Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Mathematics

Subgradient Methods for the Efficient Learning of Minimax Risk Classifiers

Thesis supervisor:
Marco Di Summa (UNIPD)

Laureando: Maialen Beristain Cortés

Thesis co-supervisor:
Santiago Mazuelas (BCAM)

Thesis co-supervisor:
Leticia Hernando (UPV/EHU)

Academic year 2023/2024

A tutti quelli che
posso chiamare amici

Acknowledgements

I would like to extend my sincere appreciation and gratitude to my supervisors at BCAM and EHU, Santiago Mazuelas and Leticia Hernando, for their guidance and support throughout the research process. Their supervision and dedication have been crucial to the development of this work and to my personal growth within the research field. I hope to continue learning from them in the future. I also want to thank my advisor from the University of Padova, Prof. Di Summa, for his advice and meticulous supervision on every detail.

I am also grateful to my colleagues and friends from BCAM and Bilbo for all the nice moments we shared during these months. I truly appreciate each one of you. I would like to give special thanks to Jose for his constant support and readiness to help from the very beginning. I have learned a great deal thanks to you, and I am happy to have you as a friend. I would also like to acknowledge my friends and university colleagues in Italy, particularly those from the climbing group and Margherita, whose companionship has made this journey very special. Sof and Guille, thank you for being an integral part of my daily life. Special mention goes to my best friends from home and my family.

Abstract

This research presents subgradient methods which are optimization techniques particularly design for non-differentiable functions. In this work subgradients methods are applied to solve the minimization problem arised during the learning stage of Minimax Risk Classifiers (MRCs). The contribution and innovative aspect of this work lies in the application of these methods to this scenario. The principal success of this study is to obtain an efficient method that balances computational efficiency with the quality of the approximation solution. To develop an effective approach, theoretical concepts have been studied and applied afterwards in various experiments in order to tune the parameters defining the algorithms. This process has been the principal and decisive task of the work. The experiments carried out provide us information about computational time and solution accuracy, so that we can propose the most useful parameters in order to provide a successful method.

Contents

Introduction

Optimization problems involve finding the best solution from a set of feasible options, often under specific constraints. These problems are inherently challenging for several reasons such as the complexity and scale of the problems. Nevertheless, such problems still need to be solved so various methods have been developed to address these challenges. These methods for resolution of optimization problems are classified in two categories: exact methods, the ones that guarantee finding the optimal solution, and approximation algorithms, which provide solutions that are close to the optimum. While exact methods guarantee finding the optimal solution, they can be computationally expensive and impractical for large-scale problems. Approximation methods offer a balance between solution quality and computational efficiency; they do not guarantee an optimal solution but can provide good enough solutions within a reasonable time frame.

Machine learning problems often involve large-scale optimization tasks due to the high dimensionality of data and the complexity of models. In fact, as datasets grow in size and feature space, the optimization process must handle millions of parameters and constraints. This complexity arises from the need to train models on vast amounts of data to achieve high accuracy and generalization. Efficient optimization methods are crucial for managing computational resources, reducing training times, and ensuring that models can scale effectively with the increasing volume and variety of data encountered in real-world applications.

Classification is a fundamental machine learning task, where the goal is to assign labels to instances based on their attributes. It is a type of supervised learning that involves training a model on a labeled dataset, so it can predict the labels for new, unseen instances. For such training process, large datasets are needed in order to obtain the most accurate predictions, by providing the model a large quantity of instances. The main goal of a classification task is to obtain a classifier that fails less. The concept of failure is represented by a loss function, that is, a function that measures the success on the performance of the classifier. Then, the definition of the

loss function determines the classifier. Minimax Risk Classifiers (MRCs) are specific classifiers minimizing the worst case 0-1 loss (Mazuelas et al., 2023) that define the objective function for our minimization problem. That is, the optimization problem given in this work consists in minimizing the function obtained in the learning step of MRCs.

Subgradient methods (Boyd, 2014) are crucial optimization techniques for non-differen-tiable functions, particularly useful for those large-scale problems emerging in machine learning, such as classification. While they do not guarantee the exact optimum, subgradient methods are especially helpful for problems where exact methods are impractical due to computational constraints.

Traditional gradient-based methods (Boyd and Vandenberghe, 2004) rely on the smoothness of the objective function, which limits their application to differentiable functions. In contrast, subgradient methods extend the notion of gradient to non-differentiable functions by introducing the subgradient (Boyd et al., 2022), making these methods suitable for a broader range of optimization tasks. However, these methods typically converge slower than gradient descent due to the lack of precise gradient information. To mitigate this, various strategies such as adaptive step sizes and averaging techniques are employed to enhance convergence rates.

This research presents subgradient methods applied to solve the minimization problem that arises during the learning stage of MRCs. The focus is primarily on the efficiency of these methods: the main point of this work is to handle the optimization task efficiently. To develop an effective approach, theoretical concepts have been studied and applied afterwards in various experiments in order to tune the parameters defining the algorithms. This process has been the principal and decisive task of the work, enabling the development of a successful and optimized method.

These methods have not been previously applied to solve the specific problem given from the MRCs. The contribution and innovative aspect of this work lies in their application to this scenario, aiming to develop an efficient approach for solving the minimization problem. The principal success of this study is to obtain an efficient method that balances computational efficiency with the quality of the approximation solution.

For the purpose of this study, basic theoretical concepts about classification and supervised learning are introduced in Chapter 1, and the main concepts of MRCs are described in Chapter 2, in order to introduce the minimization problem addressed in this work. Chapter 3 deals entirely with the subgradient methods, introducing variants of the method, the algorithms and the parameters used during the learning stage. Chapter 4, focuses on the

experimental part, analyzing obtained numerical results. Finally, in Chapter 5 we present the conclusions and some future work. The codes for the experiments can be found in Appendeces A,B and C.

Chapter 1

Supervised Classification

Classification is a standard learning task that involves assigning a category to each instance (Mohri, 2018). In this type of task, learning consists of accurately predicting the label of unseen instances or new items, and the primary objective of a classification algorithm is to build a model that can make such predictions.

For example, document classification consists of assigning a category representing a topic, such as *business*, *sports* or *weather*, to each document, while spam detection is the problem of learning to automatically classify email messages as either *spam* or *non-spam*. Note that the first problem allows several labels for the classification, whereas spam problem has just two options. If we categorize the classification task according to this rule, we obtain the following two categories: *binary classification* tasks for the ones with just two labels, and *multiclass classification* with more than two labels.

1.1 Problem formulation

Classification techniques assign instances in a set $\mathcal{X}$ to labels or classes in a set $\mathcal{Y}$, where $\mathcal{X}$ is a Borel subset of $\mathbb{R}^d$, and we say that x has d variables or features, and $\mathcal{Y}$ is a finite set represented by $\{1, 2, \ldots, |\mathcal{Y}|\}$. We denote by $\Delta(\mathcal{Y})$ the space of probability distributions over $\mathcal{Y}$, and by $\Delta(\mathcal{X} \times \mathcal{Y})$ the set of probability distributions over $\mathcal{X}$ and $\mathcal{Y}$.

Classifiers can be defined in two ways: they can be represented by functions from the set of instances to the set of labels, $\mathrm{h} : \mathcal{X} \longrightarrow \mathcal{Y}$, where $\mathrm{h}(x)$ represents the label deterministically assigned to the instance x. Classifiers

can also be given by functions from instances to probability distributions on labels, $\mathrm{h} : \mathcal{X} \longrightarrow \Delta(\mathcal{Y})$, and we denote by $\mathrm{h}(y|x)$ the set of probabilities with which h classifies x, for each $y \in \mathcal{Y}$. Let $\mathrm{T}(\mathcal{X}, \mathcal{Y})$ be the set of all classification rules, our goal is to find a classification rule $\mathrm{h} \in \mathrm{T}(\mathcal{X}, \mathcal{Y})$. In the following, we will consider deterministic classification rules $h : \mathcal{X} \longrightarrow \mathcal{Y}$.

We want classification rules with small losses, that is, functions that measure the difference, or loss, between a predicted and a true label. We denote by $\ell(\mathrm{h}, (x, y))$ the loss corresponding to the classifier h giving the predicted value for x, whose real label is y.

The most common and intuitive loss function used for classification is the 0-1 loss: 0 if the classifier success, that is, if the prediction coincides with the true label, 1 if it fails. That is,

$$\ell_{0\text{-}1}(\mathrm{h}, (x, y)) = \mathbb{I}\{\mathrm{h}(x) \neq y\}. \tag{1.1}$$

The risk of a classification rule $\mathrm{h} \in \mathrm{T}(\mathcal{X}, \mathcal{Y})$ is its expected loss. That is, the risk related to the instance-label pair (x, y) following the underlying distribution $\mathrm{p}^*(x, y) \in \Delta(\mathcal{X} \times \mathcal{Y})$, i.e., the true distribution that generates the data, is

$$R(\mathrm{h}) = \mathbb{E}_{\mathrm{p}^*(x,y)} \ell(\mathrm{h}, (x, y)) \in \mathbb{R}.$$

In particular, the risk related to the 0-1 loss represents the error probability of the classifier h.

Finding the best rule is equivalent to minimizing the possible error probability, so, for the 0-1 loss, the problem lies in solving the following minimization problem:

$$\begin{aligned}
\min_{\mathrm{h}:\mathcal{X}\to\mathcal{Y}} \mathbb{E}_{\mathrm{p}^*(x,y)} \ell_{0\text{-}1}(\mathrm{h}, (x, y)) &= \min_{\mathrm{h}:\mathcal{X}\to\mathcal{Y}} \sum_{x\in\mathcal{X}, y\in\mathcal{Y}} \mathrm{p}^*(x, y)\mathbb{I}\{\mathrm{h}(x) \neq y\} \\
&= \sum_{x\in\mathcal{X}} \min_{\mathrm{h}(x)\in\mathcal{Y}} \left(\sum_{y\in\mathcal{Y}} \mathrm{p}^*(x, y)\mathbb{I}\{\mathrm{h}(x) \neq y\} \right) \\
&= \sum_{x\in\mathcal{X}} \min_{\mathrm{h}(x)\in\mathcal{Y}} \left(\sum_{y\in\mathcal{Y}} \mathrm{p}^*(x)\mathrm{p}^*(y|x)\mathbb{I}\{\mathrm{h}(x) \neq y\} \right) \\
&= \sum_{x\in\mathcal{X}} \mathrm{p}^*(x) \min_{\mathrm{h}(x)\in\mathcal{Y}} \sum_{y\in\mathcal{Y}} \mathrm{p}^*(y|x) \left(1 - \mathbb{I}\{\mathrm{h}(x) = y\}\right) \\
&= \sum_{x\in\mathcal{X}} \mathrm{p}^*(x) \min_{\mathrm{h}(x)\in\mathcal{Y}} \left(1 - \sum_{y\in\mathcal{Y}} \mathrm{p}^*(y|x)\mathbb{I}\{\mathrm{h}(x) = y\} \right).
\end{aligned} \tag{1.2}$$

The equalities are justified by basic probability theory.

Then, the best rule is the so called *Bayes' rule*, $\mathrm{h} : \mathcal{X} \to \mathcal{Y}$ assigning the image

$$\mathrm{h}(x) = \arg\max_{y \in \mathcal{Y}} \mathrm{p}^*(y|x)$$

to a fixed $x \in \mathcal{X}$, and the best error probability is

$$\sum_{x \in \mathcal{X}} \mathrm{p}^*(x) \left(1 - \max_{y \in \mathcal{Y}} \mathrm{p}^*(y|x)\right) = 1 - \sum_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \mathrm{p}^*(x, y),$$

also known as *Bayes' risk*.

1.2 Supervised scenario

There are different machine learning scenarios that differ in the types of training data available to the learner, the order and method by which data is received, and the test data used to evaluate the learning algorithm. The most common scenario associated with classification is *supervised learning*, where the learner receives a set of labeled samples as training data and makes predictions for all unseen instances. The spam detection problem discussed in the previous section is an example of this situation.

Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be n training instance-label pairs i.i.d. drawn from the underlying distribution p^*. Learning means obtaining the classifier h from the given instances.

For example, given the previous data, the nearest neighbor method classifies a new instance x with the label of the closest training instance x_i, that is, the classifier h assigns to instance x the image $\mathrm{h}(x) = y_i$, where $i \in \arg\min_{j=1,\ldots,n} \|x - x_j\|$. Note that this example does not do any optimization.

In general, we would like to solve the optimization problem given in (1.2) in order to find the Bayes' rule. However, this is not possible because p^* is unknown. Instead of minimizing the true expectation, an alternative is minimizing the average of the losses of the training samples.

This approach is called the Empirical Risk Minimization (ERM), and it is the most common approach for learning (Mohri, 2018; Vapnik, 1998; Shalev-Shwartz and Ben-David, 2014). ERM seeks to minimize the error on the training sample by solving

$$\min_{\mathrm{h}:\mathcal{X} \to \mathcal{Y}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathrm{h}, (x_i, y_i)). \tag{1.3}$$

1.2.1 Linear rules

By definition, a classification rule is any function from $\mathcal{X}$ to $\mathcal{Y}$. In order to easily illustrate these rules, let us consider the binary situation where the label space is $\mathcal{Y} = \{-1, +1\}$. We define a classifier $\mathrm{h} : \mathcal{X} \longrightarrow \mathcal{Y}$ for a new instance x and a fixed $\boldsymbol{\mu} \in \mathbb{R}^d$ as $\mathrm{h}(x) = \mathrm{sign}(x^T\boldsymbol{\mu})$, and we consider as positive if the argument is 0.

Intuitively, we can think of a classifier as a hyperplane that classifies x with label -1 or 1 based on its relative position to the hyperplane.

Note that, in this case, the 0-1 loss is

$$\ell_{0\text{-}1}(\mathrm{h}, (x, y)) = \mathbb{I}\{yx^T\boldsymbol{\mu} \leq 0\}$$

and the ERM is the solution of

$$\min_{\boldsymbol{\mu}\in\mathbb{R}^d} \sum_{i=1}^{n} \mathbb{I}\{y_i x_i^T\boldsymbol{\mu} \leq 0\}.$$

But the objective function is not convex, so this results on an NP-hard problem (Garey and Johnson, 1990), making the achievement of an exact solution computationally infeasible for large scale instances. Then, using the 0-1 loss is not an effective option.

Instead, *surrogate losses* are used: different loss functions which are convex and approximate the 0-1 loss (Bartlett et al., 2006). One specific way of doing this approximation is by the *logistic loss*:

$$\ell(\boldsymbol{\mu}, (x, y)) = \log(1 + \exp(-yx^T\boldsymbol{\mu})).$$

By using this logistic loss to compute the ERM, we obtain the so called *logistic regression* as learning method. So the learning stage consists of solving the convex optimization problem

$$\min_{\boldsymbol{\mu}\in\mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-yx^T\boldsymbol{\mu})),$$

and the best classifier is

$$\mathrm{h}(x) = \mathrm{sign}(x^T\boldsymbol{\mu}^*),$$

where $\boldsymbol{\mu}^*$ is the solution of the previous minimization problem.

1.2.2 Multiclass case

Classifying according to the sign does no longer work in cases with more than two classes. Instead, in this situation, we use *one-hot-encodings*,

$$\Phi(x, y) = e_y \otimes x \in \mathbb{R}^m, \tag{1.4}$$

where $\boldsymbol{e}_y$ is the y-th element of the canonical basis of $\mathbb{R}^{|\mathcal{Y}|}$ and $\otimes$ denotes the Kronecker product.

We can now consider classification rules determined as

$$\mathrm{h}(x) \in \arg\max_{y \in \mathcal{Y}} \Phi(x, y)^T \boldsymbol{\mu}$$

for a fixed $\boldsymbol{\mu}$.

For the binary case, where $\mathcal{Y} = \{1, 2\}$, we consider $\boldsymbol{\mu} = [\boldsymbol{\mu_1}^T, \boldsymbol{\mu_2}^T]^T$. Then,

$$\begin{aligned} \mathrm{h}(x) &\in \arg\max\{\Phi(x, y = 1)^T \boldsymbol{\mu}, \Phi(x, y = 2)^T \boldsymbol{\mu}\} \\ &= \arg\max\{x^T \boldsymbol{\mu_1}, x^T \boldsymbol{\mu_2}\}, \end{aligned} \tag{1.5}$$

so we have label 1 if $x^T(\boldsymbol{\mu_1} - \boldsymbol{\mu_2}) \geq 0$, and label 2 otherwise.

1.3 Performance evaluation

Recalling our problem, we have the i.i.d. samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ drawn from the distribution p^* as the training samples and we want to find a classifier $\mathrm{h} : \mathcal{X} \to \mathcal{Y}$ such that $R(\mathrm{h}) = \mathbb{E}_{\mathrm{p}^*} \ell(\mathrm{h}, (x, y))$ is small.

There are different methods to compute the risk.

(i) We can approximate the quantity by averages of training samples.

$$\mathbb{E}_{\mathrm{p}^*} \ell(\mathrm{h}, (x, y)) \approx \frac{1}{n} \sum_{i=1}^{n} \ell(\mathrm{h}, (x_i, y_i)).$$

However, the terms in the sum are not independent, resulting in a biased estimator.

(ii) *Leave one-out* method. We can use just $(x_1, y_1), (x_2, y_2), \ldots, (x_{n-1}, y_{n-1})$ to train the classifier h, and then estimate the risk in the following way

$$\mathbb{E}_{\mathrm{p}^*} \ell(\mathrm{h}, (x, y)) \approx \ell(\mathrm{h}, (x_n, y_n)).$$

This time, the estimator is unbiased, so it will be a suitable method, even though in this case, the classifier h will differ slightly.

(iii) *Leave p-out* method. Follows the idea of the previous method using p training samples.

(iv) *Hold out* method. Follows the idea of the leave one-out method, dividing samples into training and testing sets. This time, each set is built by adding a percentage of samples.

(v) *k-fold* method. This method randomly divides the samples into k blocks. For each repetition, we train the classifier with $k-1$ blocks and test it with the remaining one.

Chapter 2

Minimax Risk Classifiers

Most learning methods are based on the ERM approach that minimizes the empirical expected loss over the training samples. In this chapter, we introduce Minimax Risk Classifiers (MRCs), that minimize the worst-case 0-1 loss over general classification rules.

Recall the ERM approach aiming to solve (1.3). That is, given a training set of n samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, we want to find a classifier h that minimizes the error over the training set, i.e.,

$$\mathrm{h} \in \arg\min_{\mathrm{h}:\mathcal{X}\to\mathcal{Y}} \frac{1}{n}\sum_{i=1}^{n} \ell(\mathrm{h}, (x_i, y_i)).$$

The ideal solution for solving the optimization problem defining the ERM consists of finding the Bayes' rule according to Section 1.1. Equivalently, we have to solve

$$\mathscr{P}^* : \min_{\mathrm{h}\in\mathrm{T}(\mathcal{X},\mathcal{Y})} \mathbb{E}_{\mathrm{p}^*}\ell(\mathrm{h}, (x, y)).$$

In practice, solving this problem is not possible, since the underlying distribution p^* is unknown. So instead of minimizing the true expectation, the problem $\mathscr{P}^*$ can be approximated in different ways, and the optimization problem becomes

$$\mathscr{P} : \min_{\mathrm{h}\in\mathcal{F}} \sup_{\mathrm{p}\in\mathcal{U}} \mathbb{E}_{\mathrm{p}}\mathrm{L}(\mathrm{h}, (x, y)),$$

where $\mathcal{F}$ is a family of classification rules, $\mathcal{U}$ an uncertainty set of distributions, and L a surrogate loss function.

The ERM considers uncertainty sets containing only the empirical distribution of training samples. It is a very useful approach, but strongly relies on the specific training samples available. So it is not clear how to handle situ-

ations with corrupted samples or distribution shifts. Moreover, the quantity optimized at learning is not very meaningful.

For instance, we introduce the context of adversarial manipulation, that involves creating scenarios where an adversary intentionally manipulates input data to deceive a model. This technique is used to test and improve the robustness of machine learning algorithms. By introducing adversarial examples in order to make trouble in our learning procedure with the ERM, the computed learning procedure would not be efficient for this modified version; that is, could not classify the corrupted instances properly.

Another approach that makes these situations tractable, in order to develop a successful learning, is the Robust Risk Minimization (RRM), also known as distributionally robust learning (Farnia and Tse, 2016; Fathony et al., 2016). The main idea is to consider multiple scenarios which are consistent with the data, and to choose a rule with small error in the worst scenario.

RRM minimizes the worst-case expected loss with respect to an uncertainty set of distributions, i.e. aims to find the classifier

$$\mathrm{h} \in \arg\min_{\mathrm{h}} \max_{\mathrm{p} \in \mathcal{U}(z_n)} \mathbb{E}_{\mathrm{p}} \ell(\mathrm{h}, (x, y)),$$

where $\mathcal{U}(z_n)$ is an uncertainty set corresponding to the training set of n samples

$$z_n = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}.$$

RRM approach do not require that the training samples follow the same distribution as the testing samples, so this method is capable to deal with corrupted samples or distribution shifts. Moreover, if the true scenario is one of the scenarios considered, the error on the worst case would be an upper bound of the actual error. Nevertheless, this approach can be too pessimistic if too many scenarios are considered.

Note that the approximation of problem $\mathscr{P}^*$ by $\mathscr{P}$ depends on the choice of the uncertainty set. We want general enough sets in order to contain the underlying distribution, but reduced enough to provide a tight upper bound. This trade-off relies on parameters related to the size of the uncertainty set.

The uncertainty set considered by the MRCs can contain the true underlying distribution with certain confidence. Such uncertainty sets are defined by constraints on the expectations of a function $\Phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^m$, denoted as feature mapping, as

$$\mathcal{U} = \{\mathrm{p} \in \Delta(\mathcal{X} \times \mathcal{Y}) : |\mathbb{E}_{\mathrm{p}}\{\Phi\} - \boldsymbol{\tau}| \preceq \boldsymbol{\lambda}\}, \tag{2.1}$$

where $|\cdot|$ is the componentwise absolute value, $\boldsymbol{\tau}$ denotes the mean vector estimation corresponding to Φ, and $\boldsymbol{\lambda} \succeq 0$ is a confidence vector for inaccuracies in the estimate (Mazuelas et al., 2023).

The goal of MRCs is to find a classifier that minimizes the maximum expected loss over these uncertainty sets, using estimates of expectation $\boldsymbol{\tau}$ and confidence vector $\boldsymbol{\lambda}$.

2.1 MRCs with 0-1 loss

In this work we focus on MRCs with 0-1 loss, recall it from (1.1). It can also be represented as $\ell(\mathrm{h}, (x, y)) = 1 - \mathrm{h}(y|x)$, and it is specially suitable for classification tasks since it quantifies the classification error. The MRC minimizes the worst-case expected 0-1 loss with respect to distributions in uncertainty sets $\mathcal{U}$ defined in (2.1).

The mean vector $\boldsymbol{\tau} = [\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(m)}]^{\mathrm{T}}$ is an estimate on the expectation with respect to the true underlying distribution p^* of the feature mapping $\mathbb{E}_{\mathrm{p}^*(x,y)}\Phi(x, y)$. We will consider expectation estimates obtained as the sample average, where each component $j = 1, \ldots, m$ is calculated as

$$\tau^{(j)} = \frac{1}{n}\sum_{i=1}^{n} \Phi^{(j)}(x_i, y_i)$$

obtained from the n training samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, assumed to be independent samples drawn from p^*. The feature mapping we will use is linear and is obtained by the one-hot-encoding introduced in (1.4), so the dimension m is the product between the length of each sample d, and the number of possible labels $|\mathcal{Y}|$.

The confidence vector $\boldsymbol{\lambda} = [\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(m)}]^{\mathrm{T}}$ is an estimate of the mean vector accuracy, and controls the size of the uncertainty set. We consider the sample standard deviation, and it can be obtained using the training sample as

$$\lambda^{(j)} = \frac{\sqrt{\sum_{i=1}^{n}(\Phi^{(j)}(x_i, y_i) - \tau^{(j)})^2}}{n}$$

component-wise for $j = 1, \ldots, m$.

The minimax risk optimization problem for 0-1 loss

$$\mathscr{P}_{MRC} : \min_{\mathrm{h}\in\mathcal{F}} \sup_{\mathrm{p}\in\mathcal{U}} \mathbb{E}_{\mathrm{p}}\ell(\mathrm{h}, (x, y))$$

for $\mathcal{U}$ in (2.1), determines the learning stage of MRCs.

Definition 2.1.1. (Mazuelas et al., 2023) A classification rule $\mathrm{h}^{\mathcal{U}}$ is a *0-1 MRC* for the uncertainty set $\mathcal{U}$ if

$$\mathrm{h}^{\mathcal{U}} \in \arg \inf_{\mathrm{h}\in\mathrm{T}(\mathcal{X},\mathcal{Y})} \sup_{\mathrm{p}\in\mathcal{U}} \ell(\mathrm{h}, \mathrm{p}).$$

Moreover, the following theorem from (Mazuelas et al., 2023) shows how 0-1 MRCs can be determined by a linear-affine combination of the feature mapping, and the coefficients of such combination can be obtained by solving

$$\min_{\boldsymbol{\mu}} \; 1 \; - \boldsymbol{\tau}^{\mathrm{T}} \boldsymbol{\mu} + \boldsymbol{\lambda}^{\mathrm{T}} |\boldsymbol{\mu}| + \varphi(\boldsymbol{\mu}), \tag{2.2}$$

where $\boldsymbol{\mu}$ is the learning parameter in $\mathbb{R}^m$ and

$$\varphi(\boldsymbol{\mu}) = \max_{x \in \mathcal{X}, \mathcal{C} \subseteq \mathcal{Y}} \left(\frac{\sum_{y \in \mathcal{C}} \Phi(x,y)^{\mathrm{T}} \boldsymbol{\mu} - 1}{|\mathcal{C}|} \right), \tag{2.3}$$

implicitly assuming $\mathcal{C} \neq \emptyset$.

Theorem 2.1.1. *Let $\boldsymbol{\mu}^*$ be a solution of the optimization problem in (2.2), and $\mathcal{U}$ be an uncertainty set of the form (2.1) that satisfies one of the following statements:*

(i) *The set $\mathcal{X}$ is finite and there exists a probability measure p in $\mathcal{U}$.*

(ii) *There exists a probability measure p in $\mathcal{U}$ such that $|\mathbb{E}_{\mathrm{p}}\{\Phi(x,y)^{(i)}\} - \boldsymbol{\tau}^{(i)}| < \lambda^{(i)}$ for any $i \in \{1, 2, \ldots, m\}$ with $\lambda^{(i)} > 0$ and*

   (a) $\lambda^{(i)} > 0$ for all $i \in \{1, 2, \ldots, m\}$ or

   (b) the support of the r.v. $\Phi(x,y)$ for $(x,y) \sim \mathrm{p}$ is not contained in a proper affine subspace of $\mathbb{R}^m$.

If a classification rule $\mathrm{h}^{\mathcal{U}} \in \mathrm{T}(\mathcal{X}, \mathcal{Y})$ satisfies

$$\mathrm{h}^{\mathcal{U}}(y|x) \geq \Phi(x,y)^{\mathrm{T}} \boldsymbol{\mu}^* - \varphi(\boldsymbol{\mu}^*), \forall x \in \mathcal{X}, y \in \mathcal{Y},$$

then $\mathrm{h}^{\mathcal{U}}(y|x)$ is a 0-1 MRC for $\mathcal{U}$.

Proof. See Appendix B in (Mazuelas et al., 2023). □

2.2 MRCs with 0-1 loss and fixed marginals

The uncertainty set can also include an additional constraint that fixes the instances' marginal distribution p_x with the empirical marginal distribution p_x^n as

$$\mathcal{V} = \{\mathrm{p} \in \Delta(\mathcal{X} \times \mathcal{Y}) : |\mathbb{E}_{\mathrm{p}}\{\Phi\} - \boldsymbol{\tau}| \preceq \boldsymbol{\lambda} \text{ and } \mathrm{p}_x = \mathrm{p}_x^n\}. \tag{2.4}$$

In this case, the MRC classifier and its corresponding optimization problem in order to obtain it are given in the following result from (Mazuelas et al., 2023):

Theorem 2.2.1. *Let $\boldsymbol{\tau}, \boldsymbol{\lambda} \in \mathbb{R}^m$ be such that the uncertainty set $\mathcal{V}$ in* (2.4) *is not empty. If $\boldsymbol{\mu}^*$ is a solution of the optimization problem*

$$\min_{\boldsymbol{\mu}} \; 1 \; - \boldsymbol{\tau}^{\mathrm{T}} \boldsymbol{\mu} + \boldsymbol{\lambda}^{\mathrm{T}} |\boldsymbol{\mu}| + \frac{1}{n} \sum_{i=1}^{n} \varphi(\boldsymbol{\mu}, x_i), \tag{2.5}$$

where

$$\varphi(\boldsymbol{\mu}, x) = \max_{\mathcal{C} \subseteq \mathcal{Y}} \left(\frac{\sum_{y \in \mathcal{C}} \Phi(x, y)^{\mathrm{T}} \boldsymbol{\mu} - 1}{|\mathcal{C}|} \right), \tag{2.6}$$

and $\mathrm{h}^{\mathcal{V}}$ is the classification rule

$$\mathrm{h}^{\mathcal{V}}(y|x) = \left(\Phi(x, y)^T \boldsymbol{\mu}^* - \varphi(\boldsymbol{\mu}^*, x) \right)_+, \forall x \in \mathcal{X}, y \in \mathcal{Y},$$

where the subscript $\boldsymbol{v}_+$ denotes the vector given by the component-wise positive part of $\boldsymbol{v}$, then, $\mathrm{h}^{\mathcal{V}}$ is a 0-1 MRC for $\mathcal{V}$, that is

$$\mathrm{h}^{\mathcal{V}} \in \arg \inf_{\mathrm{h} \in \mathrm{T}(\mathcal{X}, \mathcal{Y})} \sup_{\mathrm{p} \in \mathcal{V}} \ell(\mathrm{h}, \mathrm{p}).$$

Proof. See Appendix C in (Mazuelas et al., 2023). □

Chapter 3

Subgradient methods

The subgradient method is a simple algorithm for minimizing a nondifferentiable convex function. It follows the idea of the ordinary gradient method for differentiable functions.

3.1 Gradient descent method

The gradient descent (GD) is an iterative optimization algorithm used to find a global minimum/maximum of a given function. However, this method does not work for all functions. In order to minimize, it needs to satisfy the following conditions: differentiability and convexity. The first condition guarantees a derivative for each point in the domain of the function, whereas the second one implies that a local minimum is also a global one. For a maximum, concavity is required instead of convexity.

Since the gradient can be interpreted as the fastest increasing direction of the function, this method iteratively calculates the next point by moving on the inverse direction in order to minimize.

Suppose f is the function we want to minimize, i.e., find the vector x^* such that

$$x^* \in \arg\min_x f(x).$$

Algorithm 1 describes the pseudocode for the gradient descent. It starts from the point x_0 and iterates until the gradient of the objective function at the current point, $\nabla f(x^{(k)})$, is zero. Each iteration k comprises the computation of the descent direction, p_k, the one determined by the opposite direction of the gradient, and the step α_k, determining how much to move. Then, the

update consists of determining the new point $x^{(k+1)}$ by moving α_k units on the descent direction, from the current point $x^{(k)}$, and updating the iteration number by $k+1$.

Algorithm 1 Gradient descent

1: $k = 0,\ x^{(k)} \in \mathbb{R}^n$
2: **repeat**
3: Compute the descent direction $p_k \leftarrow -\nabla f(x^{(k)})$
4: Compute the step of descent α_k
5: Update $x^{(k+1)} \leftarrow\ x_k + \alpha_k p_k$
6: $k \leftarrow k+1$
7: **until** $\nabla f(x^{(k)} = 0$

The definition of the descent step is also known as *line search*, since the selection of the step size determines where along the line $\{x + \alpha p \mid \alpha \in \mathbb{R}^+\}$ will be the next iterate.

However, the functions we want to minimize described in (2.2) and (2.5) are not differentiable, so that we cannot compute the gradient for every point. Instead, as its name suggests, the subgradient method uses subgradients in order to compute the new points at each iteration.

3.2 Subgradients

The subgradient is an extension of the notion of gradient for the cases where the latter cannot be computed, that is, where the function is not differentiable.

Definition 3.2.1. A vector $\boldsymbol{g} \in \mathbb{R}^n$ is a *subgradient* of $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ at $x \in domf$ if for all $z \in domf$,

$$f(z) \geq f(x) + \boldsymbol{g}^{\mathrm{T}}(z - x). \tag{3.1}$$

The set of subgradients of f at the point x is called the *subdifferential* of f at x, and is denoted by $\partial f(x)$.

The definition of subdifferential is motivated by cases such as the one shown in Figure 3.1. We can see in the figure that in x_1, where the function f is differentiable, its gradient is a subgradient and, in fact, it is unique. Whereas in x_2, where the function is not differentiable, there are illustrated two subgradients, g_2, g_3, and actually there are infinitely many.

Furthermore, if f is convex and $\partial f(x) = \{\boldsymbol{g}\}$, then f is differentiable at x and $\boldsymbol{g} = \nabla f(x)$, where ∇ indicates the gradient of f. A function f is called *subdifferentiable* at x if there exists at least one subgradient at x.

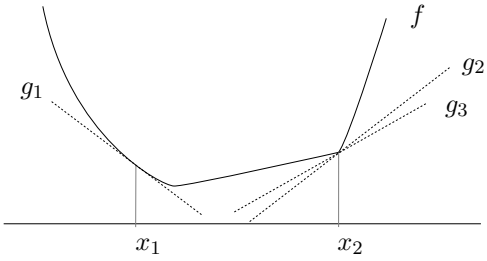


Figure 3.1: This figure shows the different subgradients of the convex function f according to different points. At x_1, the convex function f is differentiable, and g_1 is the unique subgradient at x_1. At the point x_2, f is not differentiable and has infinitely many subgradients. Two of them are g_2 and g_3.

Example 3.2.1. Let us consider the absolute value $f(x) = |x|$. For $x < 0$ and $x > 0$, the absolute value is differentiable so the subgradient is unique:

$$\partial f(x) = \{\nabla(-x)\} = \{-1\}, \text{ if } x < 0,$$

$$\partial f(x) = \{\nabla x\} = \{1\}, \text{ if } x > 0.$$

At $x = 0$ the subdifferential is defined by the inequality $|z| \geq \boldsymbol{g}z$ for all z, which is satisfied if and only if $\boldsymbol{g} \in [-1, 1]$. Therefore,

$$\partial f(0) = [-1, 1].$$

The whole set of inequalities (3.1) for $z \in dom f$ can be seen as a set of linear constraints defining the set $\partial f(x)$. Therefore, by definition, the subdifferential is a closed convex set.

3.2.1 Existence of subgradients

From Figure 3.1 we can see that for any point x in the domain of f a subgradient can be defined. In fact, we claim that for a convex function f, there is at least a subgradient for any point in the interior of its domain. Equivalently, the subdifferential of a convex function is nonempty. In order to prove these statements, we recall the *supporting hyperplane theorem* from (Boyd and Vandenberghe, 2004).

Definition 3.2.2. The *epigraph* of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\mathbf{epi} f = \{(x,t) \ : \ x \in dom f, f(x) \leq t\},$$

which is a subset of $\mathbb{R}^{n+1}$.

Definition 3.2.3. A *supporting hyperplane* of a set $C \subseteq \mathbb{R}^n$ at a boundary point $x_0 \in \mathbf{bd}(C) = \mathbf{cl}(C) \backslash \mathbf{int}(C)$ is the hyperplane

$$\{x \mid a^T x = a^T x_0\}$$

for a fixed $a \neq 0$ satisfying $a^T x \leq a^T x_0$ for all $x \in \mathcal{C}$.

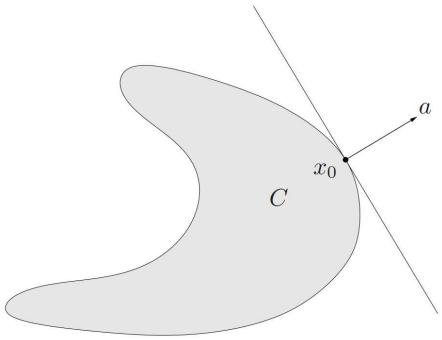


Figure 3.2: The line illustrated in this figure defined by $\{x \mid a^T x = a^T x_0\}$ is a hyperplane for the set C at x_0. Also said as $\{x \mid a^T x = a^T x_0\}$ supports C.

In other words, this means that the set C is tangent to the hyperplane $\{x \mid a^T x = a^T x_0\}$ at x_0, and the set is entirely contained in one of the half-spaces bounded by the hyperplane $\{x \mid a^T x = a^T x_0\}$, that is, C is contained in $\{x \mid a^T x \leq a^T x_0\}$, as illustrated in Figure 3.2.

The connection between convex sets and convex functions is via their epigraph: a set C is convex if and only if its epigraph is a convex set. In order to prove the existence of subgradients, or, equivalently, the nonemptyness of the subdifferential, we can analyze the subdifferential of a convex function as a convex set. In fact, the following theorem describes the existence of a supporting hyperplane for convex sets:

Theorem 3.2.1. ***Supporting hyperplane theorem.*** *For any convex set C and any boundary point $x_0 \in$ **bd**(C), there exists a supporting hyperplane for C at x_0.*

The proof of this theorem requires the separation theorem and can be found in Section 2.5.2 from (Boyd and Vandenberghe, 2004).

The following theorem in (Boyd et al., 2022) proves the existence of subgradients:

Theorem 3.2.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then, $\partial f(x)$ is nonempty for any $x \in \boldsymbol{int}(domf)$, where x is an interior point.*

Proof. Since f is convex, $\mathbf{epi}f$ is a convex set. Let $(x, f(x))$ be a boundary point of the epigraph and apply Theorem 3.2.1 to $\mathbf{epi}f$ at $(x, f(x))$. That is, there exist $(a, b) \in \mathbb{R}^n \times \mathbb{R}$, not both zero, such that

$$\begin{bmatrix} a \\ b \end{bmatrix}^T \left(\begin{bmatrix} z \\ t \end{bmatrix} - \begin{bmatrix} x \\ f(x) \end{bmatrix} \right) = 0$$

for all $(z, t) \in \mathbf{epi}f$. In particular, for $(z, f(z)) \in \mathbf{epi}f$, we have

$$a^T(z - x) + b(f(z) - f(x)) \leq 0. \tag{3.2}$$

If $b \neq 0$, dividing by b in (3.2), we have that

$$f(x) \geq f(z) + \left(-\frac{a}{b}\right)^T (z - x).$$

Then, by definition, this implies that $-a/b$ is a subgradient of f at x, i.e., $-a/b \in \partial f(x)$. So that we found a subgradient for an arbitrary point $x \in \mathbf{int}(domf)$.

If $b = 0$, the inequality (3.2) simplifies to

$$a^T(z - x) \leq 0 \tag{3.3}$$

for all $z \in domf$. In particular for those z in a neighborhood of x. If x is an interior point, the inequality in (3.3) implies that $a = 0$. But this contradicts the assumption since both a and b would be zero.

Therefore, b must be nonzero and then we already proved that the subdifferential of f at an arbitrary point x is nonempty. □

3.2.2 Calculus of subgradients

Describing the complete subdifferential is generally a hard task, but many algorithms for nondifferentiable convex optimization require only one subgradient at each step. The subgradient method is one of them.

In this section, we introduce some necessary rules for the calculus of a subgradient in our problem. The proofs involve support functions and directional derivatives theory not introduced in these pages. This machinery is

detail in Section C2 and Section D1 from (Hiriart-Urruty and Lemaréchal, 2004). The following theorem, Theorem 4.1.1 from (Hiriart-Urruty and Lemaréchal, 2004), explains the behavior of subdifferentials when positive combinations of functions are given.

Theorem 3.2.3. *Let f_1, f_2 be two convex function from $\mathbb{R}^n$ to $\mathbb{R}$ and t_1, t_2 be positive. Then*

$$\partial(t_1 f_1 + t_2 f_2)(x) = t_1 \partial f_1(x) + t_2 \partial f_2(x), \quad \forall x \in \mathbb{R}^n.$$

Theorem 3.2.3 proves the following (i) and (ii) rules.

(i) **Nonnegative scaling.** For $\alpha \geq 0$,

$$\partial(\alpha f)(x) = \alpha \partial f(x).$$

(ii) **Sum.** Let $f_1, f_2, \ldots, f_m$ be convex functions. Then,

$$\partial \left(\sum_{i=1}^{m} f_i(x) \right) = \sum_{i=1}^{m} \partial f_i(x).$$

We will consider one additional rule to compute a subgradient for the problem given either by (2.2)-(2.3) or (2.5)-(2.6). This rule determines how to calculate a subgradient of the maximum of convex functions.

(iii) **Pointwise maximum.** Let f be the pointwise maximum of convex functions $f_1, f_2, \ldots, f_m$, i.e.,

$$f = \max_{i=1,\ldots,m} f_i$$

where f_i is subdifferentiable for all i. Let k be an index where the functions take the maximum at point x, that is $f_k(x) = f(x)$, and let $\boldsymbol{g} \in \partial f_k(x)$. Then,

$$\boldsymbol{g} \in \partial f(x).$$

Proof. Let f and f_k be two functions defined as before. Then, for all $z \in dom f$,

$$f(z) \geq f_k(z) \geq f_k(x) + \boldsymbol{g}^{\mathrm{T}}(z - x) = f(x) + \boldsymbol{g}^{\mathrm{T}}(z - x).$$

□

In order to find a subgradient of the maximum, it is enough to find a subgradient of a function taking the maximum at the point.

Example 3.2.2. Let $f_1(x) = (x+1)^2$, $f_2(x) = (x-1)^2$ and define their maximum as $h(x) = \max_x\{f_1(x), f_2(x)\}$. In order to find a subgradient of h at x we need to see which of the functions achieves the maximum at x.

If $x < 0$, we have $f_2(x) > f_1(x)$ so $h(x) = f_2(x)$. Then, if $\boldsymbol{g} \in \partial f_2(x)$ we have that $\boldsymbol{g} \in \partial h(x)$. Since $f_2(x)$ is differentiable, its subdifferential consists of its gradient, so $\boldsymbol{g}(x) = \nabla f_2(x) = 2(x-1)$.

Similarly when $x > 0$, we have $\boldsymbol{g}(x) = \nabla f_1(x) = 2(x+1)$.

At $x = 0$, we have $h(0) = f_1(0) = f_2(0)$ so we can choose either a subgradient of f_1 or f_2 at $x = 0$ in order to obtain a subgradient of the maximum. So, for instance, we have $2(x+1) \in \partial h(0)$.

3.3 Subgradient method

The main difference of the subgradient method with respect to the GD is the direct application to nondifferentiable functions. Unlike the GD, the subgradient method is not a descendent one; the function value can, and often does, increase.

On its basic case, the subgradient method uses the simple iteration

$$x^{(k+1)} = x^{(k)} - \alpha_k \boldsymbol{g}^{(k)},$$

where $x^{(k)}$ is the point at iteration k, $\boldsymbol{g}^{(k)}$ is any subgradient of f at $x^{(k)}$, i.e., $\boldsymbol{g}^{(k)} \in \partial f(x^{(k)})$, and $\alpha_k > 0$ is the k-th step size. At each iteration we take a step in the direction of a negative subgradient.

Recall that when f is differentiable, the unique subgradient is the gradient itself, so the subgradient method reduces to GD.

It can happen that $-\boldsymbol{g}^{(k)}$ is not a descending direction for f at $x^{(k)}$, so we have $f(x^{(k+1)}) > f(x^{(k)})$. Therefore, as mentioned before, the subgradient method is not a descendent method and an iteration can increase the objective function value.

Exactly for this reason, different aspects about the algorithm might be tuned in order to obtain a proper approximation of the solution. It is important to remark that this method is not an exact one, so the result we obtain will be an approximation to the real solution. In fact, we will see how the following elements might influence the convergence of the algorithm to the real solution.

3.3.1 Evaluation approaches

As we mentioned, the subgradient method may worsen the result obtained in the previous iteration. Then, there are another approaches to evaluate the objective function that may perform better. For instance, a common approach is to keep track of the best point found so far, the one with smallest objective value.

Instead of using the last iterate as the solution, the best point technique selects the best iterate obtained so far, according to the objective function value. At each iteration k, the subgradient method computes the point $x^{(k)}$, calculates its corresponding objective function value, and compares it to the previous values corresponding to the previous points, choosing the best among all, i.e.,

$$f_{best}^{(k)} = \min\{f(x^{(0)}), \ldots, f(x^{(k)})\}.$$

In fact, it is not necessary to store all the values $f(x^{(0)}), f(x^{(1)}), \ldots, f(x^{(k)})$ at each iteration, note that the best objective function value can be iteratively computed as

$$f_{best}^{(k)} = \min\{f(x_{best}^{(k-1)}), f(x^{(k)})\}.$$

We describe the best point evaluation method in Algorithm 2. The best

Algorithm 2 Subgradient method for best point evaluation

1: $k = 0,\ x^{(k)} \in \mathbb{R}^n\ f_{best}^{(k)} \leftarrow f(x^{(k)})$
2: **repeat**
3: Find a subgradient $\boldsymbol{g}^{(k)} \in \partial f(x^{(k)})$
4: Update $x^{(k+1)} \leftarrow\ x^{(k)} + \alpha_k \boldsymbol{g}^{(k)}$
5: Calculate $f_{best}^{(k+1)} \leftarrow \min\{f(x^{(k+1)}), f_{best}^{(k)}\}$
6: $k \leftarrow k + 1$
7: **until** Stopping criterion is satisfied

point evaluation has several advantages: it obtains an improved solution quality by keeping track of the best objective function value. In this form, the final output is the best iterate obtained during the optimization process, leading to potentially better solutions. Moreover, it provides robustness against fluctuations and poor performance of some iterates, which is specially useful in non-smooth optimization scenarios.

Nevertheless, the computation of the objective function at each iteration requires much time. Additionally, if at some iteration the solution is worse, the subgradient is computed again, even if it is the same as the previous one. Moreover, if the algorithm returns a sequence of increasing objective

function values, the minimum will be achieved during the first iterations. In this case, we will not improve anymore and making more iterations results useless.

In order to exploit the subgradient method and take some improvement still during the last iterations, and to improve the time complexity, there are some other methods such as evaluating the mean of all the points found so far, also known as averaging technique.

The points generated by the algorithm can be averaged to produce a more stable sequence that converges better. The idea is to compute the average of all points up to the current iteration, rather than just relying on the last one as described in the pseudocode of Algorithm 3. Averaging process reduces the variance of the iterates, leading to a smoother convergence path, particularly useful in non-smooth optimization where subgradients can cause significant fluctuations. In any event, the choice of the evaluation approach

Algorithm 3 Subgradient method for averaging approach

1: $k = 0,\ x^{(k)} \in \mathbb{R}^n,\ x_{avg}^{(k)} \leftarrow x^{(k)}$
2: **repeat**
3: Find a subgradient $\boldsymbol{g}^{(k)} \in \partial f(x^{(k)})$
4: Update $x^{(k+1)} \leftarrow\ x^{(k)} + \alpha \boldsymbol{g}^{(k)}$
5: Update $x_{avg}^{(k+1)} \leftarrow k/(k+1) \cdot x_{avg}^{(k)} + 1/(k+1) \cdot x^{(k+1)}$
6: $k \leftarrow k + 1$
7: **until** Stopping criterion is satisfied

must be made by previously analyzing the behavior of the real objective function value, the one obtained by the points given by the subgradient method, even if not descendent ones. For example, we can see in Figure 3.3 the comparison of the objective function values for Haberman dataset with $\alpha = 1.5/\sqrt{i}$ and 10^4 number of iterations, considering the function given in (2.5)-(2.6), obtained by the subgradient method with previously described evaluation approaches (the usual method in blue, best point approach in yellow, and averaging technique in red).

This illustrates how using the mean gives a more stable value, without fluctuations, so that we obtain a good approximation of the minimum. Moreover, computing the mean does not require much time. Instead, choosing the best value obtained so far gives also a good approximation, but it is the most expensive option in terms of computational time, since it requires to compute the objective function at each iteration.

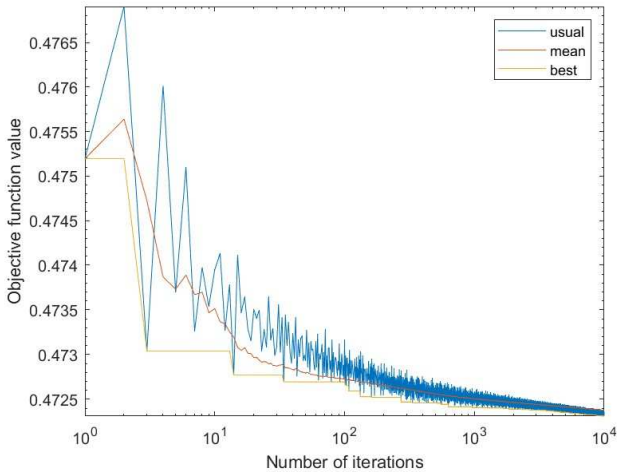


Figure 3.3: This figure shows the value of the objective function on the y axis, during the iterations, x axis. It is used the Haberman dataset with $\alpha = 1.5/\sqrt{i}$ and 10^4 number of iterations. The blue line shows the objective function value obtained by evaluating each point given by the algorithm; the red line, the function value using the mean of all the points obtained so far; and the yellow one, the objective function value obtained by choosing the best value found so far, i.e., the minimum.

3.3.2 Stopping criterion

The stopping criterion is a critical aspect that determines when the algorithm should terminate. Selecting an appropriate stopping criterion ensures that the algorithm stops after a reasonable amount of computational time, while still providing a good approximation to the optimal solution.

The most common stopping criteria are the maximum number of iterations and the improvement of the objective function. The first one sets a maximum number of iterations K after which the algorithm stops, as shown in Algorithm 4, regardless of whether it has converged. The second one, instead, terminates the algorithm when the improvement in the objective function value between iterations falls below a certain threshold ϵ, see Algorithm 5.

The maximum number of iterations approach is simple to implement and ensures that the algorithm stops within a known time frame. Nevertheless, it does not guarantee convergence to the optimal solution and may stop too early or run unnecessarily long.

On the other hand, the improvement method focuses on the actual improvement of the solution. It can avoid unnecessary iterations by stopping early

Algorithm 4 Stopping criterion: Maximum number of iterations

1: $k = 0$
2: **repeat**
3: Find a subgradient $\boldsymbol{g}^{(k)} \in \partial f(x^{(k)})$
4: Update $x^{(k+1)} \leftarrow x^{(k)} + \alpha_k \boldsymbol{g}^{(k)}$
5: $k \leftarrow k + 1$
6: **until** $k = K$

Algorithm 5 Stopping criterion: Improvement of the objective function

1: $k = 0, x^{(k)} \in \mathbb{R}^n, f_{current} \leftarrow f(x^{(k)})$
2: **repeat**
3: Update $f_{prev} \leftarrow f_{current}$
4: Find a subgradient $\boldsymbol{g}^{(k)} \in \partial f(x^{(k)})$
5: Update $x^{(k+1)} \leftarrow x^{(k)} + \alpha_k \boldsymbol{g}^{(k)}$
6: Evaluate $f_{current} \leftarrow f(x^{(k+1)})$
7: $k \leftarrow k + 1$
8: **until** $|f_{current} - f_{prev}| < \epsilon$

if significant progress is made, but it may take too long if the objective function improves slowly. Moreover, this method requires careful selection of the threshold ϵ.

Stopping criteria based on the duality gap

The duality gap is a concept used in optimization to measure the difference between the objective values of the primal and dual problems, for duality basic theory see Chapter 5 from (Boyd and Vandenberghe, 2004). From this chapter, recall the Lagrange duality problem: consider the following primal minimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0. \end{aligned} \tag{3.4}$$

Then, after computing its Lagrange dual function, the dual problem is explicitly

$$\begin{aligned} \underset{\nu}{\text{maximize}} \quad & -b^T \nu \\ \text{subject to} \quad & A^T \nu - c \geq 0. \end{aligned} \tag{3.5}$$

Let $p^\star$ and $d^\star$ be the optimal values of the minimization problem (3.4) (the primal) and its Langrange dual problem (3.5), respectively. Then, by

definition, $d^\star$ is the best lower bound for $p^\star$. In particular, the *weak duality* property is satisfied:

$$d^\star \leq p^\star.$$

We define as duality gap the difference between these optimal values, i.e., $p^\star - d^\star$, always nonnegative. The duality gap measures the difference between the best objective function value obtained (in the primal problem) and the best lower bound (from the dual problem).

This bound can be used to find a lower bound on the optimal value of a problem that is difficult to solve, since the dual problem is always convex, and in many cases can be solved efficiently. This criterion, defined by using these bounds, is more rigorous than simply observing the improvement in the objective function because it provides a direct measure of how much improvement is possible.

In Section 3.3 from (Boyd, 2014), we can find some lower bounds for the optimal value of the principal optimization problem. The main idea is to obtain a lower bound l_k at each iteration,

$$l_k \leq p^\star, \ \ \forall k.$$

The sequence $l_1, l_2, \ldots$ needs not increase, so we can keep track of the best lower bound found, the tighter one, that is,

$$l_{best}^{(k)} = \max\{l_1, \ldots, l_k\}.$$

We can use this information to define a stopping rule and terminate the algorithm when the difference $f_{best}^{(k)} - l_{best}^{(k)}$ is smaller than some threshold ϵ, where $f_{best}^{(k)}$ represents the best objective function value obtained until iteration k, the minimum one.

The pseudocode for this stopping criterion is shown in Algorithm 6. At each iteration, it computes the point by the subgradient method, its objective value and the respective lower bound. In both cases, it compares these two values iteratively to the ones obtained in the previous iteration, in order to choose the best one. The method performs while the difference between these values is smaller than some threshold ϵ.

This stopping criterion ensures that the algorithm stops when a near-optimal solution is found. It is based on the duality gap defined before, which provides a measure of how close the current solution is to the optimal solution.

Nevertheless, selecting the appropriate stopping criterion for subgradient methods depends on the specific problem at hand, computational resources, and desired solution accuracy. Often, a combination of criteria (e.g., maximum iterations and improvement threshold) is used to balance between convergence guarantee and computational efficiency.

Algorithm 6 Stopping criterion: Based on duality gap

1: $k = 0, x^{(k)} \in \mathbb{R}^n, f_{best}^{(k)} \leftarrow f(x^{(k)}), l_{best}^{(k)} \leftarrow -\epsilon$
2: **repeat**
3: Find a subgradient $\boldsymbol{g}^{(k)} \in \partial f(x^{(k)})$
4: Update $x^{(k+1)} \leftarrow x^{(k)} + \alpha_k \boldsymbol{g}^{(k)}$
5: Compute lower bound of the solution l_{k+1}
6: Evaluate $f_{k+1} \leftarrow f(x^{(k+1)})$
7: Update lower bound $l_{best}^{(k+1)} \leftarrow \max\{l_{k+1}, l_{best}^{(k)}\}$
8: Update best objective function value $f_{best}^{(k+1)} \leftarrow \min\{f_{k+1}, f_{best}^{(k)}\}$
9: $k \leftarrow k + 1$
10: **until** $|f_{best}^{(k)} - l_{best}^{(k)}| \leq \epsilon$

3.3.3 Step size

In subgradient methods, choosing an appropriate step size is crucial for the convergence and efficiency of the algorithm. The step size determines how far the algorithm moves along the direction of the subgradient at each iteration. The most common step size strategies are the following:

(i) **Constant step size.** A positive constant, independent of the iteration k,

$$\alpha_k = \alpha > 0.$$

The step size is fixed throughout the iterations. It is simple to implement but might not converge to the optimal solution so it is effective for problems where an approximate solution is acceptable.

Choosing the right step size is crucial: if it is too large, the algorithm may oscillate or diverge; if it is too small, the algorithm may converge very slowly. Typically it requires experimentation or domain knowledge to set an appropriate α.

(ii) **Constant step length.**

$$\alpha_k = \gamma / \|\boldsymbol{g}^{(k)}\|_2,$$

where $\gamma > 0$ and $\boldsymbol{g}^{(k)}$ is the subgradient used at the moment. This means that the distance between each step is always γ, that is

$$\|x^{(k+1)} - x^{(k)}\|_2 = \|x^{(k)} - \frac{\gamma}{\|\boldsymbol{g}^{(k)}\|_2} \boldsymbol{g}^{(k)} - x^{(k)}\|_2 = \gamma.$$

This step provides a way to maintain consistent progress by ensuring each iteration moves the same distance in the direction of the subgradient. While it may not guarantee convergence to the exact optimal

solution, it offers a practical balance between simplicity and effective progress. The key challenge lies in selecting an appropriate step length γ, which requires careful consideration of the problem's characteristics.

(iii) **Square summable but not summable.** The step sizes satisfy

$$\alpha_k \geq 0, \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty, \qquad \sum_{k=1}^{\infty} \alpha_k = \infty.$$

One typical example is $\alpha_k = a/(b+k)$, where $a > 0$ and $b \geq 0$.

This approach obtains a balance ensuring the algorithm makes steady progress towards the optimal solution while gradually refining the steps to converge accurately. Larger initial steps help explore the solution space, while diminishing steps allow the algorithm to exploit and fine-tune the solution as it progresses.

(iv) **Nonsummable diminishing step size.** The step sizes satisfy

$$\alpha_k \geq 0, \qquad \lim_{k\to\infty} \alpha_k = 0, \qquad \sum_{k=1}^{\infty} \alpha_k = \infty.$$

Step sizes that satisfy this condition are called *diminishing step size rules*. A typical example is $\alpha_k = a/\sqrt{k}$, where $a > 0$.

Similar to the previous case, the divergent behaviour of the sum of the steps guarantees the progress of the algorithm. While the diminishing nature ensures that the steps become smaller over time, allowing the algorithm to refine the solution and avoid overshooting.

(v) **Nonsummable diminishing step length.** The step sizes are chosen as $\alpha_k = \gamma_k/\|g^{(k)}\|_2$, where

$$\gamma_k \geq 0, \qquad \lim_{k\to\infty} \gamma_k = 0, \qquad \sum_{k=1}^{\infty} \gamma_k = \infty.$$

Follows the same idea as previous approaches, its nonsummable property ensures the progress while the diminishing property tunes the step in order to obtain more accuracy.

Note that these step sizes are already determined before the algorithm is run, and they do not depend on any data computed during the algorithm, apart from the current subgradient. This is very different from the exact line search used in GD, which depends on the current point and search direction.

3.3.4 Convergence results

The convergence behavior of subgradient methods depends significantly on the parameters previously defined during this chapter. Mainly, the choice of the step size. Different types of step size rules can lead to different convergence rates and properties. Understanding these properties helps in selecting the appropriate step size strategy for different types of optimization problems.

In this section we discuss the convergence results for previously introduced step sizes. For constant step size and constant step length the following is satisfied:

$$\lim_{k\to\infty} f_{best}^{(k)} - p^\star < \epsilon,$$

where $p^\star$ denotes the optimal value of the primal problem, and ϵ is a function of the step size parameter α, and decreases with it. Then, the subgradient method is guaranteed to find an ϵ-suboptimal point within a finite number of steps, meaning that the objective function value corresponding to that point differs in ϵ units from the optimal value $p^\star$. So the method does not guarantee convergence to the exact optimal solution, but an ϵ-suboptimal solution provides a practical way to obtain a solution that is close to optimal within a predefined tolerance; by letting ϵ be small enough we can obtain a better approximation.

For the diminishing step size and step length rules, and therefore also the square summable but not summable one, the method is guaranteed to converge to the optimal value, i.e.,

$$\lim_{k\to\infty} f(x^{(k)}) = p^\star.$$

Convergence proof

We present the convergence proofs described in Section 3 from (Boyd, 2014). For these proofs some assumptions are needed. We will assume there is a minimizer $x^\star$ of f, and that the norm of the subgradients is bounded. That is, there exists a G such that

$$\|\boldsymbol{g}^{(k)}\|_2 \leq G, \quad \forall k.$$

We also assume that there is a real value R satisfying

$$R \geq \|x^{(1)} - x^\star\|_2,$$

that can be interpreted as an upper bound on the distance from $x^{(1)}$ to the optimal set $X^\star$, that is, the set of all optimal solutions: $\mathbf{dist}(x^{(1)}, X^\star)$.

For the subgradient methods, the convergence proofs are based on the *Euclidean distance to the optimal set*, instead of focusing on the objective function value, which often increases.

We now present some inequalities. For $x^\star$ an arbitrary optimal point, and exploiting the property of the euclidean norm $\|\boldsymbol{v}\|_2^2 = \boldsymbol{v}^T\boldsymbol{v}$ for a vector $\boldsymbol{v}$, we have the following:

$$\begin{aligned}
\|x^{(k+1)} - x^\star\|_2^2 &= \|x^{(k)} - \alpha_k \boldsymbol{g}^{(k)} - x^\star\|_2^2 \\
&= (x^{(k)} - x^\star - \alpha_k \boldsymbol{g}^{(k)})^T (x^{(k)} - x^\star - \alpha_k \boldsymbol{g}^{(k)}) \\
&= (x^{(k)} - x^\star)^T (x^{(k)} - x^\star) - 2(x^{(k)} - x^\star)^T \alpha_k \boldsymbol{g}^{(k)} + \\
&\quad (\alpha_k \boldsymbol{g}^{(k)})^T (\alpha_k \boldsymbol{g}^{(k)}) \\
&= \|x^{(k)} - x^\star\|_2^2 - 2\alpha_k \boldsymbol{g}^{(k)T}(x^{(k)} - x^\star) + \alpha_k^2 \|\boldsymbol{g}^{(k)}\|_2^2 \\
&\leq \|x^{(k)} - x^\star\|_2^2 - 2\alpha_k (f(x^{(k)} - p^\star) + \alpha_k^2 \|\boldsymbol{g}^{(k)}\|_2^2,
\end{aligned}$$

where $p^\star = f(x^\star)$. The last inequality follows from the definition of the subgradient, for $x^\star, x^{(k)} \in dom f$. Applying this inequality recursively, we obtain

$$\|x^{(k+1)} - x^\star\|_2^2 \leq \|x^{(1)} - x^\star\|_2^2 - 2\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - p^\star) + \sum_{i=1}^{k} \alpha_i^2 \|\boldsymbol{g}^{(i)}\|_2^2.$$

Reordering and using the bound R,

$$\begin{aligned}
2\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - p^\star) &\leq 2\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - p^\star) + \|x^{(k+1)} - x^\star\|_2^2 \\
&\leq \|x^{(1)} - x^\star\|_2^2 + \sum_{i=1}^{k} \alpha_i^2 \|\boldsymbol{g}^{(i)}\|_2^2 \\
&\leq R^2 + \sum_{i=1}^{k} \alpha_i^2 \|\boldsymbol{g}^{(i)}\|_2^2.
\end{aligned}$$

In particular,

$$2\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - p^\star) \leq R^2 + \sum_{i=1}^{k} \alpha_i^2 \|\boldsymbol{g}^{(i)}\|_2^2, \tag{3.6}$$

and combining (3.6) with

$$\sum_{i=1}^{k} \alpha_i (f(x^{(i)}) - p^\star) \geq \left(\sum_{i=1}^{k} \alpha_i\right) \min_{i=1,\ldots,k} (f(x^{(i)}) - p^\star) = \left(\sum_{i=1}^{k} \alpha_i\right) (f_{best}^{(k)} - p^\star)$$

we obtain the inequality

$$f_{best}^{(k)} - p^\star = \min_{i=1,\dots,k} f(x^{(i)}) - p^\star \leq \frac{R^2 + \sum_{i=1}^k \alpha_i^2 \|\boldsymbol{g}^{(i)}\|_2^2}{2\sum_{i=1}^k \alpha_i}. \tag{3.7}$$

Using the fact that the subgradients are bounded by G, we finally obtain

$$f_{best}^{(k)} - p^\star \leq \frac{R^2 + G^2 \sum_{i=1}^k \alpha_i^2}{2\sum_{i=1}^k \alpha_i}. \tag{3.8}$$

Now, from this last inequality (3.8), we can deduce several convergent results.

Constant step size. For $\alpha_k = \alpha$ constant, we have

$$f_{best}^{(k)} - p^\star \leq \frac{R^2 + G^2\alpha^2 k}{2\alpha k}.$$

The righthand side converges to $G^2\alpha/2$ as $k \to \infty$.

Thus, for the subgradient method with fixed constant step size α, $f_{best}^{(k)}$ converges to within $G^2\alpha/2$ of optimal.

Constant step length. Using the step size $\alpha_k = \gamma/\|\boldsymbol{g}^{(k)}\|_2$, inducing a constant step length of size γ, the inequality (3.7) becomes

$$f_{best}^{(k)} - p^\star \leq \frac{R^2 + \gamma^2 k}{2\sum_{i=1}^k \gamma/\|\boldsymbol{g}^{(i)}\|_2} \leq \frac{R^2 + \gamma^2 k}{2\gamma k/G}$$

by using the bound G for the subgradient. The righthand side converges to $G\gamma/2$ as $k \to \infty$. Therefore, in this case the subgradient method converges to within $G\gamma/2$ of optimal.

Square summable but not summable. Let the step size be such that

$$\|\alpha\|_2^2 = \sum_{k=1}^\infty \alpha_k^2 < \infty, \qquad\qquad \sum_{k=1}^\infty \alpha_k = \infty.$$

Then,

$$f_{best}^{(k)} - p^\star \leq \frac{R^2 + G^2\|\alpha\|_2^2}{2\sum_{i=1}^k \alpha_i},$$

which converges to 0 as $k \to \infty$ and thus, the subgradient method converges to the optimal value.

Diminishing step size rule. If the step size is diminishing, that is,

$$\alpha_k \geq 0, \qquad \lim_{k\to\infty} \alpha_k = 0, \qquad \sum_{k=1}^\infty \alpha_k = \infty,$$

the righthand side of (3.8) converges to zero, implying that the subgradient method converges.

In fact, let $\epsilon > 0$, then there exists $N_1 \in \mathbb{Z}$ such that $\alpha_i \leq \epsilon/G^2$ for all $i > N_1$. There is also an $N_2 \in \mathbb{Z}$ such that

$$\sum_{i=1}^{N_2} \alpha_i \geq \frac{1}{\epsilon}\left(R^2 + G^2 \sum_{i=1}^{N_1} \alpha_i^2\right),$$

since $\sum_{i=1}^{\infty} \alpha_i = \infty$. Let $N = \max\{N_1, N_2\}$. Then for $k > N$, we have

$$\begin{aligned}
\frac{R^2 + G^2 \sum_{i=1}^{k} \alpha_i^2}{2\sum_{i=1}^{k} \alpha_i} &\leq \frac{R^2 + G^2 \sum_{i=1}^{N_1} \alpha_i^2}{2\sum_{i=1}^{k} \alpha_i} + \frac{G^2 \sum_{i=N_1+1}^{k} \alpha_i^2}{2\sum_{i=1}^{N_1} \alpha_i + 2\sum_{i=N_1+1}^{k} \alpha_i} \\
&\leq \frac{R^2 + G^2 \sum_{i=1}^{N_1} \alpha_i^2}{(2/\epsilon)\left(R^2 + G^2 \sum_{i=1}^{N_1} \alpha_i^2\right)} + \frac{G^2 \sum_{i=N_1+1}^{k} \left(\epsilon\alpha_i/G^2\right)}{2\sum_{i=1}^{N_1} \alpha_i} \\
&= \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.
\end{aligned}$$

Nonsummable diminishing step lengths. Suppose $\alpha_k = \gamma_k / \|\boldsymbol{g}^{(k)}\|_2$, where

$$\gamma_k \geq 0, \qquad \lim_{k\to\infty} \gamma_k = 0, \qquad \sum_{k=1}^{\infty} \gamma_k = \infty.$$

In this case, inequality (3.7) becomes

$$f_{best}^{(k)} - p^\star \leq \frac{R^2 + \sum_{i=1}^{k} \gamma_k^2}{2\sum_{i=1}^{k} \gamma_k / \|\boldsymbol{g}^{(i)}\|_2} \leq \frac{R^2 + \sum_{i=1}^{k} \gamma_k^2}{(2/G)\sum_{i=1}^{k} \gamma_i},$$

which converges to zero as $k \to \infty$.

3.3.5 Subgradient method for constrained problems

The subgradient method described before applies to the general unconstrained case. It can be extended to solve inequality constrained problems of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, 2, \ldots, m, \end{array} \tag{3.9}$$

where f_i are convex functions.

This algorithm iterates as the subgradient method. The difference is on the subgradient: it has to choose either an objective subgradient, i.e., a subgradient of the objective function, or one of the constraint functions.

If the current point is feasible, the method chooses an objective subgradient and iterates so as to decrease the objective function value. While if the point is infeasible, it uses a subgradient of a violated constraint function, any of them if there are more than one. In this case, the algorithm tries to decrease the value of the violated constraint function, in order to make it valid.

The method iterates in the following way:

$$x^{(k+1)} = x^{(k)} - \alpha_k \boldsymbol{g}^{(k)},$$

where $\alpha_k > 0$ is a step size, and $\boldsymbol{g}^{(k)}$ is a subgradient chosen as

$$\boldsymbol{g}^{(k)} \in \begin{cases} \partial f_0(x^{(k)}), & \text{if } f_i(x^{(k)}) \leq 0, \quad i = 1 \ldots, m, \\ \\ \partial f_j(x^{(k)}), & \text{if } f_j(x^{(k)}) > 0, \quad \text{for such a } j. \end{cases}$$

As in the subgradient method, the steps given in this algorithm are not necessarily descendent, so the iterations can be infeasible, meaning that the point obtained after computing an iteration may be an infeasible point. The key is again to tune those parameters that enable us to obtain a proper result efficiently.

In any case, this efficience is hard to obtain if a dataset has a lot of instances. Computing the mean can be very slow and the algorithm will take much time to return a reasonable good result. So, in order to make it faster, we introduce a variant of the subgradient method: the Stochastic Subgradient method.

3.4 Stochastic Subgradient Method

These methods are extensions of the subgradient method to stochastic settings and are particularly useful for problems where the exact evaluation of gradients or subgradients is computationally infeasible due to the large size of the dataset. In large-scale settings, computing the exact subgradient can be impractical. Instead, a stochastic subgradient is used, which is an unbiased estimator of the true subgradient (Boyd et al., 2018).

Stochastic methods are particularly common in scenarios where the objective function is a sum of a large number of terms. This happens almost every

time we are doing ERM over a number of samples. In order to simplify, suppose we want to solve the following problem

$$\min \sum_{i=1}^{m} f_i(x),$$

where f_i is a convex function for every i.

Recall how subgradients behave with the sum,

$$\partial \sum_{i=1}^{m} f_i(x) = \sum_{i=1}^{m} \partial f_i(x).$$

The Classical Subgradient Method (CSM), the one defined in Section 3.3, would repeat

$$x^{(k+1)} = x^{(k)} - \alpha_k \sum_{i=1}^{m} \boldsymbol{g_i}^{(k)}, \quad k = 1, 2, 3, \ldots,$$

where $\boldsymbol{g_i}^{(k)} \in \partial f_i(x^{(k)})$. So if m is quite large, we have to compute $m - 1$ sums at each iteration of CSM in order to obtain a subgradient.

Instead, the Stochastic Subgradient Method (SSM) just computes the subgradient of one f_i at iteration. That is, a step of the stochastic method iterates as

$$x^{(k+1)} = x^{(k)} - \alpha_k \boldsymbol{g_{i_k}}^{(k)}, \quad k = 1, 2, 3, \ldots,$$

where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration k.

In other words, rather than computing the full subgradient, only the subgradient $\boldsymbol{g_{i_k}}$ corresponding to one of the functions f_{i_k} is used at each iteration.

We can follow two methods for choosing which $\boldsymbol{g_i}$ to use, that is, two rules for choosing index i_k at iteration k:

(i) *Cyclic rule:* choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$,

(ii) *Randomized rule:* choose $i_k \in \{1, \ldots, m\}$ uniformly at random.

Computationally, m stochastic steps approximately correspond to one classic step; but furthermore, using the stochastic method we do not need to have all data points in memory. So naturally, computing SSM is faster than computing the classic one. But we still need to see this new method works, that is, that the stochastic method converges to the solution of the problem.

In fact, a stochastic step performs quite similar to a classic step, the difference on iteration k is

$$\epsilon_{i_k} = \partial \sum_{i=1}^{m} f_i - \partial f_{i_k}.$$

Roughly speaking, different errors compensate each other and, on average, the accumulated error is insignificant.

The expected value of the subgradient on the stochastic case will be almost the real subgradient. Then, computing a stochastic step gives not a big difference from the real value, the one obtained by applying a classic step, and moreover it is much faster.

In order to obtain a good performance, the behaviour of SSM also needs to be analyzed so that the elements defined in Subsections 3.3.1 - 3.3.3 can be tuned.

3.4.1 Convergence results

We will state very basic convergence results for the stochastic method, considering step sizes that are square summable but non summable. See (Boyd et al., 2018) for the proofs.

Assume there is an $x^\star$ that minimizes f, and a G for which $\mathbb{E}\|\boldsymbol{g}^{(k)}\|_2^2 \leq G^2$ for all k. We also assume that R satisfies $\mathbb{E}\|x^{(1)} - x^\star\|_2^2 \leq R^2$.

Then,

$$\mathbb{E} f_{best}^{(k)} \longrightarrow f^\star$$

as $k \to \infty$, i.e., we have convergence in expectation. We also have convergence in probability: for any $\epsilon > 0$,

$$\lim_{k\to\infty} \textbf{Prob}\left(f_{best}^{(k)} \geq f^\star + \epsilon\right) = 0.$$

More sophisticated methods can be used to show almost sure convergence.

3.5 Application to MRCs

The innovation of this work is to apply these subgradient methods to solve the optimization problems described in (2.2) and (2.5). This is the first time such algorithms are used for this specific task.

3.5.1 Binary case

In order to start from the simplest case, let us consider the binary case where $\mathcal{Y} = \{1, 2\}$. Then, the vector we want to learn at the minimization task is of the form $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_{2d}\}^{\mathrm{T}}$. In order to apply the subgradient method, we need to calculate a subgradient of the objective function. In this case,

$$f(\boldsymbol{\mu}) = 1 - \boldsymbol{\tau}^{\mathrm{T}}\boldsymbol{\mu} + \boldsymbol{\lambda}^{\mathrm{T}}|\boldsymbol{\mu}| + \frac{1}{n}\sum_{i=1}^{n}\varphi(\boldsymbol{\mu}, x_i), \tag{3.10}$$

with

$$\varphi(\boldsymbol{\mu}, x) = \max\left\{[x^{\mathrm{T}},\ \mathbf{0}^{\mathrm{T}}]\ \boldsymbol{\mu} - 1,\quad [\mathbf{0}^{\mathrm{T}},\ x^{\mathrm{T}}]\ \boldsymbol{\mu} - 1,\quad \frac{[x^{\mathrm{T}},\ x^{\mathrm{T}}]\ \boldsymbol{\mu} - 1}{2}\right\}, \tag{3.11}$$

where $\mathbf{0}$ is the null vector in $\mathbb{R}^d$.

Recalling the rule of sum of subdifferentials (ii) described in Section 3.2.2, it is enough to compute a subgradient of each addend in (3.10) in order to obtain a subgradient of f, because every addend is a convex function itself. So let us consider the first three addend as f_i functions on μ, for $i = 1, 2, 3$.

Since $f_1(\boldsymbol{\mu}) = 1$ is differentiable, we have that

$$\partial f_1(\boldsymbol{\mu}) = \{\nabla 1\} = \{0\}.$$

The function $f_2(\boldsymbol{\mu}) = -\boldsymbol{\tau}^{\mathrm{T}}\boldsymbol{\mu}$ is also differentiable. Therefore,

$$\partial f_2(\boldsymbol{\mu}) = \{\nabla\left(-\boldsymbol{\tau}^{\mathrm{T}}\boldsymbol{\mu}\right)\} = \{-\boldsymbol{\tau}^{\mathrm{T}}\}.$$

However, $f_3(\boldsymbol{\mu}) = \boldsymbol{\lambda}^{\mathrm{T}}|\boldsymbol{\mu}|$ is not differentiable. Then, since $\boldsymbol{\lambda} \geq 0$, applying the nonnegative scaling rule (i) from Section 3.2.2, we just need to compute $\partial|\boldsymbol{\mu}|$.

According to Example 3.2.1, we define $\boldsymbol{g} = (g_1, \ldots, g_{2d}) \in \partial|\boldsymbol{\mu}|$ as

$$g_i = \operatorname{sign}(\mu_i), \quad i = 1, 2 \ldots, 2d.$$

The most challenging part might be the computation of the subgradient of the maximum given in (3.11). Following rule (iii) for the pointwise maximum, we have to obtain a subgradient of the function achieving the maximum at x.

If the maximum is achieved either on the first or second argument of $\varphi(\boldsymbol{\mu}, x)$, since they are differentiable functions, we have

$$\partial\varphi(\boldsymbol{\mu}, x) = \{\Phi(x, j)\},$$

where $j = 1$ if the maximum is achieved on the first argument, and $j = 2$ if it is achieved on the second.

Otherwise, if $\varphi(\boldsymbol{\mu}, x) = ([x^{\mathrm{T}}, x^{\mathrm{T}}]\boldsymbol{\mu} - 1)/2$, it is still differentiable and in this case the subdifferential is

$$\partial\varphi(\boldsymbol{\mu}, x) = \{([x^{\mathrm{T}}, x^{\mathrm{T}}]/2)\}.$$

So in order to obtain a subgradient of f, we just need to add the subgradients obtained in the previous steps. Then, the CSM is run as described in Section 3.3 with the obtained subgradient.

Note that for this method we need to compute the average of all the subgradients $\boldsymbol{g_i} \in \partial\varphi(\boldsymbol{\mu}, x_i)$ for $i = 1, 2 \ldots, n$, whereas for the SSM just one i is chosen at each step, following one of the rules described in Section 3.4.

3.5.2 Generalization to the multiclass case

The previous computations can easily be generalized to the multiclass case. Again, the main problem is to compute the subgradient of the maximum defined in (2.6), i.e.,

$$\varphi(\boldsymbol{\mu}, x) = \max_{\mathcal{C} \subseteq \mathcal{Y}} \left(\frac{\sum_{y \in \mathcal{C}} \Phi(x, y)^{\mathrm{T}} \boldsymbol{\mu} - 1}{|\mathcal{C}|} \right).$$

The computational cost of the maximum is very high, since we have to compute the maximum among every subset of $\mathcal{Y}$. For a fixed x, it is enough to compare the values given by

$$\sum_{y \in \mathcal{C}} \Phi(x, y)^{\mathrm{T}} \boldsymbol{\mu} \tag{3.12}$$

for each subset $\mathcal{C}$.

Then, we have to compare $2^{|\mathcal{Y}|} - 1$ values, the number of nonempty subsets of $\mathcal{Y}$. But, in fact, note that we can refuse most of the subsets. If we obtain the greatest value among all the subsets of the same cardinality, we just need to compare those greatest values in order to obtain the maximum. In this case, we reduce the comparison of $2^{|\mathcal{Y}|} - 1$ elements to $|\mathcal{Y}|$.

Let us compute just the value corresponding to subsets with one element, and order them. For simplicity, let us suppose that the order is given by just ordering the labels. That is,

$$\Phi(x, 1)^{\mathrm{T}} \boldsymbol{\mu} \geq \ldots \geq \Phi(x, |\mathcal{Y}|)^{\mathrm{T}} \boldsymbol{\mu}. \tag{3.13}$$

It is evident that the sum of the highest two values will be greater than any other sum of two different values, i.e.,

$$\Phi(x,1)^{\mathrm{T}}\boldsymbol{\mu} + \Phi(x,2)^{\mathrm{T}}\boldsymbol{\mu} \geq \Phi(x,i)^{\mathrm{T}}\boldsymbol{\mu} + \Phi(x,j)^{\mathrm{T}}\boldsymbol{\mu}$$

for every $i \neq 1, j \neq 1, 2$.

So for the value corresponding to subsets of two elements, we just take into account the two greatest ones given in (3.13) and discard the others, that is, discard every subset of two elements $\mathcal{C} \neq \{1,2\}$.

The same reasoning follows for every possible cardinality of $|\mathcal{C}|$. For a generic case where $|\mathcal{C}| = s$, we will take the greatest s values in (3.13) for the sum in (3.12), and discard the rest.

Then, the maximum we have to compute is reduced to

$$\max\left\{\Phi(x,1)^{\mathrm{T}}\boldsymbol{\mu} - 1, \frac{\Phi(x,1)^{\mathrm{T}}\boldsymbol{\mu} + \Phi(x,2)^{\mathrm{T}}\boldsymbol{\mu} - 1}{2}, \right.$$
$$\left. \ldots, \frac{\Phi(x,1)^{\mathrm{T}}\boldsymbol{\mu} + \ldots + \Phi(x,|\mathcal{Y}|)^{\mathrm{T}}\boldsymbol{\mu} - 1}{|\mathcal{Y}|}\right\}.$$

At this point, the subgradient is computed as described in the binary case, by finding at which argument the maximum is achieved and calculating its subgradient.

3.5.3 Constrained problem

The minimization problem given in (2.2)-(2.3) does not involve the calculus of the average among instances, so SSM cannot be applied to it. Instead, in this case, the maximum has to be computed among all the instances. This takes even more time, and the CSM will not be a very efficient method. Instead, we can convert this problem to a constrained optimization problem, and use the machinery introduced in Section 3.3.5. For that, we use a model transformation technique that deletes the maximum.

We define the following problem by introducing a new variable $\nu \in \mathbb{R}$,

$$\begin{aligned} \min_{\boldsymbol{\mu},\nu} \quad & 1 \; - \boldsymbol{\tau}^{\mathrm{T}}\boldsymbol{\mu} + \boldsymbol{\lambda}^{\mathrm{T}}|\boldsymbol{\mu}| + \nu \\ \text{s.t.} \quad & \sum_{y\in\mathcal{C}} \frac{\Phi(x_i,y)^{\mathrm{T}}\boldsymbol{\mu} - 1}{|\mathcal{C}|} \leq \nu, \quad \forall x_i, \mathcal{C} \subseteq \mathcal{Y}. \end{aligned} \tag{3.14}$$

Reorganizing it in order to have the same structure as in (3.9), we have the

following objective and constraint functions:

$$f_0(\boldsymbol{\mu}, \nu) = 1 \ - \boldsymbol{\tau}^{\mathrm{T}}\boldsymbol{\mu} + \boldsymbol{\lambda}^{\mathrm{T}}|\boldsymbol{\mu}| + \nu, \tag{3.15}$$

$$f_j(\boldsymbol{\mu}, \nu) = \sum_{y \in \mathcal{C}} \frac{\Phi(x_i, y)^{\mathrm{T}}\boldsymbol{\mu} - 1}{|\mathcal{C}|} - \nu, \tag{3.16}$$

where there are as many f_j as constraints, i.e., one j for each instance x_i and subset $\mathcal{C}$.

At this point, the algorithm defined in Section 3.3.5 is run for these functions. In this case, since there are two variables, we update each one using the corresponding subgradient at each step. Every f_j behaves with respect to $\boldsymbol{\mu}$ in the same way as described in previous sections. So the calculus of a subgradient is already explained.

On the other hand, every function is differentiable with respect to ν. So in order to obtain its subdifferential, it is enough to derive wrt ν the corresponding function at each iteration, i.e.,

$$\boldsymbol{g_\nu}^{(k)} = \begin{cases} 1, & \text{if } x^{(k)} \text{ feasible,} \\ \\ -1, & \text{if } x^{(k)} \text{ infeasible.} \end{cases}$$

The main problem is to check whether the current point is feasible or not, i.e., to check whether every constraint is satisfied. This can be a hard task since there might be too many constraints but, in fact, it results quite simple: it is enough to calculate the maximum among all classes for each instance, and check if it is not positive. If these values are non-positive, so are the rest, and every constraint is satisfied. On the contrary, if one of those values is positive, the point is infeasible and it already determines a violated constraint, so we can use that constraint function to compute the subgradient at that step.

By Section 3.5.2, we already know how to calculate this maximum efficiently, so we can easily compute this method. In this way, the problem is restricted to check just n number of constraints.

Chapter 4

Numerical results

For the experiments in Chapter 4, we have considered different binary and multiclass classification datasets, available in the publicly available UCI repository (Dua and Graff, 2017). Each dataset mentioned in the tables and captions includes a link to its corresponding page on the UCI website. These experiments were run using MATLAB and they were conducted on a machine with Intel Core i5 CPU, RAM 8 GB memory. The main goal of this experimental part is to see how our algorithm works, comparing different situations in order to choose the right parameters introduced in Section 3.3. Various aspects are taken into account. Mainly, execution time and approximation to the real solution, where the aim is to obtain a good trade-off so as to achieve both situations. For these purposes, we test different step sizes, number of iterations, variable evaluation, etc. and we train the method using various datasets, in order to assess its performance.

Recall that subgradient methods are not exact methods and can just provide an approximation of the solution. So, the performance criterion has been whether the obtained result is *close enough* to the solution of the minimization problem. In order to determine the allowed error factor, we take into account that the values we are optimizing refer to classification error probabilities, and we will consider errors around 10^{-2} as acceptable.

Then, we have to compare the approximation solutions obtained by the algorithms with the real one. To do so, the real solution is needed, but obtaining it is a very hard task. Therefore, achieving the real solution is not an option then, but we still need a value so as to evaluate the completion of the algorithm. So we use CVX, a MATLAB-based software for disciplined convex programming (Grant and Boyd, 2014), in order to obtain a very close approximation, a value we will use as a reference to determine an effective method.

It is natural to ask why we are developing a new algorithm if CVX can already provide a reliable solution. It does for relatively small datasets, but it will take very long to perform with large ones. That is why we use these tractable datasets to train our method and guarantee a good performance even with larger data.

4.1 Results for MRCs with 0-1 loss and fixed marginals

In this section we analyze the results obtained for the optimization problem given in the learning stage of MRCs with 0-1 loss and fixed marginals, the one described in equations (2.5)-(2.6).

We begin with a small simple dataset (Haberman, binary with 306 instances of 3 variables) and see how the algorithm works. A small step size may imply a slower convergence but it also provides a more accurate result. The idea is to obtain a method which performs fast enough to allow the use of a precise step size. We present in Table 4.1 the results obtained by each method,

Table 4.1. Results obtained with both Classic and Stochastic methods for a binary dataset with 306 samples.

Dataset	**CVX**				**CSM**		**SSM**	
Haberman	t	Solution	n	α	t	Solution	t	Solution
	4.0675	**0.4722**	10^3	10^{-1}	1.8287	0.4726	0.2163	0.6772
				10^{-2}	1.8499	0.4727	0.2014	0.4802
				10^{-3}	1.8848	0.4855	0.2176	0.4857
				$0.02/i$	1.8132	0.4833	0.1690	0.5034
				$0.02/\sqrt{i}$	1.3327	0.4833	0.1334	0.4937
			10^4	10^{-1}	15.9164	0.4725	0.2754	0.7923
				10^{-2}	16.0396	0.4724	0.2713	0.4785
				10^{-3}	15.9771	0.4727	0.2725	0.4729
				$0.02/i$	15.8553	0.4732	0.2544	0.5022
				$0.02/\sqrt{i}$	11.6459	0.4732	0.2143	0.4735
			10^5	10^{-1}	>4min	0.4726	0.7779	0.5982
				10^{-2}	-	-	0.7438	0.4782
				10^{-3}	-	-	0.7379	0.4729
				$0.02/i$	>5min	0.4726	0.8748	0.5012
				$0.02/\sqrt{i}$	>3min	0.4726	0.5394	0.4726

CVX, the CSM and the SSM, for Haberman dataset, for different number of

iterations n and step sizes α, including the adaptive step sizes depending on the number of iterations i. For each method the results are represented in two columns corresponding to solution value (Solution) and execution time (t). We can observe that the Stochastic method is significantly faster than the Classic one. However, CSM gives very accurate results while SSM needs more iterations in order to obtain such good approximations. For instance, the results of SSM for 10^3 iterations are not very promising, but the speed of this method allows to make more iterations so that we can obtain a better approximation, as it happens with 10^5 iterations.

Moreover, even if the results given by the Classic method are acceptable, the increment on the number of iterations takes already too long in this binary little dataset, so we can assume it will not be possible to compute it for large ones. That is why we are not showing the results for CSM with $\alpha = 10^{-2}$ and $\alpha = 10^{-3}$ for 10^5 iterations; these processes will take pretty long, as it happens with $\alpha = 10^{-1}$, and we already obtained a very good approximation, also with $\alpha = 10^{-1}$.

In order to give an example where CVX takes longer, we present the following results in Table 4.2 for Redwine dataset (1599 instances of 11 variables and 6 classes). Not only larger datasets, according to the number of instances, make this process slower, but also datasets of multiclass scenario, as this Redwine one; the time increases as the number of classes does. A few results are shown in the table organized as previously described: we consider some number of iterations n and a specific step size α for each n, and represent the results obtained by the CSM, the SSM and CVX (solution value (Solution) and execution time (t)) by columns. We can see that the CVX takes more than a minute, while the implementation of the stochastic method is very fast. Nevertheless, the classic one takes much time already for 10^4 iterations, so computing the results for 10^5 is not possible.

Table 4.2. Successful results obtained with both Classic and Stochastic methods for Redwine dataset with 1599 samples and 6 labels.

Dataset	**CVX**				**CSM**		**SSM**	
Redwine	t	Solution	n	α	t	Solution	t	Solution
	>1min	**0.6705**	10^3	10^{-1}	13.4132	0.6706	0.2194	1.2818
			10^4	10^{-3}	>4min	0.6740	0.3185	0.6756
			10^5	10^{-3}	-	-	1.0823	0.6728

4.1.1 Importance of the step size

The choice of the step size does not only determine the convergence speed, but the convergence in general, as proved in Section 3.3.4. Figure 4.1 shows the importance of a proper step size in order to achieve a good performance on the method. Figure 4.1 is obtained using Credit dataset with 690 classes of 15 variables and 2 classes, and shows the objective function values given by the SSM, on the y axis, for different step sizes during 10^4 iterations, on the x axis. We can see the algorithm for the step size $\alpha = 0.001$, red line, gets very close to the value given by CVX for this scenario, yellow line. While for $\alpha = 0.01$, blue line, the method is actually not working, it keeps fluctuating and even the minimum value is not an accurate approximation to the solution. Apart from the constant step sizes, we try different ones

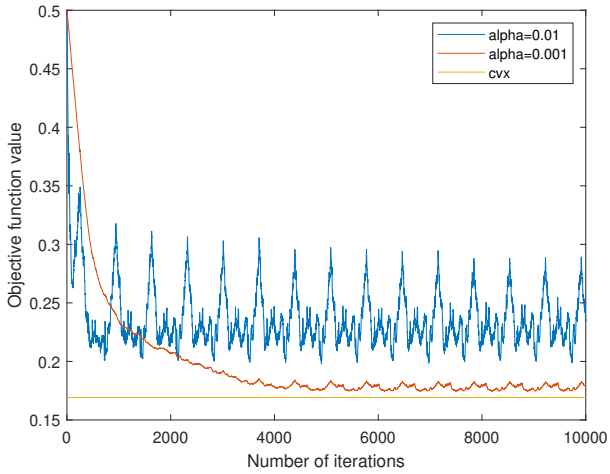
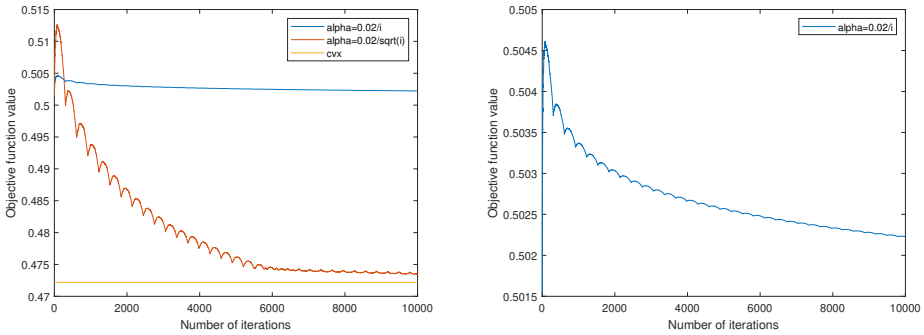


Figure 4.1: Behavior of the SSM for $n = 10^4$ number of iterations with Credit dataset, binary with 690 samples. The blue line shows the method for $\alpha = 0.01$, the red one for $\alpha = 0.001$, and the yellow one is the reference value obtained with CVX, the one to be approximated.

that depend on the current number of iteration i: $\alpha = \mathrm{c}/i$ and $\alpha = \mathrm{c}/\sqrt{i}$, where c is a nonzero constant. The second one is an attempt to correct the first as shown in Figure 4.2. The figure shows the objective function values obtained with the stochastic method for previously introduced Haberman dataset. The graphics are computed using 10^4 iterations and $\alpha = 0.02/i$ and $\alpha = 0.02/\sqrt{i}$ step sizes. The blue line corresponding to $\alpha = 0.02/i$, increases the objective value considerably during the first steps and converges very slowly; this makes the algorithm harder to achieve a lower final value. On the other hand, the square root, $\alpha = 0.02/\sqrt{i}$, red line, makes the step size smaller as the number of iterations increases, so that the convergence is

faster at the beginning and enables the achievement of a lower value. By Figure 4.2b we see that the behavior of the blue line, values for $\alpha = 0.02/i$, is actually similar to the red line, values corresponding to $\alpha = 0.02/\sqrt{i}$, on a smaller scale so that takes much longer to decrease. The constant we use



(a) (b)

Figure 4.2: Comparison of the SSM for Haberman dataset with 10^4 iterations. In Figure 4.2a we have in blue the values for $\alpha = 0.02/i$, in red for $\alpha = 0.02/\sqrt{i}$, and in yellow the value obtained with CVX. In Figure 4.2b we can see clearer the behavior of the method with $\alpha = 0.02/i$. That is, both blue lines represent the same function.

is 0.2, motivated by analyzing the method behavior with $\alpha = 1/i$. That is, for this step size the method returns increasing values. After different trials, 0.2 is the best value reducing this increment.

4.1.2 Performance of different evaluation techniques

We already introduced in Section 3.3.1 other evaluation techniques that allow to achieve a better solution, in exchange of, probably, a higher computational cost.

Firstly, the first approach, described as keeping track of the best point, needs the calculation of the objective function value at each step. This is a very costly task, since it needs to store every instance and compute the respective maximum at each iteration, in order to obtain the objective value and, then, compare to the previous one. On the contrary, the simple approach used until now, makes this evaluation just on the last iteration, in order to return the solution value.

It is obvious the success of this new approach, since, in this case, we will obtain the best value achieved during the whole method. Using this result, this type of procedure can be useful in the cases where the algorithm

fluctuates and it is not very stable as shown in Figure 4.3. The objective function values corresponding to the stochastic method for Glass dataset (214 instances of 9 variables with 6 classes) are shown, using 10^3 instances and $\alpha = 10^{-2}$ step size. The blue line represents the SSM with the usual points given by the algorithm, while the red one is using the best point approach to evaluate the function. This removes the fluctuations and keeps the best value obtained along the computation of the method. We see how in general the best point method keeps closer to the CVX value, the one we would like to achieve. However, this method will take much longer than

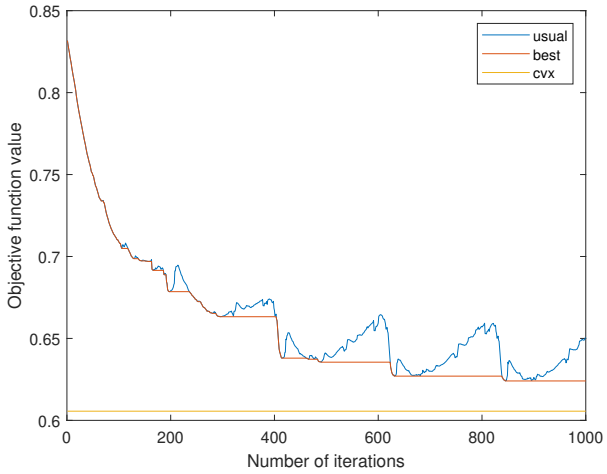


Figure 4.3: Objective function values for the SSM with Glass dataset, multiclass of 6 labes with 214 instances, $n = 10^3$ number of iterations and $\alpha = 10^{-2}$. In blue, the performance of the algorithm as usual; in red, performance using the best point approach; in yellow, the CVX value.

teh usual one. We show in Table 4.3 the results for the same dataset as the one used in Figure 4.3, Glass. Mostly, the difference of execution time between the evaluation approaches is remarkable. Results corresponding to the value of the solution and the execution time t are shown by columns, for the SSM with each evaluation technique (the usual and the best point method). We can see the fast performance of the usual method comparing to the best evaluation. For the purpose of decreasing this computational cost but, at the same time, trying to keep an improvement on the solution value, the mean approach was described. This method computes the mean of the points given by the algorithm at each iteration, so no function evaluation is needed.

Moreover, for the calculus of the mean, we just need to compute a multiplication and a sum. That is, we compute the mean of points $x^{(1)}, \ldots, x^{(i)}$

Table 4.3. Results for SSM using the usual evaluation method, and the best point approach. Glass dataset with 214 samples and 6 labels.

Dataset	**CVX**				**SSM-usual**		**SSM-best**	
Glass	t	Solution	α	n	t	Solution	t	Solution
	4.9693	**0.6056**	10^{-2}	10^3	0.1608	0.6504	2.0184	0.6241
				10^4	0.3612	0.6462	18.5665	0.6172
				10^5	1.0605	0.6194	>3min	0.6172

recursively by following

$$\text{mean}^{(i)} = \frac{i-1}{i}\text{mean}^{(i-1)} + \frac{1}{i}x^{(i)}.$$

Figure 4.4 and Table 4.4 illustrate this new process and the comparison of times. In Figure 4.4 we have the comparison of the SSM for Glass dataset reduced to 4 classes (192 samples of 9 variables) using the usual and the mean evaluation technique. Results obtained for $n = 4000$ number of iterations and $\alpha = 10^{-2}$ are shown in blue for the usual approach, and in red for the mean evaluation values. The mean performs a stable behavior getting close to the CVX value, in yellow. Table 4.4 shows numerical re-

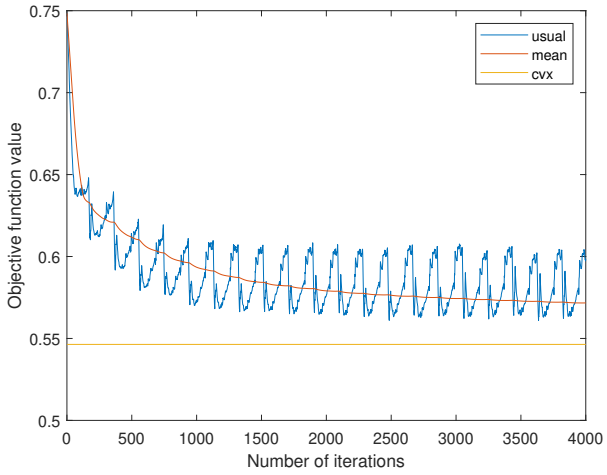


Figure 4.4: Behavior of the SSM for $n = 4000$ number of iterations and $\alpha = 10^{-2}$ with the reduction of Glass dataset to 4 classes, multiclass of 4 labes with 192 samples. The blue line shows the method for the usual approach, the red one for the mean and the yellow value obtained with CVX.

sults, solution values and computation time, for the stochastic method in

the previous scenario (Glass dataset reduction to 4 classes, $n = 4000$ iterations and $\alpha = 10^{-2}$) with each evaluation approach introduced so far: usual one, the best point method, and the mean evaluation; together with the CVX results, too. We can appreciate how the usual and the mean approach

Table 4.4. Results obtained with SSM for the reduction of Glass dataset to 4 classes with $\alpha = 10^{-2}$ and $n = 4000$.

Dataset	**CVX**		**SSM-usual**		**SSM-mean**		**SSM-best**	
Glass4	t	Solution	t	Solution	t	Solution	t	Solution
	5.5706	**0.5463**	0.2186	0.5984	0.4585	0.5717	5.1871	0.5609

are quite similar on time, unlike the best procedure, that takes considerably longer. Furthermore, the algorithm performs better using the mean than using the usual points. In this case, the exact number of iterations is not such significant as it can be in the usual approach, i.e., in the latter method, the values corresponding to consecutive iterations might vary significantly, while using the mean avoids this issue.

4.2 Results for MRCs with 0-1 loss

The main difference of the problem used is this section, the one described in equations (2.2)-(2.3), compared to the one used in the previous section, is the computation of the maximum among all instances, instead of calculating the average.

We will analyze the accuracy of the subgradient method for the constrained problem given in 3.14, using different datasets and steps sizes. The first idea is to replicate the experiments done in the previous section, in order to compare the methods.

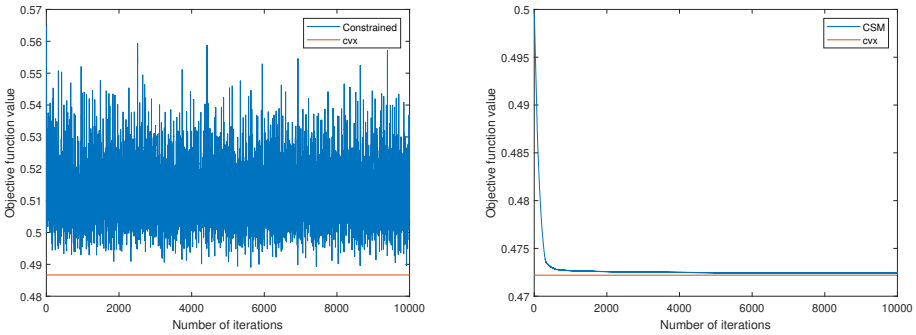
4.2.1 Importance of the step size

A few examples given in Table 4.5 are enough to illustrate that big constant step sizes, such as 10^{-1} or 10^{-2}, are not working in this problem as in the prior one. The table shows the results obtained by the subgradient method for the constrained problem using Haberman dataset (binary with 306 instances of 3 variables), representing the solution value and execution time by columns for different number of iterations $n = 10^3, 10^4, 10^5$ and step sizes $\alpha = 10^{-1}, 10^{-2}, 10^{-3}$. Note that the first two step sizes do not provide good results for any number of iterations. In the latter problem, the

Table 4.5. Results from subgradient method for constrained problem (SM) with Haberman dataset.

Dataset	**CVX**				**SM**	
Haberman	t	Solution	n	α	t	Solution
	5.3952	**0.4866**	10^3	10^{-1}	0.9650	0.8540
				10^{-2}	0.9805	0.5209
				10^{-3}	0.9462	0.4936
			10^4	10^{-1}	8.5978	0.8819
				10^{-2}	8.6323	0.5172
				10^{-3}	8.7319	0.4890
			10^5	10^{-1}	>2min	0.6205
				10^{-2}	>2min	0.5026
				10^{-3}	>3min	0.4897

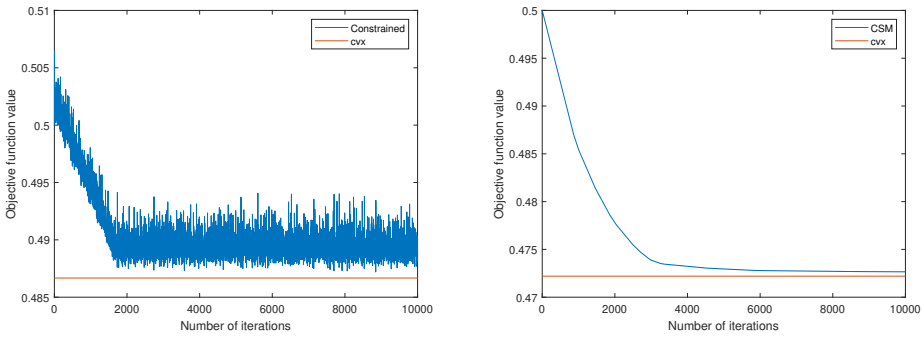
steps 10^{-1} or 10^{-2} where pretty accurate, and, in fact, Figure 4.5 shows the comparison of both situations, CSM and constrained method, for the same scenario, same step size and number of iterations. In the figure the algorithm is computed for Haberman dataset, using 10^4 iterations and $\alpha = 10^{-2}$. The blue lines show the objective function values for the constrained problem algorithm with already mentioned parameters, and the red line represents the CVX value. Moreover, Figure 4.5a shows the extreme fluctuation of the constrained method for Haberman dataset, the new algorithm is not actually converging. Whereas Figure 4.5b shows how the CSM works under the same circumstances. On the other hand, going back to Table 4.5, the step size $\alpha =$



(a) (b)

Figure 4.5: Comparison of the CSM and constrained method for Haberman dataset with 10^4 iterations and $\alpha = 10^{-2}$. In Figure 4.5a we have the behavior of the constrained method in blue and its CVX value in red. Whereas in Figure 4.5b we recall the CSM problem for the same scenario in blue, and its corresponding CVX value in red.

10^{-3} seems to perform pretty well, but it needs at least 10^4 iterations. We can see in Figure 4.6a that the constrained method tends to converge, even if it fluctuates, and 4.6b indicates, one more time, the behavior of the CSM under the same circumstances. We are using Haberman dataset, with 10^{-3} step size and 10^4 number of iterations to show these results. Other step sizes



(a) (b)

Figure 4.6: Both figures show in blue the objective function value obtained by the respective algorithms, and in red the CVX value for each case for Haberman dataset, 10^{-3} step size and 10^4 number of iterations. 4.6a uses the subgradient method for constrained problems and 4.6b the CSM.

have been also used for this problem, such as $\alpha = 0.3/\sqrt{i}$ or $\alpha = 0.1/(0.3 + \sqrt{i})$. Table 4.6 shows some results obtained, including these adaptive step sizes, together with results obtained for $\alpha = 10^{-3}$ for different datasets and number of iterations. The presented dataset are the following: Haberman (binary with 306 instances of 3 variables), Credit (binary with 690 intances of 15 variables), Glass3 which is the reduction of Glass dataset to 3 classes (163 intances of 9 variables and 3 classes), Glass (214 instances of 9 variables and 6 classes). These results are the best ones obtained after different trials with various step sizes in order to determine the constants. For instance, we compute the algorithm for Haberman dataset with $0.2/\sqrt{i}$, the step size used in the previous problem; but we noticed a different constant is working better for this problem: $\alpha = 0.3/\sqrt{i}$. We can observe that the values obtained are better for smaller datasets, as it is easier to perform in a more accurate way. Moreover, computing the algorithm for larger datasets, both for more instances and more classes, takes longer as already analyzed.

4.2.2 Performance of different evaluation techniques

Note that the subgradient method for the constrained problem fluctuates more than the classic or the stochastic one in the previous section, as see in

Table 4.6. Some results from subgradient method for constrained problem with different datasets and parameters.

Dataset	Instances	**CVX**				**SM**	
		t	Solution	n	α	t	Solution
Haberman	306	5.3952	**0.4866**	10^3	$0.3/i$	1.3575	**0.4871**
					$0.3/\sqrt{i}$	1.3092	0.4886
Credit	690	7.4179	**0.1721**	10^4	10^{-3}	22.3866	**0.1778**
					$0.3/\sqrt{i}$	39.7145	0.1911
					$0.1/(0.3+\sqrt{i})$	22.0585	0.1804
				10^5	$0.3/\sqrt{i}$	>7min	**0.1734**
					$0.1/(0.3+\sqrt{i})$	>4min	**0.1753**
Glass3	163	2.4887	**0.6270**	10^4	10^{-3}	4.4659	**0.6293**
					$0.3/i$	4.4622	0.6307
					$0.3/\sqrt{i}$	4.6131	0.6301
				10^5	$0.3/i$	>1min	0.6288
					$0.3/\sqrt{i}$	>1min	**0.6273**
Glass	214	5.5299	**0.6598**	10^4	10^{-3}	13.6199	0.7143
					$0.3/\sqrt{i}$	14.8806	0.7245
					$0.3/(0.1+\sqrt{i})$	11.5370	0.6830
				10^5	10^{-3}	>3min	0.6710
					$0.3/\sqrt{i}$	>3min	0.6957
					$0.3/(0.1+\sqrt{i})$	>2min	**0.6631**

the figures presented during this section 4.2, Figures 4.5 and 4.6. Given the fluctuations, it is a good idea to apply the best point evaluation approach, or either the mean approach described in Section 3.3.1. We can see in Figure 4.7 the behavior of these different methods, and in Table 4.7 their corresponding values. Figure 4.7 represents the objective function values for the subgradient method using different evaluation techniques: in blue the usual results, in red the mean evaluation, in yellow the best point approach, and in purple the corresponding CVX value. Here Haberman dataset is used, with 10^3 number of iterations and $\alpha = 10^{-2}$ step size. Table 4.7 shows the

Table 4.7. Results obtained from different evaluation methods for the subgradient constrained method, for 10^3 number of iterations and 10^{-2} step size. Best point approach and mean evaluation method are compared to the usual evaluation.

Dataset	**SM-usual**		**SM-mean**		**SM-best**	
Haberman	t	Solution	t	Solution	t	Solution
	1.6049	0.5209	1.3904	0.5020	2.7900	0.4930

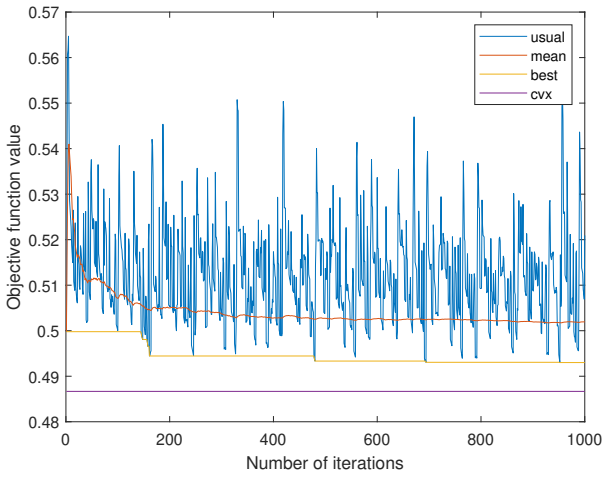


Figure 4.7: Behavior of the subgradient method for constrained problem using 10^3 number of iterations and $\alpha = 10^{-2}$ step size with Haberman dataset. In blue, the usual evaluation approach; in red, the mean method; in yellow, the best point evaluation; and in purple, CVX value.

corresponding numerical results from the computation of the subgradient method for Haberman dataset with 10^3 number of iterations and 10^{-2} step size. The presented three evaluation techniques are represented by columns: usual results (SM-usual), mean evaluation (SM-mean) and the best point approach (SM-best). We can observe that these alternative approaches, the mean and the best point one, perform slightly better than the usual one. The best point approach takes longer than the other techniques, since it has to compute the objective function value for each iteration, as explained in Section 3.3.1.

4.3 Discussion of Results

During this chapter different experiments and computations have been done with the aim of determining a successful method. First we analyzed the fixed marginal case where CSM performs very well. The approximations obtained by the classic method are very accurate, but the algorithm already takes some time for "bigger" datasets as Credit, so it could not be possible to compute the CSM for large datasets, since it will take too much time. In general, we can deduce that this method will just be useful for simple datasets (binary with around 1000 instances or less, for example).

The stochastic method seems to provide less accurate results, but it is faster

so more iterations can be computed in order to improve the approximation and obtain a good solution. In general, small constant step sizes (10^{-3}) provide accurate solutions, but this also makes the convergence slower. The best step size assuring this trade-off is $\alpha = 10^{-3}$. Adaptive step sizes also have pretty successful results, for instance, we propose $\alpha = 0.02/\sqrt{i}$ after several computations.

According to evaluation approaches, the best point method obtains the best value but it takes very long, so this evaluation is not actually computable. On the other hand, the mean approach is used because of its computation speed and can usually reduce the result value, so it is a good variant in order to make the algorithm stable and obtain a good solution on an effective way.

During the analysis of the problem for MRCs with 0-1 loss without fixed marginals, we used the subgradient method for constrained problem where constant step sizes are not working as well as in the previous problem. So the step size $\alpha = 10^{-3}$ could be used, and with at least 10^4 iterations. This does not take very long yet, so it can be computed.

Much more fluctuation as in the classic one can also be noted, so in order to reduce them alternative evaluation techniques have been tested: best point and mean approach. They perform as they do for the previous problem, the best point method takes too long so that it is not an option for future large datasets, while the mean evaluation is a good alternative usually providing a better solution than the usual algorithm.

In general, we can see that the constrained approach is not as accurate as the classic method, and do not perform as well as the classic does. Furthermore, it does not perform as fast as the stochastic one, so that the convergence will be slower.

Chapter 5

Conclusions and Future work

In this work several variants of the subgradient method have been proposed and executed, in order to analyze their behavior and specify the necessary parameters so as to obtain a successful algorithm. Firstly, the necessary background about Machine Learning and MRCs has been established in order to present the minimization problem which the research is focused on. Then, theoretical concepts have been presented and explained in Chapter 3, while in Chapter 4, we have computed several experiments to realize these issues, applying the corresponding methods to the two problems described in this study: the minimization problem arisen in the learning stage of MRCs with 0-1 loss (2.2)-(2.3), and the one including fixed marginals (2.5)-(2.6).

First, we analyzed the fixed marginal case applying the classical method, with the aim of evolving to the stochastic method. In this scenario, the CSM performs very well with different step sizes, but the algorithm may take very long for large datasets; so that in general, this method will just be useful for small ones.

Just by the theory, we can deduce that the stochastic method will not give such accurate results as the classic one: the CSM computes the exact subgradient of the objective function at each step by using the entire dataset, while the SSM uses just a subset of the data. This fact makes the algorithm faster to be computed, as already described, and that is exactly why we are interested in this method. Indeed, the results obtained during Chapter 4 support this conclusions.

The SSM needs more study to determine a good performance. In general, constant step sizes need to be pretty small in order to obtain an accurate solution, but this also makes the convergence slower. Adaptive step sizes also have pretty successful results. This method performs much faster than

the classic one, so that it enables to perform many iterations.

Different evaluation approaches can be used, too. As we already described, the best point method obtains the best value achieved during the method so it is a successful approach, but it takes very long. On the other hand, computing the mean of the points obtained by the algorithm, and evaluating this one in order to obtain the result, is used because of its computation speed: quite similar to the usual approach but performs in a more stable way than the usual method. Overall, the best point approach can only be used with small datasets. For larger ones, using the mean can usually reduce the result value.

Regarding the problem for MRCs with 0-1 loss, without fixed marginals, we used a constrained approach, that is, we converted the problem in an equivalent constrained one. For this situation, constant step sizes are not working as well as in the previous problem. This algorithm suffers from much more fluctuation than the Classic one, so this implies a slower convergence: the time is spent while the method fluctuates, and this does not allow to decrease the value. These fluctuations give a good situation to use the best point or mean approach, as mentioned in the previous method.

Moreover, it is notorious that the approximations for this problem are not as accurate as the results obtained with the CSM, nor as fast as the ones from SSM. These statements are sensible: on the one hand, the constrained method does not focus each step on the reduction of the objective function value, it has to make the value feasible, too; so it takes longer to perform a successful result. On the other hand, it is not computing a subgradient of the actual objective function we want to minimize, making the approach not as accurate as the classic, that actually performs an exact subgradient.

Studying the results, we can see the algorithm succeeds for small datasets as Haberman, and loses accuracy as the dataset is larger; either ut has many instances (Credit), or many classes (Glass). Furthermore, in these challenging cases, more iterations are needed in order to obtain better approximations. This is reasonable considering the fluctuations, that is, the method commits many steps that worsen the objective function value. So in order to achieve a low enough value, these bad steps need be corrected and many iterations are needed for this purpose.

As a final conclusion, we can mention that we obtained a pretty satisfying algorithm for the first case, fixed marginals, while for the second case the developed method is not such accurate in general, and this case may require an exhaustive study or the development of other methods.

In future work, several strategies could be explored to improve the perfor-

mance of the subgradient method for the constrained problem we presented. For instance, testing different step sizes could yield better results. Additionally, alternative approaches could be considered, such as developing new techniques to efficiently compute the required maximum. In this work the problem is handled by using a constrained conversion, the development of other approaches could also make a difference. By exploring these avenues, we aim to enhance the method's efficiency and effectiveness in solving the given problem.

Bibliography

Peter L. Bartlett, Michael I. Jordan, and Jon D. McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

Stephen Boyd. Subgradient methods. *Lecture notes of EE364b, Stanford University, Spring Quarter*, 2014.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Stephen Boyd, Almir Mutapcic, and John Duchi. Stochastic subgradient methods. *Lecture notes of EE364b, Stanford University, Spring Quarter*, 2018.

Stephen Boyd, Jhon Duchi, Mert Pilanci, and Lieven Vandenberghe. Subgradients. *Lecture notes of EE364b, Stanford University, Spring Quarter*, 2022.

Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Farzan Farnia and David Tse. A minimax approach to supervised learning. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems*, pages 4240 – 4248, 2016.

Rizal Fathony, Anqi Liu, Kaiser Asif, and Brian D. Ziebart. Adversarial multiclass classification: A risk minimization perspective. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems*, pages 559 – 567, 2016.

Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1, 2014. URL `https://cvxr.com/cvx`.

Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Fundamentals of convex analysis*. Springer Science & Business Media, 2004.

Santiago Mazuelas, Mauricio Romero, and Peter Grünwald. Minimax risk classifiers with 0-1 loss. *Journal of Machine Learning Research*, 24:1–48, 2023.

Mehryar Mohri. *Foundations of machine learning*. MIT press, 2018.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Vladimir N. Vapnik. *Statistical learning theory*. wiley New York, 1998.

Appendix A

Codes for CVX

A.1 Code for CVX for MRCs with 0-1 loss and fixed marginals problem

```
%LOAD DATASET
load()

%CALCULUS OF NECESSARY PARAMETERS
n=size(X,1);
d=size(X,2);
c=length(unique(Y));

I=eye(c);
kron_matrix=zeros(c*d,n);
for i=1:n
    kron_matrix(:,i)=kron(I(:,Y(i)),X(i,:)');
end
[standesv,tau]=std(kron_matrix,1,2);
lambda=standesv/sqrt(n);


%CVX PROBLEM FORMULATION
cvx_begin
    variable mu(c*d,1)
    minimize(1-tau'*mu+lambda'*abs(mu)+average_max(X
       ,mu,c))
cvx_end
```

A.1.1 average_max function

```
function mean_of_maxs=average_max(X,mu,c)
    n=size(X,1);
    sum=0;
    for i=1:n
        sum=sum+phi_general(X(i,:)',mu,c);
    end
    mean_of_maxs=sum/n;
end
```

A.1.2 phi_general function

```
function max_of_class=phi_general(x,mu,c)
    d=length(x);

    M=zeros(2^c-1,c*d);
    denominators=zeros(2^c-1,1);
    aux=0;
    for i=1:c
       options=nchoosek(1:c,i);

       for j=1:size(options,1)
           v=options(j,:);
           M(aux+j,:)=encodings(v,x,c);
           denominators(aux+j)=1/i;
       end
       aux=aux+j;
    end

    results=(M*mu-1).*denominators;
    max_of_class=max(results);
end
```

A.1.3 encodings function

```
function f=encodings(v,x,c)
    d=length(x);
    I=eye(c);

    sums=zeros(1,c*d);
```

```
    for i=1:length(v)
        sums=sums+(kron(I(:,v(i)),x))';
    end
    f=sums;
end
```

A.2 Code for CVX for MRCs with 0-1 loss problem

```
%LOAD DATASET
load()

%CALCULUS OF NECESSARY PARAMETERS
n=size(X,1);
d=size(X,2);
c=length(unique(Y));

I=eye(c);
kron_matrix=zeros(c*d,n);
for i=1:n
    kron_matrix(:,i)=kron(I(:,Y(i)),X(i,:)');
end
[standesv,tau]=std(kron_matrix,1,2);
lambda=standesv/sqrt(n);


%CVX PROBLEM FORMULATION
cvx_begin
    variable mu(c*d,1)
    minimize(1-tau'*mu+lambda'*abs(mu)+maxmax(X,mu,c
       ))
cvx_end
```

A.2.1 maxmax function

```
function max_of_all=maxmax(X,mu,c)
    n=size(X,1);

    for i=1:n
        maxis(i)=phi_general(X(i,:)',mu,c);
```

```
    end
    max_of_all=max(maxis);
end
```

Note that `phi_general` and `encodings` functions are the same as the ones defined in sections A.1.2-A.1.3.

Appendix B

Codes for the subgradient methods for MRCs with 0-1 loss and fixed marginals problem

B.1 Classic Subgradient Method

```
%LOAD DATASET
load()

%CALCULUS OF NECESSARY PARAMETERS
n=size(X,1);
d=size(X,2);
c=length(unique(Y));

I=eye(c);
kron_matrix=zeros(c*d,n);
for i=1:n
    kron_matrix(:,i)=kron(I(:,Y(i)),X(i,:)');
end
[standesv,tau]=std(kron_matrix,1,2);
lambda=standesv/sqrt(n);


mu=zeros(c*d,1);
mu_aux=mu; %for averaging
```

```
f_best=Inf; %for best point approach

%CHOOSE CTE STEP SIZE
alpha=...;

%CHOOSE NUMBER OF ITERATIONS
iter=...;

%CALCULUS OF SUBGRADIENT
g1=-tau;

G3=zeros(c*d,n);
for i=1:iter
    %CHOOSE CHANGING STEP SIZE (if not constant)
    %alpha=...;

    g2=lambda.*sign(mu);

    for j=1:n
        [~,g3]=sg_max(mu,X(j,:)',c);
        G3(:,j)=g3;
    end
    g3=mean(G3,2);

    g=g1+g2+g3;

    %SUBGRADIENT METHOD STEP
    diff=-alpha*g;

    %NEW POINT
    mu=mu+diff; %usual point
    mu_aux=((i-1)/i)*mu_aux+(1/i)*mu; %average of
       all the points obtained so far

    %CALCULUS OF BEST POINT OBJECTIVE FUNCTION VALUE
    f_i=f(mu,X,tau,lambda,c);
    f_best=min([f_best f_i]);

end

f(mu,X,tau,lambda,c) %usual final objective function
    value
f(mu_aux,X,tau,lambda,c) %averaging technique final
   objective function value
```

```
f_best %best point approach final objective function
    value
```

B.1.1 sg_max function

```
function [value,subgradient]=sg_max(mu,x,c)
    d=size(x,1);

    v=zeros(c,1);
    I=eye(c);

    K=kron(I,x)';

    for i=1:c
        v(i)=x'*mu((i-1)*d+1:i*d);
    end

    [values,order]=sort(v,'descend');

    value=values(1)-1;
    subgradient=K(order(1),:);

    for i=2:c
        new=((i-1)*value+values(i))/i;
        if new>=value
            value=new;
            subgradient=((i-1)*subgradient+K(order(i
               ),:))/i;
        else
            break
        end
    end

subgradient=subgradient';
end
```

B.1.2 f function

```
function objective=f(mu,X,tau,lambda,c)
    objective=1-tau'*mu+lambda'*abs(mu)+phi(X,mu,c);
end
```

B.1.3 phi function

```
function f=phi(X,mu,c)
    n=size(X,1);

    sum=0;
    for i=1:n
        [maxi,~]=sg_max(mu,X(i,:)',c);
        sum=sum+maxi;
    end
    f=sum/n;
end
```

B.2 Stochastic Subgradient Method

```
%LOAD DATASET
load()

%CALCULUS OF NECESSARY PARAMETERS
n=size(X,1);
d=size(X,2);
c=length(unique(Y));

I=eye(c);
kron_matrix=zeros(c*d,n);
for i=1:n
    kron_matrix(:,i)=kron(I(:,Y(i)),X(i,:)');
end
[standesv,tau]=std(kron_matrix,1,2);
lambda=standesv/sqrt(n);


mu=zeros(c*d,1);
mu_aux=mu; %for averaging
f_best=Inf; %for best point approach

%CHOOSE CTE STEP SIZE
alpha=...;

%CHOOSE NUMBER OF ITERATIONS
iter=...;
```

```
%CALCULUS OF STOCHASTIC SUBGRADIENT
g1=-tau;

k=0;
for i=1:iter
    %CHOOSE CHANGING STEP SIZE (if not constant)
    %alpha=...;

    k=k+1;
    if k>n
        k=1;
    end

    g2=lambda.*sign(mu);
    [~,g3]=sg_max(mu,X(k,:)',c);

    g=g1+g2+g3;

    %SUBGRADIENT METHOD STEP
    diff=-alpha*g;

    %NEW POINT
    mu=mu+diff; %usual point
    mu_aux=((i-1)/i)*mu_aux+(1/i)*mu; %average of
       all the points obtained so far

    %CALCULUS OF BEST POINT OBJECTIVE FUNCTION VALUE
    f_i=f(mu,X,tau,lambda,c);
    f_best=min([f_best f_i]);

end

f(mu,X,tau,lambda,c) %usual final objective function
    value
f(mu_aux,X,tau,lambda,c) %averaging technique final
   objective function value
f_best %best point approach final objective function
    value
```

Note that `sg_max`, `f` and `phi` functions are the same as the ones defined in sections B.1.1-B.1.3-B.1.2.

Appendix C

Codes for the subgradient methods for MRCs with 0-1 loss problem

C.1 Classic Subgradient Method for constrained problem

```
%LOAD DATASET
load()

%CALCULUS OF NECESSARY PARAMETERS
n=size(X,1);
d=size(X,2);
c=length(unique(Y));

I=eye(c);
kron_matrix=zeros(c*d,n);
for i=1:n
    kron_matrix(:,i)=kron(I(:,Y(i)),X(i,:)');
end
[standesv,tau]=std(kron_matrix,1,2);
lambda=standesv/sqrt(n);

mu=zeros(c*d,1);
nu=-1/2;

%FOR AVERAGING
```

```
mu_aux=mu;
nu_aux=nu;

f_best=Inf; %for best point approach

%CHOOSE CTE STEP SIZE
alpha=...;

%CHOOSE NUMBER OF ITERATIONS
iter=...;

%CALCULUS OF SUBGRADIENT
g1=-tau;

for i=1:iter
    %CHOOSE CHANGING STEP SIZE (if not constant)
    %alpha=...;

    g2=lambda.*sign(mu);
    g_mu=g1+g2;
    g_nu=1;

    for j=1:n
        [maximum,sg]=sg_max(mu,X(j,:)',c);

        if maximum-nu>0
            g_mu=sg;
            g_nu=-1;
            break
        end
    end

    %NEW POINT
    mu=mu-alpha*g_mu;
    nu=nu-alpha*g_nu;

    %AVERAGING METHOD
    mu_aux=((i-1)/i)*mu_aux+(1/i)*mu;
    nu_aux=((i-1)/i)*nu_aux+(1/i)*nu;

    %CALCULUS OF BEST POINT OBJECTIVE FUNCTION VALUE
    f_i=f_constrain(mu,X,tau,lambda,c);
    f_best=min([f_best f_i]);
```

```
end
f_constrain(mu,X,tau,lambda,c) %usual final
   objective function value
f_constrain(mu_aux,X,tau,lambda,c) %averaging
   technique final objective function value
f_best %best point approach final objective function
    value
```

C.1.1 f_constrain function

```
function objective=f_constrain(mu,X,tau,lambda,c)
    objective=1-tau'*mu+lambda'*abs(mu)+
       phi_constrain(X,mu,c);

end
```

C.1.2 phi function

```
function f=phi_constrain(X,mu,c)
    n=size(X,1);

    [maxi,~]=sg_max(mu,X(1,:)',c);
    for i=2:n
        [new,~]=sg_max(mu,X(i,:)',c);
        if new>maxi
            maxi=new;
        end
    end
    f=maxi;
end
```

Note that `sg_max` function is the same as the one defined in section B.1.1.