Contents

1 Application to Model Selection for Primal SVM

Skalarprodukt anpassen, Vektoren nicht fett oder neue definition, notation, $\lambda \in \Lambda$ einfugen

1.1 Introduction

In this part of the thesis the nonconvex inexact bundle algorithm is applied to the problem of model selection for *support vector machines* (SVMs) solving classification tasks. It relies on a bilevel formulation proposed by Kunapuli in [6] and Moore et al. in [8].

A natural application for the inexact bundle algorithm is an optimization problem where the objective function value can only be computed iteratively. This is for example the case in bilevel optimization.

A general bilevel program can be formulated as in [6, p. 20]

$$\begin{aligned} \min_{x \in X, y} \quad & F(x,y) && \text{upper level} \\ \text{s.t.} \quad & G(x,y) \leq 0 && \\ & y \in \left\{ \begin{array}{ll} \underset{y \in Y}{\arg\max} & f(x,y) \\ \text{s.t.} & g(x,y) \leq 0 \end{array} \right\}. && \text{lower level} \end{aligned} \tag{1.1}$$

It consists of an *upper* or *outer level* which is the overall function to be optimized. Contrary to usual constrained optimization problems which are constrained by explicitly given equalities and inequalities a bilevel program is additionally constrained by a second optimization problem, the *lower* or *inner level* problem.

Solving bilevel problems can be divided roughly in two classes: implicit and explicit solution methods. In the explicit methods the lower level problem is usually rewritten by its KKT conditions, these are then added as constraints to the upper level problem. With this solution method the upper and lower level are solved simultaneously. For the setting of model selection for support vector machines as it is used here, this method is described in detail in [6].

The second approach is the implicit one. Here the lower level problem is solved directly in every iteration of the outer optimization algorithm and the solution is plugged into the upper level objective.

Obviously if the inner level problem is solved numerically, the solution cannot be exact. Additionally the *solution map* $S(x) = \{y \in \mathbb{R}^k \mid y$, that solves the lower level problem, is can be nondifferentiable [9] and since elements of the solution map are plugged into the outer level objective function in the implicit approach, the outer level function then becomes nonsmooth itself.This is why the inexact bundle algorithm seems a natural choice to tackle these bilevel problems.

Moore et al. use the implicit approach in [8] for support vector regression. However they use a gradient decent method which is not guaranteed to stop at an optimal solution. In [7] he also suggests the nonconvex exact bundle algorithm of Fuduli et al. [3] for solving the bilevel regression problem. This allows for nonsmooth inner problems and can theoretically solve some of the issues of the gradient descent method. It ignores however, that the objective function values can only be calculated approximately. A fact which is not addressed in Fuduli's algorithm.

1.2 Introduction to Support Vector Machines

Support vector machines are linear learning machines that were developed in the 1990's by Vapnik and co-workers. Soon they could outperform several other programs in this area [2] and the subsequent interest in SVMs lead to a very versatile application of these machines [6].

The case that is considered here is binary support vector classification using supervised learning. For a throughout introduction to this subject see also [2]. Here a summary of the most important expressions and results is given.

In classification data from a possibly high dimensional vector space $\tilde{X} \subset \mathbb{R}^n$, the *feature* or *input space* is divided into two classes. These lie in the *output domain* $\tilde{Y} = \{-1, 1\}$. Elements from the feature space will mostly be called *data points* here. They get *labels* from the feature space. Labeled data points are called *examples.* The functional relation between the features and the class of an example is given by the usually unknown *response* or *target function* $f(x)$. Supervised learning is a kind of machine learning task where the machine is given examples of input data with associated labels, the so called *training data* (X, Y). Mathematically this can be modeled by assuming that the examples are drawn identically and independently distributed (iid) from the fixed joint distribution $P(x, y)$. This usually unknown distribution states the probability that a data point x has the label y [13, p. 988]. The overall goal is then to optimize the generalization ability, meaning the ability to predict the output for unseen data correctly [2, chapter 1.2].

1.2.1 Risk minimization

The concept of SVM's was originally inspired by the statistical learning theory developed by Vapnik. A detailed examination of the subject is given in [12]. In [14] the subject is approached from a more explaining point of view.

The idea of *risk minimization* is to find from a fixed set or class of functions the one that is the best approximation to the response function. This is done by minimizing a loss function that compares the given labels of the examples to the response of the learning machine.

As the response function is not known only the expected value of the loss can be calculated. It is given by the *risk functional*

$$R(\lambda) = \int \mathcal{L}(y, f_\lambda(x))\mathrm{d}P(x, y). \tag{1.2}$$

Here $\mathcal{L} : \mathbb{R}^2 \to \mathbb{R}$ is the loss function, $f_\lambda : \mathbb{R}^n \cap \mathcal{F} \to \mathbb{R},\ \lambda \in \Lambda$ the approximate response function found by the learning machine and $P(x, y)$ the joint distribution the training data is drawn from. The goal is now to find a function $f_{\bar{\lambda}}(x)$ in the chosen function space $\mathcal{F}$ that minimizes this risk functional [13, 989].

As the only given information is provided by the training set inductive principles are used to work with the *empirical risk*, rather than with the risk functional. The empirical risk only depends on the finite training set and is given by

$$R_{\mathrm{emp}}(\lambda) = \frac{1}{l}\sum_{i=1}^{l} \mathcal{L}(y_i, f_\lambda(x^i)), \tag{1.3}$$

where l is the number of data points. The law of large numbers ensures that the empirical risk converges to the risk functional as the number of data points grows to infinity. This however does not guarantee that the function $f_{\lambda,\mathrm{emp}}$ that minimizes the empirical risk also converges towards the function $f_{\bar{\lambda}}$ that minimizes the risk functional. The theory of consistency provides necessary and sufficient conditions that solve this issue [13, p. 989].

Vapnik therefore introduced the structural risk minimization (SRM) induction principle . It ensures that the used set of functions has a structure that makes it strongly consistent [13]. Additionally it takes the complexity of the function that is used to approximate the target function into account. “The SRM principle actually suggests a tradeoff between the quality of the approximation and the complexity of the approximating function” [13, p. 994]. This reduces the risk of *overfitting*, meaning to overly fit the function to the training data with the result of poor generalization [2, chapter 1.3].

Support vector machines fulfill all conditions of the SRM principle. Due to the kernel trick that allows for nonlinear classification tasks it is also very powerful. For more detailed information on this see [6] and references therein.

1.2.2 Support Vector machines

In the case of linear binary classification one searches for a an affine hyperplane $w \in \mathbb{R}^n$ shifted by $b \in \mathbb{R}$ to separate the given data. The vector w is called weight vector and b is the bias.

Let the data be linearly separable. The function deciding how the data is classified can then be written as

$$f(x) = \operatorname{sign}(\langle w, x \rangle - b).$$

Support vector machines aim at finding such a hyperplane that separates also unseen data optimally.

???Picture of hyperplane

One problem of this intuitive approach is that the representation of a hyperplane is not unique. If the plane described by (w, b) separates the data, there exist infinitely many hyperplanes (tw, b), $t > 0$, that separate the data in the same way. To have a unique description of a separating hyperplane the *canonical hyperplane for given data $x \in X$* is defined by

$$f(x) = \langle w, x \rangle - b \quad \text{s.t.} \quad \min_i | \left\langle w, x^i \right\rangle - b| = 1.$$

This is always possible in the case where the data is linearly separable and means that the inverse of the norm of the weight vector is equal to the distance of the closest point $x \in X$ to the hyperplane [6, p. 10].

This gives rise to the following definition: The *margin* is the minimal Euclidean distance between a training example x^i and the separating hyperplane. A bigger margin means a lower complexity of the function [2].

A *maximal margin hyperplane* is the hyperplane that realizes the maximal possible margin for a given data set.

Proposition 1.1 ([2, Proposition 6.1]) Given a linearly separable training sample $\Omega = \{(x^i, y_i), ..., (x^l, y_l)\}$ the hyperplane (w, b) that solves the optimization problem

$$\|w\|^2 \quad \text{s.t.} \quad y_i(\langle w, x\rangle - b) \geq 1, \quad i = 1, ..., l,$$

realizes a maximal margin hyperplane.

The proof is given in [2, chapter 6.1].

Generally one cannot assume the data to be linearly separable. This is why in most applications a so called *soft margin classifier* is used. It introduces the slack variables ξ_i that measure the distance of the misclassified points to the hyperplane:

Fix $\gamma > 0$. A *margin slack variable of the example* (x^i, y_i) with respect to the hyperplane (w, b) and target margin γ is

$$\xi_i = \max(0, \gamma - y_i(\langle w, x\rangle + b))$$

If $\xi_i > \gamma$ the point is considered misclassified. One can also say that $\|\xi\|$ measures the amount by which training set "fails to have margin γ" [2].

For support vector machines the target margin is set to $\gamma = 1$.

This results finally in the following slightly different optimization problems for finding an optimal separating hyperplane (w, b):

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{l} \xi_i \\ \text{s.t.} \quad & y_i(\left\langle w, x^i \right\rangle - b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \\ & \forall i = 1, \ldots, l \end{aligned} \tag{1.4}$$

and

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{l} \xi_i^2 \\ \text{s.t.} \quad & y_i(\left\langle w, x^i \right\rangle - b) \geq 1 - \xi_i \\ & \forall i = 1, \ldots, l. \end{aligned} \tag{1.5}$$

The first part of the respective objective functions are the regularizations, the second part are the actual loss functions. The parameter $C > 0$ gives a trade-off between the richness of the chosen set of functions f_λ to reduce the error on the training data and the danger of overfitting to have good generalization. It has to be chosen a priori [6]. The two optimization problems only differ in the norm chosen for the loss function. In (1.4) the one-norm is chosen, in (1.5) the squared two-norm is used. Problem (1.5) is the one that is finally used in the bilevel approach where smoothness of the objective function of the inner level problem is needed to calculate all needed subgradients.

1.3 Bilevel Approach and Inexact Bundle Method

The hyper-parameter C in the objective function of the classification problem has to be set beforehand. This step is part of the model selection process. To set this parameter optimally different methods can be used. A very intuitive and widely used approach is doing and *cross validation* (CV) with a grid search implementation.

To prevent overfitting and get a good parameter selection, especially in case of little data, commonly T-fold cross validation is used. For this technique the training data is randomly partitioned into T subsets of equal size. One of these subsets is then left out of the training set and instead used afterwards to get an estimate of the generalization error. To use CV for model selection it has to be embedded into an optimization algorithm over the hyper-parameter space. Commonly this is done by discretizing the parameter space and for T-fold CV training T models at each grid point. The resulting models are then compared to find the best parameters in the grid. Obviously for a growing number of hyper-parameters this is very costly. An additional drawback is that the parameters are only chosen from a finite set [6, p. 30].

1.3.1 Reformulation as Bilevel Problem

A more recent approach is the formulation as a bilevel problem used in [6] and [8]. This makes it possible to optimize the hyper-parameters continuously.

Let $\Omega = \{(x^1, y_1), ..., (x^l, y_l)\} \subset \mathbb{R}^{n+1}$ be a given data set of size $l = |\Omega|$. The associated index set is denoted by $\mathcal{N}$. For classification the labels y_i are ± 1. For T-fold cross validation let $\bar{\Omega}_t$ and Ω_t be the training set and the validation set respectively within the t'th fold and $\bar{\mathcal{N}_t}$ and $\mathcal{N}_t$ the respective index sets. Furthermore let $f^t : \mathbb{R}^{n+1} \cap \mathcal{F} \to \mathbb{R}$ be the response function trained on the t'th fold and $\lambda \in \Lambda$ the hyper-parameters to be optimized. For a general machine learning problem with upper and lower loss function $\mathcal{L}_{upp}$ and $\mathcal{L}_{low}$ respectively the bilevel problem reads

$$\begin{aligned}
\min_{\lambda, f^t} \quad & \mathcal{L}_{upp}\left(\lambda, f^1|_{\Omega_1}, ..., f^T|_{\Omega_T}\right) && \text{upper level} \\
\text{s.t.} \quad & \lambda \in \Lambda \\
& \\
& \text{for } t = 1, ..., T : \\
& f^t \in \left\{ \begin{matrix} \underset{f \in \mathcal{F}}{\arg\min} & \mathcal{L}_{low}(\lambda, f, (x^i, y_i)_{i=1}^l \in \bar{\Omega}_t) \\ \text{s.t.} & g_{low}(\lambda, f) \leq 0 \end{matrix} \right\}. && \text{lower level}
\end{aligned} \tag{1.6}$$

In the case of support vector classification the T inner problems have the classical SVM formulation (1.5). The problem can also be rewritten into an unconstrained form. This form is helpful when using the inexact bundle algorithm for solving the bilevel problem. For the t'th fold the resulting hyperplane is identified with the pair $(w^t, b_t) \in \mathbb{R}^{n+1}$. The inner level problem for the t'th fold can therefore be stated as

$$(w^t, b_t) \in \underset{w,b}{\arg\min} \left\{ \frac{\lambda}{2} \|w\|_2^2 + \sum_{i \in \bar{\mathcal{N}_t}} \max \left\{ 1 - y_i \left(\left\langle w, x^i \right\rangle - b \right), 0 \right\}^2 \right\} \tag{1.7}$$

Where the hyper-parameter $\lambda = \frac{1}{C}$ is used due to numerical stability [6, p. 38].

For the upper level objective function there are different choices possible. Simply put the outer level objective should compare the different inner level solutions and pick the best one. An intuitive choice is therefore to pick the misclassification loss, that counts how many data points of the respective validation set Ω_t are misclassified when taking function f^t.

The misclassification loss can be written as

$$\mathcal{L}_{mis} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\left[-y_i\left(\left\langle w^t, x^i\right\rangle - b_t\right)\right]_\star, \tag{1.8}$$

where the step function $(\cdot)_\star$ is defined component wise for a vector as

$$(r_\star)_i = \begin{cases} 1, & \text{if } r_i > 0, \\ 0, & \text{if } r_i \leq 0 \end{cases}. \tag{1.9}$$

The drawback of this simple loss function is that it is not continuous and as such not suitable for subgradient based optimization. Therefore another loss function is used for the upper level problem - the *hinge loss*. It is an upper bound on the misclassification loss and reads

$$\mathcal{L}_{hinge} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\max\left(1 - y_i\left(\left\langle w^t, x^i\right\rangle - b_t\right), 0\right). \tag{1.10}$$

It is also possible to square the max term. This results in the loss function

$$\mathcal{L}_{hinge} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\max\left\{1 - y_i\left(\left\langle w^t, x^i\right\rangle - b_t\right), 0\right\}^2. \tag{1.11}$$

In figure (**??**) it can be seen that its minimum and overall progress is more similar to the misclassification loss than the one of the hinge loss. For this reason we progress taking the squared form of the hinge loss, abbreviating with *hingequad loss* for convenience. No, take hingeloss because of nonsmoothness

Hence the final resulting bilevel formulation for model selection in support vector classification is

$$\begin{aligned} \min_{w,b} \quad & \mathcal{L}_{hinge}(w,b) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\max\left\{1 - y_i\left(\left\langle w^t, x^i\right\rangle - b_t\right), 0\right\}^2 \\ \text{s.t.} \quad & \lambda > 0 \\ & \text{for } t = 1, \dots, T \\ & (w^t, b_t) \in \arg\min_{w,b}\left\{\frac{\lambda}{2}\|w\|_2^2 + \sum_{i\in\bar{\mathcal{N}}_t}\max\left\{1 - y_i\left(\left\langle w, x^i\right\rangle - b\right), 0\right\}^2\right\}. \end{aligned} \tag{1.12}$$

1.3.2 Solution of the Bilevel Program

1.4 Numerical Experiments

In this section algorithm ??.1 is used to solve the bilevel problems presented above for different synthetic and real world data sets.

- Problem what to optimize
- comment on nonlinear
- overfitting
- pictures that show situation

1.4.1 Selection of the Data Sets

- parameter λ against overfitting, there to make generalization better
- data sets that "allow" for high overfitting: not too big, many features

The following two real world data sets are used:

Data set	l_{train}	l_{test}	n	T
Wisconsin Breast Cancer Database	240	443	9	3
John Hopkins University Ionosphere Database	240	111	33	3

Table 1

Plots

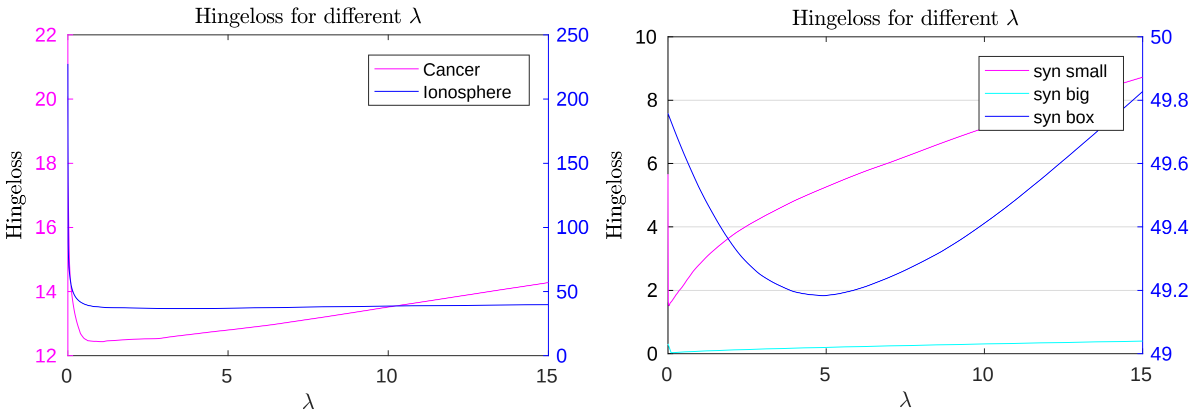

Figure 1: *Plots of the hingloss error for different λ values. The figure on the right shows the plot for the* cancer *(left axis) and* ionosphere *(right axis) data sets. The blot on the left depicts the situation for the sythetic sets* syn small*,* syn big *(left axis) and* syn box *(right axis).*

results

Data set	algorithm	0.1	1	10	100
cancer	Bundle	1.0975	1.0974	1.0974	1.1341
	Noll Bundle	1.0974	1.0974	1.0974	1.0974
	fminsearch	1.0974	1.0974	1.0974	-10
	fmindnb	1.0974			
ionosphere	Bundle	2.6081	2.3108	0.3953	4.7401
	Noll Bundle	3.5386	3.6521	10	8.2773
	fminsearch	3.5104	3.5104	3.5104	3.5104
	fminbnd	3.5104			
syn small	Bundle	0.0337	0.031	0.0329	0.03
	Bundle Noll	0.0323	0.0352	0.0352	0
	fminsearch	0.0346	-0.1939	-1.79	-3.5673
	fminbnd	0.0346			
syn big	Bundle	0.1	1	10	100
	Bundle Noll	0.0099	0.0108	0.0109	0.0086
	fminsearch	0.0114	-0.8748	-0.8748	-0.8748
	fminbnd	0.0206	plot: 0.0114		
syn box	Bundle	0.1	1	4.8486	4.45336
	Bundle Noll	0.1	1	4.2760	0
	fminsearch	4.8950	4.8950	4.8950	4.8950
	fminbnd	4.8950			

Table 2

times

Data set	algorithm	0.1	1	10	100
`syn big`	Bundle Noll	202	320	386	503
	fminsearch	672	637	587	741
	fminbnd	557			

Table 3

sometimes says problem not convex, then negative values. Why??? Because tries negative lambda values in course of optimization and then it gets negative
sometimes even for negative lambda in the end right solution
can I find out why bundle methods so bad?
try matlab method (smooth) with subgradient???

1.4.2 Solution of the Bilevel Program

ab hier: Theorie fehlt
!!! notation - oder in prelininaries einfugen

To solve the given bilevel problem with the above presented nonconvex inexact bundle algorithm the algorithm jumps between the two levels. Once the inner level problems are solved for a given λ - this is possible with any QP-solver as the problems are convex - the bundle algorithm takes the outcome w and b and optimizes the hyper-parameter again.

The difficulty with this approach is that the bundle algorithm needs one subgradient of the outer level objective function with respect to the parameter λ. However to compute this subgradient also one subgradient of w and b with respect to λ has to be known.

The Differentiable Case example in differentiable case
Let us first assume that the outer and inner objective functions and $w(\lambda) = \arg\min \mathcal{L}_{low}(w, \lambda)$ are sufficiently often continuously differentiable to demonstrate the procedure of calculating the needed (sub-)gradients.

Let $\mathcal{L}_{upp}(w, \lambda)$ be the objective function of the outer level problem, where the variable b was left out for the sake of simplicity. To find an optimal hyper parameter λ given the input w the gradient g_λ^{upp} of $\mathcal{L}_{upp}$ with respect to λ is needed in every iteration of the solving algorithm. In order to calculate this gradient the chain rule is used yielding

$$g_\lambda^{upp} = \left(\frac{\partial}{\partial w}\mathcal{L}_{upp}(w, \lambda)\right)^\top \frac{\partial w(\lambda)}{\partial \lambda} + \frac{\partial}{\partial \lambda}\mathcal{L}_{upp}(w, \lambda).$$

The challenge is here to find the term $\frac{\partial w(\lambda)}{\partial \lambda}$ because

$$\frac{\partial w}{\partial \lambda} \in \frac{\partial}{\partial \lambda} \arg\min_{w} \mathcal{L}_{low}(w, \lambda).$$

Assuming $\mathcal{L}_{low}$ is twice continuously differentiable at the optimal solution w^* of the lower level problem the optimality condition for any parameter $\lambda_0 > 0$

$$0 = \frac{\partial}{\partial w} \mathcal{L}_{low}(w^*, \lambda_0) \tag{1.13}$$

can be used to calculate the needed gradient in an indirect manner.

In the differentiable case the theoretical framework for the following calculations is given by the implicit function theorem.

Theorem 1.2 (c.f. [5, chapter 3.4]) *Let $F : U \times V \to Z$, $U \in \mathbb{R}^m, V, Z \in \mathbb{R}^n$, be a $\mathcal{C}^1$ mapping, $(x_0, y_0) \in U \times V$ and $F(x_0, y_0) = 0$. If the matrix $\frac{\partial}{\partial y} F(x_0, y_0)$ is invertible, there exist neighborhoods $U_0 \subset U$ of x_0 and $V_0 \subset V$ of y_0 and a continuously differentiable mapping $f : U_0 \to V_0$ with*

$$F(x, y) = 0, (x, y) \in U_0 \times V_0 \quad \Leftrightarrow \quad y = f(x), x \in U_0.$$

Identifying $x \mathrel{\hat{=}} \lambda$, $y \mathrel{\hat{=}} w$ and $F(x_0, y_0) \mathrel{\hat{=}} \frac{\partial}{\partial w} \mathcal{L}_{low}(w^*, \lambda_0)$ and assuming $\frac{\partial^2}{\partial w^2} \mathcal{L}_{low}(w^*, \lambda_0)$ is invertible this theorem provides the existence of the continuously differentiable function $w(\lambda)$ whose gradient is needed.
what about the neighborhoods?

If the inner level loss function yields a linear optimality condition in w it is possible to calculate the gradient explicitly. This is for example the case for SVM loss functions with a squared one- or two-norm as given in problem (1.5). The optimality condition can then be written as the linear system

$$H(\lambda) w = h(\lambda).$$

By taking the partial derivative with respect to λ on both sides of the system one gets

$$\frac{\partial H(\lambda)}{\partial \lambda} w + H(\lambda) \frac{\partial w}{\partial \lambda} = \frac{\partial h(\lambda)}{\partial \lambda}.$$

If $H(\lambda)$ is invertible for all $\lambda \in \Lambda$ then the needed gradient is given by

$$\frac{\partial w}{\partial \lambda} = H^{-1}(\lambda)\left(\frac{\partial h(\lambda)}{\partial \lambda} - \frac{\partial H(\lambda)}{\partial \lambda} w\right).$$

The Nondifferentiable Case now for subgradients

In practice we cannot expect $\mathcal{L}_{low}$ to satisfy such strong differentiability properties. It is therefore only assumed that $\mathcal{L}_{low}$ is once continuously differentiable in w. This assures that the optimality condition of the lower level problem is an equality like in (1.13). Contrary to the exemplary calculations from above in practice the second derivative $\frac{\partial^2}{\partial w \partial \lambda}\mathcal{L}_{low}(w(\lambda), \lambda)$ however is not existent in this form, but a set of subgradients.

Notation

First the theoretical framework given to derive the results from above in the nondifferentiable case

An important result about Lipschitz functions is Rademacher's theorem which states that these functions are differentiable almost everywhere but on a set of Lebesgue measure zero[4, Theorem 3.1]. Clarke deduces from this that the subdifferential at each of the nondifferentiable points is the convex hull of the limits of the sequence gradients a these points [1, see Theorem 2.5.1].

This motivates the multidimensional definition of Clake's generalized gradient

Definition 1.3 ([1, Definition 2.6.1]) *generalized Jacobian*: $F : \mathbb{R}^n \to \mathbb{R}^m$, with locally Lipschitz component functions $F(x) = (f_1(x), ..., f_m(x))$.
Denote generalized Jacobian by $\partial F(x) = conv\,(\lim JF(x_i) | x_i \to x, x_i \notin \Omega_F)$ where Ω_F is the set of nondifferentiable points of F

(after that comes proposition with properties of ∂F)

To facilitate readability we use the following notation for the derivation of the nondifferentiable results.

The *'partial' subdifferential* of a function $f(a^*, b_0, c_0, ...)$ at the point a^* with respect to one variable a when all other variables are fixed is denoted by

$$\partial^a f(a^*, b_0, c_0, ...).$$

A subgradient of this subdifferential is written $g^a \in \partial^a f(a^*, b_0, c_0, ...)$.

Next step: show that chain rule is still valid in the nonsmooth case Chain rules for subdifferential

Theorem 1.4 ([1, Theorem 2.6.6]) *Let $f(x) = \phi(F(x))$, with the locally Lipschitz functions $F : \mathbb{R}^n \to \mathbb{R}^m$ and $\phi : \mathbb{R}^m \to \mathbb{R}$. Then f is locally Lipschitz and it holds*

$$\partial f \subset conv\{\partial\phi(F(x))\partial F(x)\}\,.$$

If in addition ϕ is strictly differentiable at $F(x)$, then equality holds.

strictly differentiable: c.f. [11, Theorem 9.17 and 9.18] locally Lipschitz continuous and at most one subgradient at the point in question (see also comment to Definition 91 in [11])

Theorem 1.5 (c.f. [10, Theorem 7.1]) *Let $p(x) = f(F(x))$, where $F : \mathbb{R}^n \to \mathbb{R}^d$ is locally Lipschitz and $f : \mathbb{R}^d \to \mathbb{R}$ is lower semicontinuous. Assume*

$$\nexists y \in \partial^\infty f(F(\bar{x})), y \neq 0 \quad with \quad 0 \in y\partial F(\bar{x}).$$

Then for the sets

$$M(\bar{x}) := \partial f(F(\bar{x}))\partial F(\bar{x}), \quad M^\infty(\bar{x}) := \partial^\infty f(F(\bar{x}))\partial F(\bar{x}),$$

one has $\hat{\partial} p(\bar{x}) \subset M(\bar{x})$ and $\hat{\partial}^\infty p(\bar{x}) \subset M^\infty(\bar{x})$.

The main idea is to replace the inner level problem by its optimality condition

$\partial(w, b)$ means in this case that the subdifferential is taken with respect to the variables w and b.
-> theory for subdifferentials in more than one variable!!!

For convex inner level problem this replacement is equivalent to the original problem.

The difference to the approach described in [6] is that the problem is not smoothly replaced by its KKT conditions but only by this optimality condition. The weight vector w and bias b are treated as a function of λ and are optimized separately from this hyperparameter. The reformulated bilevel problem becomes:

$$\begin{aligned} \min_{w,\boldsymbol{b}} \quad & \mathcal{L}_{hinge}(w,\boldsymbol{b}) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\max\Big(1-y_i((w^t)^\top x-b_t),0\Big) \\ \text{subject to} \quad & \lambda > 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (1.14) \\ & \text{for } t=1,...,T \\ & 0 \in \partial(w,b)\mathcal{L}_{low}(\lambda,w^t,b_t) \end{aligned}$$

where $\mathcal{L}_{low}$ can be the objective function of either of the two presented lower level problems.

solve the inner level problem (quadratic problem in constrained case) by some QP solver put solution into upper level problem and solve it by using bundle method
difficulty: subgradient is needed to build model of the objective function –> need subgradient $\frac{\partial\mathcal{L}}{\partial\lambda}$ –> for this need $\frac{\partial(W,b)}{\partial\lambda}$
but (w,b) not available as functions -> only values

Moore et al. [8] describe a method for getting the subgradient from the KKt-conditions of the lower level problem:

lower level problem convex -> therefore optimality conditions (some nonsmooth version -> source???) necessary and sufficient -> make "subgradient" of optimality conditions and then derive subgradient of w, b from this.
—> what are the conditions? optimality condition Lipschitz?

Say (show) that all needed components are locally Lipschitz; state theorems about differentiability almost everywhere and convex hull of gradients gives set of subgradients introduce special notation (only for this section) and because of readability adopt "gradient writing"

Subgradients: $\mathcal{G}_{upp,\lambda}, \mathcal{G}_{upp,w}, \mathcal{G}_{upp,b}$ -> subgradients of outer objective
g_w, g_b -> subgradient of w, b

$$finalsubgradient = (\mathcal{G}_{upp,w}(w,b,\lambda))^\top g_w + (\mathcal{G}_{upp,b}(w,b,\lambda))^\top g_b + \mathcal{G}_{upp,\lambda}(w,b,\lambda)$$

subgradients $\mathcal{G}_{upp,...}$ easy to find (assumption that locally Lipschitz) -> in this application differentiable

difficulty: find g_w, g_b important: optimality condition must be a linear system in w, b ->

this is the case in this application

$$H(\lambda) \cdot (w, b)^\top = h(\lambda)$$

find subgradients of each element (from differentiation rules follows)

$$\partial_\lambda H \cdot (w, b)^\top + H \cdot (\partial_\lambda w, \partial_\lambda b)^\top = \partial_\lambda h$$

solve this for (w, b):

$$(\partial_\lambda w, \partial_\lambda b)^\top = H^{-1} \left(\partial_\lambda h - \partial_\lambda H \cdot (w, b)^\top \right)$$

matrix H has to be inverted -> in the feature space so scalable with size of data set -> still can be very costly [8]

Applied to the two bilevel classification problems derived above, the subgradients have the following form:

derivative of upper level objective: Notation: $\delta_i := 1 - y_i(w^\top x^i - b)$

$$\partial_w \mathcal{L}_{upp} = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{\mathcal{N}_t} \sum_{i \in \mathcal{N}_t} \begin{cases} -y_i x^i & \text{if } \delta_i > 0 \\ 0 & \text{if } \delta_i \leq 0 \end{cases} \tag{1.15}$$

$$\partial_b \mathcal{L}_{upp} = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{\mathcal{N}_t} \sum_{i \in \mathcal{N}_t} \begin{cases} y_i & \text{if } \delta_i > 0 \\ 0 & \text{if } \delta_i \leq 0 \end{cases} \tag{1.16}$$

here at the kink subgradient 0 is taken

for hingequad: -> here subgradient
optimality condition:

$$0 = \partial_w \mathcal{L}_{low} = \lambda w + 2 \sum_{i \in \bar{\mathcal{N}}_t} \begin{cases} (1 - y_i(w^\top x^i - b))(-y_i x^i) & \text{if } \delta_i > 0 \\ 0 & \text{if } \delta_i \leq 0 \end{cases} \tag{1.17}$$

$$0 = \partial_b \mathcal{L}_{low} = 2 \sum_{i \in \bar{\mathcal{N}}_t} \begin{cases} (1 - y_i(w^\top x^i - b))(y_i) & \text{if } \delta_i > 0 \\ 0 & \text{if } \delta_i \leq 0 \end{cases} \tag{1.18}$$

subgradient??? is this smooth? with respect to λ

$$0 = w + \lambda \partial_\lambda w + 2 \sum_{i \in \bar{\mathcal{N}}_t} \begin{cases} (-y_i(\partial_\lambda w^\top x^i - \partial_\lambda b))(-y_i x^i) & \text{if } \delta_i > 0 \\ 0 & \text{if } \delta_i \leq 0 \end{cases} \tag{1.19}$$

$$0 = 2 \sum_{i \in \bar{\mathcal{N}}_t} \begin{cases} (-y_i(\partial_\lambda w^\top x^i - \partial_\lambda b))(y_i) & \text{if } \delta_i > 0 \\ 0 & \text{if } \delta_i \leq 0 \end{cases} \tag{1.20}$$

From this the needed subgradients can be calculated via:

$$2 \cdot \begin{pmatrix} \frac{\lambda}{2} + \sum_{i \in \bar{\mathcal{N}}_t} y_i^2 x^i (x^i)^\top & \sum_{i \in \bar{\mathcal{N}}_t} -y_i^2 x^i \\ \sum_{i \in \bar{\mathcal{N}}_t} -y_i^2 (x^i)^\top & \sum_{i \in \bar{\mathcal{N}}_t} y_i^2 \end{pmatrix} \cdot \begin{pmatrix} \partial_\lambda w \\ \partial_\lambda b \end{pmatrix} = \begin{pmatrix} -w \\ 0 \end{pmatrix} \tag{1.21}$$

for hinge not quad:

not as much information in the subgradient/derivative

similar calculation leads to

$$\partial_\lambda w = -\frac{w}{\lambda} \tag{1.22}$$

$$\partial_\lambda b = 0 \tag{1.23}$$

does not work with just Hingeloss because set valued optimality condition –> have to find "corresponding" equation to the chosen subgradient –> do not know how

1.4.3 Multi Group Model

Idea: different samples within the group have different have different quality –> put the samples with similar quality within one group –> give it its own λ_g such that the groups are weighted depending on their quality.

two goals: better modleing and implicit information about the groups provided by the parameter λ_g -> small λ good quality, large λ bad quality???

Examle: different people do same experiment; measure same data...

Model (Bilevel Problem)

$$\begin{aligned}
\min_{w,b} \quad & \mathcal{L}_{hinge}(w,b) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\max\left\{1 - y_i\left(\left\langle w^t, x^i\right\rangle - b_t\right), 0\right\} \\
\text{s.t.} \quad & \lambda = [\lambda_1, ..., \lambda_G]^\top > 0 \text{ and} \\
& \text{for } t = 1, ..., T \\
& (w^t, b_t) \in \arg\min_{w,b}\left\{\sum_{g=1}^{G}\left(\frac{\lambda_g}{2}\|w\|_2^2 + \sum_{i\in\bar{\mathcal{N}}_t^g}\max\left\{1 - y_i\left(\left\langle w, x^i\right\rangle - b\right), 0\right\}^2\right)\right\}.
\end{aligned} \tag{1.24}$$

Reformulation of the lower level objective:

$$\mathcal{L}_{low}(w,b) = \frac{\sum_{g=1}^{G}\lambda_g}{2}\|w\|_2^2 + \sum_{i\in\bar{\mathcal{N}}_t}\max\left\{1 - y_i\left(\left\langle w, x^i\right\rangle - b\right), 0\right\}^2$$

Derivatives

overall gradient:

$$\frac{\mathrm{d}\mathcal{L}_{upp}}{\mathrm{d}\lambda} = \underbrace{\frac{\partial\mathcal{L}_{upp}}{\partial(w,b)}}_{\in\mathbb{R}^{n+1}}^{\top}\underbrace{\frac{\partial(w,b)}{\partial\lambda}}_{\in\mathbb{R}^{n+1\times G}} \in \mathbb{R}^G$$

gradient outer level objective:

$$\frac{\partial\mathcal{L}_{upp}}{\partial w} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\delta_i\left(-y_i x^i\right) \in \mathbb{R}^n$$

$$\frac{\partial\mathcal{L}_{upp}}{\partial b} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|\mathcal{N}_t|}\sum_{i\in\mathcal{N}_t}\delta_i y_i \in \mathbb{R}$$

$$\delta_i = \begin{cases} 1 & \text{if } 1 - y_i\left(\langle w, x^i\rangle - b\right) > 0 \\ 0 & \text{else} \end{cases}$$

in MATLAB:

$$\frac{\partial\mathcal{L}_{upp}}{\partial w} = \frac{1}{T}\texttt{sum}(-\texttt{bsxfun}(\texttt{@times}, X, \texttt{delta}.*Y), 2) \in \mathbb{R}^n$$

$$\frac{\partial \mathcal{L}_{upp}}{\partial b} = \frac{1}{T}\texttt{sum}(\texttt{delta.} * Y, 1) \in \mathbb{R}$$

gradient lower level objective:
first optimality conditions $0 = \frac{\partial \mathcal{L}_{low}}{\partial w}, 0 = \frac{\partial \mathcal{L}_{low}}{\partial b}$:

$$0 = \sum_{g=1}^{G}(\lambda_g w) + 2\sum_{i\in\bar{\mathcal{N}}_t} \delta_i \left\{1 - y_i\left(\left\langle w, x^i\right\rangle - b\right)\right\}(-y_i x^i) \in \mathbb{R}^n$$

$$0 = 2\sum_{i\in\bar{\mathcal{N}}_t} \delta_i \left\{1 - y_i\left(\left\langle w, x^i\right\rangle - b\right)\right\}(y_i) \in \mathbb{R}$$

derivatives with respect to $\lambda_g, g = 1, ..., G$, then $\in \mathbb{R}^{n+1\times G}$

write one column of the matrix $\in \mathbb{R}^{n+1} \rightarrow G$ such columns:

$$0 = w + \lambda_g \frac{\partial w}{\partial \lambda_g} + 2\sum_{i\in\bar{\mathcal{N}}_t} \delta_i \left(y_i^2 x^i (x^i)^\top \frac{\partial w}{\partial \lambda_g} - y_i^2 x^i \frac{\partial b}{\partial \lambda_g}\right) \in \mathbb{R}^n$$

$$0 = 2\sum_{i\in\bar{\mathcal{N}}_t} \delta_i \left(-y_i^2 (x^i)^\top \frac{\partial w}{\partial \lambda_g} + y_i^2 \frac{\partial b}{\partial \lambda_g}\right) \in \mathbb{R}$$

rewrite to calculate $[\partial w/\partial\lambda, \partial b/\partial\lambda]^\top \in \mathbb{R}^{n+1\times G} \rightarrow$ have to calculate every column of this matrix on its own
solve G times this system of equations

$$2\begin{pmatrix} \frac{\lambda_g}{2}\mathbb{I} + \sum_{i\in\bar{\mathcal{N}}_t} \delta_i (y_i^2 x^i (x^i)^\top) & \sum_{i\in\bar{\mathcal{N}}_t} \delta_i (-y_i^2 x^i) \\ \sum_{i\in\bar{\mathcal{N}}_t} \delta_i (-y_i^2 (x^i)^\top) & \sum_{i\in\bar{\mathcal{N}}_t} \delta_i (y_i^2) \end{pmatrix} \begin{pmatrix} \frac{\partial w}{\partial \lambda_g} \\ \frac{\partial b}{\partial \lambda_g} \end{pmatrix} = \begin{pmatrix} -w \\ 0 \end{pmatrix}$$

Rechtschreibfehler, Namen, Stil überprüfen

References

[1] F. H. Clarke. *Optimization and nonsmooth analysis.* Classics in Applied Mathematics. Society for Industrial and Applied Mathematics Philadelphia, 1990.

[2] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[3] A. Fuduli, M. Gaudioso, and G. Giallombardo. Minimizing nonconvex nonsmooth functions via cutting planes and proximity control. *SIAM J. Optim*, 14(3):743–756, 2004.

[4] J. Heinonen. Lectures on Lipschitz analysis. Lectures at the 14th Jyväskylä Summer School in August 2004, 2004.

[5] K. Königsberger. *Analysis 2.* Springer Berlin Heidelberg, 2002.

[6] G. Kunapuli. *A bilevel optimization approach to machine learning.* PhD thesis, Rensselaer Polytechnic Institute Troy, New York, 2008.

[7] G. Moore, C. Bergeron, and K. P. Bennett. Gradient-type methods for primal SVM model selection. Technical report, Rensselaer Polytechnic Institute, 2010.

[8] G. Moore, C. Bergeron, and K. P. Bennett. Model selection for primal SVM. *Machine Learning*, 85(1):175–208, 2011.

[9] J. Outrata, M. Kočvara, and J. Zowe. *Nonsmooth Approach to Optimization Problems with Equilibrium Constraints.* Springer US, 1998.

[10] R. Rockafellar. Extensions of subgradient calculus with applications to optimization. *Nonlinear Analysis: Theory, Methods & Applications*, 9(7):665–698, jul 1985.

[11] R. T. Rockafellar and R. J. B. Wets. *Variational Analysis*, volume 317 of *Grundlehren der mathematischen Wissenschaften.* Springer Berlin Heidelberg, 3rd edition, 2009.

[12] V. N. Vapnik. *Statistical Learning Theory.* JOHN WILEY & SONS INC, 1998.

[13] V. N. Vapnik. An overview of statistical learning theory. *IEEE Trans. Neural Networks*, 10(5):988–999, 1999.

[14] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer New York, 2013.