Supervised Learning

Sophie de Buyl

2020-21

VRIJE
UNIVERSITEIT
BRUSSEL

Outline of the course

1. Introduction
2. Technique #1 : Linear regression
3. Technique #2 : Linear discriminant analysis (LDA)
4. Technique #3 : Neural Networks
5. Resampling methods
6. Shrinkage methods : Ridge and Lasso

1. Introduction
2. Technique #1 : Linear regression
3. Technique #2 : Linear discriminant analysis (LDA)
4. Technique #3 : Neural Networks
5. Resampling methods
6. Shrinkage methods : Ridge and Lasso

[see chapter 1 and 2 in the book `http://faculty.marshall.usc.edu/gareth-james/ISL/`]

What is supervised learning and why is it used?

Our starting point is a "labeled" dataset (X, Y) where

- X are the inputs (think gene expression levels).
- Y is the output (think type of cell).

The idea behind supervised learning is that there is a relationship between the outputs and inputs:

$$Y = f(X) + \epsilon.$$

Because we cannot measure the values of the inputs with infinite precision, and we typically cannot take into account all inputs influencing the output, there is a random error term ϵ in the relationship.

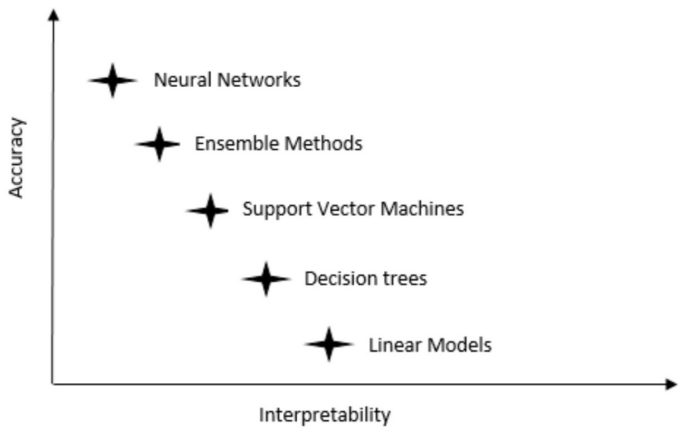
Rem : We assume that the error term is a random variable with zero mean and which is **independent** of X.

Goal: Supervised learning built estimates $\hat{f}$ for f in order **to be able to compute the output** $\hat{Y} = \hat{f}(X)$ corresponding to the input X and/or **to learn about the functional relationship between inputs and outputs**.

Think : you get gene expression data X from one cell and you want to predict its type Y.

Which method to choose to estimate f?

A wide range of methods exist to build estimates of f. Selecting the best approach can be the one of the most challenging part of machine learning.



output is continuous or categorical?

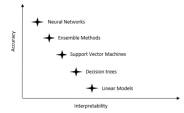
parametric vs. non-parametric?

trade-off between **interpretability** and **flexibility**

There is no free lunch in statistical learning $\Rightarrow$ Not one method dominates all others over all possible data sets!

Which method to choose to estimate *f*?

A wide range of methods exist to build estimates of ***f***. Selecting the best approach can be the one of the most challenging part of machine learning.



output is continuous or categorical?

parametric vs. non-parametric?

trade-off between **interpretability** and **flexibility**

There is no free lunch in statistical learning ⇒ Not one method dominates all others over all possible data sets!

Parametric model : A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) / a fixed functional dependence between output and inputs. No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs. Ex: linear regression, neural networks (at least the simplest ones).

Nonparametric models: Nonparametric methods make few or no assumptions about the functional dependence between the output and inputs. Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features. Ex: nearest neighbor methods, decision trees.

In conclusion with parametric models to predict new data, you only need to know the parameters of the model. In nonparametric methods are more flexible and for forecasting new data you need to know the parameters of the model and the state of the data that has been observed.

source: `https://medium.com/@lamiae.hana/parametric-and-nonparametric-machine-learning-algor`

figure: `https://medium.com/@abenav.sankar/motivation-944de6a71de7`

Evaluate the `goodness' of the estimator $\hat{f}$

- Training error: use the dataset and compute the MSE (see next slide), this tells you how well the model works on the dataset you used to build $\hat{f}$.

- Test error: provides an estimate of the 'goodness' of $\hat{f}$ for a **new input** (not belonging to the original dataset you used to build $\hat{f}$).

Evaluate the ‘goodness’ of the estimator $\hat{f}$

- Training error: use the dataset and compute the MSE (see next slide), this tells you how well the model works on the dataset you used to build $\hat{f}$.
- Test error: provides an estimate of the ‘goodness’ of $\hat{f}$ for a **new input** (not belonging to the original dataset you used to build $\hat{f}$).

Example: We can use these patients to train a statistical learning method to predict risk of diabetes based on clinical measurements. In practice, we want this method to accurately predict diabetes risk for future patients based on their clinical measurements. We are not very interested in whether or not the method accurately predicts diabetes risk for patients used to train the model (which would be the training error), since we already know which of those patients have diabetes.

How would you compute the test and training errors?

Training MSE: assessing the goodness of $\hat{f}$ with the dataset itself

To evaluate the 'goodness' of the estimate $\hat{f}$, we can compute the mean squared error MSE:

$$MSE = \frac{1}{n}\sum_{\ell=1}^{n}(y_\ell - \hat{y}_\ell)^2.$$

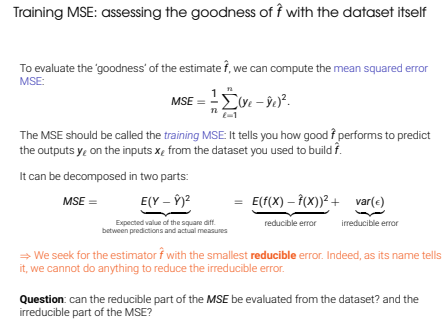
The MSE should be called the *training* MSE: It tells you how good $\hat{f}$ performs to predict the outputs y_ℓ on the inputs x_ℓ from the dataset you used to build $\hat{f}$.

It can be decomposed in two parts:

$$MSE = \underbrace{E(Y - \hat{Y})^2}_{\substack{\text{Expected value of the square diff.} \\ \text{between predictions and actual measures}}} = \underbrace{E(f(X) - \hat{f}(X))^2}_{\text{reducible error}} + \underbrace{var(\epsilon)}_{\text{irreducible error}}$$

$\Rightarrow$ We seek for the estimator $\hat{f}$ with the smallest **reducible** error. Indeed, as its name tells it, we cannot do anything to reduce the irreducible error.

Question: can the reducible part of the *MSE* be evaluated from the dataset? and the irreducible part of the MSE?


Training MSE: assessing the goodness of $\hat{f}$ with the dataset itself

To evaluate the 'goodness' of the estimate $\hat{f}$, we can compute the mean squared error MSE:

$$MSE = \frac{1}{n}\sum_{\ell=1}^{n}(y_\ell - \hat{y}_\ell)^2.$$

The MSE should be called the *training* MSE: It tells you how good $\hat{f}$ performs to predict the outputs y_ℓ on the inputs x_ℓ from the dataset you used to build $\hat{f}$.

It can be decomposed in two parts:

$$MSE = \underbrace{E(Y-\hat{Y})^2}_{\text{Expected value of the square diff. between predictions and actual measures}} = \underbrace{E(f(X)-\hat{f}(X))^2}_{\text{reducible error}} + \underbrace{var(\epsilon)}_{\text{irreducible error}}$$

⇒ We seek for the estimator $\hat{f}$ with the smallest **reducible** error. Indeed, as its name tells it, we cannot do anything to reduce the irreducible error.

Question: can the reducible part of the *MSE* be evaluated from the dataset? and the irreducible part of the MSE?

Example of random error term : we want to predict the response to a drug for patient and it depend on the patient's general feeling of well-being on that day. This general 'condition' of the patient is obviously something we can not fully quantify.

Decomposition of the MSE :

$$\begin{aligned} E[(Y-\hat{Y})^2] &= E[(f(X)+\epsilon-\hat{f}(X))^2] = E[(f(X)-\hat{f}(X)+\epsilon)^2] \\ &= E[(f(X)-\hat{f}(X))^2] + E[(\epsilon)^2] + 2E[(f(X)-\hat{f}(X))\epsilon] \quad (1) \\ &= E[(f(X)-\hat{f}(X))^2] + var(\epsilon) \quad (2) \end{aligned}$$

where E stands for the expectation (mean). The last equality comes from the facts that, (1) by assumption, $E(\epsilon) = 0$ and (2) ϵ and x are independent.

Answers to the questions on the slide : The *MSE* can be evaluated from the dataset, the y_ℓ are given with the dataset and the $\hat{y}_\ell$ can be computed via $\hat{f}(x_\ell)$ (x_ℓ are given with the dataset et $\hat{f}$ is evaluated from the dataset). On the other hand, the reducible error cannot be evaluated from the dataset. Indeed, we do not know the 'true' f. Similarly, $var(\epsilon)$ can not be computed directly. We can however build an estimator $\hat{\epsilon}$ for ϵ.

Training MSE: assessing the goodness of $\hat{f}$ with the dataset itself

To evaluate the 'goodness' of the estimate $\hat{f}$, we can compute the mean squared error MSE:

$$MSE = \frac{1}{n}\sum_{\ell=1}^{n}(y_\ell - \hat{y}_\ell)^2.$$

The MSE should be called the *training* MSE: It tells you how good $\hat{f}$ performs to predict the outputs y_ℓ on the inputs x_ℓ from the dataset you used to build $\hat{f}$.

It can be decomposed in two parts:

$$MSE = \underbrace{E(Y-\hat{Y})^2}_{\text{Expected value of the square diff. between predictions and actual measures}} = \underbrace{E(f(X)-\hat{f}(X))^2}_{\text{reducible error}} + \underbrace{var(\epsilon)}_{\text{irreducible error}}$$

⇒ We seek for the estimator $\hat{f}$ with the smallest **reducible** error. Indeed, as its name tells it, we cannot do anything to reduce the irreducible error.

Question: can the reducible part of the *MSE* be evaluated from the dataset? and the irreducible part of the MSE?

Training MSE: Reducible vs. irreducible error on one example

Assume that someone used the formula $y = f(x) = \beta_0 + \beta_1 x$ with $\beta_0 = 2$ and $\beta_1 = 3$ to generate a couple of (x, y) samples: $\{(x^1, y^1_{\text{true}}), (x^2, y^2_{\text{true}}), \ldots, (x^n, y^n_{\text{true}})\}$.

Then, they were send to you but the (output) message got a bit distorted and you got the couples

$$\{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\} = \{(x^1, y^1_{\text{true}} + \epsilon^1), (x^2, y^2_{\text{true}} + \epsilon^2), \ldots, (x^n, y^n_{\text{true}} + \epsilon^n)\}.$$

We know a bit of math and you have been able to estimate the values of the intercept $\hat{\beta}_0 = 2.8$ and of the slope $\hat{\beta}_1 = 3.1$ from the samples. This allows you to build an estimate for the function f: $\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$.

The mean error you are making with you estimator $\hat{f}$ can be quantified as:

$$MSE = \frac{1}{n}\sum_{\ell=1}^{n}(y^\ell - \hat{y}^\ell)^2 = \sum_{\ell=1}^{n}((\beta_0 + \beta_1 x^\ell + \epsilon^\ell) - (\hat{\beta}_0 + \hat{\beta}_1 x^\ell))^2 \tag{3}$$

$$= \underbrace{\sum_{\ell=1}^{n}((\beta_0 - \hat{\beta}_0) + (\beta_1 - \hat{\beta}_1)x^\ell)^2}_{\text{error due to not perfect estimations}} + \underbrace{\sum_{\ell}(\epsilon^\ell)^2}_{\text{irreducible error}} \tag{4}$$

Rem: the term $\sum_\ell((\beta_0 - \hat{\beta}_0) + (\beta_1 - \hat{\beta}_1)x^\ell)\,\epsilon_\ell$ is zero because of the assumptions made on the random error term ϵ (zero mean and independent of x).

Test MSE : Assessing goodness of $\hat{f}$ on an unseen datapoint

As said above, to evaluate the quality of $\hat{f}$, our estimation of f, we can compute the mean squared error MSE. However, we are typically interested in the accuracy of the predictions that we obtain when we apply our method to previously **unseen** input.
Think about medical data $\Rightarrow$ you want to do prediction for unseen data!

If you have access to m new data points $(x_{\ell'}, y_{\ell'})$, you can simply evaluate the test error as

$$MSE_{\text{test}} = \frac{1}{m} \sum_{\ell'=1}^{m} (y_{\ell'} - \hat{y}_{\ell'})^2,$$

where $y_{\ell'} = y_{\ell'}(x_{\ell'})$ and $\hat{y}_{\ell'} = \hat{f}(x_{\ell'})$.

Typically, we do **not** have access to new datapoints and we need to be able to estimate the test error from our original dataset (see later, resampling methods).

Assessing models quality : test MSE vs. training MSE

Let x_0 be a 'new' input:

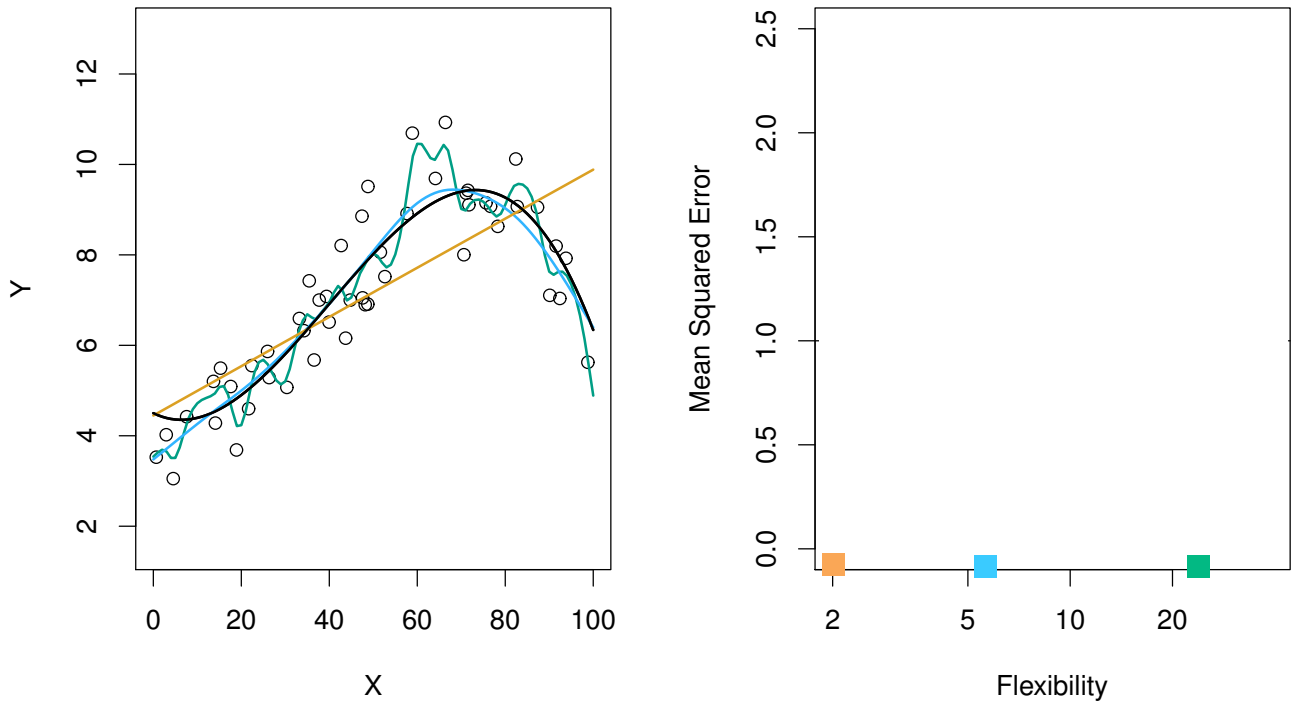
$$\text{test MSE}(x_0) = E(y_0 - \hat{f}(x_0))^2$$

Again, there is a reducible and irreducible part. The reducible part can be decomposed into two sources or errors, can you think about their origin?

$$\text{test MSE}(x_0) = \underbrace{\text{Var}(\hat{f}(x_0))}_{\text{choice of training set}} + \underbrace{[\text{Bias}(\hat{f}(x_0))]^2}_{\text{choice of method}} + \underbrace{\text{Var}(\epsilon)}_{\text{unavoidable}} .$$

- **variance** : Variance refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set
- **bias** : it refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.
- ϵ : is the unavoidable error $y = f(x) + \epsilon$.

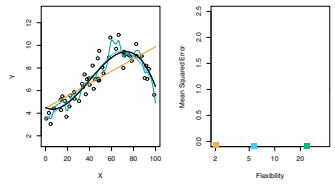
Assessing models quality : test MSE vs. training MSE - an example



Left: simulated data (dots) for the know function $f(x)$ (solid black line) ; yellow - linear regresion ; blue & green - smoothing splines with different levels of smoothness.

Right: Sketch the shape of training MSE and the test MSE.

Assessing models quality : test MSE vs. training MSE - an example



Left: simulated data (dots) for the know function $f(x)$ (solid black line) ; yellow - linear regresion ; blue & green - smoothing splines with different levels of smoothness.

Right: Sketch the shape of training MSE and the test MSE.

Rem: The flexibility is given here by the number of parameters to fit. The linear regression in yellow has 2 free parameters. The 'smoothing splines' consist of methods with more parameters to fit, here in the blue one has 6 free parameters and the green one 22.

Questions on next slide:

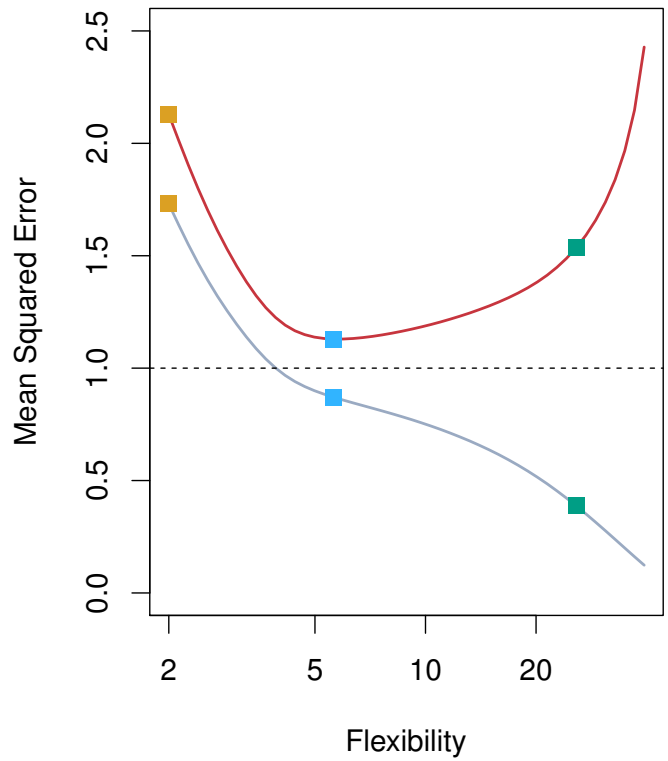
- The dashed line is the minimum error which can be reach, it is due to the stochasticity of the relationship $y = f(x) + \epsilon$. The minimal error is given by the variance of the noise σ.

- The U-shape comes from the fact that when you increase the flexibility, you start fitting the noise in the data. Imagine that for the test set you remove one of the point (black circle), which of the yellow, blue or green curves will change most? Clearly the linear fit will not be much affected (and therefore the "variance" part of the error is not much affected. At the opposite side of flexibility, the green curve will be very much affected by the remove of one of the points from the training set. This means that the "variance" part of the test MSE increases with the flexibility.
To take a more extreme example, imagine that the flexibility is so high that it makes the fitted curve pass by all training points (but one that you hold off the training process to be able to test). Then the training MSE will be zero. But the kept aside data point will clearly not be on the curve [the true curve here is the black one, so there is not reason for the 'kept apart point' to be an the curve passing by all training points spread around the 'true' black line. Clearly making the curve $\hat{f}$ passing by all training points amounts to fit the noise on top of fitting the true black curve.].

Assessing models quality : Why U-shape for the test MSE?

Training MSE (gray)

Test MSE (red)



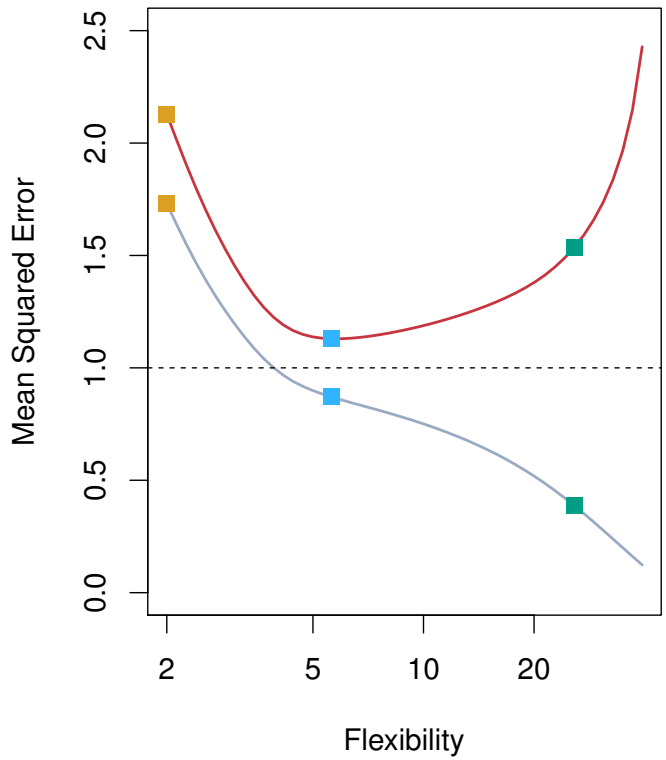
What is the dashed gray line (right)?
Why is the test MSE U-shaped?
For which models is there an overfitting issue?

Assessing models quality : test MSE vs. training MSE

training MSE : As the flexibility increases $\rightarrow$ monotone decrease in the training MSE.

test MSE : The increase is due to the fact that the statistical method is picking up some patterns that are just caused by random chance rather than by true properties of the unknown function $f(x)$.

When a given method yields a small training MSE but a large test MSE, we are said to be **overfitting** the data.



Which method to choose?

One typically aims at selecting the technique giving the lowest test MSE.

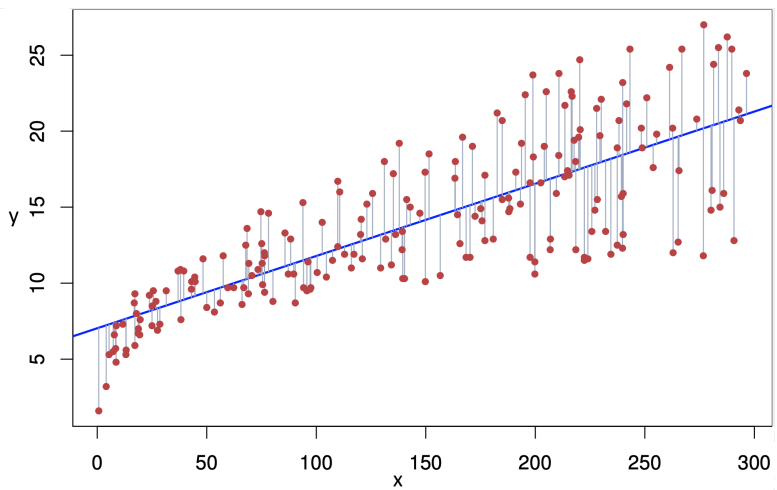
[see section 3.1. in the book **http://faculty.marshall.usc.edu/gareth-james/ISL/**]

Linear regression

We assume

$$y = f(x) + \epsilon = \beta_0 + \beta_1 x + \epsilon$$

and we want to estimate the two parameters β_0 and β_1.



We can do this easily by minimizing

$$MSE = \frac{1}{n}\sum_{\ell=1}^{n}(y^\ell - \hat{\beta}_0 - \hat{\beta}_1 x^\ell)^2 .$$

Linear regression: estimation of the parameters

Let us reformulate linear regression in matrix notations. Our model is:

$$y = X\beta + \epsilon$$

where

- $X = (1, x)$.
- The array $\boldsymbol{\beta}$ is given by $[\beta_0, \beta_1]$ (in column).
- $\boldsymbol{\epsilon}$ is the random error term assumed to be gaussian $\epsilon \sim N(0, \sigma^2)$.

In matrix notation, the MSE is given by

$$MSE = \frac{1}{n} \underbrace{(\underbrace{\boldsymbol{Y}}_{n\times 1} - \underbrace{\boldsymbol{X}}_{n\times 2}\underbrace{\boldsymbol{\beta}}_{2\times 1})^T}_{1\times n} \underbrace{(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta})}_{n\times 1}$$

where $\boldsymbol{X}$ is a matrix whose first column contains ones and second column contains the inputs $\boldsymbol{x}^\ell$ and $\boldsymbol{Y}$ is an array containing all the outputs $\boldsymbol{y}^\ell$.

To minimize the error and obtain estimation for $\boldsymbol{\beta}$, we derive it with respect to $\boldsymbol{\beta}$:

$$\frac{\partial MSE}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) \Rightarrow \hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}.$$

Linear regression: estimation of the parameters

Let us reformulate linear regression in matrix notations. Our model is:

$$y = X\beta + \epsilon$$

where

- $X = (1, x)$.
- The array $\boldsymbol{\beta}$ is given by $[\beta_0, \beta_1]$ (in column).
- ϵ is the random error term assumed to be gaussian $\epsilon \sim N(0, \sigma^2)$.

In matrix notation, the MSE is given by

$$MSE = \frac{1}{n} \underbrace{(\underbrace{\boldsymbol{Y}}_{n\times 1} - \underbrace{\boldsymbol{X}}_{n\times 2} \underbrace{\beta}_{2\times 1})^T}_{1\times n} \underbrace{(\boldsymbol{Y} - \boldsymbol{X}\beta)}_{n\times 1}$$

where $\boldsymbol{X}$ is a matrix whose first column contains ones and second column contains the inputs x^ℓ and $\boldsymbol{Y}$ is an array containing all the outputs y^ℓ.

To minimize the error and obtain estimation for β, we derive it with respect to $\boldsymbol{\beta}$:

$$\frac{\partial MSE}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) \Rightarrow \hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}.$$

Linear regression is a parametric method and the outputs are continuous. Though it may seem somewhat dull compared to some of the more modern statistical learning approaches, linear regression is still a useful and widely used statistical learning method.

If you want expressions in non-matricial notations (check this as an exercise):

$$\hat{\beta}_1 = \frac{\sum(x^\ell - \bar{x})(y^\ell - \bar{y})}{\sum(x^\ell - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

Rem: here it is a bit tricky to follow conventions (bold, greek..) for matrices like we did in unsupervised learning.

Linear regression: error on estimated parameters

We see that the estimated parameters $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\,\boldsymbol{X}^T\boldsymbol{Y}$ are linear combinations of the outputs $\boldsymbol{Y}$ which are random variables.

$\Rightarrow$ The $\hat{\boldsymbol{\beta}}$s are therefore random variables too.

We can estimate the covariance matrix of $\hat{\boldsymbol{\beta}}$:

$$\mathrm{cov}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\,\sigma^2$$

If we know the value of σ (from the experimental design for instance):

$$\Rightarrow \qquad \hat{\beta} \sim \mathrm{N}(\beta, (X^TX)^{-1}\sigma^2).$$

Exercise: check that the mean values of $\hat{\boldsymbol{\beta}}$ are indeed given by $\boldsymbol{\beta}$.

Reading the diagonal elements of the covariance matrix, we find

$$\hat{\sigma}^2_{\beta_1} = \frac{\sigma^2}{\sum_{\ell=1}^{n}(x_\ell - \bar{x})^2} \text{ and } \sigma^2_{\hat{\beta}_0} = \sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{\sum_{\ell=1}^{n}(x_\ell - \bar{x})^2}\right).$$

We can use this standard deviation to provide confidence intervals.

> There is approximately 95% chance that the interval $[\hat{\beta}_1 - 2\sigma_{\hat{\beta}_1}, \hat{\beta}_1 + 2\sigma_{\hat{\beta}_1}]$ contains the true values of β_1.

Can you guess why it is 'approximately' only?

Linear regression: error on estimated parameters

We see that the estimated parameters $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$ are linear combinations of the outputs $\boldsymbol{Y}$ which are random variables.

⇒ The $\hat{\beta}$s are therefore random variables too.

We can estimate the covariance matrix of $\hat{\boldsymbol{\beta}}$:

$$\mathrm{cov}(\hat{\boldsymbol{\beta}}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\sigma^2$$

If we know the value of σ (from the experimental design for instance):

$$\Rightarrow \quad \hat{\beta} \sim \mathrm{N}(\beta, (X^TX)^{-1}\sigma^2).$$

Exercise: check that the mean values of $\hat{\boldsymbol{\beta}}$ are indeed given by $\boldsymbol{\beta}$.

Reading the diagonal elements of the covariance matrix, we find

$$\hat{\sigma}^2_{\hat{\beta}_1} = \frac{\sigma^2}{\sum_{\ell=1}^n (x_\ell - \bar{x})^2} \text{ and } \sigma^2_{\hat{\beta}_0} = \sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{\sum_{\ell=1}^n (x_\ell - \bar{x})^2}\right).$$

We can use this standard deviation to provide confidence intervals.

> There is approximately 95% chance that the interval $[\hat{\beta}_1 - 2\sigma_{\hat{\beta}_1}, \hat{\beta}_1 + 2\sigma_{\hat{\beta}_1}]$ contains the true values of β_1.

Can you guess why it is 'approximately' only?

$Y = X\cdot\beta + \epsilon$ $\qquad E(\epsilon) = 0$

$\hat{\beta} = \underbrace{(X^TX)^{-1}X^T}_{A}\ \overset{\text{rand. var.}}{Y} = AY$ $\qquad$ The $\hat{\beta}$'s are linear combinations of the random variables Y.

$A \rightarrow$ real matrix

$$E(\hat{\beta}) = E(AY) = AE(Y) = AE(X\cdot\beta + \epsilon) = AE(X\cdot\beta) = AX\cdot\beta$$

$$= \underbrace{(X^TX)^{-1}X^TX}_{\mathbb{1}}\beta = \beta$$

⇒ It's a good thing for an estimator to be such that $E(\hat{\beta}) = \beta$ [the mean value of the estimator is the true value]

$$\mathrm{covar}(\hat{\beta}) = \mathrm{covar}(AY) = E[(AY - A\bar{Y})^2] = E[(AY - A\bar{Y})\underbrace{(AY - A\bar{Y})^T}_{(Y^T - \bar{Y}^T)A^T}]$$

$$= E[A(Y-\bar{Y})(Y^T - \bar{Y}^T)A^T]$$

$$= AE[(Y-\bar{Y})(Y-\bar{Y})^T]A^T$$

$$= A\,\mathrm{cov}(Y)\,A^T$$

where $\mathrm{cov}(Y) = \mathrm{cov}(\epsilon) = \begin{pmatrix}\sigma^2 & 0\\ 0 & \sigma^2\end{pmatrix}$

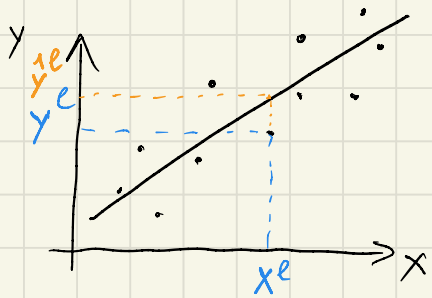
$(X^TX)^T = X^TX$

$$\Rightarrow \mathrm{cov}(\hat{\beta}) = (X^TX)^{-1}X^T\,\mathrm{cov}(Y)\,X\left((X^TX)^{-1}\right)^T = \sigma^2\,(X^TX)^{-1}X^T X (X^TX)^{-1} = \sigma^2(X^TX)^{-1}$$

Here we still miss something...

Linear regression: estimating σ from our dataset

We will show that $\hat{\sigma} = \sqrt{\frac{nMSE}{(n-2)}}$ is a 'good' estimator. This means that its expectation value is $E(\hat{\sigma}) = \sigma$ and that we know of this estimator (which is a random variable) is ditributed.



For each couple of points (x^e, y^e), we can estimate ϵ^e with $\hat{\epsilon}^e = y^e - \hat{y}^e$.

← build out of data set

$$\Rightarrow \quad \hat{\epsilon} = \begin{bmatrix} \hat{\epsilon}^1 \\ \vdots \\ \hat{\epsilon}^n \end{bmatrix} \text{ is given by } \hat{\epsilon} = y - \hat{y} = y - X\cdot\hat{\beta} = \underbrace{(I - X(X^TX)^{-1}X^T)}_{Q} y = Qy = Q(X\cdot\beta + \epsilon)$$

because $QX\beta = 0$, we have $\hat{\epsilon} = Q\epsilon$

An estimator for σ^2 is $\hat{\sigma}^2 \propto \|\hat{\epsilon}\|^2 = \frac{1}{n}\sum_{e=1}^{n} \hat{\epsilon}^{e\,2}$ ← our estimator is a sum of square of random variable.

↑ proportional to (we'll see later why)

We can easily check that $Q^2 = Q = Q^T$ and $\mathrm{tr}\, Q = n - p$ where $p = \#$ parameters. ↳ i.e. 2 here

$\Rightarrow$ Q can has eigenvalues 1 and 0 only with multiplicities $(n-p)$ and p respectively. Q can be diagonalized to $D = \begin{pmatrix} \underbrace{1 \cdots 1}_{n-p} & & \\ & \underbrace{0 \cdots 0}_{p} \end{pmatrix}$: $D = VQV^T$

and $k = {}^tV\hat{\epsilon}$.

Linear regression: estimating σ from our dataset

Since $\hat{\epsilon} \sim N(0, \sigma^2 Q)$, we have $k \sim N(0, \underbrace{\sigma^2 D}_{\text{diag. cov. matrix with } D \text{ given on previous slide}})$ $\Rightarrow k_{n-p-1} = \dots = k_n = 0$

$$\Rightarrow \frac{\|k\|^2}{\sigma^2} = \frac{\|k^*\|^2}{\sigma^2} \sim \chi^2_{n-p} \qquad k^* = (k_1, \dots, k_{n-p})$$

Moreover as V is a unitary matrix, we have $\|\hat{\epsilon}\|^2 = \|k^*\|^2 \Rightarrow \frac{\|\hat{\epsilon}\|^2}{\sigma^2} \sim \chi^2_{n-p}$

$$\Rightarrow \hat{\sigma}^2 = \frac{\|\hat{\epsilon}\|^2}{n-p} \text{ is an estimator for } \sigma^2 \text{ such that } E[\hat{\sigma}^2] = \sigma^2$$

https://stats.stackexchange.com/questions/20227/why-is-rss-distributed-chi-square-times-n-p

We can know come back to $\hat{\beta}$

Linear regression: Is a linear model a good fit?

Standard errors can be used to perform hypothesis test on the coefficients. Typically we test the null hypothesis H_0:

$$H_0 : \text{There is no relation between } X \text{ and } Y \text{ (aka } \beta_1 = 0).$$

The alternative hypothesis is

$$H_1 : \text{There is a relation between } X \text{ and } Y \text{ (aka } \beta_1 \neq 0).$$

The idea is that a claim, *i.e.* the alternative hypothesis, is assumed valid **if** its counter-claim (the null hypothesis) is improbable.

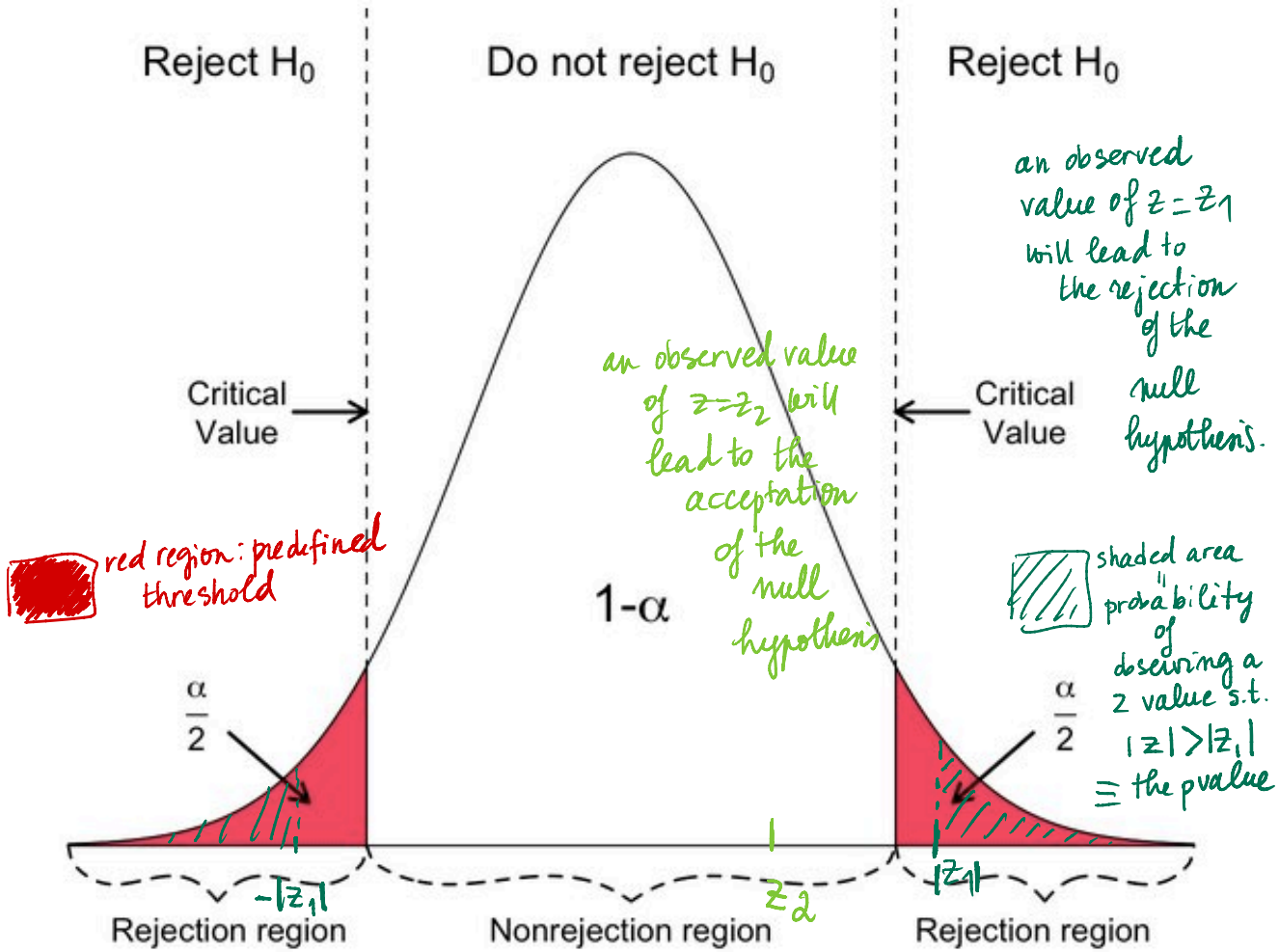
We need to know **how far from zero our estimate of β_1 is**. Obviously, to do the comparison we need to know the accuracy of β_1:

$$t = \frac{\hat{\beta}_1 - 0}{\sqrt{\sigma^2_{\hat{\beta}_1}}} .$$

If there is no relation between x and y, we expect t to be distributed according to a t-distribution with $n - 2$ d.o.f. . Now, we simply need to compute the probability of picking a value $|t|$ or larger. This probability is called the p-value.

A small p-value (typically ≤ 0.05) indicates strong evidence against the null hypothesis, so you reject the null hypothesis.

How do decide : accept or reject the null hypothesis?



Linear regression: Accuracy of the Model

Once we rejected the null hypothesis, we might want to assess the quality of the model.

The residual sum of squares *RSS*

$$RSS = E(y - \hat{y})^2 = \sum_{\ell=1}^{n} (y^\ell - \hat{\beta}_0 - \hat{\beta}_1 x^\ell)^2 \, .$$

gives an idea of this quality. What could be a drawback of this quantity?

It is measured in units of y and it is not always clear what a good value is. The R^2 statistic provides a number between 0 and 1 that describes the proportion of the variance explained:

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS} \, ,$$

where $TSS = \sum (y_\ell - \bar{y})^2$ is the total sum of squares.

Summary linear regression

1. ESTIMATIONS OF PARAMETERS: Minimise training MSE $\Rightarrow$ estimated of β_0 and β_1.

2. CONFIDENCE INTERVAL FOR PARAMETERS: Assuming gaussian error ϵ with variance σ^2, one can compute the standard deviation of the parameters (and therefore confidence intervals). **Rem**: one should estimate σ^2 from the data too.

3. IS THE LINEAR MODEL GOOD FOR OUR DATA? test null hypothesis

4. IF THE RESPONSE IS YES (i.e. small p value, meaning we reject the null hypothesis), we compute the proportion of the variance explained to see how good is the linear model (R^2 close to 1 is good).

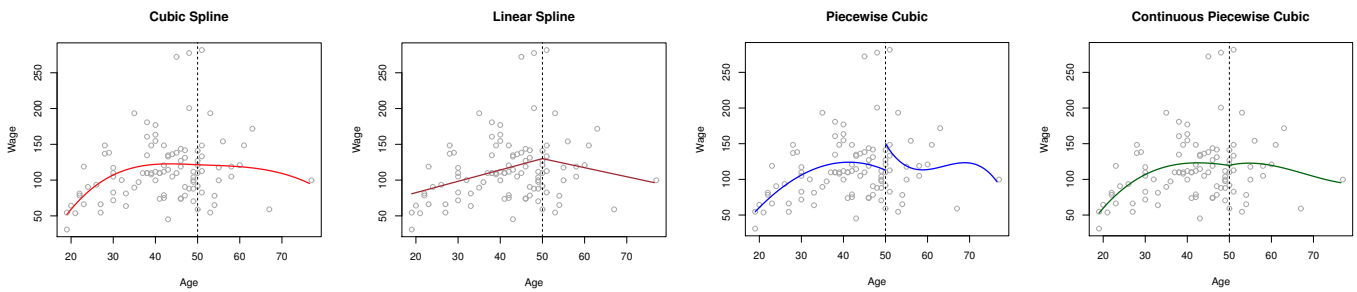
Generalizations of linear regression

- if the output is categorical (*i.e.* yes/no), we can use linear regression to model the probability distribution of obtaining one of the outputs. This is known as the logistic model. We model the conditional probability of getting the output yes for instance, knowing that the input in x.

- Multidimensional case of linear regression ($x \rightarrow (x_1, ..., x_p)$).

- Moving beyond linearity : Similar technique but $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$ for instance. We should be careful about correlations between the error terms, non-constant variance of error terms, ...

- Splines



Good to know

⋆ We can also use other quantities than ***RSS*** to be minimized as a quality criteria. For instance:

- Ridge regression: $\boldsymbol{RSS} \Rightarrow \boldsymbol{RSS} + \lambda \sum_1^p \beta_i^2$, the term proportional to λ is called a shrinkage penalty. It favors small values of β's.
- Lasso regression: $\boldsymbol{RSS} \Rightarrow \boldsymbol{RSS} + \lambda \sum_1^p |\beta_i|$

⋆ Resampling methods are key tools in modern statistics, they involve

- splitting data into a training set and a test set
- repeatedly drawing samples from a training set and refitting a model of interest on each sample.

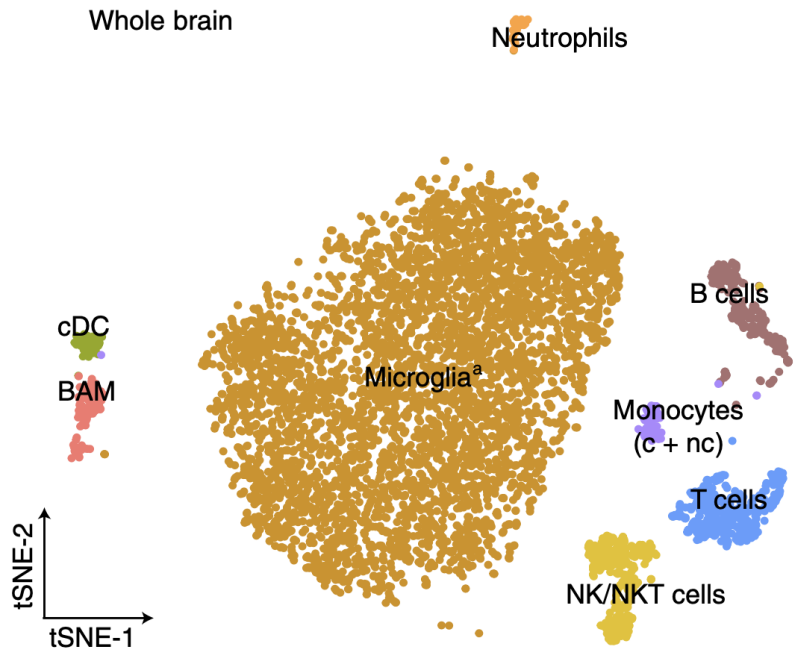
Most common methods are called *cross-validation* and *bootstrap*.

1. Introduction
2. Technique #1 : Linear regression
3. Technique #2 : Linear discriminant analysis (LDA)
4. Technique #3 : Neural Networks
5. Resampling methods
6. Shrinkage methods : Ridge and Lasso

[see section 4.6.3 in the book `http://faculty.marshall.usc.edu/gareth-james/ISL/`]

Linear Discriminant analysis

To goal of LDA is to predict the output for a new input. Imagine that gene expression for a new cell has been measured and we want to know what type of cell it is...



`https://www.nature.com/articles/s41593-019-0393-4`

Linear Discriminant analysis

LDA is a popular technique to predict the categorical output when the number of categories is bigger than 2. We aim at estimating the probability that a given input x belongs to the category k: $p_k(x)$.

We need:

- π_k = prior probability that an observation belongs to the class k
- $f_k(X) := \Pr(X = x|Y = k)$ probability distribution of the inputs knowing the class it belongs to (here $Y = k$ means that the output is the category k).

Then we can use Bayes theorem $P(k|X) = P(X|k)P(k)/P(X)$ to compute $p_k(x) := \Pr(Y = k|X = x)$:

$$p_k(x) = \frac{\pi_k f_k(x)}{\sum_{\ell=1}^{K} \pi_\ell f_\ell(x)}$$

π_k are easy to estimate from the data set : how would you do?

$f_k(x)$ is more subtle, we will see know to do this...

LDA : how to estimate $f_k(x)$?

The idea is to assume a simple form for this probability distribution density.
$p_k(x)$ is then referred to as the posterior probability.

LDA simplest case: one input

Let us assume

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k}} \exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right)$$

and also that $\sigma_k = \sigma$ (which we will need to estimate from the data) $\Rightarrow$

$$p_k(x) = \frac{\frac{\pi_k}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{N}$$

where the normalization $N = \sum_\ell \frac{\pi_\ell}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_\ell)^2\right)$.

We will assign the new observation x to the class k for which $p_k(x)$ is maximal. We can easily show that this is equivalent to assigning x to the class k for which $\delta_k(x)$ is maximal:

$$\delta_k(x) = x\mu_k/\sigma^2 - \mu_k^2/2/\sigma^2 + \log(\pi_k)$$

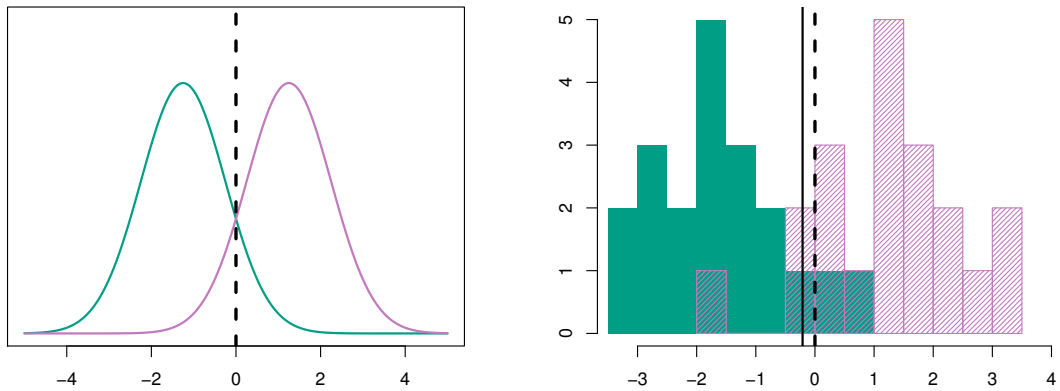
.

LDA simplest case: one input - estimation of parameters

$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$
$\hat{\sigma}^2 = \frac{1}{n-K} \sum_k \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$
$\hat{\pi}_k = n_k/n$

$\Rightarrow$ with these estimations, we can compute $p_k(x)$ (or equivalently $\delta_k(x)$) and assign x to the category with the highest value.

LDA simplest case - an example



Left : We have one input which belong either to the green of pink classes. We assume that within each class the distribution of the variable is gaussian with the same variance but different means
Right : The histogram is obtained from drawing 20 inputs for each class.

The **dashed line** represents the Bayes decision boundary. The **solid one** is the boundary estimated from the training data.

If you pick up a new input $x = -1$, to which class would you assign it? and if $x = -0.1$?

LSA - assess quality

		True "default status"		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

The training error is only 2.75 %.

true positive rate – here we want to identify people who can't pay credit cards
sensitivity : percentage of true defaulters ('Yes') that are identified (81/333 = 24,3 %).

false positive rate
1 - specificity : percentage of false positive that are identified (23/9667 = 2 %).

specificity : percentage of true non-defaulter ('No') that are identified (9644/9667 = 99.8%).

		True "default status"		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

LSA - assess quality

The Bayes classifier works by assigning an observation to the class for which the posterior probability $p_k(X)$ is greatest. We can change the threshold, for instance to

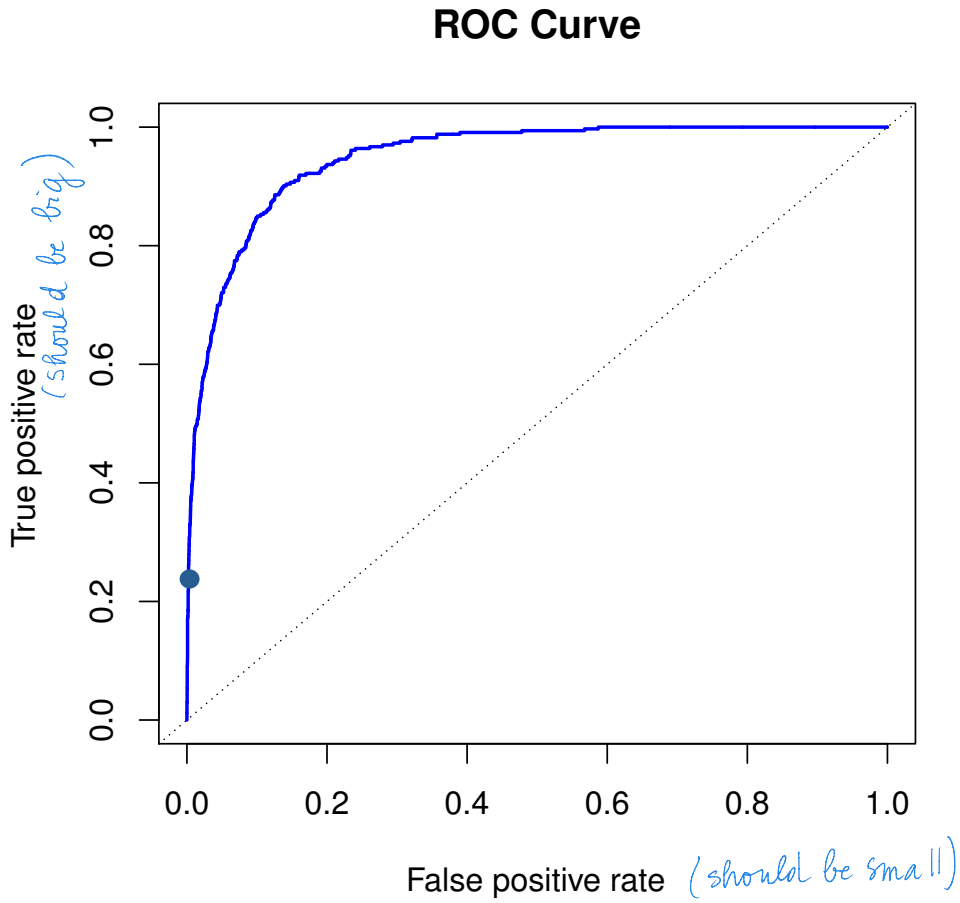
$$P(default = Yes|X = x) > .2$$

This will change the confusion matrix:

		True default status		
		No	Yes	Total
Predicted default status	No	9,432	138	9,570
	Yes	235	195	430
	Total	9,667	333	10,000

TABLE 4.5. *A confusion matrix compares the LDA predictions to the true default statuses for the* 10,000 *training observations in the* `Default` *data set, using a modified threshold value that predicts default for any individuals whose posterior default probability exceeds* 20 %.

LSA - ROC table



A ROC curve traces out two types of error as we vary the threshold value for the posterior probability of default. [the dot corresponds to the 0.5 threshold]

LDA generalizations - multiples inputs

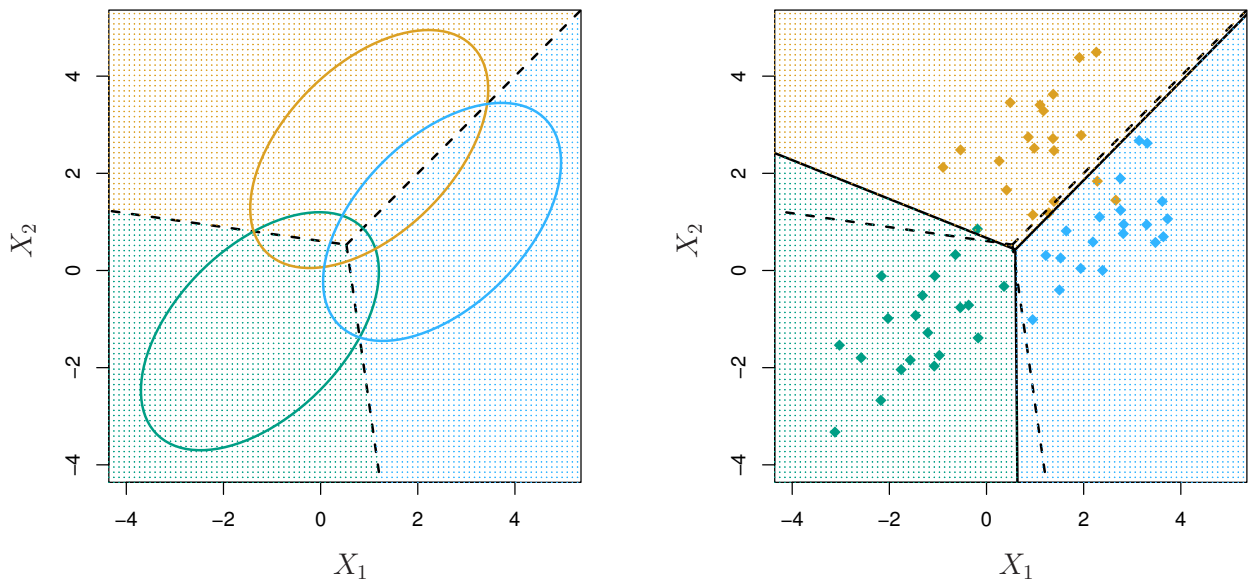
You assume now that the inputs are drawn from multivariate Gaussian distributions:

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right)$$

A little bit of algebra gives you the Bayes classifier :

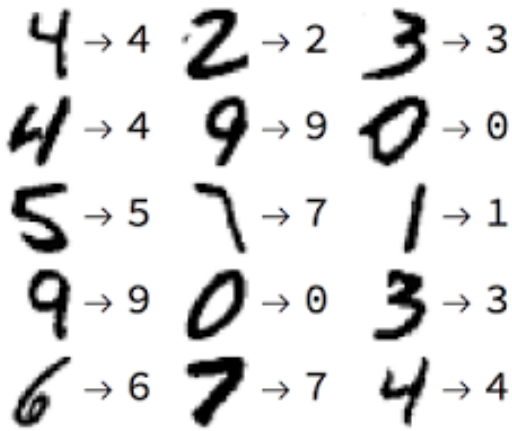
$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k$$

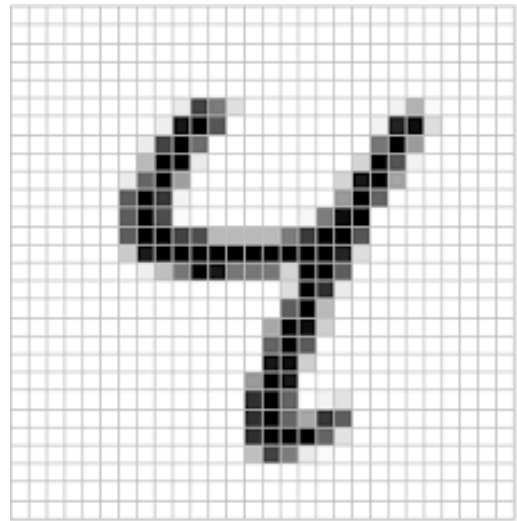
Then we need to estimate π_k, μ_k and Σ as before.



1. Introduction
2. Technique #1 : Linear regression
3. Technique #2 : Linear discriminant analysis (LDA)
4. Technique #3 : Neural Networks
5. Resampling methods
6. Shrinkage methods : Ridge and Lasso

Neural network: an introduction with digits recognition





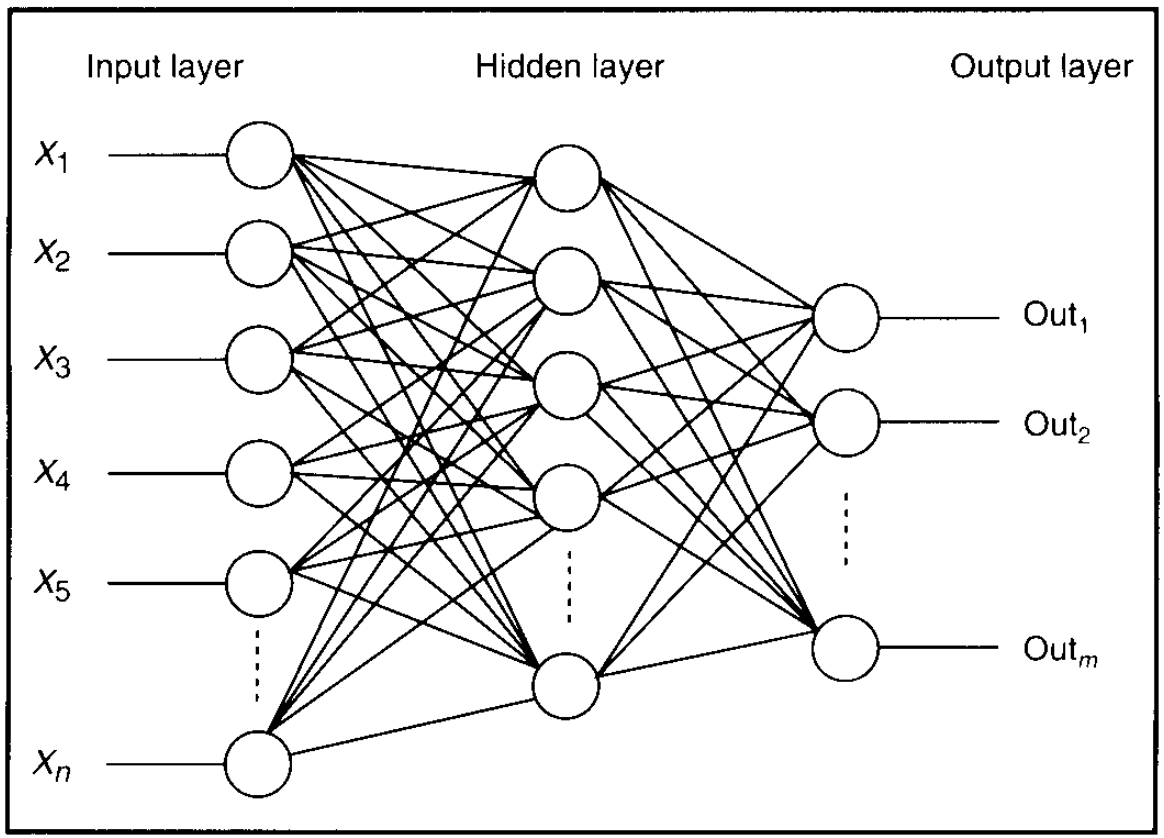
Each digit is represented by a low resolution image resolution (28 by 28 pixels).
The various 4's that we see here correspond to different 'black' pixels, however our brain can easily recognize the three 4's.

Our input (the image) is a 28×28 matrix filled with numbers between 0 and 1 representing the color (0 for white, 1 for black and in between for levels of grey).

How to go from 784 numbers to the identification of the digit?

Neural network: an introduction with digits recognition

Neuron = something that holds a number between 0 and 1
Activation = the number taken by one neuron is its activation level



Neural network: an introduction with digits recognition

The activation of one layer is determined by the previous layer. For each of the neurons on the first hidden layer you compute

$$w_1 x_1 + ... + w_n x_n$$

where the w_i are weights (specific to each neuron on this first hidden layer). Since you want that activation is between 0 and 1, you can use a 'squeezing function', typically the sigmoid function

$$\sigma(x) = \frac{1}{1 + exp^{-x}} .$$

This will 'activate' the neuron if you weighted sum is bigger than 0. You can introduce a bias if you want the neuron to be activated if this sum is bigger than an other number b (you have a biais for the neuron to be inactive).

$$w_1 x_1 + ... + w_n x_n - b$$

We will take an example with $n = 784$ inputs (pixels), $m = 10$ outputs (a digit from 0 to 9) and 2 hidden layers with each 16 nodes (this is an arbitrary choice).

For our simple example, we have :
$\underbrace{784 \times 16 + 16 \times 16 + 16 \times 10}_{\text{weigths}} + \underbrace{(16 + 16 + 10)}_{\text{biais}} = 13002$ parameters to adjust.

Neural network: an introduction with digits recognition

Notation: the action vector in the m^{th} layer is given by the activation vector in the previous layer:

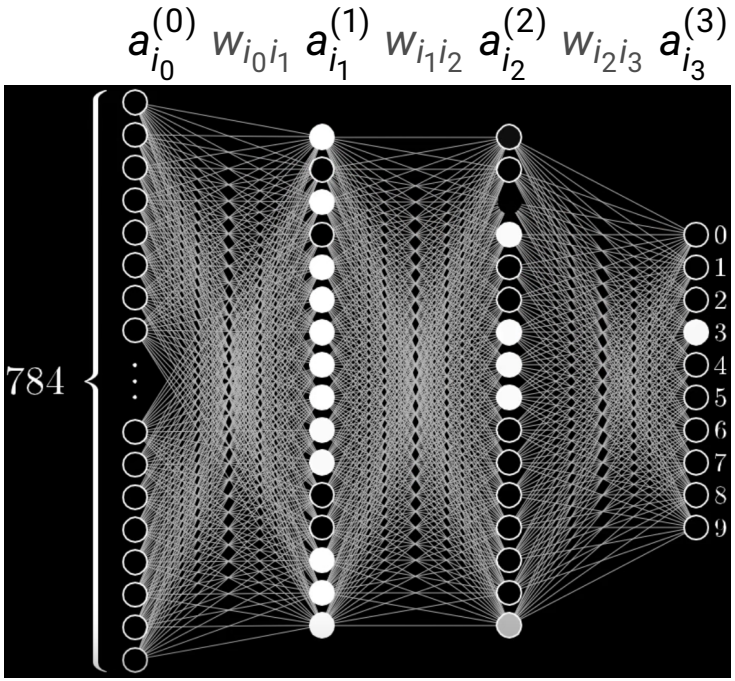
$$a^{(m)} = \sigma(Wa^{(m-1)} + b)$$

in a more explicit form,

$$a_j^{(m)} = \sigma(\sum_{i=0}^{n} w_{ji} a_i^{(m-1)} + b_j^{(m-1)})$$

Neural network: an introduction with digits recognition

Let us look in more details on this specific example how the third output $a_3^{(3)}$ depends on all weights and bias by back propagating the information.



$$a_3^{(3)} =$$

$$\sigma\Big(\sum_{i=0}^{n_2} w_{3i} a_i^{(2)} + b_3^{(2)}\Big) =$$

$$\sigma\Big(\sum_{i=0}^{n_2} w_{3i}\,\sigma\Big(\sum_j w_{ij}\, a_j^{(1)} + b_i^{(1)}\Big) + b_3^{(2)}\Big) =$$

$$\sigma\Big(\sum_{i=0}^{n_2} w_{3i}\,\sigma\Big(\sum_j w_{ij}\,\sigma\Big(\sum_k w_{jk} a_k^{(0)} + b_j^{(0)}\Big) + b_i^{(1)}\Big) + b_3^{(2)}\Big)$$

$\Rightarrow$ the outputs $a_{i_3}^{(3)}$ are functions of the weights, bias and input values $a_{i_0}^{(0)}$

rem: $i_0 = 0, ..., 783$, $i_1, i_2 = 0, ..., 15$, and $i_3 = 0, ..., 9$.

Neural network: an introduction with digits recognition

How to train the network?

You need data sets for which you know the answer. The (public) MNIST Database contains tens of thousands of annotated images of digits.

We start put choosing randomly your weights and bias. Let us call w the 13002 dimensional vector containing all weights and biais.

Then, you need a cost function $C(w)$:

$$C(w) = \text{mean value over the training set of } (\hat{a} - a^{(m_{\max})})^2.$$

How to update the weights and biais? Use gradient descent for our cost function!

$$w^{\text{new}} = w - \gamma \nabla C$$

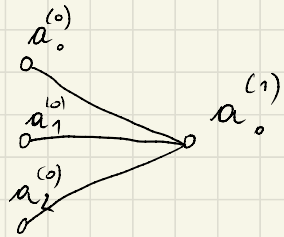
To compute ∇C, you need to the partial derivatives of C with respect to each w_{ij} and $b_i^{(m)}$. `https://www.youtube.com/watch?v=IHZwWFHWa-w`

Neural network: an introduction with digits recognition

It classifies 96 % of the new images correctly. If you change a bit the structure of the hidden layers, your get 98. The state of the art performance is 99,79 %.

We never told the network how to recognize digits. The network is not at all looking like we might think (recognizing segments and loops for instance). Also, if you feed it with a random imagine, it will provide an answer showing a great confidence.

Neural network: the simplest NN ever



$$a_0^{(1)} = \sigma\left(w_{00}^{(0)} a_{(0)}^{(0)} + w_{01}^{(0)} a_1^{(0)} + w_{02}^{(0)} a_2^{(0)} \right)$$

$$\text{cost function} \quad C(w) = \sum_{\ell=1}^{n} \left(a_{0\,\text{true}}^{(1)\ell} - a_0^{(1)\ell} \right)^2$$

$a_0^{(1)\ell}$: computed with NN i.e. the formula above

$\sum_{\ell=1}^{n}$: sum over training set

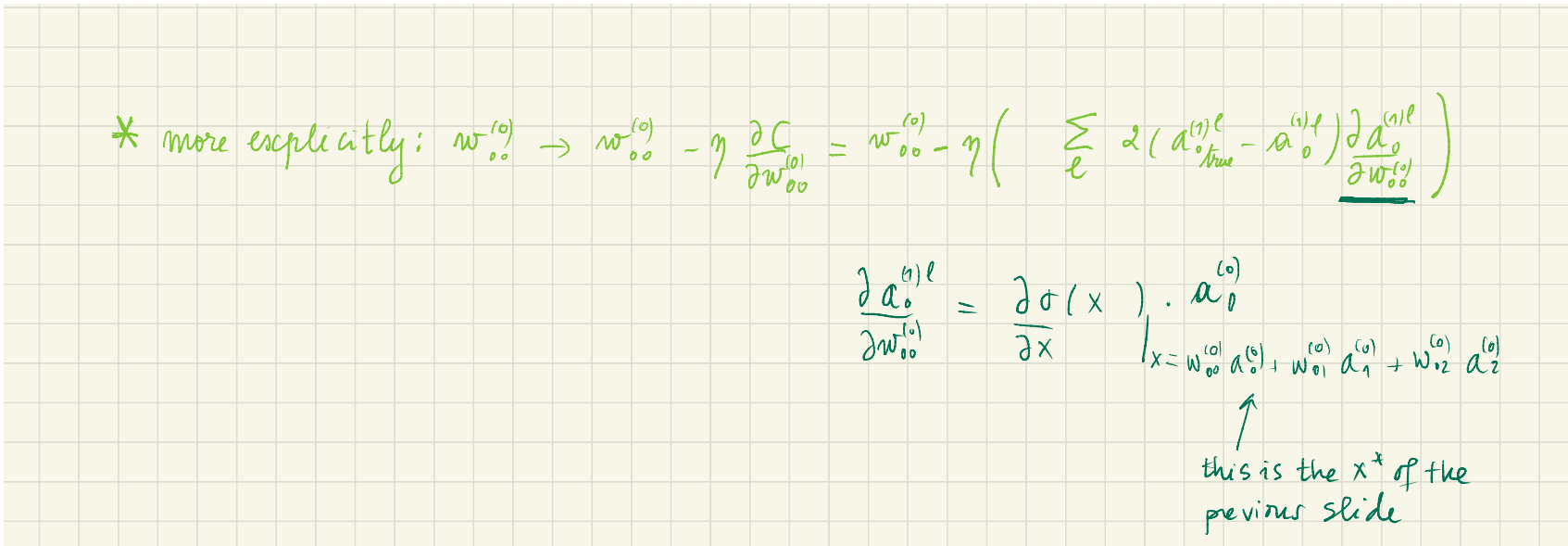
NN algorithm

1. Assign random values to the parameters of the NN, i.e. to $w_{00}^{(0)}$, $w_{01}^{(0)}$ and $w_{02}^{(0)}$.
2. compute the cost function $C(w)$.
3. go down the cost function: $\mathbf{w} \to \mathbf{w} - \eta \frac{\partial C}{\partial \mathbf{w}} \overset{*}{=} \mathbf{w} - \eta 2 \sum_\ell (a_{\text{true}}^{(1)\ell} - a^{(1)\ell}) \frac{\partial \sigma(x)}{\partial x}\Big|_{x=x^*} \mathbf{a}^{(0)}$

   where η = learning rate.
4. repeat step 3. (for the "simplest NN ever", you simply define in advance how many times you repeat that step.)

*see next slide for an explicit derivation.

Neural network: the simplest NN ever



Exercise: implement this NN and check that it is performing quite well despite its simplicity.

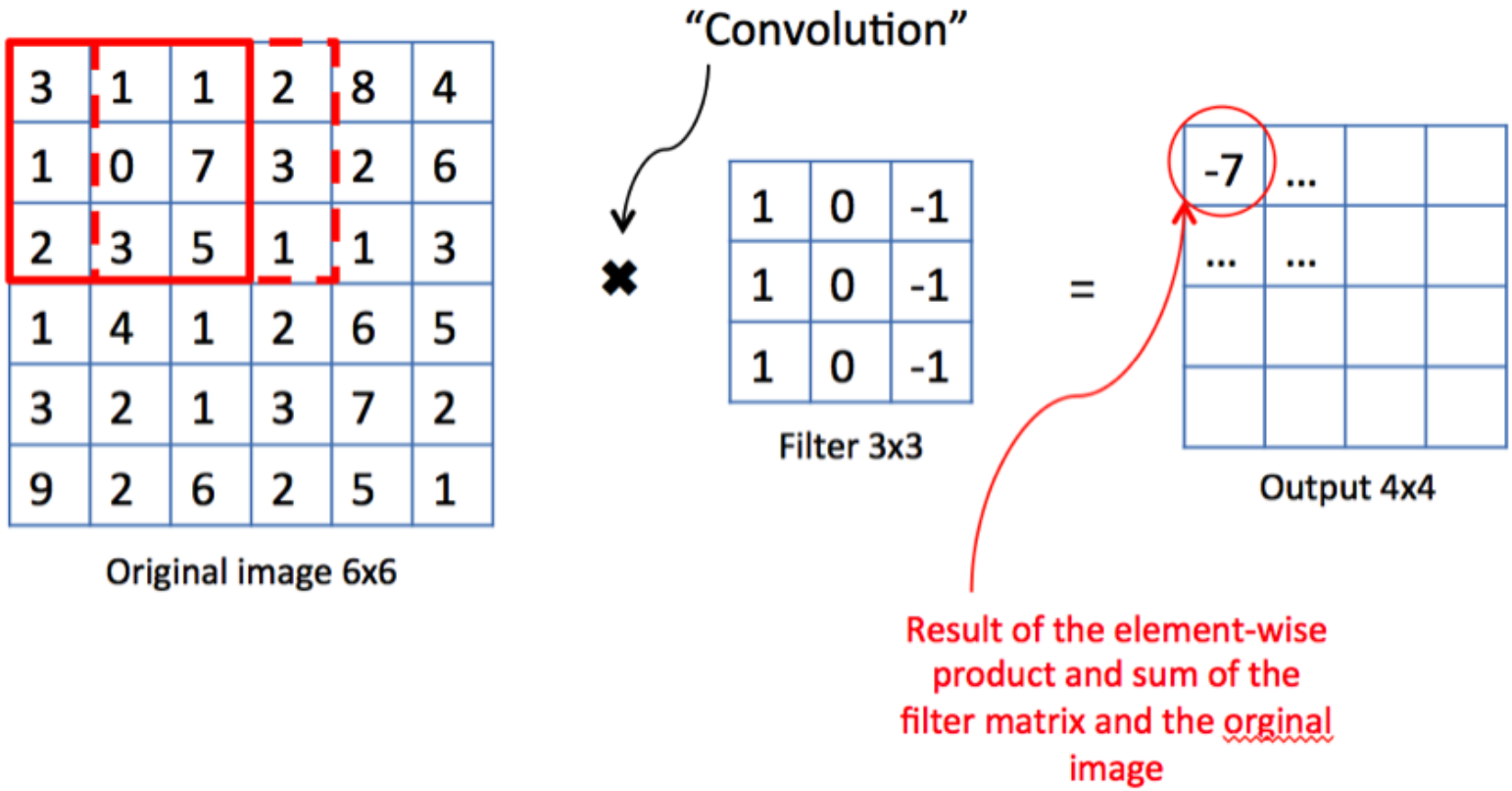
Beyond ‘simple‘ neural networks

Deep learning (DL) is a class of machine learning techniques that uses techniques that allow a system to automatically discover the representations needed for feature detection or classification from raw data.

DL processes the input data by using many layers and nonlinear transformation of the input. The different layers correspond to different levels of abstraction of the input data.

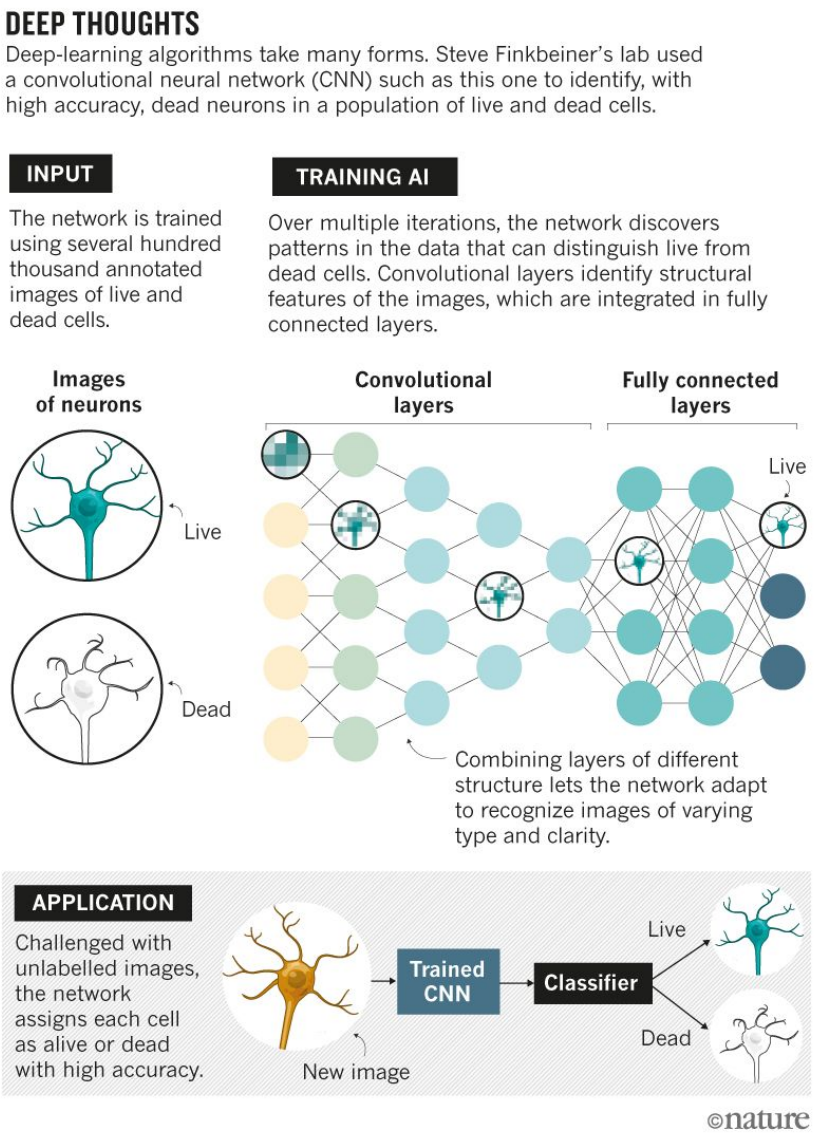
Beyond 'simple' neural networks : Convolutional Neural Network

One example of DL algorithm is a Convolution Neural Network (CNN). CNN have special layers called convolutional layers that can detect patterns (such as edges, circles,...), the main idea is that you define a 'filter' (it is simply a matrix of the size of your choice) and you convolve this matrix by all matrices of the same size from the input.



https://www.youtube.com/watch?v=YRhxdVk_sIs

One example


DEEP THOUGHTS
Deep-learning algorithms take many forms. Steve Finkbeiner's lab used a convolutional neural network (CNN) such as this one to identify, with high accuracy, dead neurons in a population of live and dead cells.
INPUT
The network is trained using several hundred thousand annotated images of live and dead cells.
TRAINING AI
Over multiple iterations, the network discovers patterns in the data that can distinguish live from dead cells. Convolutional layers identify structural features of the images, which are integrated in fully connected layers.
Images of neurons
Live
Dead
Convolutional layers
Fully connected layers
Live
Combining layers of different structure lets the network adapt to recognize images of varying type and clarity.
APPLICATION
Challenged with unlabelled images, the network assigns each cell as alive or dead with high accuracy.
New image
Trained CNN
Classifier
Live
Dead
©nature

1. Introduction
2. Technique #1 : Linear regression
3. Technique #2 : Linear discriminant analysis (LDA)
4. Technique #3 : Neural Networks
5. Resampling methods
6. Shrinkage methods : Ridge and Lasso

[see chapter 5 in the book `http://faculty.marshall.usc.edu/gareth-james/ISL/`]

Resampling methods

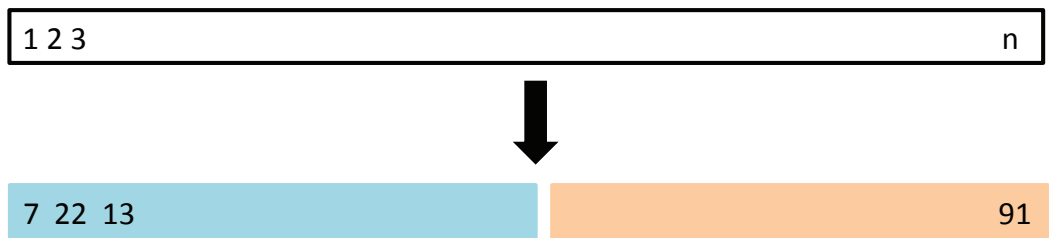
Idea : To get more out of one data set, repeatedly draw samples from a training set and refit the model.

These methods can be used to estimate the test MSE for instance. We will see the most used methods:

(1) Cross-validation which can be used for **model assessment** (=estimate quality of a model) or **model selection** (=estimate best level of flexibility). Cross validation can be used to estimate the test MSE.

(2) Bootstrap which is typically used **to estimate the error made on a parameter** for a specific model.

Cross-validation : The Validation Set Approach



Idea: split the dataset into a **training set** (in blue here) and a **test set** (in beige) $\Rightarrow$ train the model with the training set and test it (i.e. you compute the test MSE) with the test set.

Cross-validation : The Validation Set Approach

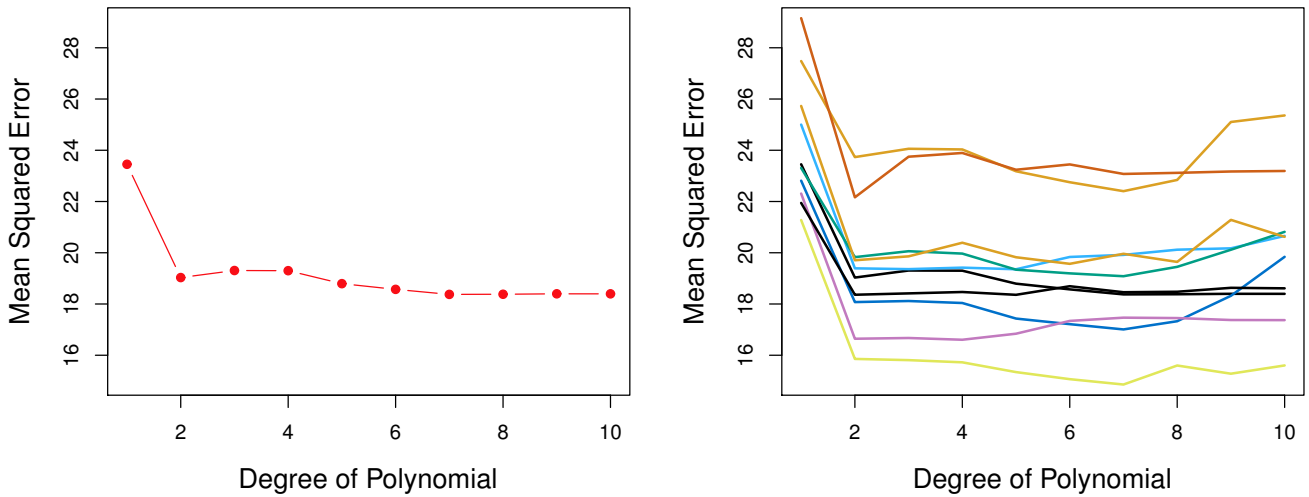
Idea: split the dataset into a **training set** (in blue here) and a **test set** (in beige) ⇒ train the model with the training set and test it (i.e. you compute the test MSE) with the test set.

The validation set approach : Estimate the test error rate by holding out a subset of the training observations from the fitting process (divide data into two sets), and then applying the statistical learning method to those held out observations.

blue : training set (it contains among others samples 7, 22, and 13)

beige : validation set (it contains among others the sample 91)

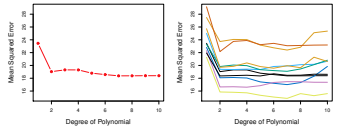
Cross-validation : The Validation Set Approach - not robust



Left: test MSE computed on one choice of slipping of the data.

Right: idem for various splittings $\Rightarrow$ the method is not robust against different splitting choices for the training and test sets.

Cross-validation : The Validation Set Approach - not robust



Left: test MSE computed on one choice of slipping of the data.

Right: idem for various splittings ⇒ the method is not robust against different splitting choices for the training and test sets.

The figures represent the estimations of the test MSE obtained from fitting the data with polynomial functions of various degrees. The procedure for each polynomial fit is to train (i.e. find the coefficient of the polynomial function) on the training set and the compare the prediction of that model and 'true' values on the test set.

Cross-validation : Leave-One-Out Cross-Validation

Same idea as 'The validation set approach' but dealing with the bad aspects:

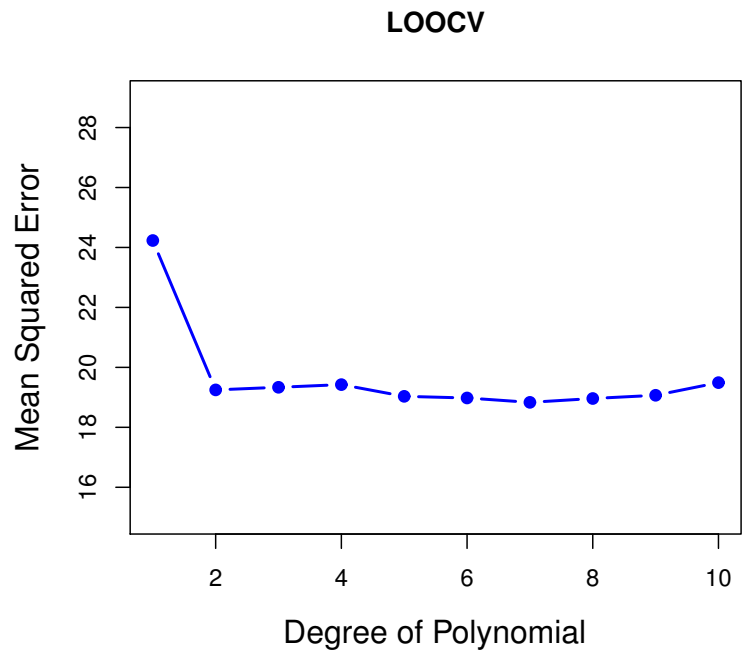
The estimation of the test MSE is

$$\text{test } \hat{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} \text{MSE}_i \,,$$

where MSE_i is the estimation made on the test set consisting of observation i.

Cross-validation : Leave-One-Out Cross-Validation - pros and cons

☺ no more randomness in the data splitting choice, i.e. the method is essentially unbiased,

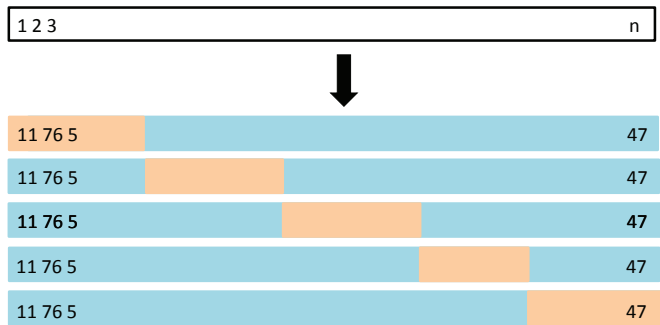


☹ We need to refit the model n times...

Exception with the 'Magic formula': for linear or polynomial fitting, we can compute MSE_i we do not need to refit n times a model $\text{MSE}_i = \sum(\frac{y_i - \hat{y}_i}{1-h_i})^2/n$ where $\hat{y}_i$ is the estimation of y_i with the original regression method and $h_i = 1/n + \frac{(x_i - \bar{x})^2}{\sum_{i'}(x_{i'} - \bar{x})^2}$.

Cross-validation : k-fold Cross-Validation

An intermediate solution between the two previous methods: randomly divide the set of observations into k groups, or folds, of approximately equal size.



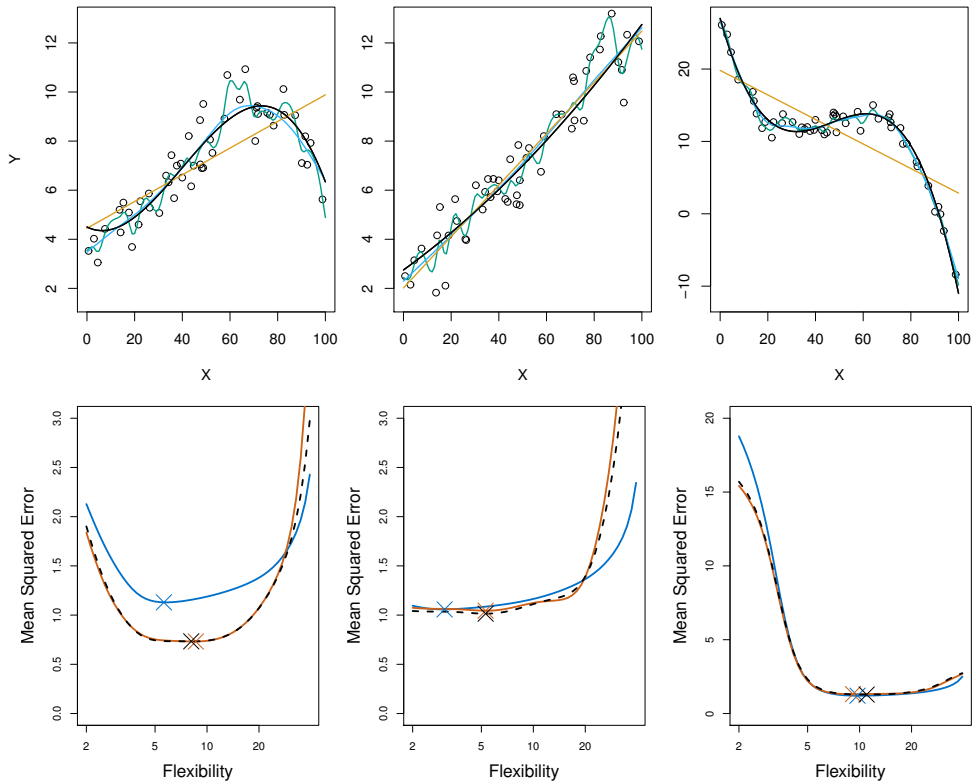
The estimation of the test error is given by

$$\mathrm{CV}_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \mathrm{MSE}_i \,,$$

where MSE_k = error on the kth fold obtained by fitting on the rest.

k-fold cross-validation necessitates to fit k times the statistical learning method. Choosing $k = 5$ or 10 for instance will lower the number of fitting to do over Leave-One-Out Cross-Validation ($k = n$).

Cross-validation : do the different methods estimate well the test MSE?



true test MSE – LOOCV – Dashed: 10fold CV.

Both methods give very similar results. They tend sometimes to underestimate the true test MSE. They are however in general good at finding the optimal flexibility which is given by the minimum of the test MSE curves.

Cross-validation : bias versus variance

(1) LOOCV has approximatively unbiased estimated . Why?

LOOCV has low bias because it is averaging over n models (i.e. over many 'models').

(2) LOOCV has higher variance than k-fold CV . Why?

LOOCV has higher variance than k-fold because in LOOCV all models are trained on almost the same training sets, effectively this gives n times the same 'model'. With the k-fold testing we average over less models but the models are less correlated.

$\Rightarrow$ A choice typically done is to go for k-fold CV with $k = 5$ or 10. This amounts to make a bias-variance trade-off.

[the bias is the error due to the choice of model; the variance is the error due to the choice of training set]

CV on classification problems

Instead of using the MSE, we use the number of misclassified observations. For the LOOCV for instance, we would estimate our 'test' classification error as:

$$\text{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^{n} \text{Err}_i$$

Bootstrap : Method illustrated with an example

Suppose that we want to invest a fixed sum of money in two financial assets X and Y (they are random variables).

What is the fraction α of the money that we should invest in X?

One can show that the value that minimizes the risks (ie that minimizes $\text{Var}(\alpha X + (1-\alpha)Y)$ is

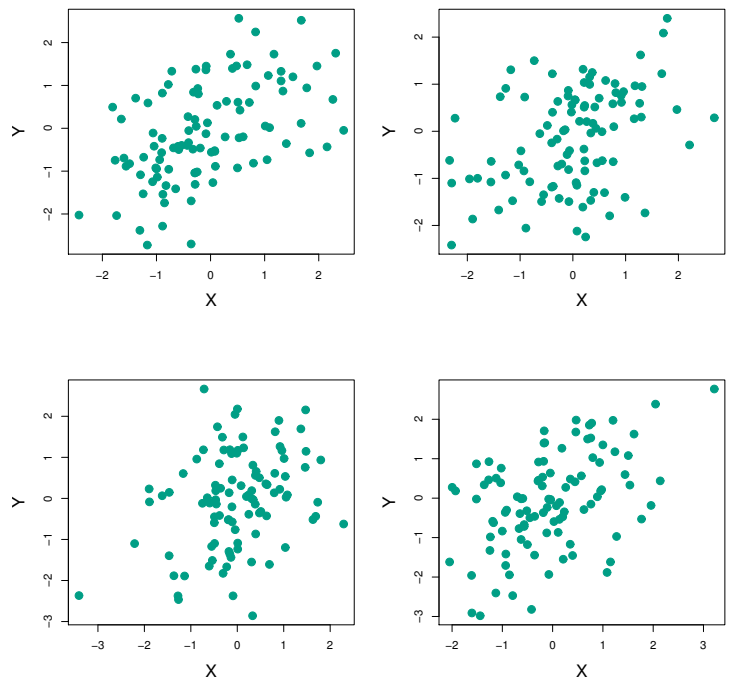
$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

If we had access for instance to 1000 datasets containing each 100 pairs of observations X and Y, we could then estimate α and its variability. We would get 1000 estimations of α (called $\hat{\alpha}_i$) and deduce:

$$\hat{\alpha} = \frac{1}{1000}\sum_{i=1}^{1000}\hat{\alpha}_i \qquad \text{and} \qquad \hat{\sigma}_\alpha = \sqrt{\frac{1}{1000-1}\sum_{i=1}^{1000}(\hat{\alpha} - \hat{\alpha}_i)^2}$$

Bootstrap : Method illustrated with an example

In general we do not have a such big dataset. Let us do instead an 'in silico' experiment and draw 1000 times 100 pairs from a bivariate normal distribution (X, Y) with e.g. zero means and $\sigma_X^2 = 1$, $\sigma_Y^2 = 1.25$ and $\sigma_{XY} = 0.5$:



We get a very good estimate of α, indeed the 'true' α which we know by construction of the dataset is 0.6 and our estimation from the 1000 experiments gives $\hat{\alpha} = 0.5996$.

The estimated standard deviation is 0. 083 which is also close to the real value.

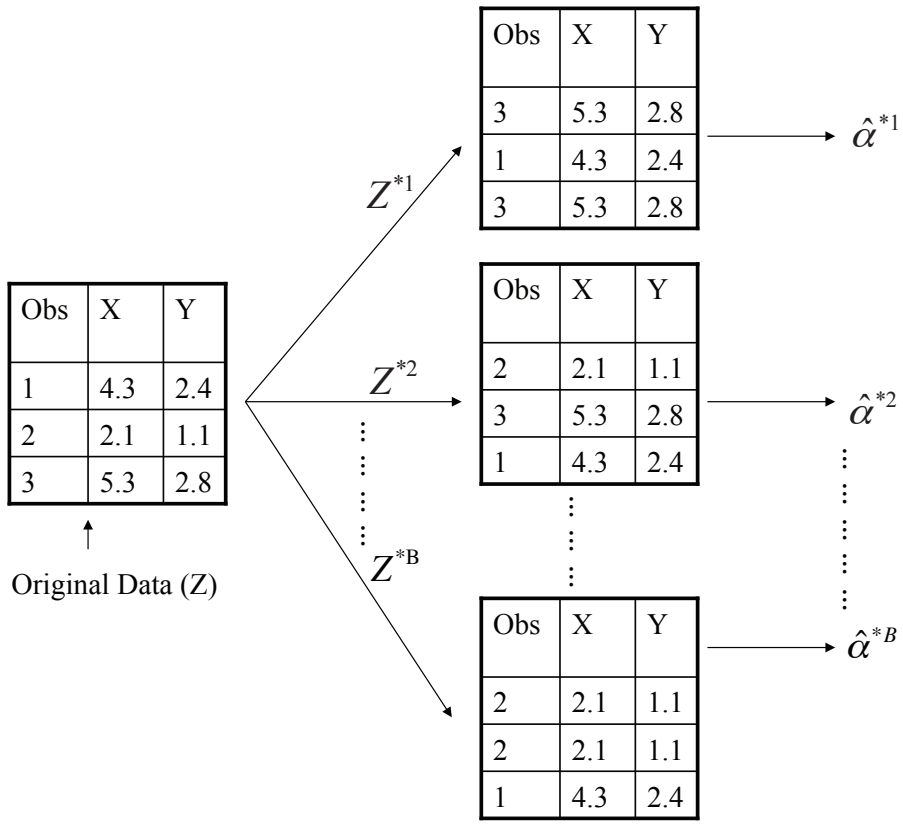
Here we represented only 4 out of the 1000 datasets.

Bootstrap : Method illustrated with an example

Generally we are interested in 'real' problems and not in in silico experiments...
Bootstrapping is a technique that emulates the fact of getting new datasets.

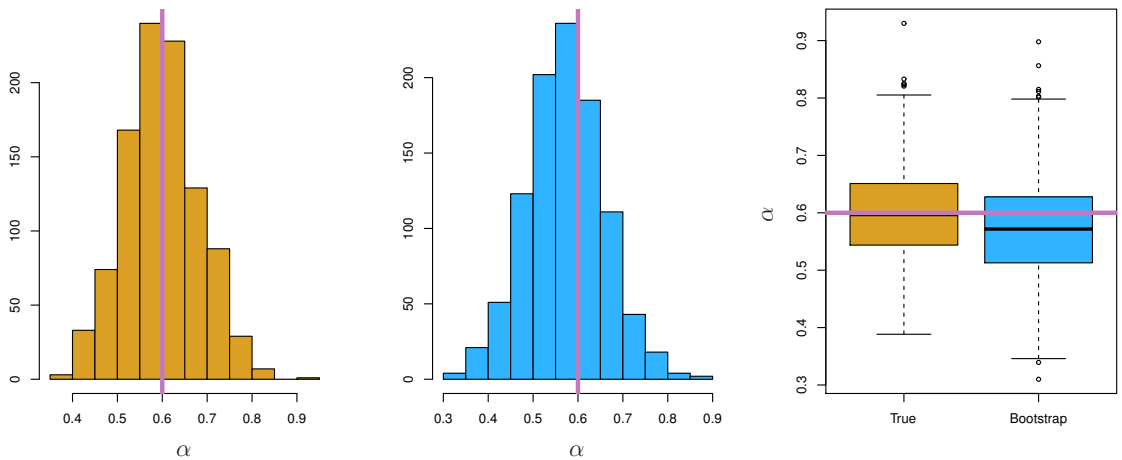
Bootstrap : Method illustrated with an example

Rather than repeatedly obtaining independent data sets, we obtain distinct datasets by repeatedly sampling observations from the original data set.



Bootstrap : Method illustrated with an example

Drawing new observations vs. bootstrap (histograms of estimations):



$\Rightarrow$ bootstrapping is very powerful!

1. Introduction
2. Technique #1 : Linear regression
3. Technique #2 : Linear discriminant analysis (LDA)
4. Technique #3 : Neural Networks
5. Resampling methods
6. Shrinkage methods : Ridge and Lasso

[see section 6.2 in the book **http://faculty.marshall.usc.edu/gareth-james/ISL/**]

Ridge regression

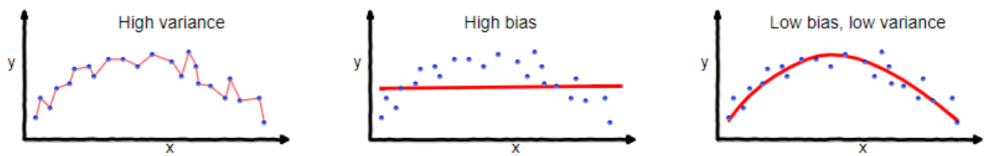
By minimizing a 'penalized' residual sum of squares (RSS), one can effectively decrease the test MSE.

For a linear regression for instance, if we replace the RSS by

$$RSS \Rightarrow RSS + \lambda\beta_1^2 \, .$$

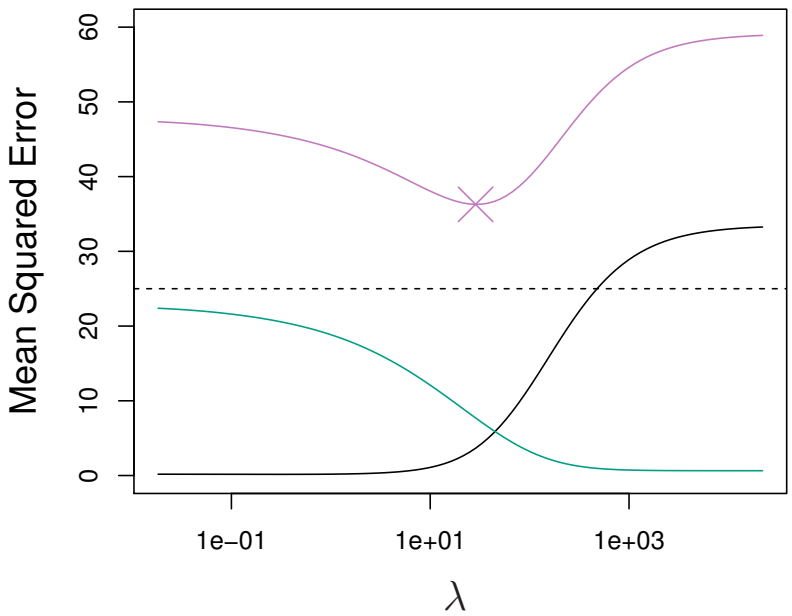
A high λ (called a shrinkage penalty) favors small values of β_1 and thereby reduces the flexibility of the model. This means that 'bias part of the error' (which is related to the choice of method) is increased because our method is effectively less flexible when we impose restrictions on the parameters.

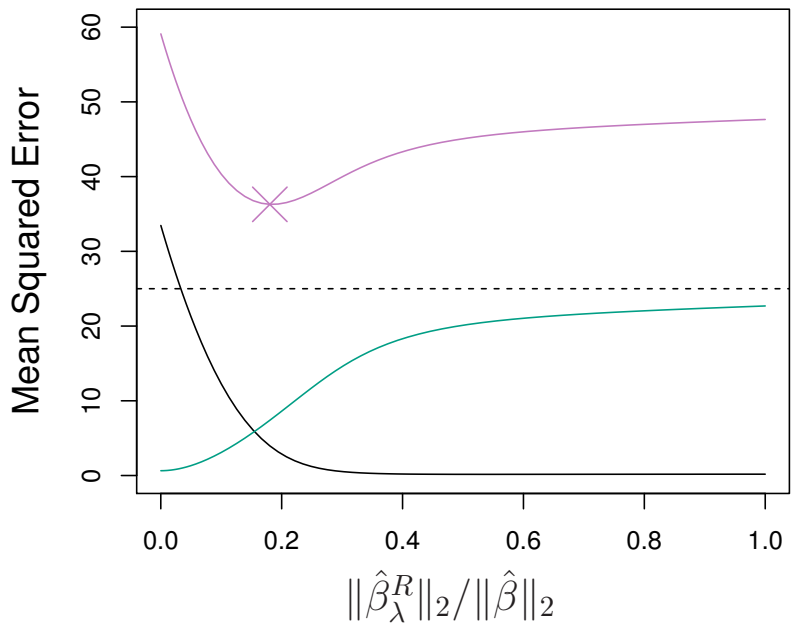
The interesting point is that the 'variance part of the error' will decrease as a less flexible method will be less sensitive to the choice of training set.



overfitting **underfitting** **Good balance**

https://towardsdatascience.com/bias-variance-and-regularization-in-linear-regression-lasso-ridge-and-elastic-net-8bf8199


Mean Squared Error
0
10
20
30
40
50
60
1e−01
1e+01
1e+03
λ


Mean Squared Error
0
10
20
30
40
50
60
0.0
0.2
0.4
0.6
0.8
1.0
$\|\hat{\beta}_\lambda^R\|_2/\|\hat{\beta}\|_2$

Lasso regression

It is the same idea as for the ridge regression. The only difference is that instead of taking the square of the parameters, we take their absolute values.

This causes one main difference. Now the parameters associated to the shrinkage can become zero. This can be good for models with a large number of parameters. Since with a Lasso regression these parameters can be set to zero, it allows to do **parameter selection**.

To quote the book: “In general, one might expect the lasso to perform better in a setting where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or that equal zero. Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size.”