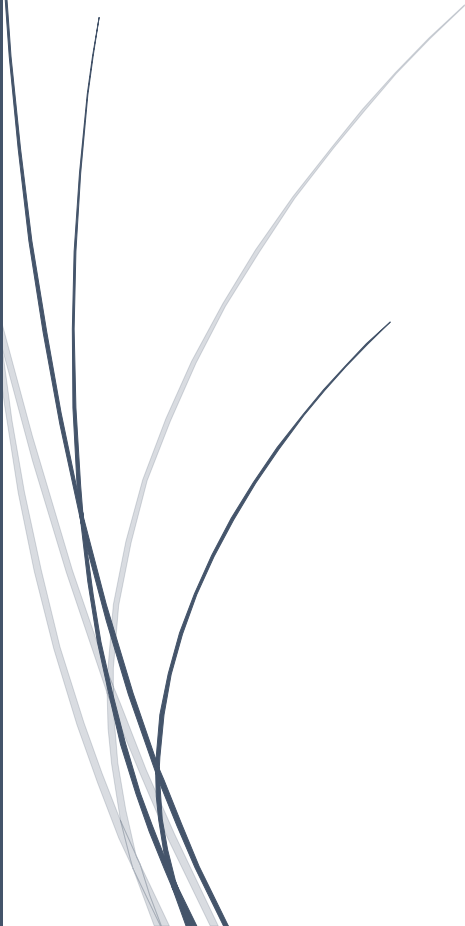




30/01/2019

# Understanding Machine Learning: From Theory To Algorithms

Bibliographic Project



Adam Basbas & Mohamed Amine Benbada  
2EME ANNEE GENIE INFORMATIQUE

- I – Introduction** ..... 3
  - 1. **What is learning ?** ..... 3
  - 2. **When do we need machine learning ?**..... 4
  - 3. **Types of learning**..... 4
- II – Laying the basis for our study** ..... 6
  - 1. **The statistical learning framework** ..... 6
  - 2. **ERM : Empirical Risk Minimization** ..... 7
    - a. **How this model can fail : overfitting**..... 8
  - 3. **ERM with inductive bias** ..... 8
    - a. **Finite hypothesis classes** ..... 8
    - b. **The realizability assumption** ..... 9
    - c. **The confidence parameter** ..... 9
    - d. **The accuracy parameter** ..... 9
- III – A formal learning model**..... 10
  - 1. **PAC learning**..... 10
  - 2. **PAC learnability**..... 10
  - 3. **Sample complexity**..... 10
  - 4. **A more general learning model**..... 11
    - a. **Agnostic PAC learning**..... 11
    - b. **The Bayes Optimal Predictor** ..... 12
    - c. **Agnostic PAC learnability** ..... 12
    - d. **The scope of learning problems modeled** ..... 12
    - e. **Generalized loss function** ..... 13
    - f. **Agnostic PAC learnability for general loss functions**..... 14
- IV – Linear predictors**..... 15
  - 1. **Halfspaces** ..... 15
  - 2. **Perceptron for halfspaces**..... 16
    - a. **How it works** ..... 16

<b>b. The algorithm</b> .....	16
<b>c. Perceptron in Java</b> .....	17
<b>V – Support-Vector Machine and k-Nearest Neighbors</b> .....	21
<b>1. Support-Vector Machine (SVM)</b> .....	21
<b>2. k-Nearest Neighbours (KNN)</b> .....	21
<b>VIII – Bibliography / Webography</b> .....	23
<b>IX – Table of figures</b> .....	24

## I – Introduction

In today's context, we read or hear the words machine learning multiple times a week, but do we really know its meaning? The term machine learning mostly refers to the automated detection of meaningful patterns in data. In the current context, it is mostly a tool to extract information from large data sets and everyone is using it every day that goes by without feeling its impact: search engines when they bring up the best results, spam detectors or credit card transactions, or more scientific fields such as bioengineering, medicine or astrophysics.

The common feature between these fields is that the extraordinary amount of data or information available makes it hard for mere human beings to detect any pattern, be it manually or through a hard coded program. Machine learning programs are here to save the day: they can learn and adapt.

### 1. What is learning?

Before diving deep into the subject, let's answer a primordial question: what is learning?

Nature is giving us multiple examples of beings learning in different ways, rats for example always taste their foods in small bites before judging its toxicity, using then their previous experience. While this might seem a great way to judge unknown items, if spam systems were using the same logic, then newly written spams would not be labelled as spam, which requires a certain generalization, called inductive reasoning: the machine should be able to label a message by looking for keywords.

While nature might give us interesting ideas, it might also be quite stupid. Hungry pigeons were being fed at random times, and while a human could "detect" the randomness of this behaviour, pigeons based it on superstition, which made them reenact whatever action they were doing the first time they were fed in order to get food, and since all they did was reenact the same action, it might appear to them that whatever they were doing worked, which is not the case whatsoever.

Back to the rats: researchers discovered that rats' learning mechanism is quite complicated: if we make the rats get electrical shocks after getting food, they won't conclude a cause-effect relationship, they will not avoid the same food: rats can tell that tasting food cannot cause electrical shocks, and therefore avoiding this food would be futile.

This discrepancy introduces the concept of **inductive bias** : the incorporation of inductive bias that biases the learning mechanism.

## 2. When do we need machine learning ?

Since we defined multiple directions for our potential machine learning algorithms, when do we need them essentially ?

We usually need machine learning for tasks that are too complex to program, such as image understanding or pattern recognition, or tasks that are beyond our mere human capabilities, like the analysis of genomic data or weather prediction. In our day and age, we have access to an insurmountable amount of data that we can't fully analyze manually.

Another aspect of machine learning algorithm is adaptivity. Classic programs are rigid, hard-coded and cannot evolve by itself. However, machine learning algorithms offer solutions to these issues. An example of this is decoding handwritten text or spam detection programs.

## 3. Types of learning

We can describe four types of learning which paradigms can be classified :

- Supervised vs. unsupervised learning : a simple example would be taking the spam-detection problems. Feeding a spam-detection algorithm with a domain set of emails and labels (spam or not-spam) would be called a supervised learning, while feeding this algorithm with unlabelled emails and asking it to detect "anomalies" is called unsupervised learning.
- Active vs. passive learners : active learners interact with the environment by actively posing queries while a passive learner would only observe the information provided by the environment.
- Helpfulness of the teacher : the natural process of learning that everyone is familiar with usually involves a teacher trying to feed the learner with informations. But a teacher is not always helpful and might be adversarial, however such teachers could be quite helpful for the learner.
- Online vs. batch learning protocol : an example of this would be a stockbroker who needs to make daily decisions based on the experience collected so far. The outcome could be catastrophic day-by-day but we may hope that he would become an expert.

On the other side of the spectrum, a data-miner has access to a large amounts of data before outputting conclusions.

Now that we have describe the why of machine learning, let us lay the theoretical basis of our study.

## II – Laying the basis for our study

Let us consider this example : we have arrived to a Pacific island where papayas are a major component of everyday meals. We will describe papayas according to two criterias, color and softness : color ranging from dark green, orange, red to dark blue, and softness from rockhard to musky.

### 1. The statistical learning framework

It is time to establish a formal framework for our analysis before diving into the practical side of it.

#### Learners' input

- Domain set : an arbitrary domain set  $\chi$  of objects that we want to label, domain points will be represented by a vector of features, in our case, the couple (color, softness);
- Label set : for our example,  $Y$  is the set of possible labels, here  $\{0, 1\}$ , 1 meaning tasty and 0 meaning awful taste.
- Training data :  $S = ((x_1, y_1) \dots (x_n, y_n))$ , a finite sequence of pairs in  $\chi \times Y$ , basically a sequence of labeled domain points. It is a data the learner has access to, in our example, a set of papayas ((color, softness), tastiness). We might also call  $S$  training set or training example.

#### Learners' output

The learner is expected to output a prediction rule  $h: \chi \rightarrow Y$ .  $h$  is called a predictor/hypothesis. It is used to predict the label of new domain points. In our example, it is a rule that the learner will use to predict whether the future papayas will be tasty or not.  $A(S)$  will be used to denote the hypothesis  $h$  that a learning algorithm  $A$  returns upon receiving the training set  $S$ .

After laying down the basis for what is given and what is expected, it is time to explain how training data is generated.

We will assume that the instances (here papayas) are generated by some probability distribution, in our example, a “natural” probability distribution.

We will denote the probability distribution over  $\chi$  by  $\mathbf{D}$ , we assume the learner knows nothing about this distribution.

Let us also assume that there is a “factual” labeling function  $f: \mathcal{X} \rightarrow \mathcal{Y}$ , and that  $y_i = f(x_i) \forall i$ .

The learner doesn't know  $f$ , in fact, the learner is trying to output a predictor as perfect as  $f$ . The training set is generated by sampling  $x_i$  according to  $\mathbf{D}$  and labelling it by  $f$ .

### Measure of success

We define the error of a classifier/predictor the probability that the latter does not predict the correct label on a random data point. Basically, the error of  $h$  is the probability to draw on a random instance  $x$  over  $D$ , such as  $f(x) \neq h(x)$ .

Formally, let us denote  $A$  an event, we have :

$$\pi: \mathcal{X} \rightarrow \{0,1\}, A = \{x \in \mathcal{X}, \pi(x) = 1\}, \mathbb{P}_{x \sim D} \text{ to express } D(A)$$

We also define the error of a predictor  $h$  to be :

$$L_{D,f}(h) = \mathbb{P}_{x \sim D} [h(x) \neq f(x)] = D(\{x; h(x) \neq f(x)\})$$

Essentially, the error of  $h$  is the probability of randomly choosing  $x$  for which  $f(x) \neq h(x)$ .

$L_{D,f}(h)$  is called the **true error of  $h$** . We use the letter  $L$  since it represents **the loss of the learner**.

We remind that the learner does not know the distribution, the “factual” labeling function and neither the training set  $S$ .

## 2. ERM : Empirical Risk Minimization

Let us remind ourselves of the given data and the expected output : a learning algorithm receives a training set  $S$ , distributed over  $D$ , labeled by a “factual” labeling function  $f$ , to output a predictor  $h: \mathcal{X} \rightarrow \mathcal{Y}$ . The goal of this algorithm is to minimize  $L$  with respect to the unknown  $D$  or  $f$ .

Since both the distribution and the target function are unknown to the learner, the true error is not available to the learner, we define the training error :

$$L_S(h) = \frac{|\{i \in [m]: h(x_i) \neq y_i\}|}{m}$$



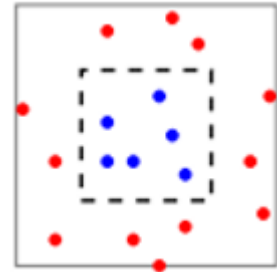
We will call it empirical error or empirical risk. Coming up with an  $h$  minimizing  $L_S(h)$  is called Empirical Risk Minimization (ERM).

#### a. How this model can fail : overfitting

Although the ERM rule seems natural, it may spectacularly fail.

Let us consider this distribution.

We assume the existence of a distribution  $D$  and a target function. In our example, we label points inside the dotted-square by 1, 0 otherwise. The surface of the inner square is 1, while the area of the bigger square is 2.



We consider :

$$h_s(x) = \begin{cases} y_i & \text{if } \exists i \in [m] \text{ s. t. } x_i = x \\ 0 & \text{otherwise} \end{cases}$$

No matter the sample,  $L_S(h_s) = 0$  which means that  $h_s$  is perfect. However, the true error  $L_{D,f}(h) = \frac{1}{2}$ , we have then a perfect predictor on the training set, but with a very poor performance on the true world : we call this overfitting, it happens when the predictor fits the data too well.

### 3. ERM with inductive bias

To avoid overfitting, the learner should choose a set of predictors, called hypothesis class  $\mathcal{H}$ . Each  $h \in \mathcal{H}$  is a function mapping  $\mathcal{X} \rightarrow \mathcal{Y}$ .

Formally, for a given class  $\mathcal{H}$ , a training set  $S$ , the  $ERM_{\mathcal{H}}$  learner uses the ERM rule to choose a predictor  $h \in \mathcal{H}$  to minimize the error over  $S$  :

$$ERM_{\mathcal{H}}(S) \in \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$$

$\operatorname{argmin}$  stands for the set of  $h$  in  $\mathcal{H}$  that minimize  $L_S(h)$  over  $\mathcal{H}$ . Since we bias the learner to chase  $h$  from  $\mathcal{H}$ , we will call this inductive bias. Since we determine the choice of such restriction before the learner sees  $S$ , it should be based on some prior knowledge about the problem.

#### a. Finite hypothesis classes

The simplest type of restriction on a class  $\mathcal{H}$  is to limit the number of predictors.

If  $\mathcal{H}$  is a finite class then  $ERM_{\mathcal{H}}$  won't overfit if it is based on a sufficiently large  $S$ .

### b. The realizability assumption

$\exists h^* \in \mathcal{H}$  s.t.  $L_{D,f}(h^*) = 0$ , which also assumes  $L_S(h^*) = 0$ .

This assumption implies that for every ERM hypothesis,  $L_S(h) = 0$ . However, we are interested in the true risk of  $h$ ,  $L_{D,f}(h)$ .

Since the algorithm only has access to  $S$ , the assumption in statistical machine learning is that  $S$  is generated by sampling points from  $D$  independently from each other.

Formally, the instances in the training set are **independently and identically distributed** (i.i.d.) according to  $D$ . Every  $x_i$  in  $S$  is sampled according to  $D$  and labeled according to  $f$ .

Basically,  $S$  is a window through which a learner gets a partial information about  $D$  and  $f$ . The larger  $S$  gets, the more likely it is to accurately reflect  $D$  or  $f$ .

### c. The confidence parameter

Since  $L_{D,f}(h)$  depends on  $S$  that is randomly picked, there is randomness in the choice of  $h$ , so is  $L_{D,f}(h)$ : it is not realistic to expect  $S$  to suffice to direct the learner to a good predictor, since there is a probability that  $S$  is not actually representative of the distribution  $D$ . In our example, there is a small chance that all papayas we tasted were awful, although 70% of papayas in the island are tasty, therefore  $ERM_{\mathcal{H}}(S)$  may be the constant function that labels every papaya as not tasty and has a 70% error on the true distribution.

We address the probability of getting a non-representative sample  $\delta$ , and we call  $(1 - \delta)$  **the confidence parameter**.

### d. The accuracy parameter

Since we cannot guarantee a perfect label prediction, we introduce  $\varepsilon$ , **the accuracy parameter**.

The event  $L_{D,f}(h) > \varepsilon$  is a **failure of the learner**, while  $L_{D,f}(h) \leq \varepsilon$  is seen as an **approximately correct predictor**.

## III – A formal learning model

### 1. PAC learning

We have shown that for a finite  $\mathcal{H}$ , if  $ERM_{\mathcal{H}}$  is applied on a sufficiently large  $S$  then  $h$  will be **Probably Approximately Correct**, we now define PAC learning.

### 2. PAC learnability

A hypothesis class  $\mathcal{H}$  is PAC learnable if there exist a function  $m_{\mathcal{H}}: [0,1]^2 \rightarrow \mathbb{N}$  and a learning algorithm with a following property : for every  $\varepsilon, \delta \in [0,1]$ , for every distribution  $D$  over  $\mathcal{X}$ , for every labeling function  $f: \mathcal{X} \rightarrow \{0,1\}$ , if the realizable assumption holds with respect to  $D, \mathcal{H}, f$ , then when running the algorithm on  $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$  i.i.d. examples generated by  $D$  and labeled by  $f$ , the algorithm returns a hypothesis  $h$  such that, with the probability of at least  $(1 - \delta)$ :

$$L_{D,f}(h) \leq \varepsilon$$

The definition of PAC learnability contains two approximation parameters : the accuracy parameter determines how far the output classifier can be from the optimal one, and a confidence parameter indicating how likely the classifier is to meet that accuracy requirement.

### 3. Sample complexity

The function  $m_{\mathcal{H}}: [0,1]^2 \rightarrow \mathbb{N}$  determines the sample complexity of learning  $\mathcal{H}$ , basically, how many examples are required to guarantee a PAC solution.

Every finite hypothesis class is PAC learnable with sample complexity such that :

$$m_{\mathcal{H}}(\varepsilon, \delta) \leq \left\lceil \frac{\log \left( \frac{|\mathcal{H}|}{\delta} \right)}{\varepsilon} \right\rceil$$

#### 4. A more general learning model

We can generalize the described model in two aspects :

- Removing the realizability assumption
- Considering learning problems beyond binary classification (predicting real valued numbers or a label picked from a finite set of labels).

Our analysis can be extended to such scenarios by allowing a variety of loss functions.

##### a. Agnostic PAC learning

In many practical problems, this assumption does not hold. It is also not realistic to assume that the labels are fully determined by the features we measure on input elements. In our example, it is safe to assume that in the real world, papayas of same color/hardness may have a different taste.

Therefore, we relax the realizability assumption with a **data-labels generating distribution**.

$D$  is now a **joint-distribution** over domain points and labels, and is composed of two parts :

- $D_x$  : a distribution over unlabeled points (called **marginal distribution**)
- $D(x, y/x)$  : a **conditional probability** over labels for each domain points

In our example,  $D_x$  determines the probability of encountering a papaya whose color and hardness fall in some color-hardness values domain, and  $D(x, y/x)$  the probability that a papaya with a color/hardness represented by the vector  $x$  is tasty.

Therefore, this model allows two papayas of same color-hardness to taste differently.

### b. The Bayes Optimal Predictor

Given any probability distribution  $D$  over  $\mathcal{X} \times \{0,1\}$ , the best label predicting from  $\mathcal{X}$  to  $\{0,1\}$  will be :

$$f_D(x) = \begin{cases} 1 & \text{if } \mathbb{P}[y = 1/x] \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

Since the learner does not have access to the distribution  $D$ , we cannot use the Bayes Optimal Predictor (B.O.P.), the learner only has access to the training sample.

We also cannot expect a learning algorithm to find a predictor as good as the B.O.P., we will instead require the learning algorithm to find a predictor whose error is not much larger than the best possible error of a predictor in some given hypothesis class.

### c. Agnostic PAC learnability

A hypothesis class  $\mathcal{H}$  is **agnostic PAC learnable** if there exist a function  $m_{\mathcal{H}}: [0,1]^2 \rightarrow \mathbb{N}$  and a learning algorithm with the following property : for every  $\varepsilon, \delta \in [0,1]$ , for every distribution  $D$  over  $\mathcal{X} \times Y$ , when running the algorithm on  $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$  i.i.d. examples generated by  $D$ , the algorithm returns a hypothesis  $h$  such that, with the probability of at least  $(1 - \delta)$ :

$$L_{D,f}(h) \leq \min_{h' \in \mathcal{H}} L_D(h') + \varepsilon$$

Under this definition, a learner can still declare success if its error is not much larger than the best error achievable by a predictor from  $\mathcal{H}$ .

### d. The scope of learning problems modeled

Let's consider some examples of different learning tasks :

- Multiclass classification : not all classifications are binary. Let's say we wish to design a program that classes articles or documents according to topics such as sports, politics, science. A learning algorithm will have access to examples of correctly classified documents and output a program that takes as inputs a document, and outputs its topic classification. The domain set would be represented by features such as keywords or the origin, the label set the set of possible topics, our training set a finite sequence of (features, label).
- Regression : as an example, we would like to predict a newborn weight by checking 3 basic ultrasound measures The domain set is a subset of  $\mathbb{R}^3$ , the label set a set of real

numbers (weight in grams). In this case, we might evaluate the quality of  $h$  by the expected square difference.

$$L_D(h) = \mathbb{E}_{(x,y) \sim D}(h(x) - y)^2$$

#### e. Generalized loss function

We generalize our formalism of the measure of success as follows :

Given any set  $\mathcal{H}$  and some domain  $\mathcal{Z}$  (here,  $\mathcal{X} \times Y$ ), we define  $l: \mathcal{H} \times \mathcal{Z} \rightarrow \mathbb{R}^+$ , we call such functions **loss functions**.

- The risk function

We define the risk function to be the expected loss of a classifier  $h \in \mathcal{H}$ , with respect to a distribution  $D$  over a domain  $\mathcal{Z}$ .

$$L_D(h) = \mathbb{E}_{z \sim D}[l(h, z)]$$

- The empirical risk

We define the empirical risk to be the expected loss over a given sample  $S = (z_1, \dots, z_m) \in \mathcal{Z}^m$ .

$$L_S(h) = \frac{1}{m} \sum_{i=1}^m l(h, z_i)$$

We therefore define loss functions used in the preceding examples as follows :

- 0-1 loss : used in binary or multiclass classification problems.

$$l_{0-1}(h, (x, y)) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$

- Square loss : used in regression problems.

$$l_{sq}(h, (x, y)) = (h(x) - y)^2$$

We revisit our previous definition of PAC learnability as follows :

f. **Agnostic PAC learnability for general loss functions**

A hypothesis class  $\mathcal{H}$  is **agnostic PAC learnable with respect to** as et  $\mathcal{Z}$  and a loss function  $l: \mathcal{H} \times \mathcal{Z}$  if there exist a function  $m_{\mathcal{H}}: [0,1]^2 \rightarrow \mathbb{N}$  and a learnign algorithm with the following property : for every  $\varepsilon, \delta \in [0,1]$ , for every distribution  $D$  over  $\mathcal{Z}$ , when running the algorithm on  $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$  i.i.d. examples generated by  $D$ , the algorithm returns a hypothesis  $h$  such that, with the probability of at least  $(1 - \delta)$ :

$$L_{D,f}(h) \leq \min_{h' \in \mathcal{H}} L_D(h') + \varepsilon$$

## IV – Linear predictors

The family of linear predictors is one of the most successful families of hypothesis classes, they are intuitive, easy to interpret and fit the data reasonably well in many natural learning problems.

We define the class of affine functions as :

$$L_d = \{h_{w,b} : w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

Where :

$$h_{w,b} = \langle w, x \rangle + b = \left( \sum_{i=1}^d w_i x_i \right) + b$$

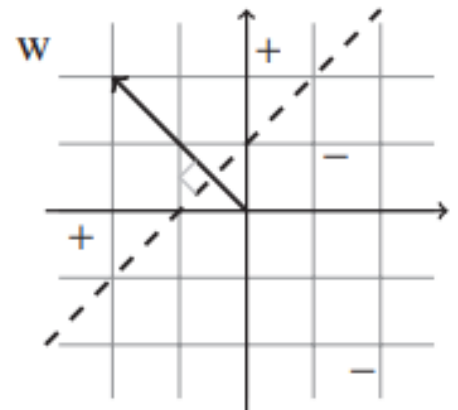
The different hypothesis classes of a function  $\phi: \mathbb{R} \rightarrow Y$  on  $L_d$ . In binary classification, we can choose  $\phi$  to be the sign function. We will add the bias  $b$  into  $w$  for simplifying measures.

### 1. Halfspaces

We consider the class of halfspaces, designed for binary classification problems, namely  $\mathcal{X} = \mathbb{R}^d, Y = \{-1, +1\}$ . The class of halfspaces is defined as follows :

$$HS_d = \text{sign} \circ L_d = \{x \rightarrow \text{sign}(h_{w,b}(x)) : h_{w,b} \in L_d\}$$

Each halfspace hypothesis in  $HS_d$  is parameterized by  $w \in \mathbb{R}^d$  and  $b \in \mathbb{R}$ , and upon receiving a vector  $x$ , the hypothesis returns the label  $\text{sign}(h_{w,b}(x))$ .



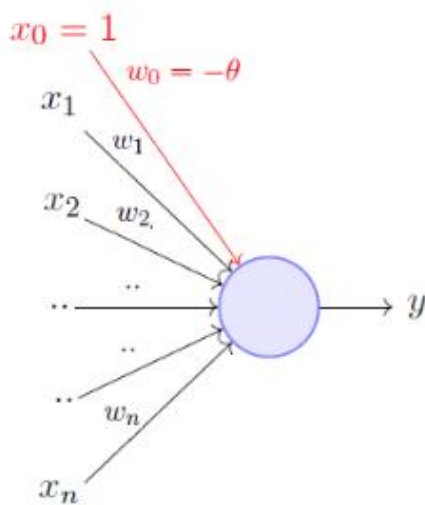


## 2. Perceptron for halfspaces

A perceptron is a linear binary classifier, which decides whether or not an input belongs to a specific class or not.

### a. How it works

In simple words, a perceptron takes an input, aggregates it and returns 1 only if the aggregated sum (that has used randomly generated weights) is more than some threshold, otherwise returns 0.



A more accepted convention,

$$y = 1 \quad \text{if} \quad \sum_{i=0}^n w_i * x_i \geq 0$$

$$= 0 \quad \text{if} \quad \sum_{i=0}^n w_i * x_i < 0$$

where,  $x_0 = 1$  and  $w_0 = -\theta$

### b. The algorithm

#### Batch Perceptron

**input:** A training set  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$

**initialize:**  $\mathbf{w}^{(1)} = (0, \dots, 0)$

**for**  $t = 1, 2, \dots$

**if**  $(\exists i \text{ s.t. } y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0)$  then

$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$

**else**

**output**  $\mathbf{w}^{(t)}$

Figure 1 : Perceptron algorithm

### c. Perceptron in Java

For a more practical understanding of the algorithm, we propose to elaborate a program that can determine whether or not a papaya is tasty or not.

Let us assume some hypothesis first :

- To simplify our study, a color or a hardness of a papaya can range from 0 to 5.
- We assume that a papaya is tasty as long as the sum of both the features is superior or equal to 5, formally :

$$f(x) = \begin{cases} 1 & \text{if } color + hardness \geq 5 \\ 0 & \text{otherwise} \end{cases}$$

#### Code structure

In order to show limitations described in precedent paragraphs, we will first write a function to generate a domain set  $\chi$

```
public double[][] DomainSetGenerator() // Génère un training set donné
{
    System.out.println("Veuillez saisir la taille du training set : ");
    Scanner M = new Scanner(System.in);
    int c = M.nextInt();
    double[][] DomainSet = new double[c][2];
    for(int i=0;i<c;i++)
    {
        Random r = new Random();
        Random r2 = new Random();
        double Color = 5*r.nextDouble();
        double Hardness = 5*r2.nextDouble();
        DomainSet[i][0]=Color;
        DomainSet[i][1]=Hardness;
    }
    return DomainSet;
}
```

Figure 2 : Random domain set generator

We also write the factual labelling function  $f$  as follows :

```
public int[] LabelSet(double[][] DomainSet) // Détermine la sortie du training set
{
    int b = DomainSet.length;
    int[] LabelSet = new int [b];
    for (int j=0;j<b;j++)
    {
        if (DomainSet[j][0]+DomainSet[j][1]>=5)
            LabelSet[j]=1;
        else
            LabelSet[j]=0;
    }
    return LabelSet;
}
```

We now have a training data split between 2 arrays which the algorithm can use to predict the tastiness of a given set of papayas.

Afterwards, we define a training function that takes as parameters the generated domain set, its label set, a threshold (equals 5 in our example), a learning rate, and the number of epochs (how many times the algorithm trains itself).

```
public void Train(double[][] DomainSet, int[] LabelSet, double threshold, double lrate, int epoch)
{
    this.threshold = threshold;
    int n = DomainSet[0].length; // Nombre de features des papayas
    int p = LabelSet.length; // Nombre de papayas du training set
    weights = new double[n];
    Random r = new Random(); // Génère un nombre aléatoire entre 0 et 1
    for(int i=0;i<n;i++) // On initialise les poids de façon aléatoire
    {
        weights[i] = r.nextDouble();
    }
    for(int i=0;i<epoch;i++) // On "entraîne" l'algorithme -epoch- fois
    {
        int totalError = 0;
        for(int j =0;j<p;j++)
        {
            int predictedTastiness = PredictedTastiness(DomainSet[j]);
            int error = LabelSet[j] - predictedTastiness;

            totalError +=error;

            for(int k = 0;k<n;k++)
            {
                double delta = lrate * DomainSet[j][k] * error;
                weights[k] += delta;
            }
        }
        if(totalError == 0)
            break;
    }
}
```

Figure 3: Perceptron code in Java

While training against the domain set, the algorithm is weighting the color-hardness that will be used afterwards when given an “anonymous” papaya.

We finally define the predictor as follows :

```
public int PredictedTastiness(double[] UnknownPapayaFeatures)
{
    double predictor = 0.0;
    for(int i=0;i<UnknownPapayaFeatures.length;i++)
    {
        predictor += weights[i]*UnknownPapayaFeatures[i];
    }
    if(predictor>threshold)
        return 1;
    else
        return 0;
}
```

### Execution

So let's use as our example multiple papayas, we remind that a papaya is labeled tasty if the sum of its features is superior or equal to 5.

```
public static void main(String[] args)
{
    Perceptron p = new Perceptron();
    double[][] DomainSet = p.DomainSetGenerator();
    int[] LabelSet = p.LabelSet(DomainSet);
    p.Train(DomainSet, LabelSet, 5, 0.1, 100);
    System.out.println(p.PredictedTastiness(new double[]{3,1})); // This papaya is not tasty
    System.out.println(p.PredictedTastiness(new double[]{2,1})); // This papaya is not tasty
    System.out.println(p.PredictedTastiness(new double[]{2,3})); // This papaya is tasty
    System.out.println(p.PredictedTastiness(new double[]{2,2.5})); // This papaya is not tasty
}
```

As we've shown before, the bigger the training set is, the closer to the “factual” labeling function the predictor becomes, we can illustrate that by supposing a training set of 50 papayas, we get the following result :

```
Veillez saisir la taille du training set :
50
1
0
1
1
BUILD SUCCESSFUL (total time: 1 second)
```

The algorithm mislabeled 2 papayas as being tasty while they were not.

If we suppose now a training set of correctly labelled 100,000 papayas, we get the following result :

```
Veillez saisir la taille du training set :  
100000  
0  
0  
1  
0  
BUILD SUCCESSFUL (total time: 3 seconds)
```

All 4 papayas were correctly labelled.

We can try labelling a papaya which sum of features is equal to 4.9, the more the algorithm trains against itself while providing it a bigger training set, the more accurate it's predictor becomes.

```
Veillez saisir la taille du training set :  
1000000  
0  
BUILD SUCCESSFUL (total time: 2 seconds)  
|
```

### Limitations

The main limitation of a perceptron is that it can output 2 values (0 or 1), it can also only classify linearly separable sets of vectors. Basically, if a straight line can be draw to separate the input vectors, then they are linearly separable. If that is not the case, then learning will never reach a point where all vectors are classified properly. This problem can however be solved by making networks using more than one perceptron to solve more difficult problems.

## V – Support-Vector Machine and k-Nearest Neighbors

### 1. Support-Vector Machine (SVM)

SVMs are a supervised learning algorithm models that analyze data used for classification and regression analysis. Such models are particularly powerful when dealing with a huge number of features.

The main objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N – the number of features) that distinctly classifies the data point.

**Soft-SVM**

**input:**  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$

**parameter:**  $\lambda > 0$

**solve:**

$$\min_{\mathbf{w}, b, \xi} \left( \lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \xi_i \right) \tag{15.4}$$

s.t.  $\forall i, y_i((\mathbf{w}, \mathbf{x}_i) + b) \geq 1 - \xi_i$  and  $\xi_i \geq 0$

**output:**  $\mathbf{w}, b$

Figure 4: Soft-SVM algorithm

Two types of SVM are

mostly used : soft and hard SVMs. The hard-SVM formulation assumes that the training set is linearly separable which is a rather strong assumption, unlike the soft-SVM which can be applied even if the training set is not linearly separable.

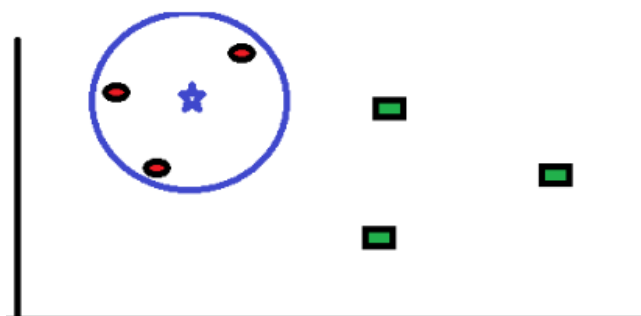
### 2. k-Nearest Neighbours (KNN)

k-NN algorithms are non-parametric methods used for classification and regression analysis, but mostly used for classification problems.

In simple words, the k-NN algorithm works as follows :

- We wish to label the blue star below (either a Red Circle or a Green Square)
- The “k” in k-NN stands for the number of nearest neighbours we wish to get the information from. In our case if

$k = 3$ . Since all the neighbours in the circle are Red Circles, we can assume that the unlabelled blue star is a Red Circle.



Getting the optimal value of  $k$ , we need to segregate the training and validation from the initial dataset : it is therefore highly data-dependent. In general, a larger  $k$  suppresses the effects of noise, but makes the classification boundaries less distinct.

The  $k$ -NN pseudo-code is as follows :

```
kNN (dataset, sample)
{
  1. Go through each item in my dataset, and calculate the "distance"
  from that data item to my specific sample.
  2. Classify the sample as the majority class between K samples in
  the dataset having minimum distance to the sample.
}
```

Figure 5:  $k$ -NN pseudo-code

The word distance refers to the Euclidian distance :

$$\rho(x, x') = \|x - x'\| = \sqrt{\sum_{i=1}^d (x_i - x'_i)^2}$$

## VIII – Bibliography / Webography

Shalev-Shwartz, S. and Ben-David, S. (2017). *Understanding machine learning*. Cambridge: Cambridge University Press.

DasGupta, A. (2011). *Probability for statistics and machine learning*. New York: Springer.

Mathur, P. (2018). *Machine Learning Applications Using Python*. Berkeley, CA: Apress L. P.

NBA.com. (2019). *Kia Race To The MVP Ladder | NBA.com*. [online] Available at: <http://www.nba.com/mvp-ladder>

Honors, A. and Winners, M. (2019). *NBA MVP & ABA Most Valuable Player Award Winners | Basketball-Reference.com*. [online] Basketball-Reference.com. Available at: <https://www.basketball-reference.com/awards/mvp.html>

Scikit-learn.org. (2019). *Documentation scikit-learn: machine learning in Python — scikit-learn 0.20.2 documentation*. [online] Available at: <https://scikit-learn.org/stable/documentation.html>



## IX – Table of figures

Figure 1 : Perceptron algorithm .....	16
Figure 2 : Random domain set generator.....	17
Figure 3: Perceptron code in Java .....	18
Figure 4: Soft-SVM algorithm .....	21
Figure 5: k-NN pseudo-code .....	22
Figure 6 : Histogram of historical vote share .....	<b>Error! Bookmark not defined.</b>