

# Linear Models and SVMs

## Optimization for Machine Learning — Exercise 02

Monday 24<sup>th</sup> April, 2023

### Part I: Theory

#### I.1. Linear model example

**Exercise I.1** (Polynomial Curve Fitting). Given a set of points and their targets  $\{x_i, t_i\}_{i=1}^N$  so that for  $i \in [N]$ ,  $x_i \in \mathbb{R}$  and  $t_i \in \mathbb{R}$ , the *curve fitting problem* is loosely defined as finding a function  $f: \mathbb{R} \rightarrow \mathbb{R}$  such that  $f(x_i) \approx t_i$  for all  $i \in [N]$ .

In order to find such a function, we restrict ourselves to a set of parametrized functions  $\mathcal{F}$ : each function can be parametrized with a vector  $\mathbf{w} \in \mathbb{R}^D$ .

To quantify the problem further, in this exercise, we limit ourselves to *polynomial functions* of degree  $D$  for the set  $\mathcal{F}$ , and can therefore write

$$f(x, \mathbf{w}) = w_0 + w_1x + \dots + w_Dx^D = \sum_{k=0}^D w_kx^k \quad (1)$$

Notice how  $f$  is *linear* in  $\mathbf{w}$ , the parameter. Such model is called a *linear model*.

With  $N$  samples, we defined the loss (or error, or energy) of our parameter as the point-wise square distance between its estimation and the target:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (f(x_i, \mathbf{w}) - t_i)^2 \quad (2)$$

1. Is the function  $E$  convex in  $\mathbf{w}$ ? How to find the optimal parameter  $\mathbf{w}^*$  at which the loss is minimum?

2. Compute the gradient  $\nabla_{\mathbf{w}} E(\mathbf{w}) = \begin{pmatrix} \partial_{w_0} E(\mathbf{w}) \\ \vdots \\ \partial_{w_D} E(\mathbf{w}) \end{pmatrix} \in \mathbb{R}^D$ .

3. Show that the optimal parameter  $\mathbf{w}^*$  satisfies the following system of equation:

$$\sum_{j=0}^D A_{ij} w_j = T_i,$$

where

$$A_{ij} = \sum_{k=1}^N (x_k)^{i+j}, \quad T_i = \sum_{k=1}^N (x_k)^i t_k. \quad (3)$$

4. Is such a system of equation solvable? When / not?

It is usual to add a *regularizer* to the objective, penalizing “complex” models. This also can help selecting a model when several models are solutions to the optimization problem.

One of the most common regularizer is the parameter squared-norm: with a *penalizer weight*  $\lambda \in \mathbb{R}_+$ , the Equation (2) is modified to give

$$E_\lambda(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \sum_{i=1}^N (f(x_i, \mathbf{w}) - t_i)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2. \quad (4)$$

5. What is the role of  $\lambda$ ?

6. Is  $E_\lambda$  convex?

7. Show that each component of the optimal weight  $w_i^*$  is now found by solving

$$\sum_{j=0}^D (A_{ij} + \lambda) w_j = T_i,$$

with  $A_{ij}$  and  $T_i$  defined as in Equation (3).

**Matrix expression** It is sometimes preferable to deal with vector and matrices, rather than scalar expressions. When the model is linear in  $\mathbf{w}$ , it is possible to express it as a *linear product* between a matrix and a vector. The expression in Equation (1) can be thought as a dot product

between  $w$  and the vector of powers of  $x$ , that define as  $\phi(x) := \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^D \end{pmatrix}$ , so that

$$f(x, \mathbf{w}) = \mathbf{w}^\top \phi(x).$$

Stacking all the  $N$  examples in a matrix, and denoting  $\phi_i := \phi(x_i)$ , we define

$$\Phi := \begin{pmatrix} | & | & \cdots & | \\ \phi_1 & \phi_2 & & \phi_N \\ | & | & & | \end{pmatrix} \in \mathbb{R}^{D \times N}$$

and can therefore compute the model *on the whole dataset* in one expression:  $\mathbf{y}(\mathbf{w}) = \Phi^\top \mathbf{w} \in \mathbb{R}^N$ . Each entry  $i$  of  $\mathbf{y}$  corresponds to a different sample  $x_i$ . Then, stacking the targets into a vector  $\mathbf{t} \in \mathbb{R}^N$ , the error function (2) can equivalently written as

$$E(\mathbf{w}) = \frac{1}{2} \|\mathbf{y}(\mathbf{w}) - \mathbf{t}\|^2,$$

and the regularized error as

$$E_\lambda(\mathbf{w}) = \frac{1}{2} \|\mathbf{y}(\mathbf{w}) - \mathbf{t}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

8. Show that  $\nabla E_\lambda(\mathbf{w}) = \Phi(\Phi^\top \mathbf{w} - \mathbf{t}) + \lambda \mathbf{w}$ , so that  $\mathbf{w}^*$  solves the linear equation

$$(\Phi\Phi^\top + \lambda I_D) \mathbf{w}^* = \Phi \mathbf{t}.$$

## I.2. Support Vector Machines (SVM)

In order to implement a first SVM algorithm, we will need the notion of *subgradient* we give next.

### I.2.1. Subgradients

When a convex loss function  $E: \mathbb{R}^d \rightarrow \mathbb{R}$  is not differentiable, its *subgradient* can be used. It is defined as the set, for  $x \in \mathbb{R}^d$ ,

$$\partial E(x) = \{g \in \mathbb{R}^d \mid \forall y \in \mathbb{R}^d, E(y) \geq E(x) + \langle g, y - x \rangle\}.$$

If  $E$  is differentiable at  $x$ , then  $\partial E(x) = \{\nabla E(x)\}$ .

For instance, for  $E: \mathbb{R} \rightarrow \mathbb{R}$ ,  $x \mapsto E(x) = |x|$ ,  $E$  is differentiable at any  $x \neq 0$ , with gradient  $-1$  on  $(-\infty, 0)$  and  $1$  on  $(0, +\infty)$ .

At  $x = 0$ , we compute, for any  $y \in \mathbb{R}$  and  $g \in \mathbb{R}$ :

$$\begin{aligned} E(y) \geq E(0) + \langle g, y - 0 \rangle &\iff |y| \geq \langle g, y \rangle \\ &\iff |y| \geq gy \end{aligned}$$

This condition has to be true for *any*  $y \in \mathbb{R}$ . This is only true when  $g \in [-1, 1]$ . Therefore,

$$\partial E(x) = \begin{cases} \{-1\} & \text{if } x < 0 \\ [-1, 1] & \text{if } x = 0 \\ \{1\} & \text{if } x > 0 \end{cases}$$

Geometrically, this can be interpreted as having, for the absolute value at the origin, any lines with slope between  $-1$  and  $1$  lower-bounding the graph of the function.

**Exercise I.2.** Let  $E: \mathbb{R} \rightarrow \mathbb{R}$ ,  $x \mapsto E(x) = \max(0, 1 - x)$ .

1. Where is  $E$  differentiable?

2. Show that  $\partial E(x) = \begin{cases} \{-1\} & \text{if } x < 1 \\ [-1, 0] & \text{if } x = 1 \\ \{0\} & \text{if } x > 1 \end{cases}$

### I.2.2. SVM problem

Now, the SVM problem can be presented. We will discuss the (pure) linear case.

Support Vector Machines solve a binary classification task. Given  $N$  couples samples / targets  $\{x_i, t_i\}_{i \in [N]}$ , with  $x_i \in \mathbb{R}^d$  and  $t_i \in \{-1, 1\}$  for each  $i \in [N]$ , the goal is to classify the samples, i.e. find a **hyperplane that separates them**, with positive samples on one side of the hyperplane and the negative on the other. We assume that such an hyperplane exists (the samples are said to be *linearly separable*).

A hyperplane in  $\mathbb{R}^d$  is represented with a vector  $w \in \mathbb{R}^d$  and a *bias*  $b \in \mathbb{R}$  with the equation

$$y(x) = \langle w, x \rangle + b.$$

This equation splits  $\mathbb{R}^d$  into three regions:

- points such that  $y(x) > 0$ ,
- points such that  $y(x) = 0$  (the hyperplane itself),
- points such that  $y(x) < 0$ .

Therefore, we would like to find an hyperplane such that the samples  $x_i$  that have a positive target  $t_i = 1$  all lie on the side of the hyperplane where  $y(x) > 0$ , and reciprocally all samples  $x_i$  such that  $t_i = -1$  should be on the side where  $y(x) < 0$ .

Therefore, the product  $t_i y(x_i)$  should always be positive. Based on that, the loss we defined is

$$E(w) = \sum_{i=1}^N \max(0, 1 - t_i y_i) = \sum_{i=1}^N \max(0, 1 - t_i (\langle w, x_i \rangle + b)) \quad (1)$$

- Exercise I.3.**
1. Is the loss  $E$  convex? Differentiable?
  2. Compute the subgradient of the loss  $E$  at  $w$ .
  3. What should be the (sub-) gradient descent algorithm to minimize  $E$ ?

## Part II: Programming

**Exercise II.1.** Model fitting This exercise implements some results found in Exercise I.1.

1. **Generation of the target.** In this toy example, we generate the  $N$  points ourselves. The true target  $t_i$  will be sinusoidal, with some noise, i.e.  $t_i = \sin(2\pi x_i) + \varepsilon$ , where  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ . The different scales (for  $\sigma, x_i$ ) are given as  $\sigma = 0.1$ , and  $x_i \sim \mathcal{U}([0, 1])$ , uniform distribution on the segment  $[0, 1]$ .

The generation of the data is performed by the function `gen_sin_data` in the file `ex02/Utils.py`.

2. Implement the parametrization function (1) as  $f(x, w)$ , where the dimensions  $D$  is implied by the size of  $w$ .
3. Implement the error function  $E$  defined in (2), and its gradient  $\nabla E(w)$ .
4. Find  $w^*$ , either by
  - a) gradient descent; or
  - b) solving the linear system of equations (3).

**Exercise II.2 (SVM).** This exercise implements some material from Exercise I.3. The data is generated with the helper function `gen_binary_data` in `ex02/utils.py`. The dimension is set to  $d = 2$  in order to visualize the result at the end. The generation simply draws some random points on the plane, draws an hyperplane, and classify the points depending on the sign of  $w^\top x + b$ . Therefore, the training data is linearly separable.

1. Implement the loss from (1).
2. Implement the (sub-) gradient algorithm derived in I.3.3
3. Visualize the solution found by the algorithm. What happens if the data is not linearly separable?