=cl$cluster+2, main="blue/green is cluster")
```

---

[1] Apply a linear transformation so that covariance matrix is identity.

K-means on Crabs

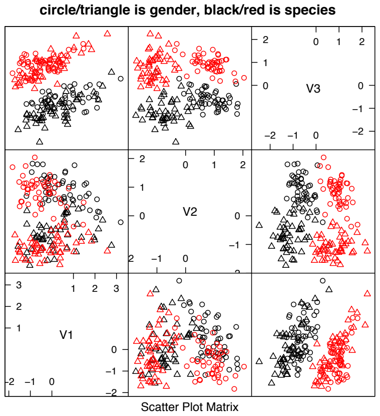


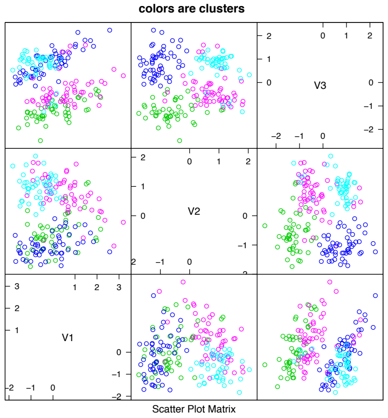


Discovers gender difference...
Results depends crucially on sphering the data first.

K-means on Crabs

Using 4 cluster centers.

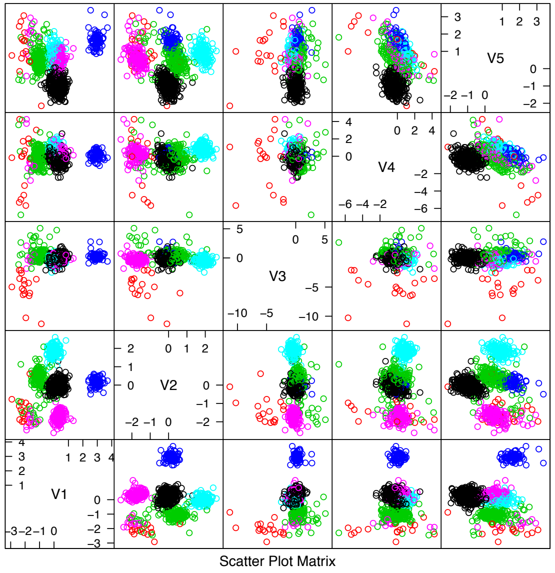




K-means on Spike Waveforms

```
library(MASS)
library(lattice)
spikespca <- read.table("spikes.txt")
cl <- kmeans(data,6,nstart=20)
splom(data,col=cl$cluster)
```

K-means on Spike Waveforms



Stochastic Optimization

- Each iteration of K-means requires a pass through whole dataset. In extremely large datasets, this can be computationally prohibitive.
- Stochastic optimization: update cluster means after assigning each data point to the closest cluster.
- Repeat for $t = 1, 2, \ldots$ until satisfactory convergence:
  1. Pick data item x_i either randomly or in order.
  2. Assign x_i to the cluster with the nearest centre,
  $$c_i := \underset{k}{\operatorname{argmin}} \|x_i - \mu_k\|_2^2$$
  3. Update cluster centre:
  $$\mu_k := \mu_k + \alpha_t (x_i - \mu_k)$$
  where $\alpha_t > 0$ are **step sizes**.
- Algorithm stochastically minimizes the objective function. Convergence requires slowly decreasing step sizes:

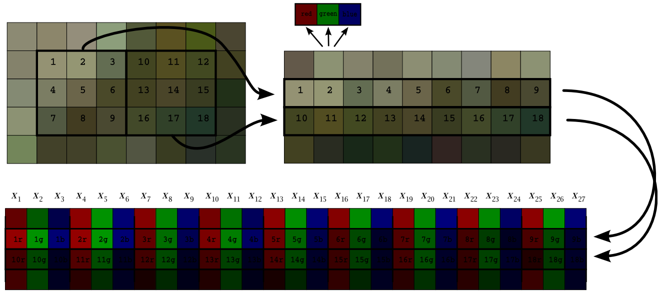
$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Vector Quantization

- A related algorithm developed in the signal processing literature for **lossy data compression**.
- If $K \ll n$, we can store the **codebook** of **codewords** $\mu_1, \ldots, \mu_K$, and each vector x_i is encoded using c_i, which only requires $\lceil \log K \rceil$ bits.
- As with K-means, K must be specified. Increasing K improves the quality of the compressed image but worsens the data compression rate, so there is a clear tradeoff.
- Some audio and video codecs use this method.
- Stochastic optimization algorithm for K-means was originally developed for VQ.

VQ Image Compression

3×3 block VQ: View each block of 3×3 pixels as single observation



VQ Image Compression

Original image (24 bits/pixel, uncompressed size 1,402 kB)

VQ Image Compression

Codebook length 1024 (1.11 bits/pixel, total size 88kB)

VQ Image Compression

Codebook length 128 (0.78 bits/pixel, total size 50kB)

VQ Image Compression

Codebook length 16 (0.44 bits/pixel, total size 27kB)

K-means Additional Comments

- **Sensitivity to distance measure.** Euclidean distance can be greatly affected by measurement unit and by strong correlations. Can use Mahalanobis distance,

  $$\|x - y\|_M = \sqrt{(x-y)^\top M^{-1}(x-y)}$$

  where M is positive semi-definite matrix, e.g. sample covariance.
- **Other partition based methods.** There are many other partition based methods that employ related ideas. For example K-medoids differs from K-means in requiring cluster centres μ_i to be an observation x_i[2], K-medians (use median in each dimension) and K-modes (use mode).
- **Determination of K.** The K-means objective will always improve with larger number of clusters K. Determination of K requires an additional **regularization** criterion. E.g., in DP-means[3], use

  $$W = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|_2^2 + \lambda K$$

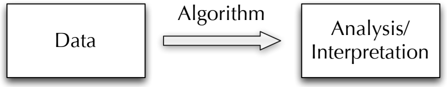
---

[2] See also Affinity propagation.

[3] DP-means paper.

Probabilistic Methods

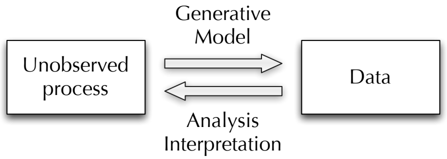
- Algorithmic approach:


Data

Algorithm

Analysis/ Interpretation

- Probabilistic modelling approach:


Generative Model

Unobserved process

Data

Analysis Interpretation

Mixture Models

- Mixture models suppose that our dataset was created by sampling iid from K distinct populations (called **mixture components**).
- Typical samples in population k can be modelled using a distribution $F(\phi_k)$ with density $f(x|\phi_k)$. For a concrete example, consider a Gaussian with unknown mean ϕ_k and known symmetric covariance $\sigma^2 I$,

$$f(x|\phi_k) = |2\pi\sigma^2|^{-\frac{p}{2}} \exp\left(-\frac{1}{2\sigma^2}\|x-\phi_k\|_2^2\right).$$

- Generative process: for $i = 1, 2, \ldots, n$:
  - First determine which population item i came from (independently):

$$Z_i \sim \text{Discrete}(\pi_1, \ldots, \pi_K) \qquad \text{i.e. } \mathbb{P}(Z_i = k) = \pi_k$$

    where **mixing proportions** are $\pi_k \geq 0$ for each k and $\sum_{k=1}^K \pi_k = 1$.
  - If $Z_i = k$, then $X_i = (X_{i1}, \ldots, X_{ip})^\top$ is sampled (independently) from corresponding population distribution:

$$X_i|Z_i = k \sim F(\phi_k)$$

- We observe that $X_i = x_i$ for each i, and would like to learn about the unknown parameters of the process.

Mixture Models

- Unknowns to learn given data are
  - **Parameters**: $\pi_1, \ldots, \pi_K, \phi_1, \ldots, \phi_K$, as well as
  - **Latent variables**: $z_1, \ldots, z_K$.
- The joint probability over all cluster indicator variables $\{Z_i\}$ are:

$$p_Z((z_i)_{i=1}^n) = \prod_{i=1}^n \pi_{z_i} = \prod_{i=1}^n \prod_{k=1}^K \pi_k^{\mathbb{1}(z_i=k)}$$

- The joint density at observations $X_i = x_i$ given $Z_i = z_i$ are:

$$p_X((x_i)_{i=1}^n|(Z_i = z_i)_{i=1}^n) = \prod_{i=1}^n \prod_{k=1}^K f(x_i|\phi_k)^{\mathbb{1}(z_i=k)}$$

- So the joint probability/density[4] is:

$$p_{X,Z}((x_i, z_i)_{i=1}^n) = \prod_{i=1}^n \prod_{k=1}^K (\pi_k f(x_i|\phi_k))^{\mathbb{1}(z_i=k)}$$

---

[4] In this course we will treat probabilities and densities equivalently for notational simplicity. In general, the quantity is a density with respect to the product base measure, where the base measure is the counting measure for discrete variables and Lebesgue for continuous variables.

Mixture Models - Posterior Distribution

- Suppose we know the parameters $(\pi_k, \phi_k)_{k=1}^K$.
- Z_i is a random variable, so the posterior distribution given data set $\mathbf{X}$ tells us what we know about it:

$$Q_{ik} := p(Z_i = k|x_i) = \frac{p(Z_i = k, x_i)}{p(x_i)} = \frac{\pi_k f(x_i|\phi_k)}{\sum_{j=1}^K \pi_j f(x_i|\phi_j)}$$

  where the marginal probability is:

$$p(x_i) = \sum_{j=1}^K \pi_j f(x_i|\phi_j)$$

- The posterior probability Q_{ik} of $Z_i = k$ is called the **responsibility** of mixture component k for data point x_i.
- The posterior distribution **softly partitions** the dataset among the k components.

Mixture Models - Maximum Likehood

- How can we learn about the parameters $\theta = (\pi_k, \phi_k)_{k=1}^K$ from data?
- Standard statistical methodology asks for the **maximum likelihood** estimator (MLE).
- The log likelihood is the log marginal probability of the data:

$$\ell((\pi_k, \phi_k)_{k=1}^K) := \log p((x_i)_{i=1}^n | (\pi_k, \phi_k)_{k=1}^K) = \sum_{i=1}^n \log \sum_{j=1}^K \pi_j f(x_i|\phi_j)$$

$$\begin{aligned}\nabla_{\phi_k} \ell((\pi_k, \phi_k)_{k=1}^K) &= \sum_{i=1}^n \frac{\pi_k f(x_i|\phi_k)}{\sum_{j=1}^K \pi_j f(x_i|\phi_j)} \nabla_{\phi_k} \log f(x_i|\phi_k) \\ &= \sum_{i=1}^n Q_{ik} \nabla_{\phi_k} \log f(x_i|\phi_k)\end{aligned}$$

- A difficult equation to solve, as Q_{ik} depends implicitly on ϕ_k...

Mixture Models - Maximum Likehood

$$\sum_{i=1}^n Q_{ik} \nabla_{\phi_k} \log f(x_i|\phi_k) = 0$$

- What if we ignore the dependence of Q_{ik} on the parameters?
- Taking the mixture of Gaussian with covariance $\sigma^2 I$ as example,

$$\begin{aligned}&\sum_{i=1}^n Q_{ik} \nabla_{\phi_k} \left(-\frac{p}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \|x_i - \phi_k\|_2^2 \right) \\ =&\frac{1}{\sigma^2} \sum_{i=1}^n Q_{ik}(x_i - \phi_k) = \frac{1}{\sigma^2} \left(\left(\textstyle\sum_{i=1}^n Q_{ik} x_i \right) - \phi_k \left(\textstyle\sum_{i=1}^n Q_{ik} \right) \right) = 0 \\ \phi_k^{MLE?} =& \frac{\sum_{i=1}^n Q_{ik} x_i}{\sum_{i=1}^n Q_{ik}}\end{aligned}$$

Mixture Models - Maximum Likehood

- The estimate is a weighted average of data points, where the estimated mean of cluster k uses its responsibilities to data points as weights.

$$\phi_k^{MLE?} = \frac{\sum_{i=1}^n Q_{ik} x_i}{\sum_{i=1}^n Q_{ik}}$$

- Makes sense: Suppose we knew that data point x_i came from population z_i. Then $Q_{iz_i} = 1$ and $Q_{ik} = 0$ for $k \neq z_i$ and:

$$\pi_k^{MLE} = \frac{\sum_{i:z_i=k} x_i}{\sum_{i:z_i=k} 1}$$

- Our best guess of the originating population is given by Q_{ik}.

Mixture Models - Maximum Likehood

- For the mixing proportions, we can similarly derive an estimator.
- Include a Lagrange multiplier λ to enforce constraint $\sum_k \pi_k = 1$.

$$\begin{aligned}&\nabla_{\log \pi_k} \Big(\ell((\pi_k, \phi_k)_{k=1}^K) - \lambda(\textstyle\sum_{k=1}^K \pi_k - 1) \Big) \\ =& \sum_{i=1}^n \frac{\pi_k f(x_i|\phi_k)}{\sum_{j=1}^K \pi_j f(x_i|\phi_j)} - \lambda \pi_k \\ =& \sum_{i=1}^n Q_{ik} - \lambda \pi_k = 0 \\ \pi_k^{MLE?} =& \frac{\sum_{i=1}^n Q_{ik}}{n}\end{aligned}$$

- Again makes sense: the estimate is simply (our best guess of) the proportion of data points coming from population k.

Mixture Models - The EM Algorithm

- Putting all the derivations together, we get an iterative algorithm for learning about the unknowns in the mixture model.
- Start with some initial parameters $(\pi_k^{(0)}, \phi_l^{(0)})_{k=1}^K$.
- Iterate for $t = 1, 2, \ldots$:
  - **Expectation Step**:

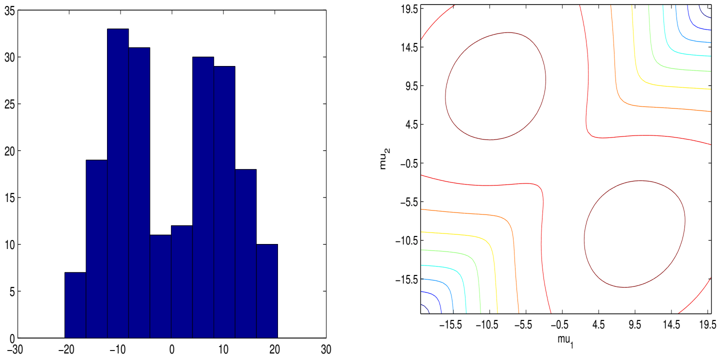
$$Q_{ik}^{(t)} := \frac{\pi_k^{(t-1)} f(x_i|\phi_k^{(t-1)})}{\sum_{j=1}^K \pi_j^{(t-1)} f(x_i|\phi_j^{(t-1)})}$$

  - **Maximization Step**:

$$\pi_k^{(t)} = \frac{\sum_{i=1}^n Q_{ik}^{(t)}}{n} \qquad \phi_k^{(t)} = \frac{\sum_{i=1}^n Q_{ik}^{(t)} x_i}{\sum_{i=1}^n Q_{ik}^{(t)}}$$

- Will the algorithm converge?
- What does it converge to?

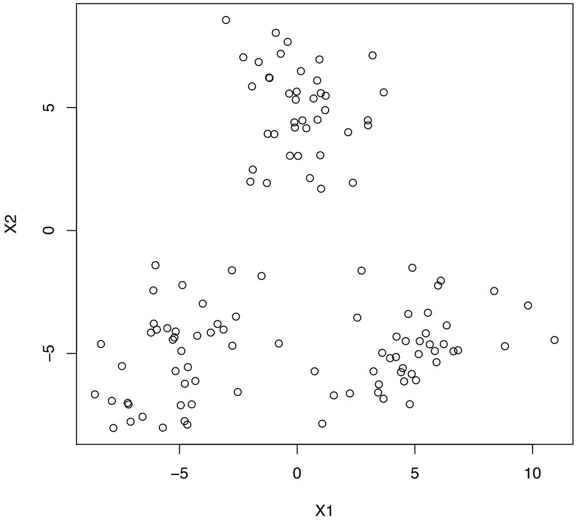
Likelihood Surface for a Simple Example



(left) $n = 200$ data points from a mixture of two 1D Gaussians with $\pi_1 = \pi_2 = 0.5$, $\sigma = 5$ and $\mu_1 = 10, \mu_2 = -10$.
(right) Log likelihood surface $\ell(\mu_1, \mu_2)$, all the other parameters being assumed known.

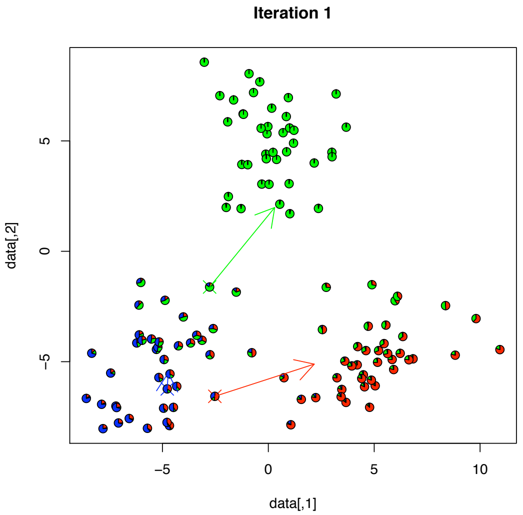
Example: Mixture of 3 Gaussians

An example with 3 clusters.



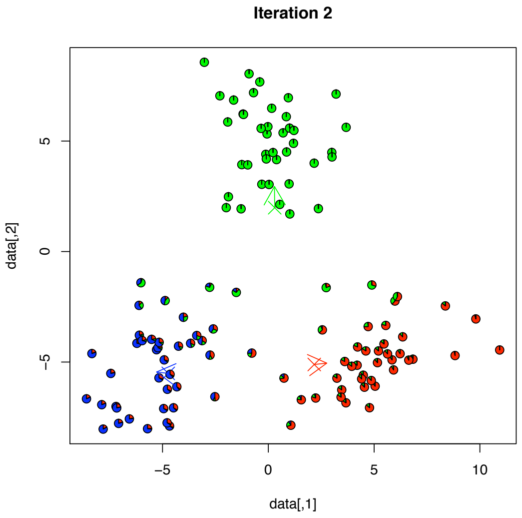
Example: Mixture of 3 Gaussians

After 1st E and M step.



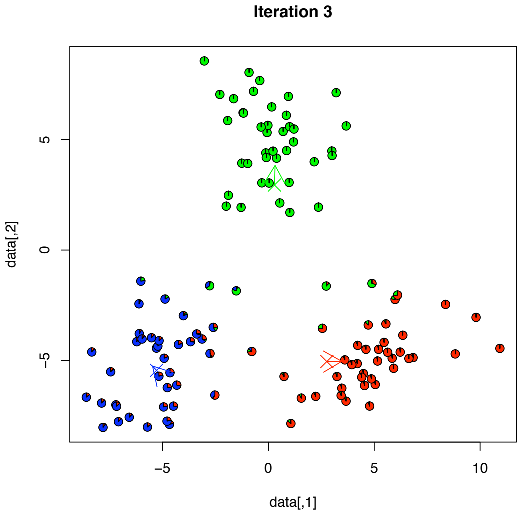
Example: Mixture of 3 Gaussians

After 2nd E and M step.



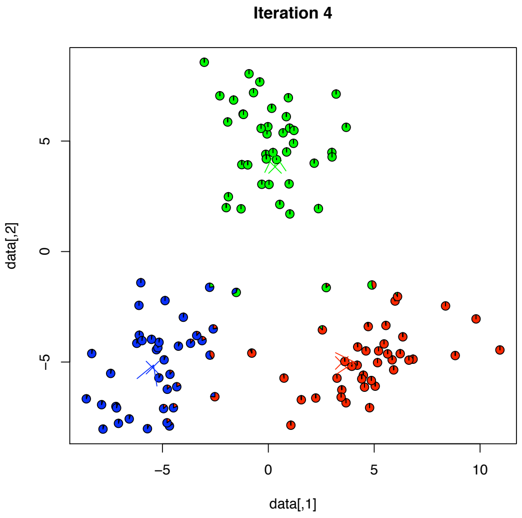
Example: Mixture of 3 Gaussians

After 3rd E and M step.



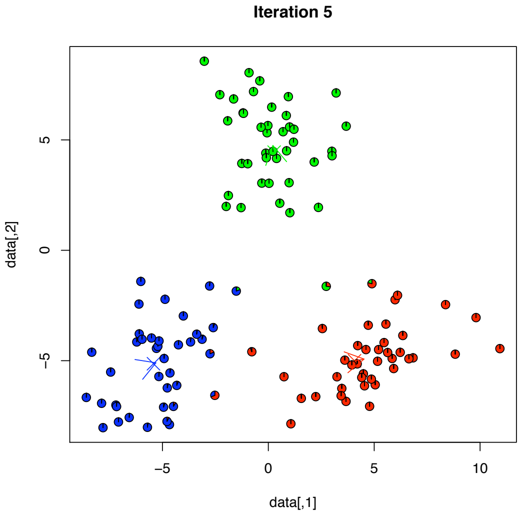
Example: Mixture of 3 Gaussians

After 4th E and M step.



Example: Mixture of 3 Gaussians

After 5th E and M step.



The EM Algorithm

- In a maximum likelihood framework, the objective function is the log likelihood,

$$\ell(\theta) = \sum_{i=1}^{n} \log \sum_{j=1}^{K} \pi_j f(x_i|\phi_j)$$

  Direct maximization is not feasible.
- Consider another objective function $\mathcal{F}(\theta, q)$ such that:

$$\mathcal{F}(\theta, q) \le \ell(\theta) \text{ for all } \theta,\ q,$$
$$\max_q \mathcal{F}(\theta, q) = \ell(\theta)$$

  $\mathcal{F}(\theta, q)$ is a lower bound on the log likelihood.
- We can construct an alternating maximization algorithm as follows: For $t = 1, 2 \ldots$ until convergence:

$$q^{(t)} := \operatorname*{argmax}_q \mathcal{F}(\theta^{(t-1)}, q)$$
$$\theta^{(t)} := \operatorname*{argmax}_\theta \mathcal{F}(\theta, q^{(t)})$$

EM Algorithm

- The lower bound we use is called the **variational free energy**.
- q is a probability mass function for some distribution over (Z_i) and

$$\begin{aligned}\mathcal{F}(\theta, q) =& \mathbb{E}_q[\log p((x_i, z_i)_{i=1}^n) - \log q((z_i)_{i=1}^n)] \\ =& \mathbb{E}_q\left[\left(\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(z_i = k)\left(\log \pi_k + \log f(x_i|\phi_k)\right)\right) - \log q(\mathbf{z})\right] \\ =& \sum_{\mathbf{z}} q(\mathbf{z}) \left[\left(\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(z_i = k)\left(\log \pi_k + \log f(x_i|\phi_k)\right)\right) - \log q(\mathbf{z})\right]\end{aligned}$$

Using $\mathbf{z} := (z_i)_{i=1}^n$ to shorten notation.

EM Algorithm - Solving for q

- Introducing Lagrange multiplier to enforce $\sum_{\mathbf{z}} q(\mathbf{z}) = 1$, and setting derivatives to 0,

$$\begin{aligned}\nabla_{q(\mathbf{z})} \mathcal{F}(\theta, q) &= \sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(z_i = k)\left(\log \pi_k + \log f(x_i|\phi_k)\right) - \log q(\mathbf{z}) - 1 - \lambda \\ &= \sum_{i=1}^{n} \left(\log \pi_{z_i} + \log f(x_i|\phi_{z_i})\right) - \log q(\mathbf{z}) - 1 - \lambda = 0 \\ q^*(\mathbf{z}) &= \frac{\prod_{i=1}^n \pi_{z_i} f(x_i|\phi_{z_i})}{\sum_{\mathbf{z}'} \prod_{i=1}^n \pi_{z'_i} f(x_i|\phi_{z'_i})} = \prod_{i=1}^{n} \frac{\pi_{z_i} f(x_i|\phi_{z_i})}{\sum_k \pi_k f(x_i|\phi_k)} = \prod_{i=1}^{n} p(z_i|x_i, \theta)\end{aligned}$$

- Optimal q^* is simply the posterior distribution.
- Plugging in optimal q^* into the variational free energy,

$$\mathcal{F}(\theta, q^*) = \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k f(x_i|\phi_k) = \ell(\theta)$$

EM Algorithm - Solving for θ

- Setting derivative with respect to ϕ_k to 0,

$$\begin{aligned}\nabla_{\phi_k} \mathcal{F}(\theta, q) &= \sum_{\mathbf{z}} q(\mathbf{z}) \sum_{i=1}^{n} \mathbb{1}(z_i = k) \nabla_{\phi_k} \log f(x_i|\phi_k) \\ &= \sum_{i=1}^{n} q(z_i = k) \nabla_{\phi_k} \log f(x_i|\phi_k) = 0\end{aligned}$$

- This equation can be solved quite easily. E.g., for mixture of Gaussians,

$$\phi_k^* = \frac{\sum_{i=1}^n q(z_i = k) x_i}{\sum_{i=1}^n q(z_i = k)}$$

- If it cannot be solved exactly, we can use **gradient ascent** algorithm:

$$\phi_k^* = \phi_k + \alpha \sum_{i=1}^{n} q(z_i = k) \nabla_{\phi_k} \log f(x_i|\phi_k)$$

- This leads to **generalized EM algorithm**. Further extension using **stochastic optimization** method leads to **stochastic EM algorithm**.
- Similar derivation for optimal π_k as before.

EM Algorithm

- Start with some initial parameters $(\pi_k^{(0)}, \phi_l^{(0)})_{k=1}^K$.
- Iterate for $t = 1, 2, \ldots$:
  - **Expectation Step**:

$$q^{(t)}(z_i = k) := \frac{\pi_k^{(t-1)} f(x_i|\phi_k^{(t-1)})}{\sum_{j=1}^K \pi_j^{(t-1)} f(x_i|\phi_j^{(t-1)})} = \mathbb{E}_{p(z_i|x_i,\theta^{(t-1)})}[\mathbb{1}(z_i = k)]$$

  - **Maximization Step**:

$$\pi_k^{(t)} = \frac{\sum_{i=1}^n q^{(t)}(z_i = k)}{n} \qquad \phi_k^{(t)} = \frac{\sum_{i=1}^n q^{(t)}(z_i = k) x_i}{\sum_{i=1}^n q^{(t)}(z_i = k)}$$

- Each step increases the log likelihood:

$$\ell(\theta^{(t-1)}) = \mathcal{F}(\theta^{(t-1)}, q^{(t)}) \le \mathcal{F}(\theta^{(t)}, q^{(t)}) \le \mathcal{F}(\theta^{(t)}, q^{(t+1)}) = \ell(\theta^{(t)}).$$

- Additional assumption, that $\nabla_\theta^2 \mathcal{F}(\theta^{(t)}, q^{(t)})$ are negative definite with eigenvalues $< -\epsilon < 0$, implies that $\theta^{(t)} \to \theta^*$ where θ^* is a local MLE.

Notes on Probabilistic Approach and EM Algorithm

Some good things:

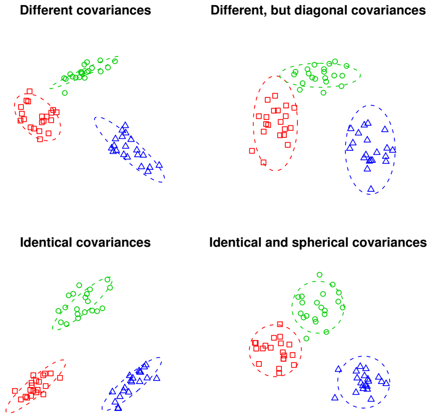
- Guaranteed convergence to locally optimal parameters.
- Formal reasoning of uncertainties, using both Bayes Theorem and maximum likelihood theory.
- Rich language of probability theory to express a wide range of generative models, and straightforward derivation of algorithms for ML estimation.

Some bad things:

- Can get stuck in local minima so multiple starts are recommended.
- Slower and more expensive than K-means.
- Choice of K still problematic, but rich array of methods for model selection comes to rescue.

Flexible Gaussian Mixture Models

- We can allow each cluster to have its own mean and covariance structure allows greater flexibility in the model.



Probabilistic PCA

- A probabilistic model related to PCA has the following generative model: for $i = 1, 2, \ldots, n$:
  - Let $k < n, p$ be given.
  - Let Y_i be a k-dimensional normally distributed random variable with 0 mean and identity covariance:

$$Y_i \sim \mathcal{N}(0, I_k)$$

  - We model the distribution of the ith data point given Y_i as a p-dimensional normal:

$$X_i \sim \mathcal{N}(\mu + LY_i, \sigma^2 I)$$

    where the parameters are a vector $\mu \in \mathbb{R}^p$, a matrix $L \in \mathbb{R}^{p \times k}$ and $\sigma^2 > 0$.
- EM algorithm can be used for ML estimation, but PCA can more directly give a MLE (note this is not unique).
- Let $\lambda_1 \ge \cdots \ge \lambda_p$ be the eigenvalues of the sample covariance and let $V \in \mathbb{R}^{p \times k}$ have columns given by the eigenvectors of the top k eigenvalues. Let $R \in \mathbb{R}^{k \times k}$ be orthogonal. Then a MLE is:

$$\mu^{\text{MLE}} = \bar{x} \qquad (\sigma^2)^{\text{MLE}} = \frac{1}{p-k} \sum_{j=k+1}^p \lambda_j$$

$$L^{\text{MLE}} = V \operatorname{diag}((\lambda_1 - (\sigma^2)^{\text{MLE}})^{\frac{1}{2}}, \ldots, (\lambda_k - (\sigma^2)^{\text{MLE}})^{\frac{1}{2}}) R$$

Mixture of Probabilistic PCAs

- We have learnt two types of unsupervised learning techniques:
  - Dimensionality reduction, e.g. PCA, MDS, Isomap.
  - Clustering, e.g. K-means, linkage and mixture models.
- Probabilistic models allow us to construct more complex models from simpler pieces.
- Mixture of probabilistic PCAs allows both clustering and dimensionality reduction at the same time.

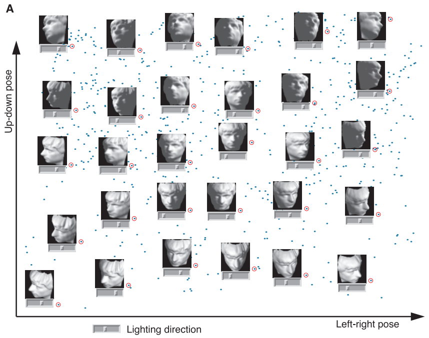
$$\begin{aligned} Z_i &\sim \text{Discrete}(\pi_1, \ldots, \pi_K) \\ Y_i &\sim \mathcal{N}(0, I_d) \\ X_i | Z_i = k, Y_i = y_i &\sim \mathcal{N}(\mu_k + L y_i, \sigma^2 I_p) \end{aligned}$$

- Allows flexible modelling of covariance structure without using too many parameters.

Ghahramani and Hinton 1996

Mixture of Probabilistic PCAs

- PCA can reconstruct x given low dimensional embedding z, but is linear.
- Isomap is non-linear, but cannot reconstruct x given any z.



- We can learn a probabilistic mapping between the k-dimensional Isomap embedding space and the p-dimensional data space.
- Demo: [Using LLE instead of Isomap, and Mixture of factor analysers instead of Mixture of PPCAs.]

Teh and Roweis 2002

Further Readings—Unsupervised Learning

- Hastie et al, Chapter 14.
- James et al, Chapter 10.
- Venables and Ripley, Chapter 11.
- Tukey, John W. (1980). We need both exploratory and confirmatory. The American Statistician 34 (1): 23-25.

Supervised Learning

Unsupervised learning:

- To “extract structure” and postulate hypotheses about data generating process from observations $x_1, \ldots, x_n$.
- Visualize, summarize and compress data.

We have seen how response or grouping variables are used to validate the usefulness of the extracted structure.

Supervised learning:

- In addition to the n observations of X, we also have a response variable $Y \in \mathcal{Y}$.
- Techniques for predicting Y given X.
  - Classification: discrete responses, e.g. $\mathcal{Y} = \{+1, -1\}$ or $\{1, \ldots, K\}$.
  - Regression: a numerical value is observed and $\mathcal{Y} = \mathbb{R}$.

Given training data (x_i, y_i), $i = 1, \ldots, n$, the goal is to accurately predict the class or response Y on new observations of X.

Regression Example: Boston Housing

The original data are 506 observations on 13 variables X; medv being the response variable Y.

crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000 sq.ft
indus	proportion of non-retail business acres per town
chas	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
nox	nitric oxides concentration (parts per 10 million)
rm	average number of rooms per dwelling
age	proportion of owner-occupied units built prior to 1940
dis	weighted distances to five Boston employment centers
rad	index of accessibility to radial highways
tax	full-value property-tax rate per USD 10,000
ptratio	pupil-teacher ratio by town
b	1000(B - 0.63)^2 where B is the proportion of blacks by t
lstat	percentage of lower status of the population
medv	median value of owner-occupied homes in USD 1000's

Regression Example: Boston Housing

```
> str(X)
'data.frame':   506 obs. of  13 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.8
 $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524
 $ rm     : num  6.58 6.42 7.18 7.00 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...

> str(Y)
num[1:506] 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

Goal: predict median house price $\hat{Y}(X)$, given 13 predictor variables X of a new district.

Classification Example: Lymphoma

We have gene expression measurements X of $n = 62$ patients for $p = 4026$ genes. For each patient, Y denotes one of two subtypes of cancer. Goal: predict cancer subtype $\hat{Y}(X) \in \{0, 1\}$, given gene expressions of a new patient.

```
> str(X)
'data.frame':   62 obs. of  4026 variables:
 $ Gene 1   : num  -0.344 -1.188  0.520 -0.748 -0.868 ...
 $ Gene 2   : num  -0.953 -1.286  0.657 -1.328 -1.330 ...
 $ Gene 3   : num  -0.776 -0.588  0.409 -0.991 -1.517 ...
 $ Gene 4   : num  -0.474 -1.588  0.219  0.978 -1.604 ...
 $ Gene 5   : num  -1.896 -1.960 -1.695 -0.348 -0.595 ...
 $ Gene 6   : num  -2.075 -2.117  0.121 -0.800  0.651 ...
 $ Gene 7   : num  -1.8755 -1.8187  0.3175  0.3873  0.0414 ...
 $ Gene 8   : num  -1.539 -2.433 -0.337 -0.522 -0.668 ...
 $ Gene 9   : num  -0.604 -0.710 -1.269 -0.832  0.458 ...
 $ Gene 10  : num  -0.218 -0.487 -1.203 -0.919 -0.848 ...
 $ Gene 11  : num  -0.340  1.164  1.023  1.133 -0.541 ...
 $ Gene 12  : num  -0.531  0.488 -0.335  0.496 -0.358 ...

> str(Y)
 num [1:62] 0 0 0 1 0 0 1 0 0 0 ...
```

Decision Theory

- Suppose we made a prediction $\hat{Y} \in \mathcal{Y}$ based on observation of X.
- How good is the prediction? We can use a **loss function** $L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}^+$ to formalize the quality of the prediction.
- Typical loss functions:
  - **Misclassification loss** (or **0-1 loss**) for classification

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

  - **Squared loss** for regression

$$L(Y, \hat{Y}) = (Y - \hat{Y})^2.$$

- Alternative loss functions are often useful (later). For example, **weighted misclassification error** often appropriate. Or **log-likelihood loss** (sometimes shortened as **log loss**) $L(Y, \hat{p}) = -\log \hat{p}(Y)$, where $\hat{p}(k)$ is the estimated probability of class $k \in \mathcal{Y}$.

Decision Theory

- For a given loss function L, the **risk** R of a learner is given by the expected loss
$$R(\hat{Y}) = \mathbb{E}(L(Y, \hat{Y}(X))),$$
where the expectation is with respect to the true (unknown) joint distribution (X, Y).
- The risk is unknown, but we can estimate it by the **empirical risk**:

$$R(\hat{Y}) \approx R_n(\hat{Y}) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{Y}(x_i)).$$

The Bayes Classifier

- What is the optimal classifier if the joint distribution (X, Y) were known?
- The joint distribution f of X can be written as a mixture

$$f(X) = \sum_{k=1}^{K} f_k(X)\mathbb{P}(Y = k),$$

where, for $k = 1, \dots, K$,
  - the prior probabilities over classes are $P(Y = k) = \pi_k$
  - and distributions of X, conditional on $Y = k$, is $f_k(X)$.
- The **Bayes classifier** $\hat{Y}(X) \mapsto \{1, \dots, K\}$ is the one with minimum risk:

$$\begin{aligned} R(\hat{Y}) =& \mathbb{E}\left[L(Y, \hat{Y}(X))\right] = \mathbb{E}\left[\mathbb{E}[L(Y, \hat{Y}(x)) \middle| X = x]\right] \\ =& \int_{\mathcal{X}} \mathbb{E}\left[L(Y, \hat{Y}(x)) \middle| X = x\right] f(x)dx \end{aligned}$$

- The minimum risk attained by the Bayes classifier is called **Bayes risk**.
- Minimizing $\mathbb{E}[L(Y, \hat{Y}(x)) | X = x]$ separately for each x suffices.

The Bayes Classifier

- Consider the situation of the 0-1 loss.
- The risk simplifies to:

$$\begin{aligned} \mathbb{E}\left[L(Y, \hat{Y}(x)) \middle| X = x\right] =& \sum_{k=1}^{K} L(k, \hat{Y}(x))\mathbb{P}(Y = k|X = x) \\ =& 1 - \mathbb{P}(Y = \hat{Y}(x)|X = x) \end{aligned}$$

- The risk is minimized by choosing the class with the greatest posterior probability:

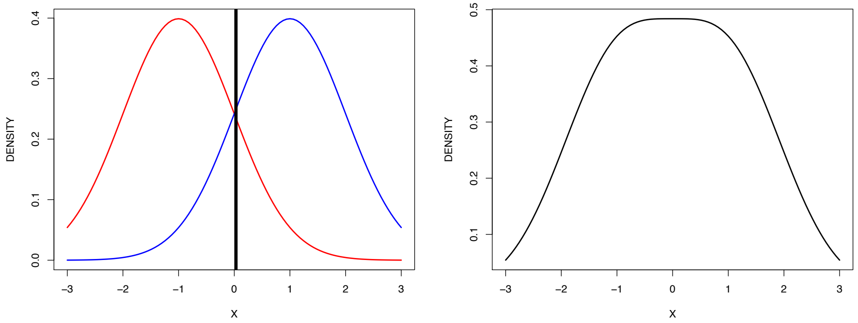
$$\begin{aligned} \hat{Y}(x) &= \operatorname*{arg\,max}_{k=1,\dots,K} \mathbb{P}(Y = k|X = x) = \operatorname*{arg\,max}_{k=1,\dots,K} \frac{\pi_k f_k(x)}{\sum_{k=1}^{K} \pi_k f_k(x)} \\ &= \operatorname*{arg\,max}_{k=1,\dots,K} \pi_k f_k(x). \end{aligned}$$

- The functions $x \mapsto \pi_k f_k(x)$ are called **discriminant functions**. The function with maximum value determines the predicted class of x.

The Bayes Classifier

A simple two Gaussians example: Suppose $X \sim \mathcal{N}(\mu_Y, 1)$, where $\mu_1 = -1$ and $\mu_2 = 1$ and assume equal priors $\pi_1 = \pi_2 = 1/2$.

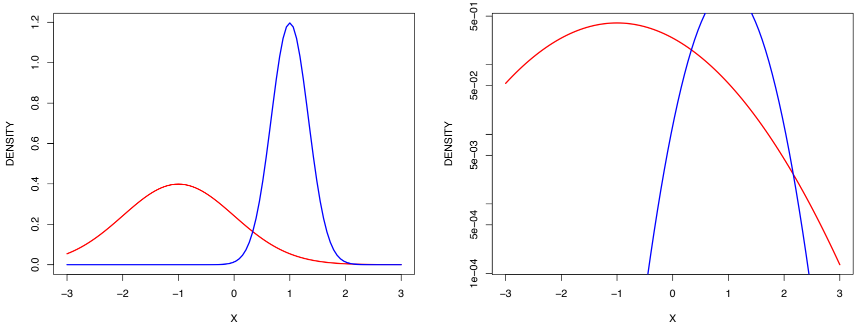
$$f_1(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x - (-1))^2}{2}\right) \quad \text{and} \quad f_2(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x - 1)^2}{2}\right).$$



Optimal classification is $\hat{Y}(x) = \operatorname*{arg\,max}_{k=1,\dots,K} \pi_k f_k(x) = \begin{cases} 1 & \text{if } x < 0, \\ 2 & \text{if } x \geq 0. \end{cases}$

The Bayes Classifier

How do you classify a new observation x if now the standard deviation is still 1 for class 1 but 1/3 for class 2?



Looking at density in a log-scale, optimal classification is class 2 if and only if $x \in [-0.39, 2.15]$.

Plug-in Classification

- The Bayes Classifier chooses the class with the greatest posterior probability

$$\hat{Y}(x) \quad = \quad \arg\max_{k=1,\dots,K} \pi_k f_k(x).$$

- Unfortunately, we usually know neither the conditional class probabilities nor the prior probabilities.
- We can estimate the joint distribution with:
  - estimates $\hat{\pi}_k$ for π_k and $k = 1, \dots, K$ and
  - estimates $\hat{f}_k(x)$ of conditional class densities,
- The **plug-in classifiers** chooses the class

$$\hat{Y}(x) = \arg\max_{k=1,\dots,K} \hat{\pi}_k \hat{f}_k(x).$$

- **Linear Discriminant Analysis** will be an example of plug-in classification.

Linear Discriminant Analysis

- LDA is the most well-known and simplest example of plug-in classification.
- Assume a multivariate Normal form for $f_k(x)$ for each class k:

$$X|Y = k \sim \mathcal{N}(\mu_k, \Sigma),$$

  - each class can have a **different mean** μ_k
  - but all classes share the **same covariance** Σ.
- For an observation x,

$$\begin{aligned}\log \mathbb{P}(Y = k|X = x) &= \kappa + \log \pi_k f_k(x) \\ &= \kappa + \log \pi_k - \frac{1}{2}(x - \mu_k)^\top \Sigma^{-1}(x - \mu_k)\end{aligned}$$

  The quantity $(x - \mu_k)^\top \Sigma^{-1}(x - \mu_k)$ is the square of the **Mahalanobis distance**. It gives the distance between x and μ_k in the metric given by Σ.
- If $\Sigma = I_p$ and $\pi_k = \frac{1}{K}$, $\hat{Y}(x)$ simply chooses the class k with the nearest (in the Euclidean sense) mean.

Linear Discriminant Analysis

- Expanding the **discriminant** $(x - \mu_k)^\top \Sigma^{-1}(x - \mu_k)$,

$$\begin{aligned}\log \mathbb{P}(Y = k|x) &= \kappa + \log(\pi_k) - \frac{1}{2}\left(\mu_k^\top \Sigma^{-1}\mu_k - 2\mu_k^\top \Sigma^{-1}x + x^\top \Sigma^{-1}x\right) \\ &= \kappa + \log(\pi_k) - \frac{1}{2}\mu_k^\top \Sigma^{-1}\mu_k + \mu_k^\top \Sigma^{-1}x\end{aligned}$$

- Setting $a_k = \log(\pi_k) - \frac{1}{2}\mu_k^\top \Sigma^{-1}\mu_k$ and $b_k = \Sigma^{-1}\mu_k$, we obtain

$$\log \mathbb{P}(Y = k|X = x) = \kappa + a_k + b_k^\top x$$

  i.e. a **linear** discriminant function.
- Consider choosing class k over k':

$$a_k + b_k^\top x > a_{k'} + b_{k'}^\top x \qquad \Leftrightarrow \qquad a_\star + b_\star^\top x > 0$$

  where $a_\star = a_k - a_{k'}$ and $b_\star = b_k - b_{k'}$.
- The Bayes classifier partitions $\mathcal{X}$ into regions with the same class predictions via **separating hyperplanes**.
- The Bayes classifier under these assumptions is more commonly known as the **LDA classifier**.

Parameter Estimation

- The final piece of the puzzle is to estimate the parameters of the LDA model.
- We can achieve this by maximum likelihood.
- EM algorithm is not needed here since the class variables y_i are observed.
- Let $n_k = \#\{j : y_j = k\}$ be the number of observations in class k.

$$\ell(\pi, (\mu_k), \Sigma) = \kappa + \sum_{k=1}^{K} \sum_{j:y_j=k} \log \pi_k - \frac{1}{2}\left(\log|\Sigma| + (x_j - \mu_k)^\top \Sigma^{-1}(x_j - \mu_k)\right)$$

Then:

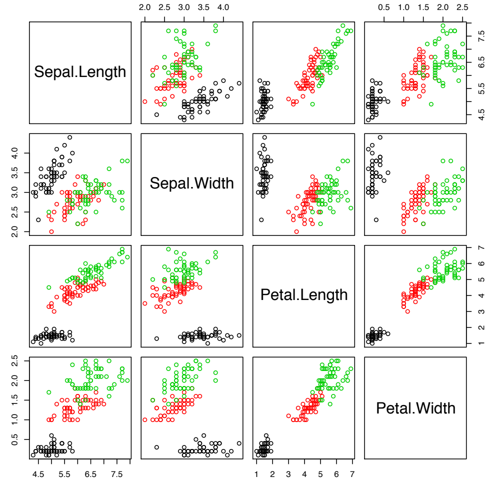
$$\hat{\pi}_k = \frac{n_k}{n} \qquad \hat{\mu}_k = \frac{1}{n_k} \sum_{j:y_j=k} x_j$$

$$\hat{\Sigma} = \frac{1}{n} \sum_{k=1}^{K} \sum_{j:y_j=k} (x_j - \hat{\mu}_k)(x_j - \hat{\mu}_k)^\top$$

- Note: the ML estimate of Σ is not unbiased. For an unbiased estimate we need to divide by $n - K$.

Iris Dataset

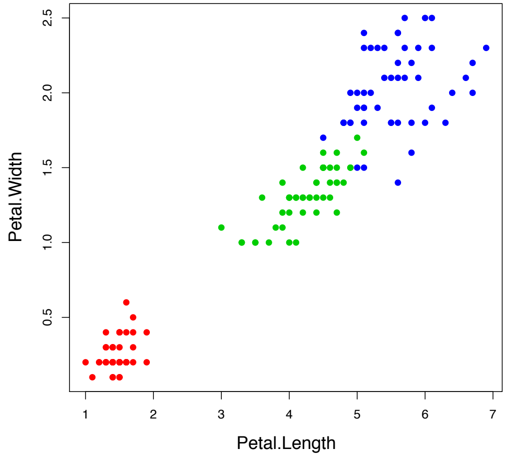
```
library(MASS)
data(iris)
##save class labels
ct <- rep(1:3,each=50)
##pairwise plot
pairs(iris[,1:4],col=ct)
```



Iris Dataset

Just focus on two predictor variables.

```
iris.data <- iris[,3:4]
plot(iris.data,col=ct+1,pch=20,cex=1.5,cex.lab=1.4)
```

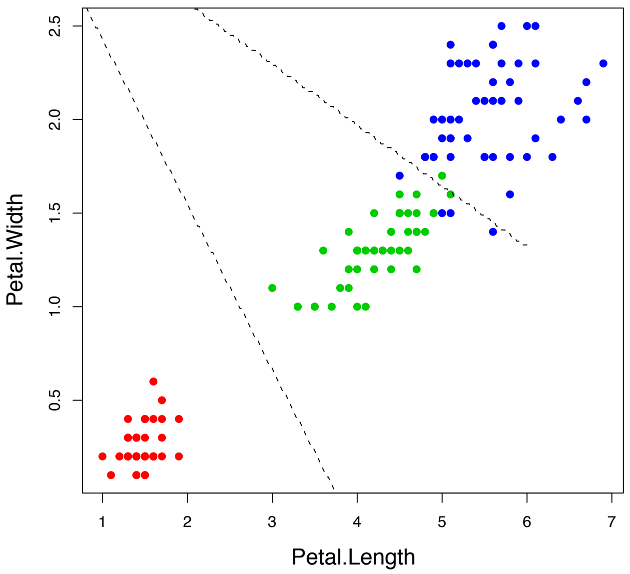


Iris Dataset

Computing and plotting the LDA boundaries.

```
##fit LDA
iris.lda <- lda(x=iris.data,grouping=ct)

##create a grid for our plotting surface
x <- seq(-6,6,0.02)
y <- seq(-4,4,0.02)
z <- as.matrix(expand.grid(x,y),0)
m <- length(x)
n <- length(y)

##classes are 1,2 and 3, so set contours at 1.5 and 2.5
iris.ldp <- predict(iris.lda,z)$class
contour(x,y,matrix(iris.ldp,m,n),
           levels=c(1.5,2.5), add=TRUE, d=FALSE, lty=2)
```

Iris Dataset



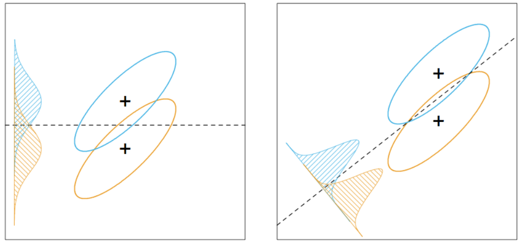
Fisher's Linear Discriminant Analysis

- In LDA, data vectors are classified based on Mahalanobis distance from cluster means, which lie on a $K-1$ affine subspace.
- In measuring these distances, directions orthogonal[5] to the subspace can be ignored.
- Projecting data vectors onto the subspace can be viewed as a dimensionality reduction technique that preserves discriminative information about $(y_i)_{i=1}^n$.
- As with PCA, we can visualize the structure in the data by choosing an appropriate basis for the subspace and projecting data onto it.
- Choose a basis by finding directions that are separate classes best.

---

[5] Orthogonality defined in terms of the inner product corresponding to Mahalanobis distance: $\langle x, y\rangle = x\Sigma^{-1}y$.

Fisher's Linear Discriminant Analysis



- Find a direction $v \in \mathbb{R}^p$ to maximize the variance ratio

$$\frac{v^\top B v}{v^\top \Sigma v}$$

where

$$\Sigma = \tfrac{1}{n}\textstyle\sum_{i=1}^n (x_i - \mu_{y_i})(x_i - \mu_{y_i})^\top \qquad \text{(within class covariance)}$$

$$B = \tfrac{1}{n}\textstyle\sum_{k=1}^K n_k(\mu_{y_i} - \bar{x})(\mu_{y_i} - \bar{x})^\top \qquad \text{(between class covariance)}$$

B has rank at most $K-1$.

Figure from Hastie et al.

Discriminant Coordinates

- To solve for the optimal v, we first reparameterize it as $u = \Sigma^{\frac{1}{2}} v$.

$$\frac{v^\top B v}{v^\top \Sigma v} = \frac{u^\top (\Sigma^{-\frac{1}{2}})^\top B \Sigma^{-\frac{1}{2}} u}{u^\top u} = \frac{u^\top B^* u}{u^\top u}$$

where $B^* = (\Sigma^{-\frac{1}{2}})^\top B \Sigma^{-\frac{1}{2}}$.

- The maximization over u is achieved by the first eigenvector u_1 of B^*.
- We also look at the remaining eigenvectors u_l associated to the non-zero eigenvalues and defined the **discriminant coordinates** as $v_l = \Sigma^{-\frac{1}{2}} u_l$.
- The v_l's span exactly the affine subspace spanned by $(\Sigma^{-1}\mu_k)_{k=1}^K$ (these vectors are given as the "linear discriminants" in the R-function `lda`).

Crabs Dataset

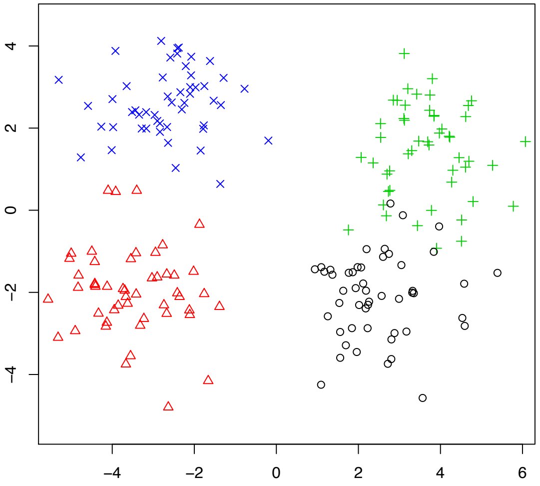
```
library(MASS)
data(crabs)

## numeric and text class labels
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)

## Projection on Fisher's linear discriminant directions
print(cb.lda <- lda(log(crabs[,4:8]),ct))
```

Crabs Dataset

```
> > > > > > > > > Call:
lda(log(crabs[, 4:8]), ct)

Prior probabilities of groups:
   0    1    2    3
0.25 0.25 0.25 0.25

Group means:
        FL       RW       CL       CW       BD
0 2.564985 2.475174 3.312685 3.462327 2.441351
1 2.852455 2.683831 3.529370 3.649555 2.733273
2 2.672724 2.443774 3.437968 3.578077 2.560806
3 2.787885 2.489921 3.490431 3.589426 2.701580

Coefficients of linear discriminants:
          LD1        LD2        LD3
FL -31.217207  -2.851488  25.719750
RW  -9.485303 -24.652581  -6.067361
CL  -9.822169  38.578804 -31.679288
CW  65.950295 -21.375951  30.600428
BD -17.998493   6.002432 -14.541487

Proportion of trace:
   LD1    LD2    LD3
0.6891 0.3018 0.0091
```

Crabs Dataset

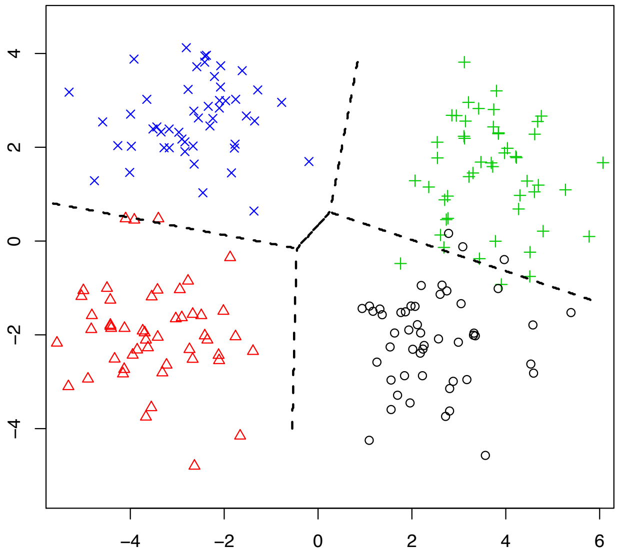
```
cb.ldp <- predict(cb.lda)$x[,1:2]
eqscplot(cb.ldp,pch=ct+1,col=ct+1)
```



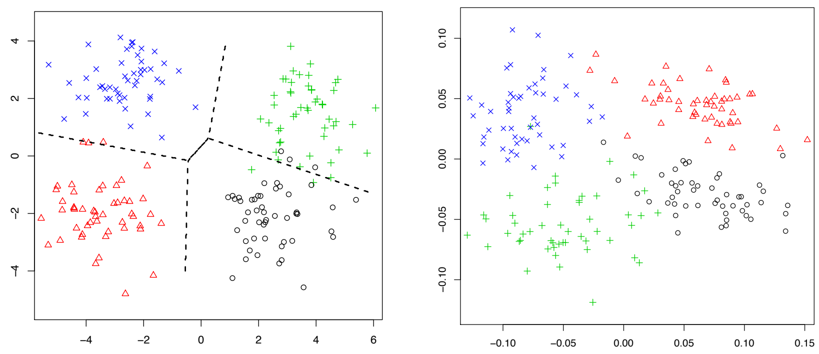
Crabs Dataset

```
## display the decision boundaries
## take a lattice of points in LD-space
x <- seq(-6,6,0.02)
y <- seq(-4,4,0.02)
z <- as.matrix(expand.grid(x,y))
m <- length(x)
n <- length(y)

## perform LDA on first two discriminant directions
cb.lda <- lda(cb.ldp,ct)
## predict onto the grid
cb.ldpp <- predict(cb.lda,z)$class

## classes are 0,1,2 and 3 so set contours
## at 0.5,1.5 and 2.5
contour(x,y,matrix(cb.ldpp,m,n),
            levels=c(0.5,2.5),
            add=TRUE,d=FALSE,lty=2,lwd=2)
```

Crabs Dataset



Crabs Dataset



LDA separates the groups better.

Naïve Bayes

- Assume we are interested in classifying documents; e.g. scientific articles or emails.
- A basic but standard model for text classification consists of considering a pre-specified dictionary of p words (including say physics, calculus.... or dollars, sex etc.) and summarizing each document i by a binary vector x_i where

$$x_{ij} = \begin{cases} 1 & \text{if word } j \text{ is present in document} \\ 0 & \text{otherwise.} \end{cases}$$

- To implement a probabilistic classifier, we need to model $f_k(x|\phi_k)$ for each class $k = 1, ..., K$.

Naïve Bayes

- A Naïve Bayes approach ignores feature correlations and assumes $f_k(x) = f(x|\phi_k)$ where

$$f_k(x_i) = f(x_i|\phi_k) = \prod_{j=1}^{p} (\phi_{kj})^{x_{ij}} (1 - \phi_{kj})^{1-x_{ij}}$$

- Given dataset, the MLE is easily obtained

$$\hat{\pi}_k = \frac{n_k}{n} \qquad \hat{\phi}_{kj} = \frac{\sum_{i:y_i=k} x_{ij}}{n_k}$$

- One problem: if word j did not appear in documents labelled as class k then $\hat{\phi}_{kj} = 0$ and

$$\mathbb{P}(Y = k|X = x \text{ with } j\text{th entry equal to } 1) = 0$$

  i.e. we will never attribute a new document containing word j to class k.
- This problem is called **overfitting**, and is a major concern in modelling high-dimensional datasets common in machine learning.

Generative and Discriminative Learning

- **Generative learning**: find parameters that **explains all the data.**

$$\theta^* = \operatorname*{argmax}_\theta \sum_{i=1}^n \log p(x_i, y_i|\theta)$$

Examples: LDA, Naïve Bayes.

  - Makes use of all the data.
  - Flexible framework, can incorporate other tasks.
  - Stronger modelling assumptions.

- **Discriminative learning**: find parameters that help to **predict relevant data.**

$$\theta^* = \operatorname*{argmax}_\theta \sum_{i=1}^n \log p(y_i|x_i, \theta) \qquad \text{or} \qquad f^* = \operatorname*{argmin}_f \sum_{i=1}^n L(y_i, f(X_i))$$

Examples: linear and logistic regression, rest of the course.

  - Learns to perform better on the given task.
  - Weaker modelling assumptions.
  - Can overfitting more easily.

Statistical Learning Theory

- We work with a joint distribution $p^*(X, Y)$ over data vectors and labels.
- A learning algorithm constructs a function $f(X)$ which predicts the label of X.
- Given a loss function L, the risk R of $f(X)$ is

$$R(f) = \mathbb{E}_{X,Y}[L(Y, f(X))]$$

For classification, the best function $f^*(X)$ is the Bayes classifier, achieving the minimum risk (Bayes risk).

- Hypothesis space $\mathcal{H}$ is the space of functions under consideration.
- Find best function minimizing the risk:

$$\operatorname*{argmin}_{f\in\mathcal{H}} \mathbb{E}_{X,Y}[L(Y, f(X))]$$

  - **Empirical Risk Minimization**: minimize the empirical risk instead, since we typically do not know $p^*(X, Y)$.
  - **Regularization**: Large hypothesis spaces can lead to overfitting,

$$\operatorname*{argmin}_{f\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^n L(y_i, f(x_i)) + \lambda\|f\|_{\mathcal{H}}$$

Training and Test Performance

- **Training error** is the empirical risk

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

For 0-1 loss in classification, this is the misclassification error on the training data, **which were used in learning** $f(x)$.

- **Test error** is the empirical risk on **new, previously unseen**, observations

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

which were NOT used in learning.

- Test error is a much better gauge of how well learned function **generalizes** to new data.
- The test error is in general larger than the training error.

Logistic Regression

- Assume we have two classes $\{+1, -1\}$.
- Recall that the discriminant functions in LDA are linear. Assuming that data vectors in class k is modelled as $\mathcal{N}(\mu_k, \Sigma)$, choosing class $+1$ over -1 involves:

$$a_{+1} + b_{+1}^\top x > a_{-1} + b_{-1}^\top x \quad \Leftrightarrow \quad (a_{+1} - a_{-1}) + (b_{+1} - b_{-1})^\top x > 0$$

- If we care about minimizing classification errors, we can try to find a, b to minimize directly the average misclassification error (empirical risk associated with 0-1 loss):

$$\begin{aligned} &\operatorname*{argmin}_{a,b} \frac{1}{n}\sum_{i=1}^n \begin{cases} 0 & \text{if } y_i = \operatorname{sign}(a + b^\top x) \\ 1 & \text{otherwise} \end{cases} \\ = &\operatorname*{argmin}_{a,b} \frac{1}{n}\sum_{i=1}^n \frac{1}{2} - \frac{1}{2}\operatorname{sign}(y_i(a + b^\top x)) \end{aligned}$$

- An example of **Empirical Risk Minimization**. Unfortunately not typically possible to solve...

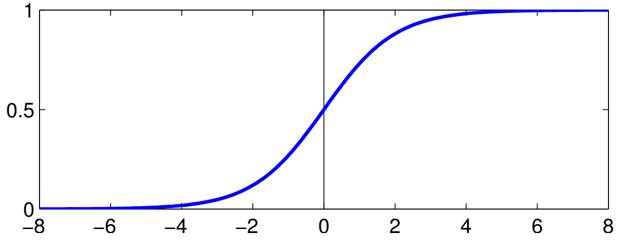
Logistic Regression

- **Logistic regression** replaces the 0-1 loss with the log loss.
- A model parameterizing the conditional distribution of labels given data vectors:

$$p(Y=1|X=x) = \frac{1}{1+\exp(-(a+b^\top x))} =: s(a+b^\top x)$$

$$p(Y=-1|X=x) = \frac{1}{1+\exp(+(a+b^\top x))} = s(-a-b^\top x)$$

where $s(\cdot)$ is the **logistic function**



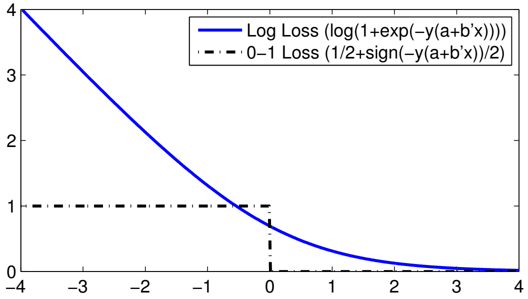
Logistic Regression

- Consider maximizing the **conditional log likelihood**:

$$\ell(a,b) = \sum_{i=1}^n \log p(Y=y_i|X=x_i) = \sum_{i=1}^n -\log(1+\exp(-y_i(a+b^\top x_i)))$$

- Equivalent to minimizing the empirical risk associated with the **log loss**:

$$R^{\text{emp}}_{\text{log}} = \frac{1}{n}\sum_{i=1}^n \log(1+\exp(-y_i(a+b^\top x_i)))$$



Logistic Regression

- Not possible to find optimal a, b analytically.
- For simplicitiy, absorb a as an entry in b by appending '1' into x vector.
- Objective function:

$$R^{\text{emp}}_{\text{log}} = \frac{1}{n}\sum_{i=1}^n -\log s(y_i x_i^\top b)$$

Logistic Function

$$s(-z) = 1 - s(z)$$

$$\nabla_z s(z) = s(z)s(-z)$$

$$\nabla_z \log s(z) = s(-z)$$

$$\nabla_z^2 \log s(z) = -s(z)s(-z)$$

- Differentiate wrt b:

$$\nabla_b R^{\text{emp}}_{\text{log}} = \frac{1}{n}\sum_{i=1}^n -s(-y_i x_i^\top b) y_i x_i = \frac{1}{n}\sum_{i=1}^n -((.5+.5y_i) - s(x_i^\top b))x_i$$

$$\nabla_b^2 R^{\text{emp}}_{\text{log}} = \frac{1}{n}\sum_{i=1}^n s(y_i x_i^\top b)s(-y_i x_i^\top b) x_i x_i^\top$$

Logistic Regression

- Second derivative is positive-definite: objective function is **convex** and there is **a single unique global minimum**.
- Many different algorithms can find optimal b, e.g.:
  - Gradient descent:

$$b^{\text{new}} = b + \epsilon \frac{1}{n}\sum_{i=1}^n s(-y_i x_i^\top b) y_i x_i$$

  - Stochastic gradient descent:

$$b^{\text{new}} = b + \epsilon_t \frac{1}{|I(t)|}\sum_{i\in I(t)} s(-y_i x_i^\top b) y_i x_i$$

    where $I(t)$ is a subset of the data at iteration t, and $\epsilon_t \to 0$ slowly ($\sum_t \epsilon_t = \infty, \sum_t \epsilon_t^2 < \infty$).
  - Newton-Raphson:

$$b^{\text{new}} = b - (\nabla_b^2 R^{\text{emp}}_{\text{log}})^{-1} \nabla_b R^{\text{emp}}_{\text{log}}$$

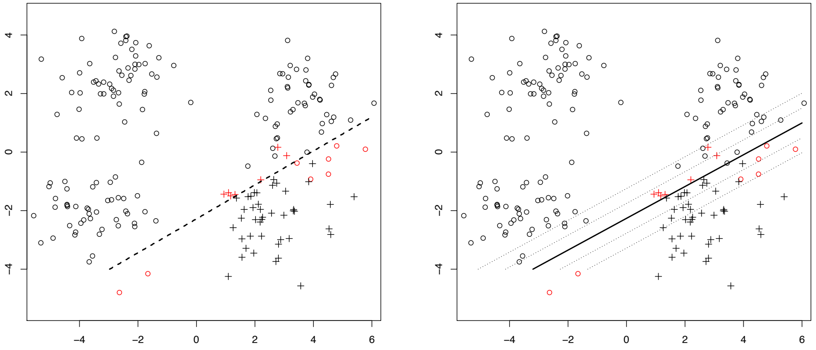
    This is also called **iterative reweighted least squares**.
  - Conjugate gradient, LBFGS and other methods from numerical analysis.

Logistic Regression

Properties of logistic regression:

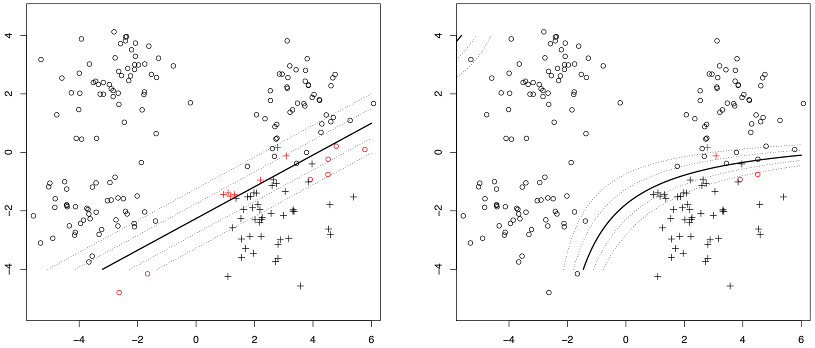
- Makes less modelling assumptions than LDA and naïve Bayes.
- Models only the conditional distribution of labels, not the marginal distribution of X.
- A linear method: decision boundary is a separating hyperplane.
- Logistic regression can be made **non-linear** by applying a non-linear transformation $X \mapsto \phi(X)$.
- Logistic regression is a simple example of a generalised linear model (GLM). Much statistical theory:
  - assessment of fit via deviance and plots,
  - interpretation of entries of b as **odds-ratios**,
  - fitting categorical data (sometimes called **multinomial logistic regression**),
  - well founded approaches to removing insignificant features (drop-in deviance test, Wald test),

Crab Dataset



Comparing LDA and logistic regression.

Crab Dataset



Comparing logistic regression with and without quadratic interactions.

Crab Dataset

```
library(MASS)
## load crabs data
data(crabs)
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)
## project into first two LD
cb.lda <- lda(log(crabs[,4:8]),ct)
cb.ldp <- predict(cb.lda)
x <- cb.ldp$x[,1:2]
y <- as.numeric(ct==0)
eqscplot(x,pch=2*y+1,col=y+1)

## visualize decision boundary
gx1 <- seq(-6,6,.02)
gx2 <- seq(-4,4,.02)
gx <- as.matrix(expand.grid(gx1,gx2))
gm <- length(gx1)
gn <- length(gx2)
gdf <- data.frame(LD1=gx[,1],LD2=gx[,2])

lda <- lda(x,y)
y.lda <- predict(lda,x)$class
eqscplot(x,pch=2*y+1,col=2-as.numeric(y==y.lda))
y.lda.grid <- predict(lda,gdf)$class
contour(gx1,gx2,matrix(y.lda.grid,gm,gn),
   levels=c(0.5), add=TRUE,d=FALSE,lty=2,lwd=2)
```

Crab Dataset

```
## logistic regression
xdf <- data.frame(x)
logreg <- glm(y ~ LD1 + LD2, data=xdf, family=binomial)
y.lr <- predict(logreg,type="response")
eqscplot(x,pch=2*y+1,col=2-as.numeric(y==(y.lr>.5)))
y.lr.grid <- predict(logreg,newdata=gdf,type="response")
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
   levels=c(.1,.25,.75,.9), add=TRUE,d=FALSE,lty=3,lwd=1)
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
   levels=c(.5), add=TRUE,d=FALSE,lty=1,lwd=2)
```

```
## logistic regression with quadratic interactions
logreg <- glm(y ~ (LD1 + LD2)^2, data=xdf, family=binomial)
y.lr <- predict(logreg,type="response")
eqscplot(x,pch=2*y+1,col=2-as.numeric(y==(y.lr>.5)))
y.lr.grid <- predict(logreg,newdata=gdf,type="response")
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
   levels=c(.1,.25,.75,.9), add=TRUE,d=FALSE,lty=3,lwd=1)
contour(gx1,gx2,matrix(y.lr.grid,gm,gn),
   levels=c(.5), add=TRUE,d=FALSE,lty=1,lwd=2)
```

Spam Dataset

```
> library(kernlab)
> data(spam)
> dim(spam)
[1] 4601   58

> spam[1:2,]
  make address  all num3d  our over remove internet order mail receive wil
1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00     0 0.00    0.00 0.6
2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07     0 0.94    0.21 0.7
  people report addresses free business email  you credit your font num000
1   0.00   0.00      0.00 0.32     0.00  1.29 1.93      0 0.96    0   0.00
2   0.65   0.21      0.14 0.14     0.07  0.28 3.47      0 1.59    0   0.43
  money hp hpl george num650 lab labs telnet num857 data num415 num85
1  0.00  0   0      0      0   0    0      0      0    0      0     0
2  0.43  0   0      0      0   0    0      0      0    0      0     0
  technology num1999 parts pm direct cs meeting original project re edu ta
1          0    0.00     0  0      0  0       0        0       0  0   0 0
2          0    0.07     0  0      0  0       0        0       0  0   0 0
  conference charSemicolon charRoundbracket charSquarebracket charExclamat
1          0             0            0.000                 0       0.778
2          0             0            0.132                 0       0.372
  charDollar charHash capitalAve capitalLong capitalTotal type
1       0.00    0.000      3.756          61          278 spam
2       0.18    0.048      5.114         101         1028 spam
```

Spam Dataset

Use logistic regression to predict spam/not spam.

```
library(kernlab)
data(spam)

## let Y=0 be non-spam and Y=1 be spam.
Y <- as.numeric(spam[, ncol(spam)])-1
X <- spam[ ,-ncol(spam)]

gl <- glm(Y ~ ., data=X,family=binomial)
```

Which predictor variables seem to be important? Can for example check which ones are significant in the GLM.

```
> summary(gl)
Call:
glm(formula = Y ~ ., family = binomial, data = X)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-4.127e+00  -2.030e-01  -1.967e-06   1.140e-01   5.364e+00
```

Spam Dataset

```
Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)       -1.569e+00  1.420e-01 -11.044  < 2e-16 ***
make              -3.895e-01  2.315e-01  -1.683 0.092388 .
address           -1.458e-01  6.928e-02  -2.104 0.035362 *
all                1.141e-01  1.103e-01   1.035 0.300759
num3d              2.252e+00  1.507e+00   1.494 0.135168
our                5.624e-01  1.018e-01   5.524 3.31e-08 ***
over               8.830e-01  2.498e-01   3.534 0.000409 ***
remove             2.279e+00  3.328e-01   6.846 7.57e-12 ***
internet           5.696e-01  1.682e-01   3.387 0.000707 ***
order              7.343e-01  2.849e-01   2.577 0.009958 **
mail               1.275e-01  7.262e-02   1.755 0.079230 .
receive           -2.557e-01  2.979e-01  -0.858 0.390655
will              -1.383e-01  7.405e-02  -1.868 0.061773 .
people            -7.961e-02  2.303e-01  -0.346 0.729557
report             1.447e-01  1.364e-01   1.061 0.288855
addresses          1.236e+00  7.254e-01   1.704 0.088370 .
business           9.599e-01  2.251e-01   4.264 2.01e-05 ***
email              1.203e-01  1.172e-01   1.027 0.304533
you                8.131e-02  3.505e-02   2.320 0.020334 *
credit             1.047e+00  5.383e-01   1.946 0.051675 .
```

Spam Dataset

```
your                2.419e-01  5.243e-02   4.615 3.94e-06 ***
font                2.013e-01  1.627e-01   1.238 0.215838
num000              2.245e+00  4.714e-01   4.762 1.91e-06 ***
money               4.264e-01  1.621e-01   2.630 0.008535 **
hp                 -1.920e+00  3.128e-01  -6.139 8.31e-10 ***
hpl                -1.040e+00  4.396e-01  -2.366 0.017966 *
george             -1.177e+01  2.113e+00  -5.569 2.57e-08 ***
num650              4.454e-01  1.991e-01   2.237 0.025255 *
lab                -2.486e+00  1.502e+00  -1.656 0.097744 .
labs               -3.299e-01  3.137e-01  -1.052 0.292972
telnet             -1.702e-01  4.815e-01  -0.353 0.723742
num857              2.549e+00  3.283e+00   0.776 0.437566
data               -7.383e-01  3.117e-01  -2.369 0.017842 *
num415              6.679e-01  1.601e+00   0.417 0.676490
num85              -2.055e+00  7.883e-01  -2.607 0.009124 **
technology          9.237e-01  3.091e-01   2.989 0.002803 **
num1999             4.651e-02  1.754e-01   0.265 0.790819
parts              -5.968e-01  4.232e-01  -1.410 0.158473
pm                 -8.650e-01  3.828e-01  -2.260 0.023844 *
direct             -3.046e-01  3.636e-01  -0.838 0.402215
cs                 -4.505e+01  2.660e+01  -1.694 0.090333 .
meeting            -2.689e+00  8.384e-01  -3.207 0.001342 **
original           -1.247e+00  8.064e-01  -1.547 0.121978
project            -1.573e+00  5.292e-01  -2.973 0.002953 **
re                 -7.923e-01  1.556e-01  -5.091 3.56e-07 ***
```

Spam Dataset

```
edu                -1.459e+00  2.686e-01  -5.434 5.52e-08 ***
table              -2.326e+00  1.659e+00  -1.402 0.160958
conference         -4.016e+00  1.611e+00  -2.493 0.012672 *
charSemicolon      -1.291e+00  4.422e-01  -2.920 0.003503 **
charRoundbracket   -1.881e-01  2.494e-01  -0.754 0.450663
charSquarebracket -6.574e-01  8.383e-01  -0.784 0.432914
charExclamation     3.472e-01  8.926e-02   3.890 0.000100 ***
charDollar          5.336e+00  7.064e-01   7.553 4.24e-14 ***
charHash            2.403e+00  1.113e+00   2.159 0.030883 *
capitalAve          1.199e-02  1.884e-02   0.636 0.524509
capitalLong         9.118e-03  2.521e-03   3.618 0.000297 ***
capitalTotal        8.437e-04  2.251e-04   3.747 0.000179 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6170.2  on 4600  degrees of freedom
Residual deviance: 1815.8  on 4543  degrees of freedom
AIC: 1931.8

Number of Fisher Scoring iterations: 13
```

Spam Dataset

How good is the classification?

```
> proba <- predict(gl,type="response")
> predicted_spam <- as.numeric( proba>0.5)
> table(predicted_spam,Y)
              Y
predicted_spam    0    1
             0 2666  194
             1  122 1619

> predicted_spam <- as.numeric( proba>0.99)
> table(predicted_spam,Y)
              Y
predicted_spam    0    1
             0 2776 1095
             1   12  718
```

So out of 730 emails marked as spam, 12 were actually not spam. Advantage of a probabilistic approach: probabilities give interpretable confidence to predictions.

Spam Dataset

Success rate is calculated on the same data that the GLM is trained on! Separate in training and test set.

```
n <- length(Y)
i <- sample( rep(c(TRUE,FALSE),each=n/2),round(n) ,replace=FALSE )
train <- (1:n)[i]
test  <- (1:n)[!i]
```

Fit only on training set and predict on both training and test set.

```
gl <- glm(Y[train] ~ ., data=X[train,],family=binomial)

proba_train <- predict(gl,newdata=X[train,],type="response")
proba_test  <- predict(gl,newdata=X[test,],type="response")

predicted_spam_train <- as.numeric(proba_train > 0.95)
predicted_spam_test  <- as.numeric(proba_test > 0.95)
```

Spam Dataset

Results for training and test set:

```
> table(predicted_spam_train, Y[train])
predicted_spam_train    0    1
                   0 1403  354
                   1   11  567


> table(predicted_spam_test, Y[test])
predicted_spam_test    0    1
                  0 1346  351
                  1   28  541
```

It is no coincidence that test performance is worse than training performance.

Spam Dataset

Compare with LDA.

```
library(MASS)
lda_res <- lda(x=X[train,],grouping=Y[train])

proba_lda <- predict(lda_res,newdata=X[test,])$posterior[,2]
predicted_spam_lda <- as.numeric(proba_lda > 0.95)

> table(predicted_spam_test, Y[test])
predicted_spam_test    0    1
                  0 1346  351
                  1   28  541


> table(predicted_spam_lda, Y[test])
predicted_spam_lda    0    1
                 0 1364  533
                 1   10  359
```

It seems as if LDA beats logistic regression here, but would need to adjust decision threshold to get proper comparison. Use **ROC curves**.

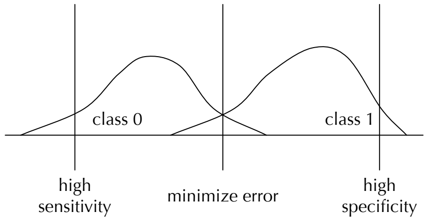
Performance Measures

- **Confusion matrix**:

True	state	0	1
Prediction	0	**# true negative**	**# false negative**
	1	**# false positive**	**# true positive**

  - **Accuracy**: $(TP + TN)/(TP + TN + FP + FN)$.
  - **Error rate**: $(FP + FN)/(TP + TN + FP + FN)$.
  - **Sensitivity (true positive rate)**: $TP/(TP + FN)$.
  - **Specificity (true negative rate)**: $TN/(TN + FP)$.
  - **Precision**: $TP/(TP + FP)$.
  - **Recall**: $TP/(TP + FN)$.
  - **F1**: harmonic mean of precision and recall.

- As we vary the prediction threshold c from 0 to 1:
  - Specificity varies from 0 to 1.
  - Sensitivity goes from 1 to 0.



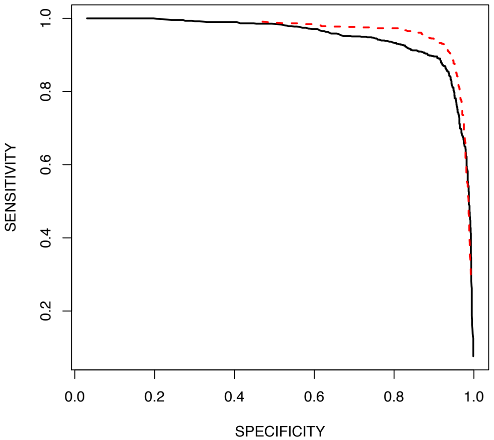
ROC Curves

ROC curve plots sensitivity versus specificity as threshold varies.

```
cvec <- seq(0.001,0.999,length=1000)
specif <- numeric(length(cvec))
sensit <- numeric(length(cvec))

for (cc in 1:length(cvec)){
  sensit[cc] <- sum( proba_lda> cvec[cc] & Y[test]==1)/sum(Y[test]==1)
  specif[cc] <- sum( proba_lda<=cvec[cc] & Y[test]==0)/sum(Y[test]==0)
}
plot(specif,sensit,xlab="SPECIFICITY",ylab="SENSITIVITY",type="l",lwd=2)
```

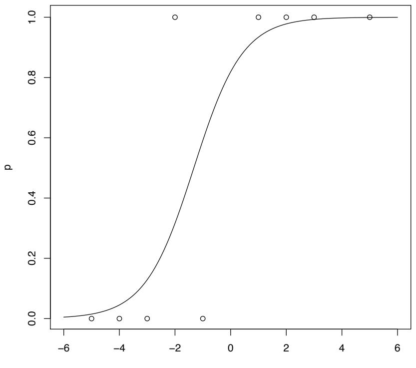
ROC Curves

ROC curve for LDA and logistic regression classification of spam dataset.
LDA = unbroken black line; LR = broken red line.



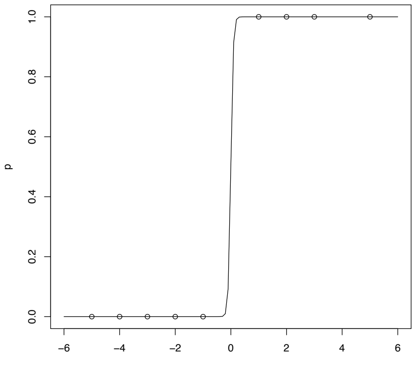
Obvious now that LR is better for this dataset than LDA, contrary to the first impression.

Overfitting in Logistic Regression



```
dx <- c(-5,-4,-3,-2,-1,1,2,3,5)
d <- data.frame(dx)
x <- seq(-6,6,.1)
y <- c(0,0,0,1,0,1,1,1,1)
lr <- glm(y ~ ., data=d,family=binomial)
p <- predict(lr,newdata=data.frame(dx=x),type="response")
plot(x,p,type="l")
points(dx,y)
```

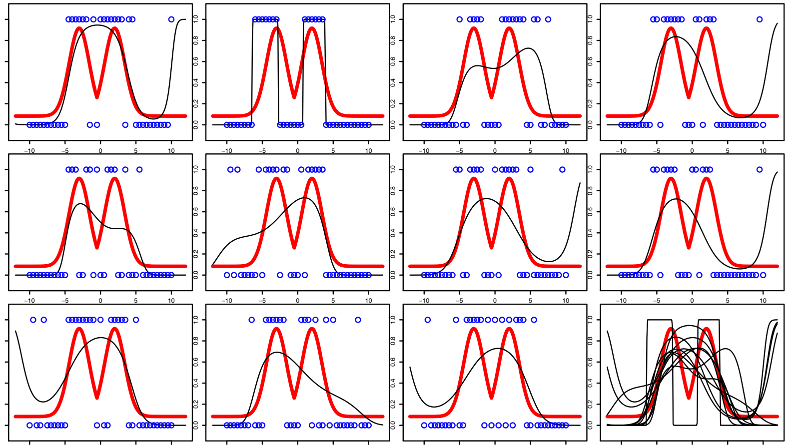
Overfitting in Logistic Regression



```
dx <- c(-5,-4,-3,-2,-1,1,2,3,5)
d <- data.frame(dx)
x <- seq(-6,6,.1)
y <- c(0,0,0,0,0,1,1,1,1)
lr <- glm(y ~ ., data=d,family=binomial)
p <- predict(lr,newdata=data.frame(dx=grid),type="response")
plot(x,p,type="l")
points(dx,y)
```

Demo on Overfitting in Logistic Regression

True conditional probabilities in Red. Blue circles are training data, Black curve is predicted conditional probability. 11 datasets are sampled from true distribution and used to learn a logistic regression model with non-linear features $\phi(x) = (1, x, x^2, \ldots, x^{p-1})$.



Demo on Overfitting in Logistic Regression

```
## true conditional probabilities
truep <- function(x) {
  return((pmax(exp(-(x-2)^2/4),exp(-(x+3)^2/4))+.1)/1.2)
}
## features are {x^i}
phi <- function(x,deg) {
  d <- matrix(0,length(x),deg+1)
  for (i in 0:deg) {
    d[,i+1] <- x ^ i
  }
  return (data.frame(d))
}
## demo learning logistic regression, with different datasets generated,
## and using different degree polynomials as features
demolearn <- function(trainx,testx,truep,deg) {
  trainp <- truep(trainx)
  testp  <- truep(testx)
  par(mfrow=c(3,4),ann=FALSE,cex=.3,mar=c(1,1,1,1))
  predp <- matrix(0,length(testx),11)
  for (i in 1:11) {
    trainy <- as.numeric(runif(length(trainx)) < trainp)
    lr <- glm(trainy ~ .,data=phi(trainx,deg),family=binomial)
    predp[,i] <- predict(lr,newdata=phi(testx,deg),type="response")
    plot(testx,testp,type="l",col=2,lwd=3,ylim=c(-.1,1.1))
    lines(testx,predp[,i],type="l")
    points(trainx,trainy,pch=1,col=4,cex=2)
  }
  plot(testx,testp,type="l",lwd=3,col=2,ylim=c(-.1,1.1))
  for (i in 1:11) {
    lines(testx,predp[,i],type="l")
  }
  return(predp)
}

trainx <- seq(-10,10,.5)
testx  <- seq(-12,12,.1)
pp <- demolearn(trainx,testx,truep,4)
```

Regularization

- Flexible models for high-dimensional problems require many parameters.
- With many parameters, learners can easily overfit to the noise in the training data.
- **Regularization**: Limit flexibility of model to prevent overfitting.
- Typically: add term **penalizing** large values of parameters θ.

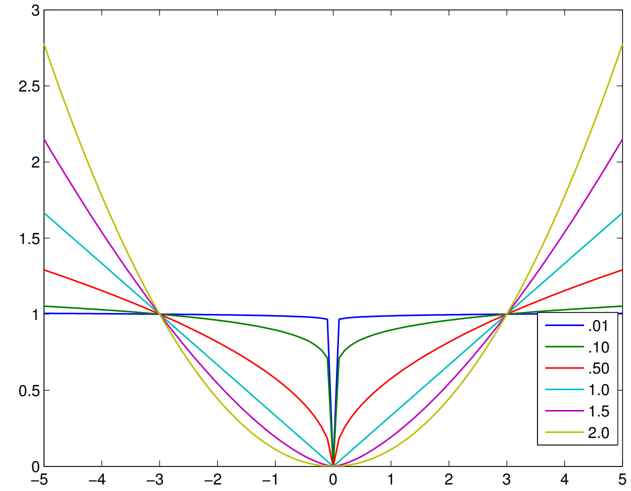
$$R^{\text{emp}}(\theta) + \lambda\|\theta\|_\rho^\rho = \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(a+b^\top x_i))) + \lambda\|b\|_\rho^\rho$$

where $\rho \in [1,2]$, and $\|z\|_\rho = (\sum_{j=1}^{p}|z_j|^\rho)^{1/\rho}$ is the L_ρ norm of b (also of interest when $\rho \in [0,1)$, but is no longer a norm).

- Also known as **shrinkage** methods—parameters are shrunk towards 0.
- Typical cases are $\rho = 2$ (Euclidean norm, **ridge regression**) and $\rho = 1$ (**LASSO**). When $\rho \leq 1$ it is called a **sparsity** inducing regularization.
- λ is a **tuning parameter** (or **hyperparameter**) and controls the amount of regularization, and resulting complexity of the model.

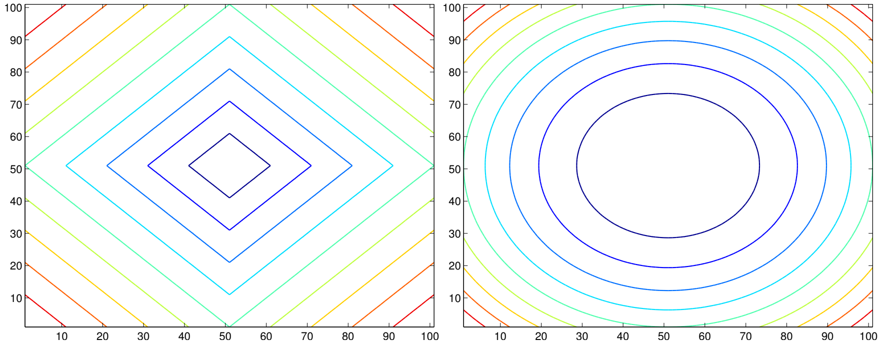
Bach, Sparse Methods in Machine learning.

Regularization



L_ρ regularization profile for different values of ρ.

Regularization



L_1 and L_2 norm contours.

Sparsity Inducing Regularization

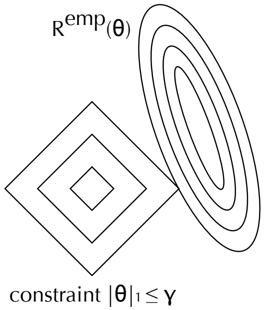
- Consider constrained optimization problem

$$\min_{\theta} R^{\text{emp}}(\theta) \text{ s.t. } \|\theta\|_1 < \gamma$$

- Lagrange multiplier $\lambda > 0$ to enforce constraint,

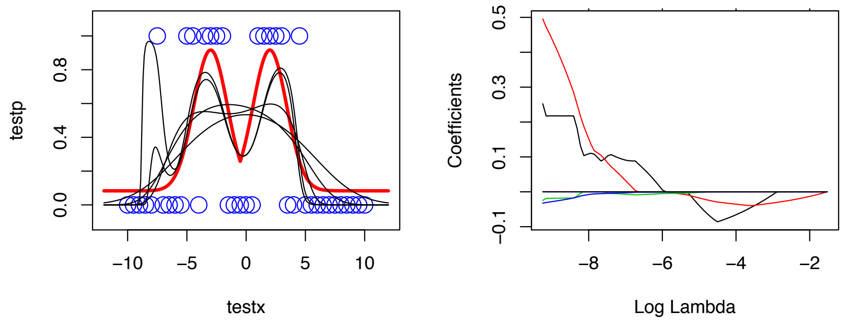
$$\min_{\theta} R^{\text{emp}}(\theta) + \lambda(\|\theta\|_1 - \gamma)$$

  - At the optimal value of λ, the parameter θ is the one minimizing the regularized empirical risk objective.
  - Conversely, given λ, there is a value of γ such that the corresponding optimal Lagrange multiplier is λ.
- Using L_1 regularization, optimal θ has $\theta_2 = 0$.
- Generally: L_1 regularization leads to optimal solutions with many zeros, i.e. the regression function depends only on the (small) number of features with non-zero parameters.



Demo on L_1 Regularized Logistic Regression

Use `glmnet` for regression with L_1, L_2 and combination regularization.



Demo on L_1 Regularized Logistic Regression

```
## true conditional probabilities
truep <- function(x) {
  return((pmax(exp(-(x-2)^2/4),exp(-(x+3)^2/4))+.1)/1.2)
}
## features are {x^i}
phi <- function(x,deg) {
  d <- matrix(0,length(x),deg+1)
  for (i in 0:deg) {
    d[,i+1] <- x ^ i
  }
  return (data.frame(d))
}
## demo L1 regularized learning of logistic regression,
## with different datasets generated, and using different
## degree polynomials as features

trainx <- seq(-10,10,.5)
testx  <- seq(-12,12,.1)

demolearnL1 <- function(trainx,testx,truep,deg) {
  trainp <- truep(trainx)
  testp  <- truep(testx)
  trainy <- as.numeric(runif(length(trainx)) < trainp)
  slr <- glmnet(as.matrix(phi(trainx,deg)),as.factor(trainy),
          family="binomial")
  s <- c(0,.0001,.001,.01,.05)
  predp <- predict(slr,newx=as.matrix(phi(testx,deg)),
        s=s,type="response")
  par(mfrow=c(1,2),mar=c(4,4,1,2))
  plot(testx,testp,type="l",col=2,lwd=3,ylim=c(-.1,1.1))
  points(trainx,trainy,pch=1,col=4,cex=2)
  for (i in 1:dim(predp)[2]) {
    lines(testx,predp[,i],type="l")
  }
  plot(slr,xvar="lambda")
  print(coef(slr,s))
  return(predp)
}

demolearnL1(trainx,testx,truep,10)
```

Optimization

- Many more complex models in statistics and machine learning do not have analytic solutions to ML estimators.
- In most models parameters are learned by some numerical optimization technique.

$$\min_{\theta} F(\theta)$$

- How many minima are there?
- How do we find optimal θ?
- Are we guaranteed to find the global optimum θ^*, rather just a local one?
- How efficiently can we solve for θ?
- What if there are constraints?

Constrained Optimization

- Optimization problems with constraints, e.g.

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} \quad & F(\theta) \\ \text{subject to} \quad & g_i(\theta) \le 0 \quad \text{for } i = 1, \dots, I \\ & h_j(\theta) = 0 \quad \text{for } j = 1, \dots, J \end{aligned}$$

  where g_i enforce inequality constraints and h_j equality constraints.
- Can write this succinctly:

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} \quad & F(\theta) \\ \text{subject to} \quad & g(\theta) \preceq 0 \\ & h(\theta) = 0 \end{aligned}$$

  where $g : \mathbb{R}^d \to \mathbb{R}^I$ is a vector-valued function with $g(\theta)_i = g_i(\theta)$. Similarly $h(\theta) : \mathbb{R}^d \to \mathbb{R}^J$. $x \preceq y$ iff $x_i \le y_i \forall i$.
- These problems are called **programmes**.

Constrained Optimization

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} \quad & F(\theta) \\ \text{subject to} \quad & g(\theta) \preceq 0 \\ & h(\theta) = 0 \end{aligned}$$

- We can enforce constraints by using **Lagrange multipliers** or **dual variables** $\lambda \in \mathbb{R}^I$ and $\kappa \in \mathbb{R}^J$.
- The optimization problem can be written as a mini-max optimization of the Lagrangian:

$$\min_{\theta} \max_{\lambda \succeq 0, \kappa} \mathcal{L}(\theta, \lambda, \kappa) = \min_{\theta} \max_{\lambda \succeq 0, \kappa} F(\theta) + \lambda^\top g(\theta) + \kappa^\top h(\theta)$$

- Intuition: For any θ, we have:

$$\max_{\lambda \succeq 0, \kappa} \mathcal{L}(\theta, \lambda, \kappa) = \begin{cases} +\infty & \text{if there is some unsatisfied constraint,} \\ F(\theta) & \text{if all constraints are satisfied.} \end{cases}$$

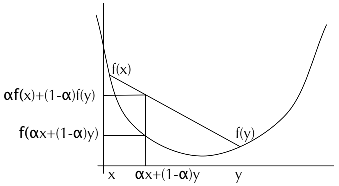
  So the outer minimization over θ results in the same optimization problem.

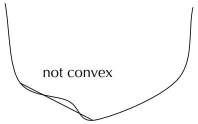
Convex Optimization

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if

$$f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y)$$

  for all $x, y \in \mathbb{R}^d$, $\alpha \in [0, 1]$.
- For smooth functions: Equivalent to 2nd derivative (**Hessian**) being positive semidefinite.
- A programme is a **convex programme** if:
  - $F(\theta)$ is convex,
  - $g_i(\theta)$ is convex for each i,
  - $h(\theta) = A\theta + b$ is affine.
- Examples: linear, quadratic, semidefinite programming.
- Convex programmes have a unique minimum (typically), which can be efficiently found.





Boyd and Vandenberghe, Convex Optimization. 2004. MOOC right now.

Convex Duality

- Say the minimum is p^*, and occurred at θ^*.
- The **dual programme** inverts the order of max and min:

$$p^* = \min_{\theta} \max_{\lambda \succeq 0, \kappa} \mathcal{L}(\theta, \lambda, \kappa) \ge \max_{\lambda \succeq 0, \kappa} \min_{\theta} \mathcal{L}(\theta, \lambda, \kappa) = d^*$$

  where the dual optimum is d^*.
- **Karush-Kuhn-Tucker Theorem**: Subject to regularity conditions, a solution θ^* is the optimal solution of a convex programme, if and only if there are λ^* and κ^* (the dual optimal solution) such that:
  - **Primal feasible**: $g(\theta^*) \preceq 0$, $h(\theta^*) = 0$.
  - **Dual feasible**: $\lambda^* \succeq 0$.
  - $(\theta^*, \lambda^*, \kappa^*)$ is a **saddle point** of $\mathcal{L}$: For every $\theta, \lambda \succeq 0, \kappa$, we have

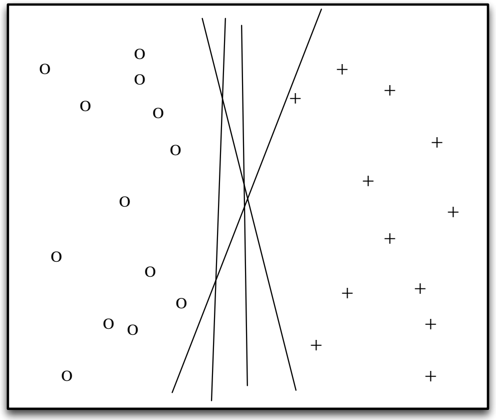
$$\mathcal{L}(\theta^*, \lambda, \kappa) \le \mathcal{L}(\theta^*, \lambda^*, \kappa^*) \le \mathcal{L}(\theta, \lambda^*, \kappa^*)$$

  - $\nabla_\theta \mathcal{L}(\theta^*, \lambda^*, \kappa^*) = \nabla_\theta F(\theta^*) + (\lambda^*)^\top \nabla_\theta g(\theta^*) + (\kappa^*)^\top \nabla_\theta h(\theta^*) = 0$
  - **Complementary slackness**: For every i,

$$\lambda_i^* g_i(\theta^*) = 0$$

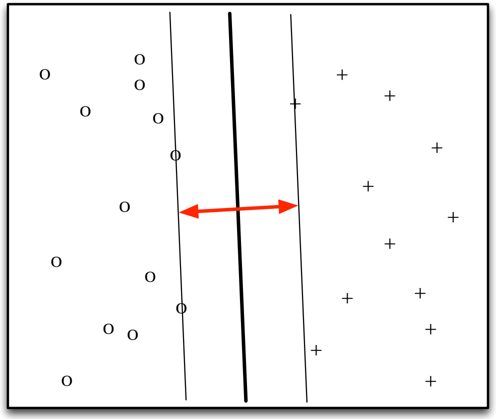
Linear Classification

- A dataset with $\{+1,-1\}$ labels is **linearly separable** if there is a hyperplane separating two classes.
- Typically there will be an infinite number of such **separating hyperplanes**.



Maximum Margin Classification

- Good choice of separating hyperplane: one with **large margin**.
- Such a hyperplane will be defined by a number of data vectors close to the boundary—the **support vectors**, leading to a method called **support vector machines**.

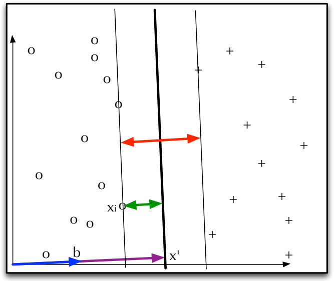


Support Vector Machines

- A hyperplane can be parametrized as:

$$g(x) = a + b^\top x = 0$$

with the classification given by $\operatorname{sign}(g(x))$.



- Distance and classification of a point x_i from hyperplane is $g(x_i)/\|b\|$.
- Multiplying a and b by $c > 0$ does not affect result. Rescale such that margin (closest distance of data vectors to hyperplane) is $1/\|b\|$.

$$y_i(a + b^\top x_i)/\|b\| \geq 1/\|b\|$$
$$y_i(a + b^\top x_i) \geq 1$$

- Constrained optimization problem to solve for a, b:

$$\begin{array}{lll} \max_{a,b} & 1/\|b\| & \\ \text{subject to} & y_i(a+b^\top x_i) \geq 1 & \end{array} \Leftrightarrow \begin{array}{ll} \min_{a,b} & \frac{1}{2}\|b\|^2 \\ \text{subject to} & y_i(a+b^\top x_i) \geq 1 \quad \text{for all } i \end{array}$$

Support Vector Machines

- Introduce Lagrange multipliers $\lambda_i \geq 0$ to enforce constraints:

$$\min_{a,b} \max_{\lambda \succeq 0} \mathcal{L}(a,b,\lambda) = \frac{1}{2}\|b\|^2 + \sum_{i=1}^{n} \lambda_i (1 - y_i(a + b^\top x_i))$$

- KKT optimality conditions:

Zero derivatives:

$$\nabla_a \mathcal{L}(a^*, b^*, \lambda^*) = -\sum_{i=1}^{n} \lambda_i^* y_i = 0$$
$$\nabla_b \mathcal{L}(a^*, b^*, \lambda^*) = b^* - \sum_{i=1}^{n} \lambda_i^* y_i x_i = 0$$

Primal feasibility: $y_i(a^* + (b^*)^\top x_i) \geq 1$

Dual feasibility: $\lambda_i^* \geq 0$

Complementary slackness: $\lambda_i^*(1 - y_i(a^* + (b^*)^\top x_i)) = 0$

Support Vector Machines

- Substituting optimal a^* and b^* into Lagrangian leads to the **dual optimization problem**:

$$\begin{aligned}\max_{\lambda} \quad & \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{n} \lambda_i\lambda_j y_i y_j (x_i)^\top (x_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \lambda_i y_i = 0 \\ & \lambda \succeq 0\end{aligned}$$

  A **quadratic programme**. Standard solvers can be used to find optimal λ^* in $O(n^3)$ cost.
- Those vectors with $\lambda_i > 0$ are called **support vectors**.
- Complementary slackness implies that if x_i does not lie on boundary, then $\lambda_i = 0$, i.e. not a support vector.
- Discriminant function is

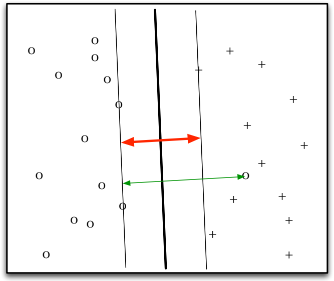
$$g(x) = a^* + \sum_{i=1}^{n} \lambda_i^* y_i x_i^\top x$$

  where a^* can be solved by noting that $y_j g(x_j) = 1$ for a support vector x_j.

Soft-Margin Support Vector Machines

- For **non-linearly separable** datasets, we can allow for **margin violations**

$$\xi_i = \begin{cases} 1 - y_i(a + b^\top x_i) & \text{if margin violated,} \\ 0 & \text{if not violated.} \end{cases}$$
$$= \max(0, 1 - y_i(a + b^\top x_i))$$



- Penalizing violations by their magnitude,

$$\begin{aligned}\min_{a,b,\xi} \quad & \frac{1}{2}\|b\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(a + b^\top x_i) \geq 1 - \xi_i \\ & \xi_i \geq 0\end{aligned}$$

  where C is a tuning parameter.

Soft-Margin Support Vector Machines

- Introduce Lagrange multipliers $\lambda_i \geq 0$, $\gamma_i \geq 0$ to enforce constraints:

$$\mathcal{L}(a, b, \xi, \lambda, \gamma) = \frac{1}{2}\|b\|^2 + C\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \lambda_i(1 - \xi_i - y_i(a + b^\top x_i)) - \sum_{i=1}^{n} \gamma_i \xi_i$$

- KKT optimality conditions:

Zero derivatives:
$$\nabla_a \mathcal{L}(a^*, b^*, \xi^*, \lambda^*, \gamma^*) = -\sum_{i=1}^{n} \lambda_i^* y_i = 0$$
$$\nabla_b \mathcal{L}(a^*, b^*, \xi^*, \lambda^*, \gamma^*) = b^* - \sum_{i=1}^{n} \lambda_i^* y_i x_i = 0$$
$$\nabla_{\xi_i} \mathcal{L}(a^*, b^*, \xi^*, \lambda^*, \gamma^*) = C - \lambda_i^* - \gamma_i^* = 0$$

Primal feasibility:
$$y_i(a^* + (b^*)^\top x_i) \geq 1 - \xi_i^*$$
$$\xi_i^* \geq 0$$

Dual feasibility:
$$\lambda_i^* \geq 0$$
$$\gamma_i^* \geq 0$$

Complementary slackness:
$$\lambda_i^*(1 - \xi_i^* - y_i(a^* + (b^*)^\top x_i)) = 0$$
$$\gamma_i^* \xi_i^* = 0$$

Soft-Margin Support Vector Machines

- Setting derivatives of primal variables to zero leads to the dual programme:

$$\begin{aligned}\max_{\lambda} \quad & \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{n} \lambda_i\lambda_j y_i y_j (x_i)^\top (x_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \lambda_i y_i = 0 \\ & 0 \preceq \lambda \preceq C\end{aligned}$$

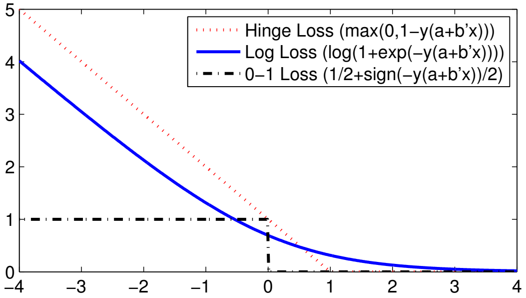
  Only difference is the **box constraint** on $\lambda_i \in [0, C]$.

Soft-Margin Support Vector Machines

- From primal programme, we can first minimize over ξ_i's, leading to an unconstrained convex programme:

$$\min_{a,b} \quad \frac{1}{2}\|b\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(a + b^\top x_i))$$

- Interpretation: regularized empirical risk minimization with the **hinge loss**.

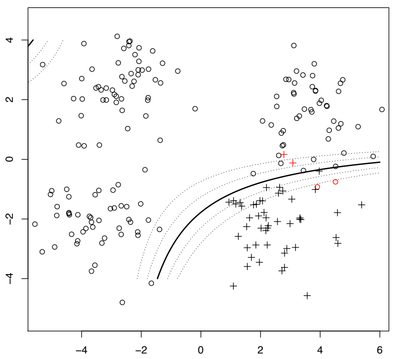


Support Vector Machines – Discussion

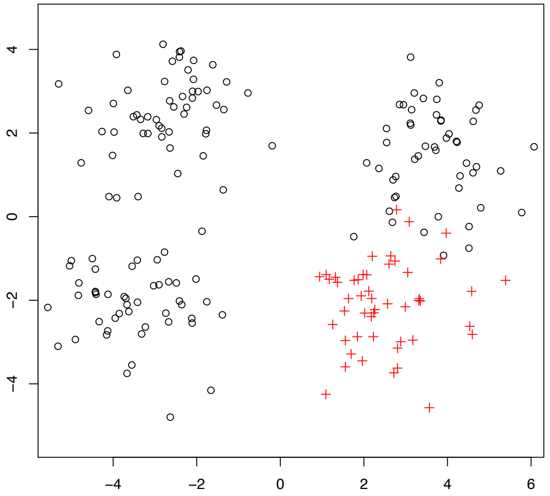
- **Multiclass classification**: If there are more than two classes, there are multiclass generalizations of the SVM.
- A simple practical idea: treat a multiclass problem as multiple binary classification problems.
  - One-vs-one: train $K(K-1)$ binary SVMs, for each pair of classes. At test time, predict class that got the most votes.
  - One-vs-rest: train K binary SVMs, one for each class vs all other classes. At test time, predict class with largest discriminant value $a_k + b_k^\top x$.
- Optimization for large scale problems:
  - Standard quadratic programme solvers not scalable.
  - Sequential minimal optimization (SMO): iterative solve pairs of λ_i's.
  - Pegasos : stochastic gradient descent on regularized hinge loss objective.
- L_2 regularization controls overfitting.
- Not probabilistic and cannot produce uncertainty estimates.
- Statistical learning theory foundations.
- Further readings:
  - Bishop, Chapter 6.
  - Christopher Burgess, A Tutorial on Support Vector Machines for Pattern Recognition. 1998.

Nonlinear Methods

- Decision boundaries and regression functions often need to be nonlinear.
- One general approach: transform data $x \mapsto \phi(x)$.
- A **global** approach. Decisions and optimal parameters depend on **whole** training dataset.
- Alternative approach: $p(Y = 1|X = x)$ or $f(x)$ depends only on data cases in **local neighbourhood** of x.



Local Methods



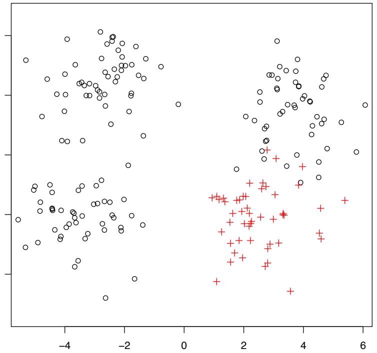
k-Nearest Neighbours

- A simple, local, nonlinear, non-model-based, method.
- Prediction at a data vector x is simply determined by the k nearest neighbours $ne_k(x)$ of x among the training set.
- Classification: predict the **majority vote** of the neighbours:

$$f^{\text{kNN}}(x) = \underset{l}{\operatorname{argmax}}\ |\{j \in ne_k(x) : y_j = l\}|.$$

- Regression: predict the average among the neighbours:

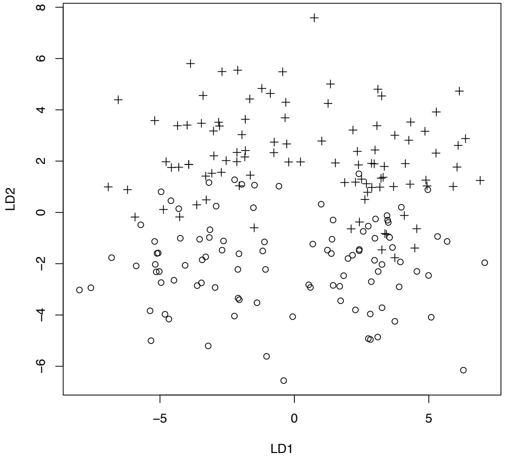
$$f^{\text{kNN}}(x) = \frac{\sum_{j \in ne_k(x)} y_j}{\sum_{j \in ne_k(x)} 1}.$$



k-Nearest Neighbours

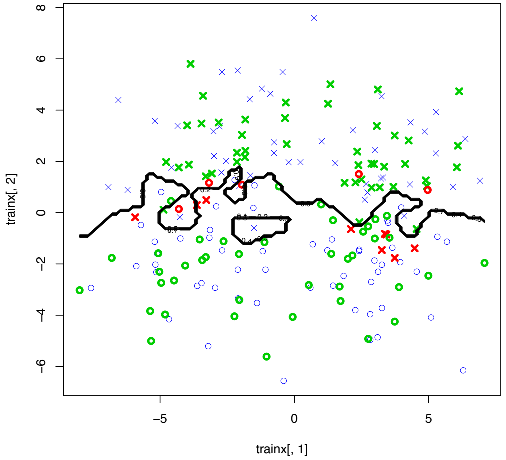
- Nearest neighbours are simple and essentially model-free methods for classification.
- Weaker modelling assumptions than e.g. LDA, Naïve Bayes and logistic regression.
- These methods are not very useful for understanding relationships between attributes and class predictions.
- As **black box** classification methods however, they are often perform reasonably on real life problems and provide a good benchmark.
- Can break down in high-dimensional data:
  - Effectively, partitions input space into regions each containing k data points, and prediction in each region estimated separately.
  - In a space of dimension $p \gg 0$, number of regions needed is $R = m^p$, so size of dataset needed is km^p.

k-Nearest Neighbour Demo



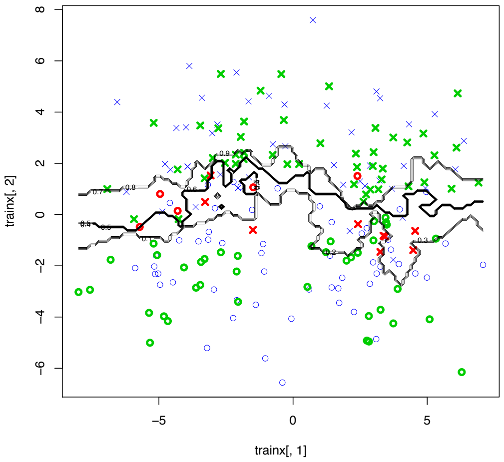
Data

k-Nearest Neighbour Demo



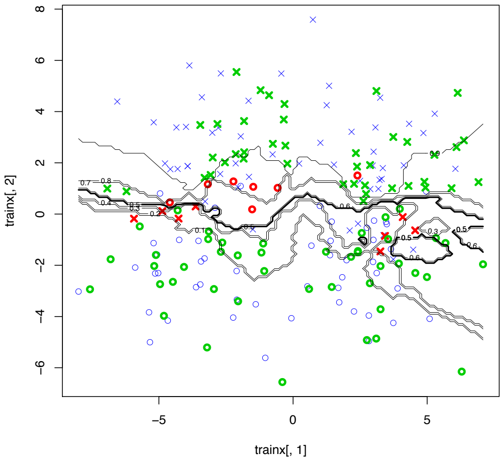
Result of 1NN

k-Nearest Neighbour Demo



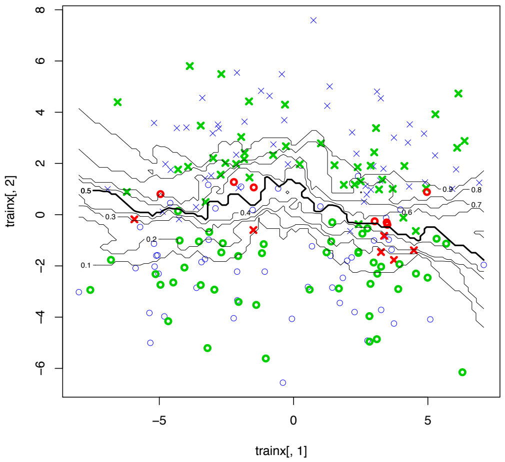
Result of 3NN

k-Nearest Neighbour Demo



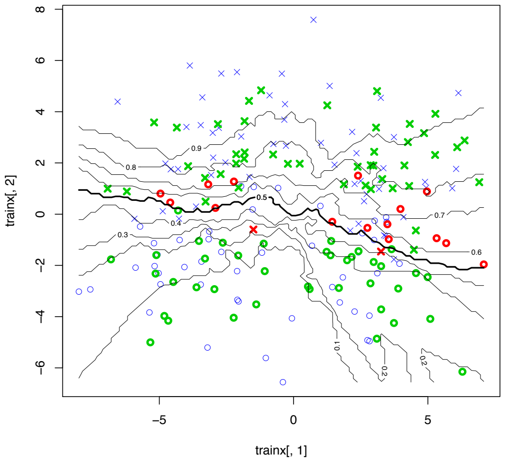
Result of 5NN

k-Nearest Neighbour Demo



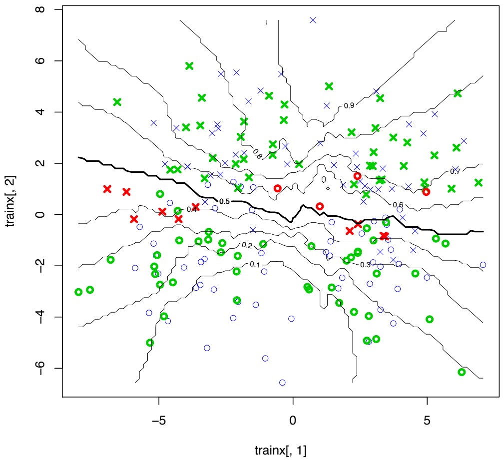
Result of 11NN

k-Nearest Neighbour Demo



Result of 21NN

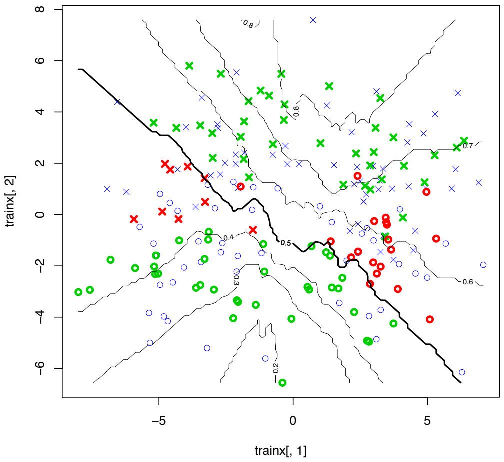
k-Nearest Neighbour Demo



Result of 31NN

---

k-Nearest Neighbour Demo



Result of 51NN

---

k-Nearest Neighbour Demo – R Code I

```
library(MASS)
## load crabs data data(crabs)
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)
## project into first two LD
cb.lda <- lda(log(crabs[,4:8]),ct)
cb.ldp <- predict(cb.lda)
x <- as.matrix(cb.ldp$x[,1:2])
y <- as.numeric(crabs[,2])-1
x <- x + rnorm(dim(x)[1]*dim(x)[2])*1.5
eqscplot(x,pch=2*y+1,col=1)

k <- 3

kNN <- function(k,x,y,gridsize=100) {

  n      <- length(y)
  p      <- dim(x)[2]
  i      <- sample(rep(c(TRUE,FALSE),each=n/2),n,replace=FALSE)
  train  <- (1:n)[i]
  test   <- (1:n)[!i]
  trainx <- x[train,]
  trainy <- y[train]
  testx  <- x[test,]
  testy  <- y[test]


  trainn <- dim(trainx)[1]
  testn  <- dim(testx)[1]

  gridx1 <- seq(min(x[,1]),max(x[,2]),length=gridsize)
  gridx2 <- seq(min(x[,2]),max(x[,2]),length=gridsize)
  gridx  <- as.matrix(expand.grid(gridx1,gridx2))
  gridn  <- dim(gridx)[1]

  # calculate distances, smart and intelligently.
  trainxx <- t((trainx*trainx) %*% matrix(1,p,1))
```

---

k-Nearest Neighbour Demo – R Code II

```
  testxx  <- (testx*testx)   %*% matrix(1,p,1)
  gridxx  <- (gridx*gridx)   %*% matrix(1,p,1)
  testtraindist <- matrix(1,testn,1) %*% trainxx +
                   testxx %*% matrix(1,1,trainn) -
                   2*(testx %*% t(trainx))
  gridtraindist <- matrix(1,gridn,1) %*% trainxx +
                   gridxx %*% matrix(1,1,trainn) -
                   2*(gridx %*% t(trainx))

  # predict
  testp <- numeric(testn)
  gridp <- numeric(gridn)
  for (j in 1:testn) {
    nearestneighbors <- order(testtraindist[j,])[1:k]
    testp[j] <- mean(trainy[nearestneighbors])
  }
  for (j in 1:gridn) {
    nearestneighbors <- order(gridtraindist[j,])[1:k]
    gridp[j] <- mean(trainy[nearestneighbors])
  }
  predy <- as.numeric(testp>.5)

  plot(trainx[,1],trainx[,2],pch=trainy*3+1,col=4,lwd=.5)
  points(testx[,1],testx[,2],pch=testy*3+1,col=2+(predy==testy),lwd=3)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),
        levels=seq(.1,.9,.1),lwd=.5,add=TRUE)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),
        levels=c(.5),lwd=2,add=TRUE)
}
```

Asymptotic Performance of 1NN

- Let $(x_i, y_i)_{i=1}^n$ be training data where $x_i \in \mathbb{R}^p$ and $y_i \in \{1, 2, ..., K\}$.
- We define

$$\widehat{y}_{\text{Bayes}}(x) = \underset{l \in \{1,\dots,K\}}{\arg\max}\ \pi_l f_l(x)$$

and

$$\widehat{y}_{\text{1NN}}(x) = y\,(\text{nearest neigbour of } x)\,.$$

- The (optimal) Bayes risk and 1NN risk are:

$$\begin{aligned} R_{\text{Bayes}} &= \mathbb{E}\left[\mathbb{I}\left(Y \neq \widehat{y}_{\text{Bayes}}(X)\right)\right] \\ R_{\text{1NN}} &= \mathbb{E}\left[\mathbb{I}\left(Y \neq \widehat{Y}_{\text{1NN}}(X)\right)\right] \end{aligned}$$

- As $n \to \infty$, we have the following powerful result

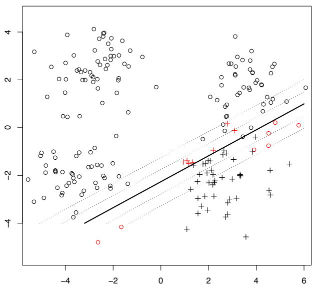
$$R_{\text{Bayes}} \leq R_{\text{1NN}} \leq 2R_{\text{Bayes}} - \frac{K}{K-1} R_{\text{Bayes}}^2.$$

K-Nearest Neighbours – Discussion

- kNN is sensitive to distances: normalize data and find suitable metric.
- Choice of k important: controls flexibility of model.
- Computational cost of kNN is very high.
  - Need to store **all** training data.
  - Need to compare each test data vector to **all** training data.
  - Need **a lot of data** in high dimensions.
- Mitigation techniques:
  - Compute approximate nearest neighbours, using kd-trees, cover trees, random forests.
  - Apply K-means to data in each class, to reduce size of data (need to use large K).

Non-linear Problems

- Linear methods (PCA, LDA, linear and logistic regression) are simple and effective techniques to learn from data "to first order".
- To capture more intricate information from data, flexible, non-linear methods are often needed.
  - Explicit non-linear transformations $x \mapsto \phi(x)$.
  - Local methods like kNN.
- **Kernel methods**: introduce non-linearities through **implicit** non-linear transforms, often local in nature.



The Kernel Method

- Back to the soft-margin SVM. The dual objective is:

$$\max_{\lambda} \sum_{i=1}^n \lambda_i - \frac{1}{2}\sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j \phi(x_i)^\top \phi(x_j) \quad \text{subject to} \quad \begin{cases} \sum_{i=1}^n \lambda_i y_i = 0 \\ 0 \preceq \lambda \preceq C \end{cases}$$

- Suppose $p = 2$, and we would like to introduce quadratic non-linearities,

$$\phi(x_i) = (1, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}, x_{i1}^2, x_{i2}^2, x_{i1}x_{i2})^\top$$

Then

$$\begin{aligned} \phi(x_i)^\top \phi(x_j) &= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + x_{i1}x_{i2}x_{j1}x_{j2} \\ &= (1 + x_i^\top x_j)^2 \end{aligned}$$

- Since only dot-products are needed in the objective function, non-linear transform need not be computed explicitly!
- Generally, m-order interactions can be implemented simply by $\phi(x_i)^\top \phi(x_j) = (1 + x_i^\top x_j)^m$. This is called a **polynomial kernel**.

The Kernel Method

- The **Gram matrix** is the matrix of dot-products, $B_{ij} = \phi(x_i)^\top \phi(x_j)$.

$$B = \begin{pmatrix} — \phi(x_1)^\top — \\ \vdots \\ — \phi(x_i)^\top — \\ \vdots \\ — \phi(x_n)^\top — \end{pmatrix} \times \begin{pmatrix} | & & | & & | \\ \phi(x_1) & \cdots & \phi(x_j) & \cdots & \phi(x_n) \\ | & & | & & | \end{pmatrix}$$

- Since $B = \Phi\Phi^\top$, it is symmetric and positive semidefinite.
- The Gram matrix is sufficient for training the soft-margin SVM.

$$\max_{\lambda} \quad \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{n} \lambda_i \lambda_j y_i y_j B_{ij} \qquad \text{subject to} \quad \begin{cases} \sum_{i=1}^{n} \lambda_i y_i = 0 \\ 0 \preceq \lambda \preceq C \end{cases}$$

The Kernel Method

- A **kernel** is a function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that:
  - Symmetric: $\kappa(x, x') = \kappa(x', x)$.
  - Positive semidefinite: given any finite set $\{x_i\}_{i=1}^{n} \subset \mathcal{X}$, the matrix $B \in \mathbb{R}^{n \times n}$ with entries $B_{ij} = \kappa(x_i, x_j)$ is positive definite. Equivalently, for any $c \in \mathbb{R}^n$,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \kappa(x_i, x_j) \geq 0$$

- **Mercer's Theorem**: if κ is continuous, symmetric and positive semidefinite, then there is a function $\phi : \mathcal{X} \to \mathcal{H}$ into a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot, \rangle$ such that

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$$

The Kernel Method

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$$

- We do not need to compute the features ever—the Gram matrix is sufficient for learning and prediction. The discriminant function (absorbing a into b) is

$$g(x) = \sum_{i=1}^{n} \lambda_i^* y_i \phi(x_i)^\top \phi(x) = \sum_{i=1}^{n} \lambda_i^* y_i \kappa(x_i, x)$$

- The function ϕ can be interpreted as non-linear **features** of our data vectors $x \in \mathcal{X}$.
- Generally, the Hilbert space can be **infinite-dimensional**, so we are effectively computing an infinite number of features of our data, and learning a SVM based on all features.
- There are an infinite number of parameters in the SVM—a **nonparametric** method.
- The L_2 regularization of SVMs is very important to prevent overfitting.

Examples of Kernels

- Polynomial kernel:

$$\kappa(x, x') = (1 + x^\top x')^m$$

- **Gaussian, radial-basis function (RBF), or squared-exponential kernel**:

$$\kappa(x, x') = \exp\left(-\frac{1}{2} \|x - x'\|_M^2\right)$$

  This leads to a discriminant function of form

$$g(x) = \sum_{i=1}^{n} \lambda_i^* y_i \exp\left(-\frac{1}{2} \|x_i - x\|_M^2\right)$$

  A local method very similar to kNN.
- If κ_1 and κ_2 are both kernels, then so are kernels defined by

$$\kappa_3(x, x') = \kappa_1(x, x') + \kappa_2(x, x')$$
$$\kappa_4(x, x') = \kappa_1(x, x') \times \kappa_2(x, x')$$

Kernel SVM Demo

```
library(MASS)
library(e1071)
## load crabs data, project onto LD space, add noise.
data(crabs)
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)
cb.lda <- lda(log(crabs[,4:8]),ct)
cb.ldp <- predict(cb.lda)
x <- as.matrix(cb.ldp$x[,1:2])
y <- as.numeric(crabs[,2])-1
x <- x + rnorm(dim(x)[1]*dim(x)[2])*1.5
gridsize <- 100
xlim   <- c(min(x[,1]),max(x[,1]))
ylim   <- c(min(x[,2]),max(x[,2]))
gridx1 <- seq(xlim[1],xlim[2],length=gridsize)
gridx2 <- seq(ylim[1],ylim[2],length=gridsize)
gridx  <- as.matrix(expand.grid(gridx1,gridx2))
gridn  <- dim(gridx)[1]
plot(x,pch=2*y+1,col=1,xlim=xlim,ylim=ylim)

n      <- length(y)
p      <- dim(x)[2]
i      <- sample(rep(c(TRUE,FALSE),each=n/2),n,replace=FALSE)
train  <- (1:n)[i]
test   <- (1:n)[!i]
trainx <- x[train,]
trainy <- y[train]
testx  <- x[test,]
testy  <- y[test]

svmdemo <- function(kernel,gamma=1,coef0=0,cost=1,degree=3) {
  model <- svm(trainx,trainy,kernel=kernel,gamma=gamma,coef0=coef0,degree=degree,cost=cost)
  gridp <- predict(model,gridx)
  predy <- as.numeric(predict(model,testx)>.5)

  plot(trainx[,1],trainx[,2],pch=trainy*3+1,col=4,lwd=.5,xlim=xlim,ylim=ylim)
  points(testx[,1],testx[,2],pch=testy*3+1,col=2+(predy==testy),lwd=3)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),levels=seq(.1,.9,.1),lwd=.5,add=TRUE)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),levels=c(.5),lwd=2,add=TRUE)
}
```

Kernel Methods – Discussion

- The kernel method allows for very flexible and powerful machine learning models.
- Kernels can be defined over much more complex structures than vectors, e.g. graphs, strings.
- Can be hard to interpret.
- $O(n^3)$ computation and $O(n^2)$ memory cost can be prohibitive.
- Further readings:
  - Bishop, Chapter 6.
  - Christopher Burgess, A Tutorial on Support Vector Machines for Pattern Recognition. 1998.
  - Rasmussen and Williams, Gaussian Processes for Machine Learning. 2006.