Machine Learning Introduction

Andrea Nigri

Sapienza, University of Rome

07/05/2019

What is Machine Learning

Just one or...more definitions?

- is a subfield of computer science that is concerned with building algorithms.
- old statistical techniques that are back on stage thanks to computational improvement.

Let's check the class opinion! ▸ Link

What is deductive learning?

Aristotele (384 aC, 322 aC)

General rule to **Specific case**

- General rule: *All man are mortal*
- *Socrates is a man*
- Specific case: *Socrates is mortal*

What is inductive learning?

Leonardo da Vinci; Galileo Galilei; ...; Isaac Newton.

Specific case to **General rule**

- Specific case: *Socrates was a man*
- *Socrates died*
- General rule: *all men are mortal*

Inductive learning-Machine learning

- The most common form of inductive reasoning is **generalization**, with which we get information about a group of things, people, events, objects, and so on, by examining a portion - or **sample** - of that group.
- It is widely believed that this phenomenon can be mitigated in the era of BIG DATA. In which, thanks to the high and constant flow of information (5 V's of big data), the concept of sample was overcome by obtaining the entire population.
- Starting from here, we can connect to a fairly accepted view of Machine Learning

Machine learning

- It is a **system** that, starting from some examples during the training phase, is able to **learn**. Then it is able to **generalize** and manage new data (in the same application area).
- learning from mistakes: One of the best applications is by Turing who deciphered the Enigma code during the world war

Machine learning

APPLICATION AREAS:

- Identify the risk factors for prostate cancer.
- Predict whether someone will have a heart attack on the basis of demographic, diet and clinical measurements.
- Customer segmentation
- Customers churn rate
- Classify a tissue sample into one of several cancer classes, based on a gene expression profile.

Types of learning algorithms

Supervised learning:

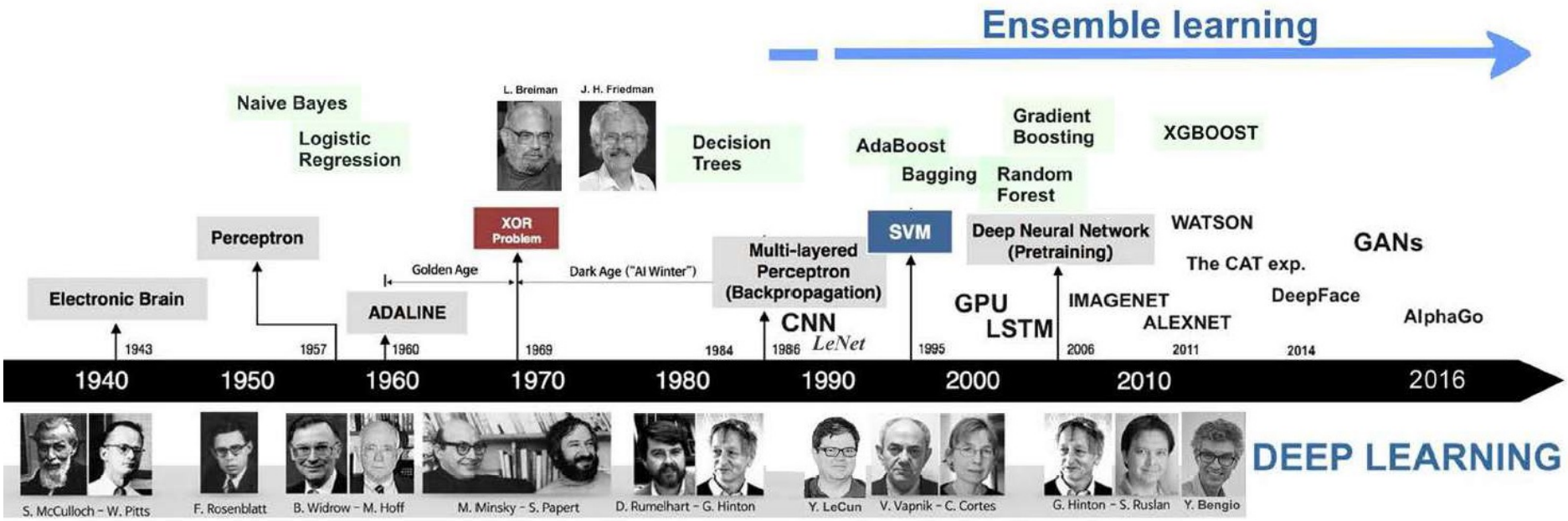
- We provide the model of some examples in the form of input-output relation. The goal is to extract a general rule that associates the input to the correct output.
- We have a target variable Y, which we want to predict in relation to explanatory variables X.
- Examples: Linear Regression, Logistic Regression, Regression tree, Neural Network

Types of learning algorithms

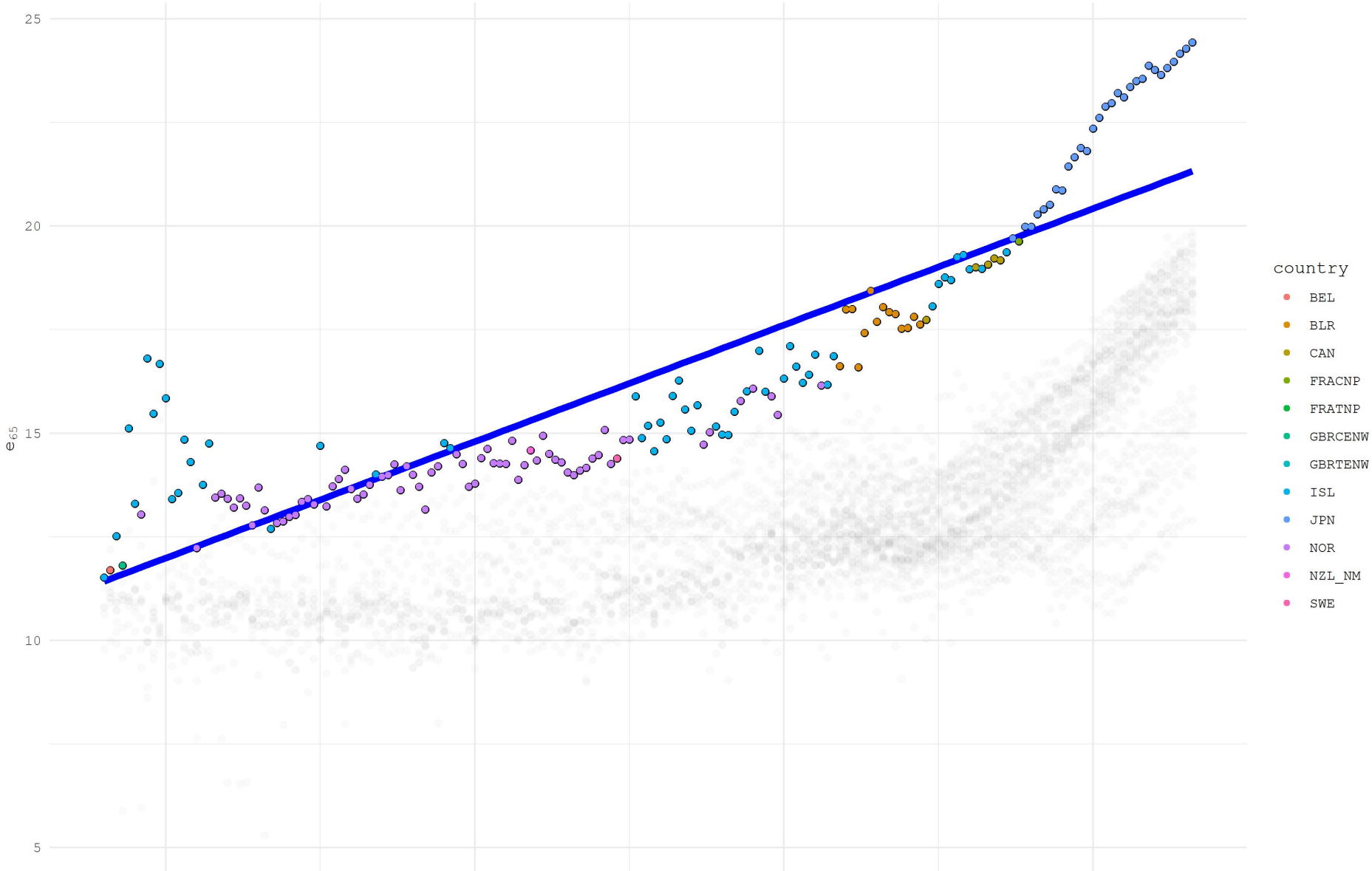
Unsupervised learning:

- We do not have a target variable, the model aims to find a (hidden) structure in the inputs provided, without the inputs being labeled in any way.
- Examples: PCA, Cluster Analysis, Autoencoder Neural Network

Models


Ensemble learning
Naive Bayes
Logistic Regression
L. Breiman
J. H. Friedman
Decision Trees
AdaBoost
Gradient Boosting
XGBOOST
Bagging
Random Forest
Perceptron
XOR Problem
SVM
Deep Neural Network (Pretraining)
WATSON
GANs
Golden Age
Dark Age ("AI Winter")
Multi-layered Perceptron (Backpropagation)
The CAT exp.
Electronic Brain
ADALINE
GPU
LSTM
IMAGENET
ALEXNET
DeepFace
AlphaGo
CNN
LeNet
1943
1957
1960
1969
1984
1986
1995
2006
2011
2014
1940
1950
1960
1970
1980
1990
2000
2010
2016
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton
Y. LeCun
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
Y. Bengio
DEEP LEARNING

ML: Starting from Linear Regression



ML: Starting from Linear Regression

$$\hat{y} = b_0 + b_1 x + \varepsilon \tag{1}$$

- y is the dependent variable, this is what we are trying to predict
- x is the independent variable we use to make prediction
- ε noise

$$min \sum_i (\hat{y}_i - y_i)^2 \tag{2}$$

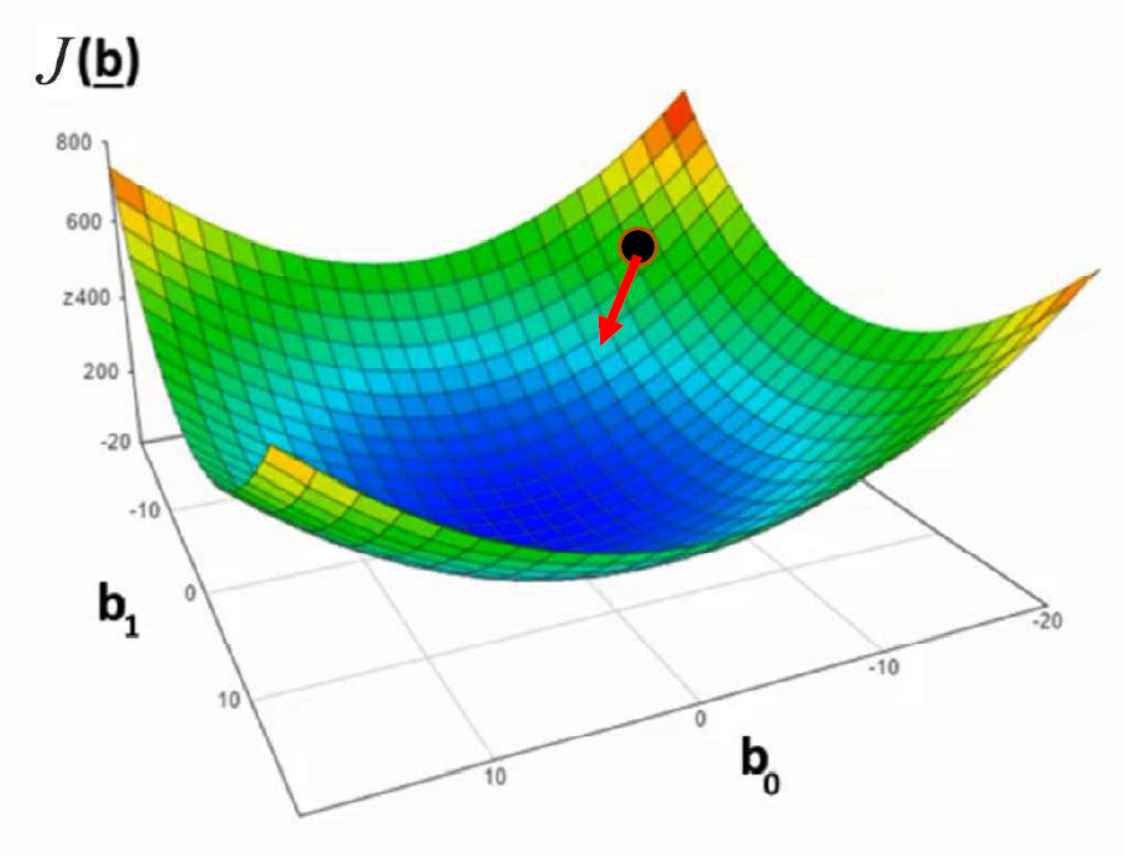
by using algebraic derivation

$$b = (X'X)^{-1}X'y \tag{3}$$

we can consider an alternative way...

ML: Starting from Linear Regression

Gradient Descent Let us consider a convex function, we can describe the principle behind gradient descent as climbing down a hill until a local or global minimum is reached. At each step, we take a step into the opposite direction of the gradient.



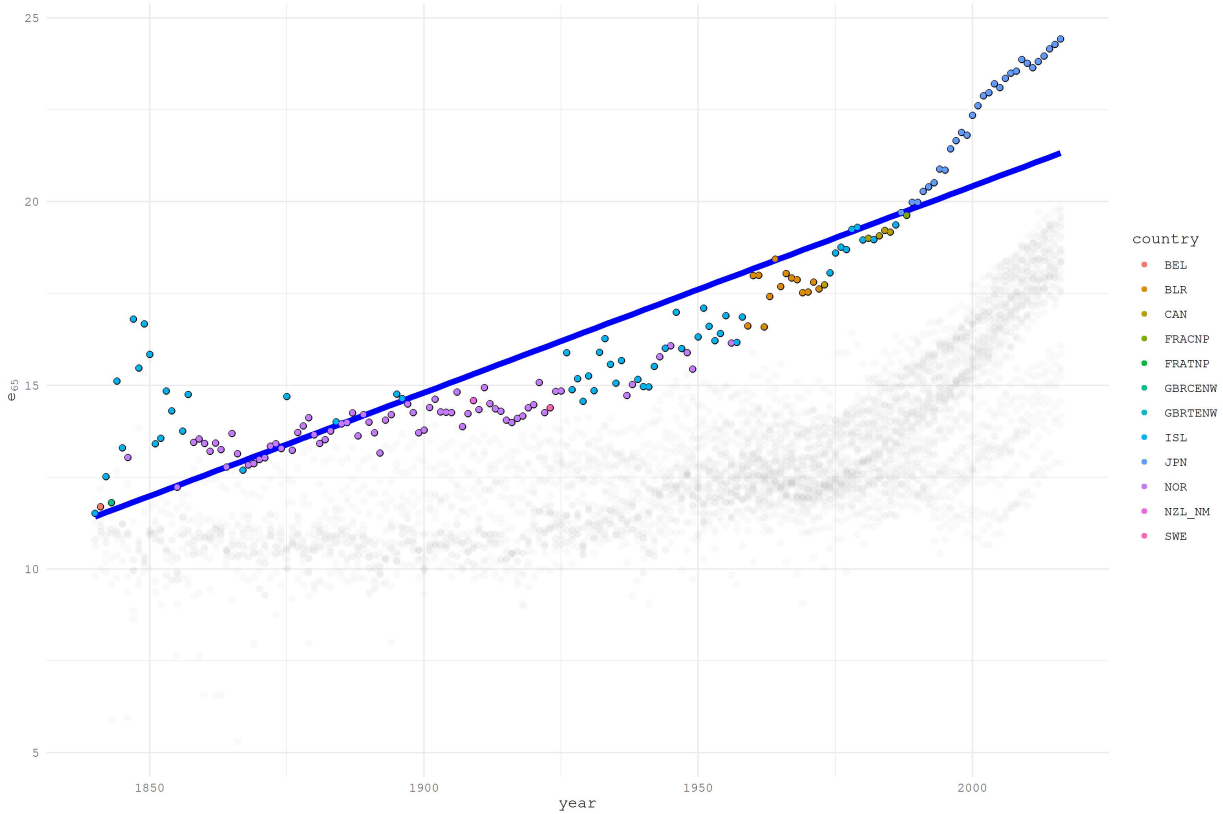
Linear regression drawbacks.

It is based on strong hypotheses such as:

- Indepedecy and homoskedasticity of residual term.
- Residues follow a Normal Distribution
- The explanatory variables are independent ($X^{'}X$ is invertible, to avoid multicollinearity)

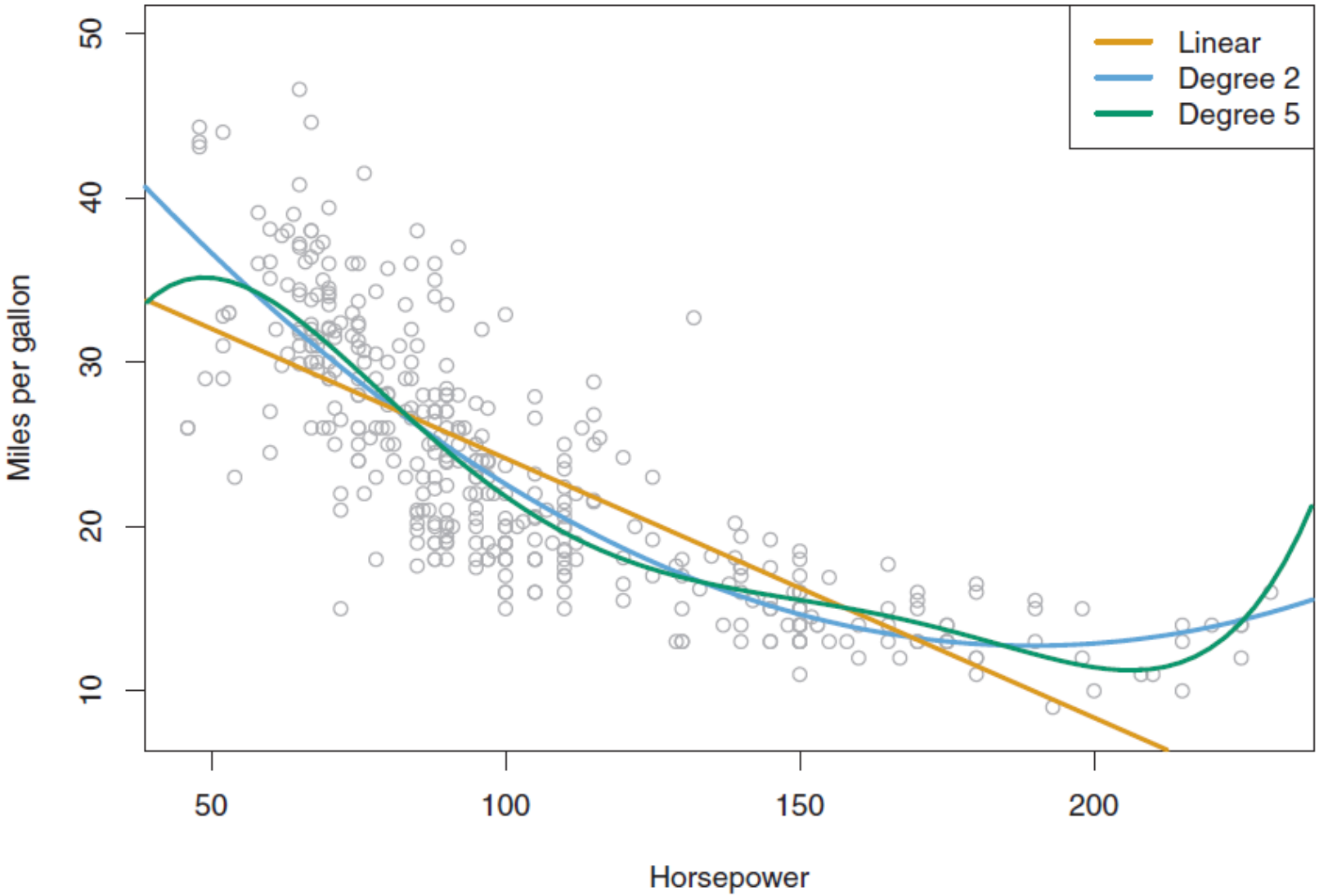
These hypotheses allow us to estimate the optimal parameters of the model and evaluate its predictive ability. In the context of linear regression, no other data is used to evaluate the model

- The evaluation of the model is based on R^2 (percentage of variability explained by the model), used to prevent underfitting.



- To improve the model we can add constructed variables (the variable is always the same but calculated at a different power). In this way, we obtain a linear model (linear in the parameters).

$$\hat{y} = b_0 + b_1 x + b_3 x^2 + b_4 x^3 + b_5 x^4 + \varepsilon \quad (4)$$



Regularizzation in Statistic: Overfitting

- the model adapts perfectly to the data so it is totally unrealistic as it only captures the systematic component ignoring the random one.

In classical statistics

- Ridge Regression: Regularization L2, it is better if the number of variables is greater than the number of units observed

$$\sum_{i=1}^{n} = (y_i = b_0 - b_1 x_{i1} - ... - b_p x_{ip})^2 + \lambda (b_1^2 + ... + b_p^2) = RSS + \lambda \sum_{j=1}^{p} b_j^2 \tag{5}$$

- Lasso Regression: Regularization L1, tends to reduce the coefficient of the less important variables to zero

$$\sum_{i=1}^{n} = (y_i = b_0 - b_1 x_{i1} - ... - b_p x_{ip})^2 + \lambda |b_j| = RSS + \lambda \sum_{j=1}^{p} |b_j| \tag{6}$$

Tackling Overfitting: Machine learning perspective

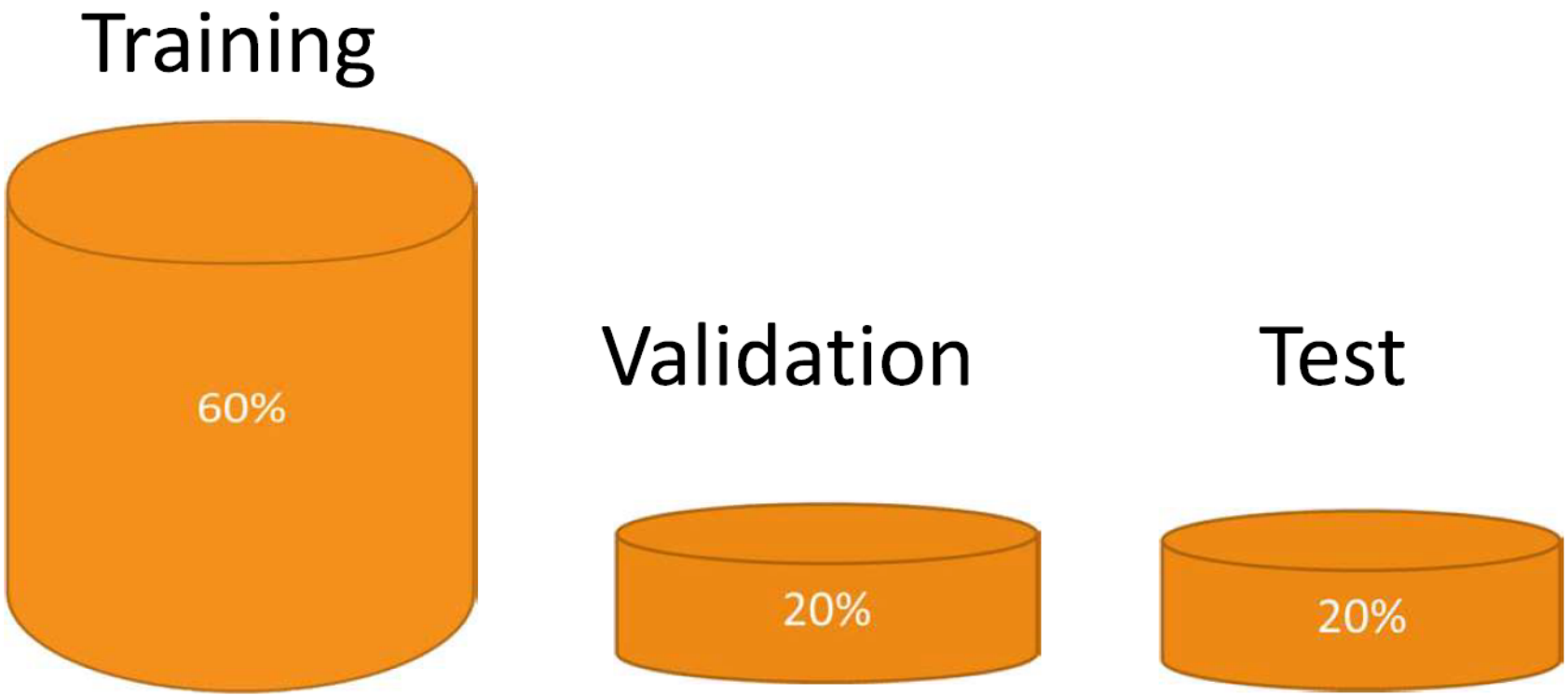
- When you first start out with machine learning you load a dataset and try models.
- You might think to yourself, why can not. I just build a model with all of the data and evaluate it on the same dataset?
- It seems reasonable. More data to train the model is better, right? Evaluating the model and reporting results on the same dataset will tell you how good the model is, right?

.....Wrong!

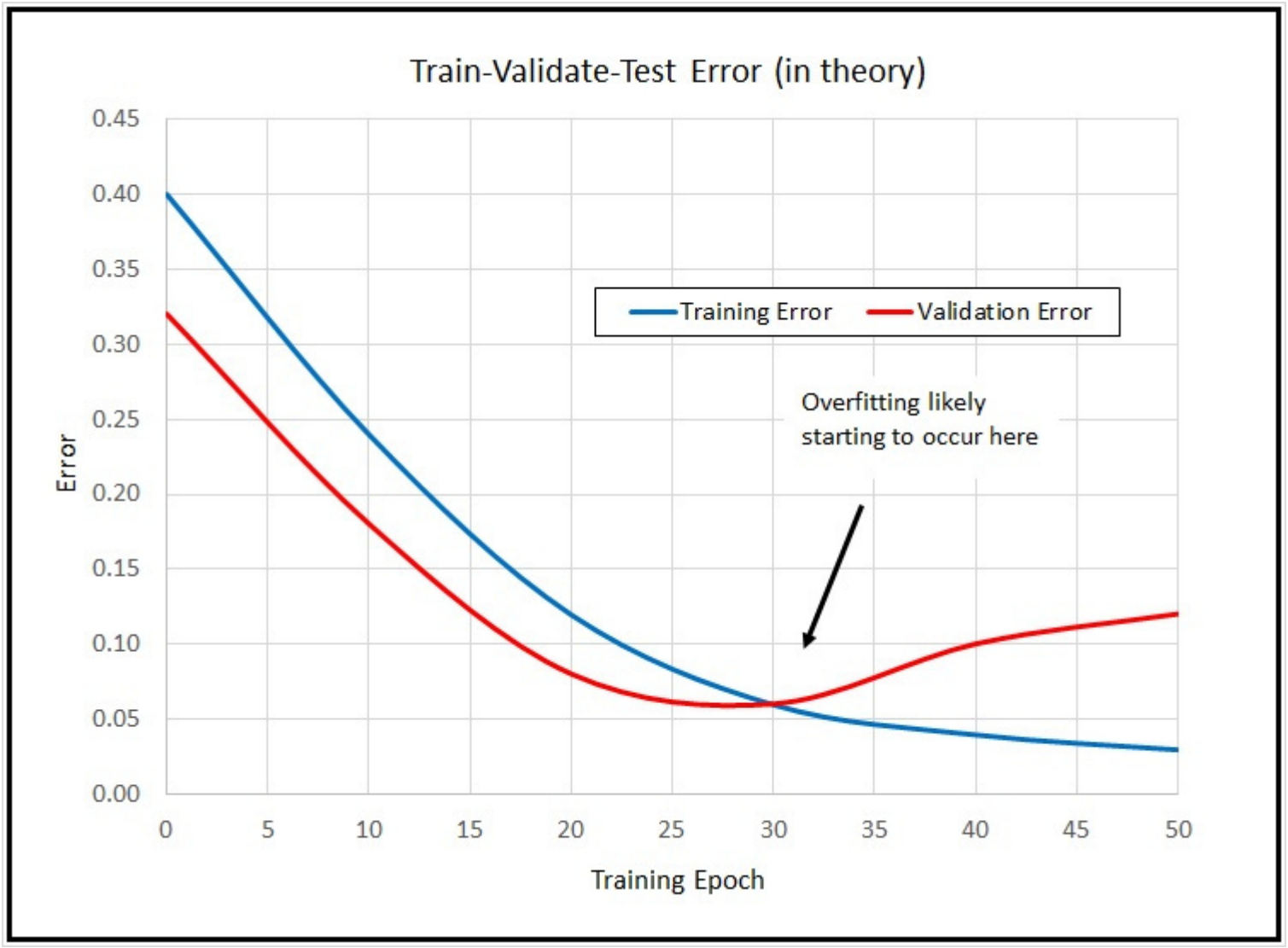
Tackling Overfitting: Machine learning perspective

- You are asking the model to make predictions to data that it has seen before. Data that was used to create the model.
- The drawback with evaluating a predictive model on training data is that it is not able to generalize to new unseen data because it is specialized to the structure in the training dataset.
- You must test your model on unseen data to counter overfitting.
- A split of data 70-30 for training to test datasets is a good start.

- Using cross validation is better, and using multiple runs of cross validation is better again. You want to spend the time and get the best estimate of the models accurate on unseen data. You can increase the accuracy of your model by decreasing its complexity.
- In the case of decision trees for example, you can prune the tree (delete leaves) after training. This will decrease te amount of specialisation in the specific training dataset and increase generalisation on unseen data.



The difference in performance between train and test suggests the presence of overfitting.


Train-Validate-Test Error (in theory)
Error
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30
35
40
45
50
Training Epoch
Training Error
Validation Error
Overfitting likely
starting to occur here

Regression Tree

Regression Tree

Model:Regression Tree

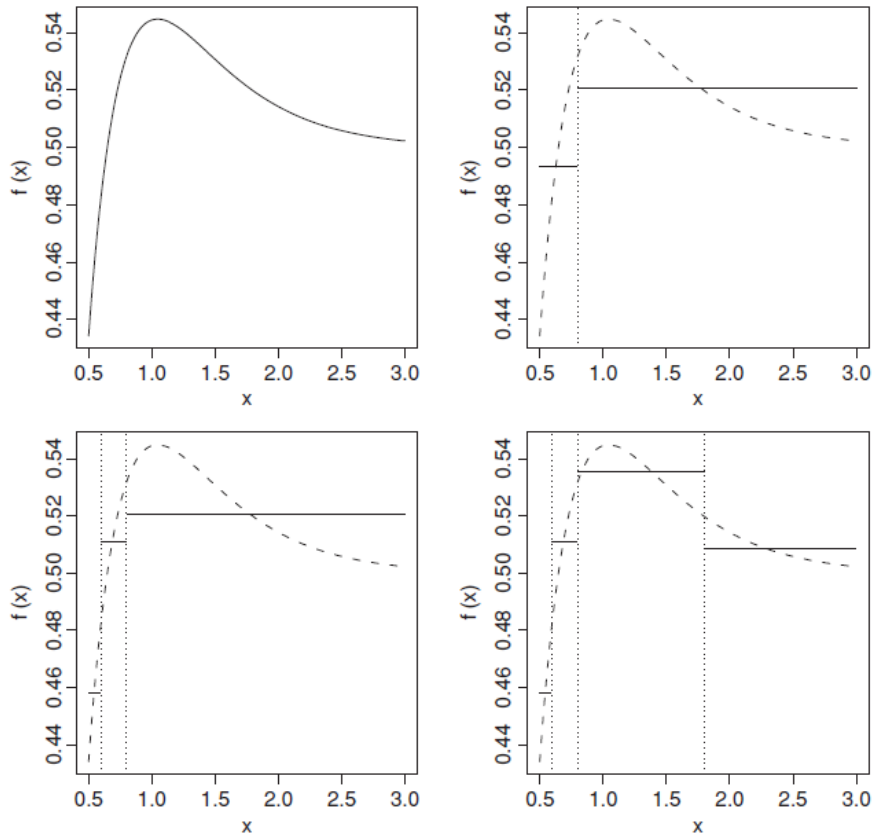
Decision trees can be applied to both regression and classification problems.
We can use Regression Tree for getting the best approximation of a function. In one sense, the simplest way to approximate a generic function:

$$y = f(x) \tag{7}$$

with $x \in \mathbb{R}$ is to use a step function, that is, a piecewise constant function.
But...

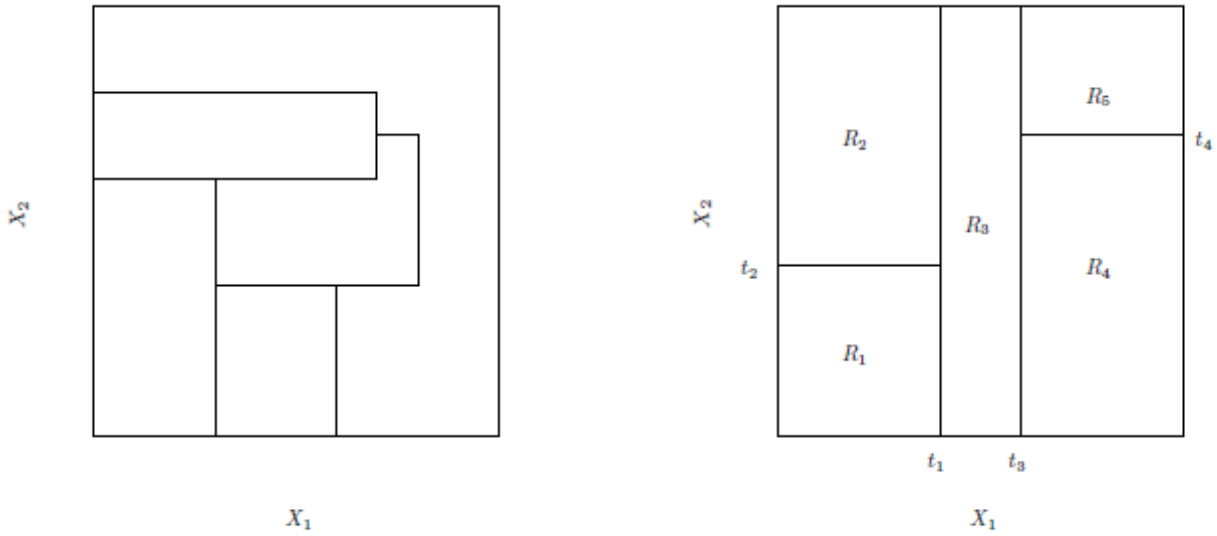
Model:Regression Tree

(a)how many subdivisions of thex-axis must be considered?
(b)where are the subdivision points to be placed?
(c)which value of y must be assigned to each interval?



Model:Regression Tree

- We divide the predictor space that is, the set of possible values for $X_1, X_2, ..., X_p$, into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$.
- For every observation that falls into the region R_j, wemake the same prediction, which is simply the mean of the response values for the training observations in R_j



Model:Regression Tree

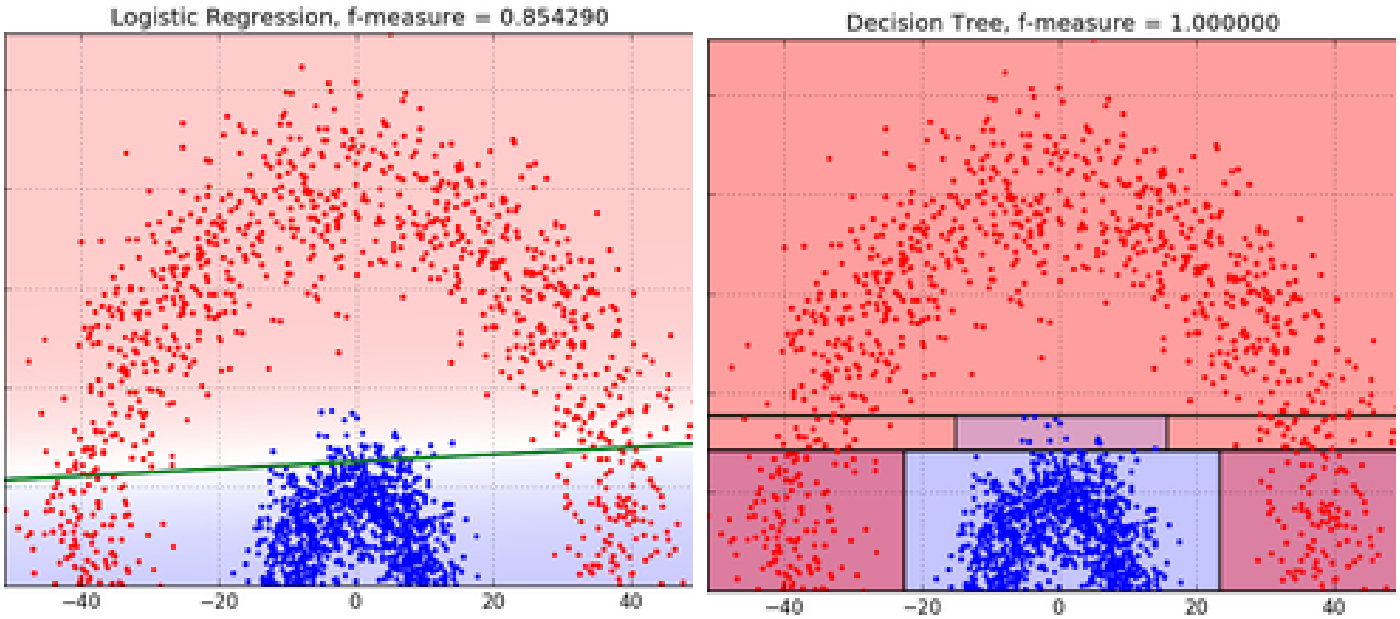
- In theory, the regions could have any shape. However,for simplicity we choose to divide the predictor space into high-dimensional rectangles, or boxes.
- The goal is to find boxes $R_1, ..., R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{8}$$

- Once the regions $R_1, ..., R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

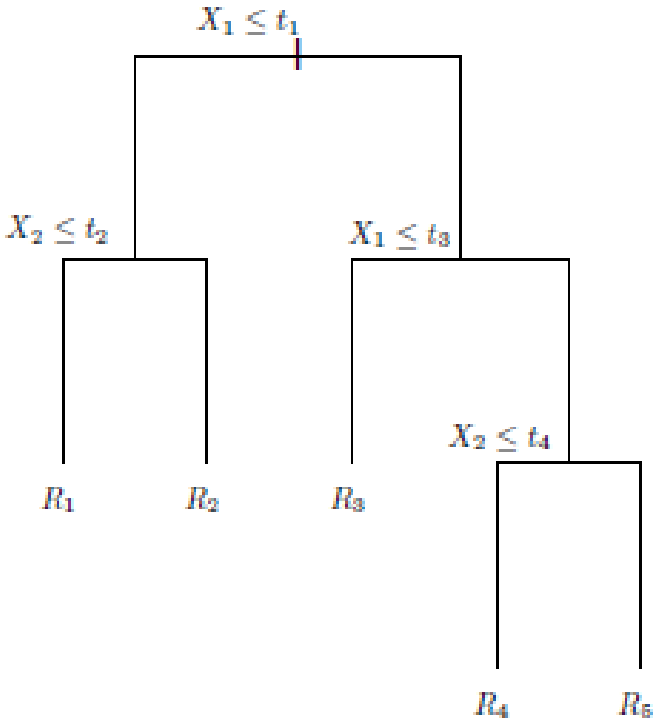
But... It is computationally infeasible to consider every possible partition of the feature

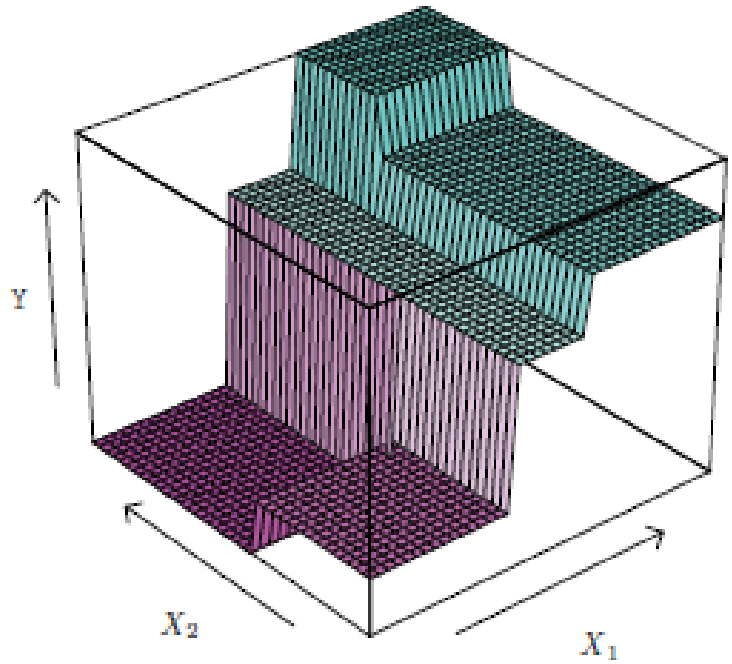
- For this reason, we take a top-down, approach that is known as recursive binary splitting.
- The recursive binary splitting approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region)
- and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.



- The trees start from the root node, which generates the subsequent nodes up to the final ones: leaf nodes.
- split rule:
  - if $X_1 < t$ then the unit belongs to the left branch. (the cutpoint t such that the space predictor splitting leads to the greatest possible reduction in RSS.)
  - otherwise it belongs to the right branch
- To identify the best split, consider a purity measurement (homogeneity) for each node measured using: ENTROPY or INDEX OF GINI
- The maximum purity is when only one class of Y is present in the node

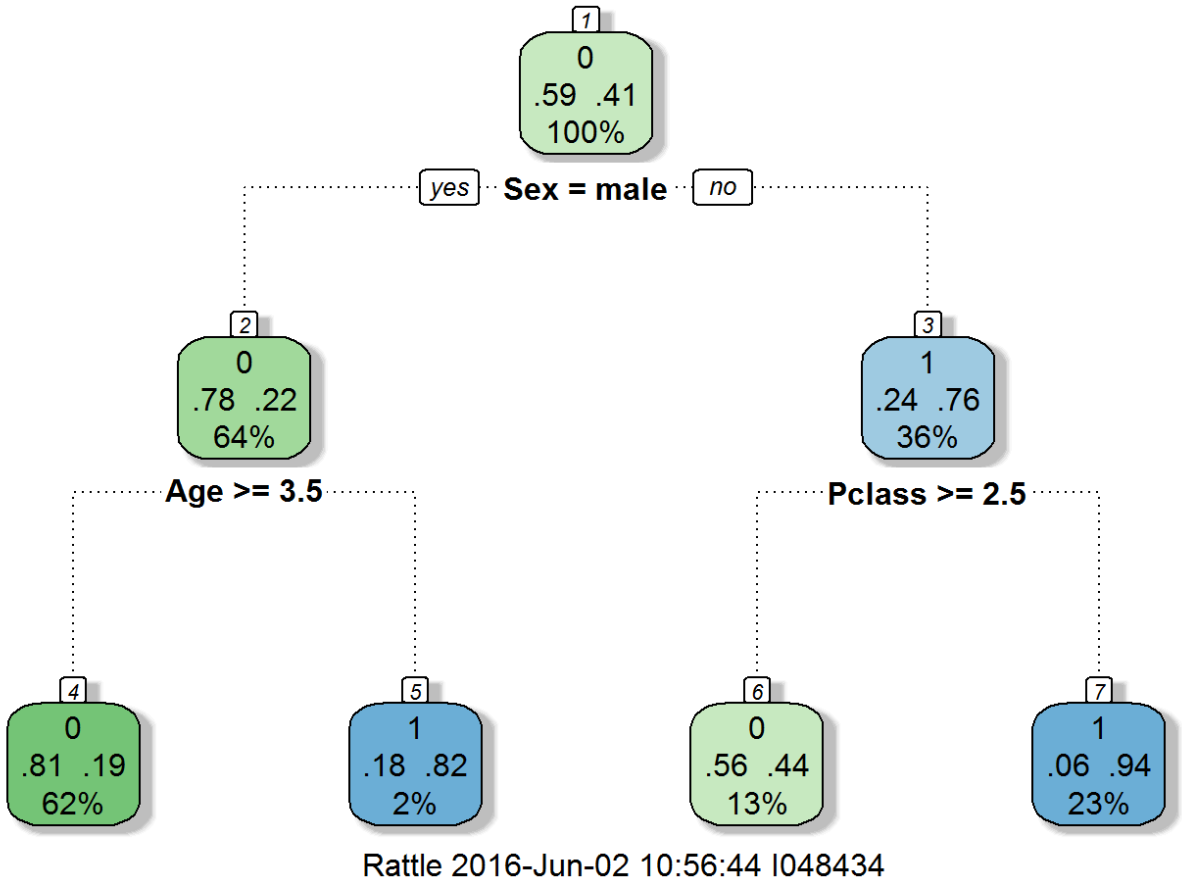
Model:Regression Tree





Model:Regression Tree

Titanic survival

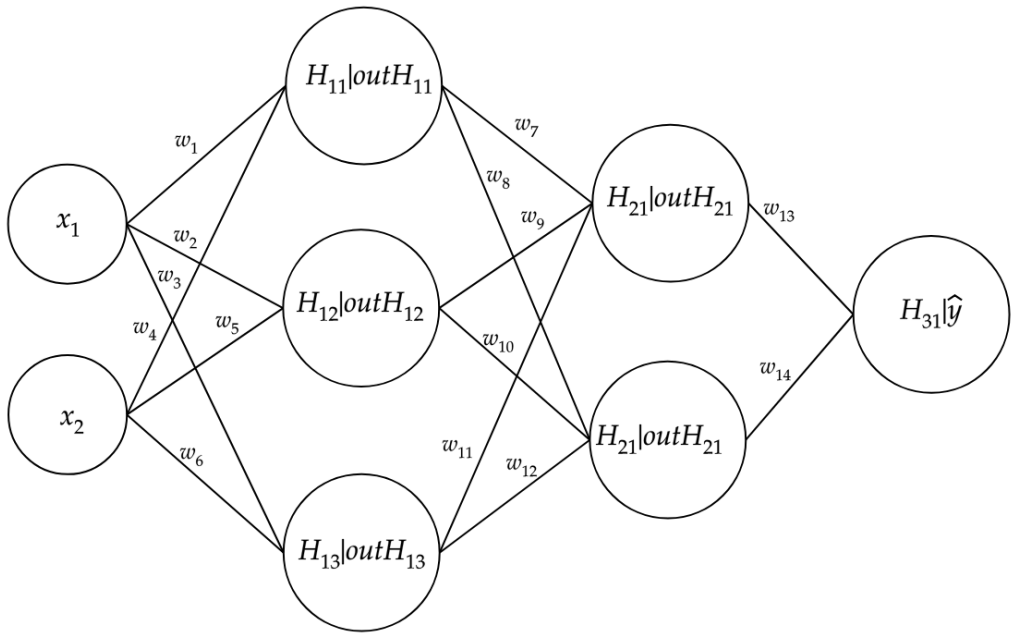


Neural Network

Neural Network

In a nutshell: Neural Network

It's a way to approximate a function. Universal approximation theorem (Hornik et al. 1989):



Model:Neural Network

- The term neural network (NN) originated as a mathematical model that replicates the biological neural networks of the human brain.
- NN architecture includes neurons, synaptic connections that link the neurons, and learning algorithms.
- Typically, NN is formed by three types of layers, respectively, called input, hidden and output layer and each one has several neurons.
- Each unit in a network gets weighted information through synaptic links from the other connected ones and returns an output by using an activation function transforming the weighted sum of input signals.

Model:Neural Network

Training a NN means calibrating the weights by repeating two steps:

- **In forward propagation** we apply a set of weights to the input data and calculate an output.

$$H_n = \sum_{j=0}^{n} x_j w_j = w^T x \tag{9}$$

  - In each node, in every hidden layer, we obtain an output came from the sigmoidal activaction function:

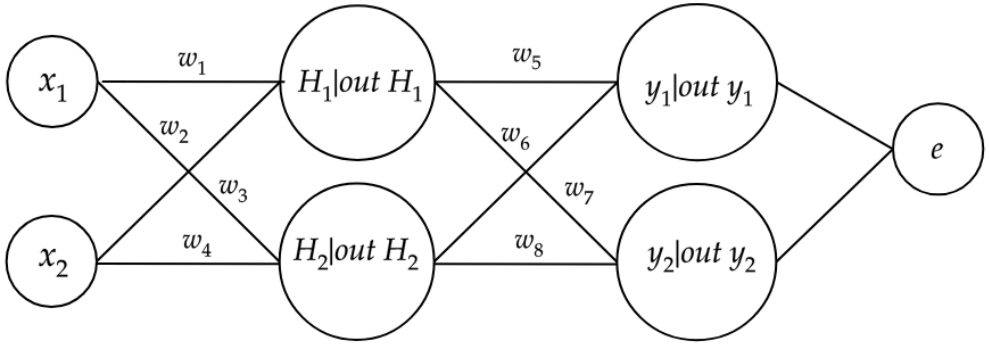
$$outH_n = f(H) = \frac{1}{1 + e^{-H_n}} \tag{10}$$

- **In back propagation** we measure the margin of error of the output and adjust the weights accordingly to decrease the error.

Model:Neural Network

1. FORWARD PROPAGATION: we apply a set of weights to the input data and calculate an output.

1. $H_n = \sum_{j=0}^{n} x_j w_j = w^T x$ 2. $out\ H_n = f(H) = \frac{1}{1+e^{-H_n}}$ 3. $y_m = \sum_{j=0}^{m} out\ H_j w_j$ 4. $out\ y_m = f(y) = \frac{1}{1+e^{-y_m}}$



3. BACK PROPAGATION:
we measure the error of the output and adjust the weights accordingly to decrease the error (gradient descent).

5. $e = (y - \hat{y});\ J = \sum \frac{1}{2}(e)^2$ 6. $\Delta w = -r \nabla J(w)$

4. DERIVATION STEPS:

$for\ w_k;\ k = 5,\ldots,8$

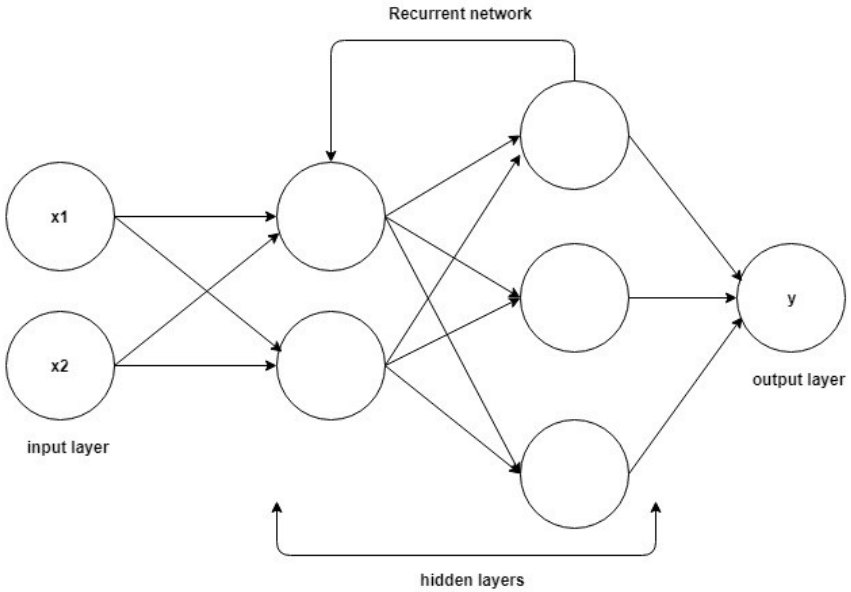
$$\frac{\partial total\ E}{\partial w_k} = \frac{\partial total\ E}{\partial out\ y_k} \frac{\partial out\ y_k}{\partial y_k} \frac{\partial y_k}{\partial w_k}$$

$for\ w_z;\ z = 1,\ldots,4$

$$\frac{\partial total\ E}{\partial w_z} = \frac{\partial total\ E}{\partial out\ H_n} \frac{\partial out\ H_n}{\partial H_n} \frac{\partial H_n}{\partial w_z} \qquad \frac{\partial total\ E}{\partial out\ H_n} = \frac{\partial E_1}{\partial out\ H_n} \frac{\partial E_2}{\partial out\ H_{n+1}}$$

Neural Network Classification

- Feedforward Neural Network
- Recurrent Neural Network



  - LSTM
- Convolutional Neural Network

Appendix:Neural Network

Mathematical details

Appendix:Neural Network

- Considering a single neuron H—called perceptron—its output is defined as:

$$outH = \phi(\mathbf{w}^T \mathbf{x} + b), \tag{11}$$

- where $\mathbf{x} \in \mathbb{R}^d$ is the input, $\mathbf{w} \in \mathbb{R}^d$ is the associate synaptic weight, $d \in \mathbb{N}$ is the number of input signals and ϕ is the activation function.
- The term $b \in \mathbb{R}$ represents the bias, also called activation threshold. It is important to notice that the function ϕ must be differentiable because the learning equations are gradient-based.

Appendix:Neural Network

- The output $\mathbf{outH} \in \mathbb{R}^{n_h}$ of a generic hidden layer with n_h neurons is defined:

$$\mathbf{outH} = \phi(W^T \mathbf{x} + \mathbf{b}), \tag{12}$$

- where $W \in \mathbb{R}^{d \times n_h}$ is a weight matrix and $\mathbf{b} \in \mathbb{R}^{n_h}$ is a biases vector. In the MLP scheme, the output of a hidden layer becomes the input for the next layer.

Appendix:Neural Networl

- Considering a regression problem, where $g \in \mathbb{N}$ is the number hidden layers, the output $\hat{y} \in \mathbb{R}$ is obtained by:

$$\mathbf{outH}_1 = \phi_1(W_1^T \mathbf{x} + \mathbf{b}_1),$$

$$\mathbf{outH}_2 = \phi_2(W_2^T \mathbf{outH}_1 + \mathbf{b}_2),$$

$$\ldots$$

$$\hat{y} = \phi_g(W_g^T \mathbf{outH}_{g-1} + \mathbf{b}_g),$$

- where $W_1, W_2, ..., W_g$ are weight matrices, $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_g$ are bias vectors, and $\phi_1, \phi_2, ..., \phi_g$ are activation functions not necessarily different from each other.

Appendix:Neural Network

- NN training involves an unconstrained optimization problem where the aim is to minimize a function in high dimensional space. Let us define a loss function:

$$E = \frac{\Sigma(y - \widehat{y})^2}{2} \tag{13}$$

  that measures the difference between the predicted values $\hat{y}$ and observed ones y.
- The aim is to find the values of the synaptic weights that minimize the quantity E.

Appendix:Neural Network

- The back-propagation is the most used for the training of feedforward NNs. The algorithm compares the predicted values against the desired ones (objective) and modifies the synaptic weights by back-propagating the gradient of the loss function.
  - in the forward step, the prediction $\hat{y}$ are computed fixing the synaptic weights,
  - in the backward step, the weights are adjusted in order to reduce the error E of the network.
  - The NN iteratively performs forward and backward propagation and modifies the weights to find the combination that minimizes the loss function.

Appendix:Neural Network

- Analytically, the back-propagation algorithm updates the weights W_g in the last layer using the delta rule, defined as:

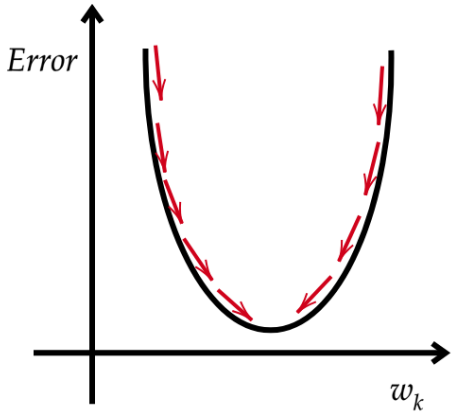
$$\Delta W_g = -r \frac{\partial E}{\partial W_g}, \tag{14}$$

- where r is the learning rate. For the other previous layers, we proceed using the chain derivation rule. The updating of the weights matrix W_{g-1} are computed:

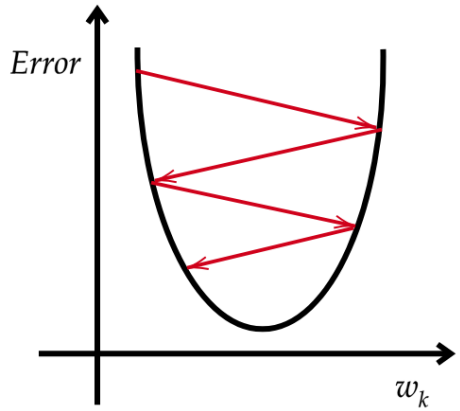
$$\Delta W_{g-1} = -r \frac{\partial E}{\partial \mathbf{outH}_{g-1}} * \frac{\partial \mathbf{outH}_{g-1}}{\partial W_{g-1}} \tag{15}$$

Appendix:Neural Network

The idea behind the gradient descent is similar to climbing down a hill until a minimum is reached. At each update, the search moves in the opposite direction of the gradient and the slope of the gradient and the learning rate determine the amplitude of this movement. In addition, the choice of the rate r is important, since a small value involves too many iterations while a large value could not allow convergence to the global minimum.



Small r:
too many iterations



Large r:
does not converge to global minimum