Lecture Notes

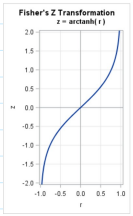
24 February 2020 14:15

Fisher z-transformation:

This can be used to inspect and get confidence intervals for the estimated correlation coefficient

It is based on the variance stabilizing transformation

$$z = \frac{1}{2} \ln\left(\frac{1+r}{1-r}\right) = \operatorname{arctanh}(r)$$



This transformation was proposed by Fisher in order to deal with the properties of correlation coefficients which, by definition $\in [-1, 1]$. The issue like that is that when observing and sampling correlation coefficients with a true correlation coefficient lying on the edges you get a very skewed distribution from which it is difficult to make statistical conclusions on the robustness of such coefficient.

Fisher explored the topic further and found that in the case of a bivariate normal distribution of (X, Y) the above transformation can be shown to be asymptotically normal distributed with

$$\mu = \frac{1}{2} \ln\left(\frac{1+\rho}{1-\rho}\right)$$

$$SE = \frac{1}{\sqrt{N-3}}$$

It follows that the above can be used to obtain confidence intervals for the correlation coef. because for large N:

$$\sqrt{N-3}\,(z-\mu) \sim N(0,1)$$

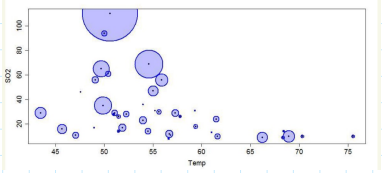
Then once you obtain the confidence interval for z you can back-transform to r using the inverse of the z-transform

$$r = \tanh(z)$$

Some Multivariate Plotting Techniques:

- Glyph: here you use different plotting attributes (such as size, color, etc.) to represent different variables

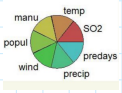
  Example: Bubble Plot



- Star plots: Here each star represents one single observation. The spikes then represent the normalized values of each variable.

The normalization that is usually applied is:

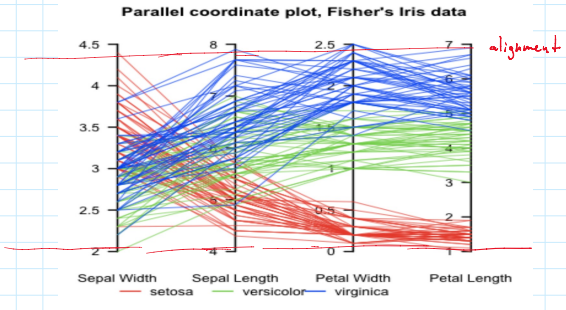
$$\frac{X_i - \min(\vec{X})}{\max(\vec{X}) - \min(\vec{X})}$$



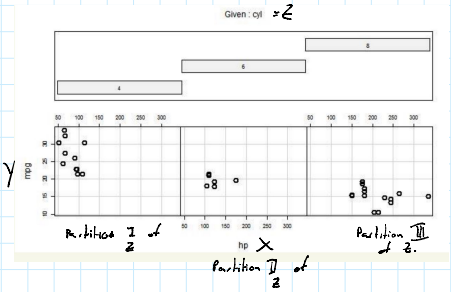
- parallel cooridinate plots:

  the different variables are expressed on the X-Axis at different positions.

  The values for the variables are expressed by the parallel vertical coordinates. This are scaled so that the max and min among all variables are aligned as visible below.



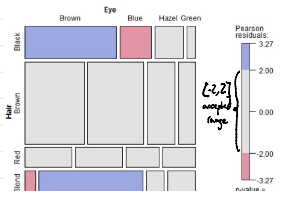
- Conditioning Plots:



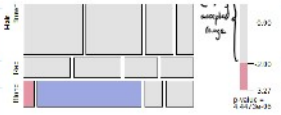
  Here the idea is to partition one of the variables (here z) into equivalent intervals and observe how the scatterplot of other variables behaves for that particular partition.

  This will help to analyze structural breaks and different patterns in the different partitions.

- Mosaic Plots: These are used for visualizing categorical variables where the techniques above fails.



  The idea here is to plot rectangles proportional to the frequency with with the categorical combination occurs.



frequency with which the categorical combination occurs.

The plot can be further enriched with colours representing the significance of a given categorical variable being independent with the other. Both the blue and the red colouring above rejects the hypothesis of independence among the variables and reasoning goes as follows:

- Consider the following contingency table.

	A=1	A=2	Total
B=1	n_{11}	n_{12}	$n_{1\cdot}$
B=2	n_{21}	n_{22}	$n_{2\cdot}$
Total	$n_{\cdot 1}$	$n_{\cdot 2}$	n

↳ Assume that A and B are independent. Then it follows:

$$P(A_i \cap B_j) = P(A_i) \cdot P(B_j)$$

$$= \frac{n_{\cdot i}}{n} \cdot \frac{n_{j\cdot}}{n} = \underbrace{\hat{\pi}_{ij}}_{\text{Expected prob. of observing } A_i \cap B_j \text{ under the Null}}$$

It follows now that the expected amount of observations should be $E_{ij} = n \cdot \hat{\pi}_{ij}$.

From this you can compute the following statistic:

$$X^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{(n_{ij} - E_{ij})^2}{E_{ij}}$$

Squared Pearson Residuals.

As under the Null $\frac{(n_{ij} - E_{ij})^2}{E_{ij}} \sim N(0,1)$

it follows that the above is $\chi^2_{(2-1)(2-1)}$ degrees of freedom and you can therefore easily get the desired test-statistics and significance values.

- Multivariate Outliers.

While it may be easy to recognise outliers in $\mathbb{R}^2$ and possibly even $\mathbb{R}^3$ the situation gets tricky in hyperspaces.

A simple solution for finding outliers of q-variate multivariate normal distributed data is the one of leveraging the Mahalanobis distance.

Such distance is defined as follows:

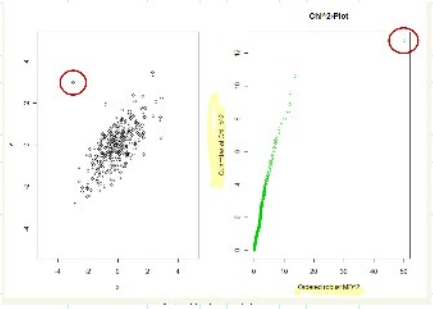
1-Dimensional-Case: $\frac{x-\mu}{\sigma} =: Z\text{-score}$

N-Dimensional-Case: $\sqrt{(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})}$

It is now clear that under the assumption a q-variate multivariate normal distribution the squared Mahalanobis distance is χ^2_q. This is by the very def of a multivariate normal dist.

It follows now that you can find outliers by checking if the Mahalanobis distance for a particular observation is way in the tail of the corresponding χ^2_q distribution.

In order to do so you can even inspect the QQ-plot.



Notice that such technique avoids the masking effect where you would not recognize outliers because they appears in clusters.

↳ Think in this sense about the dangers of using the Cook's Distance.

Finally, note that albeit in the case of not multivariate Distributions you might not be able to make very strong statistical conclusions about outliers high Mahalanobis distances would still point to potential outliers and such points would still be worth of inspection.

- Principal Component Analysis

The idea of Principal Component Analysis is to reduce the amount of variables used in an application by focussing on a few variables that at best sum up the information content of the original variables.

This solution might be especially useful to deal with the following issues:

1. Visualizing a large numbers of variables and make qualitative inferences based on the results.
2. To reduce computational bottlenecks. Sometimes processing a very large amount of variables might be computationally infeasible or too expensive.
   ↳ Reducing the amount of data to be processed while keeping the most information content might be a viable solution in such case.
3. To deal with collinearity that poses a major threat in many statistical techniques.

The question is now the following:

"How is it possible to achive to reduce the amount of variables used while keeping the most information?"

The solution proposed through PCA is the following:

Create variables $Y_1, \dots, Y_m$ being linear combinations of the original variables, such that the the variance of $X_1, \dots, X_m$ is maximal subject to

$$Y_1 \perp Y_2 \perp \dots \perp Y_m$$

variance of $Y_1, \dots, Y_m$ is maximal subject to

$$Y_1 \perp Y_2 \perp \dots \perp Y_m$$

Mathematically:

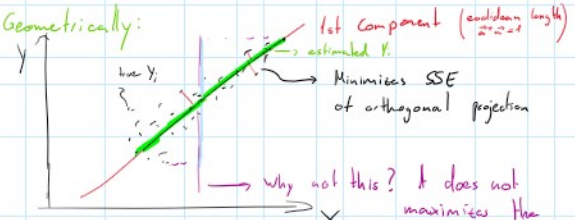
$$Y_1 = a_1 X_1 + a_2 X_2 + \dots + a_q X_q$$
$$\vdots$$
$$Y_m = m_1 X_1 + m_2 X_2 + \dots + m_q X_q$$

or in Matrix notation

$$\vec{Y}_i = \vec{a}_i^T \mathbb{X}, \text{ where}$$

$$\vec{a}_i = \operatorname{argmax} \vec{a}_i^T \Sigma \vec{a}_i \quad \text{s.t.} \quad \vec{a}_i^T \mathbb{X} \vec{a}_j \mathbb{X} = 0 \ \forall i \neq j.$$

Geometrically:



It is now clear that the above problem is a convex problem due to the properties of the variance matrix Σ.
It is moreover underspecified, as without a further constraint on the $\vec{a}_i$ you might always choose the coefficients to be as high as possible. Therefore we set the further constraint of the euclidean distance of the vector being 1.

We have therefore the final mathematical problem:

$$a_i = \underset{a}{\operatorname{argmax}} \ \vec{a}_i^T \Sigma \vec{a}_i$$

$$\text{s.t.} \quad \vec{a}_i^T \vec{a}_i = 1$$
$$\vec{a}_i^T \vec{a}_j = 0 \ \forall i \neq j$$

This can be solved via Lagrange:

$$\frac{\partial}{\partial a_i} \vec{a}_i^T \Sigma \vec{a}_i - \lambda_1 (\vec{a}_i^T \vec{a}_i - 1) - \lambda_2 \vec{a}_i^T \vec{a}_j = 0$$

$$2 \Sigma \vec{a}_i - 2 \lambda_1 \vec{a}_i - \lambda_2 \vec{a}_j = 0 \quad \text{(i)} \qquad (\lambda_2 \vec{a}_j \text{ because of (iii)})$$

$$\lambda_1 \vec{a}_i = 1 \quad \text{(ii)}$$

$$\lambda_2 \vec{a}_j = 0 \quad \text{(iii)}$$

So that $\boxed{\Sigma \vec{a}_i = \lambda_1 \vec{a}_1}$, where it is easy to see that this equation is a simple eigenvector equation.

The question on the choice of the eigenvector $\vec{a}_i$ associated with the eigenvalue λ_1 is then on which eigenvalue to choose.

The solution is given by transforming the above in our target quantity

$$\vec{a}_i^T \Sigma \vec{a}_i = \vec{a}_i^T \lambda_1 \vec{a}_i = \lambda_1 \underbrace{\vec{a}_i^T \vec{a}_i}_{=1 \text{ by def}} = \lambda_1$$

so that we choose the eigenvector $\vec{a}_i$ that corresponds to the highest eigenvalue.

=1 by def

so that we choose the eigenvector $\vec{a}$, that corresponds to the highest eigen value.

Computing the PCA loadings.

There are two possible ways to compute the loadings. Both rely on getting the Eigenvectors of the variance matrix Σ.

This can be done by leveraging:

(i) Eigendecomposition.

(ii) Singular Value Decomposition.

Eigendecomposition:

We are looking for a solution to the system:

$$A\vec{x} = \lambda\vec{x}$$

with $\vec{x} \neq \vec{0}$.

In order to get a solution we can therefore solve:

$$(A - \lambda I)\vec{x} = \vec{0}$$

The above is a homogenous system and we therefore look for a solution

$$(A - \lambda I) = \vec{0}$$

Given now the fact that the the determinant of a matrix equals 0; its columns are linearly dependent such that a non-trivial solution to a homogeneous system exists.

We therefore look in the above case for a solution

$$\det(A - \lambda I) = \vec{0}$$

Characteristic Polynom.

Solving the characteristic polynomial above you will get the desired eigenvalues and using the Eigenvector system:

$$A\vec{x} = \lambda\vec{x}$$

you can then insert your eigenvalues and get the desired eigenvectors.

Finally from the above it also holds that for a symmetric matrix A $(n \times n)$ it holds that with

$$\Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \lambda_n \end{pmatrix}$$

$$T = (\vec{x}_1, \vec{x}_2 \ldots, \vec{x}_n)$$

$$A \cdot T = T\Lambda$$

$$A = T\Lambda T^{-1}$$

And it furthermore holds for squared matrices A:

$$\det A = \det(T)\cdot\det(T^{-1})\cdot\det(\Lambda)$$
$$= \det(\Lambda) = \lambda_1\cdot\lambda_2\cdot\ldots\cdot\lambda_n$$

You can use the above to check the correctness of your computation.

Notice that the variance matrix is symmetric and thus the application of the above is straightforward.

As a final note it is possible to show that for the above case

$$\text{trace}(A) = \sum_{j=1}^{N}\lambda_j$$

In our case it holds

$$\text{trace}(\Sigma) = \sum_{j=1}^{N}\lambda_j$$

Moreover as $T^{-1} = T^T$ as the eigenvectors are orthogonal in the case of symmetric matrices,

Finally as it was proven before

$$Y_1 = \vec{a}_1\vec{X}$$

So that (As $\vec{X}$ mean centered in PCA)

$$\text{Var}(Y_1) = \vec{a}_1^T\vec{X}^T\vec{X}\vec{a}_1$$
$$= \vec{a}_1^T\Sigma\vec{a}_1 = \lambda_1\vec{a}_1^T\vec{a}_1$$

And therefore as due to orthogonality of the Principal Components

$$Y_1 \perp Y_2 \perp Y_3 \text{ etc.}$$

we have

$$\underbrace{\text{Var}(Y_1+Y_2+\ldots+Y_q)}_{\Sigma} = \text{Var}(Y_1)+\text{Var}(Y_2)+\ldots+\text{Var}(Y_q) = \lambda_1+\ldots+\lambda_q$$

It is then straight forward to appreciate that:

$$\frac{\sum_{j=1}^{m}\lambda_j}{\sum_{j=1}^{q}\lambda_j}$$

Share of total variance explained by the first m components

Finally note that if the observations fit exactly in m dimensions it would be indicated by the presence of q-m 0 Eigenvalues.

On the scale of the variables:

An important thing to notice is that PCA is not scale invariant. Moreover, the sign of the Eigenvalues are arbitrary as:

$$a\Sigma\vec{x} = a\lambda\vec{x}$$

It follows that the choice of the scale used has an important effect on the result.

If you have different variables on very different

It follows that the choice of the scale used has an important effect on the result.

If you have different variables on very different scales the one with the higher scale will likely display the highest variance and dominate the first principal component.

↳ This is why it is usually suggested that the principal components should only be extracted from the the sample covariance matrix when all of the original variables are roughly on the same scale.

A solution in this sense is to extract the principal components directly from the correlation matrix. This would then be equivalent to extracting the PC after each variable of the covariance matrix has been standardized to unit variance.

Finally note that there is no correspondence between the PC of Σ and the correlation matrix R. Hence using the latter is indeed an arbitrary decision where you give equal importance to all of the values.

Rules of thumb for number of PC selection:

When you reduce the dimensionality you keep $m < q$ PC that explain the most of the data.

The question is now how big m should be. There are three rules of thumb for selecting them:

1. $\frac{\sum_j^m \lambda_j}{\sum_j^q \lambda_j} \geq 0{,}8$; i.e. share of variance explained bigger equal than 80%.

2. λ_j: $\lambda_j \geq \frac{1}{q} \sum_j^q \lambda_j$; keep the variable if its variance is higher than the mean variance.

3. Plot λ_j vs j. Keep λ_j before you experience an elbow in the plot.

Kernel PCA:

The PCA encountered so far, imposed a linear relation between the PC and the origin variables.

Kernel PCA extends PCA by allowing to model a non-linear relation among the variables.

The basic idea is to create a richer set of variables $\tilde{X}$ by applying a number $\tilde{q}$ of non-linear transformations to the original variables X.

$$\tilde{X} = \{\phi_1(X), \phi_2(X), \dots, \phi_{\tilde{q}}(X)\}$$

Where you take $\tilde{q} > q$.

Then you continue as seen since here by computing the PC as:

$$Y_i = a_{1i}\phi_1(X) + a_{2i}\phi_2(X) + \dots + a_{\tilde{q}i}\phi_{\tilde{q}}(X)$$

with

$$Y_1 \perp Y_2 \perp Y_3 \perp Y_q$$

Albeit it would be theoretically possible to determine the PCA as above by following the schema:

Albeit it would be theoretically possible to determine the PCA as above by following the schema:

1. Transform $X \in \mathbb{R}^q$ into $\tilde{X} \in \mathbb{R}^{\tilde{q}}$ (and scale $\tilde{X}$).
2. Calculate the sample covariance $\hat{\Sigma}$ of $\tilde{X}$
3. Calculate the spectral decomposition:

$$\hat{\Sigma} = A^T \Lambda A$$

In practice this is not done but you rather use the kernel trick.

Idea it can be shown that

$$\hat{\Sigma}\,\vec{x} = \vec{\lambda}\,\vec{x}$$

is equivalent to solving

$$\tilde{K}\,\vec{v} = \vec{\lambda}\,\vec{v}$$

with:

$$\tilde{K} = \underline{\phi}(x)^T \underline{\phi}(y) \qquad X, Y \in \mathbb{R}^q$$

$$\underline{\phi}(x) = (\phi_1(x), \phi_2(x), \dots, \phi_{\tilde{q}}(x))$$

it is then possible to backtransform

$$\vec{v}, \vec{\lambda} \rightarrow \vec{\lambda}, \vec{x}.$$

The benefit of the approach lies in the computational benefit of working with

$$K = n \times \tilde{q} \cdot \tilde{q} \times n = n \times n \quad \text{instead of}$$

$$\Sigma = \tilde{q} \times \tilde{q}$$

and therefore you can work with extremely high dimensional transformations in a very high $\mathbb{R}^{\tilde{q}}$.

Multidimensional Scaling

The idea of Multidimensional Scaling is consists as well on representing points in hyperspaces of high dimension into a lower one.

Here the goal is the one of getting a representative plot in 2-3 dimensions. Then by inspecting the plot you can get meaningful hansight in the association among the variables in the hyperspace.

Albeit there might be a 1:1 relation with PCA as we will see soon, the approach is different.

While PCA acts dirctly on the multivariate matrix X, the idea of multidimensional scaling is the one of working on the information contained in the distance among the n data points in the hyperspace. Such information is summed up in proximity matrices.

The idea is then the one to find the best set of coordinates in the lower dimensional space $\mathbb{R}^m$ such that the distances among the n points respects at best the original hyperspace distances.

Interesting is moreover the case of Non-Metric Multidimensional Scaling. This allows, in contrast to PCA, to deal with a spatial low-dimensional representation of ordinal data

Interesting is moreover the case of Non-Metric Multidimensional Scaling. This allows, in contrast to PCA, to deal with a spatial low-dimensional representation of ordinal data as we will shortly see.

Classical Multidimensional Scaling - The Mathematics

The objective of MDS is to determine both the the dimensionality m of the lower dimensional space $\mathbb{R}^m$ as well as the m-dimensional coordinates $x_1, \dots, x_n$ so that the model gives a good fit for the observed proximities.

In classical multidimensional scaling the distance used to register in the proximity matrix is the euclidean distance.

The goal here is then to create a representation

$$X \in \mathbb{R}^m$$ for the n data points from the proximity matrix $$D \in \mathbb{R}^q$$ (q: dim orig. hyperspace.)

This is possible as we will shortly prove that it is possible to express:

$$B := X^T X \quad \text{in terms of} \quad D.$$

Given that it is possible to synthetize B from D and given B use spectral decomposition to get

$$B = A^T \Lambda A \quad \text{so that}$$

$$X = A^T \Lambda^{1/2}.$$

Intuition for expressing B in terms of D:

This will be shown in the case of $\mathbb{R}^2$. The generalization of it is straight.

Assume a matrix

$$X = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \quad \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \end{pmatrix}$$

and that

$$B = X^T X$$

$$= \begin{pmatrix} x_{11}^2 + x_{12}^2 & x_{11}x_{21} + x_{12}x_{22} \\ x_{21}x_{11} + x_{22}x_{12} & x_{21}^2 + x_{22}^2 \end{pmatrix}$$

Moreover for the distance among X row vectors, i.e. data points, we have

$$d_{12}^2 = (x_{11} - x_{21})^2 + (x_{12} - x_{22})^2$$

$$= x_{11}^2 + x_{21}^2 - 2x_{11}x_{21} + x_{12}^2 + x_{22}^2 - 2x_{12}x_{22}$$

$$= b_{11} + b_{22} - 2b_{12}$$

Where

b_{ij} comes from the i,j entry in the matrix B above so that

$$b_{ij} = x_{ik}x_{jk} + x_{ij}x_{ij} \qquad \forall\, i, j \in \{1, \dots, n\}$$

$$b_{ij} = x_{ii} x_{jik} + x_{ij} x_{jj} \qquad \forall\, i,j \in \{1,\dots,n\}$$

$$b_{ij} = \sum_{k=1}^{n} x_{ik} x_{jk}$$

Now as the location of your plot does not change the distance among the data we can set the location restriction:

$$\overline{x}_{ij} = 0 \quad \text{so that}$$

$$\sum_{k=1}^{n} x_{ik} = 0$$

And consequently:

$$\sum_{i=1}^{n} b_{ij} = \sum_{k=1}^{n} \underbrace{\sum_{i=1}^{n} x_{ik}}_{=0} x_{jk} = 0 \qquad \forall\, i$$

So that

$$\sum_{i=1}^{n} b_{ij} = \sum_{i=1}^{n} b_{ii} + b_{jj} - z_{ij}$$

$$= \sum_{i=1}^{n} b_{ii} + q\, b_{jj}$$

$$= \text{Trace}(B) + q\, b_{jj} \iff b_{jj} = \underbrace{\frac{1}{n}\sum_{i=1}^{n} b_{ij}}_{:= d_{\cdot j}} - \frac{\text{Trace}(B)}{q}$$

Similarly $\sum_{j=1}^{n} b_{ij} = \text{Trace}(B) + q\, b_{ii} \iff b_{ii} = \underbrace{\frac{1}{n}\sum_{j=1}^{n} b_{ij}}_{:= d_{i\cdot}} - \frac{\text{Trace}(B)}{q}$

And

$$\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} = \sum_{i=1}^{n} (\text{Trace}(B) + q\, b_{ii})$$

$$= 2n\,\text{Trace}(B) \iff 2\frac{\text{Trace}(B)}{n} = \underbrace{\frac{1}{n^2}\sum_{i}\sum_{j} b_{ij}}_{:= d_{\cdot\cdot}}$$

So that putting everything together we can rearrange

$$d_{ij}^2 = b_{ii} + b_{jj} - 2 b_{ij}$$

$$b_{ij} = \tfrac{1}{2}(d_{ij}^2 - b_{jj} - b_{ii})$$

$$= \tfrac{1}{2}\left(d_{ij}^2 - \left(d_{\cdot j} - \frac{\text{Trace}(B)}{n}\right) - \left(d_{i\cdot} - \frac{\text{Trace}(B)}{n}\right)\right)$$

$$= \tfrac{1}{2}\left(d_{ij}^2 - d_{\cdot j} - d_{i\cdot} + d_{\cdot\cdot}\right)$$

Given this derivation we can obtain multidimensional scaling by:

Step 1: Derive B from the proximity matrix D

Step 2: Obtain the spectral decomposition

$$B = A^T \Lambda A$$

Step 3:

Set

$$X = A^T \Lambda^{1/2}$$

Notice that when D $n \times q$ is of full rank then Λ will have $n-q$ zero eigenvalues so that you can represent B by

$$B = A_1^T \Lambda_1^{-1} A_1$$

where

A_1 = eigenvectors associated to $\lambda_1, \dots, \lambda_q$

Λ_1 = diagonal matrix with non-zero eigenvalues

non-zero eigenvalues

You can see now easily from the previous section that you can obtain a low-dimensional representation of your data points by using

$$X = A_n \Lambda_n^{1/2}$$

using the highest n Eigenvectors. Using all of the q Eigenvectors you will be able to recover the complete proximity Matrix D.

So far we have assumed the proximity matrix to represent euclidean distances. When this is the case B is positive definite and the result of Multidimensional Scaling corresponds to the one obtained by PCA.

However, when the proximity matrix is obtained using different distance metrics B might not be positive definit and you might get negative Eigenvalues and complex coodinates. If number of negative Eigenvalues is small a meaningful representation of the proximity matrix may still be possible using the Eigenvectors associated with the highest Eigenvalues.

Two rules of thumb in this sense are:

Trace criterion: Choose the number of coordinates so that the sum of the *positive* eigenvalues is approximately equal to the sum of *all* the eigenvalues.

Magnitude criterion: Accept as genuinely positive only those eigenvalues whose magnitude substantially exceeds that of the largest negative eigenvalue.

If the number of negative Eigenvalues is high some other methods than classical Multidimensional Scaling, such as non-metric Multidimensional Scaling might be advisable.

Non-metric Scaling:

When you have ordinal data that express preference the unit in which such preferences are expressed does not matter and a proximity matrix that focuses on some absolute distance metric does not make sense.

In non-metric Multidimensional Scaling we focus therefore on some representation that preserves the ordinal order:

$$\delta_{i_1 j_1} < \delta_{i_2 j_2} < \dots < \delta_{i_{\frac{n(n-1)}{2}} j_{\frac{n(n-1)}{2}}}$$

The idea is then that the lower dimensional coordinates of the data should not change when monotonically transforming the original data

$$f(\delta_{ij}), \text{ with } \frac{df(x)}{dx} > 0 \quad \forall x$$

This is achieved by finding a geometrical representation of your data points such that you minimize the following stress criterion:

$$S^2 = \frac{\sum_{i<j} (f(\delta_{ij}) - \tilde{\delta}_{ij})^2}{\sum_{i<j} \tilde{\delta}_{ij}^2}$$ controls for the scale of the distances.

Where

δ_{ij} = represent the original distances among the reordered data

$f(\delta_{ij})$ = is a monotone transformation.

$\tilde{\delta}_{ij}$ = are the fitted distances that result from the multidimensional scaling.

Non-Metric MDS works then by finding

(i) the monotonic transformation $f(.)$

(ii) that minimizes the stress criterion above.

In order to do that an iterative procedure is used where:

(i) it searches for $f(.)$ using monotonic regression

(ii) it minimizes the stress criterion and goes back to (i).

One of the major flaws with the above is that you might get clicked in local minima and do not find global solutions.

As a rule of thumb you can use the following:

- $S = 20\%$: poor
- $S = 10\%$: fair
- $S = 5\%$: good
- $S = 0$: perfect

Non-linear Techniques

ISOMAPS

So far we have only addressed linear scaling techniques. In fact, both PCA and Multidimensional scaling aims at representing the data points by applying rotations - linear operators - so that the new geometrical representation respects some desired properties - for instance having the axes in the direction of maximum variance.

It is thus clear that PCA and classical MDS just apply linear transformations and plot the data into linear subspaces.

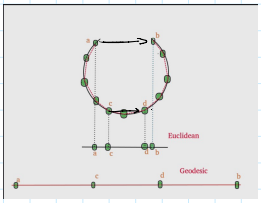
The question is now on how to accomodate some different structure. Our data might well not lie in to a linear space. Some different structure among the data might hold and metrics computed on the notion of a linear space might be flawed.

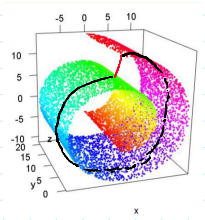
This section addresses such a case. It considers the option of using Manifolds to represent different local structures among the data so that it will be possible then to apply different transformation based on the derived manifold structure with a different focus than the linear transformations so far encountered.

For a quick and dirty intro to manifolds check at: https://bjlkeng.github.io/posts/manifolds/

Very good guy.

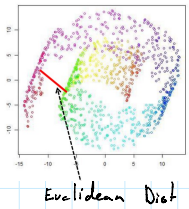
To further understand Isomap and to see the issues of relying on linear mappings when data lies on non-linear manifold consider the following examples:



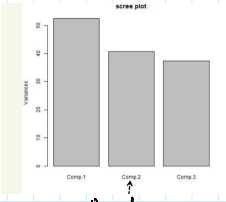


- Geodesic Distance
- Euclidean distance.

In both of the examples linear distances are highly misleading. Moreover when considering the right hand plot it is easy to see that a linear method such as PCA will hardly achieve an effective variance decomposition of the data.



Euclidean Dist



No drop in elbow plot. Two dimensions do not suffice to explain the higher dimensional variability in the data.

This is where ISOMAP chicks in. This is a technique that leverages the fact that a manifold locally resembles an Euclidean space such that "traditional" distance metrics can be computed.

Isomap works then by computing distance metrics between two points by approximating the global geodesic distance in the following way.

Step 0: Find neighbors of each point. This can be done:

- either by taking the K-nearest-neighbor
- or by taking all the points within radius ϵ.

Step 1: Form a weighted graph, where

- each point is connected with its neighbours.
- the weights are given by the euclidean distances among the points.

Step 2: Compute the graph distance between two points as the shortest path (in terms of edges weights) between the two points.

In such a way you effectively approximated the geodesic distance of your - possibly unknown - manifold.

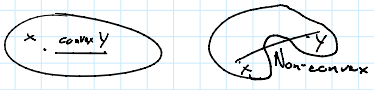
You can then compute above to obtain a full distance matrix and finally apply classical MDS on it.

In such a way ISO MAP is able to represent both the local and the global structure of your data and will therefore be able to retain the information of the two in lower dimensional representations.

both the local and the global structure of your data and will therefore be able to retain the information of the two in lower dimensional representations.

Important is moreover to stress the following when working with Isomaps.

- You must work on a convex space for extracting the local distances among neighbours. Otherwise you might jump over holes and short-circuit the true distance.



- The choice of the K-neighbours or the radius ε is of first order importance.
  - Too big K/ε will result in short circuits
  - Too small K/ε will lead to zig-zag roads considering the discrete amount of data
- Short-circuits might occur even if the manifold is too curvy or if the collected data-points are noisy.

T-SNE

This is a recent model for visualizing high-dimensional data that explicitely sets a high importance in preserving the local structure of the data. It ultimately allows for lower-dimensional plots preserving both the local structure of the data as well as the global one.

The method achieves that by the following schema:

- Compute the probability function for a data point x_i that states the probability of observing another one x_j assuming a gaussian kernel around x_i:

$$f(x_j) = \frac{1}{\sqrt{2\sigma_i^2\pi}} e^{-\frac{1}{2}\frac{(x_j-x_i)^2}{\sigma^2}}$$

- Compute the conditional probability:

$$P_{j|i} = \frac{\cancel{\frac{1}{\sqrt{2\sigma_i^2\pi}}} e^{-\frac{1}{2}\frac{(x_j-x_i)^2}{\sigma^2}}}{\sum_{k\neq i} \cancel{\frac{1}{\sqrt{2\sigma_i^2\pi}}} e^{-\frac{1}{2}\frac{(x_k-x_i)^2}{\sigma^2}}}$$

stating the probability of observing x_j as a neighbour of x_i among all of the x_k given the gaussian distribution of x_k centered around x_i.

- Compute the similarity metric:

$$P_{ij} := \frac{(P_{j|i}) + (P_{i|j})}{2N}$$

(Not symmetric)

This will then be compared to the lower dimensional similarity metric:

$$q_{ij} = \frac{f(\|y_j - y_i\|)}{\sum_{k\neq i} f(\|y_k - y_i\|)} \quad \text{with} \quad \underbrace{f(z) = \text{1-dim t-dist.}}_{\text{this for mitigating}}$$

$$q_{ij} = \frac{f(\|y_i - y_j\|)}{\sum_{k \neq l} f(\|y_k - y_l\|)} \quad \text{with} \quad \underbrace{f(z) = 1 - \tan h \, z^2}_{\text{this for mitigating the crowding problem in lower dimensions where you would get all of the points to lie close together without the occurrence of clusters.}}$$

The comparison is done via the Kullback-Leibler information criteria that would ultimately account for the preservation of both the local and the global structure of the data.

$$KL(P\|Q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

for when the neighbors are equally likely selected in the higher and lower dimension $KL = 0$.

When different. You scale σ_i^2 according to p_{ij} such that when

$$\|x_j - x_i\|^2 \gg 0 \Rightarrow p_{ij} \text{ is small}$$

(think of the gaussian kernel).

and if

$$\|x_j - x_i\|^2 \sim 0 \rightarrow p_{ij} \text{ is big so that you give higher weight to the } \log\left(\frac{p_{ij}}{q_{ij}}\right) \text{ if the local structure is not preserved.}$$

T-SNE works then by finding the lower dimensional representation that minimizes such Kullback-Leiber information criteria.

The question on how to set the σ_i^2 of the Gaussian kernel remains open. Of course setting a very high σ^2 will set a bigger weight on preserving the global structure while a smaller σ^2 will set higher weight on maintaining the local structure.

↳ In order to set such important parameter the authors of the method suggest to choose it indirectly by the choice of the perplexity measure.

The perplexity measure is an information measure that can be interpred roughly as:

> " this thing is right about as often as an x-sided dice, where x is the perplexity measure. and where right as a 6-sided dice means right 1/6 of the time. "

Mathematically it can be written as:

$$H(p(x)) = e^{\left(\sum_x p(x) \log_e 1/p(x)\right)}$$

i.e. a geometric average of the inverses of the probabilities.

Once you choose the perplexity parameter you choose σ_i so that this reached. The authors of the papers that presented the method advise on $H(p(x)) \in [5, 50]$.

Clustering

The goal of clustering is the one of creating groups of observations such that

- Elements within the cluster are similar
- Elements among different clusters are different.

This notes will treat:

1. Agglomerative Clustering: Distance-Based Aggregation
2. Divisive Clustering (Partitioning): K-Nearest Neighbours
3. Model Based: Finite Mixture Models: Gaussian Mixture Models
4. Density Based Clustering

Before going into the nitty gritty, a brief comparison of the methods:

1. Agglomerative Clustering:

- Idea: start with N-# obs clusters. Iteratively merge the two nearest clusters based on some distance metric.

2. Divisive Clustering:

- Idea: Given an a-priori defined number of clusters, find the best topological position for the cluster center such that some overall cluster distance metric will be minimized in relation to such center.

3. Model Based Cluster

- Idea: Data are generated from different probability distributions. The number of probability distributions as well as their respective parameters are defined minimizing some information criteria measures such as the Bayesian Information Criteria

4. Density Based Clustering: here the assumption is that clusters are characterized by having an high amount of data density in the cluster region. And no or little data within clusters regions. You find then the clusters by specifying two hyperparameters that will be discussed later by a heuristic argument that is as intuitive as elegant.

1. Agglomerative Clustering: This algorithm here is rather simple and the details are omitted.

Important is however to understand that the choice of the distance metric by which the closest clusters are merged.

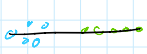
Some of the possible choices are:

1.1 Single linkage ⇒ minimal distance among two data-points pairs in different clusters.

$d_{AB} = \min_{i \in A, j \in B} d_{ij}$

two clusters

1.2 Complete linkage: ⇒ maximal distance among two datapoints pairs coming from the two different clusters



$$d_{AB} = \max_{i \in A, j \in B} d_{ij}$$

(1.3) Average - linkage: ⇔ Average among all of the datapairs distances among the two datasets

$$d_{AB} = \frac{1}{N+M} \sum_{i}^{N} \sum_{j}^{M} d_{ij}$$

(1.4) Ward's Method: ⇔ works by leveraging the notion of within group sum of squares defined as follows:

$$WGSS = \sum_{i \in C}^{N_c} \| x_i - \bar{x}_c \|^2$$

simply the sum of the L^2 distance for the points within a cluster.

The Wards Method defines the distance between two clusters as follows

$$d_{AB} = WGSS_{AB} - (WGSS_A + WGSS_B)$$

It is clear that different methods are more/less robust against outliers and have in general bigger merits/flaws. Notice, that the distance/dissimilarity metric can be chosen as one of the many and has not to be confined to the usual euclidean one.

(2.) Divisive Clustering: K-NN.

This method consists in finding the cluster centers that minimizes the WGSS given the fixed amount of chosen clusters.

To find the culsters centroids is computationally infeasible. Hence, an approximation algorithm is used: the Loyd's algorithm.

This works as follows:

1. start with k random centroids for the k clusters
2. Assign each datapoint to the center that is closest given some distance metric
3. Recompute a new center for the cluster as the average of the observations assigned to the cluster in (2).
4. Stop if the cluster mean did not change much. Go back to (2.) otherwise.

As there is no guarantee that the Loyd's algorithm will reach the global optimum it is usual to compute the KNN multiple times and observe how the results change.

The question about the clusters centroids remain open. How should such center be determined? There are essentially two options:

1. Means: Here the center of the cluster does not have to be an observed datapoint. It is simply the average of all points.
2. Medoid: Here the center is defined as the observation for which the distance/dissimilarity to all other datapoints is minimized. This is more robust toward outliers. However it is computationally more demanding.

③ Model Based Clustering: Finite Mixture Models

So far clustering was based on heuristics based on some distance metric.

Another option consits on specify a full statistical distribution for our data.

The idea as mentioned above is to specify different PDFs and assign the different datapoints to the distribution they belong to with the highest posterior probability.

The idea works as follows using different gaussian distributions with different parameterization to express the different clusters and to express the probability of observing a datapoint as the mixture of different clusters.

$$f(x) = \underbrace{\sum_{j=1}^{k} p_j \, g(x \mid \theta_j)}_{\text{gaussian finite mixture with } \sum_j p_j = 1}$$

- $f(x)$: the probability of observing x.
- k: number of clusters
- p_j: probability of observing x given cluster j parameterization θ_j

Given the above setting it is clear that once the number of clusters / mixture distributions K and their parameterization has been defined, you can classify observation x_i by means of bayesian arguments as:

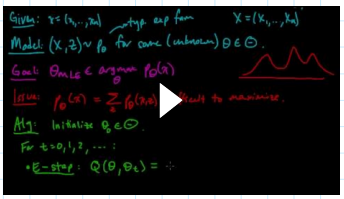
$$\{ x_i \in \theta_j \;:\; P(x \in \theta_j \mid x_i) := \frac{\overbrace{p_j \cdot g(x_i \mid \theta_j)}^{= P(x_i \cap \theta_j)}}{\underbrace{\sum_j p_j \cdot g(x_i \mid \theta_j)}_{\text{posterior of observing } x \text{ given } x_i}} > P(x \in \theta_{1 \neq j} \mid x_i) \}$$

The central question now is on how to define the number of clusters K and the parametrization of the finite mixture distributions:

$$p_j \quad \wedge \quad \theta_j$$

This is achieved by means of the Expectation - Maximization algorithm. To understand that watch the ten videos following

[(ML 16.3) Expectation-Maximization (EM) algorithm]



There you can get as well the intuition as well as the details for the Algorithm.

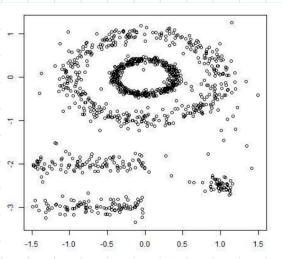
Moreover to make the optimization problem more tractable it is common practice to impose restrictions on the covariance structure of the different gaussian distributions. Finally to bound the solution the parameters are chosen minimizing some information criteria as a pure maximization of

$$f(x) = \sum_{j=1}^{K} p_j \, g(x \mid \theta_j)$$

will always increase in the number of selected clusters K.

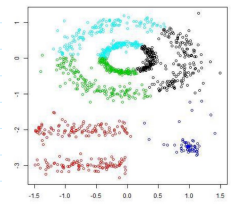
④ Density Based Clustering

This was introduced in the slides of the lecture in a very interesting way. Consider the following non-convex clusters scenes.

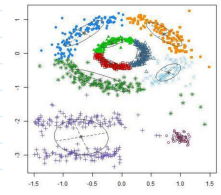


Here the four clusters can be easily recognized by humans through eyeballing. However, the mathematical constructs so far seen fail brutly when working on non-convex structures:

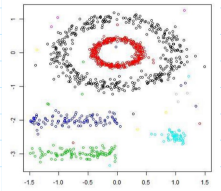
K-means



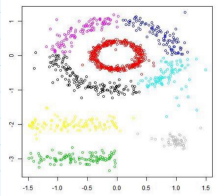
Gaussian Mixture



Agglomerative (Single Linkage)



Agglomerative (Average linkage)



DBSCAN - density based spatial clustering of applications with noise - works now as defining clusters as groups of high-density points.

↳ Different regions with a high density of points will then be classified as different clusters.

The idea works as follows:

- Define a radius ε
- Define $M \in \mathbb{N}$ ⇒ minimum number of point necessary for the definition of a core-point.

Given the two parameters you can then label three types of different points:

1. Core Points
2. Border Points
3. Noise Points.

These three categories are defined as follows; for each point check at the # of points in their neighborhood.

Then:

- If the number of points in the neighborhood is $\geq M$ then classify it as a core point.

Given then your core points:

is $\geq M$ then classify it as a core point.

Given then your core points:

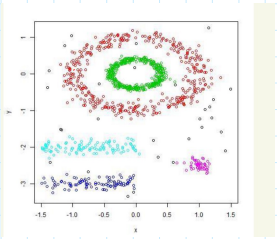
- Classify points in the neighborhood of core points that are themself not classified as core points as border points.

Classify points that are neither core nor border points as noise-points. These might be then further analyzed by looking whether they are outliers or not.

Finally you would classify a cluster as follows

- Connect all the core points that lie within distance ϵ with an edge to each other. The connected core points will form a cluster.
- Assign border points to one of the cluster they belong to.

The result for DBSCAN would look as follows in the example before:



The open question that stays is on how to deal with the hyperparameter selection.

If:

- ϵ too big and M too small then all of the points are labeled as core-points.
- if ϵ too small and M too big all noise points

Some way to select data points:

Possible **strategy for finding tuning parameters**: look at the *k*-nearest neighbor distance (***k*-NN distance**), the distance from a point to its *k*-nearest neighbor.

1. Pick an initial k. Rule of thumb $k \geq d + 1$, where d is the dimension of the data. For $d = 2$, $k = 4$ often works well.
2. Compute the *k*-NN distance for each point.
3. Sort them in increasing order and plot them.
4. Look for sharp increase, and choose this distance as ε and the k as *MinPts*. If there is no "elbow", change k.

There are then a few ways to interpret the clusters and one quantitative metric to analyze its results:

To interpret:

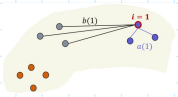
1. Analyze the centroids or representative center of the cluster (this especially for method ① - ③). What are the characteristics of the data?
2. Plot in lower dimension via PCA. Look at the linear combination of the axis. What do they represent? Where do the cluster lie w.r.t. these representative axis?

cluster the i w.r.t. these representative axis!

To quantify the goodness of the clustering:

Consider the silhouette $S(i) \in [-1; 1]$.
This is defined as:

$$S(i) = \frac{b(i) - a(i)}{\max(b(i), a(i))}$$



where:

$b(i)$ = minimal average distance between point i and all of the points belonging to another cluster

$a(i)$ = average distance of point i and all of the other points belonging to the cluster.

It is now clear that for a high silhouette the clustering performed well for the point. For silhouette ~ 0 the clustering was ok. For silhouette < 0 the point was likely badly classified.

Explainatory Factor Analysis

This technique deals with the situation where we observe some manifest variable and based on that we try to make inference on an unknown latent variable.

The situation can be formulated mathematically as:

$$\begin{aligned} X_1 &= \lambda_{11} f_1 + \cdots + \lambda_{1k} f_k + u_1 \\ X_2 &= \lambda_{21} f_1 + \cdots + \lambda_{2k} f_k + u_2 \\ &\cdots \\ X_q &= \lambda_{q1} f_1 + \cdots + \lambda_{qk} f_k + u_q \end{aligned}$$

or in Matrix notation:

$$\vec{X} = \Lambda \vec{f} + \vec{v}$$

where the major difference lie on the fact that $\vec{f}$ represent the latent or observed factors so that it is not possible to estimate the factor loadings $:= \Lambda$ by means of LS.

It is now clear that without posing some structure on the relation above it will be hardly possible to make inference on the factors and their loadings from the manifest variables.

Such structure can be posed in the following way:

- $E(v) = 0$ $\quad Cov(\vec{v}) = \Psi$, with Ψ being a diagonal matrix, meaning that the error terms are uncorrelated.
- $Cov(\vec{v}, \vec{f}) = 0_{q,k}$ being the zero Matrix such that the factors and the error terms are uncorrelated.
- $E(X) = \vec{0}$, mean centered manifest variables
- $E(f) = 0$, $Cov(f) = I$, so that factors are standardized variables.

It follows now that given such structure there is a 1:1 relation among the manifest variables covariance structure and the factor loadings as:

$$Cov(X_1, \dots, X_q) = \Sigma$$

$$\begin{aligned} \Sigma &= Cov(\Lambda \vec{f} + \vec{u}) \\ &= Cov(\Lambda \vec{f}) + Cov(\vec{u}) \\ &= \Lambda^T Cov(\vec{f}) \Lambda + \Psi \\ &= \Lambda^T \Lambda + \Psi \end{aligned}$$

(as $\vec{f}$ and $\vec{u}$ uncorr.)

So that

$$\sigma_j^2 = \sum_{i=1}^{k} \lambda_{ji}^2 + \Psi_j$$

and it is possible to see that the variance is decomposed into an a:

- common term $h_j^2 = \sum_{i=1}^{k} \lambda_{ji}^2$

and a

- specific term Ψ_j

The question consists now on obtaining some estimates for the factor loadings above by leveraging the above decomposition, where in order to do so we have to fix first:

- the number of latent factors k.

In order to do that we will explore two different methods:

1. Principal Factor Analysis.
2. Maximum Likelihood Estimation.

1. Principal Factor Analysis:

This is based on an iterative Algorithm to determine the factor loadings and the variable specific variance Ψ

The issue is namely the following: given the determined K based on the observation Σ you will need to be able to discern

$$\Lambda^T \Lambda \quad \text{and} \quad \Psi.$$

Now given one of Λ and Ψ you could determine the other. However in the specific situation you do not know neither of the two.

The idea of Principal Factor Analysis is then to mediate the issue by:

1. making an educated guess of $\hat{\Psi}$
2. determine Λ via

$$\Lambda^T \Lambda = \Sigma - \Psi$$

i.e. by finding the spectral decomposition of

$$\underbrace{\Sigma - \Psi}_{\text{symmetric}} = A^T \Phi A$$

And set $\Lambda = A^T \Phi^{1/2}$

3. Compute Ψ
4. Iterate (2.) - (3.) until convergence.

The educated guess on Ψ is then based on the following reasoning:

- First the Factor Analysis is scale invariant as:

$Y = CX$ where C is a diagonal matrix representing the scaled X.

it is then true that:

$$\begin{aligned} Var(Y) &= Var(CX) \\ &= C^T Var(X) C \\ &= C^T (\Lambda^T \Lambda + \Psi) C \\ &= C^T \Lambda^T \Lambda C + C^T \Psi C \\ &= \tilde{\Lambda}^T \tilde{\Lambda} + \tilde{\Psi} \end{aligned}$$

with: $\tilde{\Lambda} = \Lambda C$, $\tilde{\Psi} = C \Psi C$

So that applying the factor model to X and then scaling the loadings and the specific variance is the same as scaling X first and then applying the factor model to Y.

It is then possible to see that instead of the covariance matrix Σ of the manifest data you can use the correlation matrix R thereby having

$$R = \tilde{\Lambda}^T \tilde{\Lambda} + \tilde{\Psi}$$

$$\rho_j^2 = 1 = \sum_{l=1}^{K} \tilde{\lambda}_{jl} + \tilde{\psi}_j$$

So that you can obtain the estimates of $\tilde{\Psi}$, compute $\tilde{\Lambda}$ and finally scale the two back; where the initial estimates $\tilde{\Psi}$ are

$$\tilde{\Psi} = 1 - \sum_{l=1}^{K} \tilde{\lambda}_{jl}, \quad \text{where}$$

$$\sum_{l=1}^{K} \tilde{\lambda}_{jl} = \begin{cases} \text{squared of correlation coefficient of manifest variable } j\text{th with all the other manifest variables} \\ \text{or} \\ \text{largest correlation coeff of the manifest variable } j\text{th and the other variables} \end{cases}$$

2.1 Maximum Likelihood Estimator

Another possibility when estimating the factor loadings is the one of relying on MLE and its asymptotic properties.

The idea works as follows:

- Assume a multivariate normal distribution of the manifest data

Then under this assumption and the factor analysis model assumptions you can derive the Likelihood:

of the manifest data

Then under this assumption and the factor analysis model assumptions you can derive the likelihood:

$$L(\Psi, \Lambda) = -\frac{N}{2}\left(\log|2\pi\Sigma| + \text{trace}\left(\Sigma^{-1}S\right)\right)$$

(S: sample covariance matrix of x_{ij})

It is then possible to find the arguments Ψ, Λ that maximizes the above.

Notice that a sensible advantage of using MLE for estimating the factor loadings is that you can test for the hypothesis of the goodness of fit of k factors that result by comparing the restricted and unrestricted likelihoods which follow a chi-squared distribution. When you persistently fail to reject some restricted version with k being increased at each run, then there is evidence that the model fails to represent the data or that your sample is too large.

Notice finally that once the factor loadings are obtained it is common practice to apply factor rotation to them to foster interpretability of the factors.

Rotation does not in fact alter the structure of factor analysis, which is in fact invariant to such transformations.

To see that consider the rotation M, so that $MM^T = I$. It follows that with

$$f = M^T f \qquad \tilde{\Lambda} = \Lambda M$$

So that:

$$X = \Lambda M M^T f + \vec{v} = \Lambda f + \vec{v}$$

and

$$\Sigma = \Lambda M M^T \Lambda^T + \Psi = \Lambda\Lambda^T + \Psi$$

So practically this means that you have to set some restriction at first to get a solution; you can read about in the book - the gist is however to set the restriction

$$G = \Lambda^T \Psi^{-1} \Lambda$$

to be diagonal with its element sorted in decreasing order, leading to a factors structure where the first factor explains the most of the common variation of X, the second the most being uncorrelated to the first effects. in a similar reasoning to the PCA.

In order to facilitate the interpretation of the factors obtained with the above restriction; two types of rotation are possible

- orthogonal rotation: which preserves the uncorrelated structure of the factors as above
- oblique rotation: frees up the uncorrelated factors restriction.

Given the two it is clear that to facilitate the interpretation we need to find rotations such that ideally:

- Each variable is loaded on at most one factor
- All factor loadings are either large and positive or close to zero.

Two examples of reaching that consists in:

- Varimax rotation (a type of orthogonal rotation; few large and many small loadings)
- Promax rotation (oblique rotation that results in a low correlation among factors).

Notice finally how in the case of orthogonal rotations, the factor loadings can be interpreted as the correlation coefficients in the case of factor analysis with scaled manifest variables.

Proof:

$$\begin{aligned} Corr(X, f) &= Corr(\tilde{\Lambda} f + u, f) \\ &= Corr(\tilde{\Lambda} f, f) + \cancel{Corr(u, f)} \\ &= \tilde{\Lambda} \cdot \underbrace{Corr(f, f)}_{=1} \end{aligned}$$

Classification

The general problem of this chapter is of the following form; given a set of features X you have to classify a variable y.

There are two possibilities in such a case:

(I) Directly model $Pr(Y=j \mid X=x)$

(II) Use a soft bayesian approach modeling first $Pr(X=x \mid Y=y)$ and then using Bayes theorem to obtain $Pr(Y=j \mid X=x)$.
The soft -ajective used before refers to the fact that you will not use a rigorous approach when defining a prior; in the sense that you will not use a non-informative prior.

On the second approach: LDA and QDA

The general approach of the two techniques belonging to the method is then, essentially the following:

1. Assign a prior P_j that each variable belong to a class
2. Assume that the feature variables are multinomial distributed conditional on observing the categories y; i.e.
$$X \mid y_j \sim N(\mu_j, \Sigma_j) := g(X, \vec{\theta}_j) \text{ for all classes } j.$$

Such that for the model above, the unconditional PDF of X is given by:

$$Pr(X) = \sum_{j=1}^{K} P_j \cdot g(X, \vec{\theta}_j)$$

↳ marginalizing.

By applying Bayes Theorem it is now possible to see that

$$\begin{aligned} Pr(Y=j \mid X=x) &= \frac{Pr(X=x \mid Y=j) \cdot Pr(Y=j)}{Pr(X=x)} \\ &= \frac{g(X, \vec{\theta}_j) \cdot \overbrace{Pr(Y=j)}^{\text{Prior}}}{\sum_j P_j \, g(X, \vec{\theta}_j)} \end{aligned}$$

The classification is then chosen as

$$\underset{j}{\text{argmax}}\, Pr(Y=j \mid X=x) = \underset{j}{\text{argmax}}\, \log(g(X, \vec{\theta}_j)) + \log(Pr(Y=j)) - \log\left(\sum_j P_j \, g(X, \vec{\theta}_j)\right)$$

Not dependent on j

Given then the multivariate normal assumption it is clear that:

$$g(X, \vec{\theta}_j) = \frac{1}{\sqrt{(2\pi)^k |\Sigma_j|}} \cdot \exp\left(-\tfrac{1}{2}(X-\mu_j)^T \Sigma_j^{-1} (X-\mu_j)\right)$$

So that it follows:

$$\underset{j}{\text{argmax}}\ Pr(Y=j \mid X=x) = -\log(\Sigma_j) - \tfrac{1}{2}\left((X-\mu_j)^T \Sigma_j^{-1} (X-\mu_j)\right) + \log(Pr(Y=j))$$

This is in fact called quadratic discriminant analysis due to the quadratic term $(X-\mu_j)^T \Sigma_j^{-1} (X-\mu_j)$.

A final question is on how to estimate the necessary parameters of the model: Σ_j; $Pr(Y=j)$, μ_j.

These are estimated as:

- $Pr(Y=j)$ = fractions of observations belonging to j (recall that we are here in supervised learning domain).
- μ_j = sample mean $\bar{x}_j$ of all observations belonging to j
- Σ_j = sample covariance S_j

A problem that might arise with the Quadratic Discriminant Analysis is that if the feature space is particularly large the number of parameters to be estimated might be massive. Think for instance at the sample covariance Σ_j and how its size increase in the features dimension. In such case if the available dataset is not too large you might quickly run into issues.

To obviate such problem a more restrictive model is proposed - the - Linear Discriminant Analysis:

This consists of the same approach presented before with the only difference of restricting the covariance matrix to be equal across the classes j. So that $Var(X) = \Sigma \ \forall j$.

Inserting this in the expression above

$$\underset{j}{\text{argmax}}\ Pr(Y=j \mid X=x) = \log(Pr(Y=j)) + x^T \Sigma^{-1} \mu_j - \tfrac{1}{2} \mu_j^T \Sigma^{-1} \mu_j$$

Summing up:

LDA	QDA
+ Only few parameters to estimate; accurate estimates	- Many parameters to estimate; potentially less accurate estimates
- Less flexible (linear decision boundary)	+ More flexible (quadratic decision boundary)

On Naive Bayes: Note that this method widely used in Machine Learning is nothing more than a special very restrictive form of QDA, where you have:

$$\Sigma_j = \text{diag}(\sigma_{1j}^2, \ldots, \sigma_{kj}^2), \text{ where } k = \# \text{ features.}$$

so that Naive Bayes assumes uncorrelation among the different features.

On method 1: Directly modeling $Pr(Y=j)$ through logistic regression

As discussed the aim of the logistic regression is to model directly the quantity of interest

$$Pr(Y=j \mid X=x)$$

This is done through the inverse-logit function. Recall the logic of it with two-categories case. The logit is a GLM, i.e. it tries to model:

$$\underbrace{g(E(y|x))}_{\text{link function}} = \underbrace{\beta_0 + \beta_1 x_1 + \dots + \beta_k x_k}_{\text{linear model}}$$

In the case of a two-classes case:

$$E(y|x) = Pr(y=1|x) \cdot 1 + 0 \cdot Pr(y=0|x)$$

Notice now that to model

$$Pr(Y=1|x) = \beta_0 + \beta_1 x_1 + \dots$$ does not make sense as you might well end up with $Pr < 0$ and $Pr > 1$.

The solution is to use the link function: $g(x) = \log\left(\frac{x}{1-x}\right)$

this allows to model the log-odd ratio in a linear way through the linear model so that by rearranging:

$$\log\left(\frac{Pr(Y=1|X=x)}{1 - Pr(Y=1|X=x)}\right) = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

$$\Leftrightarrow$$

$$\frac{Pr(Y=1|X=x)}{1 - Pr(Y=1|X=x)} = e^{\beta_0 + \beta_1 x_1 + \dots + \beta_k x_k}$$

$$Pr(Y=1|X=x) = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_k x_k}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_k x_k}}$$ } cannot be negative; bounded to 1.

It is now clear that you can assign the class according to the maximum $Pr(Y=1|X=x)$ so far obtained.

An interesting note moreover is the one of looking at differences between LDA and the logistic model. In fact it is straightforward to see that the underlying model of the two is essentially the same as:

LDA:

$$\log\left(\frac{Pr(Y=1|X=x)}{Pr(Y=0|X=x)}\right) = \underbrace{\log\left(\frac{p_0}{p_1}\right) - \frac{1}{2}(\mu_0 + \mu_1)^T \Sigma^{-1}(\mu_1 - \mu_0)}_{\beta_0} + \underbrace{x^T \Sigma^{-1}(\mu_1 - \mu_0)}_{\vec{\beta}}$$

Logit:

$$\log\left(\frac{Pr(Y=1|X=x)}{Pr(Y=0|X=x)}\right) = \beta_0 + \beta^T X$$

However, despite the similar underlying model the LDA model estimates its parameter in a much less efficient way as it maximizes:

$$\prod_i p(x_i, y_i) = \prod_i \underbrace{p(x_i|y_i)}_{\text{gaussian}} \underbrace{p(y_i)}_{\text{bernoulli}}$$

while in the logit model

$$\prod_i p(x_i, y_i) = \underbrace{\prod_i p(y_i|x)}_{\text{Bernoulli-logit}} \cdot \underbrace{p(x)}_{\text{ignored}}$$

the conditional probability is maximized directly and involves therefore less assumptions staying more flexible.

Fisher Discriminant Analysis

In contrast to the previously seen methods which rely on statistical methods, Fisher discriminant analysis is another important classification technique that relies on heuristic methods.

The idea is the one of creating a linear combination separating the classes in

discriminant analysis is another important classification technique that relies on heuristic methods.

The idea is the one of creating a linear combination separating the classes in such a way that the Rayleigh quotient - i.e. the ratio among the between-class variance and the within class variance is maximized.

Such vector $\vec{a}$ can be calculated:

$$\underset{\vec{a}}{\operatorname{argmax}} \frac{BCV}{WCV} = \frac{\vec{a}^T B \vec{a}}{\vec{a} W \vec{a}},$$

where B = between-group sample variance

W = within-group sample variance

It is then possible to show that such maximum is given by the eigenvector $W^{-1}B$ corresponding to the largest eigenvalue.

Regression Tree and Random Forrest

This chapter explores tree-based classification and regression models. The idea of the model is to partition the space according to the features characteristics in a data driven way, and according to the partition space fit a local regression or make a classification decision.

A major point drawing the line between plain vanilla regression trees and random forests is the way to which the two models deal with overfitting. As we will soon see the plain vanilla regression/classification trees will use cross-validation for pruning the tree according to a cost-complexity criterion similar in spirit to the information criteria; in contrast random forests leverage the idea of MSE reduction by aggregating - ideally uncorrelated - trees with a reduction of the variance and an increase of the bias - i.e. a general reduction of overfitting.

Regression And Classification Trees:

Both rely on finding the partitioned feature space that leads to the best classification and regression fit.

In order to define such best possible fit some quantitative metrics have to be defined. In the case of the:

- Regression Tree:
  - → you fit according to the local average of your predictor variable, y_i.

$$\hat{c}_m = ave(y_i | x_i \in R_m) = \frac{\sum_{i:x_i \in R_m} y_i}{N_m}$$

- $N_m = \#\{x_i \in R_m\}$: number of observations in R_m

  - → you estimate the goodness of the fit according to the well known residual sum of squares.

- Classification Trees
  - → You classify according to the probability of each class in the end nodes.

    Such probability is given by the share of each class in each node. The classification decision is then based on the most probable class.

- **Proportion of class k observations in node m**

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{i:x_i \in R_m} 1_{(y_i = k)}$$

- **Majority vote**: predict class $k(m)$ for which $\hat{p}_{mk}$ is largest, i.e.,

$$k(m) = \text{argmax}_k \hat{p}_{mk}.$$

→ You then classify the goodness of the fit according to a classification error measure. Some 3 well known measures are:

- Misclassification Error:

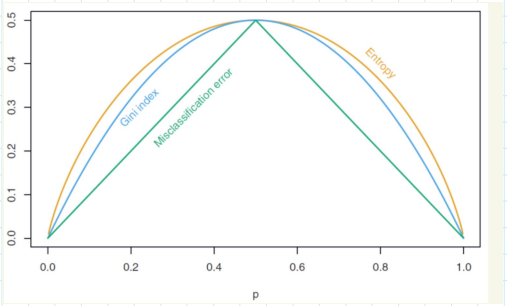
share of misclassified $\left\{ \frac{1}{N_m} \sum_{i:x_i \in R_m} 1_{(y_i \neq k(m))} = 1 - \hat{p}_{mk(m)} \right.$

- Gini Index:

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk} (1 - \hat{p}_{mk}) \quad \text{MCE}$$

- Cross-entropy or deviance:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$



Given such characterization of the goodness of fit it is possible to understand the data driven partitioning of the feature space maximizing the fit:

1. Start from the trivial partition $R = \{\mathbb{R}^p\}$
2. For a variable / feature j create two partitions

$$R_1(j,s) = \{X \mid X_j \leq s\} \quad \text{and} \quad R_2(j,s) = \{X \mid X_j > s\}$$

Choose the splitting point such that the goodness of fit defined above is maximized.

3. Iterate (2.) until a stopping criterion is reached - for instance minimal number of obs. in one node.

⚠ Important Note: Such algorithms are greedy in the sense that once a partition has been selected there is no modification of it deeper in the tree. This would be computationally infeasible.

Obviously an algorithm as above will lead to a strong overfit. In order to control for it the usual applied technique is the one of the following cost-complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

where:

α = complexity penalization coefficient. This is chosen through cross-validation.

N_m = # observations in partition m.

$|T|$ = length of the tree (number of leaves).

Q_m = goodness of fit for partition m.

Summing it up:

1. Use **greedy algorithm to grow a large tree.**
2. Apply **cost complexity pruning** to the large tree in order to obtain the best subtree for each α.[1]
3. **Use cross-validation to choose α.** I.e., divide the data into training and test data sets. Then, for each division:
   - (a) Repeat steps 1 and 2 on the training data.
   - (b) Evaluate the mean squared prediction error on the test data for each α.
   - Pick the α which minimizes the average out-of-sample error on all test sets.
4. **Return the subtree** from step 2 that corresponds to the value of α chosen in step 3.

[1] Only a finite number of α's needs to be considered since there are only finitely many subtrees. Moreover, one can show that the subtrees obtained in step 2. are nested.

Random Trees:

This is essentially the same procedure of standard regression trees with the difference that N different trees are grown according to a data bootstrapping procedure. The prediction is then based on an ensamble - aggregation - of the individual trees:

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Important is that with this second technique we do not prune.

The idea is in fact the one of reducing the MSE by reducing the variance as previously mentioned. It is in fact possible to see that through the aggregation the variance of the predictor is:

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} T_b(x)\right) = \frac{1}{B^2}\left(\sum_{b=1}^{B}\text{Var}(T_b(x)) + \sum_{\substack{i=1\\ i\neq j}}^{B}\sum_{\substack{j=1\\ i\neq j}}^{B}\text{Cov}(T_i(x), T_j(x))\right)$$

$$= \frac{1}{B^2}\sum_{b=1}^{B}\sigma^2 + \sum_{\substack{i=1\\ i\neq j}}^{B}\sum_{\substack{j=1\\ i\neq j}}^{B} \rho\cdot\sigma^2 \quad \rightarrow \text{corr.}$$

$$= \frac{1}{B^2}\left(B\sigma^2 + (B-1)B\,\rho\cdot\sigma^2\right)$$

$$= \frac{1}{B}\left(\sigma^2 + B\rho\sigma^2 - \rho\sigma^2\right)$$

$$= \rho\cdot\sigma^2 + \frac{\sigma^2(1-\rho)}{B}$$

So that it is clear that the lower the correlation ρ among the trees the lower the variance. This is in fact the reason why for each tree you select randomly m features / variables.

Estimating generalized error for the Random Forest can be done by averaging over the out-of-bag error for each tree.

The idea is the one of splitting the dataset into training and testing sets. Then you would obtain the dataset for growing the trees based on the bootstrapped training dataset.

Then you would obtain the dataset for growing the tree based on the bootstrapped training dataset.

Based on that you would then obtain the out-of-bag error by looking at the share of testing set observations that were correctly classified!

An important field of research in the area of random forests is moreover the one of determing the relevance of a particular variable / feature for the classification case. Two major approaches were proposed:

1. Mess up with a particular variable for which you want to get the importance.
   - ↳ Understand how big the impact of messing up with this variable was.

   That is more concretely: Permute variable i across all of the trees. Then get the new out-of-bag error after the permutation.

   Look at the average out-of-bag error increase after the permutation.

   The bigger the error increase the more important the variable.

   To compare the effect among different variables you can then look at the average out-of-bag error rate corrected by the standard deviation.
2. While growing the tree keep track of the decrease in impurity measure (be it RSS, Gini, MCE etc.) after each variable split.
   - ↳ This gives an indication on the variable importance.

Random and Fixed Effects Models

The idea of random effects models is the one of information pooling. The idea here is that when dealing with panel data, i.e. data collected in a cross-sectional way over a span of time.

Then one approach for dealing with the dataset is to estimate a fixed effect model of the form:

$$\vec{y}_t = \vec{\beta}_0 + \beta^T X_t + \vec{\varepsilon}$$

where you could set:

$$\vec{\beta}_0 = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_0 \end{pmatrix}$$

, i.e. a universal intercept for all the cross-sectional cases i.

Or in the case you might want to specify a different intercept for all of the states; this might even be necessary as if the underlying model requires one and you do not specify it you get correlated models and therefore you would get a biased model.

$$\vec{\beta}_0 = \begin{pmatrix} \beta_{0,1} \\ \vdots \\ \beta_{0,n} \end{pmatrix} \qquad i = 1, \dots, n \text{ states.}$$

Important to see is that in such a model you do not put any constraints on the intercepts $\beta_{0,i}$ and each one might differ sensibly from the other.

- ↳ Such an approach is therefore highly flexible but might suffer sensibly in the case of many different states i and just a few observations across time for them.
  - ↳ In such a case the estimation of the fixed effects coefficients is highly variable and can be just poorly estimated.

A different approach in such a case is the one of using random effect to model the states coefficient. The model would the look as follows:

$$\vec{y}_t = \left(\vec{\beta}_0 + \vec{v}\right) + \vec{\beta}^T \mathbb{X}_t + \vec{\varepsilon}_t$$

where,

$\vec{v}$ and $\vec{\varepsilon}_t$ are independent

$v_i \sim N(0, \sigma^2) \quad \forall\, v_i \in \vec{v}$, $\varepsilon_{it} \sim N(0, \sigma_v^2) \quad \forall\, \varepsilon_{it} \in \vec{\varepsilon}_t$

Such model is a mixed effect model in that it has a fixed effect component $\vec{\beta}^T$ and a random one: $\vec{\beta}_0 + \vec{v} = \begin{pmatrix} \beta_0 + v_1 \\ \vdots \\ \beta_0 + v_n \end{pmatrix}$

The concept of the model is the one of information pooling:

→ you estimate the $\sigma^2, \beta_0, \vec{\beta}, \sigma_v^2$.

→ you obtain an estimate of $\hat{v}_i$ given the σ^2, for each state.

↳ The model is parsimonious and it uses the concept that you allow a specific state effect while keeping a pooling to the overall mean intercept as the mean of $\vec{v}$ is $\vec{0}$.

Finally, notice that with such a model you account for the correlation of the observations across time

Proof: $Var(y_{it}) = Cov\left(\beta_0 + v_i + \beta^T \mathbb{X} + \varepsilon_{it},\ \beta_0 + v_i + \beta^T \mathbb{X} + \varepsilon_{it}\right) = Cov(v_i + \varepsilon_{it},\ v_i + \varepsilon_{it})$

$= Var(v_i) + Var(\varepsilon_{it}) + \cancel{2Cov(v_i, \varepsilon_{it})}$ $= 0$ due to independence

$Cov(y_{it}, y_{it'}) = Var(v_i)$ ⇒ no correlation in time $\neq 0$.

Correlation in time $= \rho(y_{it}, y_{it'}) = \dfrac{Var(v_i)}{Var(v_i) + Var(\varepsilon_{it})}$

Notice moreover the interesting property that correlation is the same for units close and distant in time.

The model above can be further extended to include a random term in the independent variables for the different states i.

Then in its simplest form (i.e. in the case of a single independent variable coefficient), the model will look as follows:

$$y_{it} = (\beta_0 + v_{i1}) + (\beta_1 + v_{i2})\, X_1 + \varepsilon_{it}$$

with:

$\vec{v}_i = \begin{pmatrix} v_{i1} \\ v_{i2} \end{pmatrix} \sim N(\vec{0}, \Sigma)$ ⇒ i.e. normal multivariate dist.

$\varepsilon_{it} \sim N(0, \sigma^2)$

So that now the parameters to be estimated within the model are: $\beta_0, \beta_1, \Sigma, \sigma^2$

And the covariance structure would look as follows:

$$Var(y_{it}) = Cov\left(\beta_0 + v_{i1} + (\beta_1 + v_{i2}) X_{it} + \varepsilon_{it},\ \beta_0 + v_{i1} + (\beta_1 + v_{i2}) X_{it} + \varepsilon_{it}\right)$$

$$= Cov(v_{i1}, v_{i1}) + \cancel{2Cov(v_{i1}, \varepsilon_{it})} + 2Cov(v_{i1}, v_{i2}) X_{it} + \cancel{2Cov(v_{i2}, \varepsilon_{it}) X_{it}} + Var(\varepsilon_{it}) + Cov(v_{i2}, v_{i2}) X_{it}^2$$

$$= Cov(v_{i1}, v_{i1}) + \cancel{2Cov(v_{i1}, \varepsilon_{it})} + 2Cov(v_{i1}, v_{i2})X_{it} + \cancel{2Cov(v_{i2}, \varepsilon_{it})X_{it}} + Var(\varepsilon_{it}) + Cov(v_{i2}, v_{i2})X_{it}^2$$

$$= Var(v_{i1}) + Var(v_{i2})X_{it}^2 + Var(\varepsilon_{it}) + 2Cov(v_{i1}, v_{i2})X_{it}$$

$$Cov(y_{it}, y_{it'}) = Cov(v_{i1}, v_{i1}) + Cov(v_{i1}, v_{i2})X_{it} + Cov(v_{i1}, v_{i2})X_{it'} + Cov(v_{i2}, v_{i2})X_{it} \cdot X_{it'}$$

} here the correlation through time also depends on time itself.

As a final note, notice that the above models can be estimated through the standard ways:

1. Maximum likelihood
2. Restricted Maximum Likelihood (the iterative approach).

- **Maximum likelihood (ML):**
  - Variance estimates are **biased**
  + Tests between two models with differing fixed and random effects are possible

- **Restricted maximum likelihood (REML):** Recommended (default in R)
  + Variance estimates are **unbiased**
  - Can only test between two models that have the **same fixed effects**