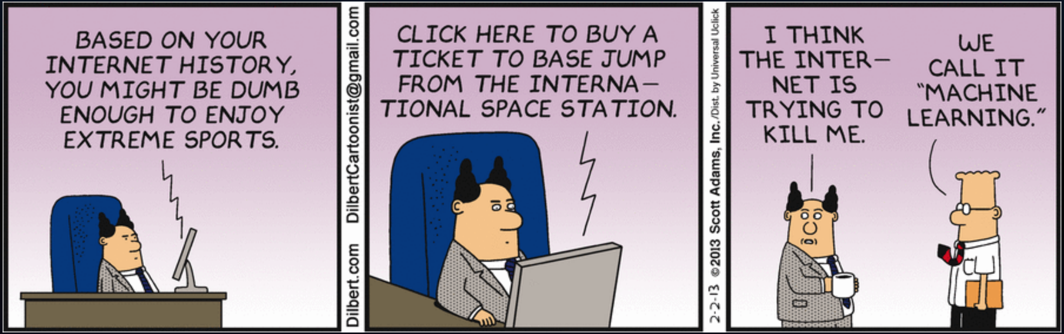
Machine Learning and Statistical Learning

Introduction

Ewen Gallic
ewen.gallic@gmail.com

MASTER in Economics - Track EBDS - 2nd Year



Source: https://dilbert.com/strip/2013-02-02%7D%7B2013-02-02.

Data everywhere

With the **era of big data**, the **volume** of data has considerably grown over the past years.

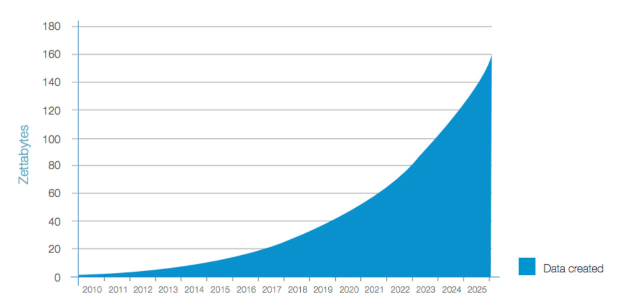


Figure 1: Annual size of the data volume.

Source: Data Age 2025: The Evolution of Data to Life-Critica Reinsel, D., Gantz J., et Rydning, J. (2017).

Data everywhere

Data also take more varied types (numbers, texts, images, videos, ...) and may come in **structured** or **unstructured** form.

The large amount of data requires **automated methods** of **data analysis**.

This is what **machine learning** provides.

Data everywhere: text

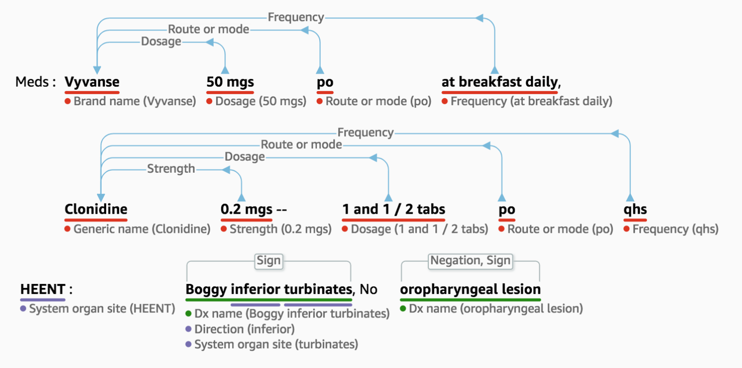


Figure 2: Amazon Comprehend Medical.

Source: Natural Language Processing for Healthcare Customers, Julien Simon (2018).

Data everywhere: text

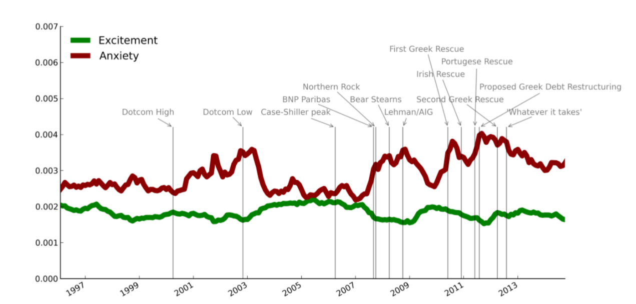


Figure 3: Excitement (green) and Anxiety (red) in RTRS (Reuters). The y-axis displays the individual aggregate word frequencies scaled by volume.

Source: Nyman et al. (2018)

See also: Text mining for central banks, Bholat, D et al. (2015)

Data everywhere: images

Let us consider these pictures.

Data everywhere: images

Figure 4: Face detection face-api.js (https://github.com/justadudewhohacks/face-api.js/).

Data everywhere: images

Figure 5: Expression recognition face-api.js (https://github.com/justadudewhohacks/face-api.js/).

1. Artificial Intelligence

1.1 A bit of history

A bit of history

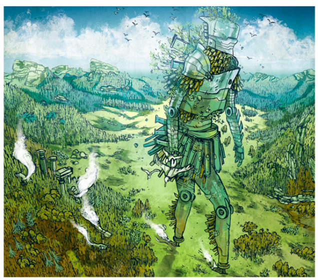


Figure 6: Talos (Source: Mathieu Bablet (2013). *Adrastée*. Ankama Éditions).

The dream of **creating robots or artifical beings that think** can be found as early as antiquity, in Greek myths, *e.g.*:

- Pygmalion (sculptor who fells in love with its statue which changed to a woman thanks to Aphrodite, the goddess associated with love)
- Talos (automaton made of bronze forged by Hephaestus the god of blacksmiths, in charge of protecting Crete)

A bit of history

Around 350 BC, the Greek philosopher Aristotle designed some **formal logic** (from Logos: "word", "reason") aimed at determining whether an argument is valid or not:

- the syllogism (from Syllogismos, "conclusion", "inference"): two premises leading to a conclusion
  - All mens are mortal
  - Socrates is a man
  - therefore Socrates is mortal

Figure 7: Bust of Aristotle.

A bit of history

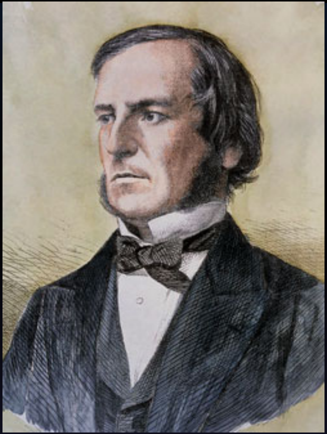


Figure 8: George Boole.

In the middle of the 19th century, the structures of mathematics were actively applied to logic. In particular, the British mathematician George Boole introduced a branch of algebra in which the values of the variables are the truth values (values indicating the relation of a proposition to truth) which was later called **Boolean algebra**.

A bit of history

In 1642, the mathematician Blaise Pascal invented the first digital calculating machine.

Figure 9: Four of Pascal's calculators and one machine built by Lépine in 1725, Musée des Arts et Métiers.

Figure 10: Watercolour portrait of Ada King, Countess of Lovelace, circa 1840, possibly by Alfred Edward Chalon.

Between 1842 and 1843, Ada Lovelace translated an article from the Italian mathematician Luigi Menebrea on the Analytical Engine (a general-purpose computer). She added notes to the articles and wondered whether such machines might become intelligent (the first general-purpose computers were built in the late 1940s).

A bit of history

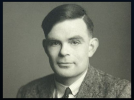


Figure 11: Alan Turing.

In 1950, Turing (1950) suggested that machines could do as humans: use available information and reason to solve problems and make decisions:

- I propose to consider the question, "Can machines think?"

A few years later, in 1956, John McCarthy, Marvin Minsky, Claude Shannon, and Nathaniel Rochester hosted a summer workshop at Dartmouth College (the Dartmouth Summer Research Project on Artificial Intelligence), a seminal event on machine learning.

Figure 12: Some participants of the DSRPAI.

A bit of history

Since then, the field of **artificial intelligence** has grown.

Some problems that are **intellectually difficult for humans** were **solved by computers**: those that can be described as a list of mathematical rules.

But other tasks that may be really easy for humans to do but **hard to formally describe** (such as interpreting a handwritten text, or recognizing a cat on a picture, *i.e.* problems we solve intuitively) have proven to be more difficult to solve with a machine.

A brief overview

The objective of the rest of the section is to provide a brief overview of what artificial intelligence is.

We will not go into details at this point.

We are going to explain different approaches to artificial intelligence, some of which will be studied in a little more detail in the following chapters:

- Knowledge-based approach
- Machine learning (in this course)
- Representation learning
- Deep learning (with Pierre Michel)

1.2 Knowledge-based approach

Knowledge-based approach

Figure 13: Akinator, a knowledge based system.

Some attempts were made in artificial intelligence projects to capture the knowledge of humans to help in the process of **decision making**.
In these projects, the computer can reason about statements about the world provided using a formal language.
This branch of artificial intelligence is known as **knowledge based approach**.

1.3 Machine learning

Machine learning

Knowledge based systems rely on **hard-code knowledge**.

Some other computer systems rather acquire their own knowledge, by extracting patterns from raw data.

They rely on **past observations** to **learn from experience**.

This capacity is known as **machine learning**.

Examples:

- automatic speach recognition
- fraud detection
- diagnosis of diseases
- ...

Definition

- According to Murphy (2012):
  - *machine learning* is defined as a set of methods that can ***automatically detect patterns in data***, and then use the uncovered patterns to ***predict future data***, or to perform other kinds of ***decision making under uncertainty***.
- We can also read in Athey (2018):
  - *machine learning* is a field that develops ***algorithms*** designed to be applied to ***datasets***, with the main areas of focus being ***prediction*** (regression), ***classification***, and ***clustering*** or ***grouping tasks***.

Machine learning

Usually, **machine learning** is divided in **two categories**:

- the **predictive** or **supervised learning** approach ;
- the **descriptive** or **unsupervised learning** approach.

In this introduction, we will briefly give an overview of these two categories.

1.3.1 Supervised learning

Supervised learning

The goal of the supervised learning approach is to learn **a mapping** from inputs $\mathbf{x}$ to outputs y, given a **labeled set** of input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where:

- $\mathcal{D}$ is the **training set**
- n is the number of training examples
- $\mathbf{x}_i$, *i.e.*, each training example is a vector a numbers called **features**, **attributes**, **covariates** or **explanatory variables**:
  - they are usually stored on a $n \times p$ **design matrix**
  - but their structure may be more complex, such as an image, a text, a sequence, a graph, ...
- Y_i is the **response variable**:
  - it can be a **categorical** or **nominal** variable from a finite set
  - or a **real-valued** scalar.

Supervised learning: classification and regression

- As we know the real value of y_i, it is possible to compare the prediction with the observable and therefore compute **error metrics**.
- When the response variable y_i is categorical, the problem is known as **classification** (or pattern recognition).
  - detecting if an e-mail is ham or spam
  - recognizing parts of speech (verbs, subject, pronouns, etc.)
  - face detection on an image
  - market segmentation
  - ...
- When the response variable y_i is a real-valued scalar, the problem is known as **regression**.
  - predict the wage of an individual
  - predict the value of a financial asset
  - predict the temperature at any location in a building
  - ...

Supervised learning

With supervised learning problems, we assume that there exists a relationship between the input variables $\mathbf{x}$ and the output variable y:

$$y = f(\mathbf{x}) + \varepsilon,$$

where f is a **fixed but unkown function** of the predictors, and ε is a **random error term**.

Supervised learning: estimating f

We are interested in estimating the function f, for two main reasons:

1. to **predict** the value of y for some inputs that may not be available
2. to understand how the value of y is affected by variations of the predictors, *i.e.*, for **inference** purposes.

Supervised learning: estimating f for prediction

If we are interested in estimating f for prediction purposes:

- we want to get $\hat{y} = \hat{f}(\mathbf{x})$ where $\hat{f}$ is the estimation of f
- we may not be interested that much in the exact form of $\hat{f}$ and may view it as a *black box*... as long as it gets **accurate predictions**
- however, in EU, with the Article 22 of the *General Data Protection Regulation*, this *black box* may represent an issue, as data subjects might have a right to **explainability**.

Supervised learning: estimating f for inference

When we are interested in estimating the mapping from $\mathbf{x}$ to y for **inference purposes**, we want to know how variations in the inputs $\mathbf{x}$ affect the output y.

In that case, we may want to know what are the **important predictors** among $\mathbf{x}$ that can explain the variations of the response.

Besides, we may want to know more about the relationship between predictors and the response:

- what is the magnitude?
- what is the sign of the relationship?
- is it linear? non-linear?

Classification

Let us consider some inputs $\mathbf{x}$ that we want to map to an output y. The output y takes its values from a finite set, *i.e.*, $y \in \{1, \ldots, C\}$, with C the number of classes.

- **If the value of C is 2**:
  - the problem is called a **binary classification**
  - for example, $y \in \{0, 1\}$ with 0 corresponding to a negative growth rate for an asset and 1 to a positive one
- **If the value of C is greater than 2**:
  - the problem is called a **multiclass classification**
  - for example, $y \in \{1, 2, 3, 4\}$ with 1 corresponding < 18 years old, 2 to $18 - 25$ yo, 3 to $26 - 55$ yo and 4 to > 55 yo.
- If the class labels are not mutually exclusive:
  - the problem is called **multi-label** classification
  - for example, $18 - 25$ years old and "woman"

Classification

As what we want to accomplish is a mapping from inputs to outputs, the problem corresponds to a **function approximation**:

- We first assume that $y_i = f(\mathbf{x}_i)$, $i = 1, \ldots, n$ for some **unknown function** f
- We want to learn how to **estimate** f, *i.e*, obtain $\hat{f}$, given a **labeled training set** $\mathbf{x}$
- Then, we would like to make predictions $\hat{y_0} = \hat{f}(\mathbf{x}_0)$, where $\mathbf{x}_0$ are new inputs.

Classification: example

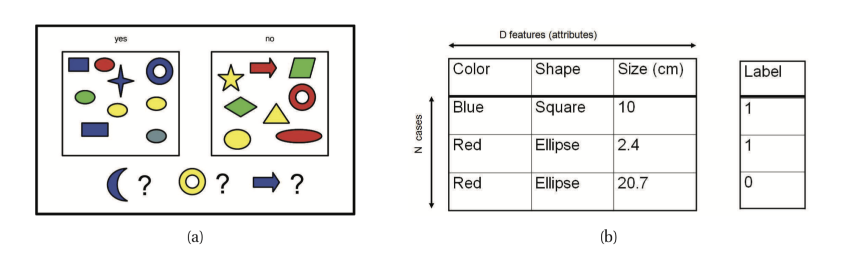


Figure 14: Left: Some labeled training examples of colored shapes, along with 3 unlabeled test cases. Right: Representing the training data as an $n \times p$ design matrix. Row i represents the feature vector x_i. The last column is the label, $y_i \in \{0, 1\}$.

Source: Murphy (2012)

We want to classify the new inputs (the blue crescent, the yellow circle and the blue arrow). These inputs were not observed before.

Classification: example

- The blue crescent: all blue items are classified as “yes”, which corresponds to the label 1.
  - It thus may be a good guess to label the blue crescent as 1
- The yellow circle: some circles are labeled 1 and other 2, some yellow objects are labeled 1 and other 2.
  - It is harder to decide the label to assign to the yellow circle
- The blue arrow: while the other arrow was labeled 0, all blue objects were labeled 1.
  - It is also unclear here.

Classification: probabilistic prediction

With the previous example, we can understand that it may be a good idea to return a **probability** associated with the label.

Let $p\left(y \mid \mathbf{x}, \mathcal{D}\right)$ be the probability distribution of y given the input vector $\mathbf{x}$ and traning set $\mathcal{D}$.

If the number of classes C is equal to two, we have:

$$p\left(y=0 \mid \mathbf{x}, \mathcal{D}\right) + p\left(y=1 \mid \mathbf{x}, \mathcal{D}\right) = 1$$

The "best guess" as to the **true label** can be set accordingly to the probability returned by the model, using:

$$\hat{y} = \hat{f}(\mathbf{x}) = \arg\max_{c=1}^{C} p\left(y=c \mid \mathbf{x}, \mathcal{D}\right)$$

- This is the **mode of the distribution** (the most probable value).

Classification: spam or ham

One the famous examples includes the classification of e-mails as ham or spam.

Some real world examples are presented on Kaggle (with Python or with R).

The basic idea consists in classifying an e-mail into two classes:

- ham: $y = 0$, the e-mail is not unsolicited or undesired
- spam: $y = 1$, the e-mail is unsolicited or undesired

We have a training set for which we know the true class of the message.

Classification: spam or ham

One approach consists in creating a Document Term Matrix (DTM), which describes the frequency of terms that occur in a collection of documents. Each document is an e-mail here.

The rows of the DTM correspond to a document, and the columns correspond to terms (words).

The element x_{ij} of that DTM corresponds to the occurrence of word j in email i.

The idea behind it is that some words, such as “buy”, “cheap”, “inherited”, “viagra”, “free”, … appear more frequently in spam than in ham.

Classification: handwriting recognition

Another example is that of **handwriting recognition**.

The famous MNIST (Modified National Institute of Standards) database contains a training set of 60,000 examples of handwritten digits (0 to 9) and a test set of 10,000 examples.

In this database, the digits have been size-normalized and centered to fit into a 28×28 pixel bounding box.

Each pixel of each 28×28 image have a grayscale value in the range $0 : 255$.

Classification: handwriting recognition

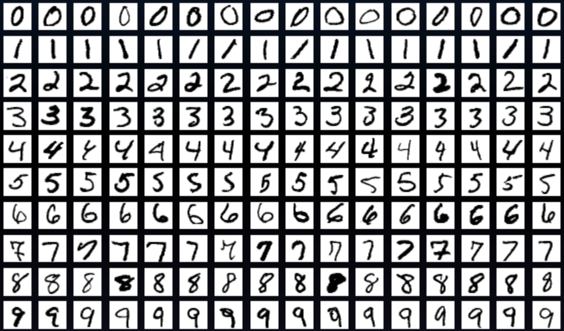


Figure 15: Sample images from the MNIST test dataset.

For each image, the correct class is already known.
You will work on this dataset with Pierre Michel.

Classification: face detection

Another classification problem is that of face detection: given a picture, we want to be able to detect if we can detect a face on it, and if so, where it is.

This is a problem of **object detection**.

To tackle this problem, one can proceed as follows:

1. transform the picture from RBG to Grayscale (it is easier to detect faces in the grayscale)
2. divide the image into many small overlapping patches at different locations, scales and orientation
3. classify each patch based on whether it contains face-like texture or not.
4. return the locations where the probability of face is high enough.

Classification: face detection

Both Figures are from Sung and Poggio (1998).

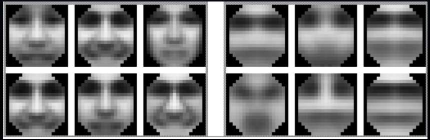


Figure 16: The 12 prototype patterns for approximating the distribution of face patterns. The 6 patterns on the left are "face" prototypes. The 6 on the right are "non-face" prototypes.

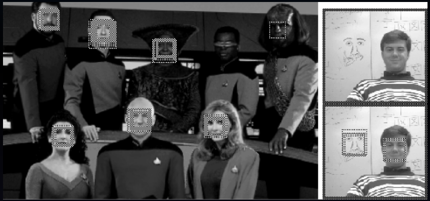


Figure 17: Face detection results.

Regression

In **regression problems**, the response variable is continuous.

As in the classification case:

- we assume that $y_i = f(\mathbf{x}_i)$, $i = 1, \ldots, n$ for some **unknowm function** f
- we want to estimate f given a **labeled training set** $\mathbf{x}$
- then we want to make predictions $\hat{y}_0 = \hat{f}(\mathbf{x}_0)$, where x_0 are new inputs.

Regression: example

For example, if our input $\mathbf{x}$ is the speed of cars and the response variable y is the stopping distance, we may be interested in estimating f such as $y_i = f(\mathbf{x}_i)$, for all $i = 1, \ldots, n$ training examples.

Table 1: Sample of the data

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

Regression: example

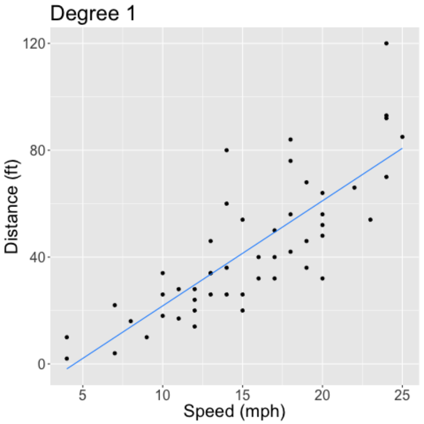


Figure 18: Degree 1

Regression: example

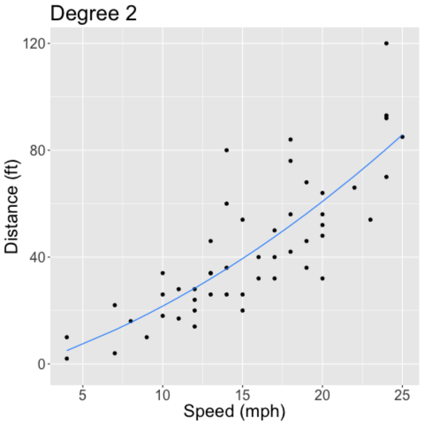


Figure 19: Degree 1

Regression: example

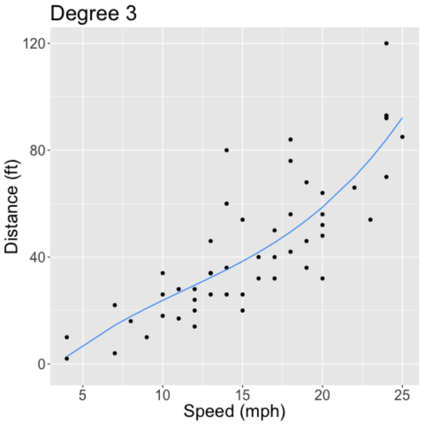


Figure 20: Degree 1

Regression: example

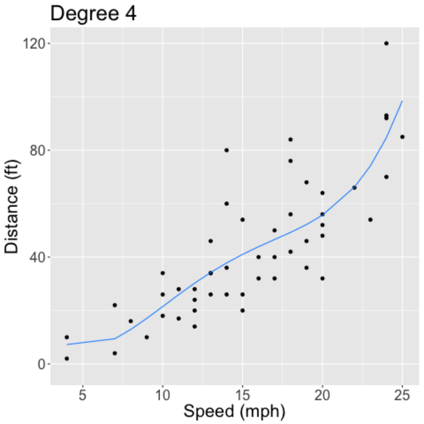


Figure 21: Degree 1

Regression: example

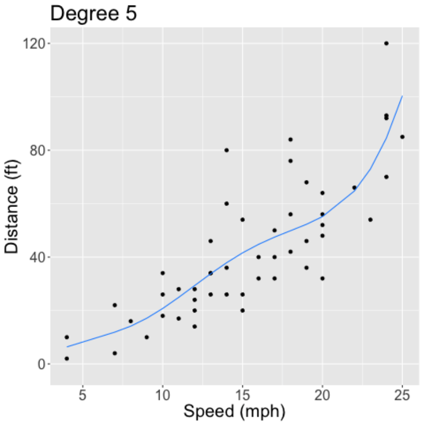


Figure 22: Degree 1

Regression: example

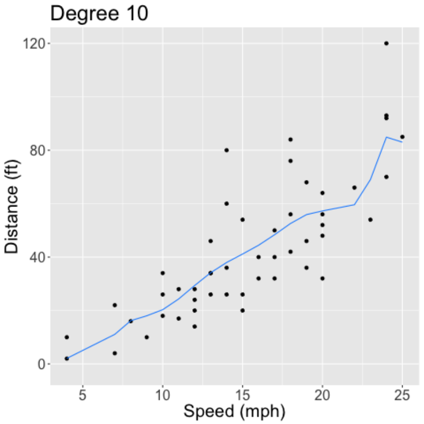


Figure 23: Degree 1

1.3.2 Unsupervised learning

Unsupervised learning

The goal of the **unsupervised learning** approach is to **discover patterns** in the data, to discover interesting things about the measurements on the inputs (such as finding subgroups)

It is often more challenging than supervised learning:

- it is more prone to **subjectivity**
- there is no labeled data, so that we do not know the kind of pattern to look for
- there is no simple goal for the analysis such as prediction of a response

In addition, as there is no response variable, it is not possible to check the results as **we don't know the "true answer"**:

- there is **no obvious error metric** to use

Unsupervised learning

Formally speaking, with **unsupervised learning**:

- we want to build models of the form $p(\mathbf{x}_i \mid \theta)$: it therefore corresponds to an unconditional density estimation
- $\mathbf{x}_i$ is a vector of variables which thus requires multivariate probability models ;

While with **supervised learning**:

- we want to build models of the form $p(y_i \mid \mathbf{x}_i, \theta)$: it corrresponds to conditional density estimation
- y_i is usually a single variable we want to predict and which requires a univariate probability model.

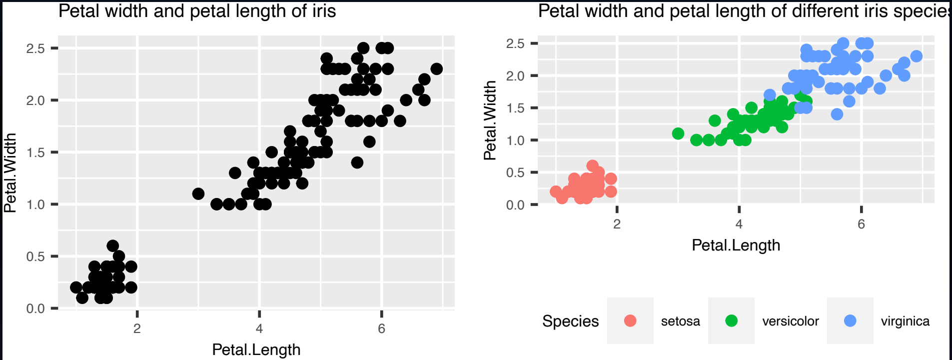
Clustering

A first example of unsupervised machine learning problem is that of **clustering** data into groups.

Let us consider a famous dataset, the Iris flower (Anderson 1935), which provides information on the sepal length and width, as well as petal length and width for 50 flowers from each of 3 species of iris (setosa, versicolor and virginica).

Figure 24: Iris flowers. (Source: Machine Learning in R for beginers.)

Clustering


Petal width and petal length of iris
Petal.Width
Petal.Length
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Petal width and petal length of different iris specie
Species
setosa
versicolor
virginica

Clustering

Let us assume we do not know *a priori* the different species of iris.

We see from the previous graph (the one on the left), that there may possibly be different subgroups in the data, different **clusters**.

We do not know how many of them. Let us say that there are K clusters.

Our goal is twofold:

1. we aim at **estimating the distribution** over the number of clusters, *i.e.*, $p\left(K \mid \mathcal{D}\right)$
   - as we do not know K, we can pick any value ; picking the "right" value is called **model selection**
2. we want to estimate which cluster each point belongs to.

Discovering latent factors

With the era of big data, not only the number of observations n has considerably grown, but also the number of variables p has exploded.

It is often very useful to **reduce the dimensionality** of the data:

- to do so, the data are projected to a lower dimensional subspace

Why doing so?

- most of the variability of the data may be explained by **latent factors** (factors that are not directly observed).

The statistical procedures of dimension reduction will be coverd in Sébastien Laurent's course.

Discovering graph structure

Sometimes, we face graph data. A **graph** is simply a collection of nodes (or vertices) and edges (or arcs) between them:

- for example, the nodes can be people on a network
- and the edges between two nodes may represent the fact that these nodes are connected

Both nodes and edges may have properties:

- the node of a person may provide information on the age, gender, etc.
- the edge between two nodes may state the date at which the these two nodes got connected.

Using unsupervised learning machine techniques on graphs may be useful to understand the structure of the graph and discover some structures that may be not be obvious to a human being (to detect community in a network or fraud, for example).

1.4 Representation learning

Representation learning

The performance of machine learning algorithms is tightly linked to the **representation** of the data they are given.

They often require inputs that are mathematically and computationally convenient to process.

For example, if an insurance company tries to predict the probability of death within the year of its clients, it provides its system some relevent information (*e.g.*, the age), some **variables** (also called **features**).

The algorithm learns how each of the features correlates with some outcomes, but it cannot influence the way that the features are defined.

Representation learning

Let us suppose that we wish to be able to detect bikes on pictures.

- We might use the presence or absence of wheels on the picture to do so.

But it may be a daunting task: the shadow of other objects falling on the wheel could make recognition difficult, as well as low luminosity or the presence of chromatic aberration...

To overcome this issue: using machine learning to:

1. discover the mapping from representation to output
2. discover the representation itself

This approach is known as **representation leaning**.

Representation learning

The **autoencoder** is an example of **representation leaning** algorithm.

It works in two steps:

- it firts converts the input data into a different representation, by means of an **encoder function** (it learns to compress input data)
- then it tries to generate from the compresses data a representation as close as possible to its original input, by means of a **decoder function** (it learns to uncompress).

1.5 Deep Learning

Deep Learning

To explain the observed data, whether designing variables ourselves or using algorithms for learning variables, the aim is usually to **separate factors of variation**.

These factors may or may not be observed directly. Plus, a huge number of factors may influence the data we want to explain.

While some of the factors may prove to be useful to explain the observed data, other should be discarded as they only add noise.

When it becomes almost as difficult to get a representation of the data (*e.g.*, detecting the presence of a wheel) as to solve the original problem (*e.g.*, detecting bikes in a picture), representation learning does not seem to be very handy...

This problem is overcome by **deep learning**, which introduces representations that are expressed in terms of other, simpler representations.

Deep Learning

The **multilayer perceptron** is an example of a deep learning model.

It consists in a mathematical function that maps input values to output values.

This function is a composition of many simpler ones. Each of them provides a new representation of the input.

To sum-up

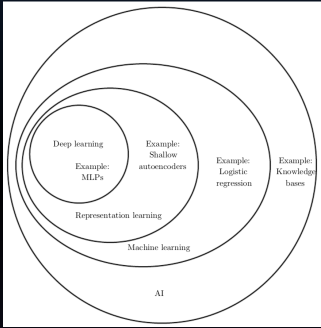


Figure 25: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI.

Source: Goodfellow et al. (2016)

And what about econometrics?

According to Varian (2014). *Machine Learning and Econometrics*:

- **Machine learning**, data mining, predictive analytics, etc. all use data to predict some variable as a function of other variables.
  - May or may not care about insight, importance, patterns
  - May or may not care about inference—how y changes as some x changes
- **Econometrics**: Use statistical methods for prediction, inference, causal modeling of economic relationships.
  - Hope for some sort of insight, inference is a goal
  - In particular, causal inference is goal for decision making

2. Some initial concepts

2.1 The estimation of f

The estimation of f

Let us consider that we have n observations of some inputs $\mathbf{x}$ and output y, for which we assume a relationship of the form:

$$y = f(\mathbf{x}) + \varepsilon$$

We wish to get $\hat{f}$, an estimate of f, so that:

$$\hat{y} = \hat{f}(\mathbf{x}).$$

The estimation of f

We will rely on the n observations to **teach** (or **train**) our method how to estimate f.

Let us denote $\mathbf{x}_{ij}$ the value of the j^{th} predictor, where $j = 1, \ldots, p$ for observation i, where $i = 1, \ldots, n$.

Let us denote y_i the output or response variable for the i^{th} observation.

Our **training data** containes n examples: $\{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i = (\mathbf{x}_{i1}, \ldots, \mathbf{x}_{ip})^\top$.

The estimation of f such that $y \approx \hat{f}(\mathbf{x})$ for any observation $(\mathbf{x}, y)$, can usually be characterized as either **parametric** or **non-parametric**.

2.1.1 Parametric methods

Parametric methods

To estimate f using a parametric method, we usually proceed in a two-step procedure:

1. We first **assume the functional form** of f
   - for example, we consider a linear relationship $(f(\mathbf{x}) = \beta_0 + \beta_1 \mathbf{x}_1 + ... + \beta_p \mathbf{x}_p)$
2. We **fit** (or **train**) the selected model
   - for the example where f is linear in $\mathbf{x}$, we estimate the parameters β_0 and β_j, $j = 1, \dots, p$

Parametric methods

Using a parametric form for f **simplifies the estimation problem**:

- it is easier to estimate some parameters than an arbitrary function f.

However, the model we choose usually does not match the **true unknown form of f**:

- as a consequence, if the model is too far from the true functional form, the estimation does not do a good job.

To overcome this issue, one may be tempted to select a more flexible model, which requires more parameters :

- but this may lead to a modeling error known as **overfitting** (more details will be given later on): the predictions correspond too closely or exactly to the data and therefore do not disentable the signal from the noise.

Parametric methods: example

Let us take an example, that of the salaries for Professors in the US in 2008-09.

The salary of a professor may be linked, among other things, to the number of years since he or she obtained their Ph.D and the number of years in activity (although we may suspect some colinearity between these two predictors).

$$\text{Salary}_i = f(\text{Years since Ph.D}_i, \text{Years of service}_i) + \varepsilon_i, \quad \forall i \in 1, \ldots, n.$$

Let us assume the following parametric form:

$$\text{Salary}_i = \beta_0 + \beta_1 \text{Years since Ph.D}_i + \beta_2 \text{Years of service}_i + \varepsilon_i$$

Parametric methods: example

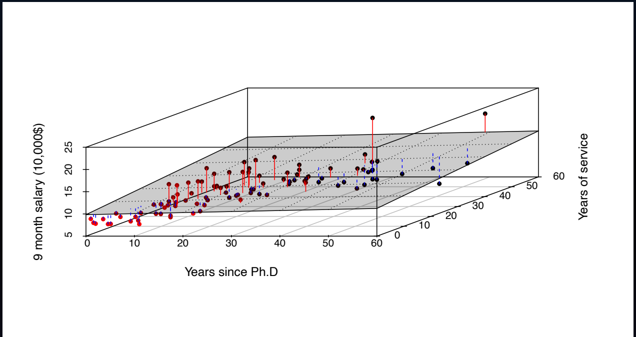


Figure 26: Linear model fit by OLS of salary as a function of years since Ph.D and years of service.

Some curvature are not well taken into account with this estimation.

2.1.2 Non-parametric methods

Non-parametric methods

With **non-parametric methods**, the functional form of f is not assumed in a first step, it is allowed to be obtained without guidance or constraints.

As they are **not assuming a specific functional form** for f, they can easily take into account **non-linearities** in the relationship between the response and the predictors.

Relatively to parametric methods, non-parametric ones require a greater number of observations for the estimation to be accurate:

- this comes from the fact that the problem is not reduced to estimate only a set of a few parameters

Non-parametric methods: example

Let us provide an example of a non-parametric estimation.

We can use the same data on the salary of professors, and estimate the relationship between salary and the years since Ph.D and.

To that end, we can estimate the function f by means of *thin-plate spline*.

Non-parametric methods: example (#1)

We can vary the **level of smoothness**: higher levels leading to getting closer to the perfect fit of the observations (and to overfitting!).

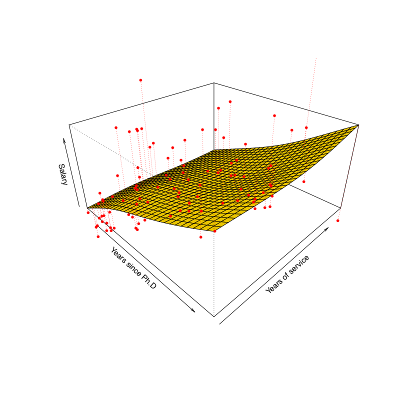


Figure 27: Thin-plate spline fit of salary as a function of years since Ph.D and years of service.

Non-parametric methods: example (#2)

We can vary the **level of smoothness**: higher levels leading to getting closer to the perfect fit of the observations (and to overfitting!).

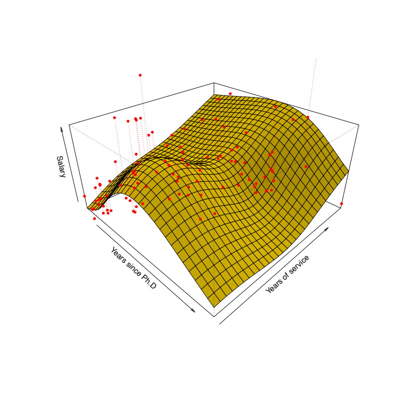


Figure 28: Thin-plate spline fit of salary as a function of years since Ph.D and years of service.

Non-parametric methods: example (#3)

We can vary the **level of smoothness**: higher levels leading to getting closer to the perfect fit of the observations (and to overfitting!).

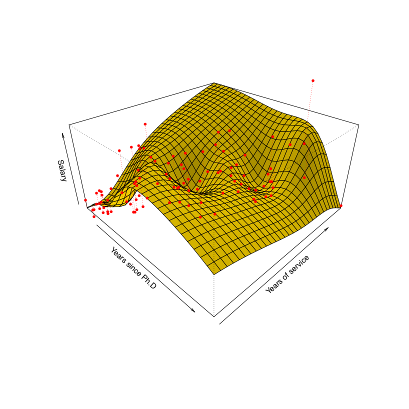


Figure 29: Thin-plate spline fit of salary as a function of years since Ph.D and years of service.

Non-parametric methods: example (#4)

We can vary the **level of smoothness**: higher levels leading to getting closer to the perfect fit of the observations (and to overfitting!).

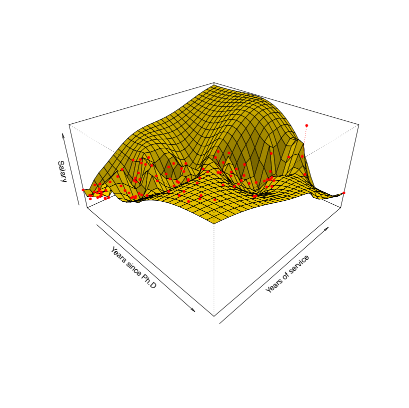


Figure 30: Thin-plate spline fit of salary as a function of years since Ph.D and years of service.

2.2 Prediction accuracy versus Model interpretability

Prediction accuracy versus Model interpretability

From the previous example, we saw that the linear model is far more **restricting** than the thin-plate spline one:

- the number of linear functions that can be fitted is far lower than the number of possible shapes that we can obtain with the thin-plate splines

Some models are less **flexible**, more restrictive than others.

- Less flexible models tend to be **less accurate** than relatively more flexible ones...
- but on the other hand, they produce results that are easier to interpret.

Prediction accuracy versus Model interpretability

There is therefore a **trade-off** between **prediction accuracy** and **model interpretability**.

Depending on the goal of the estimation, one might prefer giving-up some accuracy and turn to more restrictive model to get more interpretable results.

Prediction accuracy versus Model interpretability

Once again, the difference between **inference** and **prediction** is at play:

- if the aim of the analysis is inference, one might use a restrictive model
- if the aim is prediction, accuracy become more important and a more flexible model may be uses.

But be careful! Sometimes, more flexible models may not lead to more accurate predictions...

2.3 Model accuracy

Model accuracy

Once a model has been estimated, it is important to assess how good (or bad) the fit is.

For some specific data, it is also important to compare the results from different methods to select the one that "best fits" the data.

2.3.1 Quality of fit

Quality of fit

For **supervised learning methods**, as we can compare the fit with the observed value, it is easy to assess the performance of a given method.

In the context of a **regression**, several metrics can be used, among which the **mean squared error** (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(\mathbf{x}_i)\right)^2,$$

where $\hat{f}(\mathbf{x}_i)$ is the prediction for the ith observation.

- When the distances between the observations and the predictions are low, the MSE will be small
- When the distances between the observations and the predictions are high, the MSE will be large

Out of sample predictions

The quality of fit is usually measured not on the whole data set of observations, but rather on a subsample.

It seems more relevant to assess the quality of fit on previously unseen data (to make **out of sample predictions**):

- it tells us how good the model should perform **in the future**, when it is fed with new data
- it **avoids selecting a method that overfits the data**

In practical terms, what is done is to split the data into three subsamples, one for **training the model**, another one used for a **tuning process**, and a third one to **test the quality of fit**.

Method

The procedure, known as **the validation set approach** is as follows:

1. The learning method is fitted using a sample of **training observations** (**training data**): $\{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$:
   - the method learns from these data
   - this yields $\hat{f}$
2. Using $\hat{f}$ on previously unseen data (**validation/evaluation data**), the accuracy of the model is assessed:
   - this step helps choosing the method that gives the lowest MSE on validation data
3. Once the statistical learning procedure has been tuned, its accuracy is assessed on a third subsample of previously unseed data (**test data**):
   - this gives an "honest" assessment of the performance of the estimation

Size of the samples

Arguably the most defensible approach is to have three datasets of sufficient size: a training dataset, an evaluation dataset, and a test dataset. "Sufficient" depends on the setting, but a minimum of about 500 cases each can be effective. All three should be realizations from the same joint probability distribution. If there is only one dataset on hand that is at least relatively large (e.g., 1500 cases), a training dataset, an evaluation dataset, and a test dataset can be constructed as three, random, disjoint subsets. Then, there can important details to consider, such as the relative sizes of the three splits (Faraway 2014).

Figure 31: On the size of the samples.

Source: Berk (2008)

But :

- the split sample approach is only justified asymptotically
- in the case of skewed distribution, observations from the tail may not be included...

Splitting the data into subsamples

Berk (2008) also warns that splitting the data introduces **a new source of uncertainty**.

Using resampling could address this issue, but it is computationally expensive...

Splitting the data into subsamples: example (with R)

Let us load the dataset in R and show its dimensions:

```r
data(iris)
dim(iris)
```

```
[1] 150   5
```

The first rows:

```r
knitr::kable(head(iris))
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

Splitting the data into subsamples: example (with R)

For a 80/20 train/test:

```
set.seed(123)
n_train <- round(.8*nrow(iris))
ind_train <- sample(1:nrow(iris), size = n_train, replace = FALSE)
train <- iris[ind_train, ]
test <- iris[-ind_train, ]
```

Number of observations in the train and in the test sets:

```
nrow(train) ; nrow(test)
```

```
[1] 120
[1] 30
```

Splitting the data into subsamples: example (with R)

The index of the observations that are kept in the train set:

```
ind_train
```

```
  [1]  14  50 118  43 150 148  90  91 143  92 137  99  72  26   7  78  81 147
 [19] 103 117  76  32 106 109 136   9  41  74  23  27  60  53 126 119 121  96
 [37]  38  89  34  93  69 138 130  63  13  82  97 142  25 114  21  79 124  47
 [55] 144 120  16   6 127  86 132  39  31 134 149 112   4 128 110 102  52  22
 [73] 129  87  35  40  30  12  88 123  64 146  67 122  37   8  51  10 115  42
 [91]  44  85 107 139  73  20  46  17  54 108  75  80  71  15  24  68 133 145
[109]  29 104  45 140 101 135  95 116   5 111  94  49
```

Splitting the data into subsamples: example (with R)

It is then possible to use the same procedure to split the `train` dataset into two subsamples: training data and validation data.

```
n_train_2 <- round(.8*nrow(train))
ind_train_2 <- sample(1:nrow(train), size=n_train_2, replace=FALSE)
training <- train[ind_train_2, ]
validation <- train[-ind_train_2, ]
nrow(train) ; nrow(test)
```

```
[1] 120
[1] 30
```

Splitting the data into subsamples: example (with python)

Let us go through an example using the famous iris dataset.

```
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
```

Loading iris dataset:

```
iris=datasets.load_iris()
df_iris = pd.DataFrame(iris.data,
  columns=iris.feature_names)
print(df_iris.shape)
```

```
(150, 4)
```

Splitting the data into subsamples: example (with python)

Let us add the response variable to the pandas data frame:

```
df_iris['target'] = iris.target
print(df_iris.head())
```

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	0
4.7	3.2	1.3	0.2	0
4.6	3.1	1.5	0.2	0
5.0	3.6	1.4	0.2	0
5.4	3.9	1.7	0.4	0

Splitting the data into subsamples: example (with python)

And now we can use the `train_test_split()` method to create the train and test

```
train, test = train_test_split(df_iris, test_size=0.20,
                               random_state = 5)
print("Train shape:", train.shape)
```

```
Train shape: (120, 5)
```

```
print("Test shape:", test.shape)
```

```
Test shape: (30, 5)
```

- Here, we have 80% of observations in the training sample and 20% in the testing sample.
- We have used a specific seed (`random_state=5`) so that the same "random" splitting can be obtained in a subsequent evaluation.

Splitting the data into subsamples: example (with python)

It is then possible to use the same procedure to split the `train` dataset into two subsamples: training data and validation data.

```
training, validation = train_test_split(train, test_size=0.20,
                                         random_state = 5)
print("Training shape:", training.shape)
```

```
Training shape: (96, 5)
```

```
print("Validation shape:", validation.shape)
```

```
Validation shape: (24, 5)
```

With only a few observations

When there is only a few observations and using the validation set approach leads to too small training samples, we can use alternative approaches.

These approaches try to **approximate the out-of-sample ideal**.

One of those is known as **cross-validation** (CV). We will explain the basics for different CV methods, including:

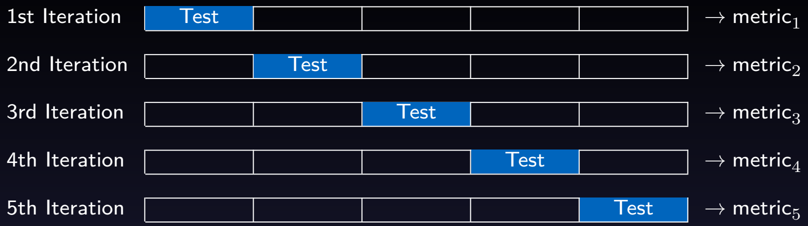
- k-fold cross-validation
- repeated cross validation
- leave one out cross validation.

These techniques may reduce problems linked to the composition of samples, which may affect the quality of the estimation.

k-fold cross-validation

K-fold Cross Validation

- Consider a dataset with n training observations. This set of observations can be divided into k subsets of roughly the same size. Each subset is called a *fold*.
- In each k-fold, the fitting procedure is performed on the $k-1$ folds and evaluated on the kth fold.
- The error metric is computed at each iteration
- Once each of the k-fold has served as an evaluation set, we can compute the average of the error metrics (the cross-validation error).



k-fold cross-validation

Choice of K

- The choice of the number of folds is not straightforward.
  - Relatively small values of k lead to larger training samples, which may result in more bias in the estimation of the true surface.
  - Relatively high values of k lead to less bias in the estimation of the true surface, but they also lead to a higher variance of the estimated test error.
- In the end, it depends on the size and structure of the dataset.
- In practice, we often pick $k = 3$, $k = 5$ or $k = 10$.

k-fold Cross-validation: example (with R)

Let us define a simple function that splits a vector into k folds:

```
#' Splits a vector into folds
#' @param x vector of observations to be splitted
#' @param k number of desired folds
split_into_folds <- function(x,k)
  split(x, cut(seq_along(x), k, labels = FALSE))
```

k-fold Cross-validation: example (with R)

Let us shuffle the row numbers of the observations from the iris dataset:

```r
ind_shuffle <- sample(seq(1, nrow(iris)), replace = FALSE)
```

Then we can use our user-defined function:

```r
ind_folds <- split_into_folds(ind_shuffle, k=3)
ind_folds
```

```
$`1`
 [1]  29  26  27   7  41 134  93  57  66   4  74 133 117  25 136  55  85  45 105
[20]  53 104 131  63  71  84  82 144  17  97   2  49 121  13  24 120  67 125 119
[39]  37 128  76 118  42  61  70  38  64  80  99  36

$`2`
 [1]  91 100  35  87 115  95  48 142  23  96   9 107  39  31  73 101  22  44 146
[20]  43  88 112  33  12  21  98  30  50  47 102 108  40 106 149 109  20 132  65
[39] 150 111 140 114  94   8 103  10 123  18  69  51
```

k-fold Cross-validation: example (with python)

In Python, we can use the `KFold()` function from the `sklearn` library:

```
import numpy as np
from sklearn.model_selection import KFold

X = ["a", "b", "c", "d", "e"]
kf = KFold(n_splits=5)
for train, test in kf.split(X):
    print("%s %s" % (train, test))
```

```
[1 2 3 4] [0]
[0 2 3 4] [1]
[0 1 3 4] [2]
[0 1 2 4] [3]
[0 1 2 3] [4]
```

Repeated k-fold Cross-validation

Another method used for resampling is known as **repeated k-fold cross validation**.

It does the same as the k-fold cross validation, but it repeats the procedure of randomly splitting the data into k folds and fitting the learning process iteratively.

With this technique, the folds are split in different ways at each repetition.

While repeated k-fold CV requires more time than k-fold CV, they may result in less biased estimates.

Repeated k-fold Cross-validation: example (with R)

In R, to get different samples, we just need reshuffle the index and to evaluate our user-defined function to get different samples...

```
$`1`
 [1] 139 108   8 114   5  29  50  70  74  26  73  11 119   6  96 138  88  67 110
[20]  36  55 123 120  94  48 116  10  25  38 115 100 105 129   4  79  52  22 113
[39]  24  39  47  60  63  37  46  54 132 125  16  19

$`2`
 [1]  72  31 107 126 109  62  80   9 150  43  30  61 142 130  89  34  57 122 149
[20] 106  78 143  18   2 128  75  21  84  71  59  98  87 137  40  68  28  32  49
[39]  35  65 104  20   1 146  92 136  76  77  97  51

$`3`
 [1] 124  82   7  83  44 131  23  85 112 117 145  81  56  33   3 103  90  53  64
[20] 148 141 135  14 118  13 121  99  45 147 102  58  12 127  86  42 134 101  27
[39] 140 133 144  93  69 111  95  41  15  91  17  66
```

Repeated k-fold Cross-validation: example (with python)

In Python, we can use the `RepeatedKFold()` function from the `sklearn` library, by specifying the parameter `n_repeats`:

```
import numpy as np
from sklearn.model_selection import RepeatedKFold
X = ["a", "b", "c", "d", "e"]
random_state = 123
rkf = RepeatedKFold(n_splits=3, n_repeats=1,
                    random_state=random_state)
for train, test in rkf.split(X):
    print("%s %s" % (train, test))
```

```
[0 2 4] [1 3]
[1 2 3] [0 4]
[0 1 3 4] [2]
```

Repeated k-fold Cross-validation: example (with python)

Now, changing the parameter n_repeats:

```
random_state = 123
rkf = RepeatedKFold(n_splits=3, n_repeats=2,
                    random_state=random_state)
for train, test in rkf.split(X):
    print("%s %s" % (train, test))
```

```
[0 2 4] [1 3]
[1 2 3] [0 4]
[0 1 3 4] [2]
[1 2 3] [0 4]
[0 1 4] [2 3]
[0 2 3 4] [1]
```

Leave one out cross-validation

Leave one out cross validation

- Leave one out cross validation is a k-fold cross validation where $k = n$, *i.e.*, the number of folds equals the number of training examples.
- The idea is to leave one observation out and then perform the fitting procedure on all remaining data. - Then, iterate on each data point.

- Each fitting procedure yields an estimation. It is then possible to average the results to get the error metric.
- While this procedure reduces the bias, as it uses all data points, it may be time consuming.
- In addition, the estimations may be influenced by outliers.

Leave one out cross-validation: example

In Python, we can use the `LeaveOneOut()` function from the library `sklearn`:

```
from sklearn.model_selection import LeaveOneOut
X = [1, 2, 3, 4, 5]
loo = LeaveOneOut()
for train, test in loo.split(X):
    print("%s %s" % (train, test))
```

```
[1 2 3 4] [0]
[0 2 3 4] [1]
[0 1 3 4] [2]
[0 1 2 4] [3]
[0 1 2 3] [4]
```

Cross-validation and split samples

In all these CV methods, there is no validation dataset (unlike unlike split sample methods), so that the **results are conditional on the training data alone**.

For CV to provide generalizable results from training data, the number of observations should be large enough so that it reflects the joint probability distribution from which data were generated.

2.3.2 Bias-variance trade-off

Bias-variance trade-off

When we estimate the response surface, we aim at getting an estimate **as close as possible to the true response surface**.

The **prediction error**, *i.e.*, the distance between the true value and the predicted one (in a regression context) can be broken down into two pieces:

- the **reducible error**, which corresponds to the sum of two elements:
  - the variance of the estimate
  - the squared bias of the estimate
- the **irreducible error**

Bias-variance trade-off

As in James et al. (2013), let us assume the following relationship between a response variable y and some predictors $\mathbf{x}$:

$$y = f(\mathbf{x}) + \varepsilon,$$

where ε is a zero mean noise with variance σ^2.

We estimate the function f using a statistical learning procedure on a training set and we are interested in the prediction error, at a given value $\mathbf{x}_0$ from a test set.

The expected Mean Squared Error, at that value, can be written as:

$$\mathbb{E}\left[\left(y_0 - \hat{f}(\mathbf{x}_0)\right)^2\right] = \underbrace{\mathbb{V}ar\left(\hat{f}(\mathbf{x}_0)\right) + \left[\text{Bias}\left(\hat{f}(\mathbf{x}_0)\right)\right]^2}_{\text{reducible error}} + \underbrace{\mathbb{V}ar(\varepsilon)}_{\text{irreducible error}},$$

where

$$\text{Bias}\left(\hat{f}(\mathbf{x_0})\right) = E\left[\hat{f}(\mathbf{x_0})\right] - f(\mathbf{x_0})$$

Bias-variance trade-off

To minimize the expected test error, the method that needs to be chosen must therefore **simultaneously** achieve **low variance** and **low bias**.

- The **variance** represents the amount by which $\hat{f}$ would change if we estimated it on a different training data set:
  - in general, more flexible statistical (and more complex) methods lead to higher variance
- The **bias** represents the amount by which the predicted values differ from the true values:
  - $\text{Bias}\left(\hat{f}(\mathbf{x})\right) = \mathbb{E}\left(\hat{f}(\mathbf{x})\right) - f(\mathbf{x})$
  - in general, more flexible statistical methods lead to lower bias.

Bias-variance trade-off: example

Before giving an example of the bias-variance trade-off, let us have a look at the quality of fit of different statistical learning methods that aim at estimating some function f.

Let us consider a function f from which we generate observations:

$$y = f(\mathbf{x}) + \varepsilon,$$

where ε is a zero mean error term with variance σ^2.

We aim at estimating f using statistical learning procedures with increasing levels of flexibility:

- we will consider a linear regression and some regression splines for wich we will vary the degree of freedom.

Bias-variance trade-off: example

We estimate f on a training set and predict values:

- on the observed data of the training set
- on unobserved data from a test set.

Then we can look at the quality of fit through the lens of the Mean Squared Error.

Bias-variance trade-off: example

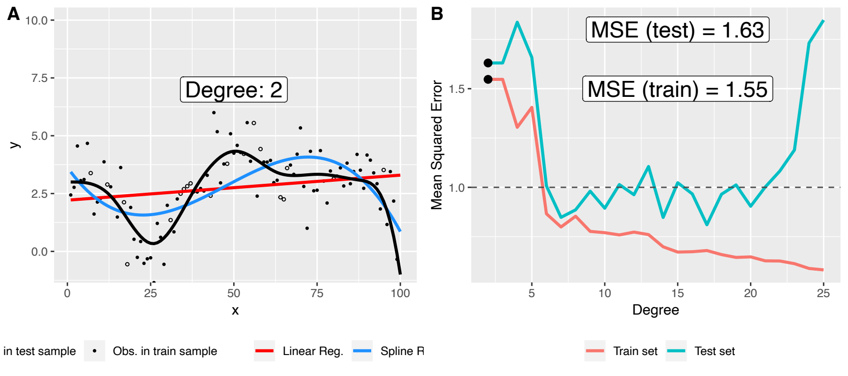


Figure 32: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

On the training sample, the higher the degree of freedom:

- the higher the flexibility
- the better the match compared with observed data.

Bias-variance trade-off: example

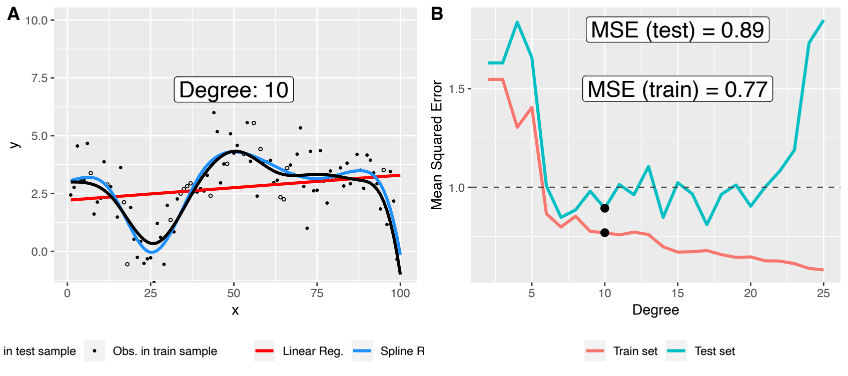


Figure 33: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

On the training sample, the higher the degree of freedom:

- the higher the flexibility
- the better the match compared with observed data.

Bias-variance trade-off: example

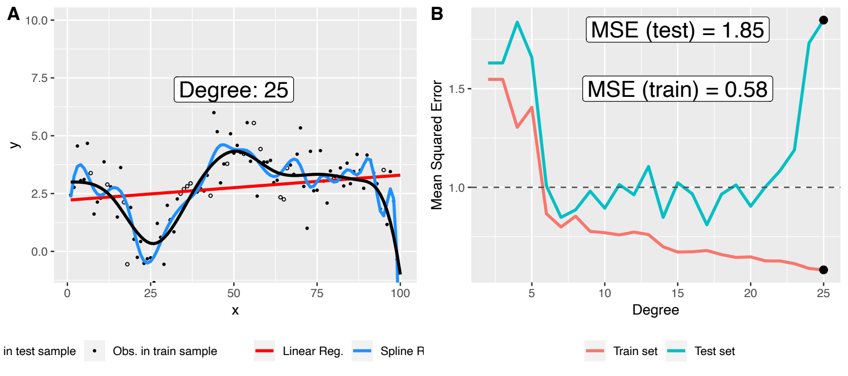


Figure 34: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

On the training sample, the higher the degree of freedom:

- the higher the flexibility
- the better the match compared with observed data.

Bias-variance trade-off: example

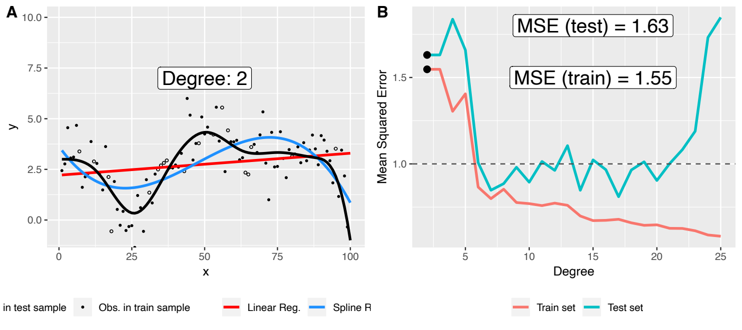


Figure 35: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

- On the test sample, the MSE first declines with flexibility and then increases with it.
- The grey dashed line on panel (B) corresponds to $\mathbb{V}ar(\varepsilon)$

Bias-variance trade-off: example

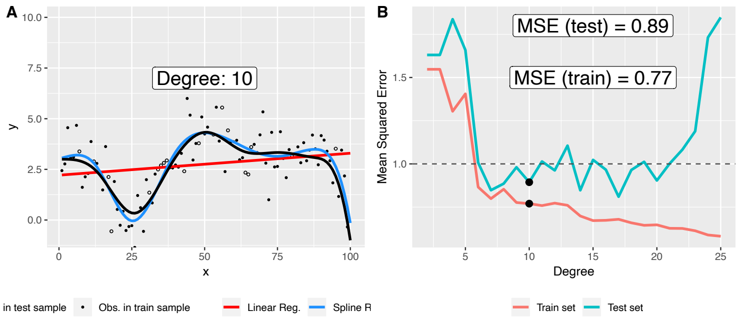


Figure 36: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

- On the test sample, the MSE first declines with flexibility and then increases with it.
- The grey dashed line on panel (B) corresponds to $\mathbb{V}ar(\varepsilon)$

Bias-variance trade-off: example

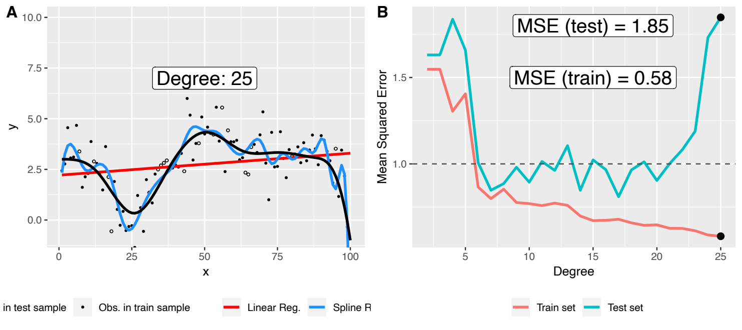


Figure 37: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

- On the test sample, the MSE first declines with flexibility and then increases with it.
- The grey dashed line on panel (B) corresponds to $\mathbb{V}ar(\varepsilon)$

Bias-variance trade-off: example

Let us consider again the function f from which we generate observations $(f(\mathbf{x}) + \varepsilon)$, with $\mathbb{E}(\varepsilon) = 0$ and $\mathbb{V}ar(\varepsilon) = \sigma^2$.

We estimate f on a training sample by means of smoothing splines for which we vary the degree of freedom.

We are interested in the prediction error of $y_0 = f(\mathbf{x_0}) + \epsilon$ at a point $\mathbf{x_0}$ from the test sample.

Bias-variance trade-off: example

We use simulations to estimate the bias, the variance and the MSE for the estimates for f at $\mathbf{x}_0$:

- we randomly create training/test dataset
- on each sample, we estimate f with different learning techniques
- then predict the value at $\mathbf{x}_0$

It is then possible to get estimates of the bias, the variance and the MSE.

Bias-variance trade-off: example

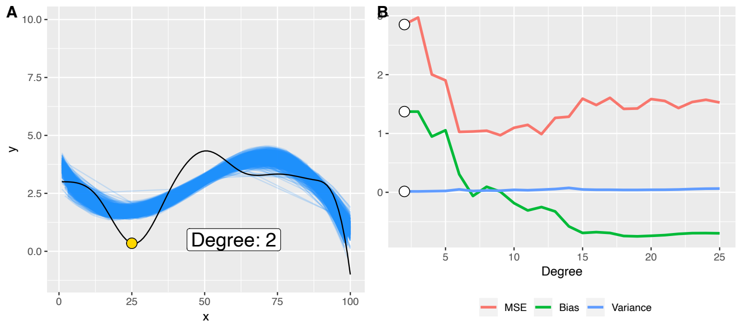


Figure 38: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

Bias-variance trade-off: example

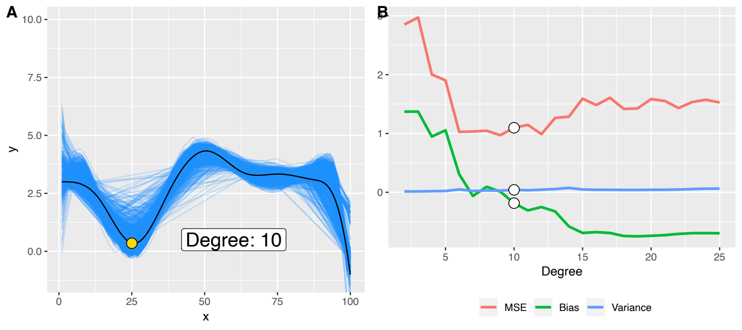


Figure 39: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

Bias-variance trade-off: example

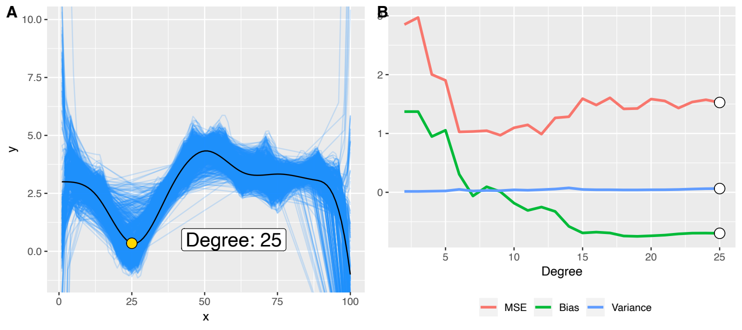


Figure 40: Quality of fit as measured by MSE, using splines depending on the degrees of freedom.

Bias-variance trade-off: example

The **tradeoff** arises as:

- it is easy to obtain a method with low bias but high variance
  - using a method with high flexibility
- it is easy to obtain a method with high bias but low variance
  - using a method with low flexibility

We need to find a method with both variance and squared bias low.

2.4 The curse of dimensionality

The curse of dimensionality

The **curse of dimensionality** describes phenomenon that arise when working in high dimensional feature space.

To explain what is the **curse of dimensionality**, let us borrow some materials from The Curse of Dimensionality in classification by Vincent Spruyt.

Let us assume we want to build a classifier trained to distinguish dogs from cats.

We might be tempted to say that the performance of the classifier will increase as we increase the number of covariables.

This is usually true, but only up to a certain number of additional covariable.

The curse of dimensionality

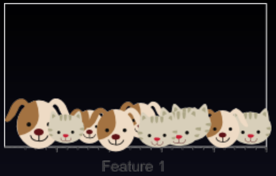


Figure 41: 1 dimension

Figure 42: 2 dimensions

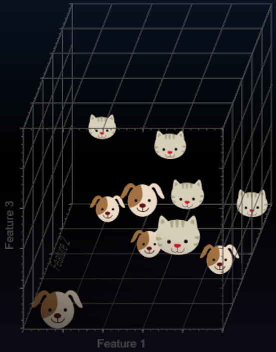


Figure 43: 3 dimensions

Source: Vincent Spruyt (2014)

The curse of dimensionality

- In this example, adding covariates helped find a plane that perfectly separates cats from dogs.
- But in the meantime, as long as we introduced these covariates (and kept our training sample with the same training examples), our data became **more sparse**:
  - the density of our training sample decreases when the dimensionality increases (combinatorics at play here).

Figure 44: A plane that separates dogs from cats.

Source: Vincent Spruyt (2014)

The curse of dimensionality

- If we add more and more dimensions, the problem becomes more and more complex, and it becomes easier to distinguish cats from dogs...
- But the risk of **overfitting** the data grows as well...

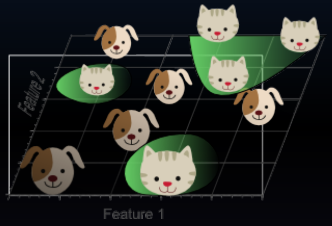


Figure 45: Hihgly dimensional classification results projected on a lower dimensional space.

Source: Vincent Spruyt (2014)

The curse of dimensionality

To sum-up (Berk 2008):

In short, higher dimensional data can be very useful when there are more associations in the data that can be exploited. But at least ideally, a large p comes with a large N. If not, what may look like a blessing can actually be a curse.

3. References

Anderson, Edgar. 1935. "The Irises of the Gaspe Peninsula." *Bulletin of the American Iris Society* 59: 2–5.

Athey, Susan. 2018. "The Impact of Machine Learning on Economics." In *The Economics of Artificial Intelligence: An Agenda*. University of Chicago Press.

Berk, Richard A. 2008. *Statistical Learning from a Regression Perspective*. Vol. 14. Springer.

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning*. Vol. 1. MIT press Cambridge.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. MIT Press.

Nyman, Rickard, Sujit Kapadia, David Tuckett, David Gregory, Paul Ormerod, and Robert Smith. 2018. "News and Narratives in Financial Systems: Exploiting Big Data for Systemic Risk Assessment."

Sung, K-K, and Tomaso Poggio. 1998. "Example-Based Learning for View-Based Human Face Detection." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20 (1): 39–51.

Turing, A. M. 1950. "Computing Machinery and Intelligence." *Mind* LIX (236): 433–60. https://doi.org/10.1093/mind/LIX.236.433.