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 chapter the nonconvex inexact bundle algorithm is applied to the problem of model selection for *support vector machines* (SVM) solving classification tasks. It relies on a bilevel formulation proposed by Kunapuli [**?**] and Moore et al. [**?**].

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 [**?**]

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

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 to 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 and 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 [**?**].

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* $S(x) = \{y \in \mathbb{R}^k | y \text{ solves the lower level problem}\}$ is often nondifferentiable [**?**] and since elements of the solution map are plugged into the outer level objective function in the implicit approach, the outer level function 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 [**?**] for support vector regression. However they use a gradient decent method which is not guaranteed to stop at an optimal solution.
In [**?**] he also suggests the nonconvex exact bundle algorithm of Fuduli et al. [**?**] 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 90's by Vapnik and co-workers. Soon they could outperform several other programs in this area [**?**] and the subsequent interest in SVMs lead to a very versatile application of these machines [**?**].

The case that is considered here is binary support vector classification using supervised learning.
In classification data from a possibly high dimensional vector space $\tilde{X} \subseteq \mathbb{R}^n$, 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 an data point x has the label y [**?**].
The overall goal is then to optimize the generalization ability, meaning the ability to predict the output for unseen data correctly [**?**].

1.2.1 Risk minimization

The concept of SVM's was originally inspired by the statistical learning theory developed by Vapnik. For a throughout analysis see [**?**].
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*

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

Where $\mathcal{L} : \mathbb{R}^2 \to \mathbb{R}$ is the loss function, $f_\lambda : \mathbb{R}^n \cap \mathcal{F} \to \mathbb{R},\ \lambda \in \Lambda$ the 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 [**?**].

As the only given information is given 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

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

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 $f_{\lambda,emp}$ that minimizes the empirical rist 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 [**?**].

Vapnik introduced therefore the structural risk minimization induction principle (SRM). It ensures that the used set of functions has a structure that makes it strongly consistent [**?**]. 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" [**?**, p. 994]. This reduces the risk of *overfitting*, meaning to overly fit the function to the training data with the result of poor generalization [**?**].

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 [**?**, **?**] and references therein.

1.2.2 Support Vector machines

In the case of linear binary classification one searches for a an affine hyperplane $\boldsymbol{w}$ shifted by b to separate the given data. The vector $\boldsymbol{w}$ 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

$$f(x) = sign(\boldsymbol{w}^\top x - b).$$

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 $(\boldsymbol{w}, b)$ separates the data there exist infinitely many hyperplanes $(t\boldsymbol{w}, b),\ t > 0$ that separate the data in the same way.
To have a unique description of a separation hyperplane the *canonical hyperplane for given data* $x \in X$ is defined by

$$f(x) = \boldsymbol{w}^\top x - b \quad \text{s.t.} \quad \min_i |\boldsymbol{w}^\top x^i - b| = 1$$

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 [**?**].

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 [**?**].

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

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

$$\|\boldsymbol{w}\|^2 \quad \textit{subject to} \quad y_i(\boldsymbol{w}^\top x - b) \geq 1 \quad i = 1, ..., l$$

realizes a maximal margin hyperplane

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 $(\boldsymbol{w}, b)$ and target margin γ is

$$\xi_i = \max(0, \gamma - y_i(\boldsymbol{w}^\top x + b))$$

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

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 $(\boldsymbol{w}, b)$:

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top x^i - b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \\ & \forall i = 1, \ldots, l \end{aligned} \tag{1.4}$$

and

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{l}\xi_i^2 \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top x^i - b) \geq 1 - \xi_i \\ & \forall i = 1, \ldots, l \end{aligned} \tag{1.5}$$

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_α to reduce the error on the training data and the danger of overfitting to have good generalization. It has to be chosen a priori [**?**]. The two optimization problems only differ in the norm chosen for the loss function. In (**??**) the one-norm is chosen, in (**??**) the squared two-norm is used. Problem (**??**) is the one that will finally be 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 for training 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 [**?**].

1.3.1 Reformulation as bilevel problem

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

Let $\Omega = (x^1, y_1), ..., (x^l, y_l) \subseteq \mathbb{R}^{n+1}$ 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 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 writes

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

In the case of support vector classification the T inner problems hve the classical SVM formulation (**??**). The problem can also be rewritten into a 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 $(\boldsymbol{w}^t, b_t) \in \mathbb{R}^{n+1}$. The inner level problem for the t'th fold can therefore be stated as

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

Where the hyper-parameter $\lambda = \frac{1}{C}$ was used due to numerical stability [**?**].

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 would therefore be to pick the misclassification loss, that count how many data points of the respective validation set Ω_t were misclassified when taking function f^t.

The misclassification loss can be written as

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

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

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

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

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

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

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

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.

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

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

1.3.2 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. [?, 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 $x \triangleq \lambda$, $y \triangleq w$ and $F(x_0, y_0) \triangleq \frac{\partial}{\partial w}\mathcal{L}_{low}(w^*, \lambda_0)$ 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 (**??**). 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 (**??**). 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[**?**, 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. [14, 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 [14])

Theorem 1.5 (c.f. [**?**, 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 $\hat{\partial}^\infty p(\bar{x}) \subset M^\infty(\bar{x})$.

show that functions fulfill assumptions

Implicit function theorem>

Theorem 1.6 ([**?**, Theorem 5.2]) *(a) Assume there exists a single valued Lipschitz function F*

in original theorem around 0< can always be done by

- notation - check
- definition of subgradient-"matrix" - check
- chain rule - check
- optimality condition /check
- welche art von inexaktheit -> Funktionswerte w, b inexakt
  -> gradient im Endeffekt exakt, da von exakter optimalit'tsbedingung ausgegangen wird

In practice this (Rademacher) means that it is possible to choose a subgradient by using the (one sided) gradients at nondifferentiable points. We keep this in mind when analyzing the procedure of finding a subgradient $g^\lambda \in \partial^\lambda w(\lambda)$ in the nondifferentiable case.

what else???

explanation

For me: f locally Lipschitz??? then partial derivatives are the same! Else: check definition of derivatives!

-> theory partial derivatives for subgradients?????????
??? Formula ??? $\in \partial L_{upp} \partial \lambda$
???one has to assume that the inner level problem is locally Lipschitz (or more general: its nonconvex subdifferential is well defined at every point).
Subdifferential has to have again a subdifferential!!! -> w.r.t. λ

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 [**?**] is that the problem is not smoothly replaced by its KKT conditions but only by this optimality condition. The weight vector $\boldsymbol{w}$ and bias b are treated as a function of λ and are optimized separately from this hyperparameter. The reformulated bilevel problem becomes:

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

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. [**?**] 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 chapter) 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

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

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 [**?**]

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:

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

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

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

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

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

From this the needed subgradients can be calculated via:

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

for hinge not quad:

not as much information in the subgradient/derivative

similar calculation leads to

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

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

1.3.3 The Algorithm???

The inexact bundle algorithm for the support vector classification task in bilevel formulation

---

Bilevel Bundle Method

Initiate all parameters, select a starting hyper-parameter λ_1 and solve the lower level problem for $\boldsymbol{w}^1$ and b_1.
Calculate arbitrary subgradients of $\boldsymbol{w}^1$ and b_1 with respect to λ via **??** and a subgradient of the upper level problem by **??**. For $k = 1, 2, 3, \ldots$

1. Calculate the step d^k by minimizing the model of the convexfied objective
2. Compute the aggregate subgradient and error and the stopping tolerance δ. If $\delta_k \leq \texttt{tol} \rightarrow$ STOP.
3. Set $\lambda^{k+1} = \hat{\lambda}^k + d^k$.

4. solve again the inner level problem and calculate all subgradients needed to compute a subgradient of the outer level objective
   Calculate function value and a subgradient for the outer level objective function and test if a serious step was done If yes, set $\hat{\lambda}^{k+1} = \lambda^{k+10}$ and select $t_{k+1} > 0$.
   Otherwise $\rightarrow$ nullstep
   Set $\hat{\lambda}^{k+1} = \hat{\lambda}^k$ and choose $0 < t_{k+1} \leq t_k$.
5. Select new bundle index set J_{k+1}. Calculate convexification parameter η_k and update the model M^k

---

Names for algorithms: BBMH -> hinge as inner level, BBMH2 -> hingequad as inner level

1.4 Numerical Experiments

The bilevel-bundle algorithm for classification was tested for four different data sets taken from the UCI Machine Learning Repository *citations as said in "names" data???* . For comparability with the already existing results presented in [**?**] the following data and specifications of it were taken:

Table like in Kunapuli

Data set	l_{train}	l_{test}	n	T
Pima Indians Diabetes Database	240	528	8	3
Wisconsin Breast Cancer Database	240	443	9	3
Cleveland Heart Disease Database	216	81	13	3
John Hopkins University Ionosphere Database	240	111	33	3

Table 1

As described in the PhD thesis the data was first standardized to unit mean and zero variance (*not the 0,1 column in ? dataset*). The bilevel problem with cross validation was executed 20 times to get averaged results. The results are compared by cross validation error, test error -> write which error this is and computation time. Additionally write $\boldsymbol{w}$, b, λ ??? The objective function and test error were scaled by 100. -> also test error (to get percentage)

After every run the calculated λ was taken and the algorithm was trained with $\frac{T}{T-1}\lambda$ on the whole training set. Then the percentage of misclassifications on the test set was calculated via

$$E_{test} = \frac{1}{l_{test}} \sum_{i=1}^{l_{test}} \frac{1}{2} |sign\left(\boldsymbol{w}^\top x^i - b\right) - y_i| \tag{1.24}$$

Table ??? shows the results

Data set	Method	$T/(T-1)\lambda$	CV Error	Test Error	Time (sec.)
`pima`	hingequad	10^{-15}	60.72 ± 9.56	24.11 ± 2.71	2.15 ± 0.52
	hinge loss				
`cancer`	hingequad	$0.6 < \lambda < 10.3$	10.75 ± 7.52	3.41 ± 1.16	3.43 ± 28.84
	hinge loss				
`heart`	hingequad	10^{-16}	48.73 ± 5.53	15.56 ± 4.44	3.43 ± 43.39
	hinge loss				
`ionosphere`	hingequad	$3 < \lambda < 7.5$	39.30 ± 5.32	12.21 ± 4.10	14.17 ± 51.27
	hinge loss				

Table 2

λ values in table not right, don‘t know with which algorithms they were reached

23.06.2017
found out: for real results, have to do it with the functions I take in the algorithm -> this is hingequad for inner level and either hingeloss or hingequad for outer level
from plots it seems that hingequad is closer to the misclassification loss
generally: from plots it looks as if all λs are best =0
with bundle bilevel and hingeloss (outer): λ very much depending on starting value -> why??? graphs seem to be monotonously decreasing into 0

plots such as objective function (upper hingeloss and lower hingequad) in bilevel bundle algorithm: No minimum visible (also for ionosphere and cancer???...)
analysis of every plot:
pima: looks the same as “old” plot, minimum is 0
wine quality red and red 56: minimum is 0
covtype: same
cancer: doesn’t really look similar to “old” plot, slope different, minimum different; minimum at 0 and not at about 10
ionosphere: slope, ect. look similar to “old” plot; but minimum at 0
heart: like “old” plot
maybe cancer and ionosphere plots just “incidents” -> because of special choice of vali-

dation set????
-> no, can also be that I averaged over 20 times...

misclassification loss as upper level objective
pima: average seems to fo to 0 as min
generally: misclassification loss seems as if no optimization of λ possible because choice of validation set seems to have much more influence

Results forλ only if it stayed there after second run with second starting value
change in λ has very little effect; only after comma for "percent-writing"
errors like in table for all but ionosphere -> has only 5% error?; ???-> error the smaller, the smaller λ???
pima simply not depending on λ
cancer not really depending in λ, only if it gets really big > 1000 (for > 10 minor change)
heart: changes a lot for the different λ, but best: $\lambda = 0$
seems that results come because of scaling of objective -> consistent with the plots I made
also consider: loss function of optimizer is not the one that calculates the test error

Extra table for $\boldsymbol{w}$, b, λ ?

First experiment: Classification

Write down bilevel classification problem and (if needed) which specification of the inexact bundle algorithm is used.

Covtype tests
Datensatz zerteilt: 1000, 5000, 10000, 50000 Datensätze
Ergebnisse mit Matlab App (linear SVM, 3 folds, parallel used):

Datensatz	Zeit	Fehler
1,000	16.22 sek	34.2%
5,000	10.524 sek	18.3%
10,000	14.689 sek	16.5%
50,000	643.57 sek	16.9%
50,000 (Rechnerhalle, 4 parallel)	326.83	16.9%
100,000 –"–	2492 sek = 41.5333 min	21.3%

scheint bei 50000 tatsächlich so lange zu dauern, da ziemlich genau doppelt so schnell bei parallel-Rechnung mit 4 anstatt 2 "Rechnern" –> kein Arbeitsspeicher Problem

Test mit bundle bilevel-Algorithmus:
for covtype, "Hare"-stopping condition

trainigs set	starting value	lambda	time	k	inull	Fehler
1,000	1		62.8280 sek	29	0	
1,000	86		130.1105 sek	61	8	19%
1,000 (quadprog)	86		6.5572 sek	61	8	
5,000 qp	1		16.2338 sek	27	0	
5,000 qp	47		34.9573 sek	60	0	14.74%
10,000 qp	1		59.7462 sek	49	0	
10,000 qp	50		85.0816 sek	69	0	15.32%
50,000 qp		588.2828 sek	45	0		
50,000 qp	60		897.8123 sek	69	0	14.86%
100,000 qp	1		1358.7 sek = 22.6455 min	37	0	39.41

!!!!!!!!!!did not take λ but the error value!!!!!!!!!!!!!!!

Fehler berechnet für Daten 1001 bis 2000 von komplettem Datensatz

Versuch mit Daten 1001-5000: scheint Problem bei matrizenaufbau zu haben

Analyse Timer: 100% der Zeit in postpro-wb-class-hinge-qpas dabei 100% der Zeit fÃ¼r qpas

Ergebnis viel!!! schneller, wenn quadprog und sparse-matrizen, gibt EXAKT! gleiches Ergebnis fÃ¼r fehler

geht dann auch mit mehr datensätzen (4000) -> Fehler nur 7.75%???

für Datensatz 5000 $\rightarrow$ Testdaten 5001 bis 10000

exemplarisch getestet: wie gross ist der Unterschied bei ergebnisse bei verschiedenen Starwerten? - bei 10000: $1e-15$, bei 50000: exact

für Datensatz 10000: Testdaten 10001 bis 15000

50000: deutlicher Anstieg der Rechenzeit merkbar irgendwo zwischen 10000 und 50000 muss eine Schwelle liegen - Speicher? - ab dieser Matrix Grösse gibt matlab fehler wenn nicht sparse matrizen explizit erstellt werden sollen (zu viel Speicher) unwahrscheinlich - siehe erklärung App

komish: warum dauert es bei näherem Startwert so viel länger???

unconstrained tested for 1000: much faster; no difference in steps for $x_0 = 1$, 70, 6 for $x_0 = 86$, 7.2 sek

0815-Funktion für bilevel optimierung:

fÃ¼r covtype1000: in sehr wenig Zeit: lambda = 100 -> selbes ergebnis im Fehler: 19%

Infos on Data sets

Data Set	instances, attributes	1/C (SVC)	C, ε (SVR)	Test Error	Source	
Adult	300 000, 10+1		0.017, 0.19	24.99%	[?]	think lin
Boston Housing	506, 12+1		0.25, 0.015	19.44%	[?]	
Adult	Tset: 11221	1/0.05			[?]	
Adult		$> 10^8$		14.1	[?]	accuracy ar

If more values found: take best

Rechtschreibfehler, Namen, Stil überprüfen

References

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

[2] Napsu Haarala, Kaisa Miettinen, and Marko M. Mäkelä. Globally convergent limited memory bundle method for large-scale nonsmooth optimization. *Mathematical Programming*, 109(1):181–205, 2007.

[3] Warren Hare and Claudia Sagastizàbal. A redistributed proximal bundle method for nonconvex optimization. *SIAM Journal on Optimization*, 20(5):2442–2473, 2010.

[4] Warren Hare, Claudia Sagastizàbal, and Mikhail Solodov. A proximal bundle method for nonsmooth nonconvex functions with inexact information. *Computational Optimization and Applications*, 63:1–28, 2016.

[5] Claude Lemaréchal and Claudia Sagastizábal. *An approach to variable metric bundle methods*, pages 144–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[6] Claude Lemaréchal and Claudia Sagastizábal. Variable metric bundle methods: From conceptual to implementable forms. *Mathematical Programming*, 76(3):393–410, 1997.

[7] Adrian S Lewis and Michael L Overton. Nonsmooth optimization via bfgs. *submitted to SIAM Journal on Optimization*, pages 1–35, 2009.

[8] L. Lukšan and J. Vlček. Globally convergent variable metric method for convex nonsmooth unconstrained minimization. *Journal of Optimization Theory and Applications*, 102(3):593–613, sep 1999.

[9] Stefan Ulbrich Michael Ulbrich. *Nichtlineare Optimierung*. Springer Basel AG, 2012.

[10] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[11] Dominikus Noll. Bundle method for non-convex minimization with inexact subgradients and function values. In *Computational and Analytical Mathematics*, pages 555–592. Springer Nature, 2013.

[12] Dominikus Noll, Olivier Prot, and Aude Rondepierre. A proximity control algorithm to minimize non-smooth and non-convex functions. *Pacific Journal of Optimization*, 4(3):571–604, 2012.

[13] Boris T. Polyak. *Introduction to Optimization*. Optimization Software , Inc., Publications Division, New York, 1987.

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

[15] Jay S. Treiman. Clarke's gradients and ε-subgradients in banach spaces. *Transactions of the American Mathematical Society*, 294(1):65–65, jan 1986.

[16] J. Vlček and L. Lukšan. Globally convergent variable metric bundle method for nonconvex nondifferentiable unconstrained minimization. *Journal of Optimization Theory and Applications*, 111(2):407–430, 2001.

[17] Claudia Sagastizàbal Warren Hare. Computing proximal points of nonconvex functions. *Mathematical Programming*, 116:221–258, 2009.

[18] Claude Lemaréchal Welington de Oliveira, Claudia Sagastizàbal. Convex proximal bundle methods in depth: a unified analysis for inexact oracles. *Mathematical Programming*, 148:241–277, 2014.