# Lecture 1

# 19th of September 2018

A. Frangioni

This course will deal with the optimization and numerical analysis of machine learning problems. We are not going to discuss difficult problems (e.g. NP-hard problems), besides we will present an efficient solution for simple ones (often **convex** ones), since we are dealing with huge amount of data.

Let us start with a warm up on machine learning problems.

## 1.1 Introduction to machine learning problems

Machine learning techniques are not as "young" as it might seem, the intuition has been there for ages, but we did not have enough calculus power. Machine learning algorithms have started working well recently, thanks to the many improvements in computer performances; for this reason, it is becoming a more and more popular subject to study.

The main idea behind machine learning is to take a huge amount of data (e.g. frames of a video for object-recognition) and squeeze them, in order to process them. This intuitive concept is translated into mathematical terms as "building a **model**" that fits our data. As in practical engineering problems, people want to construct a model (a small sized representation of the large thing we want to produce in the end) and try to understand its behaviour, before actually build such an object. Take as an example the problem of designing a jet. It is not clever to start building the plane before designing a cheap prototype to better study its behaviour in the atmosphere.

The kind of models we want to build are cheap to construct and as close as possible to the real problem we are studying. In physics, people try to find the best mathematical model to describe a real world phenomenon. The main issue is computation, since the more accurate the model, the more costly the prediction phase. Hence, a good model is a trade-off between accuracy and simplicity, namely it provides good prediction without incurring in slow computations.

The model, though, has to be parametric: we do not have only one model, we have a "shape" of a model, which is fit to our problem through the tuning of some parameters.

**Example 1.1.1.** *As an example, we are given three couples:* $f(x_1) = y_1$, $f(x_2) = y_2$, $f(x_3) = y_3$, *as shown in Figure 1.1.*

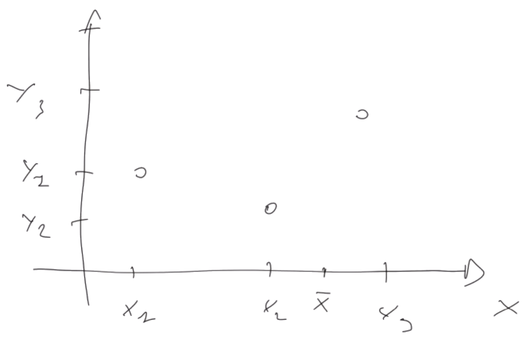

FIGURE 1.1: Geometric representation of the input. We are interested in finding a model that fits the input data and allows to predict $\bar{y}$ out of $\bar{x}$.

*We need to make some choices: first, we need to decide the kind of model we believe is a good approximation of the objective function, say a linear model* $f(x) = ax + b$. *After doing that, we are left with choosing its parameters (in order to pick a line among the whole family of linear functions), namely* $a$ *and* $b$.

The aim is to build a model that fits the data we are given and then tune the parameters in order to achieve a good accuracy for a given application (recall that the model is parametric, hence the right values for the parameters have to be learned).

Another important characteristic of a good model is that it should not take too long to be built.

In this course we do not concentrate on the problem of finding the model that best fits our data, but we are already given a problem and a model and we only study its behaviour through its parameters. In other words, within the family of models with a given outline, we want to find the one that better represents the phenomena observed. This is called **fitting** and it is clearly some sort of optimization problem, where the fitting task is typically the computational bottleneck.
However, machine learning is more than simply fitting: fitting minimizes training error (or empirical risk), but ML aims at minimizing test error (i.e. generalization error).

A machine learning solution first builds a model that fits the observed data and then performs an hyper-parameter tuning in order to achieve a good "predicting power" on unseen inputs. In machine learning, a model that fits the

data while achieving good performances on new examples is said to be "non **overfitting**".
For machine learning purposes, a mathematical model should be:

- *accurate* (describing well the process under consideration)
- *computationally inexpensive* (providing answers rapidly)
- *general* (it can be applied to many different processes)

It goes without saying that achieving all these goals is practically impossible.

## 1.2 Optimization

In the rest of this lecture we are going to better understand what an optimization problem is, through some intuitive real world examples.

**Terminology**

In this course we will refer to vectors using the bold notation $\mathbf{v} \in \mathbb{R}^n$. The $i$-th entry of vector $\mathbf{v}$ is denoted as $v_i \in \mathbb{R}$, while the $j$-th pattern in the training set is a couple of vectors $(\mathbf{x^j}, \mathbf{y^j})$.
For more details on vectors and vector spaces see **??**.

### 1.2.1 Linear estimation

Let us consider a phenomenon measured by one number $y \in \mathbb{R}$ that is believed to depend on a vector $\mathbf{x} = [x_1, \dots, x_n]$. We are provided a set of observations: $\{(y^1, \mathbf{x^1}), \dots, (y^m, \mathbf{x^m})\}$.

**Definition 1.2.1** (Linear model)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be the objective function. We call $\widetilde{f}(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + w_0 = \mathbf{w}_+{}^T \mathbf{x} + w_0$ the **linear model** of $f$ for a given set of parameters, which is a vector $\mathbf{w} = (w_0, \mathbf{w}_+) = (w_0, w_1, \dots, w_n) \in \mathbb{R}^{n+1}$.*

How can we evaluate the "similarity" between our model and the objective function? Through computing the "error" or difference between the objective function value and the model prediction on each input. Under this assumption, the error function may be used to find the best parameters for our model, through solving a minimum problem:

**Definition 1.2.2** (Least squares problem)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be the objective function, such that $f(\mathbf{x}) = \mathbf{y}$ and let $X\mathbf{w}$ be our linear model. Then we can find the best values for vector $\mathbf{w} \in \mathbb{R}^{n+1}$ by solving*

$$\min_{\mathbf{w}} \|\mathbf{y} - X\mathbf{w}\|$$

$$\textit{where } Y = \begin{pmatrix} \mathbf{y^1} \\ \vdots \\ \mathbf{y^m} \end{pmatrix} \textit{ and } X = \begin{pmatrix} 1 & \mathbf{x^1} \\ \vdots & \vdots \\ 1 & \mathbf{x^m} \end{pmatrix}.$$

If the matrix $X$ is invertible then the simple solution is $\mathbf{w} = X^{-1}\mathbf{y}$. The point is that this operation is very costly when dealing with a huge number of entries (in the next paragraph we will see a way to manage it).

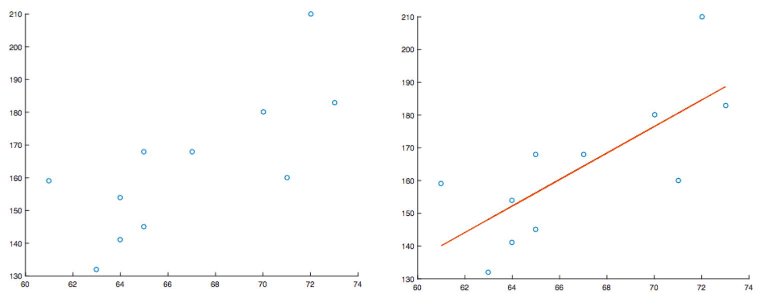

FIGURE 1.2: A linear estimation fitting example

### 1.2.2 Low-rank approximation

A (large, sparse) matrix $M \in M(n, m, \mathbb{R})$ describes a phenomenon depending on pairs (e.g., objects bought by customers) and we may want to approximate that matrix as the product between two smaller matrices (find a few features that describe most of users' choices): a "tall thin" matrix $A \in M(n, k, \mathbb{R})$ and a "fat large" $B \in M(k, m, \mathbb{R})$ (where $k \ll n, m$).

$$\boxed{M} \approx \boxed{A} \cdot \boxed{B}$$

This problem can be translated into a numerical analysis problem of the following shape

$$\min_{A,B} \|M - AB\|$$

The matrices $A$ and $B$ can be obtained from eigenvectors of $M^TM$ and $MM^T$, but that's a huge, possibly dense matrix. So a more efficient way should be used, which also avoids the explicit formation of $M^TM$ and $MM^T$ because that would need a lot of memory.

Efficiently solving this problem requires:

- low-complexity computation
- avoiding the explicit forming of $M^TM$ and $MM^T$
- exploiting the structure of $M$ (sparsity, similar columns, ...)
- ensuring that the solution is *numerically stable*

### 1.2.3 Support vector machines

Let us consider the so-called "decision problem": given a set of values of many parameters (variables) "label" a person $i$ as ill or healthy, $y^i \in \{1, -1\}$.

The geometric intuition in two dimensions is given by Figure 1.3. We would like to find the line that better splits the plane into two regions, because this could help to diagnose the next patient. The rationale here is to maximize the space between the line and the nearest points (called **margin**), in order to have a better accuracy.

The distance between the two hyperplanes $(w_0, \mathbf{w}_+)$ and $(w_0', \mathbf{w}_+)$ in Figure 1.3 is computed as $\frac{2}{\|\mathbf{w}_+\|}$, so in order to maximize the distance between the planes we aim at minimizing $\|\mathbf{w}_+\|$. For each hyperplane that lies between $(w_0, \mathbf{w}_+)$ and $(w_0', \mathbf{w}_+)$ the following holds

$$\begin{cases} \mathbf{w}_+\mathbf{x}^i + w_0 \geq 1 & \text{if } \ y^i = 1 \\ \mathbf{w}_+\mathbf{x}^i + w_0 \leq -1 & \text{if } \ y^i = -1 \end{cases}$$

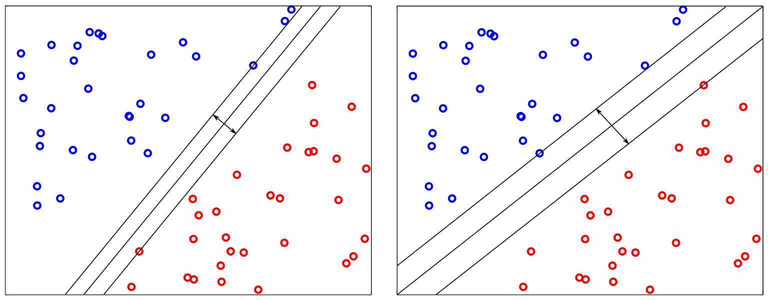

FIGURE 1.3: There are many possible boundaries that can be chosen as a model using many angular coefficients. Our best guess is the one that maximizes the distance between the line and the nearest points.

Under the hypotheses of this optimization problem, the maximum-margin separating hyperplane (assuming that any exists) is the solution of

$$\min_{\mathbf{w}} \{\|\mathbf{w}_+\|^2 \ : \ y^i(\mathbf{w}_+\mathbf{x^i} + w_0) \geq 1, \ i = 1, \ldots, m\}$$

In practice, most of the times there is no such line. To overcome this issue we introduce the concept of "penalty" that accounts for the number of points that are misclassified, through the following

**Definition 1.2.3** (SVM Primal problem)**.** *We term Support Vector Machine primal problem (SVM-P) a convex constrained problem with complex constraints that is formalized as*

$$\begin{cases} \min\limits_{\mathbf{w},\xi} \|\mathbf{w}_+\|^2 + C \sum\limits_{i=1}^{m} \xi_i \\ y^i(\mathbf{w}_+\mathbf{x}^i + w_0) \geq 1 - \xi_i \, , \, \xi_i \geq 0 \, , \, \forall i = 1, \ldots, m \end{cases} \tag{SVM-P}$$

*Where $C$ is an **hyper-parameter** that weights the violation of the separating margin.*

This definition formalizes the intuition that the approximated function may have a greater norm and lead to a very small misclassification error, or it could be the other way round. Both these solutions are acceptable and their performances depend only on the problem.

Whenever we are able to solve a multi-objective optimization problem we are also able to solve what is called the **dual problem**, which in our case has the following shape.

**Definition 1.2.4** (SVM Dual problem)**.** *We call Support Vector Machine dual problem (SVM-D) a convex constrained quadratic problem defined as*

$$\begin{cases} \max\limits_{\alpha} \sum\limits_{i=1}^{m} \alpha_i - \frac{1}{2} \sum\limits_{i=1}^{m} \sum_{j=1}^{m} \alpha_i < \mathbf{x^i}, \mathbf{x^j} > \alpha_j \\ \sum\limits_{i=1}^{m} y^i \alpha_i = 0 \\ 0 \leq \alpha_i \leq C \,, \forall i = 1, \dots, m \end{cases} \tag{SVM-D}$$

The idea behind the theory of duality is to obtain the solution of a problem by solving an (apparently) different one.

**Lemma 1.2.1.** *Given an optimal solution $\alpha^*$ for the dual problem (SVM-D), then $\mathbf{w}^*{}_+ = \sum_{i=1}^{m} \alpha_i^* y^i \mathbf{x^i}$ is optimal for the primal problem (SVM-P).*

A reader who has a deeper background in the field of machine learning knows that the scalar product in the dual form allows the usage of the so-called "kernel trick". This mathematical transformation allows the mapping of the input space into a larger feature space where the points are more likely to be linearly separable.

Theoretically, the feature space can be infinite-dimensional, provided that the scalar product can be (efficiently) computed.

This whole course has the aim of presenting some techniques for solving efficiently **convex quadratic problems**, as the ones presented above.