Linear Models and SVMs

Optimization for Machine Learning — Exercise #2

Monday 24th 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 $f\colon \mathbb{R} \to \mathbb{R}$ 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 $\boldsymbol{w} \in \mathbb{R}^{D+1}$.

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

$$f(x, \boldsymbol{w}) = w_0 + w_1 x + \ldots + w_D x^D = \sum_{k=0}^{D} w_k x^k \tag{1}$$

Notice how f is *linear* in $\boldsymbol{w}$, 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:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(f(x_i, \boldsymbol{w}) - t_i\right)^2 \tag{2}$$

1. Is the function E convex in $\boldsymbol{w}$? How to find the optimal parameter $\boldsymbol{w}^\star$ at which the loss is minimum?
2. Compute the gradient $\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \begin{pmatrix} \partial_{w_0} E(\boldsymbol{w}) \\ \vdots \\ \partial_{w_D} E(\boldsymbol{w}) \end{pmatrix} \in \mathbb{R}^{D+1}$.
3. Show that the optimal parameter $\boldsymbol{w}^\star$ satisfies the following system of equation:

$$\forall k \in [D+1], \quad \sum_{j=0}^{D} A_{kj} w_j^\star = T_k,$$

where

$$A_{kj} = \sum_{i=1}^{N} (x_i)^{k+j}, \qquad T_k = \sum_{i=1}^{N} (x_i)^k t_i. \tag{3}$$

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

Answers

1. Each term in E is convex in $\boldsymbol{w}$ as the composition of a linear function $\boldsymbol{w} \mapsto f(x, \boldsymbol{w})$ and the convex functions $z \mapsto (z-t)^2$. The function E is then convex as the sum of convex functions in $\boldsymbol{w}$.

   Therefore, the optimum $\boldsymbol{w}^\star$ can be found with the first-order optimality condition: $\boldsymbol{w}^\star$ satisfies $\nabla E(\boldsymbol{w}^\star) = 0$.

2. For $k \in [D+1]$, we write $\partial_k := \partial_{w_k}$. The gradient can then be computed as

$$\partial_k E(\boldsymbol{w}) = \partial_k \left(\frac{1}{2} \sum_{i=1}^{N} (f(x_i, \boldsymbol{w}) - t_i)^2 \right) \tag{4}$$

$$= \sum_{i=1}^{N} (f(x_i, \boldsymbol{w}) - t_i) \partial_k f(x_i, \boldsymbol{w}) \tag{5}$$

$$= \sum_{i=1}^{N} (f(x_i, \boldsymbol{w}) - t_i)(x_i)^k, \tag{6}$$

$$= \sum_{i=1}^{N} \left(\sum_{j=0}^{D} w_j (x_i)^j - t_i \right) (x_i)^k, \tag{7}$$

   and $\nabla E(\boldsymbol{w}) = (\partial_k E(\boldsymbol{w}))_{k \in [D+1]}$.

3. From 1, the minimizer $\boldsymbol{w}^\star$ is found by solving the equation $\nabla E(\boldsymbol{w}^\star) = 0$, i.e., for all $k \in [D+1]$,

$$\partial_k E(\boldsymbol{w}^\star) \stackrel{!}{=} 0 \implies \sum_{i=1}^{N} \left(\sum_{j=0}^{D} w_j^\star (x_i)^j (x_i)^k - t_i (x_i)^k \right) = 0 \tag{8}$$

$$\implies \qquad \sum_{j=0}^{D} \underbrace{\sum_{i=1}^{N} (x_i)^{j+k}}_{A_{kj}} w_j^\star = \underbrace{\sum_{i=1}^{N} (x_i)^k t_i}_{T_k}. \tag{9}$$

4. This linear system can be written as $A\boldsymbol{w} = T$, where $A = (A_{kj})_{k,j \in [D+1] \times [D+1]}$, and $T = (T_k)_{k \in [N]}$.

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

$$E_\lambda(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\sum_{i=1}^{N}\left(f(x_i, \boldsymbol{w}) - t_i\right)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2. \tag{10}$$

5. What is the role of λ?

6. Is E_λ 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 $\boldsymbol{w}$, 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 $\boldsymbol{\phi}(x) := \begin{pmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^D \end{pmatrix}$, so that

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

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

$$\Phi := \begin{pmatrix} | & | & & | \\ \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 & \cdots & \boldsymbol{\phi}_N \\ | & | & & | \end{pmatrix} \in \mathbb{R}^{(D+1)\times N}$$

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

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

and the regularized error as

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

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

$$(\Phi\Phi^\top + \lambda I_{D+1})\boldsymbol{w}^* = \Phi \boldsymbol{t}.$$

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 $E\colon \mathbb{R}^d \to \mathbb{R}$ is not differentiable, its *subgradient* can be used. It is defined as the set, for $x \in \mathbb{R}^d$,

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

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

For instance, for $E\colon \mathbb{R} \to \mathbb{R},\ x \mapsto E(x) = |x|$, 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 $g \in \mathbb{R}$:

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

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 $E\colon \mathbb{R} \to \mathbb{R},\ x \mapsto E(x) = \max(0, 1 - x)$.

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

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

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 σ,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?