MACHINE LEARNING FOR VISION AND MULTIMEDIA

Lecture notes

Carlo Migliaccio

AA 2024/2025

Contents

Chapter 1

Introduction

Among the definitions one gives of **Machine learning** we can say that it is a *"Field of study that gives computers the ability to learn without being explicitly programmed"*. Nowadays, the *Artificial intelligence* is in general that the electricity was in the 19th century. Something of paramount importance!
There are several methodologies and subfields in Machine Learning and the distinction is based on *how much and how* the human collaborate and of the type of provided data. The most important classification is the one between:

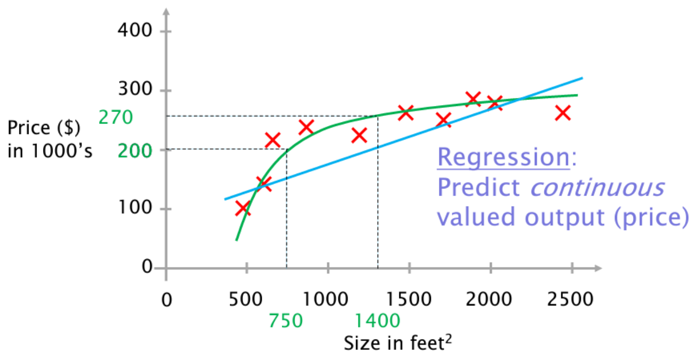
- *Supervised learning* (this course), is the approach which uses **a-priori knowledge** embedded in the data that are used for training algorithms and recognize patterns;
- *Unsupervided learning*, is the approach at the opposite whose main feature is not using *labeled data* for assess the tasks.
- *Other approaches.* Due to its vastness, in machine learning you can find for sure other subfields. For example the *Reinforcement learning, Semi-Supervised learning, trasnfer learning.* However they are all outside the purposes of this course.

1.1 Supervised learning

From Wikipedia (EN): Supervised learning (SL) is a paradigm in machine learning where input objects (for example, a vector of predictor variables) and a desired output value (also known as a human-labeled supervisory signal) train a model. In the following we are giving some simple introductory examples about two among the most used techniques in supervised learning.

1.1.1 Linear Regression

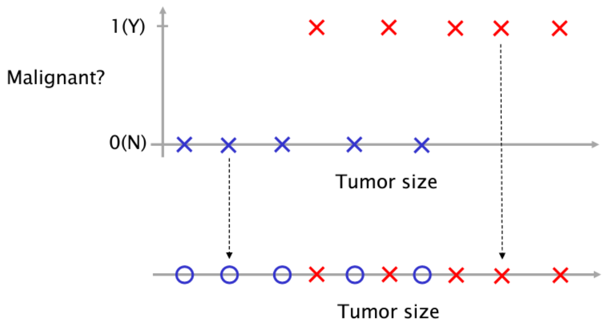
Let us imagine we are supposed to create a model that allows us to **predict the price of an house**. For sake of simplicity and clarity, suppose that our *dataset entries* have one feature (the house size in ft^2) and the *price* which represents the **correct answers**. Using this data we seek for a model which could predict, given the size of an unknown house, his price (in dollars, \$). Several choices can be made. At first, using either a linear or a nonlinear model and so on. It is remarkable that – even in such a simple example – we are facing a **supervised** problem since the right answers are given! This particular case is a problem of **regression** since we want to **predict a continuous valued output**, in our case the price.



In the figure above is shown the example in which two different models are used, clearly the predicted values for an unknown record is different according to the chosen model.

1.1.2 Classification

On the other hand, when we want to predict a discrete value (eg. YES/NO), we have a **classification** problem. Again, let us consider a trivial example: we want to predict whether a tumor is malignant or not according to its size. Even in this case we have one feature for the data (*tumor size*) and all the training data are labeled with the YES/NO answer.



The figure shows a graphical representation of the dataset. In this case since the answers are associated with different symbols, a more compact representation is given by a one-axis diagram: one feature is given, furthermore a different symbol is associated to different classes. Note that in this case we are in front of a a **binary classification problem**, in general the classes to predict are not necessarily in number of two.
This was just an example to understand and introduce the problem, but in real-world applications, one feature is not sufficient to build a good model! For sake of clarity, let us complicate

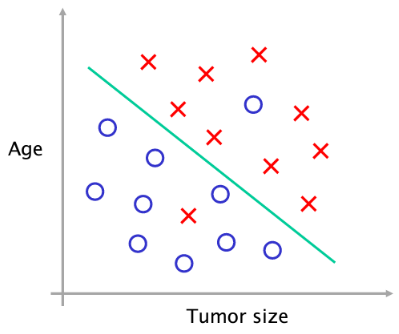


Figure 1.1: Bi-variate problem with linear decision boundary

a little bit the example we have just presented by adding a new feature associated with the *age of the patient*.

In this case the data set is represented in a 2D graph, one axis for each feature and a different symbol for each class. Now, given a record associated with a new patient, what is the class for its tumor? In this case can be useful to individuate a **decision boundary** according to which one can decide clearly what is the prediction (Positive/Negative). In the two parts there are some outliers, for this reason one can be tempted to build a more "accurate" decision boundary that perfectly split the two classes.

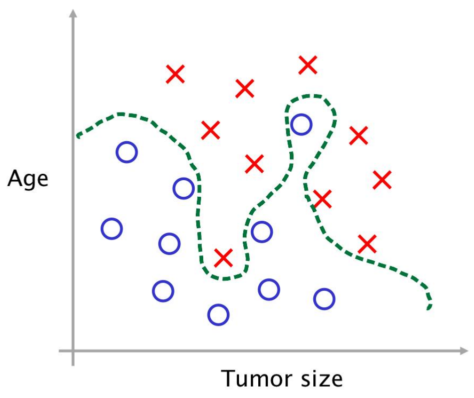


Figure 1.2: Example of overfitting

Is this a good model for the given problem? NO! This model will have very bad *performances of generalization* with new records to be classified, since it is too much related to the given dataset. In a colloquial way we say that: The model has learnt the by heart the dataset. A problem known as **overfitting**.
Finally, we can say that few features will result in a bad model, on the other hand also too much features will result in a bad model for another problem known as the **curse of dimensionality**.[1]

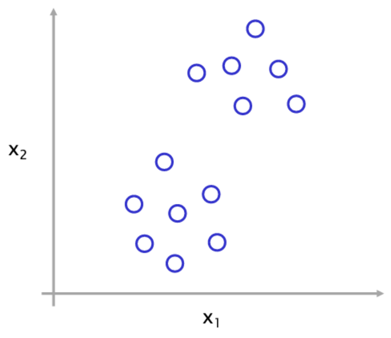
1.2 Unsupervided learning

At the opposite of the *supervised approach*, here patterns are learnt exclusively from unlabeled data. The most common example of such an approach is the **Clustering**.

[1]In these case techniques of dimensionality reduction has to be employed.

1.2.1 Clustering

In this case several algorithms are employed to discover groups called **clusters** associated with objects which are similar in some sense. In general, very often distance-based measures are used to individuate the groups. One of the most famous clustering algorithm is the *K-Mean*. The following figure is an example of bivariate clustered data.



Unsupervised techniques are used also in bioinformatics in manipulated *DNA microarrays*, for gropuing together similar web pages, for analysis of astronomical data and so on.

Chapter 2

Model, cost, parameter learning, Gradient Descent

Let us come back to the first example of *price prediction* and formalize some aspects we have only mentioned. The objective here is to exploit this example to introduce and better clarify several concepts.

2.1 Model representation

At first, the training set we are using is something similar to the following:

Size in feet^2(x)	**Price(\$) in 1000's(y)**
2104	460
1416	232
1534	315
852	178
...	...

We will indicate with m the number of samples of the training set (number of rows), x is the input (possibly multivariate) variable, y is the output variable, (x, y) indicates generically a sample from the training set, while $(x^{(i)}, y^{(i)})$ indicates the i-th sample of the training set.

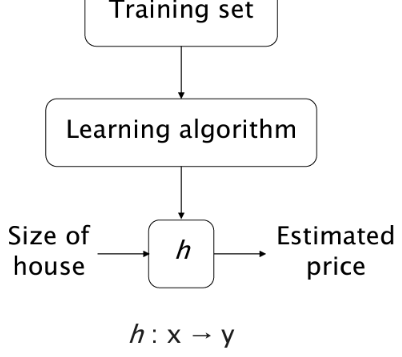


Figure 2.1: Scheme for model construction (price prediction)

The figure above shows schematically the steps in order to produce a certain model for the analysed case-study. Very briefly, a **training set** is used by a **learning algorithm** to obtain an *hypotesis* $h_\theta(x)$ which is later used for the prediction.
In the case we want to solve a **univariate linear regression problem** the hypotesis h has got the shape:

$$h_\theta(x) = \theta_0 + \theta_1 x \tag{2.1}$$

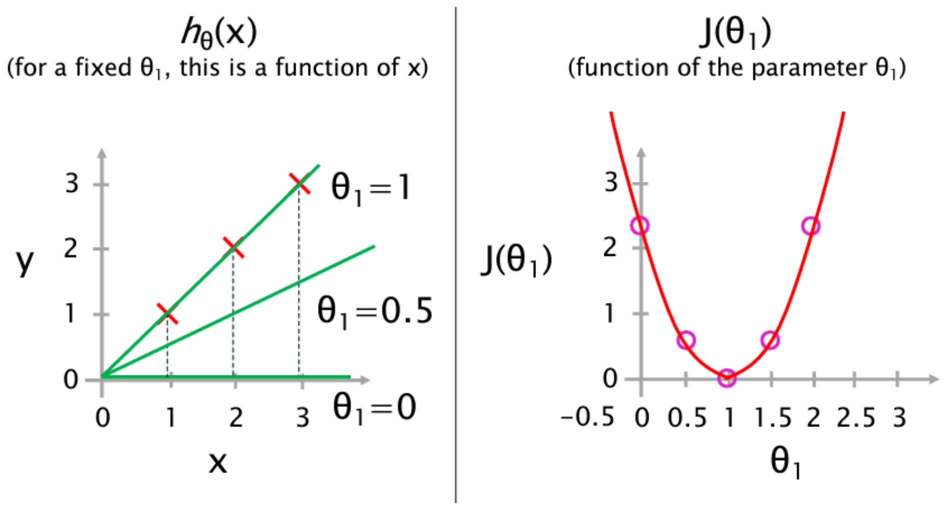
where θ_0 and θ_1 are the parameters of the line.[1] We call *univariate* the the problem since we have only one feature and it is a *linear regression* because we want to predict the price (output) according to a line.[2] **Question: How can we choose θ_0, θ_1?** Intuitively one can choose the parameters associated with the line $h_\theta(x)$ which is as closest as possible to the given y. Very often these parameters are the ones which solve the following problem:

$$\min_{\theta_0,\theta_1} \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\Big(\underbrace{h_\theta(x^{(i)})}_{\text{predicted value}} - \underbrace{y^{(i)}}_{\text{actual value}}\Big)^2 \tag{2.2}$$

The function $\frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ is the $\mathsf{Loss}(h_\theta(x), y)$ or $\mathsf{Cost}(h_\theta(x), y)$. If we call $J(\theta_0, \theta_1)$ the argument of the minimization problem the (2.2), the problem to solve can be expressed as

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1) \tag{2.3}$$

Summarizing: we want to perform a prediction using the hypotesis h which is dependent on parameters θ_0, θ_1 which are issued by minimizing a certain functional $J(\theta_0, \theta_1)$. Let us investigate better on the role of J in this supervised learning task.
At first – for sake of simplicity – we can eliminate a degree of freedom fixing the parameter θ_0 to be (without loss of generality) $\theta_0 = 0$. For each choice of θ_1 we will obtain a $h_{\theta_1}(x)$. If we compute $J(\theta_1)$ (for each θ) will obtain a certain univariate function $J(\theta_1)$, the minimization of which will give us the *optimal* θ_1 parameter for our hypotesis. An example is shown in the following figure:



Analyzing the complete model, we have two degrees of freedom (DOF) since θ_0, θ_1 can vary. In this case the functional to be minimized has to be represented in a 3D space, then we obtain a surface similar to one presented in the following figure:

[1] We can imagine them as two handles to: move up/down the line (θ_0) and to rotate it (θ_1).

[2] Note that in case of a **neural network** the parameters and the hypotesis assume a different notation. In particular the hypotesis becomes the *predicted value* indicated with $\hat{y}$, the parameters are split in a **bias**, indicated with b whose role is the one played by θ_0, while the θ_i, $i = 1, ..., n$ are the weights w_i

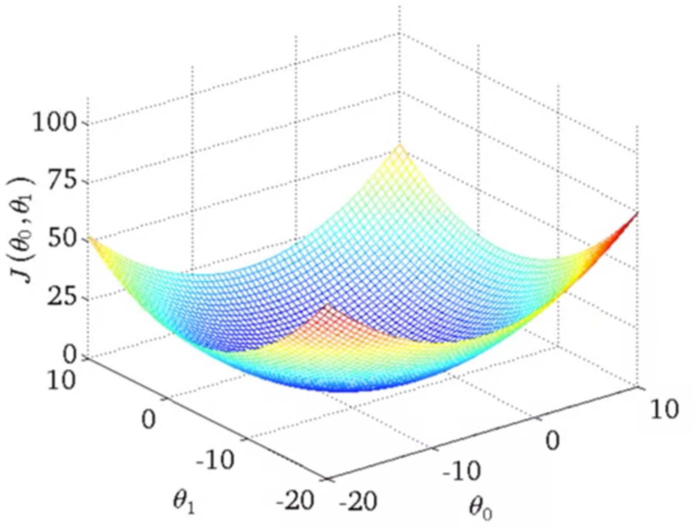
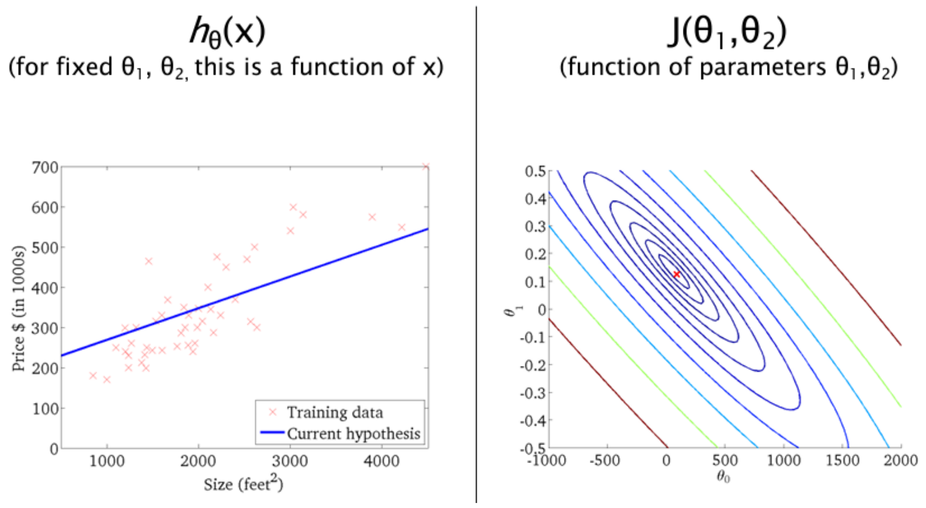


Figure 2.2: Example of $J(\theta_0, \theta_1)$

In the common case of bivariate minimization problem one can use *contour plot* which analyze the shape of the function at different heights. It is remarkable that points in the space (θ_0, θ_1) which are on the same *countour line* result in very different hypotesis. It is trivial to understand that, in this case the minimum $J(\theta_0, \theta_1)$ is attained on the bottom of such a *bowl-shaped* surface.

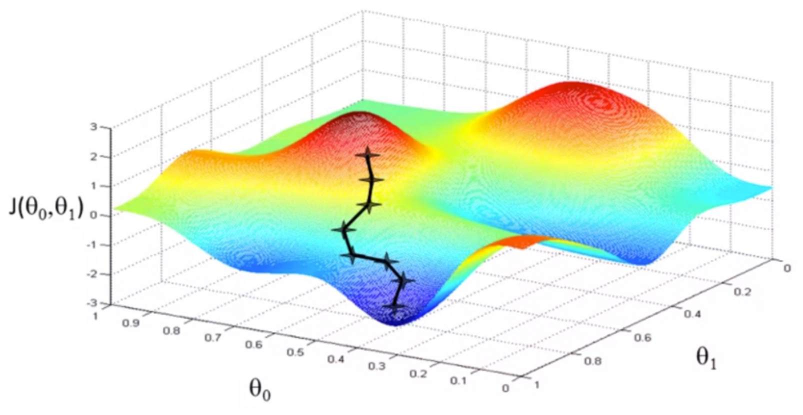


2.2 Parameter learning: Gradient descent

The objective here is to find a way to minimize a certain multivariate functional $J(\theta_1, ..., \theta_n)$, the idea is using some methods that iteratively bring us to the minimum according to a certain criteria. In this paragraph we analyse the **Gradient Descent** algorithm, the main idea here is to start with some θ_0, θ_1[3], and keep changing them until J evaluated at those parameters could reach (hopefully) the minimum, in the gradient descent this change is made up on the

[3]They are chosen either randomly or $\theta_i = 0, \ \forall i$.

basis of the direction dictated by the **gradient of the functional** computed at the current parameters value (from which the name). The algorithm for GD is simply as follows:



Algorithm 1 Gradient Descent

```
while !convergence do
    $\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1)$        ▷ for $j$=0, $j$=1,...
end while
```

The := symbol is associated with a *simultaneous update*, note that if you put together for each j the partial derivatives of J you will obtain the gradient. The parameter α is called the **learning rate** and it must be properly chosen because:

- If α is **too small**, then the convergence to the minimum (within a certain tolerance) could be very slow;
- If α is **too large** the algorithm can overshoot the minimum either failing to converge, or diverging.

Even when the learning rate is fixed the GD can converge to a (local) minimum since we are moving toward *steep* directions which decrease the functional over time. If we apply the algorithm to the functional of the problem in (2.2) we obtain:

$$\begin{aligned}\theta_0 &= \theta_0 - \alpha\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)}))\\ \theta_1 &= \theta_1 - \alpha\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)}))x^{(i)}\end{aligned} \tag{2.4}$$

this is known as **batch gradient descent** since for each step we use all the training samples. There are cases in which the minimization is particularly 'simple'. This happens when the functional is convex in θ. Besides, for the class of convex functions a local minima is also a **global and only min**.

2.3 Multivariate linear regression

It is quite immediate to understand that the linear regression can be used also for a *multivariate context* in which the samples are characterized by many features. In this context the hypotesis becomes:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n = \theta^T x \tag{2.5}$$

In this case we have n parameters and associated features x_i, so that the functional J is function of n parameters, in this case the partial derivatives to compute, obviously, will increase. Note that a *fictitious* feature $x_0 = 1$ has been added with the purpose to employ a vector notation.[4]

2.4 Data mean normalization

Sometimes, before starting with the model construction, some preliminary operations are needed. For example, often it is better for the features being on a **similar scale**. In this case we replace in each sample for each feature $x_i = x_i/s_i$ where s_i can be either the range (max-min) for that feature or some index similar to variance/standard deviation.
Other times, one is supposed to normalize the data so that they can have a *zero mean.* The trick here is replacing $x_i = x_i - \mu_i$, where μ_i is the mean for the i-th feature. Not rarely, you can find the two transformation combined such thet

$$x_i = \frac{x_i - \mu_i}{s_i} \tag{2.6}$$

2.5 Debug of Gradient Descent algorithm

The *gradient algorithm* is clearly a descent method in the sense that – being k the k-th iteration – it holds that $J(\theta_{k+1}) < J(\theta_k)$, this is the same to state that the $J(\theta)$ function is required to be strictly decreasing. An **automatic convergence test** can be performed: for example the objective function J, had had a decreasing less than a certain threshold $\varepsilon = 10^{-3}$ (for example).
Whether the algorithm is not working well the value for the **hyperparameter** α must be changed (for example decreasing it). One way to choose *manually* α is by *trial-error*[5], choosing the α in a range and then plotting $J(\theta)$ as a function of the number of iterations.

2.6 Alternative to Gradient Descent

There are alternative methods to gradient descent, for example the normal equation method which is derived by the analytical solution of the well-known **least-squares** problem. In this case θ is found by solving the system (normal equations):

$$(X^T X)\theta = X^T y \tag{2.7}$$

where the X matrix contains the dataset features and y is the vector with the "right answers". The solution of such a problem gives *one-shot* the solution without proceeding by step as in the case of gradient descent. The main limitation of such a method is the inversion of the matrix $X^T X$, which could be significantly slow if n (number of features and parameters) is very large.[6]

[4]The great majority of tools and softwares which are used for machine learning exploit vector and matrices calculus to carry out their work.

[5]A more accurate method is the **backtracking line-search** which repeat some calculations until the so-called *Armijo condition* is not met; however it requires that additive hypotesis are made on the regularity of the objective and its gradient.

[6]It is sufficient to think about the number of parameters involved in a problem of image classification. They are in a number of 3 billion for an RGB 1000×1000 image.

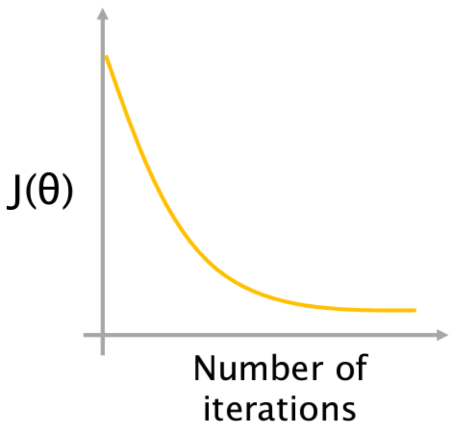


Figure 2.3: Desired behaviour for $J(\theta)$ vs # iterations

2.7 Polynomial regression