Machine Learning

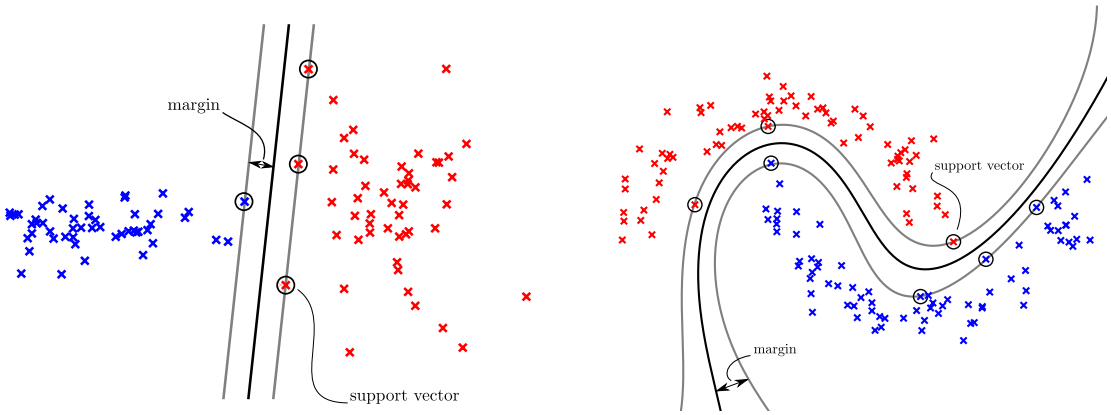
Assignment 3

June 3rd, 2020

Deadline: June 16, 2020 at 23:55h.
Submission: Upload your report and your implementation to the TeachCenter. Please do not zip your files.

1 Datasets

We are going to use two different datasets in this assignment. The two datasets are called *simple dataset* and *half-moon dataset* and are visualized in fig. 1a and fig. 1b, respectively. The visualizations also show the main parts of a Support Vector Machine (SVM), *i.e.* the decision boundary, the support vectors and the margin. The next two sections describe how these datasets can be generated.



(a) Simple Dataset (b) Half-Moon Dataset

Figure 1: Visualization of the datasets used in this assignment including the *margin*, the *support vectors* and the *decision boundary*.

1.1 Simple Dataset

The simple dataset consists of $N = 200$ points in $2D$ which are linearly separable. The positive and negative samples are drawn from two Gaussian distributions with parameters

$$\mu_1 = \begin{pmatrix} 6.5 \\ 2 \end{pmatrix} \qquad \Sigma_1 = \begin{pmatrix} 0.8 & 0 \\ 0 & 0.7 \end{pmatrix} \tag{1}$$

and

$$\mu_2 = \begin{pmatrix} 1.5 \\ 2 \end{pmatrix} \qquad \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 0.2 \end{pmatrix} \tag{2}$$

with corresponding class labels $t_n \in \{-1, +1\}$. You can draw samples from a multivariate normal distribution by using `numpy.random.multivariate_normal`.

1.2 Half-Moon Dataset

The second dataset is called the "Half-Moon" dataset and consists of $N = 200$ points in $2D$. You can generate samples for this dataset for the two target classes $t = +1$ and $t = -1$ with

$$\begin{aligned} \eta_1, \eta_2 &\sim \mathcal{N}(0, \sigma_D^2) \\ \theta &\sim \mathcal{U}(0, \pi) \\ x_t &= \begin{cases} \cos(\theta) + \eta_1 & \text{if } t = +1 \\ 1 - \cos(\theta) + \eta_1 & \text{if } t = -1 \end{cases} \\ y_t &= \begin{cases} \sin(\theta) + \eta_2 & \text{if } t = +1 \\ \frac{1}{2} - \sin(\theta) + \eta_2 & \text{if } t = -1 \end{cases}, \end{aligned}$$

where x and y are the spatial coordinates, t is the target class and $\mathcal{U}(a, b)$ denotes a uniform distribution on the interval $[a, b]$. We set $\sigma_D^2 = 0.2$ in all experiments.

Tasks

1. Generate and plot the simple dataset described in section 1.1
2. Generate and plot the half-moon dataset described in section 1.2

2 Support Vector Machine - Primal Problem

The support vector machine (SVM) is a machine learning algorithm for classification. To illustrate the main property of the SVM let us consider the linearly separable dataset shown in fig. 1. We can come up with infinitely many lines, which will perfectly separate the two classes. The aim of a SVM is to compute the *best* separating line. "Best" is here defined in terms of the margin, which measures the perpendicular distance between the decision boundary and the closest of the data points. In this assignment, we consider the following slightly generalized SVM formulation:

$$\min_{\tilde{\mathbf{w}}} \ \frac{1}{2}\|\tilde{\mathbf{w}}\|_{\mathbf{M}}^2 \tag{3}$$

$$\text{s.t. } t_n \langle \tilde{\mathbf{w}}, \phi(\mathbf{x}_n) \rangle \geq 1 \quad \forall n = 1, \dots, N \tag{4}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product, $\tilde{\mathbf{w}} = (b, \mathbf{w})^T$ is the homogeneous parameter vector, $\phi(\mathbf{x})$ denotes the feature transform and $\|\tilde{\mathbf{w}}\|_M^2 = \tilde{\mathbf{w}}^T \mathbf{M} \tilde{\mathbf{w}}$ is the quadratic regularization term but taken in a metric

$$\mathbf{M} = \text{diag}(\boldsymbol{\gamma}), \qquad \boldsymbol{\gamma} = (\gamma_1, \gamma_2, \dots) \tag{5}$$

where set $\gamma_1 = 1e^{-6}$ and $\gamma_{i>1} = 1$. Making use of this metric still allows us to interpret the first component of the parameter vector $\tilde{\mathbf{w}}$ as the bias b but will lead to a simpler dual formulation.

The primal SVM eqs. (3) and (4) is hard to solve by a gradient based method, but we can still approximately solve it by considering the soft-SVM formulation

$$\min_{\tilde{\mathbf{w}}} f(\tilde{\mathbf{w}}) := \frac{\lambda}{2}\|\tilde{\mathbf{w}}\|_{\mathbf{M}}^2 + \frac{1}{N}\sum_{n=1}^{N} \underbrace{\max(0, 1 - t_n \langle \tilde{\mathbf{w}}, \phi(\mathbf{x}) \rangle)}_{\text{hinge loss}}, \tag{6}$$

where the hinge loss function can be interpreted as a soft-penalization of the inequality constraints eq. (4) and $\lambda > 0$ is a trade-off parameter. To optimize eq. (6) we use a proximal sub-gradient method, because the max operation in the hinge-loss is only sub-differentiable. We can compute a sub-gradient of the hinge-loss in eq. (6) with

$$\mathbf{g} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{g}_n, \qquad \mathbf{g}_n = \begin{cases} 0 & \text{if } t_n \langle \tilde{\mathbf{w}}, \phi(\mathbf{x}_n) \rangle \geq 1 \\ -t_n \phi(\mathbf{x}_n) & \text{else.} \end{cases} \tag{7}$$

Algorithm 1: Proximal Sub-gradient Method

Set $\tilde{\mathbf{w}}^0 \in \mathbb{R}^D$, $\alpha > 0$
for $i > 0$ **do**
$\tilde{\mathbf{w}}_{i+1} = \dfrac{\tilde{\mathbf{w}}_i - \alpha \mathbf{g}_i}{1 + \alpha\lambda\boldsymbol{\gamma}}$ (elementwise division)
end

Then, we can use the proximal sub-gradient method shown in algorithm 1 in order to optimize eq. (6), where $\mathbf{g}_i$ is eq. (7) evaluated in the i-th iteration of the algorithm and α is the step-size.

The separating hyperplane is directly defined using the optimal parameters $\tilde{\mathbf{w}}^*$ with

$$y(\mathbf{x}) = \langle \tilde{\mathbf{w}}^*, \phi(\mathbf{x}) \rangle = 0, \tag{8}$$

and therefore the classification is given by

$$\hat{t} = \begin{cases} +1 & \text{if } y(\mathbf{x}) \geq 0 \\ -1 & \text{else.} \end{cases} \tag{9}$$

Tasks

1. Apply the following tasks to the simple dataset and the half-moon dataset using $\phi(\mathbf{x}) = (1, \mathbf{x})^T$:
   a) Implement algorithm 1
   b) Apply the sub-gradient method to the datasets and iterate the optimization until convergence.
   c) Compute the resulting hyperplane (=a line in 2D) and create a plot similar to the plot in fig. 1 including the dataset itself, the separating hyperplane, the margin and the support vectors.
2. What can you observe in the simple and in the half-moon dataset? Why can the data points not be separated in the half-moon dataset?

3 SVM Dual Problem

We have seen that the primal problem is hard to solve exactly and it becomes even harder when using a non-linear feature transform or even impossible if the feature transform is high-dimensional. However, this is perfectly possible in the *dual* formulation of the SVM. The Lagrange dual problem of eqs. (3) and (4) is given by

$$\max_{\mathbf{a}} D(\mathbf{a}) := \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \underbrace{\phi(\mathbf{x}_n)^T \mathbf{M}^{-1} \phi(\mathbf{x}_m)}_{k(\mathbf{x}_n, \mathbf{x}_m)} \tag{10}$$

$$\text{s.t. } a_n \geq 0 \quad \forall n, \tag{11}$$

where a_n is the Lagrange multipliers for the n-th data point and $k(\cdot, \cdot)$ denotes the kernel function between two data points. Observe that thanks to the metric $\boldsymbol{M}$ this dual formulation does not involve equality constraints of the form $\sum_{n=1}^{N} a_n t_n = 0$ which eases up the optimization. Since we are free to pick a kernel, we use here the infinitely dimensional Gaussian kernel

$$k_G(\mathbf{x}_n, \mathbf{x}_m) = \exp\left(\frac{-\|\mathbf{x}_n - \mathbf{x}_m\|^2}{2\sigma^2} \right), \tag{12}$$

where σ^2 is the variance of the Gaussian and $\mathbf{x}_n$ and $\mathbf{x}_m$ are the two inputs to the Gaussian kernel. Note that the kernel here is not interpreted as a probability distribution and thus not normalized.

We will use a variant of the FISTA algorithm shown in algorithm 2 to compute the optimal dual variable $\mathbf{a}^*$. The projection operation in the algorithm therein ensures that the constraint eq. (11) is fulfilled. This can be done by simply projecting the dual variables to the valid interval, *i.e.*

$$\text{proj}(\mathbf{a})_n = \max(a_n, 0), \quad n = 1, \ldots, N \tag{13}$$

Algorithm 2: Fast Iterative Shrinkage-Thresholding Algorithm (FISTA)

Set $t_1 = 1, \quad \mathbf{a}_1 = \mathbf{a}_0 = \mathbf{0}$
Choose $\alpha > 0$
for $i > 0$ **do**
  $t_{i+1} = \frac{1+\sqrt{1+4t_i^2}}{2}$
  $\tilde{\mathbf{a}}_i = \mathbf{a}_i + \frac{t_i - 1}{t_{i+1}} (\mathbf{a}_i - \mathbf{a}_{i-1})$
  $\mathbf{a}_{i+1} = \text{proj} (\tilde{\mathbf{a}}_i + \alpha \nabla D(\tilde{\mathbf{a}}_i))$
end

Note that we perform gradient ascent steps, because eq. (10) is in difference to the primal problem a MAXimization problem.

The dual variable does not directly define the separating hyperplane as in the primal problem. Instead, equipped with the optimal dual variables $\mathbf{a}^*$, we must compute the decision boundary with

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n). \tag{14}$$

Tasks Please use exactly the same datasets as in the primal SVM task.

1. Start with eqs. (3) and (4) and analytically compute the dual problem shown in eqs. (10) and (11). Follow the derivation of the dual SVM in the lecture notes but do not forget to include the metric $\mathbf{M}$ in the qudaratic norm and recall that there is no explicit bias parameter anymore. Show all the steps of your derivation.
2. Analytically compute the gradient $\nabla D(\mathbf{a})$ and state all steps of your computation.
3. Implement the FISTA algorithm shown in algorithm 2 and pick the step size $\alpha > 0$ by hand.
4. Perform the following steps with the Gaussian kernel eq. (12) on both datasets.
   a) Compute the optimal dual variables $\mathbf{a}^*$ with your FISTA implementation.
   b) Compute and plot the dual energy $D(\mathbf{a})$ over the iteration of the optimization.
   c) Plot the decision boundary $y(\mathbf{x}) = 0$ and the margins at $y(\mathbf{x}) = \pm 1$ in addition to the data points. Experiment with the parameter σ^2 of the Gaussian kernel and report your used values in the plots.
   d) Highlight the support vectors similar as shown in fig. 1.

Implementation details Implement the tasks with Python and name the file `svm.py`. Please do not use any other libraries than `numpy` and `matplotlib`.
Hint: As in the last assignment do not forget to check your gradients numerically with `scipy.optimize.approx_fprime`.