Introduction to Machine Learning

Billel BADAOUI

No Institute Given

Abstract. Machine Learning algorithms explained to Data Science beginners

Table of Contents

1 Introduction

The following paper modestly aims at introducing the main principles and algorithms of machine learning to beginners in data science. You will find descriptions of the main tools used by data scientists in order to solve different problems, starting from price prediction to unsupervised asset clustering.

I decided to write this article to gather information from different sources to make sure that everyone is able to understand the underlying principles behind what usually looks like black boxes : machine learning algorithms. Indeed, I noticed that when I was reading or watching courses I always had questions about what teachers or authors were assessing because it was too obvious for them. Saying that Random Forest helps to reduce variance. However, why exactly do you want to reduce variance ? How does this process reduce variance ? etc.

Of course the algorithm list provided will not be exhaustive, and if you have any comment or any suggestion for improvement do not hesitate to contact me ! Now let's get into the topic that we are all interested in : Machine Learning.

First of all, as you may already know there are two types of machine learning problems : supervised and unsupervised learning. The first one consists in using training data containing labels or continuous values in order to predict the values of unlabeled data. For instance, imagine that you want to sell your house but you do not know exactly what price you should ask for your property. A method to estimate your house's price may be looking at the different houses on the market, and compare their characteristics to yours in order to give a price. Another example of supervised learning could be cancer detection : you have a database of different body cells. You already labeled all your rows with 'cancerous' or 'not cancerous', you can thus train a model that will be able to predict if a body cell image might be cancerous or not when looking at it.

On the other hand, unsupervised learning is used when you do not have any label beforehand. For example, it can be the case in marketing. How could you send customized advertisements to a large database of customers ? To proceed you can classify them into different clusters based on their characteristics, and then send to these clusters customized adverts. That being said we are now ready to go deeper in supervised learning.

2 Supervised Learning : Regression

2.1 Linear Regression

There are two categories n supervised learning : classification and regression. Classification problems concern discrete labels like digit recognition (between 0 and 9) or titanic survivors prediction (survived or died). When the labels represent continuous values, this is called regression. The most famous technic is the linear regression : you want to approximate your dataset labels using a linear function of your features.
See below a set of points between 0 and 5 with a step 1, and for each of these points a label y between 2 and 40. Let's say we assume y to be linear and these x are observations, what would be an estimation of the value for $x = 2.5$?

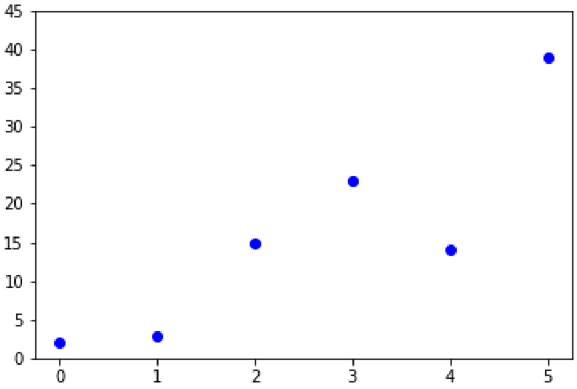

Fig. 1. Dataset for Linear Regression

The linear regression will approximate this set of points with a linear function of x. See below in red the approximation :

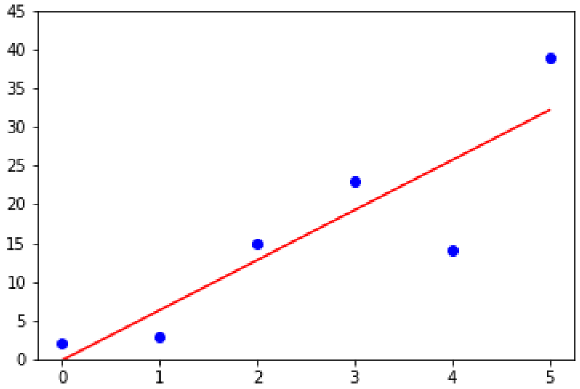

Fig. 2. Linear Regression Model

Mathematically
prerequisite : optimization, gradient descent method, convexity

If you take $x \in \mathbb{R}^n$, a linear function $h : \mathbb{R}^n \to \mathbb{R}$ of x, is of the form $h(x) = \theta_1 x + \theta_0$ with $(\theta_1, \theta_0) \in \mathbb{R}^n \text{x} \mathbb{R}^n$.
In our case study we have a set of points x_i for $i \in [0, 5]$ and their associated labels y_i for $i \in [0, 5]$.
This function h is called the hypothesis function. The goal here is to minimize the distance between our hypothesis function applied to our dataset and their actual labels. The distance used will be the squared error :
$(y_i - h(x_i))^2 = (y_i - \theta_1 x_i + \theta_0)^2$.
We want thus to minimize the average error over the whole dataset, so if you have $n \in \mathbb{N}$ training example you want to minimize the function

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \theta_1 x_i - \theta_0)^2$$

J is called the "cost" function, it represents the average error that our prediction does on our training set. To minimize this function of θ_0 and θ_1 you can use a gradient descent (this process will be detailed in a second article). The advantage of the function J is that it is convex, which means that we can look for the minimum without wondering if we found a local minimum or a global one. Indeed, for a convex function a local minimum is a global minimum.

$$\theta^* = (\theta_0^*, \theta_1^*) = \underset{\theta_0, \theta_1}{\text{argmin}}\ J(\theta_0, \theta_1)$$

Hence the gradient descent on the parameters :

$$\theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

for $j = 1$ and $j = 0$ and α the *learning rate*
However, performing a gradient descent is not the best way of solving this optimization problem because in the case of linear regression there is an analytic resolution of the problem. This is called the least square method and this will be discussed at the end of this section.
Once you determined θ^* you obtain the red line on the Fig. 2. As you can see, the linear regression is able to approximate pretty linear relationships between data and labels, however, this cannot be always possible, have a look at this dataset :

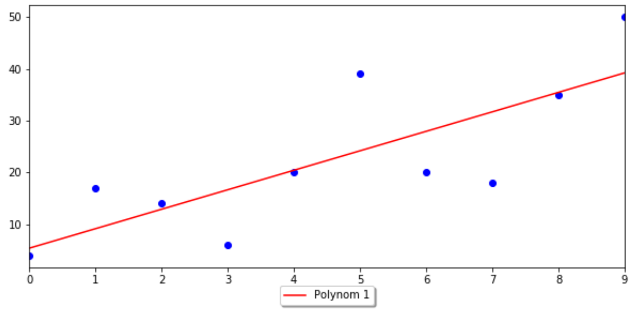

Fig. 3. Linear Regression Model not adapted

As you can see the linear prediction performs pretty badly because the labels are not linear functions of the data. To overcome this issue you can use polynomial regression.

2.2 Polynomial Regression

In order to perform a polynomial regression you define your hypothesis function h as a polynomial function of degree $n \in \mathbb{N}$:

$$h(x) = \theta_n x^n + .. + \theta_1 x + \theta_0$$

Hence the new cost function with a sample of m elements :

$$J(\theta_0, .., \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \theta_n x_i^n - .. - \theta_0)^2$$

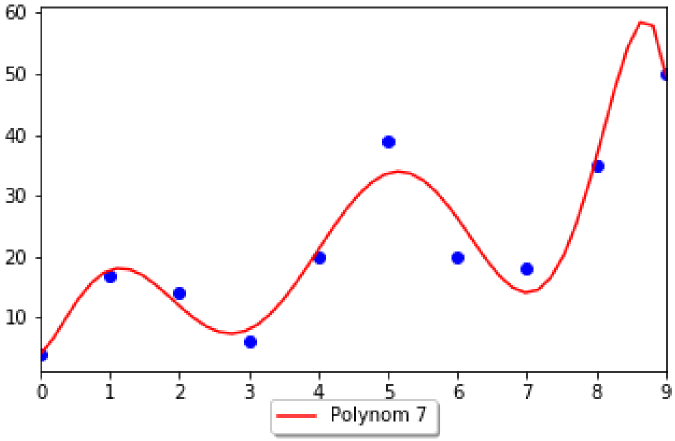

Fig. 4. Polynomial Regression

To minimize this new cost function you can use a gradient descent as well. Now you may have this question in mind : how can I choose the degree n of my polynomial assumption ? And this is a very important question !

Definition : Hyperparameters
Hyperparameters are the different parameters of your model, it can be for instance the degree of your polynomial in a polynomial regression, the number of trees in your random forest, your learning rate in a gradient descent, the number of layers in your neural network etc.

Reminder : Lagrange Polynomial
Given a set of $n + 1$ distinct points $x_0, .., x_n$ and a set of $n + 1$ labels $y_0, .., y_n$

there exists a polynomial function of degree n that assigns to each x_i its label y_i. This polynomial is called a Lagrange Polynomial and is defined as follows :

$$P(x) = \sum_{i=0}^{n} y_i \frac{\prod\limits_{k=0,k\neq i}^{n} x - x_k}{\prod\limits_{k=0,k\neq i}^{n} x_i - x_k}$$

The proof is left to the reader.

Now that we have this in mind it can look attractive to use this polynomial to approximate each discrete problem that you want to generalize. However, this is not always a good idea, as illustrated on the graph below :

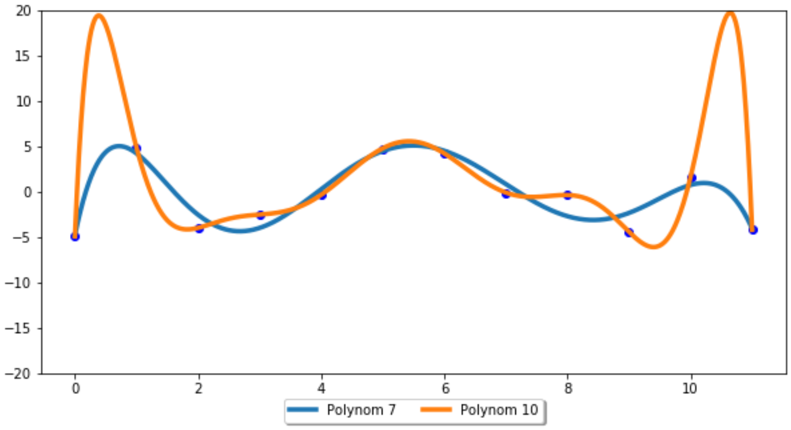

Fig. 5. Overfitting polynomial

The graph was created with a set of 12 distinct points.The orange line represents a polynomial regression of degree 11, whereas the blue line represents a polynomial regression of degree 7. The orange line perfectly fits to the data but presents a high variance, it fluctuates a lot, thus it may be not a good model to predict continuous values of another unlabeled dataset ! In this case you may prefer the model calibrated with a polynomial of degree 7. Data scientists would say the degree 11 model is *over fitting* the data, which means it learns quite well on your training set but would probably performs poorly on a new dataset. To get more details about overfitting please refer to the specific section.

2.3 Lasso, Ridge, and ElasticNet

So far, we discussed about Linear and Polynomial regressions in a one dimensional space $\mathbb{R}$. However, one dimensional problems will probably represent a very small proportion of the issues you will have to face as a data scientist. In general you will have a set of n different vector points $X_1, ..X_n$ of dimension m that describe your training data and are affected n points in $Y_1, .., Y_n$
For $i \in [1, n]$ you have

$$X_i = \begin{pmatrix} x_{i1} \\ . \\ . \\ x_{im} \end{pmatrix}$$

We now have to write the linear regression cost function as follows :

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \theta_1^T X_i - \theta_0)^2$$

Where the T exponent represents the transposed of the vector

$$\theta_1 = \begin{pmatrix} \theta_{11} \\ . \\ . \\ \theta_{1m} \end{pmatrix}$$

$\theta_1^T X_i$ is in fact the inner product of θ_1 and X_i. You can minimize this cost function J with a gradient descent on each coordinate of $\theta = (\theta_0, \theta_1)$. For now, I will always assume that minimizing operations will be operated with gradient descent.

Reminder : Norms
In this chapter we will be interested by two types of norm : L1 and L2 norms.
• L1 norm : For $m \in \mathbb{N}$, the L1 norm of the vector $X \in \mathbb{R}^m$ is

$$||X||_1 = \sum_{i=1}^{m} |x_i|$$

This norm represent the "Manhattan norm", it is simply the sum of the absolute value of all the vector's coordinates.

• L2 norm : For $m \in \mathbb{N}$, the L2 norm of the vector $X \in \mathbb{R}^m$ is

$$||X||_2 = \sqrt{\sum_{i=1}^{m} x_i^2}$$

This is the standard euclidean norm.

LASSO Regression

So far, I only presented our minimization problem as unconstrained, I said that we need to minimize the cost function with respect to the parameter θ without specifying any definition set. This can lead to instability and overfitting in high dimension if you have highly correlated features in your dataset. The LASSO Regression is a linear regression regularized by an additional term proportional to the L1 norm of your parameter vector :

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \theta_1^T X_i - \theta_0)^2 + \lambda ||\theta||_1$$

$\lambda > 0$ is a hyperparameter of your model, a large λ will penalize strongly high values of θ, whereas low values of λ will give a weaker penalization. For $\lambda = 0$ you go back to the classical linear regression. The addition of this penalty term is called L1-Regularization.
The main advantage of this LASSO Regression is that it makes a feature selection inside your data. Indeed, the result of minimizing your cost function with respect to θ will have some weights set to 0, this is called *sparsity.* And the biggest drawback of this regularization is that when you have highly correlated features this yields to instability of your model, hence not very robust predictions.

Remark : LASSO stands for Least Absolute Shrinkage and Selection Operator

Ridge Regression

Like LASSO, Ridge Regression is a regularization method, your new cost function is :

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \theta_1^T X_i - \theta_0)^2 + \lambda ||\theta||_2^2$$

Introduction the L2 norm will penalize more heavily for high values of θ than LASSO. The main advantage of Ridge regularization is that it provides stability to the model.

ElasticNet = Ridge + LASSO

We saw that L1-regularization can proceed to feature selection but can be unstable. What if you combine LASSO and Ridge in one model to provide stability and sparsity ? This is the idea behind ElasticNet, the cost function can be written as follows :

$$J(\theta_0,\theta_1) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \theta_1^T X_i - \theta_0)^2 + \lambda(\alpha||\theta||_1 + (1-\alpha)||\theta||_2^2)$$

α is a hyperparameter between 0 and 1, for α close to 1 you will have a behaviour similar to LASSO and for α close to 0 you will have Ridge behaviour. ElasticNet can be a good compromise between sparsity and stability.

We are done for the well-known linear regression models, I will speak about non linear models later in this article. What I wrote here is largely inspired from the very clear book *Data Science - Fondamentaux et études de cas* of Eric Biernat and Michel Lutz.
I was deliberately not clear enough about how LASSO provides sparsity in your dataset and also why it was not stable in presence of highly correlated variables. I strongly recommend you to watch this amazing course from David Rosenberg about L1 and L2 regularization : `https://www.youtube.com/watch?v=d6XDOS4btck&list=PLqIj-nyfGu55TQpzjv_kwZcz3pJSW9bnH&index=6`
In this video he explains clearly all the points mentioned above.

Bonus : Least Square Method
Let us now consider that we are not in a one dimensional problem to be more general. Given $n \in \mathbb{N}$ observations in $\mathbb{R}^m$ let us define the matrix

$$X = \begin{pmatrix} 1 & x_{11} & .. & x_{1m} \\ . & . & .. & . \\ . & . & .. & . \\ 1 & x_{n1} & .. & x_{nm} \end{pmatrix}$$

and

$$\theta = \begin{pmatrix} \theta_0 \\ . \\ . \\ \theta_m \end{pmatrix} \quad Y = \begin{pmatrix} y_0 \\ . \\ . \\ y_n \end{pmatrix}$$

Now let us consider the residuals $r_i = y_i - \sum_{j=0}^{m} x_{ij}\theta_j$ which is the "error" of the hypothesis function. The cost function J can be written as follows :

$$J = \frac{1}{2n}\sum_{i=1}^{n} r_i^2$$

Its partial derivative with respect to θ_j for $j \in [1, m]$ is

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{n}\sum_{i=1}^{n} r_i \frac{\partial r_i}{\partial \theta_j}$$

For the solution θ^* of our optimization problem these partial derivatives must be equal to 0. This yields to $\forall j \in [1, m]$:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} x_{ij} x_{ik} \theta_k^* = \sum_{i=1}^{n} x_{ij} y_i$$

The proof is left to the reader. This equation can be written in a matrix form : $(X^T X)\theta^* = X^T Y$ and if the matrix $X^T X$ is invertible (non degenerate) the analytic solution of θ^* is

$$\theta^* = (X^T X)^{-1} X^T Y$$

We can now talk about classification problems

3 Supervised Learning : Classification

3.1 Logistic Regression

Classification is a problem that you can face in numerous field, either when you want to decide to grant a loan or not to one of your client, or when you want to classify a mail as "spam" when it should be. To do so you need to have a function of your different features that provides you an answer to the question asked by your classification problem. One of the most famous method to solve classification questions is the Logistic Regression. This name can be misleading but we are talking about classification here. To simplify the study I will stick to a binary classification problem, and I will say that we need to assign either 1 or -1 to our data. I will talk about multiclass classification later in this paper.

Sigmoid functions

Before getting deeper in this topic and the means we use to create decision function I need to introduce sigmoid functions. Sigmoid is a generic term that refers to functions with "S" shape. See below some examples of the most used :

- Tanh (hyperbolic tangent):

For $x \in \mathbb{R}$, $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
This function is a bijection from $\mathbb{R}$ to $[-1, 1]$

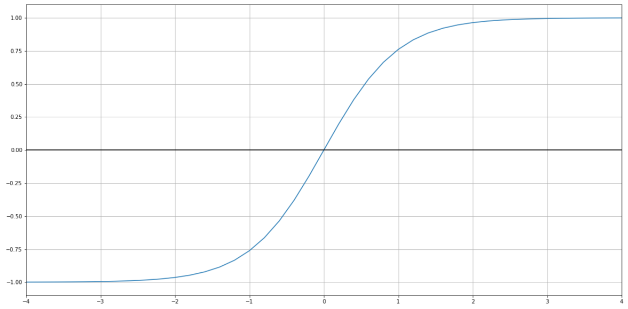

Fig. 6. tanh function

- Logistic function :

For $x \in \mathbb{R}$, $g(x) = \frac{1}{1+e^x}$. This function is a bijection from $\mathbb{R}$ to $[0, 1]$. Its graph is flatter that tanh's function, bankers would say that it has a bigger kurtosis.

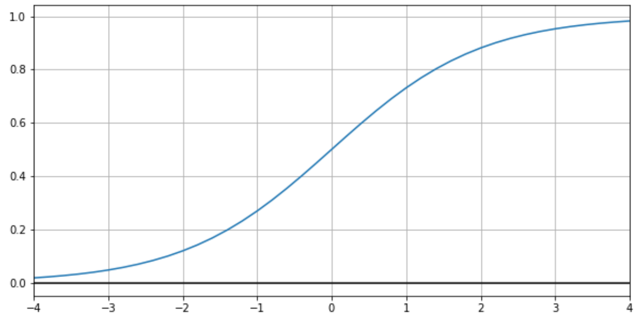

Fig. 7. Logistic function

- Cumulative distribution function of the normal law $N(0{,}1)$:

For $x \in \mathbb{R}$, $N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{x^2}{2}}\,\mathrm{d}x$

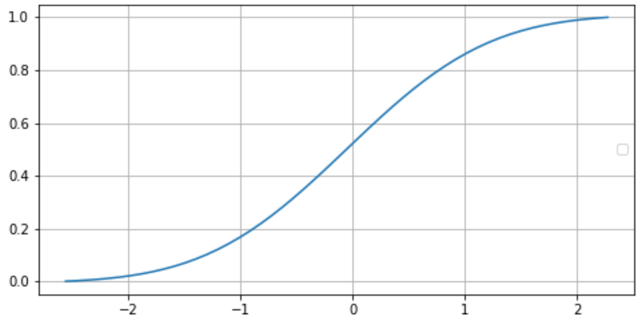

Fig. 8. Normal law distribution function

Why do we need such functions ? Indeed, if we think about it, we could have only needed discrete functions that assigns 1 or -1 with respect to our parameters. However, sigmoid functions bring something pretty interesting : it is like if we had a probability of the affiliation to a certain class. If your function brings you a result about the prediction, you may need to have a certainty measure of this affiliation likelihood.

Decision function

Let's take the same set as usual : n different vector points $X_1, ..X_n$ of dimension

m, and for $\theta = (\theta_1, .., \theta_m) \in \mathbb{R}^m$ and $i \in [1, n]$ we will consider the vector $Z_i = \theta^T X_i$. For this example I will chose the tanh as the sigmoid function g. What we want is a decision function h such that :

$$h(Z_i) = \begin{cases} 1 & \text{if } g(Z_i) > 0 \\ -1 & \text{if } g(Z_i) < 0 \end{cases}$$

Which is exactly equivalent, with tanh as the sigmoid function, to :

$$h(Z_i) = \begin{cases} 1 & \text{if } \theta^T X_i > 0 \\ -1 & \text{if } \theta^T X_i < 0 \end{cases}$$

As a reminder $\theta^T X_i$ is the inner product between θ and X_i. As a consequence the condition $\theta^T X_i > 0$ can be interpreted geometrically as "θ^T and X_i are in the same direction"

Cost function

Now that we found a decision function, previously called hypothesis function, we need a cost function. Unlike regression problems, it is not really possible to minimize distance between prediction and actual label because you either get the right label or the wrong one. So you must choose a function that will penalize the so-called false-positive (predict mistakenly 1) and the false-negative (predict mistakenly -1).

Considering $f(X) = -\log(tanh(\theta^T X))$, if your model predicted mistakenly -1, $tanh(\theta^T X)$ should be close to 0, $f(x)$ is then probably huge. This function thus penalizes false-positive cases. In the same way $p(X) = -log(1 - tanh(\theta^T X))$ penalizes the false-negatives cases, so you can add these functions to make an error measure. Since your true label y can be either 1 or -1 your error function j has the form :

$$j(X) = -(1 + y)\log(h(X)) - (1 - y)\log(1 - h(X))$$

with $h(X) = tanh(\theta^T X)$, j is called the *cross-entropy.*

Hence the cost function you will try to minimize when training your model on your data $X_1, ..X_n$

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} -(1 + Y_i)\log(h(X_i)) - (1 - Y_i)\log(1 - h(X_i))$$

You can now find your parameters $\theta_1, .., \theta_m$ by proceeding a gradient descent.

3.2 Decision Trees

For those who may be allergic to optimization and long mathematical formulas, this section should reconcile you with machine learning. You can see a decision tree as a tree where each node corresponds to an answer to a certain question.

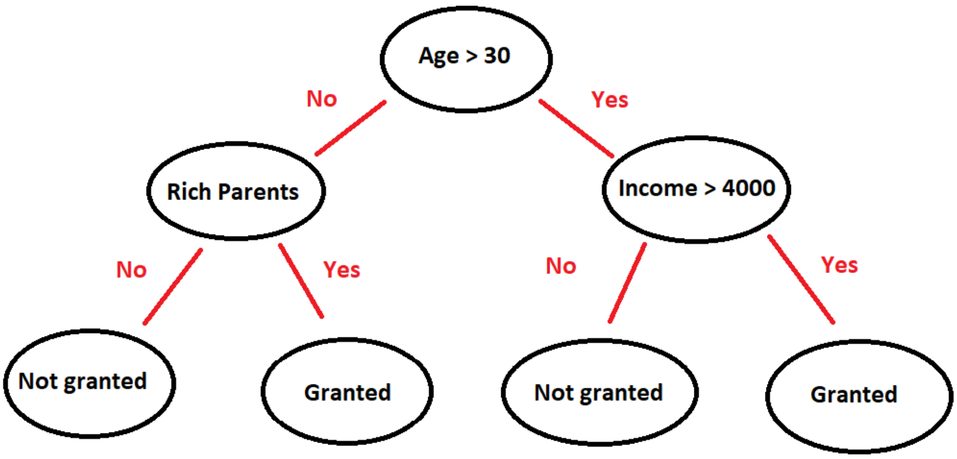

Fig. 9. Loan Decision Tree

Let's say that you want to know if your banker will accept you to take out a loan, to simplify the problem you assume that the bank has three different criteria to decide if you are eligible : your age, your parents living conditions and your income. If you are under 30 and do not have the chance to come from and upper class family, this tree tells you that you won't be able to take out a loan at this bank. On the contrary, if you are over 30 and you earn more that $4000 per month, then you will be eligible !

Now that you are familiar with decision tree you may wonder how it can be useful to solve your data science problem. And this is reasonable ! The use of decision trees is called CART (classification and regression trees) method.
In fact, a decision tree will look at the features of your dataset and group node after node your samples into group until the end. The edge of the tree is composed by *leaves*, those leaves contain the information that we are interested in : the label. So when you have an unlabeled element, you answer the questions

of each nodes to know your way through, and when you reach a leaf you know which label to assign to your element.
However, by which magical spell is your tree able to choose which feature to look at, and what selection criteria it should use on this selected feature ?

Criteria
As usual there is no magical ability behind decision tree, it's all about criteria. Indeed the tree will proceed as follows to split the nodes : first choose a splitting criterion that will be used during the whole process, and then find the splitting feature that will give the best result with respect to this criterion. It may remain a little obscure but let me introduce the two main criteria used for decision trees:

• Gini Criterion : this criterion aims at splitting the most represented class in the basket, it means that the goal of this criterion is to choose a feature and a condition about this feature such that the results will be two nodes where a class is almost absent from one of the two nodes. Let's say we are dealing with a binary class problem, let's denote p_i the proportion of the class i in the dataset ($i \in \{0, 1\}$). Let's define the Impurity of Gini : $I_G = 1 - p_0^2 - p_1^2$. This impurity is minimized when one of the two p_i^2 is far greater than the other one. See below a example of a decision tree using Gini criteria :

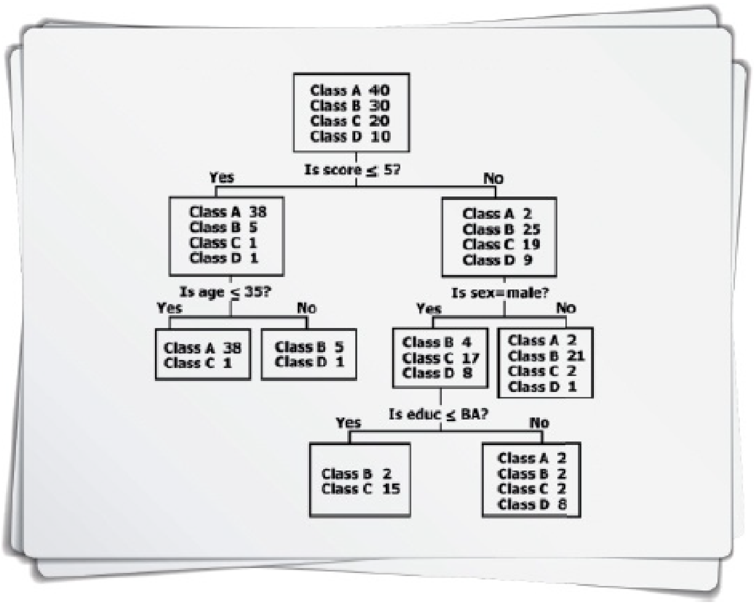

Fig. 10. Gini Criterion, from *Data Science, fondamentaux et études de cas*

• Entropy Criterion : this criterion relies on the concept of entropy, for those who remember their physics courses entropy is a measure of disorder. In classification this entropy can be modeled by the impurity $I_E = -p_0 log(p_0) - p_1 log(p_1)$ This should remind you the cross-entropy function used in logistic regression. Like Gini criterion, the goal is to minimize the impurity of each node at each split.

So which one should you use for your decision tree ? In fact it does not really matter, in general the results are pretty much the same. As a comparison see below a graph extracted from the course of David Rosenberg that shows the evolution of the two impurities defined above in function of p_0 :

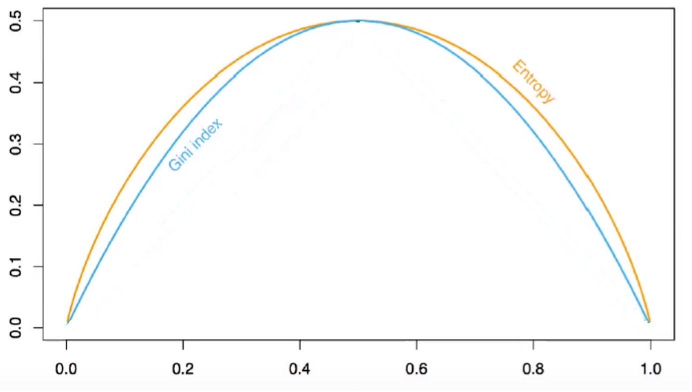

Fig. 11. Gini vs Entropy

Depth

There is still something missing in my Decision Tree presentation : the depth, and it is a very important parameter ! Indeed, you can proceed to split until having as much leaves as data samples, but is won't be really effective because your tree won't be able to generalize to other observation. Once again there is no rule of thumb for predefined depth, you will often try and see to find a good variance bias trade-off. This topic will be developed in its dedicated section.

3.3 Random Forest

Now that you understood clearly what was a Decision Tree I can finally speak about an unmissable algorithm : Random Forest. First of all let's think about what can be the drawback of CART method, indeed, the way it is built is *highly* depending on your initial sample, if you want to add new samples to your database to improve your model results it may radically change your tree, and this is bad news ! Moreover, taking the case of a person that may have a cancerous tumour, even after consulting an eminent doctor, this person should ask advice to other experts to be sure. This is exactly how random forest works.
Before introducing the algorithm I need to define two terms : *tree bagging* and *feature sampling*

Tree bagging
Given a dataset (X,Y) tree bagging will generate B new datasets (X_b, Y_b), $b \in [1, B]$ that will be created by sampling from D uniformly and with replacement. This means that you are generating a new dataset of the same size as (X,Y), but sampling with replacement will make observations repeating and some others missing.

Feature sampling
Feature sampling is a method that randomly chooses features of your dataset instead of taking the whole columns. By default if you have m features the random forest algorithm will select $\sqrt{m}$ features.

What are the advantages of these two operations ?

Let's say that the total variance of your dataset is denoted by σ^2. When you take B samples of randomly independent and identically distributed chosen variable inside your dataset, your total variance will be $V_{sampled} = \frac{\sigma^2}{B}$. If you do not have independance between those variables you can denote by ρ the correlation between the variables and write $V_{forest} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.
Now you see that tree bagging, which will create a partition of your initial datasets will reduce the term on the right side. As for feature sampling, this will reduce the correlation between trees in your forest, indeed, since they won't develop using the same information they will be less correlated than if they had access to all the features !
And why do we want to reduce variance on our predictions ? This is a good question and a possible answer is this : when a statistician is performing an estimation of a variable, he or she always provides a confidence interval, and this confidence interval is proportional to the variance of the estimator ! In our case, we are trying to estimate a label y, reducing the variance will thus provide more stability in the model, and thus a better prediction power ! This was what happened for polynomial regression, increasing the degree of your polynomial will reduce the bias, but increase the variance.

Random Forest in a nutshell : the algorithm will proceed to tree bagging and feature sampling, then it trains one tree per bag. And to finally assign a label to an element X all the threes vote and the majority wins.
Remark : despite my choice to talk about this algorithm in the classification section, it can be used for regression. Instead of getting the trees to vote it takes the average of the results.

Advantages :
Random Forest is a powerful algorithm that provides good predictions. It is also fast to train because it is a parallel process. Moreover it is pretty intuitive, because decision trees are easy to understand even if they are non linear models.

Drawback :
One of the biggest problem is that the results of this algorithm are not easily understandable, indeed when you are performing a linear regression it provides you coefficients that say "this variable is highly correlated to the outcome, this one is highly negatively correlated and this variable is uncorrelated to the result". When you use a random forest you only have access to the variables used at the top of the trees. It can give you information but you have to mitigate this information. Indeed, if you remember, decision trees split according to a criterion, it means they will favour highly scattered variable. And these variables are not always carrying information, try to insert a random noise feature in your dataset and you will see that it will appear at the top used splitting features.

3.4 Support Vector Machine (SVM)

The SVM is also a very famous algorithm for classification. We will still stick to binary classification for the examples and deal with multiclass in a later subsection.

Case of linearly separable points
See below the set of points that will be our guideline throughout this section :

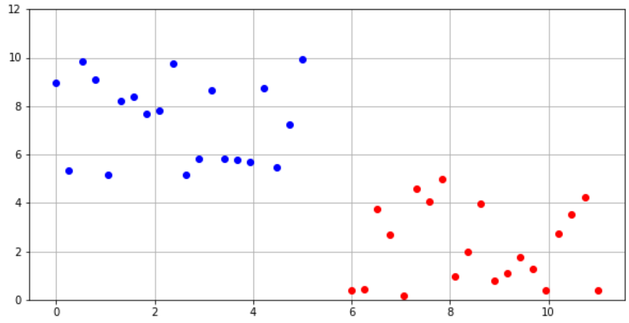

Fig. 12. Linearly Separable dataset

You would like to find a line that could separate the blue points from the red ones. I will propose the three lines below :

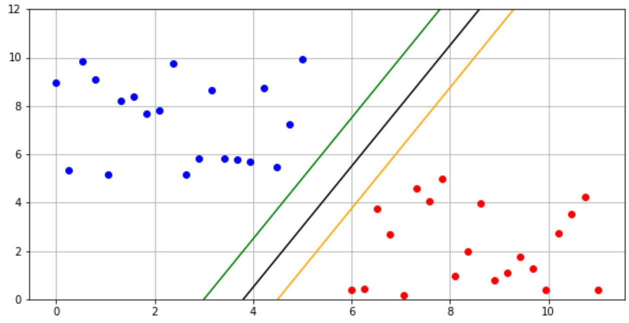

Fig. 13. Multiple splitting hyperplans

Which on would you rather take ? The black one right ? Indeed, it is more logical to take the one that is as far as possible as the points in our dataset, it is "safer". In fact, the SVM is designed the same way. The points that are the closest from the line are called the *support vectors*. The idea is to draw a line that will maximize the distance between these support vectors. But how does it proceed ?

Reminder : Linear Algebra
Let H be an Hilbert space (a complete vector space with an inner product) of dimension $n \in \mathbb{N}$, $n \geq 2$. A hyperplan of H is any subspace G of H of dimension $n-1$, such an hyperplan can be totally described with a vector $w \in H$. Indeed, for every subspace G of H there exist w_1 and $w_0 \in H$ such that $G = \{x \in H | w_1^T x + w_0 = 0\}$

Here we are in two dimensions, let's define the distance d such that for $x, y \in \mathbb{R}^2$, $d(x,y) = ||x-y||_2$. Let's denote S the index set of support vectors in our case, we thus want to find the "best" hyperplan than separates our two groups, by "best" I mean the one that maximize the margin between the hyperplan and the support vectors. Mathematically this means we are looking for

$$w^* = (w_1, w_0) = \underset{w=(w_1,w_0)\in\mathbb{R}^2}{\operatorname{argmax}} \ \min_{s\in S} d(x_s, G(w))$$

Here $G(w)$ denotes the hyperplan engendered by w, and $d(x_s, G(w))$ is the distance between x_s and $G(w)$, this distance is the L2 norm of the orthogonal projection of x_s on $G(w)$.
Now you need to calculate $d(x_s, G(w))$. You can prove that :

$$d(x_s, G(w)) = \frac{|x_s^T w_1 + w_0|}{||w_1||_2}$$

Back to SVM :
Let us consider $p \in \mathbb{N}$ different points in $\mathbb{R}^n$, with their p labels. We determined how to build a hyperplan in our vector space, we have thus to use this to determine the one we need to split the data. What we need :
• Find w^* such that $\forall p$, $y_p(x_p^T w^1 + w_0) \geq 0$ where y_p is the label of x_p, 1 or -1 (separability condition)

• Find w^* such that

$$w^* = \underset{w\in\mathbb{R}^2}{\operatorname{argmax}} \min_{s\in S} d(x_s, G(w))$$

Hence the constraint optimization problem :

$$w^* = \max_{w\in\mathbb{R}^2} \min_{s\in S} \frac{|x_s^T w_1 + w_0|}{||w||_2}$$

$$\big\{ \text{such that } \forall p,\ y_p(x_p^T w_1 + w_0) \geq 0$$

There are now two difficulties to overcome : the absolute value, and the min inside the max. As for the first one if you pay attention to the constraint you can write $|x_s^T w_1 + w_0| = y_s(x_s^T w_1 + w_0)$.
Now let's have a look at the equation that engenders a hyperplan $G(w) = \{x \in H | x^T w_1 + w_0 = 0\}$ for $w = (w_1, w_0) \in \mathbb{R}^n \text{x} \mathbb{R}^n$. If you replace w by any kw for $k \in \mathbb{R}, k \neq 0$, the equation stays the same, this means that you can take arbitrarily any kw to be the representative of $G(w)$. Let $s \in S$ the index of the closest point to the separating hyperplan, $y_s(x_s^T w_1 + w_0) = m$, now take $w_1^{'} = \frac{w_1}{m}$ $w_0^{'} = \frac{w_0}{m}$ as new representatives, you thus have $y_s(x_s^T w_1^{'} + w_0^{'}) = 1$

The problem is now

$$w^* = \max_{w \in \mathbb{R}^2} \frac{1}{||w||_2}$$

$$\big\{ \text{such that } \forall p,\ y_p(x_p^T w_1 + w_0) \geq 1$$

Maximizing $\frac{1}{||w||_2}$ is the same as minimizing $||w||_2$. Mathematicians will nonetheless prefer to minimize $\frac{1}{2}||w||_2^2$ to get rid of the square root during the optimization process. This finally leads to the problem :

$$w^* = \min_{w \in \mathbb{R}^2} \frac{1}{2}||w||_2^2$$

$$\big\{ \text{such that } \forall p,\ y_p(x_p^T w_1 + w_0) \geq 1$$

You can solve this problem taking its Lagrangian and use KKT conditions, this will be explained in the dedicated section. Then you find your separation hyperplan.

Problem : The way I presented the SVM does not allow any bias, there is thus no training errors. Have a look at this :

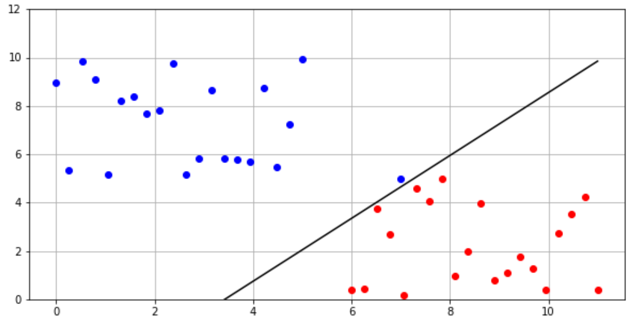

Fig. 14. Outlier issue for SVM

You probably think that we should ignore this outlier to avoid overfitting, and you are right, this is exactly the idea of "soft margin". Soft margin allows you training errors to find the right hyperplan that better separates your set of points. Hence the new formulation of the problem :

$$w^* = \min_{w \in \mathbb{R}^2} \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{p} \epsilon_i$$

$$\{ \text{such that } \forall p,\ y_p(x_p^T w_1 + w_0) \geq 1 - \epsilon_i$$

Where ϵ_i for $i \in [1,p]$ is the training error allowed on each point and C a constant. This looks like regularization (remember LASSO), and C is very important because for high values of C you go back to the first form of SVM and risks overfitting, and for low values of C you may underfit.

Kernel Trick

I exposed here the SVM in a linearly separable problem. What if it is not the case ? Should we forget about this algorithm ? Of course not ! Have a look at the problem below :

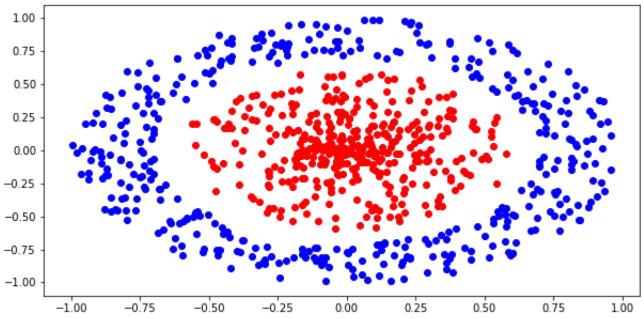

Fig. 15. Non Linearly Separable dataset

This is obviously not linearly separable. To handle this case you can try to find a new vector space with a higher dimension where the problem may be linearly separable. This is called the *kernel trick*, it consists in finding a function of your original space towards a higher dimension space. In our example our dataset is composed by two dimensional vectors. Let's denote this original set X and for $x \in X$ we write $x = (x_1, x_2)$, if we want to make the problem linearly separable we can use the function $\psi : X \rightarrow \Xi$ such that $\forall x \in X$, $\Psi(x) = (x_1, x_2, x_1^2 + x_2^2)$ There you have a new dataset and you can find a plan that separates these data :

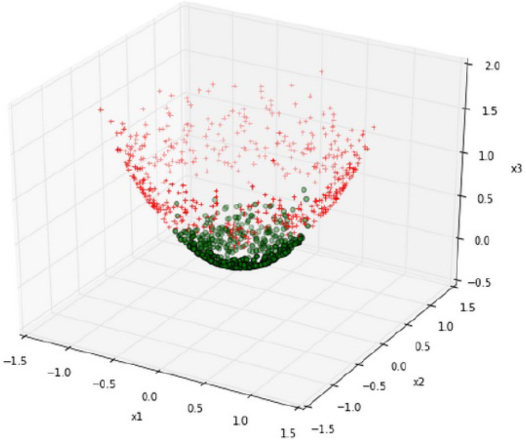

Fig. 16. Kernel Trick

For more information about kernel methods, you can have a look at the video of David Rosenberg that deals with this interesting subject : https://www.youtube.com/watch?v=m1otj-SdwYw&index=13&list=PLqIj-nyfGu55TQpzjv_kwZcz3pJSW9bnH

Remark

I decided to present Random Forest and SVM algorithms in the classification section, however, they are also used in regression problems. As for Random Forest it is pretty easy to understand how it works : instead of proceeding to a majority vote to choose the class whose x belongs, just take the average of the predictions. However, it is not that easy to understand for SVM. intuitively, as all regressors it tries to fit a line to data by minimizing a cost function. However, the interesting part about SVR is that you can deploy a non-linear kernel. In this case you end making non-linear regression, i.e. fitting a curve rather than a line (credit for this explanation to *iliasfl* on stats.stackexchange.com). You can find a complete presentation here : https://cs.adelaide.edu.au/~chhshen/teaching/ML_SVR.pdf

3.5 K-Nearest Neighbours : K-NN

After this very maths-oriented section let's go back to something more visual. We will consider a set of $m \in \mathbb{N}$ already labeled training points, no need to consider binary classification anymore. Then you introduce a new point x that you want to label, you could use one of the algorithms above, or you could use k-NN method. The k-NN algorithm (that should not be confused with k-means algorithm), for $k \in [1, m]$ assigns the most represented label among the k nearest neighbours of x. Have a look at the set of points below.

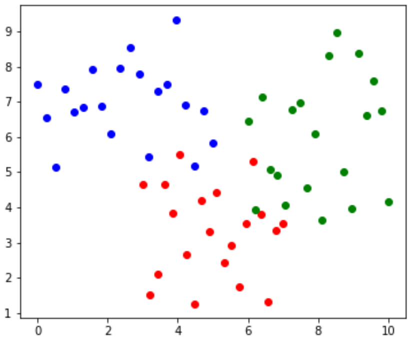

Fig. 17. Labeled dataset

Imagine that the point $x = (4.1, 4.8)$ is the point we want to predict the label (in black):

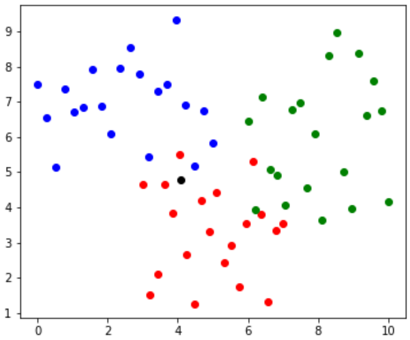

Fig. 18. Unlabeled point inserted

Applying the 4-NN yields to the below :

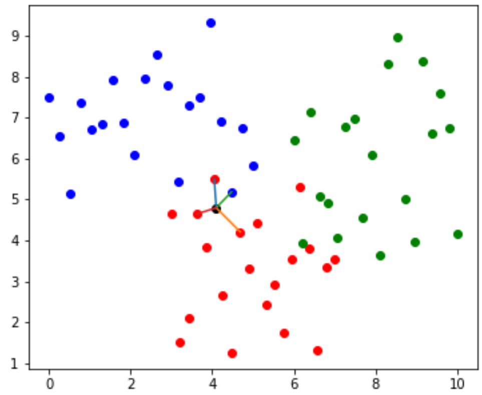

Fig. 19. K nearest neighbours selection

Here we see that our point is linked to three red points and one blue point, the predicted label will thus be 'red'.

It is interesting and should work quite well, but one thing needs to be clarified : how do you choose k ? In fact a general rule of thumb is to take k equals to $\sqrt{m}$. However, there is no real rule to do so, the best way to figure out which k suits the best to your problem is to use cross validation. You divide your dataset into one training set and one test set, apply k-NN for different values of k and choose the one that gives you the best classification score.

Advantage :
k-NN algorithm is very easy to understand and it works well on basic problems.

Disadvantages :
You have to compute a distance between each element, which is not always easy (for instance between different customers). Also, this algorithm does not really learn, it only uses the training dataset, hence instability : changing k can drastically change the predictions.

3.6 Naive Bayes

The last classification algorithm will be Naive Bayes. As a reminder please see Bayes theorem below :
For two events A and B you have

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

That can be written this way too :

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}$$

And for independent events A and B : $\mathbb{P}(A|B) = \mathbb{P}(A)$, $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$

For people who are not very familiar with these notations, $\mathbb{P}(A)$ represents the probability for an event A to happen, its intrinsic probability. And $\mathbb{P}(A|B)$ is the conditional probability of A given B, this measures the probability of the event A knowing that B happened.
Let's make an example to understand how Naive Bayes works, and why it is called "naive". Below a table extracted from mrmint.fr that describes a dataset of 1000 fruits with different features : color, shape and sweetness.

Type	Yellow	Not Yellow	Long	Not Long	Sweet	Not Sweet	Total
Banana	450	50	400	100	350	150	500
Orange	300	0	0	300	150	150	300
Other Fruit	50	150	100	100	150	50	200
Total	800	200	500	500	650	350	1000

You now have to classify a fruit that is yellow, long and sweet. Is it an orange or a banana ? First evaluate $\mathbb{P}(Banana|Yellow, Long, Sweet)$ and $\mathbb{P}(Orange|Yellow, Long, Sweet)$, then choose the greatest value to classify the fruit.
According Bayes formula :

$$\mathbb{P}(Banana|Yellow, Long, Sweet) = \frac{\mathbb{P}(Yellow, Long, Sweet|Banana)\mathbb{P}(Banana)}{\mathbb{P}(Yellow, Long, Sweet)}$$

The assumption of Naive Bayes algorithm is that every features are independent with respect to each other, this is why it is called "naive", indeed it is pretty rare that all the features of the problem are independent. Using this assumption you re-write this :

$$\mathbb{P}(Banana|Yellow, Long, Sweet)$$
$$= \frac{\mathbb{P}(Yellow|Banana) * \mathbb{P}(Long|Banana) * \mathbb{P}(Sweet|Banana) * \mathbb{P}(Banana)}{\mathbb{P}(Yellow) * \mathbb{P}(Long) * \mathbb{P}(Sweet)}$$

You have :
$\mathbb{P}(Yellow|Banana) = \frac{450}{500} = 0.9$, the same way : $\mathbb{P}(Long|Banana) = 0.8$, $\mathbb{P}(Sweet|Banana) = 0.7$, $\mathbb{P}(Banana) = \frac{500}{1000} = 0.5$, $\mathbb{P}(Yellow) = \frac{800}{1000} = 0.8$, $\mathbb{P}(Long) = \frac{500}{1000} = 0.5$, $\mathbb{P}(Sweet) = \frac{650}{1000} = 0.65$

Hence :

$$\mathbb{P}(Banana|Yellow, Long, Sweet) = \frac{0.9 * 0.8 * 07 * 0.5}{0.8 * 0.5 * 0.65} \approx 0.97$$

The same way you can find $\mathbb{P}(Orange|Yellow, Long, Sweet) = 0$ and $\mathbb{P}(Other\ Fruit|Yellow, Long, Sweet) = 0.072$. Finally, if your fruit is yellow, sweet and long you assign it the Banana label.

Advantages :
This algorithm is fast to train, it is able to work on small datasets. Moreover, contrary to some other classifiers it works with multiclass problems easily.

Disadvantages :
Independence of the features assumption.

However, despite this independence assumption the algorithm works pretty well. You can have a look at this paper of Harry Zhang that develops this topic `http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf`

4 Unsupervised Learning

4.1 Hierarchical Clustering

Now you saw how to give a label/value to a new element from an unlabeled dataset, it can be helpful in several situations as I said like house pricing, cancer detection, volatility prediction etc. However, you can face another type of problem as a data scientist : grouping data together into homogeneous groups, called *clusters*. For instance take set of flowers with unknown labels, you may want to group them into different clusters in order to classify them. Or imagine a marketing company that have to forecast how many types of advertisement they will need to buy in order to reach all their customers. They may need to group them (dimension reduction), and then analyze their features to understand which publicity suits the best to each group.

Topology is key
Before starting to learn what kind of algorithm can be used for our purpose you need to have in mind that for unsupervised learning, the topology is one of the most important parameter, and by topology I mean metrics. Indeed, while metrics were used for cost functions and validation purpose in supervised learning, it is the parameter that will drive your whole model in unsupervised learning. In fact, if you want to make groups you need to evaluate how *close* two elements are from each other, and this proximity needs to be quantified using a distance. You need to be sure that you are using a distance that is in line with the idea you have of likeness between your elements.

After pointing out the importance of topology, I can speak about one method to perform this task : Hierarchical Clustering. There are two types of hierarchical clustering, the first one is called conglomerative and the second one is called divisive.
• Conglomerative : this kind of algorithms works by aggregating elements with each other to create new groups.
• Divisive : contrary to conglomerative algorithms, divisive method consists in successive dichotomies of the original dataset.

The goal of these two methods is the same : create relevant clusters, by relevant I mean clusters that maximize the similarity intra-class and minimize the similarity extra-class. The two algorithms lead to the same structure called a dendogram :

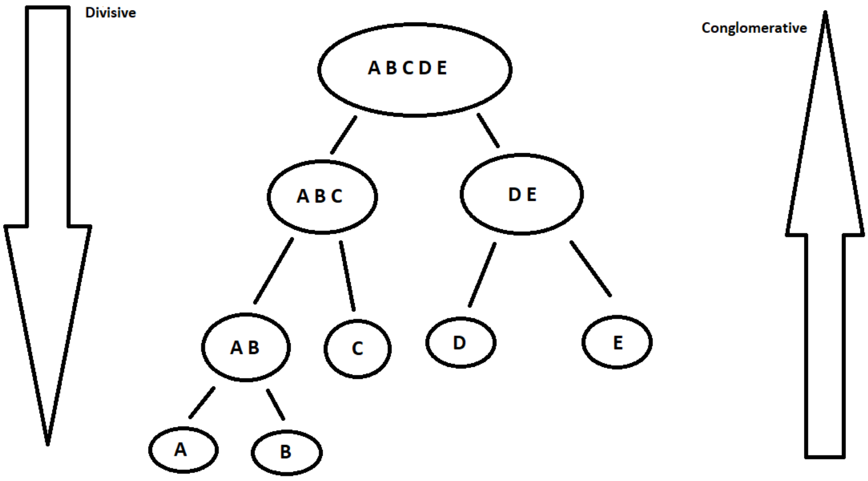

Fig. 20. Hierarchical Clustering

The most popular is the congleromative one. Assuming you have m observations it works in 5 steps :

- Step 1 : Construction of a distance matrix between the m elements
- Step 2 : Research of the two closest elements in the cluster
- Step 3 : Aggregation of these two elements and formation of a dataset of m-1 elements
- Step 4 : Creation of a new distance matrix removing the two aggregated elements and adding the new group
- Step 5 : Repeat these steps with the new dataset

Pretty easy isn't it ? Now you must have two questions in mind :
- How do you define a distance for the new created group ?
- How do you know when you have to stop aggregating ?

The first question deals with aggregation criteria. How do you evaluate the distance between two groups composed by several points ? You can use this (non exhaustive) list of criteria :
- Consider the minimum of the distance between all the points of your clusters
- Consider the maximum
- Consider the average
- Use the center of gravity of the two clusters and consider the distance between these two centroids

Let's deal with the second question, it should remind you the thought about decision trees depth which had no real answer but "try and see". Do not forget that the problem here is slightly different : you want to create groups to help you to better understand your customers, your products etc. It is a business problem, the best way to know when to cut your tree is when you do not manage to give a business meaning to your clusters anymore. However, there exist quantitative methods to choose when to stop the algorithm, they will be introduced in the next section.

4.2 K-Means Clustering

Given a dataset $X = \{x_1, .., x_n\}$ of n points, k-Means algorithm aims at finding k clusters $S = \{S_1, .., S_k\}$ that minimizes the distance between the points of a each cluster and their centroids. Mathematically this means that it looks for the best partition S of X i.e :

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in S_i} d(x, \mu_i)^2$$

Where d is the distance that you chose to characterize the problem and μ_i for $i \in [1, k]$ is the centroid of each cluster S_i. The algorithm works as follows :
First you have to randomly choose k points of the dataset. Then you make k clusters with the points that are the closest to these k points regarding the distance already defined. Once the k clusters are created, calculate their centroids. After that you can make new clusters using the same method, and so on.

To sum up I would say that there are 5 steps :
Step 1 : Define the appropriate distance for the problem (topology is key)
Step 2 : Randomly choose k points in the dataset, call them centroids
Step 3 : Look at each point of the dataset, find the closest point among the centroids, and associate it to the centroid. This will create k clusters by association
Step 4 : Calculate the new centroids of the k clusters
Step 5 : Repeat steps 3,4 and 5 until convergence

Remark : the centroids calculated at each step do not necessarily belong to the dataset

Illustration of the algorithm :

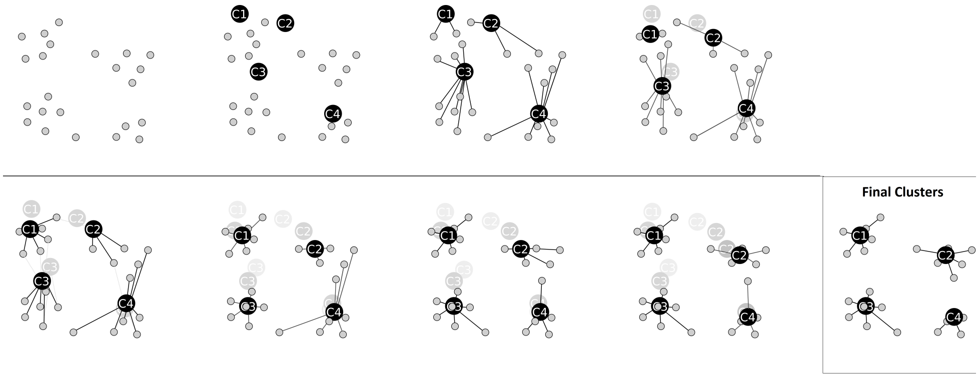

Fig. 21. K-means algorithm

Now than we understand the different steps of the algorithm we have to understand if it does converge and if it gives us the clustering verifying the optimization problem above.

Convergence :
The number of possible clusters is $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, so there is a finite number of clusters. At each step, the global entropy is decreasing, because the algorithm calculates the centroids. Then the entropy is decreasing in a finite set of possibilities, hence the convergence.

However, convergence towards a minimum does not guarantee that this is a global minimum. Indeed, the optimum clustering depends on the initialization and can thus converge to a local minimum. See below an example :

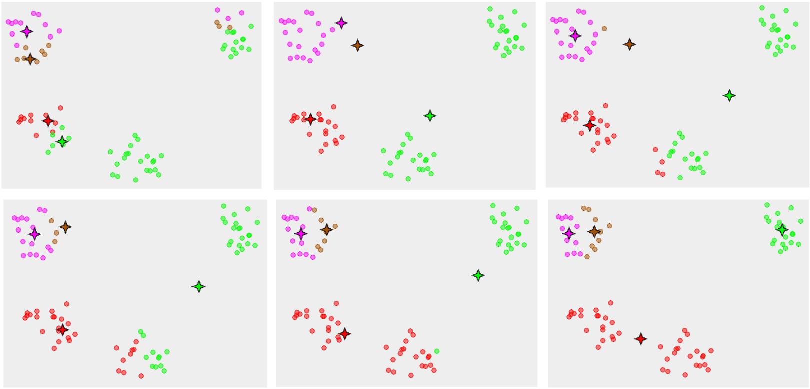

Fig. 22. K-means local minimum

The problem here is that there should be two clusters at the bottom left corner instead of one. This can be mathematically justified by a too high inter-class variance. To overcome this issue data scientists sometimes run the k-Means algorithm several times with different random initialization and keep the best clustering out of all the trials. There also exists a method called k-means++, but it is out of the scope of this short paper.

Scaling Importance:
Let's consider now a two dimensional problem in $\mathbb{R}^2$. Taking a dataset of n points as usual, and using the euclidean norm as a distance. If your two coordinates of each point are not on the same scale (for example varying from 1 to 10 for the first one and from 100 to 1000 for the second one) the distance will be highly influenced by the higher scaled coordinate. Then if you scatter plot your data

and the clusters you will see that it may not be split into the parts that you would have chosen.
Do you always need to scale the data ? In fact it is not mandatory, it only depends of the goal of the clustering, if you do want a feature to have a higher impact than another you will prefer not to scale. However, if you want each feature to have the same importance then you should scale the data.

Non-linear Issue

I did not discuss about the shape of the clusters yet. The k-Means algorithm draws lines to separate the different clusters, the different areas can be seen as the Voronoi cells of the centroids :

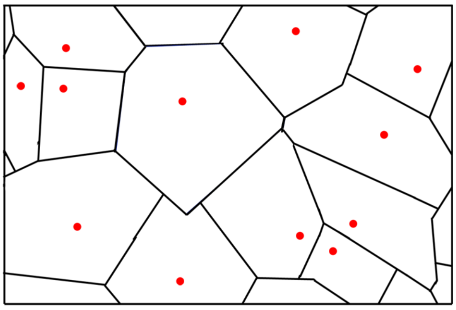

Fig. 23. Voronoi Cells

So, if you have a non linear or a non convex dataset representation, like nested circles, applying the k-Means algorithm will lead to poor results.

4.3 DBSCAN

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. While k-Means algorithm links points to clusters with respect to their distance to centroids, DBSCAN will proceed in a different way. It will consider the neighbourhood of each point and look at the neighbours which are close.

Definitions :
Given a dataset X of n points, and $x \in X$:
• n_{min}: Hyperparameter of the model used to determine *core points*
• ϵ-neighbour : $N(x) = \{y \in X, \text{ such that } d(x, y) < \epsilon\}$
• core point : a point $x \in X$ such that card$(N(x)) \geq n_{min}$
• $y \in X$ is said to be *density reachable* from $x \in X$ if and only if there exist m points $a_1, .., a_m$ such that $y \in N(a_1)$, $a_1 \in N(a_2)$,.., $a_m \in N(x)$. It is an equivalence relation.
• outlier : a point $y \in X$ such that card$(N(y)) = 1$ i.e a point not density reachable by any other point

See below an illustration of these definitions with $n_{min} = 3$: red points are the *core points*, yellow points are *density reachable* points from *core points* and the blue one is an *outlier* :

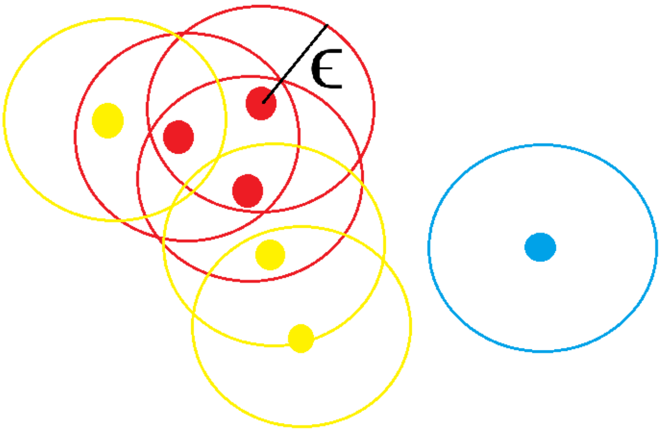

Fig. 24. DBSCAN core points, from openclassroom

Algorithm :
DBSCAN Algorithm can be described in 4 steps :

Step 1 : Find the ϵ-neighbour of each point of the dataset
Step 2 : Identify *core points*
Step 3 : Find the *density reachable* points of all the core points
Step 4 : All the density reachable points constitute clusters, outliers left aside

On the image below a clustering that would have failed using k-Means algorithm :

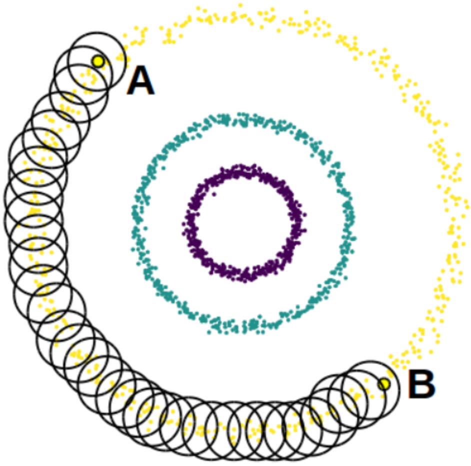

Fig. 25. Density Reachability

Here you clearly see the process of density reachable points.

Advantages :
Contrary to k-Means clustering, DBSCAN do not need to know *a priori* how many clusters you want to have, it determines this number on its own. Moreover this algorithm is very robust to outliers because it discriminates them using density reachability. And the greatest advantage is the fact that it can create clusters of different shapes like circle, commas etc.

Disadvantage :
The main characteristic of DBSCAN is the idea of neighbourhood, and this neighbourhood is defined as a ball centered on the *core point*. This yields to an exposition to the curse of dimensionality, indeed, in high dimension balls tends to be empty (this will be explained in a dedicated section). It can therefore create a lot of outliers that should be considered as such.

4.4 How many clusters ?

In this section I talked about three different clustering algorithms : Hierarchical Clustering, K-Means and DBSCAN. The first two ones need an important feature : the number of clusters. The question is how to evaluate the best k ? As you may expect there is no real answer to this question, it depends on different factors. Indeed, there is always an underlying business problem that you are trying to solve through your clustering, so this should give you an idea of how many groups you want. However, there are quantitative methods to find this k.

Elbow Method :
Remember : the goal of clustering is to minimize the inter-class variability, this can be measured and provide an inertia. Let's define the Within-cluster Sum of Square (WSS) of a cluster S with a centroid μ :

$$WSS(S) = \sum_{x \in S} d(x, \mu)^2$$

Then you can define the global inertia as follows : $I = \sum_S WSS(S)$. The elbow method consists in plotting the global inertia with respect to the number of clusters and choose the k that corresponds to a drastic change of the slope. The form of the graph is an elbow, see the illustration below :

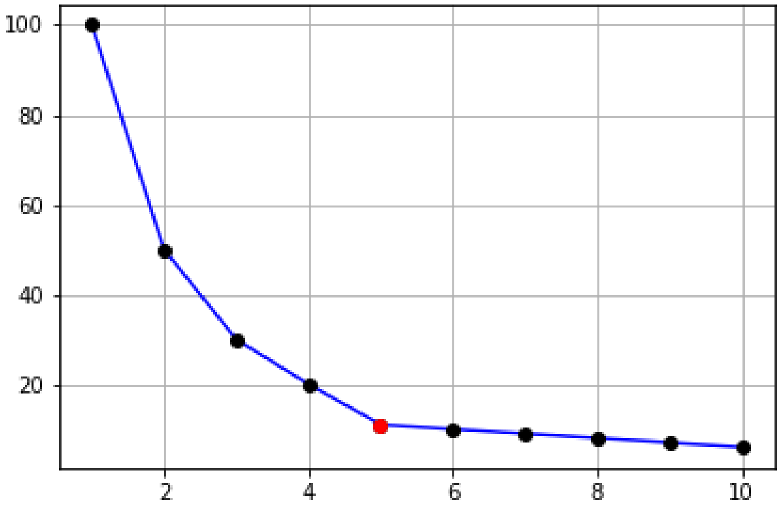

Fig. 26. Elbow Method

Here you can notice a strong change on the slope for k = 5. Hence the number of clusters we will choose with the elbow method : 5. Inertia converges towards 0 when k is equal to the number of samples in your dataset.

The elbow method can be summarized in 3 steps :
Step 1 : Run your clustering algorithm for different k and compute the global inertia for each k
Step 2 : Plot the inertia with respect to each k
Step 3 : Choose the k that corresponds to the elbow

However, it can sometimes happen that this "elbow" is not as striking as above, so you may need a more sophisticated method to find your optimal number of clusters.

Gap Statistic :
Gap Statistic method is a general approach that can be applied to any kind of clustering. The underlying idea is to compare the global inertia of the clustering to the clustering's inertia of a random uniform distribution. The farther you are from the this inertia, the more relevant your clustering is. Let us formulate it in a more mathematical way.
For $k \in \mathbb{N}$ clusters $\{S_1, ..S_k\}$ we define the inner-inertia

$$D_i = \sum_{x,y \in S_i} d(x,y)^2$$

and let us denote $n_i = \text{card}(S_i)$ for $i \in [1, k]$. The within-cluster dispersion W_k is

$$W_k = \sum_{i=1}^{k} \frac{1}{2n_i} D_i$$

You now need to define a reference distribution on which you will calculate the expectation of the within-cluster distribution. You can then define the gap statistic as follows :

$$\mathit{Gap}(k) = \mathbb{E}_{ref}[\log(W_k)] - \log(W_k)$$

The goal is thus to maximize this gap with respect to k. You now need to define your null reference distribution, and then to compute the expectation. One can prove that the random uniform distribution is the most likely to make sparse clusters for the gap test, hence the choice of this null reference. To compute the expectation you can just proceed by Monte Carlo, sampling B random datasets and take the average of the results. You now can define the gap statistic as :

$$\mathit{Gap}(k) = \frac{1}{B} \sum_{b=1}^{B} \log(W_{bk}) - \log(W_k)$$

This Monte Carlo estimation of the expectation has an error that is proportional to its standard deviation, if you write $\overline{W} = \frac{1}{B}\sum_{b=1}^{B}\log(W_{bk})$ the standard deviation of the samples is

$$s(k) = \sqrt{\frac{1}{B}\sum_{b=1}^{B}(\log(W_{bk}) - \overline{W})^2}$$

You then denote the simulation error

$$s_k = s(k)\sqrt{1 + \frac{1}{B}}$$

Once done, your optimal k* is the smallest k such that $Gap(k) \geq G(k+1) - s_{k+1}$. This process can be summarized in 4 steps :
Step 1 : Create different clusters varying k, and compute their within-clusters distribution
Step 2 : Generate B different datasets using a random uniform distribution, apply your clustering algorithm for each k.
Step 3 : Compute the gap statistic for all k Step 3 : Evaluate the quantities $Gap(k) - G(k+2) + s_{k+1}$ for all k Step 4 : Select the smalles k such that the quantity defined above is greater than 0

5 Bibliography

Supervised Learning

Eric Biernat & Michel Lutz, *Data Science, Fondamentaux et Etudes de cas*

David Rosenberg :
- About L1 and L2 regularization, https://www.youtube.com/watch?v=d6XDOS4btck&list=PLqIj-nyfGu55TQpzjv_kwZcz3pJSW9bnH&index=6
- About Kernel Methods, https://www.youtube.com/watch?v=m1otj-SdwYw&index=13&list=PLqIj-nyfGu55TQpzjv_kwZcz3pJSW9bnH

Paul Paisitkriangkrai, *SVM Regressor Presentation*, https://cs.adelaide.edu.au/~chhshen/teaching/ML_SVR.pdf

Harry Zhang, The Optimality of Naive Bayes, http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf

Unsupervised Learning

Yannis Chaouche & Chloé-Agathe Azencott, DBSCAN article,
https : //openclassrooms.com/fr/courses/
4379436−*explorez*−*vos*−*donnees*−*avec*−*des*−*algorithmes*−*non*−*supervises*/4379571−
partitionnez − *vos* − *donnees* − *avec* − *dbscan*

Robert Tibshirani, Guenther Walther and Trevor Hastie, *Gap Statistic*, https://statweb.stanford.edu/~gwalther/gap
Gap statistic illustrations,https://datasciencelab.wordpress.com/tag/gap-statistic/