PROJECT DESCRIPTION

MARKUS GRASMAIR

1. Classification and Supervised Learning

The description and mathematical formulation of the classification problem follows the presentation in [Ste15].

We are given a set of data points $x_i \in \mathbb{R}^d$, $i = 1, \dots, M$, each belonging to one of two classes A and B. The class membership of data point x_i is indicated by an additional label $y_i \in \{-1, +1\}$, where $y_i = -1$ indicates that the data point x_i belongs to class A, and $y_i = +1$ indicates that x_i belongs to class B. Based on these given labelled data points, we want to "learn" the general structure of the two classes such that we can decide the class membership of arbitrary new data points $x \in \mathbb{R}^d$. Mathematically spoken, this means that we want to find a function $F\colon \mathbb{R}^d \to \{-1, +1\}$ such that $F(x_i) = y_i$ for all given, labelled data points. Because of the difficulty of dealing with discrete, and therefore non-continuous, let alone differentiable, functions, the classifier F is usually defined as

$$F(x) := \operatorname{sgn}(f(x)),$$

where $f\colon \mathbb{R}^d \to \mathbb{R}$ is some "nice" smooth function. Here $\operatorname{sgn}\colon \mathbb{R}^d \to \{-1, 0, +1\}$ is the sign function defined as

$$\operatorname{sgn}(t) = \begin{cases} +1 & \text{if } t > 0, \\ 0 & \text{if } t = 0, \\ -1 & \text{if } t < 0. \end{cases}$$

Do note that this leads to some borderline cases, typically lying at the boundary between the two classes, where $f(x) = 0$ and the classifier F cannot decide which class the point x belongs to.

The quality of the classification is highly dependent choice of the possible functions f (as well as the availability of training data x_i that provide a good description of the two classes). In this project, we/you will consider two specific, closely related approaches, which result in convex optimisation problems with linear constraints.

2. Linear Support Vector Classification

2.1. **Hard Margin Classification.**

The underlying assumption for the first approach is that the two classes A, $B \subset \mathbb{R}^d$ can be separated by a hyperplane, see Figure 1.

In this case, we can parametrise the function f as affine function

$$f_{w,b}(x) := \langle w, x \rangle + b$$

with $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$. The hyperplane

$$H(w, -b) = \big\{x \in \mathbb{R}^d : \langle w, x \rangle = -b\big\} = \big\{x \in \mathbb{R}^d : f_{w,b}(x) = 0\big\}$$

then forms the boundary between the two classes. Moreover, the hyperplane $H(w, -b)$ separates the two classes correctly, if

$$\operatorname{sgn}\big(\langle w, x_i \rangle + b\big) = y_i \qquad \text{for } i = 1, \dots, M,$$

Date: 20th February 2025.

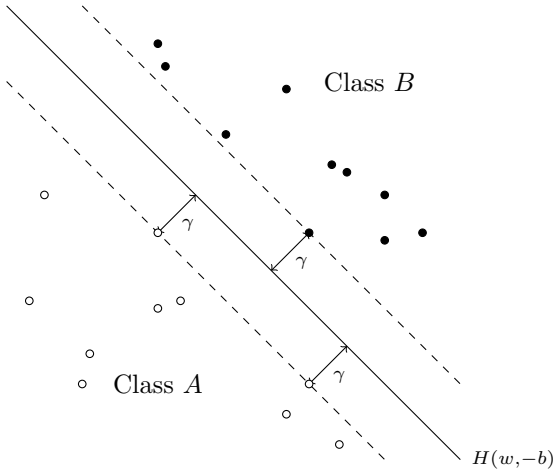

FIGURE 1. A hyperplane $H(w,-b)$ separating two classes of data points shown as filled and empty circles. The value γ is the *margin*, that is, the minimal distance between a point in one of the classes and the separating hyperplane.

which can be rewritten as

$$y_i\big(\langle w, x_i\rangle + b\big) > 0 \qquad \text{for } i = 1, \dots, M. \tag{1}$$

In order to obtain the best possible separation between the classes, it makes sense to choose the vector w and the offset b in such a way that the minimal distance (the *margin*) between data points x_i and the hyperplane $H(w,-b)$ is maximised. Assuming all the points are separated by $H(w,-b)$, we can compute their distance to the hyperplane as

$$\operatorname{dist}(x_i, H(w,-b)) = y_i\Big(\Big\langle \frac{w}{\|w\|}, x_i\Big\rangle + \frac{b}{\|w\|}\Big).$$

The margin is therefore

$$\gamma := \min_{i=1,\dots,M} \operatorname{dist}(x_i, H(w,-b)) = \min_{i=1,\dots,M} \frac{y_i(\langle w, x_i\rangle + b)}{\|w\|}.$$

In order to maximise the margin, we thus have to solve the optimisation problem

$$\max_{\substack{w\in\mathbb{R}^d,\\ b\in\mathbb{R}}} \min_{i=1,\dots,M} \frac{y_i(\langle w, x_i\rangle + b)}{\|w\|},$$

subject to the constraint that the hyperplane $H(w,-b)$ separates the points correctly, that is, that (1) holds. We now note that the hyperplane $H(w,-b)$ (including its orientation) remains unchanged, if we scale both w and b with the same positive factor $c > 0$, that is, $H(w,-b) = H(cw,-cb)$ if $c > 0$. As a consequence, we can always simultaneously rescale w and b in such a way that

$$\min_{i=1,\dots,M} y_i\big(\langle w, x_i\rangle + b\big) = 1.$$

With this rescaling, we obtain that $\gamma = 1/\|w\|$. Thus maximising γ is the same as minimising $\|w\|$. We therefore obtain the constrained optimisation problem

$$\min_{\substack{w\in\mathbb{R}^d,\\ b\in\mathbb{R}}} \frac{1}{2}\|w\|_2^2 \qquad \text{s.t. } y_i\big(\langle w, x_i\rangle + b\big) \geq 1 \text{ for } i = 1, \dots, M. \tag{2}$$

The resulting classifier $F^*(x) = \operatorname{sgn}(\langle w^*, x\rangle + b^*)$ is called the *hard margin (linear) classifier* for our data set. The margin γ can then be computed as $\gamma = 1/\|w^*\|$.

2.2. **Soft Margin Classification.** In practical applications it often happens that no hyperplane exists that neatly separates the two classes. In this situation, the problem (2) becomes infeasible. In order to still obtain a reasonable classification (though at the price that some data points will be misclassified) one can introduce *slack variables* $\xi_i \geq 0$, $i = 1, \dots, M$, and a regularisation parameter $C > 0$ into the model, resulting in the problem

$$\min_{\substack{w\in\mathbb{R}^d,\\ b\in\mathbb{R},\\ \xi\in\mathbb{R}^M}} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^M \xi_i \qquad \text{s.t.} \quad \begin{cases} y_i\big(\langle w, x_i\rangle + b\big) \geq 1 - \xi_i, \\ \xi_i \geq 0, \end{cases} \text{ for } i = 1, \dots, M. \tag{3}$$

The idea here is that we are allowed to misclassify a data point x_i, but we have to pay a price $C\xi_i$, where $\xi_i > 0$ is essentially the severity of the misclassification.[1] The resulting classifier here is called the *soft margin (linear) classifier*. The parameter $C > 0$ steers the price of the misclassification: The larger C is, the more we have to pay, and we recover the hard margin classifer in the limit $C \to \infty$.

Using the *hinge loss function*

$$\ell_h(t) := \max\{0, 1 - t\} \tag{4}$$

one can write the problem (3) equivalently as the unconstrained, but non-smooth problem

$$\min_{\substack{w\in\mathbb{R}^d,\\ b\in\mathbb{R}}} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^M \ell_h\big(y_i\big(\langle w, x_i\rangle + b\big)\big). \tag{5}$$

We note here that we recover the problem (2) in the limit $C \to \infty$.

There exist different smooth approximations to (4), for instance the *logistic loss function*

$$\ell_{\log}(t) := \ln\big(1 + e^{-t}\big).$$

Replacing the hinge loss by this logistic loss in (5) then results in the smooth unconstrained optimisation problem

$$\min_{\substack{w\in\mathbb{R}^d,\\ b\in\mathbb{R}}} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^M \ell_{\log}\big(y_i\big(\langle w, x_i\rangle + b\big)\big), \tag{6}$$

which can by solved by standard non-linear solvers.

2.3. **The Dual Problem.** Since (3) is a convex optimisation problem with linear inequality constraints, it is closely related to its Lagrangian dual. For the formulation of this dual, we introduce the matrices Y, $G \in \mathbb{R}^{M\times M}$ defined as

$$Y = \operatorname{diag}(y_1, \dots, y_M) \qquad \text{and} \qquad G = (\langle x_i, x_j\rangle)_{i,j=1,\dots,M}.$$

With these matrices, one can write the dual as the problem

$$\min_{\alpha\in\mathbb{R}^M} \frac{1}{2}\langle \alpha, YGY\alpha\rangle - \langle 1_M, \alpha\rangle \qquad \text{s.t.} \quad \begin{cases} \langle y, \alpha\rangle = 0, \\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \dots, M. \end{cases} \tag{7}$$

Here $1_M = (1, 1, \dots, 1) \in \mathbb{R}^M$ is the M-dimensional vector containing only 1's. Note here that the matrix G is necessarily positive semi-definite.

[1] To be precise, a point x_i is really only misclassified if $\xi_i > 1$, but as soon as $\xi_i > 0$ we ignore it for the computation of the margin.

Assume now that α^* is a solution of (7). Then we can define

$$I_S := \big\{1 \le i \le M : \alpha_i^* > 0\big\} \qquad \text{and} \qquad S := \big\{x_i : i \in I_S\big\}.$$

The vectors $x_i \in S$ are called the *support vectors* for the problem. Typically, one can expect that the number of support vectors is significantly smaller than the total number of data points. This is also one of the appeals of this approach, as the classifier one obtains at the end has as a result a sparse representation in terms of the data points.

One can show that one can recover the optimal solution (w^*, b^*) of (3) from α^* as

$$w^* = \sum_{i \in I_S} \alpha_i^* y_i x_i \tag{8}$$

and

$$b^* = y_i - \langle w^*, x_i \rangle \qquad \text{for any } i \in \tilde{I}_S = \big\{1 \le i \le M : 0 < \alpha_i^* < C\big\}. \tag{9}$$

3. Nonlinear Classification in RKHS's

3.1. **Reproducing Kernel Hilbert Spaces.** In many situations, the assumption that we can more or less neatly separate the two classes A and B by a hyperplane is not realistic. Because of that, it is necessary to allow the functions $f\colon \mathbb{R}^d \to \mathbb{R}$ that define our classifier to be more general. One successful approach is by working in so called *Reproducing Kernel Hilbert Spaces* (RKHS's). In the following, we will *briefly* introduce/recall the main definitions and theoretical results, and how RKHS's can be applied to classification. Knowledge about the theory of RKHS's is not necessary for completing the project.

Definition 3.1. A *Reproducing Kernel Hilbert Space* (RKHS) is a Hilbert space U consisting of functions $w\colon \mathbb{R}^d \to \mathbb{R}$ such that for all fixed $x \in \mathbb{R}^d$ the mapping from U to $\mathbb{R}$ defined by $w \mapsto w(x)$ is continuous.

Equivalently, U is an RKHS if there exists a function $K\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ (the *kernel* of U) such that

$$w(x) = \langle w, K(x, \cdot) \rangle_U$$

for every $w \in U$ and $x \in \mathbb{R}^d$, where $\langle \cdot, \cdot \rangle_U$ denotes the inner product on U. Conversely, it is possible to show that every suitable[2] kernel defines a unique RKHS.[3]

Examples of suitable kernels are the following:

- The *Gaussian kernel* with *bandwidth* $\sigma > 0$ (or *shape parameter* $1/\sigma$) defined by
$$K(x, y) = e^{-\frac{\|x-y\|_2^2}{2\sigma^2}}.$$
- The *Laplacian kernel* with bandwidth $\sigma > 0$ (or *shape parameter* $1/\sigma$) defined by
$$K(x, y) = e^{-\frac{\|x-y\|_2}{\sigma}}.$$
- The *inverse multiquadric* kernel with bandwidth $\sigma > 0$ and parameter $s > 0$ defined by
$$K(x, y) = \frac{1}{(\sigma^2 + \|x-y\|_2^2)^s}.$$

[2]That is, symmetric and *positive definite*.

[3]That is, given a suitable kernel $K\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, there exists a unique RKHS on $\mathbb{R}^d$ that has this function as kernel.

3.2. **Support Vector Machines.** The idea of *support vector machines* (SVM's) is to replace the affine function $f_{w,b}(x) = \langle w, x\rangle + b$, which we used for linear support vector classification, by a function of the form $f(x) = w(x) + b$, where w is an element of a fixed RKHS U. As in (2) and (3), we then choose the best possible function w by minimising the norm of w subject to the constraint that the classifier $F(x) = \operatorname{sgn}(w(x) + b)$ separates the two classes either strictly or softly.

In the soft margin case, this results in the optimisation problem

$$\min_{\substack{w\in U,\\ b\in\mathbb{R},\\ \xi\in\mathbb{R}^M}} \frac{1}{2}\|w\|_U^2 + C\sum_{i=1}^M \xi_i \qquad \text{s.t.} \quad \begin{cases} y_i\big(w(x_i)+b\big) \geq 1-\xi_i, \\ \xi_i \geq 0, \end{cases} \text{ for } i = 1,\dots,M. \tag{10}$$

The problem with the formulation (10) is that it is an optimisation problem over the *infinite dimensional* Hilbert space U, which at first glance appears to be difficult to solve numerically. However, it turns out that we can again pass to a dual formulation of this problem, which, as we only have finitely many constraints, is again a finite dimensional optimisation problem. Moreover, this dual problem can be formulated purely in terms of the kernel K of the RKHS (and, of course, the given data points).[4]

Having fixed the kernel K of the RKHS U (for instance as the Gaussian kernel $K(x,y) = e^{-\|x-y\|_2^2/2\sigma^2}$ for some fixed bandwidth $\sigma > 0$), we compute the matrix $G \in \mathbb{R}^{M\times M}$,

$$G_{ij} := K(x_i, x_j).$$

We note here that this matrix is necessarily positive definite, if K is any of the kernels introduced in Section 3.1 and all the data points are different (that is, $x_i \neq x_j$ if $i \neq j$). Moreover, we denote again $Y = \operatorname{diag}(y_1,\dots,y_M) \in \mathbb{R}^{M\times M}$ and $1_M := (1,1,\dots,1) \in \mathbb{R}^M$. The dual of (10) is the problem

$$\min_{\alpha\in\mathbb{R}^M} \frac{1}{2}\langle \alpha, YGY\alpha\rangle - \langle 1_M, \alpha\rangle \qquad \text{s.t.} \quad \begin{cases} \langle y, \alpha\rangle = 0, \\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1,\dots,M. \end{cases} \tag{11}$$

Apart from the definition of the matrix G, this is precisely the same problem as (7).

Now assume that $\alpha^* \in \mathbb{R}^M$ is a solution of (11). Denote (again)

$$I_S := \big\{1 \leq i \leq M : \alpha_i^* > 0\big\} \qquad \text{and} \qquad \tilde{I}_S := \big\{1 \leq i \leq M : 0 < \alpha_i^* < C\big\}.$$

Then one can recover the solution (w^*, b^*) of (10) as

$$w^*(x) = \sum_{i\in I_S} \alpha_i^* y_i K(x_i, x)$$

and

$$b^* = y_i - \sum_{j\in\tilde{I}_S} \alpha_j^* y_j K(x_j, x_i) \qquad \text{for any } i \in \tilde{I}_S.$$

Similarly to the case of linear support vector classification, one can expect the set I_S, and thus the number of support vectors, will be *relatively* small. However, in this case, this is compared to the number M of data points. Since we are describing a much more complicated set than in the linear case, the number of support vectors will usually be much larger than the dimension d.

[4]When working in an RKHS, it is extremely common that all the calculations can be performed only in terms of its kernel. This property/feature of RKHS's is often called the *kernel trick*.

4. Numerical Solution of the Dual Problem

We now discuss a numerical method for the solution of the dual problems (7) and (11), which has been developed in [DF06]. In principle, these problems can be solved by any method for general quadratic programs like an active set method. However, as the number of data points, and thus the dimension of the dual problem, increases, such methods may no longer be feasible: Many such methods require the solution of (large) linear systems in each step. Also, typical active set methods, in which only few indices are allowed to enter or leave the active set in each step, become inefficient when the number of constraints becomes too large.

The algorithm in [DF06] is based on a projected gradient descent method. That is, one computes the iterates as

$$\alpha^{(k+1)} = \alpha^{(k)} + d^{(k)},$$

where

$$d^{(k)} = \pi_\Omega(\alpha^{(k)} - \tau_k \nabla f(\alpha^{(k)})) - \alpha^{(k)} \tag{12}$$

is a projected gradient descent update,

$$f(\alpha) := \frac{1}{2}\langle \alpha, YGY\alpha\rangle - \langle 1_M, \alpha\rangle$$

is the function to be minimised,

$$\Omega = \big\{\alpha \in \mathbb{R}^M : \langle y, \alpha\rangle = 0 \text{ and } 0 \le \alpha \le C\big\}$$

is the feasible set, π_Ω denotes the projection onto Ω, and $\tau_k > 0$ is a suitable step length. In addition, it is possible (and sometimes necessary, depending on the choice of τ_k) to include an additional line search and actually compute the update as

$$\alpha^{(k+1)} = \alpha^{(k)} + \theta_k d^{(k)} \tag{13}$$

for some $0 < \theta_k \le 1$.

4.1. Calculation of Projections.

For the projected gradient descent method to work, it is necessary to have an efficient implementation of the projection π_Ω available. In the absence of the constraint $\langle y, \alpha\rangle = 0$, this would be the case, as one could simply calculate the projection of a point α as $\min\{\max\{\alpha, 0\}, C\}$, understood componentwise. However, the additional linear constraint $\langle y, \alpha\rangle = 0$ complicates the situation significantly. Still, it is possible to efficiently calculate an approximate projection.

We first note that the projection $\pi_\Omega(\beta)$ is the solution of the constrained optimisation problem

$$\min_{\alpha\in\mathbb{R}^M} \frac{1}{2}\|\alpha - \beta\|_2^2 \qquad \text{s.t } \begin{cases} \langle y, \alpha\rangle = 0, \\ 0 \le \alpha_i \le C \quad \text{for } i = 1, \dots, M. \end{cases} \tag{14}$$

We now introduce the (partial) Lagrangian

$$\mathcal{L}(\alpha; \lambda) := \frac{1}{2}\|\alpha - \beta\|_2^2 - \lambda\langle y, \alpha\rangle$$

with the single Lagrange parameter $\lambda \in \mathbb{R}$. Now we consider the constrained optimisation problem

$$\min_{\alpha\in\mathbb{R}^M} \mathcal{L}(\alpha; \lambda) \qquad \text{s.t. } 0 \le \alpha \le C. \tag{15}$$

We note that (15) has, for each $\lambda \in \mathbb{R}$ a unique solution. Moreover, by e.g. looking at the KKT conditions for the problems (14) and (15) one sees that α^* solves (14) if and only if α^* solves (15) for some $\lambda \in \mathbb{R}$ such that $\langle y, \alpha^*\rangle = 0$.

Denote now by $\alpha(\lambda)$ the unique solution of (15) for fixed λ. Then solving (14) is equivalent to solving the equation $\langle y, \alpha(\lambda)\rangle = 0$. Now, one can easily show that $\alpha(\lambda)$ has the explicit form

$$\alpha(\lambda)_i = \min\{\max\{\beta_i + \lambda y_i, 0\}, C\} = \operatorname{median}\{0, C, \beta_i + \lambda y_i\}.$$

Moreover, the function $\lambda \mapsto \langle y, \alpha(\lambda)\rangle$ is increasing (though not necessarily strictly increasing). As a consequence, an approximate solution of the equation $\langle y, \alpha(\lambda)\rangle = 0$ can be easily found by a bisection method or similar methods.[5] A rough sketch of such a method might look as follows:

- Choose values $\lambda_- < \lambda_+$ and calculate $\langle y, \alpha(\lambda_-)\rangle$ and $\langle y, \alpha(\lambda_+)\rangle$.
  - If $\langle y, \alpha(\lambda_-)\rangle > 0$, replace λ_- by $\lambda_- - \Delta$ for some $\Delta > 0$ and replace λ_+ by λ_-; continue until $\langle y, \alpha(\lambda_-)\rangle < 0$ and $\langle y, \alpha(\lambda_+)\rangle > 0$.
  - If $\langle y, \alpha(\lambda_+)\rangle < 0$, replace λ_+ by $\lambda_+ + \Delta$ for some $\Delta > 0$ and replace λ_- by λ_+; continue until $\langle y, \alpha(\lambda_+)\rangle > 0$ and $\langle y, \alpha(\lambda_-)\rangle < 0$.
- Compute $\hat\lambda = \frac{1}{2}(\lambda_- + \lambda_+)$.
  - If $|\langle y, \alpha(\hat\lambda)\rangle| < \varepsilon$, stop the iteration.
  - If $\langle y, \alpha(\hat\lambda)\rangle < 0$, replace λ_- by $\hat\lambda$, and continue the iteration.
  - If $\langle y, \alpha(\hat\lambda)\rangle > 0$, replace λ_+ by $\hat\lambda$, and continue the iteration.

This approach can easily be made more efficient by refining the selection method for the next iterate $\hat\lambda$. Instead of choosing the midpoint $(\lambda_- + \lambda_+)/2$ of the current interval, one can use a linear interpolation through the points $(\lambda_-, \langle y, \alpha(\lambda_-)\rangle)$ and $(\lambda_+, \langle y, \alpha(\lambda_+)\rangle)$ and choose $\hat\lambda$ as the parameter for which this linear interpolant becomes 0 (this is the *regula falsi* method). However, one has to be somewhat careful, as it can happen that this refined method only converges sublinearly. In order to avoid this, one needs to ensure that the length of the interval $[\lambda_-, \lambda_+]$ decreases in each step by at least a constant factor by increasing or decreasing $\hat\lambda$ if necessary.

4.2. **Step Length Selection.** One possibility for defining the step length τ_k in (12) is by using exact line search. That is, one chooses τ_k adaptively such that $f(\alpha^{(k)})$ becomes minimal. (This is the usual approach taken in gradient projection methods.) Since the function $\tau \mapsto f(\pi_\Omega(\alpha^{(k)} - \tau\nabla f(\alpha^{(k)})))$ is piecewise quadratic, this is theoretically possible, but may come at a high computational cost in high dimensions.

As an alternative, one can use a fixed step size based on the behaviour of the iteration during the previous updates. For this, we denote

$$s^{(k)} := \alpha^{(k+1)} - \alpha^{(k)} \qquad \text{and} \qquad z^{(k)} := \nabla f(\alpha^{(k+1)}) - \nabla f(\alpha^{(k)}).$$

The (long) *Barzilai–Borwein step length* is then given as

$$\tau_{k+1}^{BB} := \frac{\langle s^{(k)}, s^{(k)}\rangle}{\langle s^{(k)}, z^{(k)}\rangle}. \tag{16}$$

In case $\langle s^{(k)}, z^{(k)}\rangle \le 0$, this does not yield a well-defined (positive!) step length. In this situation, one instead has to switch to some fixed, pre-defined step length.

More generally, in order to avoid both too small and too large step lengths, the procedure is as follows: At the start of the iteration, one chooses step length bounds $\tau_{\min} < \tau_{\max}$ (e.g. $\tau_{\min} = 10^{-5}$ and $\tau_{\max} = 10^5$ in order to cover a large range of possible step lengths). After iteration k, one then computes the Barzilai–Borwein step length τ_{k+1}^{BB} according to (16). If $\langle s^{(k)}, z^{(k)}\rangle \le 0$, one then sets $\tau_{k+1} := \tau_{\max}$. Else, one defines $\tau_{k+1} := \min\{\max\{\tau_{k+1}^{BB}, \tau_{\max}\}, \tau_{\min}\}$.

[5] Note, though, that the function $\lambda \mapsto \langle y, \alpha(\lambda)\rangle$ is not differentiable (but rather piecewise linear). Newton's method might therefore be a bad choice.

A slightly more advanced step length choice can be obtained by replacing τ_{k+1}^{BB} in this procedure by

$$\tau_{k+1}^{(2)} := \frac{\langle s^{(k)}, s^{(k)} \rangle + \langle s^{(k-1)}, s^{(k-1)} \rangle}{\langle s^{(k)}, z^{(k)} \rangle + \langle s^{(k-1)}, z^{(k-1)} \rangle}$$

in cases where both $\langle s^{(k)}, z^{(k)} \rangle > 0$ and $\langle s^{(k-1)}, z^{(k-1)} \rangle > 0$.

4.3. **Line Search.** The step length selection method described in Section 4.2 may lead to a divergent iteration, as we never verify that the function values actually decrease. Because of that, it is necessary to introduce an additional line search as described in (13) in order to guarantee convergence of the algorithm. For the line search, there are two decisions to make: First, we have to decide if we need to perform a line search or if we are satisfied with the step length $\theta_k = 1$. Second, in case we decide to perform a line search, we need to decide how to choose θ_k.

We will start with the second question. Since the function f is quadratic, we can actually perform an exact line search here. Thus, *if* we choose to perform a line search, we define θ_k as the solution of the problem $\min_{0 \leq \theta \leq 1} f(\alpha^{(k)} + \theta d^{(k)})$. The solution of this problem can easily be computed analytically. Moreover, since by construction $d^{(k)}$ is a descent direction for f, we necessarily have that $\theta_k > 0$. Note also that the condition $0 < \theta \leq 1$ implies that $\alpha^{(k)} + \theta_k d^{(k)}$ is feasible for our problem.

Now the question is, *when* to perform a line search. One possibility is, to simply apply a line search all the time. A second possibility is to apply a line search when $f(\alpha^{(k)} + d^{(k)}) > f(\alpha^{(k)})$, that is, when a step with step length 1 would lead to an increase in function value. Both of these methods are possible and lead to convergence methods, but they can be somewhat slow in practice. A better possibility is to actually allow the function values to increase sometimes, as long as this does not happen too often.

For this, one keeps track of a reference function value f_{ref}, which will be updated during runtime, and performs a line search whenever $f(\alpha^{(k)} + d^{(k)}) > f_{\mathrm{ref}}$. At the start of the iterations, one sets $f_{\mathrm{ref}} := +\infty$. In addition, one denotes $f_{\mathrm{best}} := f(\alpha^{(0)})$ and $f_c := f_{\mathrm{best}}$ the current best function value and a candidate for a new value of f_{ref}. In addition, we choose a number L (e.g. $L = 10$), which denotes how many iterations in a row we are allowed to perform without decreasing f_{best} before we need to reduce f_{ref}. Moreover, we set a counter of such iterations to $\ell = 0$. After the computation of $f_k := f(\alpha^{(k)})$ we then update the values f_{ref}, f_{best}, and f_c as follows:

- If $f_k < f_{\mathrm{best}}$, then we set $f_{\mathrm{best}} := f_k$, $f_c = f_k$, and $\ell = 0$.
- If $f_k \geq f_{\mathrm{best}}$, we set $f_c = \max\{f_c, f_k\}$, $\ell = \ell + 1$.
- If $\ell = L$, we set $f_{\mathrm{ref}} = f_c$, $f_c = f_k$, and $\ell = 0$.

In addition to performing a line search when $f(\alpha^{(k)} + d^{(k)}) > f_{\mathrm{ref}}$, we have to perform a line search at the start of the algorithm (that is, for $k = 0$) since we have no control over the quality of the initial step length τ_0.

5. Project Tasks

The goals of this project are the theoretical analysis of the optimisation problems (3), (7), and (11) and the implementation and discussion of numerical methods for support vector classification.

At the minimum, you should:

- Discuss existence and uniqueness of the problems (3) and (7).
- Show that the problem (7) indeed is the Lagrangian dual of the problem (3), and that one obtains the solution of the primal problem (3) with the formulas (8) and (9).
- Formulate and implement a numerical method that constructs a classifier for a data set by (approximately) solving one of the equivalent optimisation problems discussed above (or approximations thereof). The method described in Section 4 is an example of a method that is not too difficult to implement and that scales relatively well to larger data sets: It should still run in reasonable time for $M = 10000$; the dimension d is not that important for the iterations in the algorithm, though the construction of the matrix G becomes computationally more expensive. For small data sets, though more standard methods for quadratic programs might be significantly faster.
- Explain your choice of a numerical method, describe the method in sufficient detail, and discuss its numerical behaviour.

After that, you are free to play with support vector classifiers and to steer the project in a direction of your choice. Some non-exhaustive suggestions of mine are:

- Implement solution algorithms for both the primal and the dual (linear) soft margin classifier problems and discuss the differences (e.g. challenges in implementation; scaling to large data sets; properties of the solution).
- Compare different solution methods for the dual problem and discuss the differences. Discuss also the differences between the linear and the non-linear classifier in terms of behaviour of the algorithms.
- Discuss (and try to explain!) in more detail the influence of parameter settings on the behaviour of the implemented algorithm. Keep in mind, that this behaviour might depend on the properties of the data set.
- Fill in all the theoretical details left out in the description of the optimisation method in Section 4, in particular concerning the calculation of projections. Also discuss the reasoning behind the algorithmic choices made in this approach.

Again, these are only suggestions. The only limit you have are *12 pages*. Do not try to do too many things at once; it is better to focus on one direction and follow it thoroughly than to shallowly discuss many different ones. Note also that this is a project in optimisation (and not machine learning), and that you should concentrate on optimisation related topics.

In order to produce test data and to verify that your algorithm works, you can use the python code on the wiki page. Given a vector $w \in \mathbb{R}^d$, an offset $b \in \mathbb{R}$, a margin $\gamma > 0$, this code produces samples separated by the hyperplane $H(w, -b)$ and the precise margin γ. A convergent algorithm for linear support vector classification should be able to reproduce (up to some positive scaling of w and b!) this precise hyperplane and margin provided that the parameter C in the soft classifier is sufficiently large. Be aware that the convergence can/will be slow if the number of samples becomes large. You are, of course, welcome to apply your implementation on other data sets as well. Classic applications for support vector machines can be for instance found in image classification.

Practical Information.

- Write a report of *no more than 12 pages* (including all figures, tables, and possible references). This report should be written as an actual scientific/mathematical report, not simply as an unconnected list of formulas interspersed with some random numerical results. Roughly 25% of the project grade will be based on the presentation.
- Submit a zip-file containing your relevant code. Use a jupyter notebook to demonstrate how your algorithm works by presenting some numerical results, but move the bulk of your functions to separate `.py` files and import them to the notebook. Include enough documentation in the code so that we have a chance of understanding what you are doing.
- There is no need to cite this project description as a source; neither is it necessary to provide citations for theoretical results or numerical algorithms discussed in the lecture notes accompanying this course. If you use results or techniques from other sources, though, they have to be cited.
- If you use generative AI (ChatGPT, GitHub copilot,...) at some point during the project, state this clearly in a separate section in the project report. That is, state which tools you have used and for which purpose, and be as specific as possible. This also applies, if you have used generative AI for coding or for generating figures. Remember that this is *your* project report, and that you own all the errors that AI makes.
- In your project report, you do not need to repeat how one arrives at the (primal) hard and soft margin classification problems. It is sufficient to state that these are the problems we are interested in solving. It might make sense, though, to include a few sentences concerning classification and the set-up of linear (and non-linear, if relavant) classifiers in the introduction of your report.
- Add more documentation to your code!

References

[DF06] Yu-Hong Dai and Roger Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Math. Program.*, 106(3):403–421, 2006.

[Ste15] Gabriele Steidl. Supervised learning by support vector machines. In *Handbook of mathematical methods in imaging. Vol. 1, 2, 3*, pages 1393–1453. Springer, New York, 2015.

Trondheim, Norway