Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Acceleration Methods for Classic Convex Optimization Algorithms

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

By
Alberto Torres Barrán

under the direction of
José R. Dorronsoro Ibero

Madrid, July 11, 2017

Department: Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid (UAM)
Spain

Title: Acceleration Methods for Classic Convex Optimization Algorithms

Author: Alberto Torres Barrrán

Advisor: José R. Dorronsoro Ibero

Date: June 2017

Committee:

- President: Aníbal Ramón Figueiras Vidal
- Secretary: Daniel Hernández Lobato
- Vocal 1: César Hervás Martínez
- Vocal 2: María Amparo Alonso Betanzos
- Vocal 3: David Ríos Insua
- Substitute 1: Ana María González Marcos
- Substitute 2: Pedro Antonio Gutiérrez

Abstract

Most Machine Learning models are defined in terms of a convex optimization problem. Thus, developing algorithms to quickly solve such problems its of great interest to the field. We focus in this thesis on two of the most widely used models, the Lasso and Support Vector Machines. The former belongs to the family of regularization methods, and it was introduced in 1996 to perform both variable selection and regression at the same time. This is accomplished by adding a ℓ_1-regularization term to the least squares model, achieving interpretability and also a good generalization error.

Support Vector Machines were originally formulated to solve a classification problem by finding the maximum-margin hyperplane, that is, the hyperplane which separates two sets of points and its at equal distance from both of them. SVMs were later extended to handle non-separable classes and non-linear classification problems, applying the kernel-trick. A first contribution of this work is to carefully analyze all the existing algorithms to solve both problems, describing not only the theory behind them but also pointing out possible advantages and disadvantages of each one.

Although the Lasso and SVMs solve very different problems, we show in this thesis that they are both equivalent. Following a recent result by Jaggi, given an instance of one model we can construct an instance of the other having the same solution, and vice versa. This equivalence allows us to translate theoretical and practical results, such as algorithms, from one field to the other, that have been otherwise being developed independently. We will give in this thesis not only the theoretical result but also a practical application, that consists on solving the Lasso problem using the SMO algorithm, the state-of-the-art solver for non-linear SVMs. We also perform experiments comparing SMO to GLMNet, one of the most popular solvers for the Lasso. The results obtained show that SMO is competitive with GLMNet, and sometimes even faster.

Furthermore, motivated by a recent trend where classical optimization methods are being re-discovered in improved forms and successfully applied to many problems, we have also analyzed two classical momentum-based methods: the Heavy Ball algorithm, introduced by Polyak in 1963 and Nesterov's Accelerated Gradient, discovered by Nesterov in 1983. In this thesis we develop practical versions of Conjugate Gradient, which is essentially equivalent to the Heavy Ball method, and Nesterov's Acceleration for the SMO algorithm. Experiments comparing the convergence of all the methods are also carried out. The results show that the proposed algorithms can achieve a faster convergence both in terms of iterations and execution time.

Resumen

La mayoría de modelos de Aprendizaje Automático se definen en términos de un problema de optimización convexo. Por tanto, desarrollar algoritmos para resolver rápidamente dichos problemas es de gran interés para este campo. En esta tesis nos centramos en dos de los modelos más usados, Lasso y Support Vector Machines. El primero pertenece a la familia de métodos de regularización, y fue introducido en 1996 para realizar selección de características y regresión al mismo tiempo. Esto se consigue añadiendo una penalización $\ell_1$al modelo de mínimos cuadrados, obteniendo interpretabilidad y un buen error de generalización.

Las Máquinas de Vectores de Soporte fueron formuladas originalmente para resolver un problema de clasificación buscando el hiper-plano de máximo margen, es decir, el hiper-plano que separa los dos conjuntos de puntos y está a la misma distancia de ambos. Las SVMs se han extendido posteriormente para manejar clases no separables y problemas de clasificación no lineales, mediante el uso de núcleos. Una primera contribución de este trabajo es analizar cuidadosamente los algoritmos existentes para resolver ambos problemas, describiendo no solo la teoría detrás de los mismos sino también mencionando las posibles ventajas y desventajas de cada uno.

A pesar de que el Lasso y las SVMs resuelven problemas muy diferentes, en esta tesis demostramos que ambos son equivalentes. Continuando con un resultado reciente de Jaggi, dada una instancia de uno de los modelos podemos construir una instancia del otro que tiene la misma solución, y viceversa. Esta equivalencia nos permite trasladar resultados teóricos y prácticos, como por ejemplo algoritmos, de un campo al otro, que se han desarrollado de forma independiente. En esta tesis mostraremos no solo la equivalencia teórica sino también una aplicación práctica, que consiste en resolver el problema Lasso usando el algoritmo SMO, que es el estado del arte para la resolución de SVM no lineales. También realizamos experimentos comparando SMO a GLMNet, uno de los algoritmos más populares para resolver el Lasso. Los resultados obtenidos muestran que SMO es competitivo con GLMNet, y en ocasiones incluso más rápido.

Además, motivado por una tendencia reciente donde métodos clásicos de optimización se están re- descubriendo y aplicando satisfactoriamente en muchos problemas, también hemos analizado dos métodos clásicos basados en "momento": el algoritmo Heavy Ball, creado por Polyak en 1963 y el Gradiente Acelerado de Nesterov, descubierto por Nesterov en 1983. En esta tesis desarrollamos versiones prácticas de Gradiente Conjugado, que es equivalente a Heavy Ball, y Aceleración de Nesterov para el algortimo SMO. Además, también se realizan experimentos comparando todos los métodos. Los resultados muestran que los algoritmos propuestos a menudo convergen más rápido, tanto en términos de iteraciones como de tiempo de ejecución.

Acknowledgements

First of all, I would like to thank my supervisor José R. Dorronsoro Ibero for all the help and advice during this period. I am also immensely grateful to all the professors who volunteered to come to the thesis panel and read this dissertation. I express my gratitude to Professor Johan A.K. Suykens for giving me the opportunity to join the ESAT department at KU Leuven during the research stay of 2015.

I would also like to mention all the grants and institutions that supported my research: FPU12/05163 grant, funded by "Ministerio de Economía, Cultura y Deporte"; FPI grant, funded by "Universidad Autónoma de Madrid"; "Cátedra IIC Modelado y Predicción" funded by "Instituto de Ingeniería del Conocimiento" and "Instituto de Ciencias Matemáticas" (ICMAT-CSIC). Finally, I gratefully acknowledge the computational resources provided by "Centro de Computación Científica" (CCC) at "Universidad Autónoma de Madrid".

Contents

List of Tables

List of Figures

List of Algorithms

Notation

Matrices are denoted in upper-case bold font ($\mathbf{X}$), whereas vectors are denoted in lower-case bold, ($\mathbf{x}$). Plain font stands for scalars (x), usually lower-case, although some important scalar constants are denoted in upper-case (for instance, C). Upper-case plain font is also used for random variables (X). Sets are usually denoted by calligraphic font ($\mathcal{X}$) and spaces with blackboard bold font ($\mathbb{R}$). Components of a vector are denoted in subscript (x_i); when the component is another sub-vector the bold face is mantained ($\mathbf{x}_i$). The components of a matrix $\mathbf{X}$ are denoted by two subscripts ($X_{i,j}$). A single subscript on a matrix may indicate both the vector corresponding to a single row or column ($\mathbf{X}_i$). This is indicated when it is not clear by the context. For a general sequence brackets are employed ($\{\mathbf{x}^k\}$) and element of such sequences are denoted in superscript ($\mathbf{x}^k$). A superscript $*$ indicates the limit of a sequence, such as the optimum ($\mathbf{x}^*$).

All the non-standard operators are defined on their first use. Regarding the standard ones, $\|\mathbf{x}\|$ indicates the norm of a vector $\mathbf{x}$, ∇ is used for the gradient and $\mathbf{x} \cdot \mathbf{y}$ denotes the inner product between vectors $\mathbf{x}$ and $\mathbf{y}$. The inner product is sometimes also written as $\mathbf{x}^\top \mathbf{y}$. The transpose of a matrix is $\mathbf{X}^\top$ and its inverse $\mathbf{X}^{-1}$. ∂ stands for both the partial derivative and the subdifferential, and it should be clear from the context which is which. The standard big O notation is written as $\mathcal{O}(\cdot)$.

Chapter 1

Introduction

Machine learning is a branch of artificial intelligence commonly defined as the science of learning from data without the need for the computer to be explicitly programmed. Other fields that overlap with machine learning are statistics, data mining and pattern recognition, since they often use the same methods. Some example of learning problems are:

- Learn to distinguish between spam and non-spam email messages and classify new messages accordingly.
- Predict the selling price of a house based on facts such as square meters and number of bedrooms.
- Predict the price of a stock in 6 months from now, on the basis of company performance measures and economic data (Hastie et al., 2003).
- Recognize handwritten digits from a digitized image, for example ZIP codes in letters.
- Identify the risk factors for prostate cancer, based on clinical and demographic variables (Hastie et al., 2003).
- Identify the clients of an insurance company who are likely to upgrade their policy to premium if an offer is made to them.

In the typical scenario we have an outcome measurement we want to predict, which can be either quantitative (price of the house) or categorical (spam/non-spam), based on a set of features or variables. We have a set of data, in which we observe both the features and the outcome, and the goal is to build a prediction model which allows us to predict the outcome of new samples, not used to train the model.

Notice that it is impossible to compute the accuracy of the predictions for the new data, since the real outcome is not available. So, in order to asses the quality of the model, the original dataset is usually divided into a training set and a test set. The former is used to build the prediction model while the latter is used to estimate its performance. This is done by computing the accuracy of the predictions in the test set and, since they were not used for training, they are a good estimate of the accuracy for new data (assuming they are *similar*).

The ability to predict correctly the outcome of new unseen data is known as generalization. In practice the variability of the data will be such that the training set can comprise only a small amount of all possible examples, so generalization is a central goal in machine learning.

Machine learning systems also have to deal with the representation of the data. For instance, in the case of handwritten digits recognition, it is usually convenient to represent the digitized

images as numerical vectors using a greyscale approach. Data is also usually pre-processed to transform it into another space where the problem of building the model is easier to solve. Continuing with the example of digit recognition, the images are typically traslated and scaled so that each digit is contained in a fixed size box (Bishop, 2007). Note that new data must be processed using the same steps as the training data.

So far we have only mentioned the task where the training data contains both the features and the outcome. That kind of data is said to be *labeled*, and such applications are known as *supervised learning* problems. These problems can be further divided into *classification*, if the outcome is a finite number of discrete categories, or *regression*, if the outcome consists of one or more continuous variables. Examples of classification problems are spam detection, handwritten digits recognition and identifying possible premium users. Examples of regression problems are house pricing, prostate cancer detection and stock price prediction.

There are other machine learning problems where the input data consists only of features, and no outcome is available. In those *unsupervised learning* problems the goal is to find some kind of structure in the data: groups of similar examples (*clustering*), distribution of the data in the input space (*density estimation*) or subspaces in which the data is still "meaningful" but with less dimensions (*dimensionality reduction*).

1.1 Outline

The rest of thesis is organized as follows:

1. Chapter 2 contains the necessary mathematical background to understand this thesis: learning theory, where we include examples of the simplest linear models, and the basics of convex optimization. We skip and therefore assume the reader is familiar with basic results in linear algebra and probability theory.

2. Chapter 3 presents a detailed review of the theory behind the Lasso model and the different optimization algorithms that exist in the literature to solve it. The sparsity of the Lasso regularization was exploited in Alaíz et al. (2012) and Torres et al. (2014b).

3. Chapter 4 is analogous to Chapter 3 but regarding the SVM model. We will address the formulation of the different variants and the different methods used to optimize them. The generalization capabilities of SVMs were studied in different real-world problems: high-content screening (Azegrouz et al., 2013), wind energy prediction (Torres et al., 2014a; Alonso et al., 2015; Díaz et al., 2015), and photovoltaic energy prediction (Catalina et al., 2016).

4. Chapter 5 includes the equivalence between the Lasso and SVM models. Furthermore, a practical application is suggested and several experiments are carried out. This chapter contains novel material from Alaíz et al. (2015), where cited.

5. Chapter 6 reviews two classical acceleration techniques in convex optimization, the Heavy Ball method and Nesterov's Accelerated Gradient, and two new versions of the SMO algorithm based on both of them are also derived. This chapter contains novel material from Torres-Barrán and Dorronsoro (2015), Torres-Barrán and Dorronsoro (2016a), and Torres-Barrán and Dorronsoro (2016b), where cited.

6. Chapter 7 states the conclusions of the thesis and suggests ways to extend the presented ideas

7. Appendices A to C contain some complementary material, included for completeness. Finally in Appendix D we also include a comprehensive list of all the articles accepted in conferences and journals during the elaboration of the thesis.

1.2 Contributions

In the context of Machine Learning models, two of the most widely used are the Lasso and Support Vector Machines. They are both defined in terms of an optimization problem and many algorithms exist to solve such problems. It is of current interest to find faster, scalable and more efficient algorithms to solve them, so they are able to cope with the rapid increase in the amount of available data. The contributions of the thesis can be summarized as:

- A thorough review of the Lasso and the different algorithms to solve the underlying optimization problem. We will begin with the earlier methods, such as LARS, and finish with some recent advances regarding coordinate and stochastic optimization.
- A review of SVMs and the different variants: SVC, SVR, ν-SVC and One-class SVM. We will also analyze different algorithms to solve the SVM problem, focused on its non-linear SVC variant.
- A refined version of the recent relation between the Lasso and the SVC problems, that opens the way to solve one problem using algorithms designed for the other and viceversa, among others. We will also exploit this equivalence to suggest another algorithm to solve the Lasso problem in practice (Alaíz et al., 2015).
- The proposal and analysis of two modifications of the SMO algorithm, related to Conjugate Gradient Descent (Torres-Barrán and Dorronsoro, 2015; Torres-Barrán and Dorronsoro, 2016a) and Nesterov's Accelerated Gradient (Torres-Barrán and Dorronsoro, 2016b). First we will review two classic acceleration techniques for Gradient Descent, Heavy Ball and Nesterov's Acceleration. Then, we will establish the connections between Heavy Ball and Conjugate Gradient, as well as Heavy Ball and Nesterov's Acceleration. Finally, we integrate these techniques in the SMO algorithms and derive the complexity in terms of floating point operations.
- Experimental results comparing the behavior of all the suggested algorithms.
- Software implementations of such algorithms.

Chapter 2

Mathematical background

2.1 Machine Learning

This section reviews the fundamentals of regression, classification and model selection. We start by introducing the basic learning theory concepts through the simplest linear regression model, least squares. Then, we give its classification counterpart, logistic regression. Finally we move on to the topics of model selection and regularization. For a more extensive treatment of machine learning we refer the reader to Shalev-Shwartz and Ben-David (2014).

2.1.1 Regression

A linear regression model has the form

$$\mathbf{y} = f(\mathbf{X}) = \mathbf{X}\mathbf{w} + w_0, \tag{2.1.1}$$

where $\mathbf{X}$ is the $n \times d$ data matrix and $\mathbf{y}$ is the $n \times 1$ response vector (n number of samples or observations, d number of variables). The linear model either assumes that the regression function $\mathbb{E}(y|\mathbf{X})$ is linear or that the linear model is a reasonable aproximation, which is usually the case. Therefore we assume that the true underlying model is

$$\mathbf{y} = f(\mathbf{X}) = \mathbf{X}\mathbf{w} + w_0 + \boldsymbol{\epsilon}, \tag{2.1.2}$$

where $\boldsymbol{\epsilon} \sim N(0, \sigma)$ is the *noise*, and it stresses the fact that the linear model is only an aproximation of the underlying true model, since it is very difficult in practice to have real data that comes from a perfectly linear model.

The term w_0 is known as the bias or intercept and it is usually included into the vector $\mathbf{w}$. That way, if we also add a column of $\mathbf{1}$s to the matrix $\mathbf{X}$, we can write the model in the more convenient form

$$\mathbf{y} = \mathbf{X}\mathbf{w}.$$

The bias can also be omitted if the response vector $\mathbf{y}$ and the columns of $\mathbf{X}$ are centered to have zero mean, that is, $\mathbb{E}[\mathbf{y}] = 0$ and $\mathbb{E}[\mathbf{X}_j] = 0$ for $j = 1, \ldots, d$.

The components of the vector $\mathbf{w}$, w_j, are known as parameters or coefficients and the columns of the matrix $\mathbf{x}_j$ are the variables or features. These variables can come from different sources (Hastie et al., 2003):

- quantitative inputs;
- transformations of quantitative inputs, $\mathbf{x}_3 = \mathbf{x}_1^\top \mathbf{x}_2$;

- basis expansions, such as $\mathbf{x}_2 = \mathbf{x}_1^2$;
- "dummy" variables coding of the levels of qualitative inputs. For instance if we have a feature with 5 possible values we might create five different variables that are all set to 0 but one.

Let $\mathbf{x}_i$ be now the ith pattern, that is, the ith row of the matrix $\mathbf{X}$. Typically we have a *training set* $\{\mathbf{x}_i, y_i\}, i = 1, \ldots, n$, from which to estimate the parameters $\mathbf{w}$. The most popular estimation method is ordinary least squares (OLS), in which coefficients are obtained by minimizing the residual sum of squares, defined as

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \mathbf{w})^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2, \tag{2.1.3}$$

that is

$$\widehat{\mathbf{w}} = \text{argmin} \left\{ \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \right\}, \tag{2.1.4}$$

where

$$\|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} w_j^2,$$

It is easy to show that the optimization problem (2.1.4) has a closed-form solution. Differentiating in (2.1.3) with respect to $\mathbf{w}$ we obtain

$$\begin{aligned} \frac{\partial \text{RSS}}{\partial \mathbf{w}} &= -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}), \\ \frac{\partial^2 \text{RSS}}{\partial \mathbf{w}^2} &= 2\mathbf{X}^\top \mathbf{X}. \end{aligned} \tag{2.1.5}$$

Assuming that $\mathbf{X}$ has full column rank and hence $\mathbf{X}^\top \mathbf{X}$ is positive definite, we set the first derivative to zero

$$\mathbf{X}^\top (y - \mathbf{X}\mathbf{w}) = 0, \tag{2.1.6}$$

to obtain the unique solution

$$\widehat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{2.1.7}$$

The fitted values at the training inputs are

$$\widehat{\mathbf{y}} = \mathbf{X}\widehat{\mathbf{w}} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{2.1.8}$$

The matrix $\mathbf{H} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is sometimes called the "hat" matrix. We can also make predictions for new data $\widetilde{\mathbf{X}}$ that was not used to fit the model. The predicted values are given by $f(\widetilde{\mathbf{X}}) = \widetilde{\mathbf{X}}\widehat{\mathbf{w}}$.

From a geometrical point of view, the least squares solution is the $d+1$ dimensional hyperplane that minimizes the sum of squared residuals. The coefficients are chosen so that the residual vector $\mathbf{y} - \hat{\mathbf{y}}$ is orthogonal to the subspace spanned by the columns of the input matrix $\mathbf{X}$. This orthogonality is expresed in Eq. (2.1.5), and the resulting estimate $\hat{\mathbf{y}}$ is the orthogonal projection of $\mathbf{y}$ into this subspace. Hence the matrix $\mathbf{H}$ is also known as the projection matrix. Figure 2.1 shows an example of the regression hyperplane in a two-dimensional space for randomly generated data with $w_1 = 0.2$, $w_0 = 2$ and $\epsilon \sim N(0, 2.5)$.

It may happen that the columns of $\mathbf{X}$ are not linearly independent so that $\mathbf{X}$ is not full rank. This is the case, for example, if two of the variables are perfectly correlated. Then the matrix

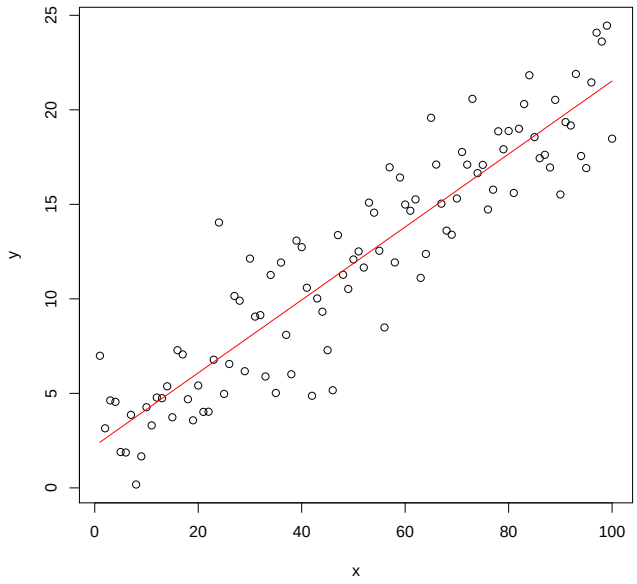

Figure 2.1: Linear least squares fitting with $\mathbf{X} \in \mathbb{R}$

$\mathbf{X}^\top\mathbf{X}$ is singular, the inverse in Eq. (2.1.7) cannot be computed and the coefficients $\mathbf{w}$ are not uniquely defined. The natural way to solve this non-unique representation is to remove from the matrix $\mathbf{X}$ all the redundant variables. Another option is to control the fit by regularization, as we will discuss in Section 2.1.4.

Rank deficiencies also occur when the number of variables d is bigger than the number of observations n. This happens frequently in fields such as image processing and bioinformatics. Redundant variables are non-existent in this case, so the less important ones must be filtered for the problem to be solvable.

2.1.2 Classification

As explained before, in the classification problem the predictor $G(x)$ takes values in a discrete set $\mathcal{G}$. Therefore we can always divide our input space into regions labeled according to the classification label. In the linear methods for classification, these *decision boundaries* will be linear.

One of the most common models is logistic regression, where we estimate the posterior probabilities of the K classes via linear functions in x, while at the same time we make sure that they sum one. Let $\mathbf{x}_i$ be the ith observation (row) of the input matrix $\mathbf{X}$ and y_i the associated label. For the sake of simplicity, we are going to assume that $K = 2$ and we are going to label the two classes as 0/1. Then, the probability of belonging to the positive class ($y_i = 1$) is defined as

$$p(y_i = 1|\mathbf{x}_i) = \pi(\mathbf{x}_i) = \frac{1}{1 + \exp(-\mathbf{w}^\top\mathbf{x}_i)}, \tag{2.1.9}$$

and the probability of belonging to the negative class ($y_i = 0$) is then

$$p(y_i = 0|\mathbf{x}_i) = 1 - \pi(\mathbf{x}_i) = \pi(-\mathbf{x}_i) = \frac{1}{1 + \exp(\mathbf{w}^\top\mathbf{x}_i)}. \tag{2.1.10}$$

Logistic regression models are usually fit by maximum likelihood, using the conditional likelihood of G given $\mathbf{X}$. The log-likelihood for n observations is

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{n} y_i \log \pi(\mathbf{x}_i) + (1 - y_i) \log \pi(-\mathbf{x}_i) \\ &= \sum_{i=1}^{n} y_i \mathbf{w}^\top \mathbf{x}_i + \log \pi(-\mathbf{x}_i).\end{aligned}$$

Next we have to maximize the log-likelihood (or minimize the negative log-likelihood, which is usually more convenient). The derivatives of the log-likelihood with respect to $\mathbf{w}$ are

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^{n} \mathbf{x}_i (y_i - \pi(\mathbf{x}_i)), \tag{2.1.11}$$

that is, $d + 1$ equations nonlinear in $\mathbf{w}$. This optimization can be performed with different algorithms such as *iteratively reweighted least squares* (IRLS) or Newton-Raphson. It is worth mentioning that maximum likelihood can exhibit severe overfitting for datasets that are linearly separable (Bishop, 2007). Once the weights are estimated, new observations can be classified using the rule

$$\hat{y} = G(\mathbf{x}) = \begin{cases} 0 & \text{if } \pi(\mathbf{x}) \geq 0.5 \\ 1 & \text{if } \pi(\mathbf{x}) < 0.5 \end{cases}$$

Last, it is also important to mention that logistic regression can be extended to the multinomial setting ($K > 2$). See Hastie et al. (2003) and Bishop (2007) for more details.

2.1.3 Model selection

Let's assume that data comes from a model

$$Y = f(X) + \epsilon \tag{2.1.12}$$

with $\mathbb{E}[\epsilon] = 0$ and $\text{Var}(\epsilon) = \sigma^2$. Throughout this section we will also assume that the matrix X is fixed in advance. Then, the expected prediction error of an estimator $\hat{f}(X)$ at a point x, also know as simply prediction, test or generalization error is defined as

$$\text{PE} = \mathbb{E}[(Y - \hat{f}(x))^2] \tag{2.1.13}$$

and it can be decomposed into

$$\begin{aligned}\text{PE} &= \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - \mathbb{E}[\hat{f}(x)]\right]^2 + \text{Var}(\epsilon) \\ &= \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - \mathbb{E}[\hat{f}(x)]\right]^2 + \sigma^2 \\ &= \text{Bias}^2 + \text{Variance} + \text{Noise}.\end{aligned}$$

The third term is the irreducible error, the inherent uncertainty in the true relationship and cannot be reduced by any model. The first and second terms are under our control and make up the *mean squared error* of $\hat{f}(x)$ in estimating $f(x)$.

In practice, for every model there is a *bias-variance tradeoff* (Geman et al., 1992), when varying its "complexity" parameters. A common problem is known as *overfitting*, and it occurs when a statistical model is too complex and exhibits poor generalization.

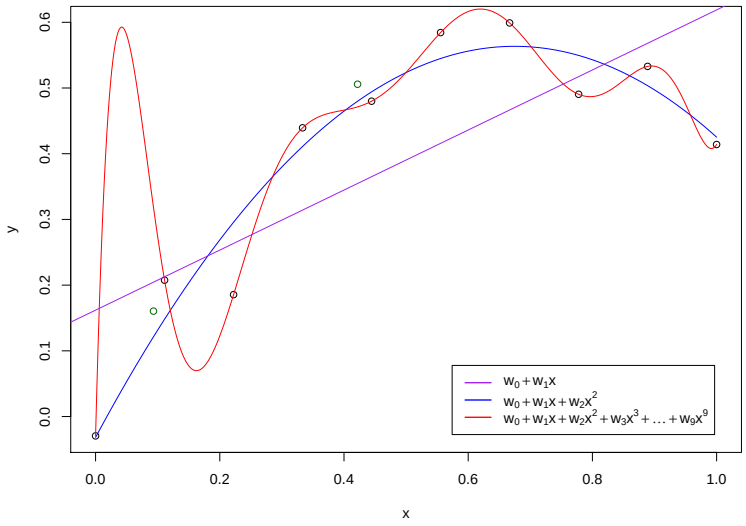

Figure 2.2: Points are generated from a random quadratic model with Gaussian noise and they are fitted to linear (purple line), quadratic (blue) and polynomial functions (red). Green points are also drawn from the same model but not used to fit the functions.

This means that the model is memorizing the particular structure of the training data, but it is unable to learn the underlying relationship. As a result, such model will perform poorly on new unseen data, also known as test data.

As an example consider the linear model,

$$y = 1.5x - x^2 + \mathcal{N}(0, 0.05).$$

If we fit a linear model to the variable x we obtain the straight line depicted in purple in Fig. 2.2. This model is clearly miss-specified, since the true model used to generate the data contains also x^2. This behavior is known as *underfitting*. On the other hand, we could always add higher-order exponents of the variable x until we achieve a perfect fit, since the model is still **linear** in the coefficients $\mathbf{w}$. In general, if we have n data points we could achieve zero training error using a polynomial of degree $n-1$. This is represented as the red line in the figure and it will clearly exhibit poor generalization because it is also learning the noise of the data. We simulate the test data as the two green points, also drawn from the same model, but not used to compute any of the fits. It is clear that they are much closer to the blue line than to the red line, even though the latter achieves a zero error in the training points. Thus we could consider the quadratic model to be a better estimation of the true model. As we mentioned before this is known as *overfitting* and it is a very important problem when fitting statistical models since in practice the true model is usually not known.

In the previous example we could consider the degree of the polynomial d to be the complexity parameter. If $d \geq 9$ the model is very complex and it fits the training points perfectly, i.e the bias is 0. However, the variance is very large, since for any other point not in the training set the model will perform poorly, for instance the green points in Fig. 2.2. On the other hand, if $d = 2$, the model is much simpler. Now the bias is clearly not zero but the variance is reduced significantly, so we may consider this to be a better model.

Typically we would like to choose our model complexity to trade bias off with variance in

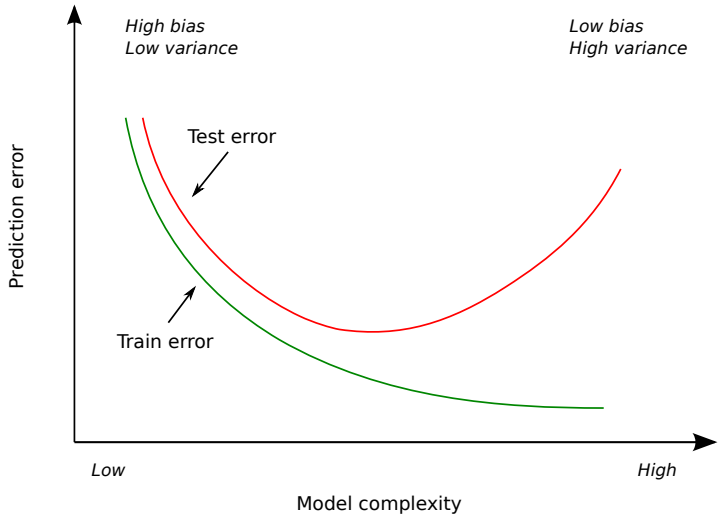

Figure 2.3: Test and training error as a function of the model complexity (Hastie et al., 2003)

such a way as to minimize the test error, which is the error of new observations (not used to fit the model). Figure 2.3 shows the typical behavior of train and test error as model complexity is changed. The training error tends to decrease whenever we increase the model complexity, overfitting the data. However, with too much fitting the model adapts too closely and will not generalize well. In contrast, if the model is not complex enough, it will underfit and may have large bias.

2.1.4 Regularization

In the previous sections, given some data $\mathcal{D}$, we minimize some kind of error function $E_{\mathcal{D}}(\mathbf{w})$. However, as we mentioned before, minimizing the error term alone is often an ill-posed problem: solutions are not unique and sensitive to data variations. Therefore, a regularization term is usually added to enforce some desirable properties on the solution, such us smoothness, sparsity, low-rank, and so on, changing the criterion function to

$$J(\mathbf{w}) = \underbrace{E_{\mathcal{D}}(\mathbf{w})}_{\text{error}} + \underbrace{\omega(\mathbf{w})}_{reg.}. \tag{2.1.14}$$

Regularization also helps to control the complexity of the problem and avoid *overfitting*. For instance if we consider the least square estimates, in theory Theorem 2.1 states that they are the *Best Linear Unbiased Estimates* (BLUE), where "best" refers to having the lowest variance.

Theorem 2.1 (Gauss-Markov (Plackett, 1950))**.** *Given a linear model with noise* $\epsilon \sim N(0, \sigma)$*, i.e*

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}, \tag{2.1.15}$$

then the Ordinary Least Squares (OLS) estimates,

$$\widehat{\mathbf{w}} = \operatorname{argmin} \left\{ \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \right\}, \tag{2.1.16}$$

are the Best Linear Unbiased Estimates (BLUE).

Proof. We begin by showing that they are indeed unbiased. Substituting Eq. (2.1.15) into Eq. (2.1.7) we get,

$$\begin{aligned}\widehat{\mathbf{w}} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}\\ &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top(\mathbf{X}\mathbf{w}+\boldsymbol{\epsilon})\\ &= \mathbf{w}+(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{\epsilon}.\end{aligned}$$

Finally, we fix the data matrix $\mathbf{X}$ and take expectations on both sides,

$$\mathbb{E}[\widehat{\mathbf{w}}] = \mathbf{w}+\mathbb{E}[(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{\epsilon}] = \mathbf{w}+(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbb{E}[\boldsymbol{\epsilon}] = \mathbf{w}, \tag{2.1.17}$$

since $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $\mathbf{X}$ is constant. Now we are going to prove that they are the ones with the lowest variance. Let $\mathrm{Var}(y) = \sigma^2$, then the variance of the OLS estimates is

$$\begin{aligned}\mathrm{Var}(\widehat{\mathbf{w}}) &= \mathrm{Var}\left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}\right)\\ &= ((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top)\,\mathrm{Var}(\mathbf{y})\left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\right)^\top\\ &= \mathrm{Var}(\mathbf{y})((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top)(\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1})\\ &= \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}.\end{aligned}$$

Suppose that $\widetilde{\mathbf{w}}$ is another linear estimator,

$$\widetilde{\mathbf{w}} = \left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D}\right)\mathbf{y}, \tag{2.1.18}$$

where $\mathbf{D}$ is a $p\times n$ non-zero matrix. The expectation of $\widetilde{\mathbf{w}}$ is

$$\begin{aligned}\mathbb{E}[\widetilde{\mathbf{w}}] &= \mathbb{E}[((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})\mathbf{y}]\\ &= \mathbb{E}[((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})(\mathbf{X}\mathbf{w}+\boldsymbol{\epsilon})]\\ &= \mathbb{E}[((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})(\mathbf{X}\mathbf{w})]+\mathbb{E}[(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})\boldsymbol{\epsilon}]\\ &= ((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})(\mathbf{X}\mathbf{w})+((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})\mathbb{E}[\boldsymbol{\epsilon}]\\ &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{X}\mathbf{w}+\mathbf{D}\mathbf{X}\mathbf{w}\\ &= (\mathbf{I}+\mathbf{D}\mathbf{X})\mathbf{w}.\end{aligned}$$

From this we conclude that $\mathbf{D}\mathbf{X} = 0$ for this estimator to be unbiased. The variance of $\widetilde{\mathbf{w}}$ is

$$\begin{aligned}\mathrm{Var}(\widetilde{\mathbf{w}}) &= \mathrm{Var}\left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D}\right)\mathbf{y})\\ &= \mathrm{Var}(\mathbf{y})((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})^\top\\ &= \sigma^2((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top+\mathbf{D})(\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}+\mathbf{D}^\top)\\ &= \sigma^2((\mathbf{X}^\top\mathbf{X})^{-1}+(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}^\top+\mathbf{D}\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}+\mathbf{D}\mathbf{D}^\top)\\ &= \sigma^2((\mathbf{X}^\top\mathbf{X})^{-1}+2(\mathbf{X}^\top\mathbf{X})^{-1}(\mathbf{D}\mathbf{X})^\top+\mathbf{D}\mathbf{D}^\top)\\ &= \sigma^2((\mathbf{X}^\top\mathbf{X})^{-1}+\mathbf{D}\mathbf{D}^\top)\\ &= \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}+\sigma^2\mathbf{D}\mathbf{D}^\top\\ &= \mathrm{Var}(\widehat{\mathbf{w}})+\sigma^2\mathbf{D}\mathbf{D}^\top\\ &\geq \mathrm{Var}(\widehat{\mathbf{w}}),\end{aligned}$$

since $\mathbf{D}\mathbf{D}^\top$ is a positive semidefinite matrix, that is, $\mathbf{D}\mathbf{D}^\top \geq 0$. Therefore the least squares estimators $\widehat{\mathbf{w}}$ are the unbiased linear estimators with the lowest variance, concluding the proof. □

However, in practice there are many reason why the least square estimates are not a satisfactory solution to the regression problem. One of these reasons was the non-uniqueness of the OLS solution when the number of variables is larger than the number of observations. Another important one is overfitting.

The simplest example of a regularized model is Ridge Regression, that adds a ℓ_2-penalty term to the least squares objective function,

$$\begin{aligned}\widehat{\mathbf{w}} &= \operatorname{argmin}\left\{\sum_{i=1}^{n}\left(y_i - \sum_j w_j x_{ij}\right)^2 + \lambda \sum_j w_j^2\right\} \\ &= \operatorname{argmin}\left\{\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2\right\}. \end{aligned} \tag{2.1.19}$$

The previous optimization problem is equivalent to (Hastie et al., 2003)

$$\min \quad \sum_{i=1}^{n}\left(y_i - \sum_j w_j x_{ij}\right)^2 \qquad \text{s.t} \quad \sum_j w_j^2 \leq t. \tag{2.1.20}$$

Using similar arguments to the ordinary least squares case, it is easy to show that Ridge Regression has also a closed form solution

$$\widehat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}. \tag{2.1.21}$$

Note that, in contrast to ordinary least squares, now the matrix $(\mathbf{X}^\top\mathbf{X}+\lambda\mathbf{I})$ is always positive definite if $\lambda > 0$ and therefore invertible, resulting in a unique solution for the ridge estimates. Furthermore, the previous models favors smaller coefficients and therefore it is less prone to overfitting.

2.2 Convex optimization

We will start these section by defining some basic concepts about convexity in the Euclidean space, although with some more careful work they can be extended to Hilbert spaces. $\mathbb{R}^n$ and $\mathbb{E}$ are used interchangeably throughout this section to denote the Euclidean space.

Definition 2.1. An Euclidean space $\mathbb{E}$ is a finite-dimensional real vector space with an inner product and its induced norm.

Definition 2.2 (Convex set)**.** A set $C \subseteq \mathbb{E}$ is a convex set if for all $t \in (0, 1)$ the following holds

$$tx + (1-t)y \in C, \quad \forall x, y \in C.$$

Intuitively, the previous definition means that all the points of the segment joining x and y must be inside the set C, and this has to be true for any two points in C. For instance an hyperplane $H = \{x \in \mathbb{R}^n : w^\top x - \beta = 0\}$ or a ball $B = \{x \in \mathbb{R}^n : |x - x_0| \leq \beta\}$ are examples of convex sets. However, the sphere $S = \{x \in \mathbb{R}^n : |x - x_0| = \beta\}$ provides an example of a set that is not convex ($\beta > 0$). Figure 2.4 (left) shows another example of an arbitrary convex set. On the other hand, Fig. 2.4 (right) is an example of a non-convex set. It is easy to see that any intersection of two convex sets is also convex. Given I convex sets S_i, the finite sum is a new set S formed by taking all the terms $\sum s_i$, where $s_i \in S_i$, $i = 1, \ldots, I$. Finite sums of convex sets are also convex. We now introduce the concept of effective domain and convex function.

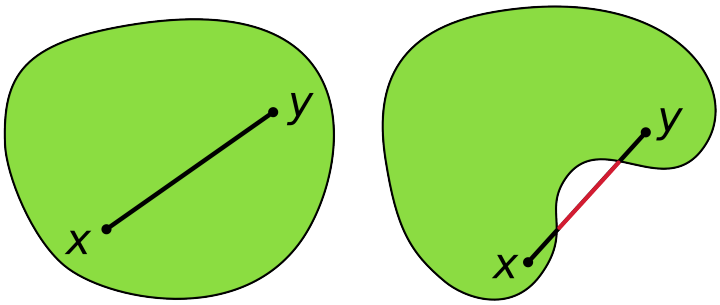

Figure 2.4: Example of convex set (left) and non-covex set (right) [Wikipedia, "Convex set"]

Definition 2.3 (Effective domain)**.** The effective domain of f is the set

$$\mathrm{dom}(f) = \{x \in \mathbb{E} : f(x) < +\infty\}.$$

If $\mathrm{dom}(f)$ is not empty, the function f is called proper.

Definition 2.4 ((Strictly) convex function)**.** A function $f : \mathbb{E} \to \mathbb{R} \cup \{+\infty\}$ is a convex function if $\mathrm{dom}(f)$ is a convex set and, for every $x, y \in \mathbb{E}$ and any $t \in [0, 1]$,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

The same function is said to be strictly convex if for every $x, y \in \mathbb{E}$, $x \neq y$ and any $t \in (0, 1)$

$$f(tx + (1-t)y) < tf(x) + (1-t)f(y).$$

The previous definition does not take into account functions that take the value $-\infty$, although it can be expanded to do so (Balder, 2008).

Most machine learning problems are defined as convex optimization problems. The basic definition of an optimization problem has the form:

Definition 2.5 (Primal optimization problem)**.** Given functions f, g_i, $i = 1, \ldots, k$, and h_i, $i = 1, \ldots, m$, defined over a domain $\Omega \subseteq \mathbb{R}^n$, the primal problem solves

$$\begin{array}{lll} \min & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \text{s.t} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

where $f(\mathbf{w})$ is known as *objective function*, $g_i(\mathbf{w})$ are the inequality constraints and $h_i(\mathbf{w})$ are the equality constraints. From now on we are going to write $g(\mathbf{w}) \leq 0$ to indicate $g_i(\mathbf{w}) \leq 0$, $i = 1, \ldots, k$, and the same with $h(\mathbf{w})$.

Definition 2.5 is general, since maximization problems can be transformed into minimization ones simply by flipping the sign of the objective function $f(\mathbf{w})$. In the same way, every constraint can be re-written in one of the previous forms. For instance, if we have the constrain $\sum \mathbf{w} < t$, we can always write it as $\sum \mathbf{w} - t < 0$.

The domain region where the objective function is defined and all the constraints are satisfied is known as the *feasible region*

$$R = \{\mathbf{w} \in \Omega \mid g(\mathbf{w}) \leq 0,\, h(\mathbf{w}) = 0\}. \tag{2.2.1}$$

Definition 2.6. An optimization problem where the objective function and all the constraints are linear is a linear problem. If the objective function is convex quadratic and the constraints are linear is a quadratic problem.

One of the most well studied problems in optimization is the convex quadratic where the objective function is convex and quadratic and the constraints are linear. The solution of a convex optimization problems is a point $\mathbf{w}^* \in \mathbb{R}^n$ such that there is no other point $\mathbf{w} \in \mathbb{R}^n$ for which $f(\mathbf{w}) < f(\mathbf{w}^*)$. This means that any local minimizer $\mathbf{w}^*$ is also a global minimizer. However, note that the minimum does not have to be unique and there can be many points with the same value $f(\mathbf{w}^*)$ of the objective function. Thus, the solution may be a set instead of a single point. If the objective function is strictly convex then it follows from the definition the minimum is indeed unique.

2.2.1 Duality

Lagrange duality is a very important concept in optimization. Given an optimization problem of the form (2.5), it provides us with an equivalent optimization problem that often has complementary properties. Thus, we can choose to indistinctly solve one or the other depending on the situation. We are going to define first the generalized Lagrangian:

Definition 2.7 (Lagrangian function)**.** Given the optimization problem (2.5), we define the generalized Lagrangian function as

$$\begin{aligned} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w}) \\ &= f(\mathbf{w}) + \boldsymbol{\alpha}^\top g(\mathbf{w}) + \boldsymbol{\beta}^\top h(\mathbf{w}). \end{aligned}$$

We can now define the Lagrangian dual problem

Definition 2.8 (Dual optimization problem)**.** The *Lagrangian dual problem* of the primal problem of Definition 2.5 is the following problem:

$$\begin{array}{ll} \max & \Theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) \\ \text{s.t} & \boldsymbol{\alpha} \geq 0 \end{array}$$

where $\Theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_w L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$

One of the fundamental relationships between the primal and the dual problem is given by the weak duality theorem, presented next.

Theorem 2.2. *(Weak duality theorem) Let $\mathbf{w} \in \Omega$ be a feasible solution of the primal problem (2.5) and $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ a feasible solution of the dual problem (2.8). Then $f(\mathbf{w}) \geq \Theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$.*

Proof. From the definition of $\Theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$ for $\mathbf{w} \in \Omega$ we have

$$\inf_u L(\mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \leq L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \boldsymbol{\alpha}^\top g(\mathbf{w}) + \boldsymbol{\beta}^\top h(\mathbf{w}) \leq f(\mathbf{w}). \tag{2.2.2}$$

The last inequality holds since the feasibility of $\mathbf{w}$ implies that

$$g(\mathbf{w}) \leq 0 \quad \text{and} \quad h(\mathbf{w}) = 0 \tag{2.2.3}$$

while the feasibility of $\boldsymbol{\alpha}$ implies

$$\boldsymbol{\alpha} \geq 0. \tag{2.2.4}$$

Therefore from Eqs. (2.2.3) and (2.2.4)

$$\boldsymbol{\beta}^\top h(\mathbf{w}) = 0 \quad \text{and} \quad \boldsymbol{\alpha}^\top g(\mathbf{w}) \leq 0.$$

□

The difference between the values of the primal and dual problems is known as *duality gap*. The strong duality theorem characterizes what kinds of optimization problems are guaranteed to have a duality gap equal to 0. This means that the dual and the primal problems have the same value. The proof of the strong duality theorem can be found in Ben-Tal and Nemirovski (2001).

Theorem 2.3. *(Strong duality theorem) Given an optimization problem like the one in Definition 2.5, where the g_i and h_i are affine functions, that is*

$$a(\mathbf{w}) = \mathbf{A}\mathbf{w} - \mathbf{b}, \tag{2.2.5}$$

for some matrix $\mathbf{A}$ and vector $\mathbf{b}$, the duality gap is 0.

The last theorem of this section characterizes the optimum solution of a general optimization problem (Cristianini, 2000).

Theorem 2.4. *(Khun-Tucker) Given an optimization problem in the convex domain $\Omega \subseteq \mathbb{R}^n$*

$$\begin{array}{lll} \min & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ s.t & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

with convex f, affine g_i, h_i, the necessary and sufficient conditions for a normal point $\mathbf{w}^*$ to be an optimum are the existence of $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$ such that

$$\begin{aligned} \frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} &= 0, \\ \frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} &= 0, \\ \alpha_i^* g_i(\mathbf{w}^*) &= 0, \quad i = 1, \ldots, k, \\ g_i(\mathbf{w}^*) &\leq 0, \quad i = 1, \ldots, k, \\ \alpha_i^* &\geq 0, \quad i = 1, \ldots, k. \end{aligned}$$

2.2.2 Subgradients

We begin this section by briefly reviewing some important definition that will be used in the subsequent chapters.

Definition 2.9 (Subdifferential)**.** The subdifferential of a proper convex function is the set-valued map $\partial f : \mathbb{E} \to 2^{\mathbb{E}}$, defined as

$$\partial f(x) = \{\xi \in \mathbb{E} : f(x) + \xi^\top (y - x) \leq f(y), \quad \forall y \in \mathbb{E}\}$$

where $2^{\mathbb{E}}$ is the power set, that is, the set of all subsets of $\mathbb{E}$.

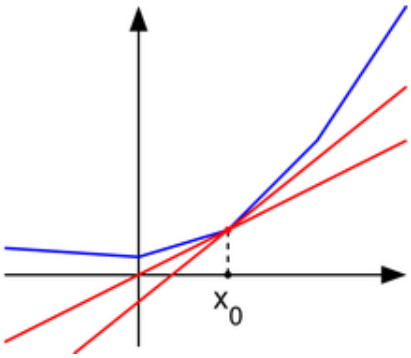

Figure 2.5: A convex function (blue) and "subtangent" lines at x_0 (red) [Wikipedia "Subdifferential"]

Let $x \in \mathbb{E}$. Then f is subdifferentiable at x if $\partial f(x) \neq \emptyset$. The elements of $\partial f(x)$ are the subgradients of f at x. Observe that this definition is only non-trivial if the function is proper, that is, $x \in \operatorname{dom} f$. Otherwise $f(x) = +\infty$ and $\partial f(x) = \emptyset$. Some properties of the subdifferential are:

(i) For any $x \in \mathbb{E}$, $\partial f(x)$ is either empty or a closed convex set (see Bauschke and Combettes, 2011, Proposition 16.3).

(ii) For any $x \in \operatorname{int}(\operatorname{dom}(f))$, $\partial f(x)$ is not empty and bounded (see Balder, 2008, Lemma 2.16).

(iii) $\partial f(x) = \{\nabla f(x)\}$ if and only if f is differentiable at $x \in \mathbb{E}$ (see Balder, 2008, Proposition 2.6).

(iv) A point $x \in \mathbb{E}$ is a (global) minimizer of f if and only if

$$0 \in \partial f(x).$$

This is known as the Fermat's rule in convex optimization, and it will be proven later in this chapter.

Let us define first the concept of convex hull, which will be important in what follows. There is also a generalization, the reduced convex hull, where the coefficients are upper-bounded by $\mu \leq 1$.

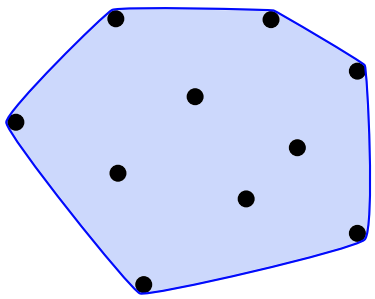

Figure 2.6: Convex hull of a set of points

Definition 2.10 (μ-Reduced Convex hull)**.** The μ-reduced convex hull of a set $\mathcal{X} = \{\mathbf{x}_i\}_1^n$ of points in a d-dimensional Euclidean space is the set of all convex combinations of points in $\mathcal{X}$ or, more formally,

$$\text{conv}_\mu(\mathcal{X}) = \left\{ \sum_{i=1}^{n} \alpha_i x_i \;\middle|\; 0 \leq \alpha_i \leq \mu, \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i = 1 \right\}, \tag{2.2.6}$$

where $\mu = 1$ for the standard convex hull and $\frac{1}{n} \leq \mu < 1$ for the reduced convex hull. For simplicity we will get rid of the subscript when referring to standard convex hulls. It is easily seen that $\text{conv}(\mathcal{X})$ is also the smallest convex set that contains $\mathcal{X}$. This is illustrated in Fig. 2.6.

We will also need some technical results regarding subgradients.

Proposition 2.1 (Maximum of subdifferentiable functions, Boyd and Vandenberghe, 2008)**.** *Suppose f is the pointwise maximum of convex functions $f_1, \ldots, f_m$, i.e.,*

$$f(\mathbf{x}) = \max_i f_i(\mathbf{x}),$$

where the functions f_i are subdifferentiable. Then, the subdifferential of $f(\mathbf{x})$ is the convex hull of the union of subdifferentials of the "active" functions at $\mathbf{x}$, i.e.,

$$\text{conv}\left(\cup\{\partial f_i(\mathbf{x}) \mid f_i(\mathbf{x}) = f(\mathbf{x})\}\right).$$

Proof. Let k be any index for which $f_k(\mathbf{x}) = f(\mathbf{x})$ and let $g \in \partial f_k(\mathbf{x})$. Then, by the definition of subdifferential (Definition 2.9) we have,

$$f(\mathbf{z}) \geq f_k(\mathbf{z}) \geq f_k(\mathbf{x}) + \mathbf{g}^\top(\mathbf{z} - \mathbf{x}) = f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{z} - \mathbf{x}),$$

and $g \in \partial f(x)$. Thus, to find a subgradient of the maximum of functions, we can choose one of the functions that achieves the maximum at the point, and choose any subgradient of that function at the point. The subdifferential is the set of all such subgradients. Doing that for every point $i = 1, \ldots, m$ we get that the subdifferential of f is the convex hull of all subdifferentials constructed this way,

$$\text{conv}\left(\cup\{\partial f_i(x) \mid f_i(x) = f(x)\}\right).$$

□

Example. The ℓ_1-norm

$$f(\mathbf{x}) = \|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n|$$

is a non-differentiable convex function. Note that it can be expressed as the maximum of 2^n linear functions (Boyd and Vandenberghe, 2008):

$$\|\mathbf{x}\|_1 = \max\left\{ \mathbf{s}^\top \mathbf{x} \;\middle|\; s_i \in \{-1,\ 1\} \right\}.$$

Therefore we can apply Proposition 2.1 to find the subdifferential. First we have to identify an active function $\mathbf{s}^\top\mathbf{x}$, that is, find an $\mathbf{s} \in \{-1,\ 1\}^n$ such that $\mathbf{s}^\top\mathbf{x} = \|\mathbf{x}\|_1$. Since the ℓ_1-norm is the sum of the absolute values of the components of x_i, we can take $s_i = +1$ if $x_i > 0$ and $s_i = -1$ if $x_i < 0$. If $x_i = 0$, more than one function is active and both -1 and 1 work. Thus we can take

$$s_i = \begin{cases} +1 & \text{if } x_i > 0, \\ -1 & \text{if } x_i < 0, \\ -1 \text{ or } 1 & \text{if } x_i = 0. \end{cases}$$

and apply Proposition 2.1. The subdifferential is the convex hull of all the subgradients that can be generated this way (Boyd and Vandenberghe, 2008):

$$\partial f(\mathbf{x}) = \left\{ \mathbf{s} \;\middle|\; \|\mathbf{s}\|_\infty \leq 1,\ \mathbf{s}^\top \mathbf{x} = \|\mathbf{x}\|_1 \right\}. \tag{2.2.7}$$

Global minimizers of unconstrained convex optimization problems are characterized by Fermat's theorem. We need first the definition of proximal mapping:

Definition 2.11 (Proximal mapping)**.** Let f be a lower semicontinuous, proper convex function. Then, te proximal mapping is the operator,

$$\text{prox}_f = (I + \partial f)^{-1}.$$

Note also that $x = \text{prox}_f(z) \;\Leftrightarrow\; z - x \in \partial f(x) \;\Leftrightarrow\; 0 \in x - z + \partial f(x)$.

Alternatively, the proximal mapping can be defined also as an optimization problem:

Definition 2.12 (Proximal mapping)**.** Let f be a lower semicontinous, proper convex function. For every $z \in \mathbb{R}^n$, the proximal mapping of f is defined as

$$\text{prox}_f(z) = \underset{y \in \mathbb{R}}{\text{argmin}} \left(f(y) + \frac{1}{2}||z - y||^2 \right).$$

An important example of proximal operator, which will use later in Chapter 3, is the proximal of the ℓ_1-norm, $f(x) = \lambda||x||_1 = \sum_{i=1}^n |x_i|$,

$$[\text{prox}_f(x)]_i = \text{sign}(x_i) \max(0, |x_i| - \lambda). \tag{2.2.8}$$

This is known as the soft-thresholding operator (see Appendix A for a formal derivation). Fermat's theorem, presented next, characterizes the solutions of unconstrained optimization problems and it is one of the most important result in convex optimization.

Theorem 2.5 (Fermat's rule)**.** *Let f be a proper convex function. Then,*

$$\text{argmin}\, f = \text{zer}\, \partial f = \{x \in \mathbb{E} \mid 0 \in \partial f(x)\} \tag{2.2.9}$$

and $x = \text{prox}_f(x)$.

In many optimization problems we further decompose the objective function into two functions: the loss function and the regularizer, as we have seen in Section 2.1.4. The minimization problem then becomes

$$\min_{x \in \mathbb{E}} \; f(x) + g(x). \tag{2.2.10}$$

Theorem 2.6 (Moreau-Rockafellar)**.** *Let f and g be proper convex functions. Then, for every $x_0 \in \mathbb{R}^n$*

$$\partial f(x_0) + \partial g(x_0) \subset \partial(f + g)(x_0).$$

Moreover, suppose that $\text{int}(\text{dom}\, f \cap \text{dom}\, g) \neq \emptyset$*. Then for every $x_0 \in \mathbb{R}^n$ we also have*

$$\partial(f + g)(x_0) \subset \partial f(x_0) + \partial g(x_0).$$

Proof. For the first part, let $\xi_1 \in \partial f(x_0)$ and $\xi_2 \in \partial g(x_0)$. Then, for all $x \in \mathbb{R}^n$

$$f(x) \geq f(x_0) + \xi_1^\top (x - x_0)$$
$$g(x) \geq g(x_0) + \xi_2^\top (x - x_0)$$

Adding the previous inequalities gives

$$f(x) + g(x) \geq f(x_0) + g(x_0) + (\xi_1 + \xi_2)^\top (x - x_0)$$

and hence $(\xi_1 + \xi_2) \in \partial(f + g)(x_0)$. For the second part, check Balder (2008). □

Using Fermat's rule and the previous theorem it is easy to see that the optimality condition for problem (2.2.10) is

$$0 \in \partial f(x) + \partial g(x).$$

The previous result does not assume that either f or g are differentiable functions. Most of the times in practice we find that one of them it is indeed differentiable, while the other is not. This can be further exploited to simplify the optimality condition. If the function f is differentiable and Lipschitz continuous, the new optimality condition is

$$0 \in \nabla f(x) + \partial g(x),$$

where we have used the third property of the subdifferential. The next theorem and proposition summarize the previous results.

Proposition 2.2. *Let f, g be proper convex functions and f differentiable. Then, the following are equivalent:*

(i) x is a solution of the problem (3.3.17)*.*

(ii) $0 \in \nabla f(x) + \partial g(x)$*.*

(iii) $x = \text{prox}_{\eta g}(x - \eta \nabla f(x))$*.*

Proof. If x is a solution of the problem (2.2.10) then, using Fermat's theorem (Theorem 2.5),

$$\begin{aligned} 0 \in \partial(f + g)(x) &\Leftrightarrow 0 \in \partial f(x) + \partial g(x) \\ &\Leftrightarrow 0 \in \{\nabla f(x)\} + \partial g(x) \\ &\Leftrightarrow -\nabla f(x) \in \partial g(x) \\ &\Leftrightarrow x - \eta \nabla f(x) \in x + \eta \partial g(x) \\ &\Leftrightarrow x = \text{prox}_{\eta g}(x - \eta \nabla f(x)). \end{aligned}$$

□

2.2.3 Algorithms

Item (iii) of Proposition 2.2 gives the idea of iterating

$$x^{k+1} = \text{prox}_{\eta g}(x^k - \eta \nabla f(x^k)), \tag{2.2.11}$$

starting from an initial point x^0. It is important to note that, on one hand, when $g = 0$, (2.2.11) reduces to the *gradient descent method*.

$$x^{k+1} = x^k - \eta \nabla f(x^k) \tag{2.2.12}$$

for minimizing a Lipschitz-differentiable function (Combettes and Pesquet, 2009). On the other hand, when $f = 0$, (2.2.11) reduces to the *proximal point algorithm*

$$x^{k+1} = \text{prox}_{\eta g}(x^k), \tag{2.2.13}$$

for minimizing a non-differentiable function (Combettes and Pesquet, 2009). The convergence of this algorithm is guaranteed for an appropriate choice of the parameter η. First we need to precisely define *smoothness*,

$$f(z) \leq f(x) + \nabla f(x)^\top (z - x) + \frac{L}{2}\|z - x\|_2^2, \tag{2.2.14}$$

for all $x, z \in \mathbb{R}^n$. The previous definition is equivalent to

$$\|\nabla f(x) - \nabla f(z)\| \leq L\|x - z\|. \tag{2.2.15}$$

This condition means that the gradient of the function f is Lipschitz continuous with constant L. We will sometimes refer to a function f satisfying the previous inequality as L-smooth. The following theorem states the convergence of the Proximal Gradient method:

Theorem 2.7 (Convergence of the proximal gradient method)**.** *If f has Lipschitz continuous gradient with constant L, $(f + g)$ admits minimizers and*

$$\eta \in \left(0, \frac{2}{L}\right),$$

then the proximal-gradient iterations $\{x^k\}$ converge to a minimizer.

The proof of Theorem 2.7 can be found in Bauschke and Combettes (2011, Corollary 27.9).

Chapter 3

Theory and algorithms for the Lasso

3.1 Lasso

The two standard techniques to improve Ordinary Least Squares (OLS) estimates, *subset selection* (Hastie et al., 2003) and *Ridge Regression*, both have disadvantages (Tibshirani, 1994). Subset selection obtains interpretable models but it can be very variable since it is a discrete process: features are either kept in the model or dropped from it. On the other hand, Ridge Regression is a continuous process, in the sense that all coefficients are shrunk (but not dropped) and hence more robust. However, it is very rare that coefficients become strictly 0 and therefore models are not interpretable.

Lasso (Least Absolute Shrinkage and Selection Operator) is a technique that reduces (shrinks) some coefficients and sets others to 0; therefore it tries to maintain the advantages of both subset selection and Ridge Regression. Let's assume, without loss of generality, that the x_{ij} are normalized to have zero mean and unit variance, that is, $\sum_i x_{ij}/n = 0$, $\sum_i x_{ij}^2/n = 1$ and the output variables y_i have zero mean. Let $\widehat{\mathbf{w}} = (\widehat{w}_1, \widehat{w}_2, \ldots, \widehat{w}_d)^\top$ be the Lasso estimate, defined by

$$\widehat{\mathbf{w}} = \operatorname{argmin} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_j w_j x_{ij} \right)^2 \right\} \quad \text{s.t} \quad \sum_j |w_j| \leq t, \tag{3.1.1}$$

where t is a tuning parameter. This parameter controls the amount of shrinking that is applied to the estimates. Let $\widehat{w}_j^0$ be the full OLS estimates and $t_0 = \sum_j |\widehat{w}_j^0|$. Values of $t < t_0$, will cause shrinking of the coefficients towards 0, and some coefficients may be strictly 0. For instance if $t = t_0/2$, the effect will be similar to finding the best subset of size $p/2$ (Tibshirani, 1994). On the other hand, if $t \geq t_0$, $\widehat{\mathbf{w}}^0$ is also the solution of the Lasso problem.

The motivation for the Lasso came from an interesting proposal of Breiman, the non-negative garotte (Breiman, 1995)

$$\min \sum_{i=1}^{n} \left(y_i - \sum_j c_j \widehat{w}_j^0 x_{ij} \right)^2 \quad \text{s.t} \quad c_j \geq 0, \quad \sum c_j \leq t. \tag{3.1.2}$$

The garotte starts with the full OLS estimates, $\widehat{w}_j^0$, and shrinks them using non-negative factors c_j whose sum is bounded. The $\widehat{w}_j(t) = c_j \widehat{w}_j^0$ are the new predictor coefficients. As t is decreased, more of the $\{c_j\}$ become 0, and the remaining non-zero $\widehat{w}_j(t)$ are shrunk.

Orthonormal case The orthonormal design case is a highly simplified situation (it is never the case in real-world datasets) where Lasso, subset selection, non-negative garrote and Ridge Regression solutions can be computed exactly. This can give interesting insight into the comparative behavior of those methods.

Recall that $\mathbf{X}$ is the $n \times d$ matrix with ij-th entry x_{ij} and suppose that $\mathbf{X}^\top \mathbf{X} = \mathbf{I}$. Let $\widehat{\mathbf{w}}^0$ be the ordinary least squares solution (Chapter 1),

$$\widehat{\mathbf{w}}^0 = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{3.1.3}$$

In the case of an orthogonal design, Eq. (3.1.3) simplifies to

$$\widehat{\mathbf{w}}^0 = \mathbf{X}^\top \mathbf{y}.$$

Using Lagrangian theory, it can be seen (Hastie et al., 2003) that the Lasso problem (3.1.1) is equivalent to

$$\widehat{\mathbf{w}} = \operatorname{argmin} \left\{ \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 \right\}, \tag{3.1.4}$$

where

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$$

and $\lambda \geq 0$. We will prove this equivalence later in Chapter 5. Note that the Lasso penalty is convex, but not strictly convex (refer to Chapter 1 for the definitions). Expanding the first term of the previous equation and using the fact that $\mathbf{X}$ is orthonormal, we get $\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^t \mathbf{X}\mathbf{w} + \mathbf{w}^\top \mathbf{w}$. Since $\mathbf{y}^\top \mathbf{y}$ does not contain any of the variables to minimize and noting that $\widehat{\mathbf{w}}^0 = \mathbf{X}^\top \mathbf{y}$, we can rewrite the problem (3.1.4) as

$$\begin{aligned} \widehat{\mathbf{w}} &= \operatorname{argmin} \left\{ 2\widehat{\mathbf{w}}^0 \mathbf{w} + \|\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 \right\} \\ &= \operatorname{argmin} \left\{ \sum_j 2w_j^0 w_j + w_j^2 + \lambda |w_j| \right\}, \end{aligned} \tag{3.1.5}$$

where $(\cdot)_+ = \max(\cdot, 0)$ denotes the positive part.

The objective function is now a sum of objectives, each corresponding to a separate variable w_j, so they may be solved individually. Fixing a certain j, then we want to minimize

$$2w_j^0 w_j + w_j^2 + \lambda |w_j|.$$

Note that if $w_j^0 > 0$ then $w_j \geq 0$ since otherwise we could flip the sign and get a lower value for the objective function. Similarly, if $w_j^0 < 0$ then $w_j \leq 0$. For the first case, differentiating with respect to w_j and setting equal to 0 we get

$$w_j = w_j^0 - \frac{\lambda}{2}.$$

But this is only positive if the right-hand side is non-negative, that is, we have to take

$$\widehat{w}_j = \left(\widehat{w}_j^0 - \frac{\lambda}{2} \right)_+ = \operatorname{sign}(\widehat{w}_j^0) \left(|\widehat{w}_j^0| - \frac{\lambda}{2} \right)_+. \tag{3.1.6}$$

Using a similar argument the same solution is obtained for the $w_j^0 < 0$ case. Finally, letting $\gamma = \frac{\lambda}{2}$ we get that the Lasso solutions for the orthonormal case are

$$\widehat{w}_j = \operatorname{sign}(\widehat{w}_j^0)(|\widehat{w}_j^0| - \gamma)_+, \tag{3.1.7}$$

where γ is determined by the condition $\sum \widehat{w}_j = t$. In fact, for every $t \geq 0$ it is possible to find a $\gamma \geq 0$ for which the solution of the problem is the same (Chapter 5).

In this scenario, the best subset selection of size k reduces to choosing the k largest coefficients in absolute value and setting the rest to 0. This is equivalent to

$$\widehat{w}_j = \begin{cases} \widehat{w}_j^0 & \text{if } |\widehat{w}_j^0| > \lambda \\ 0 & \text{otherwise} \end{cases}.$$

for some choice of λ (Tibshirani, 1994). Recall from Chapter 1 that Ridge Regression solutions are

$$\widehat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}.$$

If $\mathbf{X}^\top \mathbf{X} = \mathbf{I}$ and noting again that $\widehat{\mathbf{w}}^0 = \mathbf{X}^\top \mathbf{y}$, the solutions for the orthonormal case can be rewritten as

$$\widehat{w}_j = \frac{1}{1+\lambda} \widehat{w}_j^0.$$

Last, the solutions for the garotte are (Breiman, 1995)

$$\widehat{w}_j = \left(1 - \frac{\lambda^2}{(\widehat{w}_j^0)^2}\right)_+ \widehat{w}_j^0.$$

Figure 3.1 shows the comparison between Lasso, Ridge Regression, garotte and best subset selection solutions as a function of the OLS estimates. As it can be seen the Lasso often obtains some coefficients equal to 0, while shrinking the rest towards 0. Best subset selection also obtains coefficients equal to 0, but the rest remain unchanged. The garotte is similar to the Lasso, with less shrinkage for larger coefficients. According to Tibshirani (1994), the differences can be large if the design is not orthonormal. Finally, Rigde Regression shrinks all coefficients, but none of them is strictly 0.

Geometry Now let's look at the Lasso penalty from a geometrical point of view. Figure 3.2 shows the elliptical contours of the objective function, centered at the OLS estimates. The Lasso solution is the first place where these estimates intersect with the unit square. Sometimes this will happen at a corner and thus the coefficient associated with that axis will be 0. For example in Fig. 3.2 the optimal value for the coefficient associated to the x-axis is 0. In the Ridge Regression case, the constrain region is a circle and hence zero solutions will rarely result. As it can be deduced from the figure, Ridge Regression contour plots intersect with the circle in a point $(0, y)$ only if the corresponding OLS estimate is already in the y-axis, i.e. the coefficient is already 0, which is a much more unlikely event.

3.2 Elastic Net

We consider the usual regression model (2.1.1) with d features. As shown before there are several options to find the weight vector $\widehat{\mathbf{w}}$, such as OLS, Ridge Regression or the Lasso. It is well known that OLS estimates often do poorly in both prediction and interpretation. As a continuous shrinkage method, Ridge Regression achieves better prediction performance, although it does not produce sparse and, hence, interpretable models.

The Lasso fixes both problems, since it does simultaneously continuous shrinkage and automatic variable selection. Tibshirani (1994) and Fu (1998) compare the prediction performance of the Lasso and Ridge Regression and found that none uniformly dominates over the other.

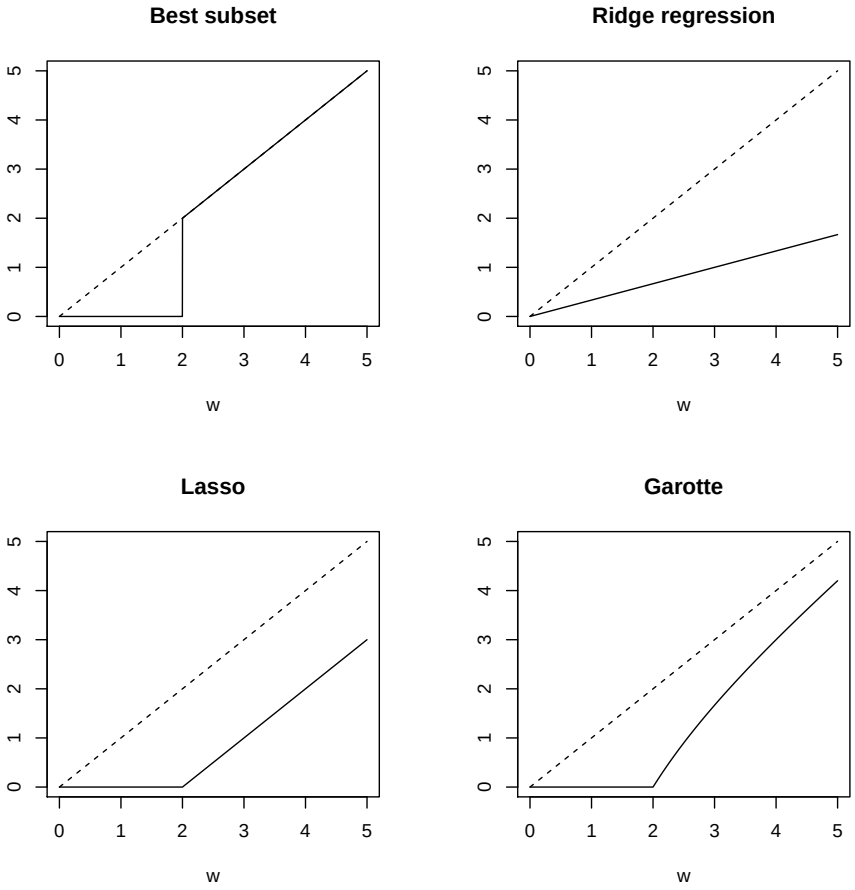

Figure 3.1: Subset regression, Ridge Regression, Lasso and garotte shrinkage comparison in the case of an orthonormal design.

However, as data is rapidly increasing in both number and dimension, Lasso is more appealing due to its sparse representation. Although Lasso has shown success in many situations, it has some limitations:

1. In the $d > n$ case, the Lasso selects at most n variables, because of the nature of the convex optimization problem (Efron et al., 2004). From a variable selection point of view, this seems like a limiting feature. However, from a regression point of view this is not a bad property since the problem is ill-posed.

2. If there is a group of variables for which pairwise correlations are very high, then the Lasso tends to select somewhat randomly only one variable from the group (Grave et al., 2011).

3. In the $n > d$ case, if there are high correlations between variables, it has been empirically observed that the Ridge Regression prediction is better than the Lasso prediction.

In addition, for some problems such as the gene selection problem in microarray data, the Lasso may not be a good variable selection method (Zou and Hastie, 2005). Typical micro-array data has many thousands of variables (genes) and often fewer than 100 samples. There are also some genes with high correlations between them, forming a group. The ideal gene selection

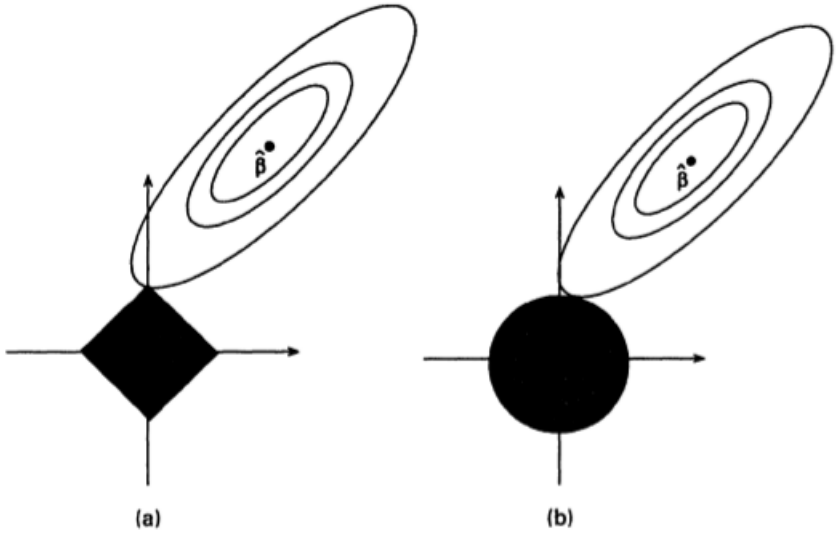

Figure 3.2: Lasso (a) and Ridge Regression (b) estimates (Tibshirani, 1994)

method should be able to eliminate the trivial genes and automatically include whole groups into the model. For this kind of $d \gg n$ and grouped variables situation, the Lasso is not the ideal method, because it can only select at most n variables out of d candidates and it lacks the ability to reveal grouping information.

The Elastic Net is a method that tries to improve on both Lasso and Ridge Regression as it simultaneously does automatic selection and continuous shrinkage like the Lasso, but it can also select groups of correlated variables. It also lacks the limitation of selecting at most n features out of d when $d > n$.

3.2.1 Naïve Elastic Net

Suppose that the dataset has n observations and d features. Let $\mathbf{y} \in \mathbb{R}^n$ be the response vector and $\mathbf{X} \in \mathbb{R}^{n \times d}$ the data matrix. As in the Lasso, we assume that the response has zero mean and the variables are standarized. The naïve Elastic Net criterion is defined as

$$L(\mathbf{w}, \lambda_1, \lambda_2) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda_2 \|\mathbf{w}\|_2^2 + \lambda_1 \|\mathbf{w}\|_1. \tag{3.2.1}$$

The naïve Elastic Net estimator $\widehat{\mathbf{w}}$ is the minimizer of Eq. (3.2.1):

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \{L(\mathbf{w}, \lambda_1, \lambda_2)\}. \tag{3.2.2}$$

Let $\alpha = \lambda_2/(\lambda_1 + \lambda_2)$, then Eq. (3.2.2) is equivalent to (Zou and Hastie, 2005)

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2, \quad \text{s.t.} \quad (1-\alpha)\|\mathbf{w}\|_1 + \alpha\|\mathbf{w}\|_2^2 \leq t. \tag{3.2.3}$$

The function $(1-\alpha)\|\mathbf{w}\|_1 + \alpha\|\mathbf{w}\|_2^2$ is the Elastic Net penalty, which is a convex combination of the Lasso penalty (ℓ_1-norm) and the Ridge Regression penalty (ℓ_2-norm). When $\alpha = 1$, the naïve Elastic Net becomes simply Ridge Regression. For all $\alpha \in (0, 1)$ the Elastic Net penalty is strictly convex. Recall that the Lasso penalty ($\alpha = 0$) is convex but not strictly convex and non-differentiable. See Fig. 3.3 for a graphical representation of the Elastic Net penalty.

We discuss next how to solve the Elastic Net problem. It turns out that minimizing Eq. (3.2.1) is equivalent to a Lasso-type optimization problem.

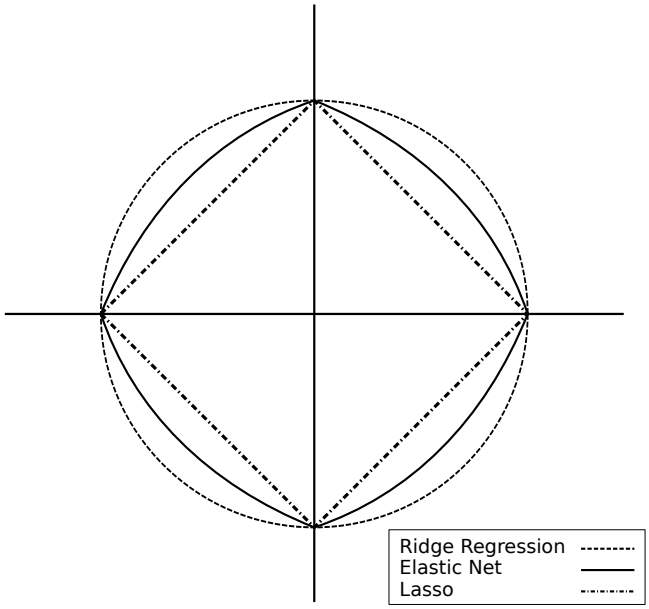

Figure 3.3: Lasso, Ridge Regression and Elastic Net penalties for the two variable case

Lemma 3.1. *Given a dataset* $(\mathbf{X}, \mathbf{y})$ *and* (λ_1, λ_2)*, define an artificial dataset* $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{y}})$ *by*

$$\widetilde{\mathbf{X}}_{(n+d)\times d} = (1+\lambda_2)^{-1/2} \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda_2}\mathbf{I}_d \end{pmatrix}, \qquad \widetilde{\mathbf{y}}_{(n+p)} = \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix}. \tag{3.2.4}$$

Let $\gamma = \lambda_1/\sqrt{(1+\lambda_2)}$ *and* $\widetilde{\mathbf{w}} = \sqrt{(1+\lambda_2)}\mathbf{w}$*. Then the naïve Elastic Net criterion can be written as*

$$L(\mathbf{w}, \gamma) = L(\widetilde{\mathbf{w}}, \gamma) = \left\| \widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\widetilde{\mathbf{w}} \right\|_2^2 + \gamma \|\widetilde{\mathbf{w}}\|_1. \tag{3.2.5}$$

Let also

$$\widetilde{\mathbf{w}}^* = \underset{\widetilde{\mathbf{w}}}{\operatorname{argmin}}\{L(\widetilde{\mathbf{w}}, \gamma)\};$$

then the Elastic Net optimum $\mathbf{w}(Naive)$ *is given by*

$$\mathbf{w}(Naive) = \frac{1}{\sqrt{(1+\lambda_2)}}\widetilde{\mathbf{w}}^*. \tag{3.2.6}$$

Proof. We are going to show how to recover the criterion function (3.2.1) from the function (3.2.5). First, substituting the values of γ and $\widetilde{\mathbf{w}}$ in Eq. (3.2.5) we get

$$\begin{aligned} L(\widetilde{\mathbf{w}}, \gamma) &= \left\| \widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\sqrt{(1+\lambda_2)}\mathbf{w} \right\|_2^2 + \frac{\lambda_1}{\sqrt{(1+\lambda_2)}} \left\| \sqrt{(1+\lambda_2)}\mathbf{w} \right\|_1 \\ &= \left\| \widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\sqrt{(1+\lambda_2)}\mathbf{w} \right\|_2^2 + \lambda_1 \|\mathbf{w}\|_1. \end{aligned}$$

Now substituting the values of $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{y}}$ into the previous equation we get

$$\begin{aligned} L(\mathbf{w}, \lambda_1, \lambda_2) &= \left\| \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix} - (1+\lambda_2)^{-1/2} \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda_2}\mathbf{I}_p \end{pmatrix} \sqrt{(1+\lambda_2)}\mathbf{w} \right\|_2^2 + \lambda_1 \|\mathbf{w}\|_1 \\ &= \left\| \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix} - \begin{pmatrix} \mathbf{X}\mathbf{w} \\ \sqrt{\lambda_2}\mathbf{w} \end{pmatrix} \right\|_2^2 + \lambda_1 \|\mathbf{w}\|_1 \\ &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \| - \lambda_2^{1/2}\mathbf{w}\|_2^2 + \lambda_1 \|\mathbf{w}\|_1 \\ &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda_2 \|\mathbf{w}\|_2^2 + \lambda_1 \|\mathbf{w}\|_1, \end{aligned}$$

which is exactly the same as Eq. (3.2.1) □

Lemma 3.1 shows that we can transform the naïve Elastic Net problem into a equivalent Lasso problem on augmented data. Note that the sample size in the augmented problem is $n + d$ and $\widetilde{\mathbf{X}}$ has rank d, which means that the naïve Elastic Net can potentially select all d features in all situations. This property overcomes the limitation of the Lasso that were described in Section 3.1. Lemma 3.1 also shows that naïve Elastic Net can perform an automatic variable selection in a similar fashion to the Lasso.

Let's briefly discuss now the grouping effect in the Elastic Net model. In the $d \gg n$ case, the grouped variables situation is particularly important, for the reasons mentioned before. Qualitatively speaking, a regression method exhibits the grouping effect if the regression coefficients of a group of highly correlated variables tend to be equal in absolute value. In particular, in the extreme situation where some variables are exactly identical, the regression method should assign identical coefficients to the identical variables. In order to quantify the grouping effect, Zou and Hastie (2005) consider the following generic penalization problem

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \{ \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda J(\mathbf{w}) \}, \tag{3.2.7}$$

where $J(\cdot)$ is positive valued for $\mathbf{w} \neq 0$. The following lemma from Zou and Hastie (2005) characterizes the grouping effect depending on the penalty term $J(\cdot)$.

Lemma 3.2. *Assume that* $\mathbf{x}_i = \mathbf{x}_j$, $i \neq j \in \{1, \ldots, p\}$.

1. *If* $J(\cdot)$ *strictly convex, then* $\widehat{w}_i = \widehat{w}_j$, $\forall \lambda > 0$.

2. *If* $J(\mathbf{w}) = \|\mathbf{w}\|_1$, *then* $\widehat{w}_i \widehat{w}_j \geq 0$ *and for any* $s \in [0, 1]$, *the weights* $\widehat{\mathbf{w}}^*(s)$

$$\widehat{w}_k^*(s) = \begin{cases} \widehat{w}_k & \text{if } k \neq i \text{ and } k \neq j, \\ (\widehat{w}_i + \widehat{w}_j)\ s & \text{if } k = i, \\ (\widehat{w}_i + \widehat{w}_j)(1 - s) & \text{if } k = j, \end{cases}$$

are also minimizers of Eq. (3.2.7).

Lemma 3.2 shows a clear distinction between strictly convex penalty functions and the Lasso penalty. Strict convexity guarantees the grouping effect, i.e., equal coefficients in the extreme situation with identical variables. In contrast, the Lasso does not even have an unique solution. The Elastic Net penalty is strictly convex for $\lambda_2 > 0$, thus enjoying property 1 of Lemma 3.2. The following Theorem quantifies the grouping effect of the Elastic Net.

Theorem 3.3 (Theorem 1 in Zou and Hastie, 2005)**.** *Given data* $(\mathbf{X}, \mathbf{y})$ *and parameters* (λ_1, λ_2)*, assume* $\mathbf{y}$ *has zero mean and the variables* $\mathbf{X}$ *are standarized. Let* $\widehat{\mathbf{w}}(\lambda_1, \lambda_2)$ *be the naïve Elastic Net estimate. Suppose that* $\widehat{w}_i(\lambda_1, \lambda_2)\widehat{w}_j(\lambda_1, \lambda_2) > 0$ *and define*

$$D_{\lambda_1,\lambda_2}(i,j) = \frac{1}{\|\mathbf{y}\|_1}|\widehat{w}_i(\lambda_1, \lambda_2) - \widehat{w}_j(\lambda_1, \lambda_2)|;$$

then

$$D_{\lambda_1,\lambda_2}(i,j) \leq \frac{1}{\lambda_2}\sqrt{2(1-\rho_{ij})},$$

where $\rho_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$ *is the* $\mathbf{x}_i, \mathbf{x}_j$ *sample correlation.*

Proofs of Lemma 3.2 and Theorem 3.3 can be found in Zou and Hastie (2005, Appendix A). The quantity $D_{\lambda_1,\lambda_2}(i,j)$ describes the difference between the coefficient paths of variables i and j. If $\mathbf{x}_i$ and $\mathbf{x}_j$ are highly correlated, Theorem 3.3 says that the difference between the coefficient paths of predictor i and predictor j is almost 0, that is, the final weights associated to those variables are very similar.

3.2.2 General Elastic Net

As an automatic variable selection model, the naïve Elastic Net overcomes the limitations of the Lasso in scenarios 1 and 2 (Section 3.2). However, empirical evidence (see Zou and Hastie, 2005, Sections 4 and 5) shows that Elastic Net may not perform well unless it is very close to either Ridge Regression or the Lasso.

As we have mentioned before in Chapter 1, an accurate penalization method achieves good prediction performance through the bias-variance tradeoff. The naïve Elastic Net estimator is a two stage procedure: for each fixed λ_2 first the Ridge Regression coefficients are found and then a Lasso-type shrinkage is done along the Lasso coefficient solution paths. This suggests that a double amount of shrinkage is being performed in the coefficients. This double shrinkage does not help to reduce the variance much and introduces unnecessary extra bias.

In order to improve the prediction performance of the naïve Elastic Net estimate this double shrinkage has to be corrected. Given data $(\mathbf{X}, \mathbf{y})$, penalty parameters (λ_1, λ_2), and augmented data $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{y}})$, the naïve Elastic Net solves the Lasso-type problem

$$\widetilde{\mathbf{w}}^* = \underset{\widetilde{\mathbf{w}}}{\operatorname{argmin}} \left\{ \left\| \widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\widetilde{\mathbf{w}} \right\|_2^2 + \frac{\lambda_1}{\sqrt{(1+\lambda_2)}} \|\widetilde{\mathbf{w}}\|_1 \right\} \tag{3.2.8}$$

The corrected Elastic Net estimates are defined by Zou and Hastie (2005) as

$$\mathbf{w}(\text{ENet}) = \sqrt{(1+\lambda_2)}\widetilde{\mathbf{w}}^*. \tag{3.2.9}$$

We had before $\mathbf{w}(\text{Naive}) = (1/\sqrt{(1+\lambda_2)})\widetilde{\mathbf{w}}^*$, therefore it follows that

$$\mathbf{w}(\text{ENet}) = (1+\lambda_2)\mathbf{w}(\text{Naive}). \tag{3.2.10}$$

Hence the Elastic Net coefficients are a rescaled version of the naïve Elastic Net coefficients. Such an scaling transformation preserves the variable selection property of the naïve Elastic Net and it is the simplest way to undo the double shrinkage. Empirically it was also found by Zou and Hastie (2005) that the Elastic Net performs very well when compared with the Lasso and Ridge Regression.

3.3 Algorithms

The Lasso problem (3.1.4) is a convex quadratic unconstrained optimization problem that, in principle, could be solved using standard optimization techniques such as gradient descent. However, there are two main difficulties when working with the ℓ_1-norm:

1. It it not differentiable. Hence, the gradient must be replaced by its generalization, the subgradient. In practice it does not make a big difference since the non-differentiability only occurs at zero, but it makes the analysis a little bit more involved.

2. It is not strongly convex. Many theoretical convergence rates enjoy a linear convergence **only** for strongly convex functions and thus they do not hold for the Lasso.

Despite everything mentioned above, the primal problem is usually the formulation of choice when solving the Lasso and many algorithms exist in the literature. In the following sections we will review the main ones, starting with LARS, which was the first efficient procedure to solve the Lasso, and moving on to more modern algorithms. LARS is an important one since it was the first procedure specifically tailored for this problem and before that only standard Quadratic Programming (QP) techniques were used.

More modern algorithms are usually concerned with scalability, and thus batch methods were slowly replaced by online-style algorithms, where the samples are all available but still streamed to the algorithm in mini-batches, in some pre-defined or random order. In the extreme case these mini-batches only contain one sample. This idea not only applies to the samples but also to the coordinates or variables, and we will also explore algorithms where the optimization occurs one coordinate at a time. The name *stochastic algorithms* is used when the samples or coordinates are drawn at random following some distribution, usually uniformly.

It is important to note that these algorithms usually tackle the more general problem of differentiable loss plus a convex regularizer (not necessarily differentiable) since the analysis barely changes. We will only cover here the Lasso case although they can usually be easily extended to cover any differentiable loss and one or more differentiable regularizers, for example the Elastic Net problem.

As mentioned before most of the procedures solve the unconstrained primal formulation, although some works also explore the constrained primal formulation (3.1.1), by using projections, or the dual formulation. In particular, the latter is often use to derive screening rules to discard variables early during the optimization algorithm, which we will briefly discuss in Section 3.4.

3.3.1 Least Angle Regression

Least Angle Regression (LARS) (Efron et al., 2004) was the first algorithmic procedure to efficiently solve the Lasso and related problems. More specifically, LARS is a model selection algorithm also related to others such as Forward Stagewise regression and Forward Stepwise regression. In fact, simple modifications of the LARS procedure implement algorithms to solve these problems using less computer time.

One of the first model selection method was Forward Stepwise regression (Roush, 1982). Given a collection of possible features, we select the one having largest absolute correlation with the response $\mathbf{y}$, say $\mathbf{x}_1$, and perform a linear regression of $\mathbf{y}$ on $\mathbf{x}_1$. This leaves a residual vector orthogonal to $\mathbf{x}_1$ that is now considered to be the response. We project the other variables orthogonally to $\mathbf{x}_1$ and repeat the selection process. After k steps we have a set of variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ that can be used to construct a linear model with k parameters. This procedure

may be overly greedy since useful predictors can be discarded too early if they are very correlated to previously selected $\mathbf{x}_i$ variables.

Next we describe Forward Stagewise regression (Efron et al., 2004), which is a much more cautious version of Forward Stepwise regression, since it makes a lot of tiny steps (more than k) as it moves towards a final model. Forward Stagewise starts with $\widehat{\mathbf{w}} = 0$ and builds up the regression function in successive small steps. If $\widehat{\mathbf{w}}$ is the current estimated weight vector and $\widehat{\mathbf{y}} = \mathbf{X}\widehat{\mathbf{w}}$, let $c(\widehat{\mathbf{y}})$ be the vector of current correlations, i.e.,

$$\widehat{\mathbf{c}} = \mathbf{c}(\widehat{\mathbf{y}}) = \mathbf{X}^\top(\mathbf{y} - \widehat{\mathbf{y}}), \tag{3.3.1}$$

so that its j-th component $\widehat{c}_j$ is proportional to the correlation between covariate x_j and the current residual vector. The next step of the algorithm is taken in the direction of the largest current correlation j, i.e., we first find j^* as

$$j^* = \operatorname{argmax} |\widehat{c}_j|$$

and then we update the estimate

$$\widehat{\mathbf{y}} = \widehat{\mathbf{y}} + \varepsilon \operatorname{sign}(\widehat{y}_{j^*})\mathbf{x}_{j^*}, \tag{3.3.2}$$

with $\operatorname{sign}(\cdot)$ being the ± 1-valued sign function and ε some small constant. Note that if $\varepsilon = |\widehat{c}_j|$ this reduces to the Forward Stepwise algorithm and the number of iterations is equal to k, the number of parameters in the final model (Efron et al., 2004). As ε approaches 0, the algorithm is going to increase the number of iterations (taking more computer time), but possibly also its accuracy, since it will not discard early useful variables very correlated with the response.

LARS is an intermediate approach (Efron et al., 2004), since a simple formula allows to implement Forward Stagewise with fairly large steps, although not as large as those in Forward Stepwise, reducing the computational burden. LARS builds up the estimates $\widehat{\mathbf{y}} = \mathbf{X}\widehat{\mathbf{w}}$ in k steps, adding one covariate to the model in each step, so at the end only k of the $\widehat{w}_j$ are non-zero.

First we start with $\widehat{\mathbf{w}} = 0$ and find the predictor most correlated with the response using Eqs. (3.3.1) and (3.3.2). We take the largest step possible in that direction until some other predictor has as much correlation with the current residual. At this point, LARS proceeds in a direction equiangular to the current variables until a third variable earns its way into the most correlated set, and so on.

More formally, we begin at $\widehat{\mathbf{w}} = 0$, that is, $\widehat{\mathbf{y}}_0 = 0$, and suppose that $\widehat{\mathbf{y}}_{\mathcal{A}}$ is the current LARS output estimate. Then, the vector of current residual correlations is given by Eq. (3.3.1)

$$\widehat{\mathbf{c}} = \mathbf{X}^\top(\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{A}}).$$

The *active set* $\mathcal{A}$ is the set of indices corresponding to covariates with the greatest absolute current correlations,

$$\widehat{C} = \max_j\{|\widehat{c}_j|\} \quad \text{and} \quad \mathcal{A} = \Big\{ j \;\Big|\; |\widehat{c}_j| = \widehat{C} \Big\}. \tag{3.3.3}$$

Assuming that the feature vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$ are linearly independent, we define the $n \times |\mathcal{A}|$-matrix

$$\mathbf{X}_{\mathcal{A}} = (\ldots \operatorname{sign}(\widehat{c}_j)\mathbf{x}_j \ldots)_{j\in\mathcal{A}}. \tag{3.3.4}$$

Let

$$\mathbf{G}_{\mathcal{A}} = \mathbf{X}_{\mathcal{A}}^\top \mathbf{X}_{\mathcal{A}} \quad \text{and} \quad A_{\mathcal{A}} = (\mathbf{1}_{\mathcal{A}}^\top \mathbf{G}_{\mathcal{A}}^{-1} \mathbf{1}_{\mathcal{A}})^{-1/2} \tag{3.3.5}$$

with $\mathbf{1}_{\mathcal{A}}$ the vector of 1's of lenght $|\mathcal{A}|$. Then the equiangular vector

$$\mathbf{u}_{\mathcal{A}} = \mathbf{X}_{\mathcal{A}} \boldsymbol{\omega}_{\mathcal{A}}, \tag{3.3.6}$$

where

$$\boldsymbol{\omega}_{\mathcal{A}} = A_{\mathcal{A}} \mathbf{G}_{\mathcal{A}}^{-1} \mathbf{1}_{\mathcal{A}}, \tag{3.3.7}$$

is the unit-norm vector making equal angles, less than 90°, with the columns of $\mathbf{X}_{\mathcal{A}}$, that is,

$$X_{\mathcal{A}}^{\top} \mathbf{u}_{\mathcal{A}} = A_{\mathcal{A}} \mathbf{1}_{\mathcal{A}} \quad \text{and} \quad ||\mathbf{u}_{\mathcal{A}}||^2 = 1. \tag{3.3.8}$$

We also compute the vector of length $|\mathcal{A}|$

$$\mathbf{a} = \mathbf{X}^{\top} \mathbf{u}_{\mathcal{A}}. \tag{3.3.9}$$

Then the next step of the LARS algorithm selects first the new covariate j^* to be added as

$$j^* = \underset{j \in \mathcal{A}^c}{\text{argmin}^+} \left\{ \frac{\widehat{C} - \widehat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\widehat{C} + \widehat{c}_j}{A_{\mathcal{A}} + a_j} \right\} = \underset{j \in \mathcal{A}^c}{\text{argmin}^+} \left\{ \gamma_j \right\}, \tag{3.3.10}$$

where by argmin^+ we indicate that the minimum is taken only over positive components. Then $\widehat{\mathbf{y}}_{\mathcal{A}}$ is updated as

$$\widehat{\mathbf{y}}_{\mathcal{A}^+} = \widehat{\mathbf{y}}_{\mathcal{A}} + \widehat{\gamma} \mathbf{u}_{\mathcal{A}}, \tag{3.3.11}$$

where $\widehat{\gamma} = \gamma_{j^*}$, i.e.,

$$\widehat{\gamma} = \min_{j \in \mathcal{A}^c}{}^+ \left\{ \frac{\widehat{C} - \widehat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\widehat{C} + \widehat{c}_j}{A_{\mathcal{A}} + a_j} \right\}. \tag{3.3.12}$$

Here $\min^+$ indicates again that the minimum is taken only over positive components for each choice of j.

We discuss next the rationale for the above choices (see Efron et al., 2004 for more details). Define

$$\mathbf{y}(\gamma) = \widehat{\mathbf{y}}_{\mathcal{A}} + \gamma \mathbf{u}_{\mathcal{A}} \tag{3.3.13}$$

for $\gamma > 0$, so that the new residual correlation is

$$\begin{aligned} c_j(\gamma) &= \mathbf{x}_j^{\top} (\mathbf{y} - \mathbf{y}(\gamma)) \\ &= \mathbf{x}_j^{\top} (\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{A}} - \gamma \mathbf{u}_{\mathcal{A}}) \\ &= \mathbf{x}_j^{\top} (\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{A}}) - \gamma \mathbf{x}_j^{\top} \mathbf{u}_{\mathcal{A}} \\ &= \widehat{c}_j - \gamma a_j. \end{aligned} \tag{3.3.14}$$

For $j \in \mathcal{A}$, Eqs. (3.3.1), (3.3.3) and (3.3.8) yield

$$|c_j(\gamma)| = \widehat{C} - \gamma A_{\mathcal{A}}, \tag{3.3.15}$$

showing that all of the maximal absolute correlations of the features already selected will decline equally for any value of γ. For $j \in \mathcal{A}^c$, γ is selected so that the next covariate to join the active set has a new correlation exactly equal to Eq. (3.3.15). Equating (3.3.14) with (3.3.15) shows that $c_j(\gamma)$ reaches its maximum value at $\gamma = (\widehat{C} - \widehat{c}_j)/(A_{\mathcal{A}} - a_j)$. Likewise $-c_j(\gamma)$ achieves its maximum at $\gamma = (\widehat{C} + \widehat{c}_j)/(A_{\mathcal{A}} + a_j)$. Therefore we want to choose $\widehat{\gamma}$ in Eq. (3.3.12) as the smallest possible such value of γ. In other words, we select the next covariate j to join the active set, i.e., $\mathcal{A}_+ = \mathcal{A} \cup \{j^*\}$, as

$$j^* = \underset{j \in \mathcal{A}^c}{\text{argmin}^+} \left\{ \frac{\widehat{C} - \widehat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\widehat{C} + \widehat{c}_j}{A_{\mathcal{A}} + a_j} \right\}.$$

The new maximum absolute correlation $\widehat{C}_+$ verifies then $\widehat{C}_+ = \widehat{C} - \widehat{\gamma} A_{\mathcal{A}}$.

It is important to note that the procedure described previously does not yield solutions for the Lasso problem, but they are closely related. In fact, Efron et al. (2004) show how the LARS algorithm can be slightly modified to generate the full set of Lasso solutions.

Let $\widehat{\mathbf{w}}$ be a Lasso solution with $\widehat{\mathbf{y}} = \mathbf{X}\widehat{\mathbf{w}}$; then the sign of any non-zero coordinate $\widehat{w}_j$ must agree with the sign s_j of the current correlation $\widehat{c}_j = \mathbf{x}_j^\top(\mathbf{y} - \widehat{\mathbf{y}})$ (Efron et al., 2004),

$$\operatorname{sign}(\widehat{w}_j) = \operatorname{sign}(\widehat{c}_j) = s_j. \tag{3.3.16}$$

This restriction is not enforced in the LARS procedure, but it can be easily modified to do so.

Assume we have just finished a LARS step giving a new active set as in Eq. (3.3.3), and the corresponding LARS estimate $\widehat{\mathbf{y}}_{\mathcal{A}}$ corresponds to a Lasso solution $\widehat{\mathbf{y}} = \mathbf{X}\widehat{\mathbf{w}}$. Define the d-vector $\mathbf{d}$ whose components are

$$d_j = \begin{cases} s_j \boldsymbol{\omega}_{\mathcal{A}} & \text{if } j \in \mathcal{A} \\ 0 & \text{else} \end{cases}$$

with $\boldsymbol{\omega}_{\mathcal{A}}$ as in Eq. (3.3.7). Moving in the positive γ direction of the LARS line (Eq. 3.3.13), we see that $\mathbf{y}(\gamma) = \mathbf{X}\mathbf{w}(\gamma)$, where

$$w_j(\gamma) = \widehat{w}_j + \gamma d_j.$$

Therefore $w_j(\gamma)$ will change sign at

$$\gamma_j = \frac{-\widehat{w}_j}{d_j}.$$

Let $\tilde{\gamma}$ be the first of such γ_j; then if $\tilde{\gamma} < \widehat{\gamma}$ (Eq. (3.3.12)), $w_j(\gamma)$ cannot be a Lasso solution for $\gamma > \tilde{\gamma}$ since the sign restriction (Eq. (3.3.16)) must be violated: $\widehat{w}_j(\gamma)$ has changed sign while $\widehat{c}_j$ has not (Eq. (3.3.15) is always positive within a single LARS step).

Summing up, we can modify the LARS procedure to check whether $\tilde{\gamma} < \widehat{\gamma}$ at every iteration. If that is the case, we stop the ongoing LARS step at $\gamma = \tilde{\gamma}$ and remove the current covariate $\tilde{j}$ from the calculation of the next equiangular direction, that is, we replace Eq. (3.3.11) by

$$\widehat{\mathbf{y}}_{\mathcal{A}^+} = \widehat{\mathbf{y}}_{\mathcal{A}} + \widehat{\gamma}\mathbf{u}_{\mathcal{A}} \quad \text{and} \quad \mathcal{A}^+ = \mathcal{A} - \left\{\tilde{j}\right\}.$$

Theorem 1 of Efron et al. (2004) states that, under this modification, the LARS procedure yields all the Lasso solutions. It is important to note that now covariates in the active set can be removed by the algorithm. These covariates may later re-enter the active set or be completely removed from the final solution. As a result, the maximum number of steps is no longer upper bounded by the number of covariates d, as it was the case in the original LARS procedure. Note also that Theorem 1 assumes that these increments or decrements of the active set only involve one index j at a time, that is, in Eq. (3.3.10) there are not two indexes with the same γ value. Efron et al. (2004, Section 5) discuss what to do when such ties occur, although they are very rare in real-world data.

3.3.2 Proximal Gradient Descent

As we have seen in Chapter 1, Ridge Regression can be solved analytically by computing the inverse of $(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}^\top\mathbf{I})$, where $\mathbf{I}$ is the identity matrix. However, in practice this operation can be very expensive and numerically unstable. Thus, other alternatives are usually preferred. One option is to use gradient descent to iteratively find the optimum of the function by taking small steps in the direction of the negative gradient. For the Lasso, since the ℓ_1-norm is not differentiable at 0, we have to use a generalization of gradient descent described in Section 2.2.3, Proximal

Gradient Descent. This algorithm is also known as *forward-backward splitting* (Combettes and Pesquet, 2009).

Given the standard optimization problem,

$$\min_{x\in\mathbb{E}} \; f(\mathbf{x}) + g(\mathbf{x}), \tag{3.3.17}$$

where f has Lipschitz continuous gradient with constant L, Proximal Gradient Descent iterates (Section 2.2.3)

$$\mathbf{x}^{k+1} = \text{prox}_{\eta g}(\mathbf{x}^k - \eta\nabla f(\mathbf{x}^k)). \tag{3.3.18}$$

Recall that the Lasso can be stated as the following optimization problem:

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1. \tag{3.3.19}$$

As it can be seen, it takes the general form of Eq. (3.3.17), where $f(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ and $g(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$. The gradient of f is

$$\nabla f(\mathbf{w}) = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}), \tag{3.3.20}$$

and the proximal mapping of the ℓ_1-norm is the soft-thresholding operator (Appendix A).

$$[\text{prox}_g(\mathbf{w})]_i = [\text{S}_\lambda(\mathbf{w})]_i = \text{sign}(w_i)(|w_i| - \lambda)_+. \tag{3.3.21}$$

Substituting Eqs. (3.3.20) and (3.3.21) in Eq. (3.3.18) we get:

$$\mathbf{w}^{k+1} = S_\lambda(\mathbf{w}^k - 2\eta\mathbf{X}^\top(\mathbf{X}\mathbf{w}^k - \mathbf{y})), \tag{3.3.22}$$

where η is an appropriate stepsize. If we iterate Eq. (3.3.22) until convergence we get an algorithm usually referred in the literature as Iterative Soft-Thresholding Algorithm or simply ISTA. The name comes from the fact that we are applying the proximal operator of the ℓ_1-norm or soft-thresholding operator after each gradient step.

Now let's see an alternative, more general, derivation of the algorithm from (Beck and Teboulle, 2009a). Consider the general formulation:

$$\min\{F(\mathbf{w}) \equiv f(\mathbf{w}) + g(\mathbf{w}) : \mathbf{w} \in \mathbb{R}^n\}. \tag{3.3.23}$$

The following assumptions are made:

- $g : \mathbb{R}^n \to \mathbb{R}$ is a continuous convex function which is possibly *nonsmooth*.
- $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth convex function of the type $C^{1,1}$, i.e, continuously differentiable with Lipschitz continuous gradient $L(f)$:

  $$||\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})|| \leq L(f)||\mathbf{x} - \mathbf{y}||,$$

  where $||\cdot||$ denotes the standard Euclidean norm and $L(f) > 0$ is the Lipschitz constant of ∇f.
- The problem is solvable, i.e. $\text{argmin}\, F \neq \emptyset$. This is almost always the case, excluding pathological cases.

Algorithm 1: ISTA with constant stepsize

Input: $L := L(f)$.
Initialize: $\mathbf{w}^0 = 0$.
for $k = 1, 2, \ldots$ **do**
  $\mathbf{w}^k = p_L(\mathbf{w}^{k-1})$
end

Now we are going to consider the following quadratic approximation of $F(\mathbf{w}) := f(\mathbf{w}) + g(\mathbf{w})$ at a given point $\mathbf{y}$:

$$Q_L(\mathbf{w}, \mathbf{y}) := f(\mathbf{y}) + \langle \mathbf{w} - \mathbf{y}, \nabla f(\mathbf{y}) \rangle + \frac{L}{2}||\mathbf{w} - \mathbf{y}||^2 + g(\mathbf{w}), \tag{3.3.24}$$

which admits an unique minimizer

$$p_L(\mathbf{y}) := \text{argmin}\{Q_L(\mathbf{w}, \mathbf{y}) : \mathbf{w} \in \mathbb{R}^n\}. \tag{3.3.25}$$

Simple algebra shows that (ignoring constant terms in $\mathbf{y}$)

$$p_L(\mathbf{y}) = \underset{\mathbf{w}}{\text{argmin}} \left\{ g(\mathbf{w}) + \frac{L}{2} \left\| \mathbf{w} - \left(\mathbf{y} - \frac{1}{L} \nabla f(\mathbf{y}) \right) \right\|^2 \right\}. \tag{3.3.26}$$

Note that the operator p_L is just the proximal operator of $(1/L)g$,

$$\text{prox}_g(x) = \underset{z \in \mathbb{R}}{\text{argmin}} \left(g(z) + \frac{1}{2}||x - z||^2 \right),$$

evaluated at the point $\mathbf{y} - \frac{1}{L}\nabla f(\mathbf{y})$,

$$\begin{aligned} \text{prox}_{\frac{1}{L}g} \left(\mathbf{y} - \frac{1}{L} \nabla f(\mathbf{y}) \right) &= \underset{\mathbf{w}}{\text{argmin}} \left\{ \frac{1}{L} g(\mathbf{w}) + \frac{1}{2} \left\| \left(\mathbf{y} - \frac{1}{L} \nabla f(\mathbf{y}) \right) - \mathbf{w} \right\|^2 \right\} \\ &= \underset{\mathbf{w}}{\text{argmin}} \left\{ g(\mathbf{w}) + \frac{L}{2} \left\| \mathbf{w} - \left(\mathbf{y} - \frac{1}{L} \nabla f(\mathbf{y}) \right) \right\|^2 \right\} = p_L(\mathbf{y}). \end{aligned}$$

that is, after taking a small step in the direction of the negative gradient of f. Finally, the basic step of the algorithm is

$$\mathbf{w}^k = p_L(\mathbf{w}^{k-1}). \tag{3.3.27}$$

Note that this is actually an alternative derivation of the proximal gradient method since $g(\mathbf{w})$ could be any nonsmooth regularizer and $f(\mathbf{w})$ any smooth convex function. However, Beck and Teboulle (2009a) still refer to this more general method as ISTA, which may be confusing. If $f(\mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2$ and $g(\mathbf{w}) = \lambda||\mathbf{w}||_1$ ($\lambda > 0$) then Eq. (3.3.27) reduces to Eq. (3.3.22) with $t = 1/L(f)$.

In practice, a possible drawback of the basic scheme showed in Algorithm 1 is that the Lipschitz constant $L(f)$ is not always known or computable. For instance, the Lipschitz constant in the Lasso problem depends on the maximum eigenvalue of $\mathbf{X}^\top\mathbf{X}$. For large-scale problems, this quantity is very expensive to compute. The trivial algorithm to compute eigenvalues needs $\mathcal{O}(n^3)$ operations, where n is the size of the matrix, although there are faster approaches if we

Algorithm 2: ISTA with backtracking

Initialize: $L_0 > 0$, $\eta > 1$, and $\mathbf{w}^0 = 0$.
for $k = 1, 2, \ldots$ **do**
  $\bar{L} = \eta^{i_k} L_{k-1}$
  Find the smallest nonnegative integers i_k such that
  $F(p_{\bar{L}}(\mathbf{w}^{k-1})) \leq Q_{\bar{L}}(p_{\bar{L}}(\mathbf{w}^{k-1}), \mathbf{w}^{k-1})$
  $L_k = \eta^{i_k} L_{k-1}$
  $\mathbf{w}^k = p_{L_k}(\mathbf{w}^{k-1})$
end

Algorithm 3: FISTA with constant stepsize

Input: $L := L(f)$.
Initialize: $\mathbf{y}_1 = \mathbf{w}^0 = 0$, $t_1 = 1$
for $k = 1, 2, \ldots$ **do**
  $\mathbf{w}^k = p_L(\mathbf{y}_k)$
  $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$
  $\mathbf{y}^{k+1} = \mathbf{w}^k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\mathbf{w}^k - \mathbf{w}^{k-1})$
end

only need a few of the eigenvalues, such as the maximum. Therefore we also analyse ISTA with a backtracking stepsize rule in Algorithm 2.

Theorem 3.4 states the convergence rate of the ISTA algorithm. The proof can be found in Beck and Teboulle (2009a).

Theorem 3.4. *Let $\{\mathbf{w}^k\}$ be the sequence generated by Algorithm 1 or Algorithm 2. Then, for any $k > 1$*

$$F(\mathbf{w}^k) - F(\mathbf{w}^*) \leq \frac{\alpha L(f)\|\mathbf{w}^0 - \mathbf{w}^*\|^2}{2k} \quad \forall \mathbf{w}^* \in W_* \tag{3.3.28}$$

where W_* is the set of possible solutions, $\alpha = 1$ for the constant stepsize setting and $\alpha = \eta$ for the backtracking stepsize setting.

The previous result can be interpreted as follows: the number of iterations required to obtain a solution $\tilde{\mathbf{w}}$ such that $F(\tilde{\mathbf{w}}) - F(\mathbf{w}^*) \leq \epsilon$, is at most $\lceil C/\epsilon \rceil$, where $C = \alpha L(f)\|\mathbf{w}^0 - \mathbf{w}^*\|^2/2$ or, in other words, the convergence rate is $\mathcal{O}(1/k)$.

Recall that ISTA is just a specific version of the more general proximal gradient method (3.3.18), which reduces to the gradient method when $g(\mathbf{x}) = 0$. Nesterov (1983) shows that it exists a gradient method with convergence rate $\mathcal{O}(1/k^2)$ which is an "optimal" first order method for smooth problems. We will describe in detail Nesterov's Acceleration in Section 6.1. Beck and Teboulle (2009a) extended the previous method to composite functions, where one of them is (possibly) non-smooth. This algorithm is known as Fast Iterative Shrinkage-Thresholding Algorithm (FISTA), an improved version of ISTA with convergence rate of $\mathcal{O}(1/k^2)$. The pseudocode for FISTA is showed in Algorithm 3.

FISTA can be also modified in the same way as ISTA in order to get rid of the Lipschitz constant $L(f)$. The pseudocode for FISTA with a backtracking stepsize rule can be seen in Algorithm 4.

Algorithm 4: FISTA with backtracking

Input: $L := L(f)$
Initialize: $L_0 > 0$, $\eta > 1$, $\mathbf{w}^0 = \mathbf{y}^1 = \mathbf{x}^0 = 0$ and $t_1 = 1$.
for $k = 1, 2, \ldots$ **do**
  $\bar{L} = \eta^{i_k} L_{k-1}$
  Find the smallest nonnegative integers i_k such that
  $F(p_{\bar{L}}(\mathbf{w}^{k-1})) \leq Q_{\bar{L}}(p_{\bar{L}}(\mathbf{w}^{k-1}), \mathbf{w}^{k-1})$
  $L_k = \eta^{i_k} L_{k-1}$
  $\mathbf{w}^k = p_L(\mathbf{y}_k)$
  $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$
  $\mathbf{y}^{k+1} = \mathbf{w}^k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\mathbf{w}^k - \mathbf{w}^{k-1})$
end

Note that the only important difference between Algorithms 1 and 2 and Algorithms 3 and 4 is that the operator p_L is not applied to the previous point $\mathbf{w}^{k-1}$ but to a smartly chosen linear combination of the previous two, $\mathbf{w}^{k-1}$ and $\mathbf{w}^{k-2}$. Since the computational burden is in the p_L operator and both algorithms require the same number of p_L evaluations, the $\mathcal{O}(\cdot)$ cost per iteration is almost identical. Clearly, the extra computation performed by FISTA is marginal in comparison to the p_L evaluation. In order to compute FISTA iteration complexity note that we can rewrite Eq. (3.3.22) as

$$\mathbf{w}^{k+1} = S_\lambda(\mathbf{w}^k - 2t(\mathbf{X}^\top \mathbf{X}\mathbf{w}^k - \mathbf{X}^\top \mathbf{y})).$$

Thus, assuming $\mathbf{X}^\top \mathbf{X}$ and $\mathbf{X}^\top \mathbf{y}$ are precomputed at a fixed initial cost $\mathcal{O}(nd^2)$, the cost per iteration of FISTA is $\mathcal{O}(d^2)$, i.e., that of computing $\left(\mathbf{X}^\top \mathbf{X}\right)\mathbf{w}^k$, which dominates the $\mathcal{O}(d)$ costs of the soft-thresholding and $\mathbf{w}$ updates.

Theorem 3.5. *Let $\{\mathbf{w}^k\}$ and $\{\mathbf{y}^k\}$ be generated by FISTA. Then, for any $k > 1$*

$$F(\mathbf{w}^k) - F(\mathbf{w}^*) \leq \frac{2\alpha L(f)\|\mathbf{w}^0 - \mathbf{w}^*\|^2}{(k+1)^2} \quad \forall \mathbf{w}^* \in \mathbf{W}_*, \tag{3.3.29}$$

where $\alpha = 1$ for the constant stepsize setting and $\alpha = \eta$ for the backtracking stepsize setting.

Theorem 3.5 shows that the number of iterations required by FISTA to obtain an ϵ-optimal solution $\mathbf{w}^*$, $F(\mathbf{w}) - F(\mathbf{w}^*) \leq \epsilon$, is at most, $\lceil C/\sqrt{\epsilon} - 1 \rceil$, where

$$C = \sqrt{2\alpha L(f)\|\mathbf{w}^0 - \mathbf{w}^*\|^2}.$$

Therefore FISTA has worst-case convergence rate of $\mathcal{O}(1/k^2)$, which clearly improves upon ISTA, and, since both algorithms have the same iteration complexity, in practice FISTA should also be much faster than ISTA. The proof of the theorem can be found in Beck and Teboulle (2009a). The convergence of ISTA and FISTA is also illustrated in Fig. 3.4. It is worth mentioning that the value of the function in FISTA it is not guaranteed to decrease in every iteration. Beck and Teboulle (2009b) suggest a modification of the algorithm, known as Monotone FISTA or MFISTA that guarantees descent at each iteration, that is

$$f(\mathbf{w}^k) \leq f(\mathbf{w}^{k-1}).$$

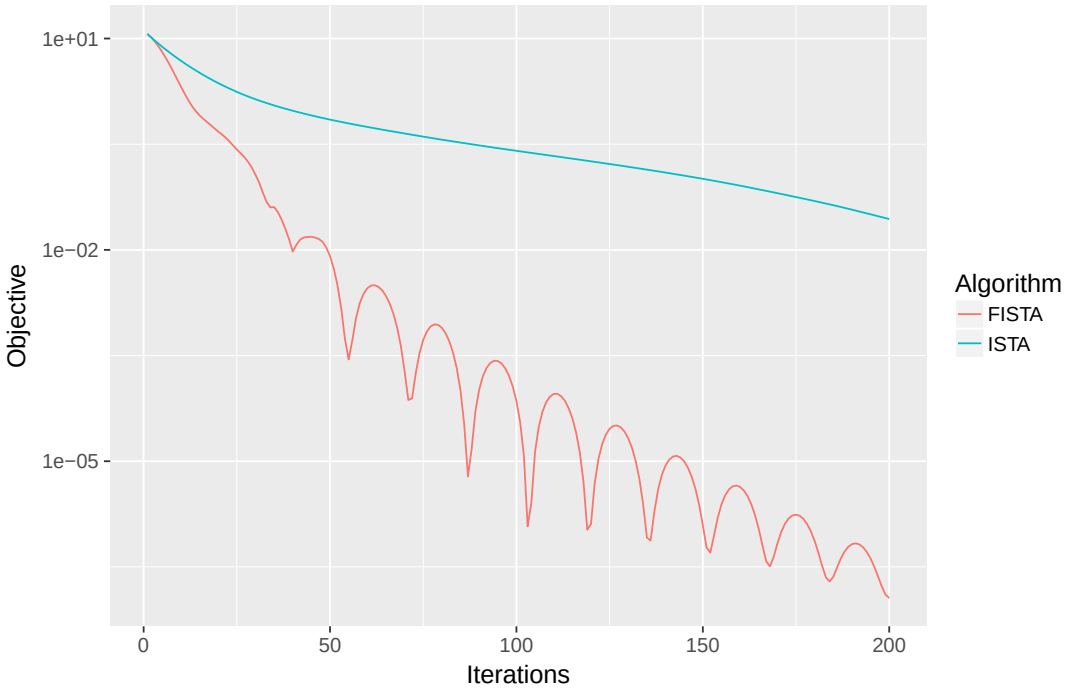

Figure 3.4: Convergence of ISTA and FISTA

The theoretical complexity of this new algorithm is the same as FISTA, and it can be found also in (Beck and Teboulle, 2009b).

Finally note that, similarly to ISTA, FISTA provides a general framework for projected gradient descent, that can be used in many other problems besides Lasso. As an example we mention here two of them, namely Elastic Net and Group Lasso. Elastic Net minimizes the function (3.2.1), thus we can select $f(\mathbf{w}) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda_2||\mathbf{w}||_2^2$ and $g(\mathbf{w}) = \lambda_1||\mathbf{w}||_1$. The gradient of f is now

$$\nabla f(\mathbf{w}) = 2(\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda_2\mathbf{w}), \tag{3.3.30}$$

and the proximal mapping is the same as in the Lasso problem (Eq. 3.3.21). Group Lasso has the same loss function as Lasso but the regularization term is now the $l_{2,1}$-norm,

$$||\mathbf{w}||_{2,1} = \sum_{j=1}^{J} \sqrt{||\mathbf{w}_j||_2}. \tag{3.3.31}$$

where J is the number of groups and $\mathbf{w}_j$ a vector with the weights in group j. Therefore, the gradient of f is given by Eq. (3.3.20) and the proximal mapping of the mixed norm is (Puig et al., 2009)

$$[\text{prox}_g(\mathbf{w})]_i = w_{ji}\left(1 - \frac{\lambda}{\|\mathbf{w}_j\|_2}\right)_+. \tag{3.3.32}$$

This flexibility and wide applicability implies a computational trade-off with the efficiency of problem-specific methods, that can usually take advantage of particular characteristics of the problem at hand.

3.3.3 Coordinate Descent

Early works on coordinate descent to solve the Lasso or, in general, ℓ_1-regularized problems include Fu (1998), Shevade and Keerthi (2003), Daubechies et al. (2004), and Kooij (2007). Extensive reviews can be found in Friedman et al. (2010) and Shalev-Shwartz and Tewari (2011). In particular, Friedman et al. (2007) recognize the importance of solving the problem along an entire path of values for the regularization parameters, using the current estimates for the warm-starts. Following the Gauss-Siedel approach of Zhang and Oles (2001), Genkin et al. (2007) describe a coordinate descent method (called BBR) for minimizing ℓ_1-regularized objectives.

Friedman et al. (2010) extend the work of Friedman et al. (2007) and develop fast cyclic coordinate descent algorithms for fitting generalized linear models with the more general Elastic Net penalties. In a series of experiments the authors show how these algorithms outperform many alternative methods such as LARS, `l1logreg` (Koh et al., 2007), BBR (Genkin et al., 2007) and the Lasso Penalized Logistic (LPL) program (Wu and Lange, 2008). Although no theoretical guarantees are provided, they also released an R package called `glmnet` that still is one of the most widely used Lasso solvers up to this day, popularizing the method. The success of the `glmnet` software can be also partly explained by its efficiency and ease of use: the package is written in Fortran with a R wrapper and it is highly optimized for solving the Lasso, with some ad-hoc tricks that make the solver very fast. On the negative side the code is pretty difficult to understand and not many contributions are built on top of it.

GLMNet builds the full regularization path by defining a sensible range of values for the regularization parameters, using the solution for a given value as a warm-start for the next. At every step the algorithm solves the problem for a given λ value by cyclically choosing one variable at a time and performing a simple analytical update until convergence. More formally, suppose we have estimates $\widetilde{\mathbf{w}}_l$ for $l \neq j$ and we want to partially optimize (3.1.4) with respect to w_j. The gradient at $w_j = \tilde{w}_j$ only exists if $\tilde{w}_j \neq 0$. In that case,

$$\frac{\partial F}{\partial w_j} = -\frac{1}{n}\sum_{i=1}^{n} x_{ij}(y_i - \mathbf{x}_i^\top \widetilde{\mathbf{w}}) + \lambda \operatorname{sign}(w_j)$$

and the w_j update is given in Friedman et al. (2010) as

$$\tilde{w}_j = \mathrm{S}_\lambda\left(\frac{1}{n}\sum_{i=1}^{n} x_{ij}(y_i - \tilde{y}_i^{(j)})\right), \tag{3.3.33}$$

where $\mathrm{S}_\lambda(\cdot)$ is the soft-thresholding operator (3.3.21)

$$\mathrm{S}_\gamma(z) = \operatorname{sign}(z)(|z| - \gamma)_+ = \begin{cases} 0, & \text{if } z \in [-\gamma, \gamma] \\ z - \gamma, & \text{if } z > \gamma \\ z + \gamma, & \text{if } z < -\gamma \end{cases}$$

and

$$\tilde{y}_i^{(j)} = \sum_{l \neq j} x_{il}\tilde{w}_l$$

is the partial fitted value, without the contribution from the variable j. It is useful for the computations to rewrite the previous equation as a function of the full fitted value $\hat{y}_i$,

$$\tilde{y}_i^{(j)} = \sum_{l \neq j} x_{il}\tilde{w}_l = \sum_{k=1}^{d} x_{ik}\tilde{w}_k - x_{ij}\tilde{w}_j = \hat{y}_i - x_{ij}\tilde{w}_j. \tag{3.3.34}$$

Algorithm 5: Cyclic Coordinate Descent (CCD), naive updates

Input : $\mathbf{w} = 0 \in \mathbb{R}^d$, $\mathbf{r} = \mathbf{y} - \mathbf{X}\mathbf{w} = \mathbf{y} \in \mathbb{R}^n$ and λ
for $t = 1, 2, \ldots$ **do**
  for $j = 1, 2, \ldots, d$ **do**
    $g = \frac{1}{n}\sum_{i=1}^{n} x_{ij} r_i$
    $\delta = \begin{cases} g - \lambda, & \text{if } w_j - g > \lambda \\ g + \lambda, & \text{if } w_j - g < -\lambda \\ -w_j, & \text{otherwise.} \end{cases}$
    $w_j \leftarrow w_j + \delta$
    if $w_j \neq 0$ **then**
      $\mathbf{r} \leftarrow \mathbf{r} + \delta \mathbf{x}_j$
    end
  end
end

Now, let $r_i = y_i - \hat{y}_i$ be the residuals; then by substituting Eq. (3.3.34) into Eq. (3.3.33) we get the final expression for the updates,

$$\begin{aligned} \tilde{w}_j &= \mathrm{S}_\lambda\left(\frac{1}{n}\sum_{i=1}^{n} x_{ij}(y_i - \tilde{y}_i^{(j)})\right) \\ &= \mathrm{S}_\lambda\left(\frac{1}{n}\sum_{i=1}^{n} x_{ij}(y_i - \hat{y}_i + x_{ij}\tilde{w}_j)\right) \\ &= \mathrm{S}_\lambda\left(\frac{1}{n}\sum_{i=1}^{n} x_{ij} r_i + \tilde{w}_j\right), \end{aligned} \tag{3.3.35}$$

since we assumed earlier that the $\mathbf{x}_j$ are standardized. The computational cost of each iteration is discussed by Friedman et al. (2010). Many coefficients are zero and remain zero after the thresholding, and thus nothing needs to be changed. The cost for such step is $\mathcal{O}(n)$ operations corresponding to the sum in Eq. (3.3.35). On the other hand, if a coefficient does change after the thresholding we need $\mathcal{O}(n)$ operations to update r_i on top of the ones to compute the sum, for a total cost of $\mathcal{O}(2n)$. Thus a complete cycle through all d variables costs $\mathcal{O}(dn)$ operations. Friedman et al. (2010) refer to Algorithm 5 as the *naive algorithm*, since we can obtain a more efficient update by carefully rearranging the computations.

Let us rewrite now the first term in Eq. (3.3.35) as

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} r_i = \sum_{i=1}^{n} x_{ij}(y_i - \hat{y}_i) = \sum_{i=1}^{n} x_{ij} y_i - \sum_{i=1}^{n} x_{ij}\hat{y}_i = \mathbf{x}_j \cdot \mathbf{y} - \sum_{i=1}^{n} x_{ij} \sum_{k=1}^{d} x_{ik}\tilde{w}_k = \\ = \mathbf{x}_j \cdot \mathbf{y} - \sum_{i=1}^{n}\sum_{k=1}^{d} x_{ij}x_{ik}\tilde{w}_k = \mathbf{x}_j \cdot \mathbf{y} - \sum_{k=1}^{d}\sum_{i=1}^{n} (x_{ij}x_{ik})\tilde{w}_k = \mathbf{x}_j \cdot \mathbf{y} - \sum_{k\,:\,|\tilde{w}_k|>0} (\mathbf{x}_j \cdot \mathbf{x}_k)\tilde{w}_k \end{aligned}$$

First, the inner products of each variable with $\mathbf{y}$ can be computed up front and then, each time a new variable $\mathbf{x}_j$ enters the model for the first time, we can compute and store its inner product with all the rest of the features in $\mathcal{O}(nd)$ operations. If one of the coefficients already in the model appears again we can update each gradient with only $\mathcal{O}(d)$ operations. Hence, the cost of a complete cycle is $\mathcal{O}(d^2)$ if no new variables become non-zero. This updates are called in

Algorithm 6: Cyclic Coordinate Descent (CCD), covariance updates

Input : $\mathbf{w} = 0 \in \mathbb{R}^d$, $\mathbf{z} = \mathbf{X}^\top \mathbf{y} \in \mathbb{R}^d$, $\mathbf{Q} \in \mathbb{R}^{d \times d}$ and λ
for $t = 1, 2, \ldots$ **do**
  for $j = 1, 2, \ldots, d$ **do**
    if j *not in the model* **then**
      for $k = 1, 2, \ldots, d$ **do**
        $\mathbf{Q}_{jk} \leftarrow \mathbf{x}_j \cdot \mathbf{x}_k$
      end
    end
    $g = z_j - \sum_{k\,:\,|w_k|>0} \mathbf{Q}_{jk} w_k$
    $w_j \leftarrow \mathrm{S}_\lambda\left(w_j + g\right) = \begin{cases} w_j + g - \lambda, & \text{if } w_j - g > \lambda \\ w_j + g + \lambda, & \text{if } w_j - g < -\lambda \\ 0, & \text{otherwise.} \end{cases}$
  end
end

Friedman et al. (2010) *covariance updates*, since we fill the entries of the covariance matrix $\mathbf{X}^\top \mathbf{X}$ as shown in Algorithm 6, which is then stored in memory for later iterations.

Note that, since we are cycling trough all the variables, both versions will greatly benefit if we are able to identify variables that are zero in the current iteration and will be zero also in the final model. Assume that is the case and let m be the number of non-zero variables, then the cost of a complete cycle can be further reduced to $\mathcal{O}(md)$ in the covariance updates setting. That is the goal of the screening methods that we will discuss later in Section 3.4. In particular, `glmnet` implements some *strong rules* for discarding predictors early (Tibshirani et al., 2012). Note that these rules are not *safe* in the sense that they may incorrectly discard predictors whose value is not zero in the final model. To solve that problem `glmnet` checks after every complete cycle if the KKT conditions of the problem are violated, in which case the offending predictors are brought back into the model.

As mentioned before, another trick that contributes to the speed of the algorithm is the regularization path, that is, `glmnet` does not compute only the solution for a single value of λ but for a decreasing sequence of values, starting at the smallest value $\lambda_{\max}$ for which $\mathbf{w}^* = 0$. The strategy is to select a minimum value $\lambda_{\min} = \epsilon \lambda_{\max}$ and construct a sequence from $\lambda_{\max}$ to $\lambda_{\min}$ of size K on the log scale. The value $\lambda_{\max}$ can be computed analytically and only depends of the data (Proposition 5.2),

$$\lambda_{\max} = \frac{\|\mathbf{X}^\top \mathbf{y}\|_\infty}{n}. \tag{3.3.36}$$

However both K and ϵ are constants that have to be selected. Computing the whole path of solutions for a given λ its often faster and more stable than solving the problem for that λ due to *warm starts* (the solution of one problem is used as a starting point for the next).

The convergence of coordinate descent has been extensively studied in a series of works, starting with Luo and Tseng (1992), and continuing with Tseng (2001), Tseng and Yun (2009a), and Tseng and Yun (2009b), just to mention a few. In particular, Luo and Tseng (1992) establish a linear convergence after an unspecified number of iterations. However, as noted by Shalev-Shwartz and Tewari (2011), the initial number of iterations scales at least quadratically with the number of samples n and it is thus useless if n is very big. In an attempt to improve that

dependence, Tseng and Yun (2009b) propose an algorithm with a runtime bound of order

$$\mathcal{O}\left(\frac{nd^2\|\mathbf{w}^*\|_2^2}{\epsilon}\right)$$

(equations 21 y 25), which scales only linearly with n. The previous result still holds only if the iteration counter is "large enough". Thus, motivated by the success of the experiments in Friedman et al. (2010), Saha and Tewari (2013) are able to prove a non-asymptotic convergence bound of $\mathcal{O}(1/k)$ for cyclic coordinate descent under some assumptions, where k is the number of iterations. This result is summarized in Theorem 3.6.

Theorem 3.6 (Theorem 16 in Saha and Tewari, 2013)**.** *Starting from a super- or sub-solution* $\mathbf{w}^0$*, let* $\left\{\mathbf{w}^k\right\}$ *denote the Cyclic Coordinate Descent (CCD) iterates. Under Assumptions 1 and 2 (Saha and Tewari, 2013), for any minimizer* $\mathbf{w}^*$ *of* (3.1.1)*, and for all* $k \geq 1$*,*

$$F(\mathbf{w}^k) - F(\mathbf{w}^*) \leq \frac{L\|\mathbf{w}^* - \mathbf{w}^0\|^2}{2k}$$

Note that the previous convergence result is the same as ISTA, and thus worse that the $\mathcal{O}(1/k^2)$ from FISTA. However, in practice Cyclic Coordinate Descent is faster to solve the Lasso problem since the updates are n times cheaper (Nesterov, 2012). Finally, even though the assumptions needed by (Saha and Tewari, 2013) to prove Theorem 3.6 are quite restrictive, the same convergence is to be expected even without them, but there has not been a formal proof yet.

3.3.4 Stochastic Coordinate Descent

In general, all the standard methods seen so far to solve the Lasso problem scale poorly with the size of the problem (n and d). To overcome that difficulty, Shalev-Shwartz and Tewari (2011) were one of the first to explore stochastic variants of well-known algorithms like coordinate descent and gradient descent. In particular, they propose a stochastic coordinate descent algorithm that chooses the next feature to optimize uniformly at random instead of cyclically. A similar approach is used by Richtárik and Takáč (2014), where they extend the results of Nesterov (2012) on stochastic gradient descent for composite smooth functions, to cover the non-smooth case, including functions like the ℓ_1-penalty. Although a more general algorithm is discussed in the previous works, able to solve also other problems like logistic regression, we only present here the Lasso version (Algorithm 7). Note that other choices for the next coordinate are also possible, like the greedy approach by Tseng and Yun (2009a), where they select the feature to optimize next as the one with the most promising descent.

There are two main differences between Algorithms 6 and 7:

1. The cyclic pass through all the coordinates is replaced by an uniform sampling.
2. We now have a step size β different from 1.

Thus, setting $\beta = 1$, Algorithm 7 is just a stochastic version of coordinate descent with naive updates (Algorithm 6). In order to see that, simply note that the residuals are defined now as $\mathbf{r} = \mathbf{X}\mathbf{w} - \mathbf{y}$ instead of $\mathbf{r} = \mathbf{y} - \mathbf{X}\mathbf{w}$, so some signs have to be reverted. Again the $\mathbf{r}$ variable is kept for efficiency purposes, since it allows us to compute g in $\mathcal{O}(sn)$ operations where

$$s = \frac{|\{(i,j) \mid x_{ij} \neq 0\}|}{nd}$$

Algorithm 7: Stochastic Coordinate Descent (SCD)

Input : $\mathbf{w} = 0 \in \mathbb{R}^d$, $\mathbf{r} = \mathbf{X}\mathbf{w} - \mathbf{y} = -\mathbf{y} \in \mathbb{R}^n$, λ and β

for $t = 1, 2, \ldots$ **do**

Sample j uniformly at random from $\{1, \ldots, d\}$

$g = \frac{1}{n} \sum_{i\,:\,x_{ij} \neq 0} r_i x_{ij}$

$$\delta = \begin{cases} -\frac{g+\lambda}{\beta}, & \text{if } w_j - \frac{g+\lambda}{\beta} > 0 \\ -\frac{g-\lambda}{\beta}, & \text{if } w_j - \frac{g-\lambda}{\beta} < 0 \\ -w_j, & \text{otherwise} \end{cases}$$

$w_j \leftarrow w_j + \delta$

$\mathbf{r} \leftarrow \mathbf{r} + \delta \mathbf{x}_j$

end

is the average number of non-zeros in our data. Interestingly, some experiments by Shalev-Shwartz and Tewari (2011) report SCD to be indistinguishable from CCD in terms of efficiency, measured by data accesses.

The step size β is related to the Lipschitz constant of the loss function, which for the squared loss is set to $\beta = \max_j L_j = 1$ in Shalev-Shwartz and Tewari (2011), where L_j are the coordinate constants. However, Richtárik and Takáč (2014) argue that this choice necessarily produces very small step lengths and they suggest to use a different constant per coordinate, computed as

$$L_j = \|\mathbf{x}_j\|_2^2.$$

Using the previous value for β, they are able to improve the bound to obtain a ϵ-optimal solution from (Shalev-Shwartz and Tewari, 2011),

$$\mathcal{O}\left(\frac{dm\beta\|\mathbf{w}^*\|_2^2}{\epsilon}\right), \tag{3.3.37}$$

to

$$\mathcal{O}\left(\frac{dm\|\mathbf{w}^*\|_L^2}{\epsilon}\right).$$

Other differences between those works is that Richtárik and Takáč (2014) analyse a block version and considers different sampling probabilities, whereas Shalev-Shwartz and Tewari (2011) do not. In fact, the experiments in Richtárik and Takáč (2014) and recent results like Allen-Zhu et al. (2016), Qu and Richtárik (2016a), and Qu and Richtárik (2016b) suggest that a non-uniform sampling could yield even faster algorithms. Finally it is also worth mentioning that there are some parallel versions of Algorithm 7 like Shotgun (Bradley et al., 2011), Scherrer et al. (2012), Fercoq and Richtárik (2015), and Richtárik and Takáč (2016).

3.3.5 Stochastic Gradient Descent

In Section 3.3.2 we described FISTA, a batch method with a convergence rate of $\mathcal{O}(1/k^2)$. However, every FISTA iteration needs the full gradient of the loss function and thus it scales poorly with the number of samples n. A natural idea for trying to improve this dependence is to explore an online version of FISTA using stochastic gradient descent. In stochastic gradient descent (SGD) we pick one sample uniformly at random at each iteration and update the weights based on the chosen sample (Shalev-Shwartz and Tewari, 2011). Thus, the asymptotic runtime of the algorithm does not depend at all on the number of samples.

Algorithm 8: Truncated Gradient Descent (TGD)

Input : $\mathbf{w} = 0 \in \mathbb{R}^d$, learning rate $\eta \in (0, 1)$, K and λ
for $k = 1, 2, \ldots$ **do**
  Sample i uniformly at random from $\{1, \ldots, n\}$
  every K *iterations* **do**
    for $j = 1, 2, \ldots, d$ **do**
      $$w_j \leftarrow \begin{cases} \max(w_j - K\lambda\eta,\, 0), & \text{if } w_j \in [0, \lambda] \\ \min(w_j + K\lambda\eta,\, 0), & \text{if } w_j \in [-\lambda, 0] \end{cases}$$
    end
  end
  $\mathbf{w} \leftarrow \mathbf{w} + 2\eta(y_i - \mathbf{x}_i \cdot \mathbf{w})\mathbf{x}_i$
end

As noted by Shalev-Shwartz and Tewari (2011), a big problem with these algorithms is that they fail to produce sparse solutions, which is usually the main reason to use ℓ_1-regularization in the first place. To overcome that problem there are a few options, the main ones being (Langford et al., 2009):

- Consider the constrained Lasso formulation (3.1.1), which is equivalent to the usual unconstrained form and project the weights into an ℓ_1-ball after every online step. The main disadvantage of this approach is the projection operation, since it is difficult to implement efficiently for large-scale data. This algorithm was proposed by Duchi et al. (2008), where they also reduce the amortized time of the projection operator from $\mathcal{O}(d)$ to $\mathcal{O}(m \log(d))$ where m is the number of non-zero entries. Other works involving efficient projections include Liu and Ye (2009) and Liu and Ye (2010). This algorithm is implemented in the SLEP package (Liu et al., 2009).

- Truncate the weights that cross 0 after every online step, achieving an online version of the Proximal Gradient Methods for the ℓ_1-norm. A simplified version of the procedure from Langford et al. (2009) is shown in Algorithm 8. Note that every iteration where the coefficients are not truncated is just a standard stochastic gradient descent update, and equivalent to setting for those iterations $\lambda = 0$. Langford et al. (2009) also provide the following runtime bound for the general case

$$\mathcal{O}\left(\frac{d\bar{x}\|\mathbf{w}^*\|_2^2}{\epsilon^2}\right), \tag{3.3.38}$$

  where $\bar{x} = 1/n \sum_i \|\mathbf{x}_i\|_2^2$ is the average square norm of the samples.

  Algorithm 8 has two hyper-parameters: the number of online steps before the truncation K and the learning rate η. Setting $K = 1$ is the most aggressive choice since it performs the truncation at every iteration and leads to sparser intermediate coefficients; see Langford et al. (2009) for more details and experiments.

Following Langford et al. (2009), Shalev-Shwartz and Tewari (2011) suggest the use of a slightly more sophisticated descent rule, stochastic mirror descent, together with the truncation operation to obtain sparse solutions. They call their algorithm SMIDAS (*Stochastic MIrror*

Descent Algorithm Made Sparse), and provide the following upper bound on the runtime to achieve an ϵ-expected accuracy:

$$\mathcal{O}\left(\frac{d\log(d)\|\mathbf{w}^*\|_1^2}{\epsilon^2}\right). \tag{3.3.39}$$

Comparing bounds (3.3.38) and (3.3.39) it can be seen that none of them dominates the other and the relative performance depends on properties of the training set and the optimal solution $\mathbf{w}^*$, although Eq. (3.3.39) is superior if $\mathbf{w}^*$ is sparse (see Shalev-Shwartz and Tewari, 2011, for more details).

In general, mirror descent algorithms (Nemirovskii et al., 1983, Chapter 3) maintain two weight vectors: primal $\mathbf{w}$ and dual $\boldsymbol{\theta}$. The connection between the two is via a link function $\boldsymbol{\theta} = f(\mathbf{w})$, which is always taken to be the gradient map of some strongly convex function and is therefore invertible. Shalev-Shwartz and Tewari (2011) use the p-norm link function, $f(\mathbf{w}) = \nabla \frac{1}{2}\|\mathbf{w}\|_q^2$, where $\|\mathbf{w}\|_q = (\sum_j |w_j|^q)^{1/q}$. That is, the jth element of f and f^{-1} are, respectively,

$$f_j(\mathbf{w}) = \frac{\operatorname{sign}(w_j)|w_j|^{q-1}}{\|\mathbf{w}\|_q^{q-2}} \quad \text{and} \quad f_j^{-1}(\boldsymbol{\theta}) = \frac{\operatorname{sign}(\theta_j)|\theta_j|^{p-1}}{\|\boldsymbol{\theta}\|_p^{p-2}}, \tag{3.3.40}$$

where $p = q/(q-1)$.

These algorithms can be applied to the Least Squares problem similarly to stochastic gradient descent. At each iteration of the algorithm we first sample a training sample i uniformly at random. Then the gradient is estimated using only that sample as

$$\mathbf{g} = (\mathbf{w} \cdot \mathbf{x}_i - y_i)\mathbf{x}_i.$$

Next the dual vector is updated as $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta\mathbf{g}$, where η is again a learning rate. Finally, the update on $\boldsymbol{\theta}$ translates into an update on $\mathbf{w}$ via the inverse of the link function, $\mathbf{w} = f^{-1}(\boldsymbol{\theta})$. In this step it is important to note that if the link function is the identity mapping we recover standard stochastic gradient descent.

When working with the Lasso, the same procedure can be used simply by subtracting any subgradient of the ℓ_1-regularization term when updating the weights,

$$\theta_j = \theta_j - \eta(g_j + \lambda \operatorname{sign}(w_j)).$$

As discussed earlier, these updates produce a dense $\boldsymbol{\theta}$, which also leads to a dense $\mathbf{w}$. Shalev-Shwartz and Tewari (2011) adopt the solution from Langford et al. (2009) and truncate the weights that crossed the zero value. The pseudocode of SMIDAS is shown in Algorithm 9.

Let us compare now the previous algorithms to stochastic coordinate descent. On one hand, the runtime bound of stochastic gradient descent and variants avoid the dependence on n. On the other hand, the dependence of stochastic coordinate descent on the dimension d is better both because the lack of the term $\log(d)$ and because $\|\mathbf{w}^*\|_2^2$ is always smaller than $\|\mathbf{w}^*\|_1^2$. Another important practical drawback is the fact that Truncated Gradient Descent and SMIDAS both have hyper-parameters to tune, like the learning rate η. Finally, it is worth mentioning that other variants of Stochastic Gradient Descent applied to the Lasso have appeared in the literature in recent years, for instance RDA (Xiao, 2010), ProxSVRG (Xiao and Zhang, 2014), SAGA (Defazio et al., 2014) and mS2GD (Konečnỳ et al., 2016).

3.3.6 Stochastic Dual Coordinate Ascent

After the success of the basic Stochastic Gradient Descent and Stochastic Coordinate Descent algorithms to solve the Lasso problem many variants have appeared in recent years. One proposal

Algorithm 9: Stochastic MIrror Descent Algorithm made Sparse (SMIDAS)

Input : $\mathbf{w} = 0 \in \mathbb{R}^d$, $\boldsymbol{\theta} = 0 \in \mathbb{R}^d$, learning rate $\eta > 0$, f^{-1} as in (3.3.40) with $p = 2\log(d)$ and λ

for $k = 1, 2, \ldots$ **do**

  Sample i uniformly at random from $\{1, \ldots, n\}$

  $\mathbf{g} = (\mathbf{w} \cdot \mathbf{x}_i - y_i)\mathbf{x}_i$

  $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} - \eta\mathbf{g}$

  forall j **do**

    $\theta_j = \text{sign}(\tilde{\theta}_j)\max\{0, |\tilde{\theta}_j| - \eta\lambda\}$

  end

  $\mathbf{w} = f^{-1}(\boldsymbol{\theta})$

end

that is very interesting to us for its relation to the SMO algorithm (Section 4.3.3) is the Proximal Stochastic Dual Coordinate Ascent (Prox-SDCA) by Shalev-Shwartz and Zhang (2014). Let $\phi_1, \ldots, \phi_n$ be a sequence of scalar convex functions, let $g(\cdot)$ be a convex function defined on $\mathbb{R}^d$ and define the convex conjugate of ϕ_i as $\phi_i^*(u) = \max_z(zu - \phi_i(z))$. The goal is to solve the problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} P(\mathbf{w}) \quad \text{where} \quad P(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\phi_i(\mathbf{x}_i \cdot \mathbf{w}) + \lambda g(\mathbf{w}). \tag{3.3.41}$$

Now, let

$$v(\boldsymbol{\alpha}) = \frac{1}{\lambda n}\sum_{i=1}^{n}\mathbf{x}_i\alpha_i,$$

the dual problem of (3.3.41) is

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} D(\boldsymbol{\alpha}) \quad \text{where} \quad D(\boldsymbol{\alpha}) = \frac{1}{n}\sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \lambda g^*\left(v(\boldsymbol{\alpha})\right) \tag{3.3.42}$$

We will assume that g is strongly convex which implies that $g^*(\cdot)$ is continuous differentiable. If we define,

$$w(\boldsymbol{\alpha}) = \nabla g^*(v(\boldsymbol{\alpha})) \tag{3.3.43}$$

then it is known that $w(\boldsymbol{\alpha}^*) = \mathbf{w}^*$, where $\boldsymbol{\alpha}^*$ is a solution of (3.3.42) and $\mathbf{w}^*$ is a solution of (3.3.41). It is also known that $P(\mathbf{w}^*) = D(\boldsymbol{\alpha}^*)$. Thus, Eq. (3.3.43) relates the solution of the primal and dual problems.

Prox-SDCA optimizes at every iteration the dual objective with respect to a single dual variable, which is associated with a single training example. In that sense the method is a mix between SCD and SGD, since a single random coordinate is updated at every iteration but now the coordinates are actually associated with training examples, since we are solving the dual problem. In the same fashion SMO could belong to the Greedy Dual Block Coordinate Ascent family where, at every iteration, two coefficients are selected in a greedy manner to maximize ascent.

Finally, if we were to apply Prox-SDCA to solve the Lasso we simply let $\phi(a) = \frac{1}{2}(a - y)^2$. The conjugate function is

$$\phi^*(b) = \max_a\{ab - \frac{1}{2}(a - y)^2\} = \frac{1}{2}b^2 + yb.$$

The regularizer is $g(\mathbf{w}) = \|\mathbf{w}\|_1$. However, the ℓ_1-norm is not strongly convex and thus not directly applicable in Prox-SDCA, since we cannot compute the derivative. Shalev-Shwartz and Zhang (2014) solve that problem by adding a slight ℓ_2-regularization

$$g(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{\sigma}{\lambda}\|\mathbf{w}\|_1.$$

Note that $\lambda g(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sigma\|\mathbf{w}\|_1$, so if we make λ small enough we get $\lambda g(\mathbf{w}) \approx \sigma\|\mathbf{w}\|_1$. More specifically, let ϵ be the target accuracy. If we set $\lambda = \epsilon(\sigma/\bar{y})^2$, where $\bar{y} = \frac{1}{2n}\sum_{i=1}^n y_i^2$, we can show that a solution $\mathbf{w}^*$ of the ℓ_1-ℓ_2 regularized problem is also a ϵ-optimal solution of the original Lasso problem with parameter σ. The conjugate of g is then (Shalev-Shwartz and Zhang, 2014)

$$\begin{aligned} g^*(\mathbf{v}) &= \max_{\mathbf{w}} \left\{ \mathbf{w}\cdot\mathbf{v} - \frac{1}{2}\|\mathbf{w}\|_2^2 - \frac{\sigma}{\lambda}\|\mathbf{w}\|_1 \right\} \\ &= \frac{1}{2}\sum_i \left(\left[|v_i| - \frac{\sigma}{\lambda} \right]_+ \right)^2 \end{aligned}$$

and its gradient

$$\begin{aligned} \nabla g^*(\mathbf{v}) &= \operatorname*{argmax}_{\mathbf{w}} \left\{ \mathbf{w}\cdot\mathbf{v} - \frac{1}{2}\|\mathbf{w}\|_2^2 - \frac{\sigma}{\lambda}\|\mathbf{w}\|_1 \right\} \\ &= \operatorname*{argmin}_{\mathbf{w}} \left\{ \frac{1}{2}\|\mathbf{w}-\mathbf{v}\|_2^2 + \frac{\sigma}{\lambda}\|\mathbf{w}\|_1 \right\}. \end{aligned}$$

Solving the previous problem (Shalev-Shwartz and Zhang, 2014) we get that the ith component of the gradient is $\nabla_i g^*(\mathbf{v}) = \operatorname{sign}(v_i)\left[|v_i| - \frac{\sigma}{\lambda}\right]_+$. Shalev-Shwartz and Zhang (2014) also suggest some acceleration technique inspired by Nesterov's Acceleration. The full algorithm can be found in Shalev-Shwartz and Zhang (2012). Putting everything together, they are able to prove a runtime bound of,

$$\tilde{\mathcal{O}}\left(d\left(n + \min\left\{ \frac{R^2\|\mathbf{w}^*\|_2^2}{\epsilon}, \sqrt{\frac{nR^2\|\mathbf{w}^*\|_2^2}{\epsilon}} \right\} \right) \right), \tag{3.3.44}$$

where $R = \max_i \|\mathbf{x}_i\|_2$ and $\mathbf{w}^*$ is the solution of the Lasso problem, without the small ℓ_2-regularization. The notation $\tilde{\mathcal{O}}(\cdot)$ or "soft O" ignores logarithmic terms,

$$\exists k \ : \ \tilde{\mathcal{O}}(g(n)) = \mathcal{O}(g(n)\log^k g(n)).$$

Comparing bound (3.3.44) with the ones from previous sections we have, for FISTA

$$\mathcal{O}\left(dn\sqrt{\frac{R^2\|\mathbf{w}^*\|_2^2}{\epsilon}} \right),$$

which is worse by a factor of at least $\sqrt{n}$ (Shalev-Shwartz and Zhang, 2014). Recall also the runtime bounds for Stochastic Coordinate Descent Eq. (3.3.37),

$$\mathcal{O}\left(\frac{dn\|\mathbf{w}^*\|_2^2}{\epsilon} \right)$$

and Stochastic Gradient Descent Eq. (3.3.38),

$$\mathcal{O}\left(\frac{dR^2\|\mathbf{w}^*\|_2^2}{\epsilon^2} \right).$$

SGD is much slower, specially when ϵ is small. Concerning the SCD bound, it depends on R^2 and Prox-SDCA is faster if $R^2 = \mathcal{O}(1)$. In the general case, Prox-SDCA is the same or better whenever $d = \mathcal{O}(n)$ (see Shalev-Shwartz and Zhang, 2014 for more details). Other works regarding SDCA have also appeared in the literature, like Csiba et al. (2015) and Qu et al. (2015), where they consider adaptive probabilities instead of an uniform distribution.

3.4 Screening

As we briefly mentioned before, almost any Lasso algorithm will benefit from having a smaller set of active features. Starting with the seminal work of El Ghaoui et al. (2010), there has been quite a substantial amount of recent proposals to accelerate Lasso by screening, i.e., removing those features that will result on zero Lasso coefficients in the final model. A good review can be found in Xiang et al. (2017). The main idea is to consider the Lasso dual problem and exploit the KKT conditions. More precisely, we can introduce a new variable $\mathbf{z}$ and write the following contrained Lasso version,

$$\min_{\mathbf{w},\mathbf{z}} \left\{ \frac{1}{2}\|\mathbf{z}\|^2 + \lambda\|\mathbf{w}\|_1 \right\} \quad \text{s.t.} \quad \mathbf{z} = \mathbf{y} - \mathbf{X}\mathbf{w}. \tag{3.4.1}$$

By introducing dual variables $\boldsymbol{\beta} \in \mathbb{R}^n$ we get the Lagrangian of the previous problem (Wang et al., 2013)

$$L(\mathbf{w}, \mathbf{z}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{z}\|_2^2 + \lambda\|\mathbf{w}\|_1 + \boldsymbol{\beta}^\top(\mathbf{y} - \mathbf{X}\mathbf{w} - \mathbf{z}).$$

The dual problem is,

$$\max_{\boldsymbol{\beta}} \left\{ \min_{\mathbf{w},\mathbf{z}} L(\mathbf{w}, \mathbf{z}, \boldsymbol{\beta}) = \min_{\mathbf{z}} \left\{ \frac{1}{2}\|\mathbf{z}\|_2^2 - \boldsymbol{\beta}^\top\mathbf{z} \right\} + \min_{\mathbf{w}} \left\{ \lambda\|\mathbf{w}\|_1 - \boldsymbol{\beta}^\top\mathbf{X}\mathbf{w} \right\} + \boldsymbol{\beta}^\top\mathbf{y} \right\} \tag{3.4.2}$$

Note that the objective function is separable and thus we can solve the two inner optimization problems independently. Let us consider first

$$\min_{\mathbf{w}} f_1(\mathbf{w}) = \min_{\mathbf{w}} \left\{ \lambda\|\mathbf{w}\|_1 - \boldsymbol{\beta}^\top\mathbf{X}\mathbf{w} \right\}. \tag{3.4.3}$$

We can write the subgradient of f_1 as

$$\partial f_1(\mathbf{w}) = \lambda\partial\|\mathbf{w}\|_1 - \boldsymbol{\beta}^\top\mathbf{X}.$$

Now we have to find a $\mathbf{w}'$ such that

$$0 \in \partial f_1(\mathbf{w}') = \left\{ \lambda\mathbf{s}' - \boldsymbol{\beta}^\top\mathbf{X} \right\},$$

where

$$\mathbf{s}' \in \partial\|\mathbf{w}\|_1 = \left\{ s \;\middle|\; \|s\|_\infty \leq 1,\, s^\top\mathbf{w} = \|\mathbf{w}\|_1 \right\}.$$

Putting everything together, $\mathbf{w}'$ and $\boldsymbol{\beta}'$ should satisfy

$$\mathbf{s}' = \frac{\boldsymbol{\beta}^\top\mathbf{X}}{\lambda}, \tag{3.4.4}$$

$$\|\mathbf{s}'\|_\infty \leq 1, \tag{3.4.5}$$

$$\mathbf{s}'^\top\mathbf{w} = \|\mathbf{w}\|_1. \tag{3.4.6}$$

By substituting Eq. (3.4.4) into Eq. (3.4.5) we get the equivalent constraint

$$\left|\mathbf{x}_j^\top \boldsymbol{\beta}\right| \leq \lambda,\ j = 1, \ldots, d.$$

Then we substitute Eqs. (3.4.4) and (3.4.6) into the objective function of problem (3.4.3)

$$f_1(\mathbf{w}') = \lambda \mathbf{s}'^\top \mathbf{w}' - \boldsymbol{\beta}\mathbf{X}\mathbf{w}' = \lambda \left(\frac{\boldsymbol{\beta}^\top \mathbf{X}}{\lambda}\right) \mathbf{w}' - \boldsymbol{\beta}^\top \mathbf{X}\mathbf{w}' = 0.$$

Thus the optimal value of problem (3.4.3) is 0. Next, let us consider the sub-problem

$$\min_{\mathbf{z}} f_2(\mathbf{z}) = \min_{\mathbf{z}}\ \left\{\frac{1}{2}\|\mathbf{z}\|_2^2 - \boldsymbol{\beta}^\top \mathbf{z}\right\}. \tag{3.4.7}$$

The gradient is $\nabla f_2(\mathbf{z}) = \mathbf{z} - \boldsymbol{\beta}$ and thus the minimum is attained at $\mathbf{z}' = \boldsymbol{\beta}$. Therefore,

$$f_2(\mathbf{z}') = f_2(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 - \boldsymbol{\beta}^\top \boldsymbol{\beta} = -\frac{1}{2}\|\boldsymbol{\beta}\|_2^2,$$

since $\|\boldsymbol{\beta}\|_2^2 = \boldsymbol{\beta}^\top \boldsymbol{\beta}$. Substituting the value of the sub-problems in (3.4.2) we get

$$\max_{\boldsymbol{\beta}}\ \left\{\boldsymbol{\beta}^\top \mathbf{y} - \frac{1}{2}\|\boldsymbol{\beta}\|_2^2\right\} \quad \text{s.t.} \quad \left|\mathbf{x}_j^\top \boldsymbol{\beta}\right| \leq \lambda,\ j = 1, 2, \ldots, d.$$

The objective function in the previous problem can be written as

$$\begin{aligned}\boldsymbol{\beta}^\top \mathbf{y} - \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 &= \frac{1}{2}\|\mathbf{y}\|_2^2 - \frac{1}{2}\|\mathbf{y}\|_2^2 + \boldsymbol{\beta}^\top \mathbf{y} - \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 \\ &= \frac{1}{2}\|\mathbf{y}\|_2^2 - \frac{1}{2}\|\boldsymbol{\beta} - \mathbf{y}\|_2^2,\end{aligned}$$

and by simple re-scaling of the dual variables $\boldsymbol{\theta} = \boldsymbol{\beta}/\lambda$ we can also write it as

$$\max_{\boldsymbol{\theta}}\ \left\{\frac{1}{2}\|\mathbf{y}\|^2 - \frac{\lambda^2}{2}\left\|\boldsymbol{\theta} - \frac{\mathbf{y}}{\lambda}\right\|^2\right\} \quad \text{s.t.} \quad \left|\mathbf{x}_j^\top \boldsymbol{\theta}\right| \leq 1,\ j = 1, 2, \ldots, d. \tag{3.4.8}$$

Now the solutions of the primal and dual problem are related via the KKT conditions. Let $\mathbf{w}^*$, $\mathbf{z}^*$ and $\boldsymbol{\theta}^*$ be the optimal primal and dual variables. The Lagrangian is

$$L(\mathbf{w}, \mathbf{z}, \boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{z}\|_2^2 + \lambda\|\mathbf{w}\|_1 + \lambda\boldsymbol{\theta}^\top(\mathbf{y} - \mathbf{X}\mathbf{w} - \mathbf{z}),$$

and from the KKT conditions we have

$$0 \in \partial_{\mathbf{w}} L(\mathbf{w}^*, \mathbf{z}^*, \boldsymbol{\theta}^*) = -\lambda \mathbf{X}^\top \boldsymbol{\theta}^* + \lambda \mathbf{s} \tag{3.4.9}$$

$$0 = \nabla_{\mathbf{z}} L(\mathbf{w}^*, \mathbf{z}^*, \boldsymbol{\theta}^*) = \mathbf{z}^* - \lambda \boldsymbol{\theta}^* \tag{3.4.10}$$

$$0 = \nabla_{\boldsymbol{\theta}} L(\mathbf{w}^*, \mathbf{z}^*, \boldsymbol{\theta}^*) = \lambda(\mathbf{y} - \mathbf{X}\mathbf{w}^* - \mathbf{z}^*) \tag{3.4.11}$$

where $\mathbf{s} \in \partial\|\mathbf{w}^*\|$. From Eqs. (3.4.10) and (3.4.11) we get

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + \lambda\boldsymbol{\theta}^*,$$

i.e., $\lambda\boldsymbol{\theta}^*$ is the optimal residual. Using Eq. (3.4.9) and the subgradient of the ℓ_1-norm we can also conclude

$$\begin{aligned}&\mathbf{x}_j^\top \boldsymbol{\theta}^* = \text{sign}(w_j^*) \quad \text{if } w_j^* \neq 0, \\ &\mathbf{x}_j^\top \boldsymbol{\theta}^* \in [-1, 1] \qquad \text{if } w_j^* = 0.\end{aligned}$$

Note that $\mathcal{F} = \left\{\boldsymbol{\theta} \;\middle|\; \left|\mathbf{x}_j^\top \boldsymbol{\theta}\right| \leq 1,\ 1 \leq j \leq d\right\}$, that is, the feasible set of (3.4.8) is a closed convex polytope, and problem (3.4.8) can be interpreted as finding the projection

$$\boldsymbol{\theta}_\lambda = \mathrm{P}_{\mathcal{F}}\left(\mathbf{y}/\lambda\right)$$

of $\mathbf{y}/\lambda$ on $\mathcal{F}$.

The general idea in screening is to exploit this by identifying a convex region $\mathcal{R} = \mathcal{R}_\lambda \subset \mathcal{F}$ that contains $\boldsymbol{\theta}_\lambda$ and then screen out all features j for which $\sup_{\mathcal{R}} |\mathbf{x}_j \cdot \boldsymbol{\theta}| < 1$. This can be done taking $\mathcal{R}$ to be an appropriate ball. More precisely, assume we know $\boldsymbol{\theta}_{\lambda_0}$ for some λ_0; then, if $\lambda < \lambda_0$, the non-expansiveness of the projection operator ensures that

$$\|\boldsymbol{\theta}_\lambda - \boldsymbol{\theta}_{\lambda_0}\| \leq \left\|\frac{\mathbf{y}}{\lambda} - \frac{\mathbf{y}}{\lambda_0}\right\| = \left(\frac{1}{\lambda} - \frac{1}{\lambda_0}\right)\|\mathbf{y}\| \,. \tag{3.4.12}$$

Thus, the ball $B\left(\boldsymbol{\theta}_{\lambda_0}, R_\lambda^{\lambda_0}\right)$ with $R_\lambda^{\lambda_0} = (1/\lambda - 1/\lambda_0)\|\mathbf{y}\|$ is an example of a screening region $\mathcal{R}_\lambda$ for any $\lambda < \lambda_0$. An extra advantage of working with balls is that their **support function**

$$\sigma_{B(c,R)}(\mathbf{z}) = \max_{\mathbf{X} \in B} \mathbf{X} \cdot \mathbf{z}$$

has a simple analytic form (Fercoq et al., 2015), namely $\sigma_{B(\mathbf{c},R)}(\mathbf{z}) = \mathbf{z} \cdot \mathbf{c} + R\|\mathbf{z}\|$, and thus for $B = B\left(\theta_{\lambda_0}, R_\lambda^{\lambda_0}\right)$,

$$\begin{aligned}\sup_{\boldsymbol{\theta} \in B} |\mathbf{x}_j \cdot \boldsymbol{\theta}| &= \max\{\sigma_B(+\mathbf{x}_j),\, \sigma_B(-\mathbf{x}_j)\} \\ &= |\mathbf{x}_j \cdot \boldsymbol{\theta}_{\lambda_0}| + R_\lambda^{\lambda_0}\|\mathbf{x}_j\| \,.\end{aligned}$$

Slightly more complicated and more precise dome regions are also used for screening (Xiang et al., 2017; Fercoq et al., 2015), as their support functions are also relatively simple to compute.

We briefly discuss now how to find a ball center $\boldsymbol{\theta}_{\lambda_0}$. Since $0 \in \mathcal{F}$, we have $\mathbf{y}/\lambda \in \mathcal{F}$ for λ big enough. In fact, setting

$$\lambda_{\max} = \max_j \{|\mathbf{x}_j \cdot \mathbf{y}|\},$$

it is well known that $\mathbf{w}_{\lambda_{\max}} = 0$, i.e., $\boldsymbol{\theta}_\lambda = \mathbf{y}/\lambda$ if $\lambda \geq \lambda_{\max}$. Therefore $B\left(\mathbf{y}/\lambda_{\max}, R_\lambda^{\lambda_{\max}}\right)$ is a screening region for any $\lambda < \lambda_{\max}$. However it is easy to see that a small enough λ results in an empty test for the basic SAFE procedure in El Ghaoui et al. (2010), as no feature meets it (Fercoq et al., 2015). The same is true for the basic strong rule of Tibshirani et al. (2012), that for unit norm features $\mathbf{x}_j$ screens out those j for which $|\mathbf{x}_j \cdot \mathbf{y}| < 2\lambda - \lambda_{\max}$. The set of values that pass the previous test becomes empty if $\lambda < \lambda_{\max}/2$. Another drawback is that it may incorrectly screen out a feature, as it is also a sphere test over $B(\mathbf{y}/\lambda, R)$ for an appropriate R but $\boldsymbol{\theta}_\lambda$ may not lie in that sphere; see Xiang et al. (2017, Sect. 4.2.2).

The usual way to get more precise regions is to work in a regularization path setting where solutions $\boldsymbol{\theta}_{\lambda_k}$ are successively computed over a sequence $\lambda_{\max} = \lambda_0 > \lambda_1 > \ldots$ and (recursive) SAFE (El Ghaoui et al., 2010) or (sequential) strong rules (Tibshirani et al., 2012) are applied when computing $\boldsymbol{\theta}_{\lambda_{k+1}}$ once $\boldsymbol{\theta}_{\lambda_k}$ is known. In fact, there has been a substantial number of recent contributions (Wang et al., 2013; Liu et al., 2014; Bonnefoy et al., 2015) with great promise and in some cases (for instance Bonnefoy et al., 2015) with publicly available Python or Matlab implementations. However, note that full regularization paths have to be found when looking for an optimal λ, but not so for the repeated construction of models with a previously fixed λ.

Chapter 4

Theory and algorithms for Support Vector Machines

4.1 Support Vector Classification

Support Vector Machines (SVM) are a classification method capable of dealing with high dimensional non-linear data. SVMs belong to the class of algorithms known as *kernel methods*, since they depend on data only through dot-products. Therefore, they can be replaced by kernel functions, that compute these products in another feature space, different from the input space, and possibly with higher dimension. This presents two main advantages:

1. The ability to generate non-linear decision functions using methods designed for linear classifiers.
2. The classifier can be applied to data that do not have a representation in a fixed dimension space.

4.1.1 Hard-margin SVC

Given a training set $\{\mathbf{x}_i, y_i\}$, $i = 1, \ldots, n$, $y_i \in \{-1, 1\}$, $\mathbf{x}_i \in \mathbb{R}^d$, let's assume that there is an hyperplane that separates positive samples from the negative ones. The points $\mathbf{x}$ that are on the hyperplane satisfy $\mathbf{w}^\top \mathbf{x} + b = 0$, where $\mathbf{w}$ is normal to the hyperplane, $|b|/\|\mathbf{w}\|_2$ is the distance perpendicular from the hyperplane to the origin and $\|\mathbf{w}\|_2$ is the Euclidean norm of $\mathbf{w}$. Let d_+ (d_-) be the distance from the hyperplane to the closest positive (negative) point. Then, we define the margin of the hyperplane as $d_+ + d_-$. For the linearly separable case, the SVM simply tries to find the maximum-margin separating hyperplane. This can be formulated more formally as follows: assume that the points satisfy the restrictions

$$\mathbf{x}_i^\top \mathbf{w} + b \geq +1 \quad \text{for} \quad y_i = +1 \tag{4.1.1}$$

$$\mathbf{x}_i^\top \mathbf{w} + b \leq -1 \quad \text{for} \quad y_i = -1 \tag{4.1.2}$$

These can be combined in

$$y_i(\mathbf{x}_i^\top \mathbf{w} + b) - 1 \geq 0 \quad \forall i. \tag{4.1.3}$$

The points that satisfy the equality constraint in Eq. (4.1.1) are on the hyperplane $\mathbf{x}_i^\top \mathbf{w} + b = 1$ and the points that satisfy the equality constraint in Eq. (4.1.2) are on the hyperplane $\mathbf{x}_i^\top \mathbf{w} + b = -1$.

If the data is linearly separable, these hyperplanes separate the data perfectly and there are no points between them. The distance from this hyperplanes to $\mathbf{x}_i^\top \mathbf{w} + b = 0$ is $d_+ = d_- = 1/\|\mathbf{w}\|_2$

and the margin defined above is simply $2/\|\mathbf{w}\|_2$. Thus, the maximum-margin hyperplane can be found by minimizing $\|\mathbf{w}\|_2^2$ subject to the constraints (4.1.3)

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{s.t} \quad & y_i(\mathbf{x}_i^\top \mathbf{w} + b) - 1 \geq 0 \quad \forall i, \end{aligned} \tag{4.1.4}$$

where the $1/2$ is introduced for convenience. Now lets transform the previous problem into the dual equivalent. The objective function (4.1.4) is clearly convex and the restrictions are affine, therefore the Lagrangian of the problem is

$$L_P = \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^{n} \alpha_i y_i (\mathbf{x}_i^\top \mathbf{w} + b) + \sum_{i=1}^{n} \alpha_i, \tag{4.1.5}$$

with $\alpha_i \geq 0$, $\forall i$. The dual problem is thus defined as

$$\max_{\boldsymbol{\alpha}} \left\{ \min_{\mathbf{w},b} L_P \right\} \quad \text{s.t.} \quad \boldsymbol{\alpha} \geq 0 \tag{4.1.6}$$

Next we have to minimize L_P with respect to $\mathbf{w}$ and b, requiring also that the derivatives of L_P with respect to α_i are 0 and everything under the constraints $\alpha_i \geq 0$. The derivatives of L_P with respect to $\mathbf{w}$ and b are

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{4.1.7}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i. \tag{4.1.8}$$

As we have seen in Theorem 2.4, the optimum $(\mathbf{w}^*, b^*)$ must satisfy the following KKT conditions:

$$\mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0, \tag{4.1.9}$$

$$\sum_i \alpha_i y_i = 0, \tag{4.1.10}$$

$$y_i(\mathbf{x}_i^\top \mathbf{w} + b) - 1 \geq 0 \quad \forall i, \tag{4.1.11}$$

$$\alpha_i \geq 0 \quad \forall i, \tag{4.1.12}$$

$$\alpha_i(y_i(\mathbf{x}_i^\top \mathbf{w} + b) - 1) = 0 \quad \forall i. \tag{4.1.13}$$

Equation (4.1.13) is called the KKT dual complementary condition, and it will be important later to show that the optimal solution of the SVC problem depends only on the coefficients $\alpha_i^* > 0$, which are called the **support vectors**. Substituting Eqs. (4.1.7) and (4.1.8) in Eq. (4.1.5) we get

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j. \tag{4.1.14}$$

Therefore the new dual problem is defined as

$$\begin{aligned} \max_{\alpha} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ \text{s.t} \quad & \alpha_i \geq 0 \quad \forall i, \\ & \sum_i \alpha_i y_i = 0. \end{aligned} \tag{4.1.15}$$

The dual problem is much easier to solve since the constraints are simpler. Note that Eq. (4.1.9) relates the solutions of the primal and dual problem. Therefore, if we were to solve the dual formulation we could recover the optimal primal weights as

$$\mathbf{w}^* = \sum_i \alpha_i^* y_i \mathbf{x}_i$$

where $\boldsymbol{\alpha}^* \in \mathbb{R}^n$ is the solution of problem (4.1.15). Similarly, Eq. (4.1.13) implies that,

$$y_i(\mathbf{x}_i^\top \mathbf{w} + b^*) - 1 = 0$$

for any i such that $\alpha_i^* > 0$. From that we can obtain the optimal value of the bias as

$$b^* = \frac{1}{y_i} - \mathbf{x}_i^\top \mathbf{w}^* = y_i - \mathbf{x}_i^\top \mathbf{w}^*,$$

since $y_i \in \{-1, 1\}$. Although the previous equation should yield the same value for every support vector i, in practice it is recommended for numerical stability to compute the bias as the average among all of them,

$$b^* = \frac{1}{|S|} \sum_{i \in S} y_i - \mathbf{x}_i^\top \mathbf{w}^*$$

where $S = \{i \mid \alpha_i^* > 0\}$, that is, the set of indices of the support vectors.

In the following section we are going to introduce the most common case in practice, that is, when data is not linearly separable.

4.1.2 Soft-margin SVC

In order to allow classification errors when data is not linearly separable, we replace the constraints (4.1.3) by

$$y_i(\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 - \xi_i \quad \forall i, \tag{4.1.16}$$

where $\xi_i \geq 0$ are slack variables that make possible for a point to be inside the margin ($0 \leq \xi_i \leq 1$) or to be misclassified ($\xi > 1$). Since a data point is wrongly classified if the value of its slack variable is larger than 1, $\sum_i \xi_i$ is an upper bound on the number of classification errors. The new goal is to maximize the margin i.e. minimize $\|\mathbf{w}\|_2^2$ but penalizing classification errors with the term $C \sum_i \xi_i$. The parameter $C > 0$ controls the tradeoff between margin maximization and error minimization. The optimization problem is now

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C \sum_i \xi_i \\ \text{s.t} \quad & y_i(\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 - \xi_i \quad \forall i \\ & \xi_i \geq 0 \quad \forall i. \end{aligned} \tag{4.1.17}$$

The dual formulation of this problem is obtained similarly to the linearly separable case. The Langrange dual function is

$$L_P = \frac{1}{2}\|\mathbf{w}\|_2^2 + C \sum_i \xi_i - \sum_i \alpha_i (y_i(\mathbf{x}_i^\top \mathbf{w} + b) - 1 + \xi_i) - \sum_i \beta_i \xi_i \tag{4.1.18}$$

where $\boldsymbol{\alpha} \geq 0 \in \mathbb{R}^n$ and $\boldsymbol{\beta} \geq 0 \in \mathbb{R}^n$ are the Lagrange dual variables. The KKT conditions are,

$$0 = \frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \tag{4.1.19}$$

$$0 = \frac{\partial L_P}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i, \tag{4.1.20}$$

$$0 = \frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \beta_i. \tag{4.1.21}$$

Substituting the previous equations into the dual function yields the equivalent dual problem

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ \text{s.t} \quad & 0 \leq \alpha_i \leq C \quad \forall i \\ & \sum_i \alpha_i y_i = 0. \end{aligned} \tag{4.1.22}$$

We can also write the previous problem more compactly in vector form

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} - \boldsymbol{\alpha}^\top \mathbf{1} \right\} \quad \text{s.t.} \quad \boldsymbol{\alpha}^\top \mathbf{y} = 0,\ 0 \leq \alpha_i \leq C,\ i = 1, \ldots, n, \tag{4.1.23}$$

where $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is a matrix whose entries are $Q_{ij} = y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$ and $\mathbf{1} \in \mathbb{R}^n$ is a vector of ones. Note how we transformed the maximization problem into a minimization problem just by flipping the sign of the objective function.

4.1.3 Kernel trick

Despite extending SVMs to deal with non-separable data, the decision function is still linear, that is, the surface that separates both classes is an hyperplane. Most problems in practice are in fact nonlinear and thus it is useful to extend the SVM formulation for this type of problems. The nonlinear SVM is based on the kernel trick explained next.

The kernel trick comes from the idea of transforming the input variables into another set of features in a higher dimensional space, where the data points are linearly separable. More formally, let $\phi(\mathbf{x}_i)$ be a function that takes a point in a d-dimensional space and returns another point in a D-dimensional space, $D >> d$. If ϕ is choosen correctly then we obtain another problem that is linearly separable in this new feature space.

However, if we try to compute explicitly the value of $\phi(\mathbf{x}_i)$, we run into two main problems, one of them practical and another one theoretical:

1. The feature space can have a very large dimension, even infinite.

2. It can be very computationally expensive to compute the mapping values every time they are needed, or even storing them in memory.

The kernel trick comes from the observation that the dual function (4.1.22) only depends on $\mathbf{x}$ through dot products, that is, they always appear in pairs. If we define the kernel function as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j), \tag{4.1.24}$$

the dual problem can be rewritten as

$$\begin{aligned} \max_{\alpha} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s.t} \quad & 0 \leq \alpha_i \leq C \quad \forall i, \\ & \sum_i \alpha_i y_i = 0. \end{aligned} \tag{4.1.25}$$

Note that the compact formulation (4.1.23) does not change, only the definition of the matrix $\mathbf{Q}$, $Q_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$. We will refer to the matrix $\mathbf{Q}$ as **kernel matrix**.

Therefore it is not neccessary to know the mapping function ϕ but only the kernel function $k(\cdot, \cdot)$. The only condition is that the kernel function must be decomposed into a dot product of two functions. Mercer's theorem states which types of kernels can be used, although in practice there are a few known kernels that met Mercer's condition and they are the most used. An example is the RBF kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2\right). \tag{4.1.26}$$

where γ is a hyper-parameter that represents the width of the kernel.

Last, the value of $\mathbf{w}$ in the transformed space is

$$\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)$$

and usually it cannot be computed explicitly. However, a new point $\mathbf{x}$ can be classified using again the kernel trick

$$\mathbf{w}^\top \phi(\mathbf{x}) = \sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}). \tag{4.1.27}$$

Note also that only patterns with $\alpha_i > 0$ (support vectors) are needed for the classification of a new point, which is a nice property.

The selection of the hyper-parameters is a big problem, since the SVM performance depends greatly on their value. For the kernel SVC formulation we need to select both the complexity parameter C and the parameters of the kernel function. In the case of the RBF kernel we only have one additional hyper-parameter, namely, the width of the kernel γ.

4.1.4 ν-Support Vector Classification

As we mentioned before, given a particular problem, it is not straightforward to select the best value for the hyper-parameter C. In general, C controls the trade-off between minimizing the errors and maximizing the margin. Thus, as C grows, the complexity of the hypothesis space also grows and the SVC becomes more intolerant to errors. Relating this to Section 2.1.4, if C is too large the model could exhibit overfitting. On the other hand, if C is very small the model could suffer from underfitting. In the extreme case where $C = 0$ the soft-margin SVC reverts to the hard-margin case, and the model only cares about maximizing the margin.

In practice, the most common method to set the C hyper-parameter is to perform cross-validation over a grid of values and select the one with the lowest error. However this computation is quite expensive since, for a given dataset we would have to train many models just to find a good value for C, with no guarantees of optimality. To alleviate this problem, Schölkopf et al. (2000) suggest a new SVC formulation which replaces the C paramater by $\nu \in (0, 1]$ which is in

principle easier to tune. The primal optimization problem of the ν-SVC is

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 - \nu\rho + \frac{1}{n}\sum_{i=1}^{n}\xi_i \\ \text{s.t} \quad & y_i(\mathbf{w}^\top\phi(\mathbf{x}_i) + b) \geq \rho - \xi_i, \\ & \xi_i \geq 0,\ i = 1,\ldots,n, \quad \rho \geq 0. \end{aligned} \tag{4.1.28}$$

And the corresponding dual problem is (Schölkopf et al., 2000)

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{Q}\boldsymbol{\alpha} \\ \text{s.t} \quad & \boldsymbol{\alpha}^\top\mathbf{y} = 0, \\ & \boldsymbol{\alpha}^\top\mathbf{1} \geq \nu, \\ & 0 \leq \alpha_i \leq 1/n,\ i = 1,\ldots,n. \end{aligned} \tag{4.1.29}$$

Compared to the original SVC dual (4.1.23), there are two differences. First, there is an additional constraint, involving the new hyper-parameter ν. Second, the linear term $\boldsymbol{\alpha}^\top\mathbf{1}$ has disappeared from the objective function.

Regarding the new hyper-parameter ν, Schölkopf et al. (2000) prove that it is an upper-bound on the fraction of margin errors and a lower bound on the fraction of SVs. Thus, ν has an intrinsic meaning and it is more easy to interpret its value compared to C. Chang and Lin (2011) also give a tighter range for the feasible values of ν,

$$\nu \leq \frac{2}{n}\min\left(|\{i \mid y_i = 1\}|, |\{i \mid y_i = -1\}|\right) \leq 1,$$

so the usable range of ν in practice is actually smaller than $(0, 1]$.

4.1.5 One-class Support Vector Machine

The One-class SVM was proposed by Schölkopf et al. (2001) for estimating the support of a high-dimensional distribution. Another application of this formulation is the problem of novelty detection, which is often reduced to estimating the density of the probability of the data. Therefore the SVM classifies examples as "novel" if the density function has high probability and viceversa. The primal problem is

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i \\ \text{s.t} \quad & \mathbf{w}^\top\phi(\mathbf{x}_i) \geq \rho - \xi_i, \\ & \xi_i \geq 0,\ i = 1,\ldots,n, \quad \rho \geq 0. \end{aligned} \tag{4.1.30}$$

and the corresponding dual problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{Q}\boldsymbol{\alpha} \\ \text{s.t} \quad & \boldsymbol{\alpha}^\top\mathbf{1} = 1, \\ & 0 \leq \alpha_i \leq 1/(\nu n),\ i = 1,\ldots,n. \end{aligned} \tag{4.1.31}$$

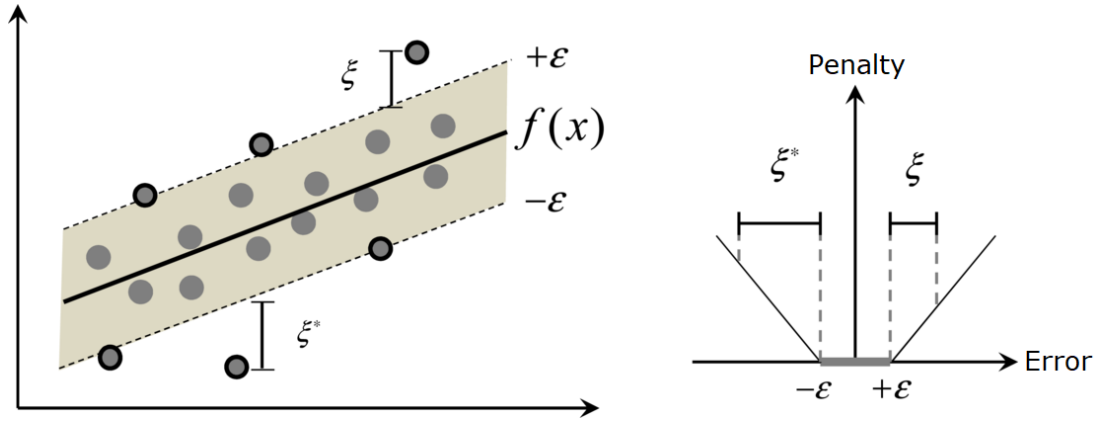

Figure 4.1: Linear Support Vector Regression (left) and ϵ-insensitive loss (right). Support vectors are drawn with a black outline (Yu et al., 2014)

4.2 Support Vector Regression

The full kernel SVM problem (4.1.25) is a binary classfication problem, since the possible values for the targets are either -1 or $+1$. However, the SVM formulation can be extended in order to solve regression problems, and we are going to do so in this section.

SVMs applied to a regression problem perform a linear regression in the feature space using the ϵ-insensitive loss as cost function and, at the same time, regularizing the weights so large values are penalized (more on regularization in Chapter 1). It is important to note that the feature space is different from the input space, and thus depending on the kernel the problem could be nonlinear on the input variables. The ϵ-insensitive loss is defined as

$$L_\epsilon(y, f(\mathbf{x}, \mathbf{w})) = \begin{cases} 0 & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \leq \epsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \epsilon & \text{otherwise} \end{cases} . \tag{4.2.1}$$

where $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$ denotes the linear model in the feature space. The previous function is plotted in Fig. 4.1 (right).

Let $\mathbf{y} \in \mathbb{R}^n$ be now continuous target values. The regression SVM solves the following optimization problem

$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}'} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i') \\ \text{s.t} \quad & y_i - f(\mathbf{x}_i, \mathbf{w}) - b \leq \epsilon + \xi_i', \\ & f(\mathbf{x}_i, \mathbf{w}) + b - y_i \leq \epsilon + \xi_i, \\ & \xi_i, \xi_i' \geq 0, \ i = 1, \ldots, n, \end{aligned} \tag{4.2.2}$$

where the parameter the parameter ϵ controls the width of the ϵ-*insensitive* tube and the parameter C determines the tradeoff between model complexity and the amount of deviations larger than ϵ allowed.

The objective function of the ϵ-SVR is illustrated in Fig. 4.1 (left). Only the points outside the ϵ-insensitive tube, depicted with a black outline, are penalized and contribute to the solution of problem (4.2.2) (support vectors). This penalty is proportional to the distance to the ϵ-tube. On the other hand, errors in the interval $[-\epsilon, \epsilon]$ are ignored. It can be shown that a good value for ϵ has to be proportional to the input noise level, i.e. $\epsilon \propto \sigma$ (Cherkassky and Ma, 2004).

The dual of problem (4.2.2) can be computed similarly to the classification case, arriving at

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}'} \quad & \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}')^\top \mathbf{Q}(\boldsymbol{\alpha}-\boldsymbol{\alpha}') + \epsilon \sum_{i=1}^{n}(\alpha_i+\alpha_i') + \sum_{i=1}^{n} y_i(\alpha_i-\alpha_i') \\ \text{s.t} \quad & (\boldsymbol{\alpha}-\boldsymbol{\alpha}')^\top \mathbf{1} = 0, \\ & 0 \leq \alpha_i, \alpha_i' \leq C,\ i = 1,\dots,n. \end{aligned}$$

where $Q_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) := \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$. It will be useful in the following sections to rewrite the previous problem into a more compact form. Let us define

$$\begin{aligned} \widetilde{\boldsymbol{\alpha}} &= \begin{bmatrix} \boldsymbol{\alpha}' \\ \boldsymbol{\alpha} \end{bmatrix} \in \mathbb{R}^{2n}, \\ \widetilde{\mathbf{Q}} &= \begin{bmatrix} \mathbf{Q} & -\mathbf{Q} \\ -\mathbf{Q} & \mathbf{Q} \end{bmatrix} \in \mathbb{R}^{2n\times 2n}, \\ \mathbf{p} &= \begin{bmatrix} \epsilon\mathbf{1}_n - \mathbf{y} \\ \epsilon\mathbf{1}_n + \mathbf{y} \end{bmatrix} \in \mathbb{R}^{2n}, \\ \widetilde{\mathbf{y}} &= [1,\dots,1,-1,\dots,-1]^\top \in \mathbb{R}^{2n}. \end{aligned}$$

Then it can be shown that problem (4.2.3) is equivalent to the following:

$$\begin{aligned} \min_{\widetilde{\boldsymbol{\alpha}}} \quad & \frac{1}{2}\widetilde{\boldsymbol{\alpha}}^\top \widetilde{\mathbf{Q}}\widetilde{\boldsymbol{\alpha}} + \mathbf{p}^\top \widetilde{\boldsymbol{\alpha}} \\ \text{s.t} \quad & \widetilde{\boldsymbol{\alpha}}^\top \widetilde{\mathbf{y}} = 0, \\ & 0 \leq \widetilde{\alpha}_i \leq C,\ i = 1,\dots,2n. \end{aligned} \tag{4.2.3}$$

Note how formulations (4.1.23) and (4.2.3) share a lot of similarities, simply by assigning in the first case $\mathbf{p} = \mathbf{1} \in \mathbb{R}^n$. Thus SVR can be seen as a classification problem with carefully chosen artificial labels and in an enlarged space.

Again, the selection of the hyper-parameters is even a bigger problem, since now we have three hyper-parameters to tune instead of two. One possible option is to use cross-validation over a 3D grid of (C, σ, ϵ) values, selecting the tuple with the lowest generalization error.

4.3 Algorithms

In this section we are going to review the main ideas to solve the SVM problem. In what follows we will usually write SVM to refer to the C-SVC problem, which is the most common and well known formulation. Nonetheless, many of these solvers can also be extended to the regression setting (ϵ-SVR) but not to other formulations like the ν-SVM, which are only considered in LIBSVM (Chang and Lin, 2011).

SVM solvers can be broadly classified into primal solvers and dual solvers. Historically, SVM solvers usually tackle the dual problem since the constrains are simpler and they can be extended seamlessly to the kernel setting, but they usually scale poorly with the sample size n. Popular dual solvers are SVM-Light (Joachims, 1998), SVM-Perf (Joachims, 2006; Joachims and Yu, 2009), LIBLINEAR (Fan et al., 2008) and LIBSVM (Chang and Lin, 2011). If we wanted to solve the primal problem directly we could in principle absorb the constraints into the objective function, similarly to the Lasso problem. However, this only works for the linear SVM, since

the primal problem of kernel SVMs involves the non-linear functions ϕ_i, which may not even be available for some common kernels like the RBF kernel.

Although the primal of the kernel SVM can not be solved in its usual formulation (4.3.1), the *representer theorem* (Kimeldorf and Wahba, 1971) allows us to re-parametrize the weights $\mathbf{w}$ and tackle the primal objective directly (Chapelle, 2007). This formulation does not depend directly on the functions ϕ_i, has no constraints and can handle kernels. Another advantage of the primal formulation is that is easy to safely stop the algorithm early to obtain an approximate solution. However, when working with the dual there are no guarantees that an ϵ-optimal solution of the dual problem corresponds to an ϵ-optimal solution of the primal problem, and thus more sophisticated stopping conditions are needed. Some examples of primal solvers include NORMA (Kivinen et al., 2004), SGD-SVM (Bousquet and Bottou, 2008; Bottou, 2010) and Pegasos (Shalev-Shwartz et al., 2007).

In the following sections we are going present what we believe are the main algorithms to solve the SVM problem, not only in terms of popularity but also in quality of the software packages implementing them and reported efficiency. One of them is a primal solver, that works for both the linear and non-linear SVMs (Pegasos), and the other two are dual solvers, both from the same authors, one solving only the linear SVM (LIBLINEAR) and the other one solving both linear and nonlinear SVMs (LIBSVM). The former implements stochastic dual coordinate descent while the latter uses greedy dual block coordinate descent with the smallest block size possible (two coefficients). Note that, in general, even though LIBSVM solves both formulations it works specially well for the nonlinear one. Therefore, in general, either LIBLINEAR or Pegasos should be used to solve the linear SVM .

4.3.1 Primal gradient methods

The primal formulation of the SVM, (4.1.17) can be rewritten as an unconstrained minimization problem

$$\min_{\mathbf{w}} \sum_{i=1}^{n} [1 - y_i f(\mathbf{x}_i)]_+ + \frac{\lambda}{2}\|\mathbf{w}\|_2^2, \tag{4.3.1}$$

where $f(\mathbf{x}) = \phi(\mathbf{x})^\top \mathbf{w} + b$ and the subscript "+" indicates the positive part. This has the form $loss + penalty$ (Eq. (2.1.14)) and it can be shown that the solution to (4.3.1), with $\lambda = \frac{1}{C}$, is the same as that for (4.1.17) (Hastie et al., 2003).

The loss function $L(y, f(x)) = [1 - yf(x)]_+$ is known as the "hinge" loss and it is a particular case of the ϵ-insensitive loss (4.2.1) used for SVR with $\epsilon = 0$.Figure 4.2 shows that it is reasonable for two-class classification, when compared with the logistic regression loss function.

As we mentioned before, the previous unconstrained problem depends on the non-linear functions $\phi_i(\cdot)$, which sometimes can not be computed explicitely. Thus, in order to extend the primal formulation to handle kernels, Chapelle (2007) considers first the equivalent problem

$$\min_{f \in \mathcal{H}} \ \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2 + \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)),$$

where $\mathcal{H}$ is a reproducing kernel Hilbert space (RKHS) with associated kernel k and L is any loss function, for instance the hinge-loss. Then, using the *representer theorem* (Kimeldorf and Wahba, 1971), the previous problem can also be written as

$$\min_{\boldsymbol{\alpha}} \ \lambda \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} + \frac{1}{n}\sum_{i=1}^{n} L(y_i, K_i^\top \boldsymbol{\alpha}) \tag{4.3.2}$$

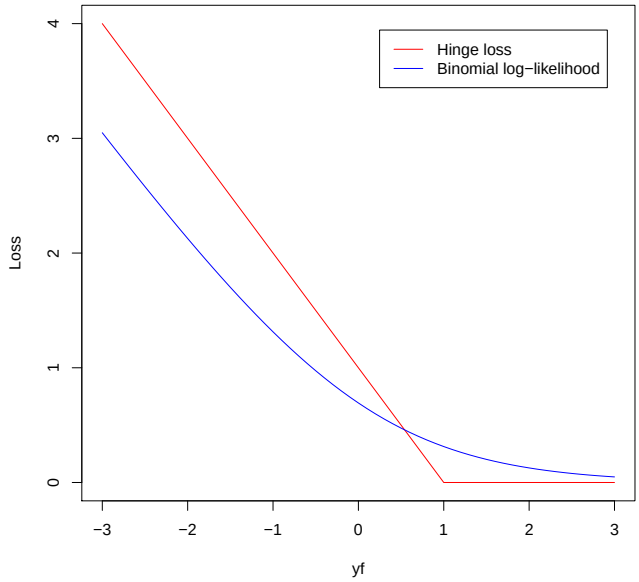

Figure 4.2: Support Vector Machines loss function (hinge loss) compared to the logistic regression loss (negative log-likelihood loss). They are shown as a function of yf rather than f because of the symmetry between the $y = +1$ and $y = -1$ case (Hastie et al., 2003).

where K is the kernel matrix and K_i the ith column of K. Problem (4.3.2) is quadratic, unconstrained and no longer depends on the non-linear functions ϕ_i. However, note that the bias term b is dropped from the calculations for simplicity, but they can be easily modified to take it into account (Chapelle, 2007, Appendix B).

The previous problem can be solved by standard techniques like gradient descent, conjugate gradient or Newton's method, as long as L is differentiable. Since the hinge-loss is not differentiable, Chapelle (2007) considers first the squared hinge-loss

$$L(y_i, f(\mathbf{x}_i)) = \max(0,\, 1 - y_i f(\mathbf{x}_i))^p, \tag{4.3.3}$$

with $p = 2$. This is sometimes referred as L2-SVM, in contrast to the L1-SVM, where $p = 1$. Another option also suggested by Chapelle (2007) is to approximate the hinge-loss by the Huber loss

$$L(y, t) = \begin{cases} 0 & \text{if } yt > 1 + h, \\ \frac{(1+h-yt)^2}{4h} & \text{if } |1 - yt| \leq h, \\ 1 - yt & \text{if } yt < 1 - h, \end{cases} \tag{4.3.4}$$

where h is a parameter to choose, typically between 0.01 and 0.5 (Chapelle, 2007). A similar approach is considered by Shalev-Shwartz et al. (2007), but instead of covering only smooth functions they cope with the non-differentiability of the hinge-loss by using subgradients instead of gradients.

For linear SVMs the Pegasos algorithm (Shalev-Shwartz et al., 2007) is a straight application of stochastic gradient descent, which we also considered to solve the Lasso problem (Section 3.3.5). The Lasso is the sum of a differentiable loss function and a convex regularizer, while the SVM is the other way around, that is, we have a differentiable regularizer (ℓ_2-norm) but a non-differentiable loss (hinge-loss).

Algorithm 10: Pegasos for the linear SVM

Input : $\mathbf{w} = 0 \in \mathbb{R}^d$ and λ
for $k = 1, 2, \ldots$ **do**
  Sample i uniformly at random from $\{1, \ldots, n\}$
  $\eta_k = 1/\lambda k$
  if $y_i \mathbf{x}_i^\top \mathbf{w}^k < 1$ **then**
    $\mathbf{w}^{k+1} \leftarrow (1 - \eta_k \lambda)\mathbf{w}^k + \eta_k y_i \mathbf{x}_i$
  else
    $\mathbf{w}^{k+1} \leftarrow (1 - \eta_k \lambda)\mathbf{w}^k$
  end
end

Every iteration Pegasos selects an index i uniformly at random and considers the following approximation of the SVC objective function:

$$f(\mathbf{x}_i, y_i) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + L(y_i, \mathbf{x}_i^\top \mathbf{w}),$$

where $L(y_i, t) = \max(0,\, 1 - y_i t)$. The subgradient of the hinge-loss is

$$\partial_k L(y_i, t) = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i t < 1 \\ 0 & \text{otherwise,} \end{cases} \tag{4.3.5}$$

and thus the subgradient of the approximate objective is

$$\partial_k f(\mathbf{x}_i, y_i) = \lambda \mathbf{w} - \mathbf{s},$$

where $\mathbf{s} \in \partial_k L$. Let $\eta_k = 1/(\lambda k)$ be a step-size; the weight updates of stochastic subgradient descent can be written as

$$\begin{aligned} \mathbf{w}^{k+1} &= \mathbf{w}^k - \eta_k \partial_k f(\mathbf{x}_i, y_i) \\ &= \mathbf{w}^k - \eta_k(\lambda \mathbf{w}^k + \mathbf{s}) \\ &= (1 - \eta_k \lambda)\mathbf{w}^k - \eta_k \mathbf{s}. \end{aligned}$$

The pseudo-code of Pegasos is shown in Algorithm 10.

Theorem 1 from Shalev-Shwartz et al. (2007) shows that Algorithm 10 finds an ϵ-optimal solution with high probability over the choice of the random examples using only $\tilde{\mathcal{O}}\left(1/(\lambda\epsilon)\right)$ iterations. Since each iteration involves one dot product between $\mathbf{w}$ and $\mathbf{x}_i$, the total runtime of the algorithm is

$$\tilde{\mathcal{O}}\left(\frac{d}{\lambda\epsilon}\right). \tag{4.3.6}$$

Pegasos can also be extended to the nonlinear case using the representer theorem. This theorem let us write $\mathbf{w}$ as a linear combination of the training examples

$$\mathbf{w} = \sum_j \alpha_j y_j \phi(\mathbf{x}_j). \tag{4.3.7}$$

For each k, let $\boldsymbol{\alpha}^k \in \mathbb{R}^n$ be the vector such that α_j^k counts how many times example j has been selected so far and it had a non-zero loss, namely,

$$\alpha_j^k = \left|\left\{k' \leq k \;\middle|\; i = j \;\text{ and }\; y_j \phi(\mathbf{x}_j)^\top \mathbf{w}^{k'} < 1\right\}\right|.$$

Algorithm 11: Pegasos for the nonlinear SVM

Input : $\boldsymbol{\alpha} = 0 \in \mathbb{R}^d$ and λ
for $k = 1, 2, \ldots$ **do**
  Sample i uniformly at random from $\{1, \ldots, n\}$
  forall $j \neq i$ **do**
    $\alpha_j^{k+1} = \alpha_j^k$
  end
  if $y_i \frac{1}{\lambda k} \sum_j \alpha_j^k y_i k(\mathbf{x}_i, \mathbf{x}_j) < 1$ **then**
    $\alpha_i^{k+1} = \alpha_i^k + 1$
  else
    $\alpha_i^{k+1} = \alpha_i^k$
  end
end

Although we cannot compute the feature map ϕ directly, note that in Algorithm 10 we only perform two operations on $\mathbf{w}$: scaling by a constant and dot-product. This last operation can be written **only** in terms of kernel evaluations,

$$\mathbf{w} = \sum_j \alpha_j y_j \phi(\mathbf{x}_j)$$
$$\mathbf{w}^\top \phi(\mathbf{x}_i) = \sum_j \alpha_j y_j \phi(\mathbf{x}_j)^\top \phi(\mathbf{x}_i)$$
$$\mathbf{w}^\top \phi(\mathbf{x}_i) = \sum_j \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i)$$

Thus, instead of keeping in memory the vector $\mathbf{w}$, nonlinear Pegasos maintains the vector $\boldsymbol{\alpha}$. The pseudocode of the implementation of Pegasos for nonlinear SVMs is shown in Algorithm 11.

Since the iterates remain the same and only the representation of the $\mathbf{w}$ changes, the convergence results for the linear case are still valid. However, the runtime complexity to obtain an ϵ-accurate solution is now

$$\tilde{\mathcal{O}}\left(\frac{n}{\lambda\epsilon}\right).$$

As pointed out by Shalev-Shwartz et al. (2007), even though the solution is represented in terms of the variables $\boldsymbol{\alpha}$, the subgradient is still computed with respect to the original weights $\mathbf{w}$. In contrast the approach from Chapelle (2007) is to rewrite the primal problem as a function of $\boldsymbol{\alpha}$ (4.3.2) and then taking gradients with respect to $\boldsymbol{\alpha}$.

The previous discussion only considers the hinge-loss. However Pegasos can be adapted to any other convex losses with the same convergence guarantees (see Shalev-Shwartz et al., 2007, for more details). An important example is the ϵ-insensitive loss used in SVRs, $L(y_i, t) = \max(0, |y_i - t| - \epsilon)$. The subgradient is

$$\partial_k L(y_i, t) = \begin{cases} \mathbf{x}_i & \text{if } t - y_i > \epsilon \\ -\mathbf{x}_i & \text{if } y_i - t > \epsilon \\ 0 & \text{otherwise.} \end{cases}$$

Finally, note that we have ignored the bias term b, since it makes the analysis much easier. Extending Pegasos to include this term is not that simple, since the two trivial extensions both have important drawbacks. We show next four different approaches (Shalev-Shwartz et al., 2007), pointing out potential disadvantages:

1. Add an artificial constant feature to the data and incorporate the bias as a new dimension into the weight vector $\mathbf{w}$. The disadvantage of this method is that b is also regularized and thus we are solving a slightly different optimization problem.

2. The bias can be incorporated in the loss function and in the analysis, obtaining the same algorithm since the subgradient does not change. However the objective function ceases to be strongly convex and thus the convergence rate is much slower.

3. We can consider a mini-batch version of Pegasos, able to incorporate the bias with the same convergence rate but at the price of a more complex algorithm (see Shalev-Shwartz et al., 2007, for more details).

4. We can search over the bias term b in an external loop, optimizing the weight vector $\mathbf{w}$ for different values of b. For a fixed b the optimization can be performed using Pegasos and the b search can be done by binary search using $\mathcal{O}(\log 1/\epsilon)$ evaluations. This method maintains the same runtime bound, although it is clear that in practice the iterations are more expensive.

4.3.2 Dual coordinate methods

The ideas behind the previous algorithms could be extended to the dual SVM. If our aim is to solve very large linear SVM problems and we ignore the bias term b, problem (4.1.23) becomes

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top \mathbf{1} \right\} \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C,\ i = 1, \ldots, n, \tag{4.3.8}$$

where $Q_{ij} = y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$. Note that when removing the bias the constraint $\sum \alpha_i y_i = 0$ disappears. Hsieh et al. (2008) consider coordinate descent methods to solve the previous problem. We have already explored these kinds of procedures in Sections 3.3.3, 3.3.4 and 3.3.6. Coordinate descent picks an index i and optimizes the objective function with respect to the coefficient α_i only, keeping the rest fixed. Let $f(\boldsymbol{\alpha})$ be the objective function of the dual SVM (4.3.8), to update the coefficient α_i we solve the following one-variable sub-problem:

$$\min_{d} \ \{f(\boldsymbol{\alpha} + d\mathbf{e}_i)\} \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C. \tag{4.3.9}$$

Ignoring terms that do not depend on α_i, we get

$$f(\boldsymbol{\alpha} + d\mathbf{e}_i) = \frac{1}{2}Q_{ii}d^2 + \nabla_i f(\boldsymbol{\alpha})d + \text{Const.} = d\left(\frac{1}{2}Q_{ii}d + \nabla_i f(\boldsymbol{\alpha})\right) + \text{Const.}$$

Problem (4.3.9) has an optimum at $d = 0$ if and only if the projected gradient,

$$\nabla_i^P f(\boldsymbol{\alpha})_i = \begin{cases} \nabla_i f(\boldsymbol{\alpha}) & \text{if } 0 < \alpha_i < C, \\ \min(0, \nabla_i f(\boldsymbol{\alpha})) & \text{if } \alpha_i = 0, \\ \max(0, \nabla_i f(\boldsymbol{\alpha})) & \text{if } \alpha_i = C, \end{cases}$$

vanishes (Hsieh et al., 2008), and in that case α_i does not need to be updated. Otherwise, the unconstrained optimal d value is

$$d^* = \frac{-\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}$$

and the update on α_i is

$$\alpha_i^{k+1} = \min\left\{\max\left\{\alpha_i^k - \frac{\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}, 0\right\}, C\right\},$$

Algorithm 12: Dual coordinate descent for the Linear SVM

Input : $\boldsymbol{\alpha} = 0 \in \mathbb{R}^n$, $\mathbf{w} = 0 \in \mathbb{R}^d$ and C
while $\boldsymbol{\alpha}$ *is not optimal* **do**
  Pick an index i from $\{1, \ldots, n\}$
  $g = y_i \mathbf{w}^\top \mathbf{x}_i - 1$
  $$p = \begin{cases} \min(g, 0) & \text{if } \alpha_i = 0, \\ \max(g, 0) & \text{if } \alpha_i = C, \\ g & \text{if } 0 < \alpha_i < C \end{cases}$$
  if $|p| \neq 0$ **then**
    $\alpha_i' \leftarrow \alpha_i$
    $\alpha_i \leftarrow \min(\max(\alpha_i - g/Q_{ii}, 0), C)$
    $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \alpha_i')y_i \mathbf{x}_i$
  end
end

where $Q_{ii} = \mathbf{x}_i^\top \mathbf{x}_i$. Thus, Q_{ii} can be precomputed and stored in memory. The gradient of the objective function with respect to α_i is

$$\nabla_i f(\boldsymbol{\alpha}) = [\mathbf{Q}\boldsymbol{\alpha}]_i - 1 = \sum_{j=1}^{n} Q_{ij}\alpha_j - 1.$$

The matrix $\mathbf{Q}$ may be too large to be precomputed and stored in memory, so we need to compute n dot products to evaluate the gradient on the fly, for a total cost of $\mathcal{O}(n\bar{d})$, where $\bar{d}$ is the average of nonzero elements per instance. Such operation is expensive, with a worst case of $\mathcal{O}(nd)$ operations when the instances are dense.

However, since we are only solving the linear formulation, we have an explicit representation of the primal weights,

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i. \tag{4.3.10}$$

Thus, we can compute the gradient as

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \mathbf{w}^\top \mathbf{x}_i - 1$$

in $\mathcal{O}(\bar{d})$ operations, by maintaining both primal and dual weights. Again, the cost of computing (4.3.10) is $\mathcal{O}(n\bar{d})$, but we can update it in only $\mathcal{O}(\bar{d})$ as

$$\mathbf{w}^{k+1} = \mathbf{w}^k + (\alpha_i^{k+1} - \alpha_i^k) y_i \mathbf{x}_i.$$

This algorithm is implemented in the well-know software package LIBLINEAR (Fan et al., 2008). The pseudo-code is shown in Algorithm 12.

To compare the update of $\mathbf{w}$ with Stochastic Gradient Descent and Stochastic Coordinate Descent, we have to discuss first how to select the index i. As it was the case for the Lasso problem, there are several options: we can cycle through all the indices, cycle through a random permutation of the indices or pick an index uniformly at random. In the latter, the algorithm would be a stochastic dual coordinate descent method, very similar to the one we analyzed for the Lasso (Section 3.3.6). The main difference is that now we need to project the gradient and clip the coefficients in order to satisfy the box constraints.

The convergence of this method was analyzed by Hsieh et al. (2008), where they report a convergence rate to obtain an ϵ-optimal solution $|f(\boldsymbol{\alpha}) - f(\boldsymbol{\alpha}^*)| \leq \epsilon$ of $\mathcal{O}(\log(1/\epsilon))$ iterations. A recent result by Wang and Lin (2014) improves the constant hidden in the $\mathcal{O}$ notation. Since every iteration has a worst case complexity of $\mathcal{O}(d)$, the total runtime of the algorithm is

$$\mathcal{O}\left(d \log\left(\frac{1}{\epsilon}\right)\right), \tag{4.3.11}$$

which is better than the Pegasos bound (4.3.6). This was also confirmed empirically by the experiments in Fan et al. (2008), where they report LIBLINEAR to be faster than Pegasos to solve the linear SVM. Interestingly, they also mention that, although dual coordinate descent could also be modified to handle the nonlinear SVM, it is not efficient to do so since we cannot maintain an explicit representation of the primal weights. Thus, even if we keep the full gradient in memory, the updates

$$\nabla f(\boldsymbol{\alpha}^{k+1}) = \nabla f(\boldsymbol{\alpha}^k) + \mathbf{Q}_i(\alpha_i^{k+1} - \alpha_i^k)$$

require $\mathcal{O}(n\bar{d})$ operations. Thus decomposition methods, which we will review in the following section, are faster in that case.

4.3.3 Decomposition methods

The main difficulty of solving the dual problem (4.1.23) of the non-linear SVM is that computing the whole kernel matrix $\mathbf{Q}$ is very expensive, hence we do not want to do it at every iteration. Although $\mathbf{Q}$ only depends on the training data $\{\mathbf{X}, \mathbf{y}\}$, pre-computing the whole matrix may not be a very good idea (Bottou and Lin, 2007) for at least two reasons:

- It's wasteful: The expression of the gradient only depends on kernel values $k(\mathbf{x}_i, \mathbf{x}_j)$ that involve at least one support vector, and we can also recognize the optima using those kernel values only. In general, no more than 15% to 50% kernel values are actually needed to train the whole model.
- It may not fit into memory: Even if we were willing to spend time pre-computing the full kernel matrix, as n grows it may become prohibitively large and cannot be stored in memory.

Decomposition methods were designed to handle such difficulties. Some earlier works on decomposition methods for SVM include, for example, Osuna et al. (1997), Joachims (1998), Platt (1998), Keerthi et al. (2001), and Hsu and Lin (2002). Subsequent developments include, for example, Fan et al. (2005b), Palagi and Sciandrone (2005), and Glasmachers and Igel (2006). A good review can be found in Chang and Lin (2011). Popular software to train the SVM dual using decomposition methods include LIBSVM (Chang and Lin, 2011) and SVM-Light (Joachims, 1998).

Similarly to block coordinate descent, in decomposition methods only a subset of the coefficients $\boldsymbol{\alpha}$ are updated at every iteration, namely, the working set B. This subset leads to a small sub-problem being optimized at every iteration. More formally, let us define the complement of the working set, $N = \{1, \ldots, n\} \setminus B$. Then, in every iteration decomposition methods fix all the coefficients $\alpha_i \in N$ and optimize only the ones in the working set B,

$$\begin{aligned} \min_{\alpha_i,\, i \in B} \quad & \frac{1}{2} \sum_{i \in B} \sum_{j \in B} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i \in B} \sum_{j \in N} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i \in B} \alpha_i \\ \text{s.t} \quad & 0 \leq \alpha_i \leq C \quad i \in B, \\ & \sum_{i \in B} \alpha_i y_i + \sum_{i \in N} \alpha_i y_i = 0, \end{aligned}$$

where we have ignored some constant factors that only depend on the coefficients in the set N.

An extreme case is the Sequential Minimal Optimization (SMO) algorithm (Platt, 1998), which updates only two coefficients per iteration. Then, for the SVM dual problem it can be shown that this sub-problem can be solved analytically and thus no actual optimization is needed. SMO working set has only two elements $B = \{i, j\}$, so the previous problem is equivalent to

$$\begin{aligned} \min_{\alpha_i,\alpha_j} \quad & \frac{1}{2}\Big(\alpha_i^2 Q_{ii} + 2\alpha_i\alpha_j Q_{ij} + \alpha_u^2 Q_{jj}\Big) + \alpha_i \sum_{k\in N} Q_{ik}\alpha_k + \alpha_j \sum_{k\in N} Q_{jk}\alpha_k - \alpha_i - \alpha_j \\ \text{s.t} \quad & y_i\alpha_i + y_j\alpha_j = -\sum_{k\in N} y_k\alpha_k, \\ & 0 \le \alpha_i, \alpha_j \le C. \end{aligned} \tag{4.3.12}$$

Note that, in a given iteration, problem (4.3.12) only depends on the kernel rows $\mathbf{Q}_i$ and $\mathbf{Q}_j$. Those rows can be computed on-the-fly and thus there is no need to keep the full matrix in memory.

Suppose we are at iteration k and our current estimate of the coefficients is $\boldsymbol{\alpha}^k$. Then, SMO's update can be written as

$$\begin{aligned} \alpha_i^{k+1} &= \alpha_i^k + \delta_i, \\ \alpha_j^{k+1} &= \alpha_j^k + \delta_j, \\ \alpha_l^{k+1} &= \alpha_l^k,\ \forall l \neq i, j. \end{aligned}$$

Since the linear constraint must hold for the updated coefficients $\boldsymbol{\alpha}^{k+1}$ and assuming $\boldsymbol{\alpha}^k$ is feasible we have

$$\begin{aligned} \mathbf{y}^\top \boldsymbol{\alpha}^{k+1} &= 0 \\ \mathbf{y}^\top \boldsymbol{\alpha}^{k} + y_i\delta_i + y_j\delta_j &= 0 \\ y_i\delta_i + y_j\delta_j &= 0. \end{aligned}$$

Thus, for an update δ_i, δ_j to be feasible it must hold that

$$y_i\delta_i = -y_j\delta_j = \rho.$$

Since $y \in \{-1, 1\}$, we can parametrize both steps as a function of ρ,

$$\delta_i = y_i\rho \quad \text{and} \quad \delta_j = -y_j\rho,$$

arriving at the updates

$$\begin{aligned} \alpha_i^{k+1} &= \alpha_i^k + y_i\rho, \\ \alpha_j^{k+1} &= \alpha_j^k - y_j\rho, \\ \alpha_l^{k+1} &= \alpha_l^k,\ \forall l \neq i, j. \end{aligned}$$

or, in vector form,

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho(y_i\mathbf{e}_i - y_j\mathbf{e}_j) = \boldsymbol{\alpha}^k + \rho\mathbf{d}, \tag{4.3.13}$$

where $\mathbf{d} = y_i\mathbf{e}_i - y_j\mathbf{e}_j$ and $\mathbf{e}_k$ is the vector with all 0 but a 1 in the kth entry. The vector $\mathbf{d}$ is actually a descent direction and thus, in this context, ρ can be interpreted as a positive stepsize that gets us closer to the optimum by taking a step in the direction $\mathbf{d}$.

As a result, due to the equality constraint, we can write both updates in α_i and α_j as a function of a single parameter ρ. Therefore, we have effectively transformed the decomposition problem associated with the working set $B = \{i, j\}$ (4.3.12) into a new problem depending only on the stepsize ρ,

$$\begin{aligned} \min_{\rho} \quad & \frac{1}{2}(\boldsymbol{\alpha}^k + \rho\mathbf{d})^\top \mathbf{Q}(\boldsymbol{\alpha}^k + \rho\mathbf{d}) - (\boldsymbol{\alpha}^k + \rho\mathbf{d})^\top \mathbf{1} \\ \text{s.t.} \quad & 0 \leq \alpha_i + y_i\rho \leq C, \\ & 0 \leq \alpha_j - y_j\rho \leq C. \end{aligned}$$

The previous problem is one-dimensional, quadratic, convex and only has two inequality constraints, thanks to the fact that every α_l, $\forall l \neq i, j$ is not updated. Minimizing the previous quadratic function with respect to ρ we get,

$$0 = \frac{1}{2}\mathbf{d}^\top \mathbf{Q}\mathbf{d}\rho + \mathbf{d}^\top \mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{d}^\top \mathbf{1},$$

and the optimum is attained at

$$\rho^* = -\frac{\mathbf{d}^\top \mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{d}^\top \mathbf{1}}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}}. \tag{4.3.14}$$

To compute the previous optimum value we have ignored the inequality constraints so it may need to be modified. However, it turns out that we can simply check if it belongs to the feasible region and clip the stepsize accordingly

$$\hat{\rho}^* = \max\{y_i\alpha_i,\, -y_j\alpha_j,\, \min\{-y_i(C - \alpha_i),\, y_j(C - \alpha_j),\, \rho^*\}\} \tag{4.3.15}$$

The final SMO updates are

$$\begin{aligned} \alpha_i^{k+1} &= \alpha_i^k + y_i\hat{\rho}^*, \\ \alpha_j^{k+1} &= \alpha_j^k - y_j\hat{\rho}^*, \end{aligned}$$

So far we have not said anything about how to select the two coefficients i and j to be updated at each iteration. The original working set selection by Platt (1998) was a quite complicated heuristic. It was improved by Keerthi et al. (2001), where they also introduce a new working set selection based on the concept of "maximal violating pair". The idea behind this selection method is to update the coefficients that violate the most the KKT conditions,

$$i \in \operatorname*{argmax}_{l \in \mathcal{I}_{\text{up}}} \Big\{ -y_l \nabla_l f(\boldsymbol{\alpha}^k) \Big\}, \tag{4.3.16}$$

$$j \in \operatorname*{argmin}_{l \in \mathcal{I}_{\text{low}}} \Big\{ -y_l \nabla_l f(\boldsymbol{\alpha}^k) \Big\}, \tag{4.3.17}$$

where

$$\mathcal{I}_{\text{up}} := \{l \mid \alpha_l < C,\, y_l = 1 \text{ or } \alpha_l > 0,\, y_l = -1\}, \tag{4.3.18}$$

$$\mathcal{I}_{\text{low}} := \{l \mid \alpha_l < C,\, y_l = -1 \text{ or } \alpha_l > 0,\, y_l = 1\}, \tag{4.3.19}$$

and $\nabla f(\boldsymbol{\alpha}) = \mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}$. Fan et al. (2005b) further refined the previous procedure by trying to maximize the decrease in the dual objective function. This is sometimes referred to as second

order working set selection. The new value of the objective function after an update is

$$\begin{aligned} f(\boldsymbol{\alpha}^{k+1}) &= f(\boldsymbol{\alpha}^k + \rho^* \mathbf{d}) \\ &= \frac{1}{2}(\boldsymbol{\alpha}^k + \rho^* \mathbf{d})^\top \mathbf{Q}(\boldsymbol{\alpha}^k + \rho^* \mathbf{d}) - (\boldsymbol{\alpha}^k + \rho \mathbf{d})^\top \mathbf{1} \\ &= \frac{1}{2}(\boldsymbol{\alpha}^k)^\top \mathbf{Q}\boldsymbol{\alpha}^k + \frac{1}{2}(\rho^*)^2 \mathbf{d}^\top \mathbf{Q}\mathbf{d} + \rho^* \mathbf{d}^\top \mathbf{Q}\boldsymbol{\alpha}^k - (\boldsymbol{\alpha}^k)^\top \mathbf{1} - \rho \mathbf{d}^\top \mathbf{1} \\ &= f(\boldsymbol{\alpha}^k) + \frac{1}{2}(\rho^*)^2 \mathbf{d}^\top \mathbf{Q}\mathbf{d} + \rho^* \mathbf{d}^\top (\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{1}) \\ &= f(\boldsymbol{\alpha}^k) + \frac{1}{2}(\rho^*)^2 \mathbf{d}^\top \mathbf{Q}\mathbf{d} + \rho^* \mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k). \end{aligned}$$

Since we already need the gradient of the dual function for finding the working set, it is useful to rewrite Eq. (4.3.14) as

$$\rho^* = -\frac{\mathbf{d}^\top \mathbf{Q}\boldsymbol{\alpha}^k + \mathbf{d}^\top \mathbf{1}}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}} = -\frac{\mathbf{d}^\top (\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{1})}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}} = -\frac{\mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k)}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}}. \tag{4.3.20}$$

Using the value of the dual function at the new coefficients and Eq. (4.3.20), we can compute the unconstrained gain in the dual objective

$$\begin{aligned} f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^{k+1}) &= -\frac{1}{2}(\rho^*)^2 \mathbf{d}^\top \mathbf{Q}\mathbf{d} - \rho^* \mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k) \\ &= -\frac{1}{2}\left(-\frac{\mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k)}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}}\right)^2 \mathbf{d}^\top \mathbf{Q}\mathbf{d} + \frac{\mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k)}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}} \mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k) \\ &= -\frac{1}{2}\frac{(\mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k))^2}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}} + \frac{(\mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k))^2}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}} \\ &= \frac{1}{2}\frac{(\mathbf{d}^\top \nabla f(\boldsymbol{\alpha}^k))^2}{\mathbf{d}^\top \mathbf{Q}\mathbf{d}} \\ &= \frac{1}{2}\frac{(y_i \nabla_i f(\boldsymbol{\alpha}^k) - y_j \nabla_j f(\boldsymbol{\alpha}^k))^2}{K_{ii} + K_{jj} - 2K_{ij}}, \end{aligned}$$

where we abbreviate $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Since it is difficult to maximize with respect to both indices i and j at the same time, Fan et al. (2005b) suggest to do it approximately. First we select i as in Eq. (4.3.16) and then we maximize the previous gain given the selected i,

$$i \in \operatorname*{argmax}_l \Big\{ -y_l \nabla_l f(\boldsymbol{\alpha}^k) \;\Big|\; l \in \mathcal{I}_{\text{up}} \Big\}, \tag{4.3.21}$$

$$j \in \operatorname*{argmin}_l \left\{ -\frac{(y_i \nabla_i f(\boldsymbol{\alpha}^k) - y_l \nabla_l f(\boldsymbol{\alpha}^k))^2}{K_{ii} + K_{ll} - 2K_{il}} \;\middle|\; l \in \mathcal{I}_{\text{low}},\ -y_l \nabla_l f(\boldsymbol{\alpha}^k) < y_i \nabla_i f(\boldsymbol{\alpha}^k) \right\}, \tag{4.3.22}$$

with $\mathcal{I}_{\text{up}}$ and $\mathcal{I}_{\text{low}}$ as in Eqs. (4.3.18) and (4.3.19). The SMO iterates continue until a stopping condition $M(\boldsymbol{\alpha}^k) - m(\boldsymbol{\alpha}^k) < \epsilon_{\text{KKT}}$ is met for a pre-selected KKT tolerance ϵ_{KKT}, where

$$m(\boldsymbol{\alpha}) = \max_{l \in \mathcal{I}_{\text{up}}} -y_l \nabla_l f(\boldsymbol{\alpha}) \quad \text{and} \quad M(\boldsymbol{\alpha}) = \min_{l \in \mathcal{I}_{\text{low}}} -y_l \nabla_l f(\alpha).$$

The floating point cost per iteration of SMO is determined by the choice of j and the computation of the gradient $\nabla f(\boldsymbol{\alpha}^k)$. Selecting j requires $2n$ products and to compute the gradient efficiently just note that

$$\begin{aligned} \nabla f(\boldsymbol{\alpha} + \rho \mathbf{d}) &= \mathbf{Q}(\boldsymbol{\alpha} + \rho \mathbf{d}) - \mathbf{1} \\ &= \nabla f(\boldsymbol{\alpha}) + \rho \mathbf{Q}\mathbf{d} \\ &= \nabla f(\boldsymbol{\alpha}) + \rho(y_i \mathbf{Q}_i - y_j \mathbf{Q}_j), \end{aligned} \tag{4.3.23}$$

Algorithm 13: Sequential Minimal Optimization (SMO)

Input : $\boldsymbol{\alpha} = 0 \in \mathbb{R}^d$ and C
while *stopping condition not met* **do**
  Select working set (i, j)
  Compute unconstrained stepsize ρ as in Eq. (4.3.14)
  Clip the stepsize if necessary as in Eq. (4.3.15)
  $\alpha_i \leftarrow \alpha_i + y_i \hat{\rho}^*$
  $\alpha_j \leftarrow \alpha_j - y_j \hat{\rho}^*$
end

where $\mathbf{Q}_k$ is the kth column of the matrix $\mathbf{Q}$. Thus, a vector with the current gradient is maintained during the optimization and updated with a cost of n products. In total $3n$ floating point operations are needed for each SMO update.

The convergence of SMO has been discussed in Fan et al. (2005b, Sect. 3) and Chen et al. (2006). Given a tolerance ϵ, Theorem 4.1 shows that exists a $\bar{k}$ such that an ϵ-optimal solution $|f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^*)| \leq \epsilon$ is obtained in

$$\bar{k} + \mathcal{O}(\log(1/\epsilon))$$

iterations. That is, SMO converges linearly to an approximate solution. However, this linear convergence only holds after an unspecified initial number of iterations $\bar{k}$. List and Simon (2007) also discuss the convergence of decomposition methods and prove linear convergence but without making the assumptions used by Chen et al. (2006). They also characterize the initial number of iterations $\bar{k}$, which scale quadratically with the number of samples n (List and Simon, 2007, Theorem 4),

$$\mathcal{O}\left(\frac{4n^2 L_{\max} C^2}{\epsilon}\right),$$

where $L_{\max}$ is related to the largest eigenvalue of the matrix $\mathbf{Q}$. The following theorem formally states SMO's linear convergence.

Theorem 4.1 (Theorem 6 in Fan et al., 2005b)**.** *Assume problem* (4.1.23) *satisfies*

1. $\mathbf{Q}$ *is positive definite. Therefore,* (4.1.23) *has a unique solution* $\boldsymbol{\alpha}^*$*.*
2. *The non-degeneracy condition. That is, the optimal solution* $\boldsymbol{\alpha}^*$ *satisfies that*

$$\nabla f(\boldsymbol{\alpha}^*)_i + \bar{b} y_i = 0 \text{ if and only if } 0 < \alpha_i^* < C,$$

where $\bar{b} = m(\boldsymbol{\alpha}^*) = M(\boldsymbol{\alpha}^*)$*.*

For the sequence $\left\{\boldsymbol{\alpha}^k\right\}$ *generated by Algorithm 13, there are* $c < 1$ *and* $\bar{k}$ *such that for all* $k \geq \bar{k}$*,*

$$f(\boldsymbol{\alpha}^{k+1}) - f(\boldsymbol{\alpha}^*) \leq c(f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^*)).$$

We conclude this section by noting another important advantage of SMO compared to the other algorithms considered. SMO is a very flexible framework that can be used to solve not only the standard C-SVC, but also another formulations such as ϵ-SVR, ν-SVC, One-class SVM and so on, with little extra effort.

For the special case of the ν-SVC this is due to the fact that the primal problem has an extra constraint that makes it harder to optimize, while the constraints of the dual problem are simpler. Besides, the dual formulations of both ν-SVC and C-SVC are very similar and thus it is easier to extend a C-SVC dual solver to consider also the ν-SVC.

4.3.4 Shrinking

An optimal solution $\boldsymbol{\alpha}^*$ of the SVM dual problem may contain some bounded elements, i.e., $\alpha_i = 0$ or C. Similarly to the screening technique for the Lasso problems, these elements are bounded in the middle of the optimization procedure and its value will stay the same in the final model. Some recent papers (Wang et al., 2013; Ogawa et al., 2013) also deal with non-support vector screening for SVM classifiers. We will concentrate here on the well-known shrinking technique for SMO, implemented in the LIBSVM software (Chang and Lin, 2011).

SMO shrinking tries to identify and remove bounded dual coefficients, effectively reducing the size of the problem to solve and thus saving training time (Joachims, 1998). It relies on the fact that after some iteration K the non-SVs coefficients do not change from their bound values. This fact is formalized in Theorem 4.2 by Fan et al. (2005b).

Theorem 4.2 (Theorem 5.1 in Chang and Lin, 2011)**.** *Consider problem* (4.1.23) *and assume* $\mathbf{Q}$ *is positive semi-definite.*

1. *The following set is independent of any optimal solution* $\boldsymbol{\alpha}^*$*.*

$$\mathcal{I} := \{i \mid -y_i \nabla_i f(\boldsymbol{\alpha}^*) > M(\boldsymbol{\alpha}^*) \text{ or } -y_i \nabla_i f(\boldsymbol{\alpha}^*) < m(\boldsymbol{\alpha}^*)\}.$$

   Further, for every $i \in \mathcal{I}$*, problem* (4.1.23) *has an unique and bounded optimal solution at* α_i*.*

2. *Assume Algorithm 13 generates an infinite sequence* $\left\{\boldsymbol{\alpha}^k\right\}$*. There exists* K *such that after* $k > K$*, every* α_i^k*,* $i \in \mathcal{I}$ *has reached the unique and bounded optimal solution. That is,* α_i^k *remains the same in all subsequent iterations. In addition,* $\forall k > K$*:*

$$i \notin \left\{l \;\middle|\; M(\boldsymbol{\alpha}^k) \leq -y_l \nabla_l f(\boldsymbol{\alpha}^k) \leq m(\boldsymbol{\alpha}^k)\right\}.$$

Thus after K iterations, provided K is guessed correctly, we can reduce the active set to those $\mathbf{x}_i$ such that their α_i are not bounded. The LIBSVM implementation of shrinking (Chang and Lin, 2011, Sect. 5.1) is as follows:

- Every $\min\{n, 1\,000\}$ iterations it tries to select non-SV candidates $\mathbf{x}_i$ according to whether $\boldsymbol{\alpha}_i^k$ is bounded and the ith gradient component of the SVM dual function at α_i^k goes "against" the bounding box constraint. Details of the exact method can be found in Chang and Lin (2011).

- The previous strategy is sometimes too aggressive. Using Lasso screening terminology, it is not a "safe" procedure and may lead to a wrong solution of the whole problem. Hence, when the algorithm achieves the following condition for the first time

$$m(\boldsymbol{\alpha}^k) \leq M(\boldsymbol{\alpha}^k) + 10\epsilon_{\text{KKT}},$$

  the whole gradient is reconstructed to perform the shrinking procedure on more accurate information.

- Finally, once the stopping condition

$$m(\boldsymbol{\alpha}^k) \leq M(\boldsymbol{\alpha}^k) + \epsilon_{\text{KKT}}$$

  of the smaller problem is satisfied, LIBSVM checks if the stopping condition of the original problem is also satisfied. If not, all coefficients are incorporated into the optimization

problem again and the algorithm is restarted. Note that when solving the smaller problems LIBSVM only maintains the gradient of the coefficients being optimized. Thus, when all variables are reactivated the full gradient has to be reconstructed. More details on the gradient reconstruction can be found in Chang and Lin (2011, Sect. 5.3).

Obviously, over very large samples for which the kernel matrix does not fit into memory, gradient reconstruction is very expensive and SMO training with shrinking may actually be costlier. In LIBSVM (Chang and Lin, 2011, Sect. 5.6) the sizes of the set

$$\mathcal{F} = \{j \mid 0 < \alpha_j < C\}$$

of unbounded coefficients and of the active set $\mathcal{A}$ are compared and a warning is issued if $2|\mathcal{F}| < |\mathcal{A}|$. Shrinking will reduce the iteration cost of SMO, $\mathcal{O}(3n)$, by a fraction $r \simeq m/2d$, with $m = |\mathcal{A}|$, i.e., the number of non-zero features in the final $\boldsymbol{\alpha}^*$ solution.

Chapter 5

Relation between the Lasso and SVMs

In this chapter we will explore the relation between two previously defined problems: the Lasso and the SVM. Both are very popular tools widely used in predictive modeling, each with its own advantages in disadvantages. Although they seem very different in principle, we will show in the following sections how one can construct a SVM instance which is equivalent to the Lasso for a given value of the hyper-parameters, and viceversa.

This reduction was suggested by Jaggi (2014) and is not only interesting from a theoretical point of view, but also, as we will show, from an algorithmic one. With this goal in mind, we will deviate from Jaggi's original reduction in order to arrive to a ν-SVC instance, which can be solved by standard software such as LIBSVM. In fact, using well known equivalences in the literature we could go all they way from the Elastic Net (Section 3.2) to the C-SVC, passing through the Lasso and the ν-SVC.

On a deeper level, we are actually transforming a regression problem into a classification one, which may suggest that they are not two very different types of problems after all. Another example of this regression-classification equivalence was shown in Section 4.2, where we demonstrated how a SVR can be formulated as a SVC problem in an enlarged space with carefully selected artificial labels.

5.1 Previous work

The relationship between the SVC and the Lasso was first investigated by Jaggi (2014). He considers SVM variants whose dual optimization problem is of the form

$$\min_{\mathbf{x}} \left\{ \|\mathbf{A}\mathbf{x}\|_2^2 \right\} \quad \text{s.t.} \quad 0 \leq \mathbf{x} \leq 1, \ \sum x_i = 1, \tag{5.1.1}$$

where $\mathbf{A} \in \mathbb{R}^{d \times n}$ contains all n datapoints as columns and $\mathbf{x} \in \mathbb{R}^n$ is the coefficient vector. Note that $\mathbf{x}$ lies in the unit simplex in $\mathbb{R}^n$, i.e. the set of probability vectors

$$\triangle_n = \left\{ \mathbf{x} \;\middle|\; 0 \leq \mathbf{x} \leq 1, \sum x_i = 1 \right\}. \tag{5.1.2}$$

This formulation includes the soft-margin C-SVC with ℓ_2-loss, both with or without a kernel. However, note that it does not include the bias term nor the more common ℓ_1-loss SVC. To incorporate the bias term we could simply add it as an extra dimension to $\mathbf{x}$, but the optimization problem is still slightly different from the original formulation since we are also regularizing the bias b (see the last paragraph in Section 4.3.1, Item 1).

Jaggi (2014) also considers a slightly different Lasso problem,

$$\min_{\mathbf{x}} \left\{ \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \right\} \quad \text{s.t.} \quad \|\mathbf{x}\|_1 \leq 1, \tag{5.1.3}$$

where $\mathbf{b} \in \mathbb{R}^n$ is the target vector and $\mathbf{A}$, $\mathbf{x}$ are defined as before. This differs from the usual Lasso formulation in two things:

1. The constraint $\|\mathbf{x}\|_1 \leq 1$ is usually written as $\|\mathbf{x}\|_1 \leq \rho$ for some $\rho > 0$. However, without loss of generality we can consider the unit-norm case and simply re-scale the coefficients $\mathbf{x}$ and the target vector $\mathbf{b}$ by a factor $1/\rho$.

2. The interpretation of the problem is different from the usual one, where the rows of $\mathbf{A}$ are samples $(1, \ldots, n)$ and the columns of $\mathbf{A}$ are features or variables $(1, \ldots, d)$. Here the roles of n and d are reversed.

We are going to show now that the two problems (5.1.1) and (5.1.3) are equivalent, in the following sense. For any Lasso instance given by $(\mathbf{A}, \mathbf{b})$ a hard-margin SVM instance can be constructed with the same optimal solution. On the other hand, it can also be shown that given a SVM there always exists an equivalent Lasso instance, although it is more difficult to construct explicitly.

5.1.1 Lasso to SVM

This is the easiest direction of the equivalence, that is, given a Lasso instance constructing the equivalent SVM instance. The main idea behind this reduction is to parametrize the ℓ_1-ball,

$$\blacklozenge_n = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\|_1 \leq 1\}, \tag{5.1.4}$$

by the simplex $\triangle_{2n}$, using two non-negative variables to represent each real variable. Formally, any vector $\mathbf{x} \in \blacklozenge_n$ can be written as (Jaggi, 2014)

$$\mathbf{x} = (\mathbf{I}_n \,|\, -\mathbf{I}_n)\, \widetilde{\mathbf{x}},$$

where $\widetilde{\mathbf{x}} \in \triangle_{2n}$ and $(\mathbf{A} \,|\, \mathbf{B})$ denotes the horizontal concatenation of two matrices $\mathbf{A}$ and $\mathbf{B}$. With this representation, given a Lasso instance of the form (5.1.3) we can write the equivalent non-negative Lasso problem

$$\min_{\widetilde{\mathbf{x}}} \left\{ \|(\mathbf{A} \,|\, -\mathbf{A})\, \widetilde{\mathbf{x}} - \mathbf{b}\|_2^2 \right\} \quad \text{s.t.} \quad \widetilde{\mathbf{x}} \in \triangle_{2n} \,. \tag{5.1.5}$$

Finally, we can translate each column of the new data matrix by the vector $-\mathbf{b}$, that is, $\widetilde{\mathbf{A}} = (\mathbf{A} \,|\, -\mathbf{A}) - \mathbf{b}\mathbf{1}_{2n}^\top$. The term $\mathbf{b}\mathbf{1}_{2n}^\top$ just means that the vector $\mathbf{b}$ is subtracted from every column of $\mathbf{A}$ and $-\mathbf{A}$. Doing so we obtain the equivalent SVM instance,

$$\min_{\widetilde{\mathbf{x}}} \left\{ \left\| \widetilde{\mathbf{A}} \widetilde{\mathbf{x}} \right\|_2^2 \right\} \quad \text{s.t.} \quad \widetilde{\mathbf{x}} \in \triangle_{2n} \,. \tag{5.1.6}$$

Here we have crucially used the simplex domain so that $\mathbf{b}\mathbf{1}_{2n}^\top \widetilde{\mathbf{x}} = \mathbf{b}$. As Jaggi (2014) points out, this equivalence not only means that the optimal solutions coincide, but it gives us a correspondence of all the feasible points, preserving the objective value: for any solution $\mathbf{x} \in \mathbb{R}^n$ to the Lasso, we have a feasible SVM solution $\widetilde{\mathbf{x}} \in \mathbb{R}^{2n}$ of the same objective value and vice versa.

5.1.2 SVM to Lasso

Although a lot harder to accomplish, this reduction can also go in the other direction, that is, given a SVM instance we can construct an equivalent Lasso instance. We will need the following definition, that measures the quality of an approximate solution $\mathbf{x}$ of the SVM problem as the attained margin:

Definition 5.1 (Definition 1 in Jaggi, 2014)**.** A vector $\mathbf{w} \in \mathbb{R}^d$ is called σ-weakly-separating for the SVM instance (5.1.1) for a parameter $\sigma > 0$, if it holds that

$$\mathbf{A}_i^\top \frac{\mathbf{w}}{\|\mathbf{w}\|_2} \geq \sigma \quad \forall i,$$

meaning that $\mathbf{w}$ attains a margin of separation of σ.

The soft-margin SVC with ℓ_2-loss, without bias term, is given by the primal optimization problem (Jaggi, 2014)

$$\begin{aligned} \min_{\mathbf{w},\rho,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 - \rho + \frac{C}{2}\sum_i {\xi_i}^2 \\ \text{s.t.} \quad & y_i \cdot \mathbf{w}^\top \mathbf{X}_i \geq \rho - \xi_i,\ i = 1, \ldots, n, \end{aligned} \tag{5.1.7}$$

where $C > 0$ is the regularization parameter and $\rho/\|\mathbf{w}\|_2$ is the final margin of separation. Note that in the classical SVC (Section 4.1) ρ is usually fixed to one, while it is explicitly used in the equivalent ν-SVC formulation (Section 4.1.4). The equivalence of the soft-margin SVC dual problem (5.1.1) to the previous optimization problem is stated in the following Lemma:

Lemma 5.1 (Lemma 2 in Jaggi, 2014)**.** *The dual of the soft-margin SVC* (5.1.7) *is a instance of* (5.1.1)*, that is*

$$\min_{\mathbf{x}} \left\{ \|\mathbf{A}\mathbf{x}\|_2^2 \right\} \quad \textit{s.t.} \quad \mathbf{x} \in \triangle_n,$$

with

$$\mathbf{A} := \begin{pmatrix} \mathbf{Z} \\ \frac{1}{\sqrt{C}}\mathbf{I}_n \end{pmatrix} \in \mathbb{R}^{(d+n)\times n}$$

where the data matrix $\mathbf{Z} \in \mathbb{R}^{d\times n}$ *consists of the* n *columns* $\mathbf{Z}_i := y_i \mathbf{X}_i$*.*

We are ready now to start with the reduction. Given an instance of the ℓ_2-loss dual SVC (5.1.1), let us show first how to obtain a (possibly non-optimal) σ-weakly-separating vector $\mathbf{w} \in \mathbb{R}^d$ for some small $\sigma > 0$. Let

$$\mathbf{w} := \begin{pmatrix} \mathbf{0}_d \\ \frac{1}{\sqrt{n}}\mathbf{1}_n \end{pmatrix} \in \mathbb{R}^{d+n},$$

then by Lemma 5.1 this direction $\mathbf{w}$ obtains a separation margin of

$$\mathbf{A}_i \frac{\mathbf{w}}{\|\mathbf{w}\|_2} = \begin{pmatrix} y_i \mathbf{X}_i \\ \frac{1}{\sqrt{C}}\mathbf{e}_i \end{pmatrix}^\top \begin{pmatrix} \mathbf{0}_d \\ \frac{1}{\sqrt{n}}\mathbf{1}_n \end{pmatrix} = \frac{1}{\sqrt{nC}} > 0$$

for all points in Definition 5.1 (Jaggi, 2014). The equivalent Lasso instance $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{b}})$, can be defined as the translated SVM datapoints

$$\widetilde{\mathbf{A}} := \left\{ \mathbf{A}_i + \widetilde{\mathbf{b}} \;\middle|\; i = 1, \ldots, n \right\}$$

where

$$\widetilde{\mathbf{b}} := -\frac{\mathbf{w}}{\|\mathbf{w}\|_2} \cdot \frac{D^2}{\sigma}.$$

Here $D > 0$ is a strict upper bound on the length of the original SVM datapoints, that is, $\|\mathbf{A}_i\|_2 < D$ for all i. By the definition of $\widetilde{\mathbf{A}}$, the Lasso objective function coincides with the original SVM objective (5.1.1) for any $\mathbf{x} \in \triangle_n$,

$$\left\|\widetilde{\mathbf{A}}\mathbf{x} - \widetilde{\mathbf{b}}\right\|_2^2 = \left\|(\mathbf{A} + \widetilde{\mathbf{b}}\mathbf{1}^\top)\mathbf{x} - \widetilde{\mathbf{b}}\right\|_2^2 = \left\|\mathbf{A}\mathbf{x} + (\mathbf{1}^\top\mathbf{x} - 1)\widetilde{\mathbf{b}}\right\|_2^2 = \|\mathbf{A}\mathbf{x}\|_2^2,$$

since $\mathbf{1}^\top\mathbf{x} = 1$. However, as Jaggi (2014) points out, this does not necessarily hold for the larger part of the Lasso domain when $\mathbf{x} \in \blacklozenge \setminus \triangle$. It turns out that for this constructed Lasso instance all the feasible points are contained in the simplex, and thus all the candidates $\mathbf{x} \in \blacklozenge \setminus \triangle$ can be discarded from the problem, since they do not contribute to any optimal solution. This is shown in Theorem 3 from Jaggi (2014), which we reproduce below.

Theorem 5.2 (Theorem 3 in Jaggi, 2014)**.** *For any candidate solution $\widetilde{\mathbf{x}} \in \blacklozenge$ to the Lasso problem* (5.1.3) *defined by $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{b}})$, there is a feasible vector $\mathbf{x} \in \triangle$ in the simplex, of the same or better Lasso objective value γ. Furthermore, this $\mathbf{x} \in \triangle$ attains the same objective value γ in the original SVM problem* (5.1.1)*.*

5.1.3 Elastic Net to SVM

After the observation in Jaggi (2014), a new relation between the Elastic Net and the SVM was suggested by Zhou et al. (2014). In particular, a Constrained Elastic Net instance is reduced to a ℓ_2-loss SVC by using a similar approach. Starting from the Constrained Elastic Net formulation, the first step is to re-scale the weights by $1/\rho$,

$$\min_{\mathbf{w}} \left\{ \frac{1}{2}\left\|\mathbf{X}\mathbf{w} - \frac{1}{\rho}\mathbf{y}\right\|_2^2 + \lambda_2\|\mathbf{w}\|_2^2 \right\} \quad \text{s.t.} \quad \|\mathbf{w}\|_1 \leq 1. \tag{5.1.8}$$

As before, we can use two non-negative variables to represent each real variable and split the coefficient vector $\mathbf{w}$ into the positive components, $\mathbf{w}^+ \geq 0$, an the negative components, $\mathbf{w}^- \geq 0$. Let $\widetilde{\mathbf{w}} \in \mathbb{R}^{2d}$ be the new weight vector, resulting from the concatenation of $\mathbf{w}^+$ and $\mathbf{w}^-$. Then, the previous problem can be written as,

$$\min_{\widetilde{\mathbf{w}} \geq 0} \left\{ \frac{1}{2}\left\|(\mathbf{X} \,|\, -\mathbf{X})\,\widetilde{\mathbf{w}} - \frac{1}{\rho}\mathbf{y}\right\|_2^2 + \lambda_2 \sum_{i=1}^{2d} \widetilde{w}_i^2 \right\} \quad \text{s.t.} \quad \sum_{i=1}^{2d} \widetilde{w}_i \leq 1. \tag{5.1.9}$$

Note that, as long as $\lambda_2 \neq 0$ the solution to the previous problem is unique and satisfies that either $w_i^+ = 0$ or $w_i^- = 0$ for all i (Zhou et al., 2014). Now Zhou et al. (2014) define the following: $\mathbf{Z} = \left(\widetilde{\mathbf{X}}_1 \,|\, -\widetilde{\mathbf{X}}_2\right)$, where $\widetilde{\mathbf{X}}_1 = \mathbf{X} - \frac{1}{\rho}\mathbf{y}\mathbf{1}_d^\top$ and $\widetilde{\mathbf{X}}_2 = \mathbf{X} + \frac{1}{\rho}\mathbf{y}\mathbf{1}_d^\top$. Substituting $\mathbf{Z}$ into (5.1.9) it becomes

$$\min_{\widetilde{\mathbf{w}} \geq 0} \left\{ \frac{1}{2}\|\mathbf{Z}\widetilde{\mathbf{w}}\|_2^2 + \lambda_2 \sum_{i=1}^{2d} \widetilde{w}_i^2 \right\} \quad \text{s.t.} \quad \sum_{i=1}^{2d} \widetilde{w}_i = 1, \tag{5.1.10}$$

since $\mathbf{1}_{2d}^\top\widetilde{\mathbf{w}} = 1$. Also note that the ℓ_1-norm constraint in (5.1.9) will always be tight, excluding the uninteresting case where ρ is extremely large (Zhou et al., 2014).

Finally Zhou et al. (2014) show that an optimal solution $\widetilde{\mathbf{w}}^*$ for the problem (5.1.10) can be obtained as $\mathbf{w}^* = \boldsymbol{\alpha}^*/\|\boldsymbol{\alpha}^*\|_1$, where $\boldsymbol{\alpha}^*$ is the solution of the ℓ_2-loss SVC dual problem with no

bias term for a carefully constructed binary classification dataset with $2d$ samples and n features. More specifically, the data matrix is constructed as the horizontal concatenation of $\widetilde{\mathbf{X}}_1$ and $\widetilde{\mathbf{X}}_2$,

$$\widetilde{\mathbf{X}} = \left(\widetilde{\mathbf{X}}_1 \,|\, \widetilde{\mathbf{X}}_2\right) \in \mathbb{R}^{2d\times n},$$

and the labels are $\widetilde{\mathbf{y}} \in \mathbb{R}^{2d}$, where $\widetilde{y}_i = +1$ for $i = 1, \ldots, d$ and $\widetilde{y}_i = -1$ for $i = d+1, \ldots, 2d$.

To solve the problem they use their own implementation based the approach in Chapelle (2007) to solve ℓ_2-loss SVMs, Suport Vector Elastic Net (SVEN), which is able to perform the training in parallel using multiple cores or a GPU. The results in Zhou et al. (2014) show that the GPU implementation is faster than the multicore one and that both are usually faster than GLMNet. SVEN also outperforms other Elastic Net algorithms, namely, LARS (Efron et al., 2004), L1_LS (Kim et al., 2007), SLEP (Liu et al., 2009), Shotgun (Bradley et al., 2011) and Accelerated Prox-SDCA (Shalev-Shwartz and Zhang, 2014).

5.2 Constrained and unconstrained Lasso

Recall that the original Lasso formulation solves the following constrained optimization problem

$$\min_{\mathbf{w}} \left\{\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\right\} \quad \text{s.t.} \quad \|\mathbf{w}\|_1 \leq \rho. \tag{5.2.1}$$

We will call this the Constrained Lasso or C-Lasso for short. However the previous problem is usually written in its unconstrained version,

$$\min_{\mathbf{w}} \left\{\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1\right\}, \tag{5.2.2}$$

which we will denote by Unconstrained Lasso or U-Lasso. As we shown in Chapter 3, most of the solvers work with the unconstrained formulation due to the absence of constraints. However the Lasso-SVM reduction by Jaggi (2014) starts with the constrained formulation. In this section we are going to show that both formulations are actually equivalent. This is something that was mentioned as early as the original Lasso paper (Tibshirani, 1994), so it its often assumed but rarely explicitly proved.

We must point out that, as we will see, they are equivalent in the sense that for a given ρ we can always find a value λ such as both problems share the same solution, and vice versa. However, there is not a one to one correspondence and, for example, there could be many values for ρ whose equivalent λ is the same, since the "useful" regions of these hyper-parameters are quite different. By "useful" region we mean values of the parameters for which the constraint is active (the solution is not OLS) and the solution is not the trivial one, i.e, $\mathbf{w} = \mathbf{0}$.

In problem (5.2.2) it is obvious that we can disable the constraint simply by taking $\lambda = 0$, obtaining the original Ordinary Least Squares problem. The following proposition (Tibshirani, 1994) characterizes the ρ values for which the constraint in the C-Lasso problem is not active.

Proposition 5.1 (Inactive constraints)**.** *Let $\mathbf{w}_0$ be the full OLS solution and $\rho_0 = \|\mathbf{w}_0\|_1$. Then, for every $\rho \geq \rho_0$, the C-Lasso solution is also $\mathbf{w}_0$. We can also get the same solution $\mathbf{w}_0$ in the U-Lasso problem simply by taking $\lambda = 0$.*

On the other hand, in problem (5.2.1) it is clear that $\rho = 0$ will yield the zero solution. Proposition 5.2 characterizes the values for λ where the same happens in the U-Lasso problem. This result is also given in Osborne et al. (2000, Remark 2) and Friedman et al. (2010). Note that in Friedman et al. (2010) they are minimizing the average squared error and thus in that case $\lambda_{\max}$ can be also scaled by $1/n$.

Proposition 5.2 (Trivial solution)**.** *Let $\lambda_{max} = \|\mathbf{X}^\top \mathbf{y}\|_\infty$. Then, for every $\lambda \geq \lambda_{max}$ $\mathbf{w}_\lambda = \mathbf{0}$. We can also get the trivial solution in the C-Lasso problem simply by taking $\rho = 0$.*

Proof. Let $F(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$ be the objective function of the U-Lasso problem. Taking the subgradient with respect to $\mathbf{w}$,

$$\partial_{\mathbf{w}} F(\mathbf{w}) = \mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{s},$$

where $\mathbf{s} \in \partial\|\mathbf{w}\|_1$, we get the optimality condition

$$\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{s} = 0.$$

For the solution $\mathbf{w}^*$ to be zero we need

$$\mathbf{X}^\top\mathbf{y} + \lambda\mathbf{s} = 0$$

or equivalently

$$\mathbf{s} = \frac{\mathbf{X}^\top\mathbf{y}}{\lambda}.$$

But $\mathbf{s}$ must also be a subgradient of the ℓ_1-norm and thus $\|\mathbf{s}\|_\infty \leq 1$. Putting everything together,

$$1 \geq \|\mathbf{s}\|_\infty = \left\|\frac{\mathbf{X}^\top\mathbf{y}}{\lambda}\right\|_\infty,$$

which implies $\lambda \geq \|\mathbf{X}^\top\mathbf{y}\|_\infty$ for $\mathbf{w}^* = \mathbf{0}$ to be a solution of U-Lasso. □

We begin by showing first the easiest direction of the equivalence between the two Lasso formulations, that is, given a U-Lasso instance we construct a C-Lasso one with the same solution.

Lemma 5.3 (U-Lasso to C-Lasso)**.** *Given $0 < \lambda < \lambda_{max}$ let*

$$\mathbf{w}_\lambda = \operatorname*{argmin}_{\mathbf{w}} \left\{\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1\right\},$$

be the solution of the U-Lasso problem (5.2.2)*. Then, if we let $\rho = \|\mathbf{w}_\lambda\|_1$ the solution of the C-Lasso problem* (5.2.1) *is also $\mathbf{w}_\lambda$.*

Proof. Suppose that there exits a better minimizer $\mathbf{w}^*$ of (5.2.1) such that $\|\mathbf{w}^*\|_1 \leq \|\mathbf{w}_\lambda\|_1$ and

$$\frac{1}{2}\|\mathbf{X}\mathbf{w}^* - \mathbf{y}\|_2^2 < \frac{1}{2}\|\mathbf{X}\mathbf{w}_\lambda - \mathbf{y}\|_2^2.$$

In that case, it is clear that $\mathbf{w}^*$ would be also a better solution of problem (5.2.2), contradicting the assumption that $\mathbf{w}_\lambda$ was the minimum. □

Next lets prove the second direction of the equivalence, i.e., starting with a given instance of the C-Lasso problem and obtaining the equivalent U-Lasso one.

Lemma 5.4 (C-Lasso to U-Lasso)**.** *Given ρ_0 as in Proposition 5.1 and $0 < \rho < \rho_0$ let*

$$\mathbf{w}_\rho = \operatorname*{argmin}_{\mathbf{w}} \left\{\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\right\} \quad \textit{s.t.} \quad \|\mathbf{w}\|_1 \leq \rho,$$

be a solution of the C-Lasso problem (5.2.1)*. Then, if we let*

$$\lambda = \frac{-\mathbf{w}_\rho^* \cdot \mathbf{X}^\top\Big(\mathbf{X}\mathbf{w}_\rho^* - \mathbf{y}\Big)}{\rho}. \tag{5.2.3}$$

the solution of the U-Lasso problem (5.2.2) *is also $\mathbf{w}_\rho$.*

Proof. Let $f(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$, $g(\mathbf{w}) = \|\mathbf{w}\|_1 - \rho$ and define

$$h(\mathbf{w}) = \max\{f(\mathbf{w}) - f(\mathbf{w}_\rho),\ g(\mathbf{w})\}. \tag{5.2.4}$$

The function $h(\mathbf{w})$ is convex because it is the maximum of two convex functions. In addition we also have $h(\mathbf{w}) \geq 0$, since when $g(\mathbf{w}) < 0$ we necessarily have

$$f(\mathbf{w}) \geq f(\mathbf{w}_\rho) \quad \Rightarrow \quad f(\mathbf{w}) - f(\mathbf{w}_\rho) \geq 0,$$

otherwise $\mathbf{w}_\rho$ would not be a solution of problem (5.2.1). On the other hand, if $g(\mathbf{w}) > 0$, $h(\mathbf{w}) \geq g(\mathbf{w}) > 0$. Let us compute now the value of the function h at $\mathbf{w}_\rho$,

$$\begin{aligned} h(\mathbf{w}_\rho) &= \max\{f(\mathbf{w}_\rho) - f(\mathbf{w}_\rho),\ \|\mathbf{w}_\rho\|_1 - \rho\} \\ &= \max\{0,\ \|\mathbf{w}_\rho\|_1 - \rho\} = 0 \end{aligned}$$

since

$$\|\mathbf{w}_\rho\|_1 \leq \rho \quad \Rightarrow \quad \|\mathbf{w}_\rho\|_1 - \rho \leq 0.$$

Putting everything together we get that the minimum of the function h is also attained at $\mathbf{w}_\rho$, since $h(\mathbf{w}_\rho) = 0$ and it is always positive. This implies that $0 \in \partial h(\mathbf{w}_\rho)$. To construct the subdifferential note that h is the maximum of two convex functions and thus by Proposition 2.1 its subdifferential at $\mathbf{w}$ can be constructed as the convex hull generated by

$$\partial(f(\cdot) - f(\mathbf{w}_\rho))(\mathbf{w}_\rho) = \partial f(\mathbf{w}_\rho) = \{\nabla f(\mathbf{w}_\rho)\}$$

and Eq. (2.2.7)

$$\partial g(\mathbf{w}_\rho) = \left\{\mathbf{s} \;\middle|\; \|\mathbf{s}\|_\infty < 1,\ \mathbf{s}^\top \mathbf{w}_\rho = \|\mathbf{w}_\rho\|_1\right\};$$

that is,

$$\partial h(\mathbf{w}_\rho) = \left\{\gamma \nabla f(\mathbf{w}_\rho) + (1-\gamma)s \;\middle|\; 0 \leq \gamma \leq 1,\ \|s\|_\infty < 1,\ s^\top \mathbf{w}_\rho = \|\mathbf{w}_\rho\|_1\right\}. \tag{5.2.5}$$

Now let $\lambda = (1-\gamma)/\gamma$ with $\gamma > 0$, then Eq. (5.2.5) can be rewritten into

$$\partial h(\mathbf{w}_\rho) = \left\{\nabla f(\mathbf{w}_\rho) + \lambda s \;\middle|\; \lambda \geq 0,\ \|s\|_\infty < 1,\ s^\top \mathbf{w}_\rho = \|\mathbf{w}_\rho\|_1\right\},$$

which is exactly the subdifferential of problem (5.2.2). Note that $\gamma > 0$ since otherwise the only possible solution would be $\mathbf{w}_\rho = 0$, contradicting the fact $\|\mathbf{w}_\rho\|_1 = \rho$. The conclusion is that $\mathbf{w}_\rho$ is also a solution for the U-Lasso with a specific value for λ that we derive next. Let $0 < \gamma \leq 1$ and $s \in \partial g(\mathbf{w}_\rho)$; then we have

$$\mathbf{s}^\top \mathbf{w}_\rho = \|\mathbf{w}_\rho\|_1 = \rho, \tag{5.2.6}$$

and

$$0 = \nabla f(\mathbf{w}_\rho) + \lambda \mathbf{s} = \mathbf{X}^\top(\mathbf{X}\mathbf{w}_\rho - \mathbf{y}) + \lambda \mathbf{s}.$$

Multiplying both sides of the previous equation by $\mathbf{w}_\rho$ and substituting Eq. (5.2.6) yields

$$-\mathbf{X}^\top(\mathbf{X}\mathbf{w}_\rho - \mathbf{y}) \cdot \mathbf{w}_\rho = \lambda \mathbf{s}^\top \mathbf{w}_\rho = \lambda \rho,$$

and we can solve for λ to get the final value

$$\lambda = \frac{-\mathbf{X}^\top(\mathbf{X}\mathbf{w}_\rho - \mathbf{y}) \cdot \mathbf{w}_\rho}{\rho}.$$

□

We now summarize the equivalence between the U-Lasso and the C-Lasso in the following Theorem.

Theorem 5.5 (Equivalence between C-Lasso and U-Lasso)**.** *Let* $\mathbf{w}_0$ *be the Ordinary Least Squares solution,*

$$\mathbf{w}_0 = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \right\},$$

let $\mathbf{w}_\lambda$ *be the solution of the U-Lasso problem,*

$$\mathbf{w}_\lambda = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1 \right\},$$

and let $\mathbf{w}_\rho$ *be the solution of the C-Lasso problem,*

$$\mathbf{w}_\rho = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \right\} \quad s.t. \quad \|\mathbf{w}\|_1 \leq \rho.$$

Then, for the U-Lasso:

$$\mathbf{w}_\lambda = \begin{cases} \mathbf{0}, & \text{if } \lambda \geq \lambda_{max}, \\ \mathbf{w}_0, & \text{if } \lambda = 0, \\ \mathbf{w}_{\rho^*}, & \text{if } 0 < \lambda < \lambda_{max}, \end{cases}$$

where $\rho^* = \|\mathbf{w}_\lambda\|_1$ *and* $\lambda_{max} = \left\|\mathbf{X}^\top \mathbf{y}\right\|_\infty$*, and for the C-Lasso:*

$$\mathbf{w}_\rho = \begin{cases} \mathbf{0}, & \text{if } \rho = 0, \\ \mathbf{w}_0, & \text{if } \rho \geq \rho_0, \\ \mathbf{w}_{\lambda^*}, & \text{if } 0 < \rho < \rho_0, \end{cases}$$

where $\rho_0 = \|\mathbf{w}_0\|_1$ *and*

$$\lambda^* = \frac{-\mathbf{X}^\top (\mathbf{X}\mathbf{w}_\rho - \mathbf{y}) \cdot \mathbf{w}_\rho}{\rho}.$$

Proof. Combine the results in Propositions 5.1 and 5.2 and Lemmas 5.3 and 5.4. □

The conclusion of Theorem 5.5 is that both problems are equivalent in the sense that, given ρ and a solution $\mathbf{w}_\rho$ of problem C-Lasso, then the problem U-Lasso shares the same solution if we take the regularization parameter λ as in Eq. (5.2.3). Likewise, given λ and a solution $\mathbf{w}_\lambda$ of problem U-Lasso, an instance of C-Lasso with $\rho = \|\mathbf{w}_\lambda\|_1$ also has solution $\mathbf{w}_\lambda$.

Finally it is important to note that we can only go from one problem to the other after it has been solved. Therefore given, for example, a ρ we can not choose to solve the equivalent U-Lasso instance before solving C-Lasso for that ρ and the other way around. However, in practice the value for the hyper-parameters is selected using procedures such as cross-validation. In that case we can indeed choose to solve one or the other since the cross-validation is finding the "best" value for either ρ or λ for that specific problem.

5.3 Constrained Lasso to Nearest Point Problem

We are also going to follow the approach by Jaggi (2014) but with some slight variations in order to arrive at an instance of the ν-SVC problem, which is directly solvable by well-established software such as LIBSVM (Chang and Lin, 2011). Since we have already shown that the C-Lasso problem is equivalent to the U-Lasso, which is the formulation usually chosen by Lasso solvers,

this will allow us to first, solve an instance of the U-Lasso for a given λ, compute the ρ for which both problems are equivalent (Lemma 5.3), transform that C-Lasso instance into a ν-SVC, solve it using LIBSVM and finally check which one was faster from an empirical point of view. In other words, this equivalence could potentially provide a faster way to solve the C-Lasso problem which, in turn, could also mean a faster way to solve the general Lasso problem.

Starting with problem (5.2.1), the first step is to re-scale $\mathbf{w}$ and $\mathbf{y}$ by $1/\rho$, that is,

$$\min_{\mathbf{w}} \left\{ \|\mathbf{X}\widehat{\mathbf{w}} - \widehat{\mathbf{y}}\|_2^2 \right\} \quad \text{s.t.} \quad \|\widehat{\mathbf{w}}\|_1 \leq 1, \tag{5.3.1}$$

where $\widehat{\mathbf{w}} = \mathbf{w}/\rho$ and $\widehat{\mathbf{y}} = \mathbf{y}/\rho$. For simplicity we will revert again to the notation $\mathbf{w}$, $\mathbf{y}$ in the rest of the derivation and assume that the problem is scaled by $1/\rho$. As before, the main part of the equivalence is to represent the ℓ_1-ball as the $2d$-dimensional simplex of probability vectors

$$\triangle_{2d} = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^{2d} \;\middle|\; 0 \leq \boldsymbol{\alpha} \leq 1, \sum \alpha_i = 1 \right\}.$$

The trick here is to use two non-negative artificial features to represent each coefficient w. Let us denote the horizontal concatenation of matrices $\mathbf{A}$ and $\mathbf{B}$ as $(\mathbf{A} \,|\, \mathbf{B})$, then the new coefficients are $\widetilde{\mathbf{w}} = (\mathbf{I}_d \,|\, -\mathbf{I}_d)^\top \mathbf{w}$. Now we can rewrite problem (5.3.1) as

$$\min_{\widetilde{\mathbf{w}}} \left\{ \|(\mathbf{X} \,|\, -\mathbf{X})\, \widetilde{\mathbf{w}} - \mathbf{y}\|_2^2 \right\} \quad \text{s.t.} \quad 0 \leq \widetilde{\mathbf{w}} \leq 1, \; \sum_{i=1}^{2d} \widetilde{w}_i = 1. \tag{5.3.2}$$

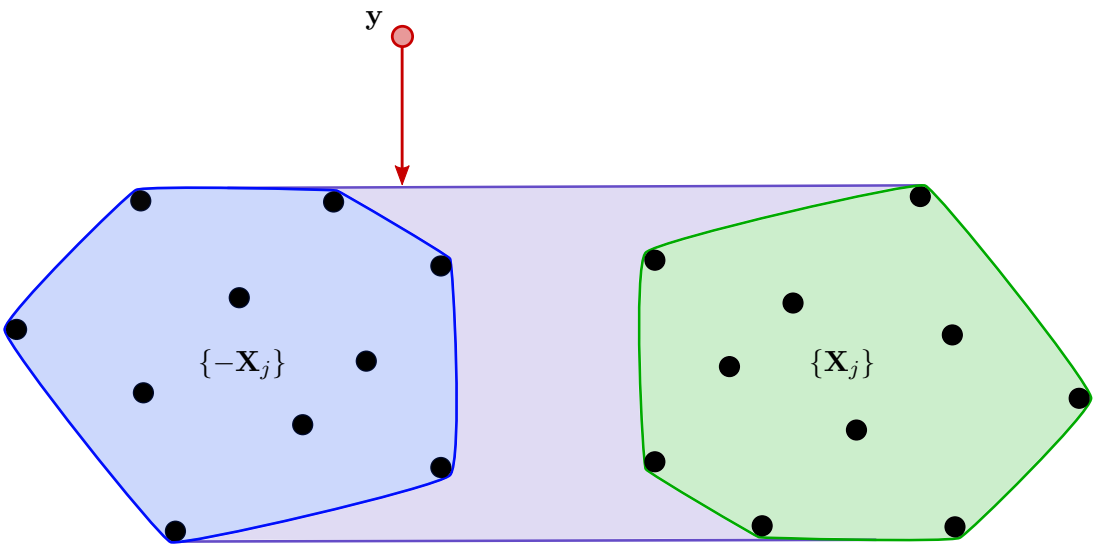

Figure 5.1: Geometrical interpretation of problem (5.3.2)

Now let $\widetilde{\mathbf{X}} = (\mathbf{X} \,|\, -\mathbf{X})$, i.e the new artificial data matrix, constructed by horizontally concatenating $\mathbf{X}$ and $-\mathbf{X}$. Then $\widetilde{\mathbf{X}}\widetilde{\mathbf{w}}$ lies in the convex hull spanned by the columns of $\widetilde{\mathbf{X}}$,

$$\mathcal{S} = \left\{ \sum_{j=1}^{2d} \alpha_j \widetilde{\mathbf{X}}_j \;\middle|\; 0 \leq \boldsymbol{\alpha} \leq 1, \sum_{j=1}^{2d} \alpha_j = 1 \right\},$$

where $\widetilde{\mathbf{X}}_k$ denotes the kth column of the matrix $\widetilde{\mathbf{X}}$. Therefore problem (5.3.2) could also be interpreted as finding the closest point between the convex hull $\mathcal{S}$ and the singleton $\{\mathbf{y}\}$, which is a convex hull containing only one point. This is shown in Fig. 5.1. A more general version of this geometrical problem, the Nearest Point Problem, is given in Definition 5.2.

Definition 5.2 (μ-Nearest Point Problem)**.** Given a set of d-dimensional points $\{\mathbf{x}_i\}_1^n$ belonging to either a positive or a negative (reduced) convex hull, according to some label $y_i = \pm 1$, the problem of finding the closest point between the hulls can be stated as the following minimization,

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \left\| \sum_{y_i=1} \alpha_i \mathbf{x}_i - \sum_{y_j=-1} \alpha_j \mathbf{x}_j \right\|^2 \right\} \quad \text{s.t.} \quad 0 \leq \boldsymbol{\alpha} \leq \mu.$$

or, equivalently,

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \right\} \quad \text{s.t.} \quad 0 \leq \boldsymbol{\alpha} \leq \mu, \ \sum_{y_i=1} \alpha_i = \sum_{y_i=-1} \alpha_i = 1, \ \forall i.$$

As mentioned above, one of the hulls only contains only one point an thus we carefully construct the following artificial binary classification dataset: data matrix $\mathbf{A} = \left(\widetilde{\mathbf{X}} \,|\, \mathbf{y}\right) \in \mathbb{R}^{n \times (2d+1)}$ and labels $\mathbf{b} = (\mathbf{1}_{2d}^\top, -1)^\top \in \mathbb{R}^{2d+1}$, i.e we assign positive labels to every column of $\widetilde{\mathbf{X}}$ and a negative label to $\mathbf{y}$. Then, (5.3.2) can be written as the following geometrical problem,

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j b_i b_j \mathbf{A}_i^\top \mathbf{A}_j \right\} \quad \text{s.t.} \quad 0 \leq \boldsymbol{\alpha} \leq 1, \ \sum_{i=1}^{2d} \alpha_i = \alpha_{2d+1} = 1, \tag{5.3.3}$$

where $\mathbf{A}_k$ represents each one of the $2d+1$ columns of the matrix $\mathbf{A}$. Note that we have added a spurious coefficient α_{2d+1} associated with the point $\mathbf{y}$ that, according to the constraint has to be 1 when the optimization finishes. Therefore is not going to be used in order to recover the original Lasso weights.

One option for trying to solve problem (5.3.3) is to do it directly using a geometrical based algorithm. In particular, Zhou et al. (2015) suggest a Wolfe-type algorithm that directly solves the following problem,

$$\min_{\boldsymbol{\alpha}} \left\{ \left\| \widetilde{\mathbf{A}} \boldsymbol{\alpha} \right\|^2 \right\} \quad \text{s.t.} \quad 0 \leq \boldsymbol{\alpha} \leq 1, \ \sum \alpha_i = 1, \tag{5.3.4}$$

where $\widetilde{\mathbf{A}} := \widetilde{\mathbf{X}} - \mathbf{y}\mathbf{1}_{2d}^\top$, i.e, they translate the matrix $\widetilde{\mathbf{X}} = (\mathbf{X} \,|\, -\mathbf{X})$ by substracting $\mathbf{y}$ from each one of its columns. This is known as the Minimum Norm Problem (MNP) and it can be interpreted as trying to find the closest point from the convex hull to the origin. In general, Wolfe's method performance for the MNP problem is sublinear and can be improved by other variants, such as the MDM algorithm (Torres-Barrán and Dorronsoro, 2015; López and Dorronsoro, 2015). MDM is closely related to SMO and, in fact, the MNP problem can also be solved by LIBSVM: simply note that the MNP problem is a particular case of the Nearest Point Problem taking the vector $\{\mathbf{0}\}$ as the second hull. Alternatively, it is also easy to see that (5.3.4) is the dual of the following one-class problem (Section 4.1.5)

$$\min_{\mathbf{w},\rho,\boldsymbol{\xi}} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 - \rho + \sum_i \xi_i \right\} \quad \text{s.t.} \quad \mathbf{w}^\top \widetilde{\mathbf{X}}_i - \rho + \xi_i \geq 0, \ \xi_i \geq 0, \ i = 1, \ldots, 2d, \tag{5.3.5}$$

which can also be solved by LIBSVM.

We deviate from the previous approach and use the well-known equivalence between the Nearest Point Problem and ν-SVC to solve problem (5.3.3). To show that equivalence, it is useful to define the following primal optimization problem:

Definition 5.3 (μ-Margin Nearest Point Problem)**.** Given a set of d-dimensional points $\{\mathbf{x}_i\}_1^n$ and $\mu \geq 0$ we define the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi},\gamma,\eta} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 - \gamma + \eta + \mu \sum_i \xi_i \\ \text{s.t.} \quad & \mathbf{w}^\top \mathbf{x}_i \geq \gamma - \xi_i, \, \forall i : y_i = 1 \\ & \mathbf{w}^\top \mathbf{x}_i \leq \eta + \xi_i, \, \forall i : y_i = -1 \\ & \xi_i \geq 0, \, \forall i. \end{aligned}$$

It turns out that problem (5.2) is the dual of problem (5.3), which is equivalent to the ν-SVC formulation (Section 4.1.4) given the following transformations

$$\nu = \frac{2}{\mu n}, \; \rho^* = \frac{\gamma^* - \eta^*}{2}, \; b^* = -\frac{\gamma^* + \eta^*}{2},$$

and scaling the coefficients by $1/(\mu n)$ (López and Dorronsoro, 2015). Thus, in our case, we can obtain a solution $\boldsymbol{\alpha}$ of the NPP problem by solving a ν-SVM instance, that is,

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi},b,\rho} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 - \nu\rho + \frac{1}{n}\sum_i \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq \rho - \xi_i, \\ & \xi_i \geq 0, \; i = 1, \ldots, n. \end{aligned} \tag{5.3.6}$$

with $\nu = 2/n$ and scaling the resulting coefficients by $1/n$. It is important to note that here the n represents the number of points in the convex hulls or, in other words, the number of columns of matrix $\mathbf{A}$ which is $2d + 1$. Thus, the original number of samples n and the dimensionality d switch places, since now we have $(2d+1)$ n-dimensional points. Note also that we have effectively transformed a regression problem into a classification problem, which are sometimes regarded as two very different settings in supervised learning.

5.4 Numerical experiments

The goal of this section is to empirically test the performance of the SMO algorithm (Section 4.3.3), as implemented by LIBSVM (Chang and Lin, 2011), for solving the Lasso problem via its equivalent ν-SVC instance. This is one of the many implications of the reduction from the previous section, that is, any algorithmic advances in one problem can be directly translated to the other one. Other implications will be discussed further in Section 5.5.

For comparison purposes, we have selected the Cyclic Coordinate Descent algorithm as implemented by the `Lasso` class in *scikit-learn*. This implementation is based on the well known GLMNet R package (Friedman et al., 2010), but it does not contain all of its features. In particular, it is lacking in 1) a screening procedure and 2) covariance updates (Section 3.3.3). Therefore in theory it should be less efficient than the R implementation. However we still choose here to use the *scikit-learn* version since the main Coordinate Descent algorithm is implemented in a few lines of Cython, and thus it is easy to modify it to measure, say, the runtime as a function of the distance to the optimum. On the other hand GLMNet's core is implemented in Fortran and we were not able to modify it to produce plots as the ones in Section 5.4.3. As an example, it is coded in a very low level programming style, using branch statements instead of loops.

Although SMO is quite an old algorithm, it is still probably one of the best available options to solve the non-linear SVC problem. In the issue at hand, the equivalent ν-SVC instance it is actually the linear version, so one may ask if there is not a better alternative. However, we are not aware of any other software that is able to solve the linear ν-SVC, whether it be the primal or dual formulation. For example neither Pegasos nor LIBLINEAR, which are two of the most popular solvers besides LIBSVM, consider the ν-SVC problem. One possible reason may be because the ν-SVC is actually harder to optimize due to having an extra equality constraint, and since it is equivalent to the C-SVC this drawback outweighs the possible benefits.

Finally, it is also worth mentioning the tight relationship between Cyclic Coordinate Descent and SMO. Recall that SMO was just an specific instance of the more general Greedy Block Coordinate Descent family of algorithms, where two coefficients are selected using the rules explained in Section 4.3.3 and optimized at the same time. On the other hand, Cyclic Coordinate Descent optimizes one coefficient at a time, selected in cyclical order. However this selection could also be random (Stochastic Coordinate Descent) or using some greedy rules, such as the Gauss-Southwell rule. In general either the Cyclic or Stochastic updates are considered to be more efficient (Nesterov, 2012) but some recent works like Nutini et al. (2015) show that Greedy Coordinate Descent actually performs better empirically, contradicting the theoretical convergence rates. In this fashion we consider worth exploring whether SMO, another variant of Greedy Coordinate Descent is actually faster than Cyclic Coordinate Descent for solving the Lasso problem in an empirical setting.

5.4.1 Implementation details

First we summarize the whole procedure. Given a Lasso instance with n samples and d variables, represented by the data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, target vector $\mathbf{y} \in \mathbb{R}^n$ and hyper-parameter ρ, we perform the following steps:

1. Create an artificial dataset $(\mathbf{A}, \mathbf{b})$, with $\mathbf{A} = (\mathbf{X} \,|\, -\mathbf{X} \,|\, \widetilde{\mathbf{y}})^\top \in \mathbb{R}^{(2d+1) \times n}$ and labels $\mathbf{b} = (\mathbf{1}_{2d}^\top, -1)^\top \in \mathbb{R}^{2d+1}$, where $\widetilde{\mathbf{y}} = \mathbf{y}/\rho$.

2. Solve the associated ν-SVC problem with data matrix $\mathbf{A}$, target vector $\mathbf{b}$ and $\nu = \frac{2}{2d+1}$ via the dual formulation, obtaining dual coefficients $\boldsymbol{\alpha}^* \in \mathbb{R}^{2d+1}$.

3. Discard the last coefficient α_{2d+1} and recover original Lasso weights as

$$\mathbf{w}^* = \rho \, (\mathbf{I}_d \,|\, -\mathbf{I}_d) \, \boldsymbol{\alpha}^*.$$

Note that, in practice, when we are solving the Lasso-NPP problem (5.3.3) through its equivalent ν-SVM instance (5.3.6), we are not assuming any kind of structure in the kernel matrix. However, due to the artificial nature of the matrix $\mathbf{A}$ it contains many redundancies. Indeed if we take into account how the kernel matrix $\mathbf{K}$ of the dual problem is computed we can see that it has the following structure:

$$\mathbf{K} = \mathbf{A}\mathbf{A}^\top = \underbrace{\begin{pmatrix} \mathbf{X}^\top\mathbf{X} & -\mathbf{X}^\top\mathbf{X} & \mathbf{X}^\top\widetilde{\mathbf{y}} \\ -\mathbf{X}^\top\mathbf{X} & \mathbf{X}^\top\mathbf{X} & -\mathbf{X}^\top\widetilde{\mathbf{y}} \\ \widetilde{\mathbf{y}}^\top\mathbf{X} & -\widetilde{\mathbf{y}}^\top\mathbf{X} & \widetilde{\mathbf{y}}^\top\widetilde{\mathbf{y}} \end{pmatrix}}_{2d+1} \Bigg\} \, 2d+1$$

As a result, we only need to compute $\mathbf{X}^\top\mathbf{X}$, $\tilde{\mathbf{y}}^\top\tilde{\mathbf{y}}$ and $\mathbf{X}^\top\tilde{\mathbf{y}}$ and fill the rest of the kernel matrix accordingly. In addition, the matrix $\mathbf{X}^\top\mathbf{X}$ is also symmetric and only half of the dot products are actually needed.

Thus, we have implemented a modified kernel in LIBSVM, namely K-SVM, where we have exploited the specific structure of the kernel matrix for this problem. It is important to note that the kernel matrix is not precomputed before the algorithm starts, since we have already seen in Section 4.3.3 that it is counterproductive. Instead, in the same manner as the original SMO, the rows of the kernel matrix (or columns, since it is symmetric) are computed on-the-fly, at most two per iteration, taking into account the structure of the kernel to avoid computing the same dot product twice.

Newly computed rows are then stored in a simple First-In-First-Out (FIFO) cache so they can be reused if SMO selects again the same pair of coefficients at a later iteration. When looking for a previously computed row in the cache we also take into account the structure of the kernel matrix. If we need an index $j \in d, \dots, 2d$ we actually look for the index $i = j - d$ and negate the result, since we now that the resulting rows are going to be identical. This way we avoid filling up the cache with redundant information, reducing the effective size of the kernel matrix to $(d+1) \times (d+1)$. This is specially important for problems where $d \leq n$.

It is also worth mentioning that the changes to LIBSVM are minor and only involve the way SMO gets the kernel row associated to each multiplier from the cache; no other change is made to the LIBSVM code. Theoretically, these changes could perform up to 8 times less dot products compared to the naive LIBSVM implementation for this specific problem, although in practice we observe gains of about a factor of 4.

Solving the ν-SVC instance associated to the Lasso should be specially beneficial in problems where $d < n$, since SVC are known to scale poorly with the number of samples (d in our case), whether in runtime or memory storage ($\mathcal{O}(d^2)$). On the other hand, if $d \gg n$ it may be better to either solve directly the Lasso or the primal formulation of the ν-SVC, although we are not aware of any popular implementation for the latter.

5.4.2 Datasets and methodology

We have selected eleven datasets, trying to cover a broad amount of different problems. Of these, eight correspond to regression problems: `prostate`, used by Tibshirani (1994) in the original Lasso paper; `housing`, `year`, `ctscan` and `cpusmall` from the UCI repository; `trajectory`, from the Machine Learning Dataset Repository; `ree` and `mnist_reg`.

The `ree` datasets is built with 2 years of Numerical Weather Forecasts (NWP) for eight meteorological variables, namely, temperature, pressure, wind speed at surface level (u-component, v-component and modulus) and wind speed at 100m (u-component, v-component and modulus). They are arranged in a rectangular grid of 35 latitudes and 57 longitudes with an spatial resolution of 0.25°, for a total of 1995 points covering the whole peninsular Spain. Thus, the total dimension is $1995 \times 8 = 15960$. The temporal resolution of the forecasts is every three hours, so the number of samples is approximately $2 \times 365 \times 24/3 = 5840$. The actual number of samples is actually less due to some missing data. The targets are the total wind energy productions every three hours as a percentage of the total installed power in Spain.

The `mnist_reg` is a regression problem built using the MNIST dataset. We proceed as Xiang et al. (2017), randomly selecting 500 feature images from each digit as well as a random digit target image. Then we build $28 \times 28 = 784$ samples with feature dimension 5 000 by pairing the i, j pixels of the 5 000 feature images with the i, j pixel of the target image. The idea is that, when trying to predict the value of a pixel of the target image, only the corresponding pixels

Table 5.1: Optimal λ values, $\lambda/\lambda_{\max}$ ratio, sample sizes and input dimensions of the datasets considered.

Dataset	Optimal λ^*	$\lambda^*/\lambda_{\max}$	Size	Dim.
`prostate`	2.035×10^{-3}	0.003	67	8
`cpusmall`	1.631×10^{-2}	0.048	6 143	12
`housing`	8.186×10^{-3}	0.011	378	13
`year`	4.534×10^{-4}	0.002	46 215	90
`trajectory`	1.021×10^{-2}	0.129	20 000	298
`ctscan`	3.748×10^{-3}	0.007	53 500	385
`mnist_reg`	7.760×10^{-3}	0.036	784	5 000
`ree`	1.290×10^{-1}	0.164	5 698	15 960
`colon_cancer`	6.191×10^{-3}	0.071	62	2 000
`leukemia`	5.294×10^{-3}	0.497	72	7 129
`breast_cancer`	3.537×10^{-2}	0.347	44	7 129

in the images containing the same digit should be relevant features. Therefore this is a way to construct a regression dataset where 1) only about 10% of the features should be relevant and 2) we could have a dimension up to 60000, which is the total number of digits in the MNIST dataset, assuming all the digits are represented equally. Note that we have arbitrarily selected here only 500 images per digit for a total dimension of 5000.

In addition to the previous regression datasets, we also consider three classification datasets for biomedical problems: `leukemia`, `colon_cancer` and `breast_cancer`, from the LIBSVM dataset repository. These two-class datasets are transformed to a regression problem by simply predicting the value of the -1/1 class label. The reason for this addition is to further explore the case where the number of features is much greater than the number of patterns, which is easier to find in classification problems from the medical domain.

Table 5.1 summarizes in columns 4 and 5 the sample sizes and input dimensions of all the previous datasets. It is worth mentioning that all of them correspond to real-world problems with a wide variety of sizes. Some of them are also quite big for a regression setting, either in number of patterns (`ctscan`, `year`) or input dimension (`mnist_reg`, `ree`, `leukemia`, `breast_cancer`).

It is well known that training complexity greatly depends on λ. We will consider in the experiments three possible λ values for U-Lasso: an optimal λ^* obtained as the one with the lowest cross-validation error in the regularization path (Section 3.3.3), a smaller $\lambda^*/2$ value which should result in longer training and possibly less sparsity, and a stricter penalty $2\lambda^*$ value with the opposite effect. Another possibility would be to fix a desired level of sparsity as measured by the ratio $\lambda/\lambda_{\max}$ (for instance 10^{-3}) and then compute the corresponding λ for each specific problem, possibly with different multiples as well.

As discussed by Xiang et al. (2017), the ratio $\lambda/\lambda_{\max}$ is invariant to scaling and is thus a better measure of the regularization strength being applied in the Lasso. However we choose the λ^* with the lowest cross-validation error because is the procedure applied in practice if one wants to train the best performing models. In any case, we give the optimal λ^* and their $\lambda_{\max}$ scalings for the problems considered in the second and third columns of Table 5.1. Most problems have small scaled λ values; they are higher and close to 0.5 in `leukemia` and `breast_cancer`.

Since GLMNet solves the U-Lasso (5.2.2) and we are solving the C-Lasso formulation (5.2.1), we proceed as follows. First we run GLMNet for each λ, obtaining an ϵ-optimal solution $\mathbf{w}^*_\lambda$

with $\epsilon = 10^{-6}$. Then we compute $\rho = \|\mathbf{w}_\lambda^*\|_1$ and solve the C-Lasso problem via the associated ν-SVM instance, as described in the previous section. Finally we compare for each algorithm the evolution of $f(\mathbf{w}^k) - f(\mathbf{w}^*)$, with $\mathbf{w}^k$ the coefficients at iteration k.

As mentioned, we will use the *scikit-learn* implementations of GLMNet and of ν-SVM, which is actually a wrapper over LIBSVM (Chang and Lin, 2011), to which we add the modified LIBSVM code described also in the previous section to reduce the number of kernel operations and cache misses. We denote these algorithms as GLMNet, SVM and K-SVM respectively. Note that all methods have a C core, so we may expect time comparisons to be broadly homogeneous. As we discussed before we could have also compared to other algorithms since many of them have reported to be better than GLMNet. However, very recent works on Lasso algorithms such as Frandi et al. (2016) still compare to GLMNet, which suggest that it is performing very good empirically and it should be the first one to beat. Other classical algorithms such as FISTA (Section 3.3.2) are at a clear disadvantage with respect to SMO and GLMNet, since they have to compute the full gradient at every iteration. Furthermore Alaíz et al. (2015) have observed that FISTA needs more iterations than GLMNet or SMO to achieve the same precision on the objective function so we believe that it is not competitive with the others.

5.4.3 Results

All the experiments were run in an Intel(R) Xeon(R) server with 16 E5-2680 2.70GHz CPUs and 128 Gb of RAM. Table 5.2 shows in columns 3 to 5 the number of iterations that GLMNet, SVM and K-SVM respectively require to compute a ϵ-optimal solution with $\epsilon = 10^{-6}$. Columns 6 to 8 show the runtimes needed to achieve the same precision.

We consider GLMNet and SVM iterations to be roughly equivalent even though GLMNet only changes one coefficient per iteration and SVM two. Table 5.2 shows that K-SVM beats GLMNet for all datasets and λ values except for the `trajectory` problem. Besides, for the regression datasets, K-SVM improves SVM running times by a factor that approximately lies between 2 and 4, while both perform the same number of iterations in all problems, as it should be. However, for the classification datasets, SVM is faster than K-SVM. We believe this is due to these datasets having a very small number of patterns and dimensions between 40 and 100 times larger; therefore the dot products computed are very cheap while the kernel matrix is rather large. This, combined with the fact that the optimization finishes in very few iterations (note that at most two dot products are computed in each iteration), makes the overhead of taking into account the kernel matrix structure to be worse than just computing the full dot products. As a conclusion, SVM may be better suited for problems where the ratio d/n is very big. However, for the three classification datasets both SVM and K-SVM are still faster than GLMNet in our experiments.

Problems with a large dimensionality, such as `mnist_reg`, `ree` and the classification datasets, do need very few SVM iterations. This is related to the sparsity of the final solutions, given in the second column of Table 5.2, in the form s/d, with s the number of non-zero coefficients and d the problem dimension. Note that this sparsity is given in terms of the original Lasso weights $\mathbf{w}$, and not ν-SVC dual weights $\boldsymbol{\alpha}$. However, as pointed out by Jaggi (2014), the number of non-zero coefficients of the Lasso instance is related to the number of $\boldsymbol{\alpha}$ elements different from zero (i.e. support vectors). Thus this is another implication of the reduction, and we could try to gain understanding about the final number of support vectors by looking at known results about Lasso sparsity. This assumes that those results are applicable for the type of translation $\mathbf{b}$ that Jaggi (2014) uses to construct the equivalent the Lasso instance and it is not clear how to apply this in practice.

Table 5.2: Sparsity of final solutions, number of iterations and running times.

		Iterations			Time (ms)		
Dataset	Spars.	GLMNet	SVM	K-SVM	GLMNet	SVM	K-SVM
prostate	8/8	102	40	40	0.051	0.030	0.023
	8/8	103	35	35	0.052	0.026	0.020
	7/8	93	28	28	0.048	0.023	0.017
housing	12/13	607	71	71	0.567	0.160	0.073
	12/13	594	53	53	0.555	0.152	0.064
	11/13	216	33	33	0.204	0.143	0.053
year	90/90	5 153	693	693	456.753	757.949	182.591
	89/90	5 148	559	559	451.329	731.710	176.815
	85/90	4 998	445	445	425.374	718.278	175.916
ctscan	273/385	101 130	944	944	8 371.474	11 334.768	3 707.740
	226/385	78 710	629	629	5 949.542	9 316.175	3 354.670
	165/385	53 630	387	387	3 488.810	6 755.625	2 702.005
cpusmall	9/12	135	21	21	1.126	1.363	0.366
	8/12	144	13	13	1.150	1.201	0.338
	6/12	45	11	11	0.337	0.896	0.308
mnist_reg	40/5 000	1 040 395	87	87	1 451.346	371.682	217.461
	22/5 000	635 452	42	42	940.499	185.491	106.909
	13/5 000	550 356	24	24	694.931	109.587	63.693
trajectory	45/297	8 555	164	164	228.461	583.402	264.339
	17/297	2 588	32	32	61.546	253.426	108.617
	6/297	1 313	9	9	28.673	84.417	36.852
ree	73/15 960	1 784 764	185	185	9 949.901	17 033.588	9 386.198
	43/15 960	1 385 271	108	108	7 058.281	8 691.214	5 078.536
	23/15 960	1 114 658	52	52	5 541.830	5 097.402	2 789.337
leukemia	28/7 129	415 227	71	71	197.762	49.557	87.672
	20/7 129	225 846	28	28	58.096	26.253	50.891
	8/7 129	115 898	9	9	60.959	10.155	22.217
colon_cancer	47/2 000	184 353	469	469	88.677	31.467	63.017
	33/2 000	101 231	227	227	45.856	19.183	38.180
	25/2 000	62 377	54	54	29.894	7.411	14.476
breast_cancer	29/7 129	219 503	100	100	97.303	50.856	92.861
	16/7 129	107 397	27	27	47.075	23.024	40.219
	7/7 129	59 597	8	8	26.265	5.919	16.873

In the other direction, which is the one of interest here, there are also some results about the asymptotic number of support vectors as the number of samples grow, for example Steinwart (2003). Note that in theory the Lasso and SVM solutions could potentially have different numbers of non-zero coefficients: a fully sparse $\mathbf{w} = 0$ Lasso solution can be derived from a fully non-sparse $\boldsymbol{\alpha}$, where $\alpha_j = 1/2d$ for $j = 1, \ldots, d$ and $\alpha_j = -1/2d$ for $j = d+1, \ldots, 2d$. However we have not observed that to happen in practice and both the sparsity in $\boldsymbol{\alpha}$ and the sparsity in $\mathbf{w}$ coincide in our experiments. Thus we report sparsity values in just a single column. As the table shows, the sparsities in `mnist_reg`, 22 non-zero coefficients out of 5 000 for λ^*, or `ree`, 43 out of 15 960, are very large. In the `mnist_reg` case this is to be expected since, as we mentioned before, many features correspond to zero pixels and also only 1/10 of the features correspond to images from the digit that we are trying to predict. In `ree` it is possibly due to the large correlation between weather variables at nearby grid points. The classification datasets have also a very big sparsity factor, probably because the ratio d/n is huge and therefore there are many irrelevant variables, as it is usually the case in medical data.

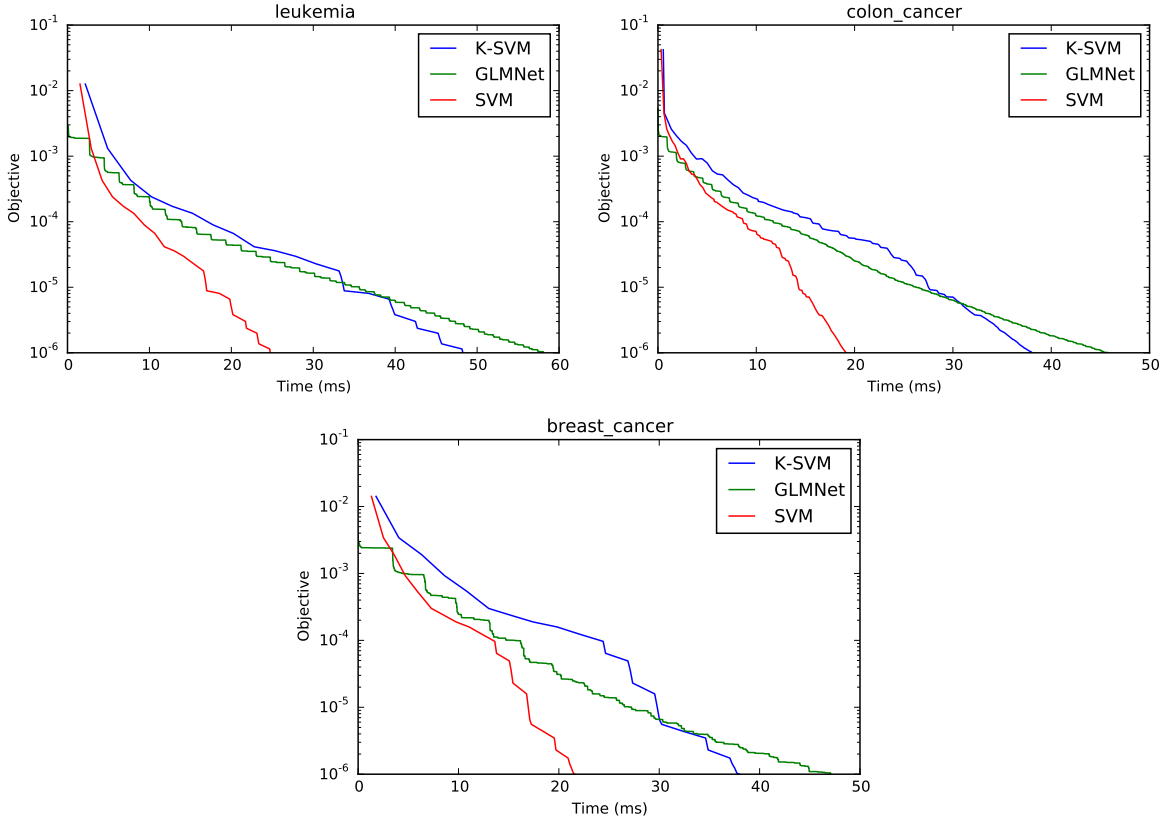

Figure 5.2: Time evolution of the objective function for the three classification datasets with λ^* as the penalty factor

Figure 5.2 shows the evolution of the objective function with respect to time for the three classification problems. As mentioned, in this setting where $d \gg n$ there are very few kernel operations (because of the small number of iterations until convergence) and they are also very cheap (because of a very small n). Thus, for these datasets reducing the number the cache misses it is not critical, since the cache is probably not filling up in the first place, and the SVM without any modifications performs very well. Even though they are very sparse problems, the inner loop of GLMNet still has to cycle through all coefficients and thus the number of iterations for this

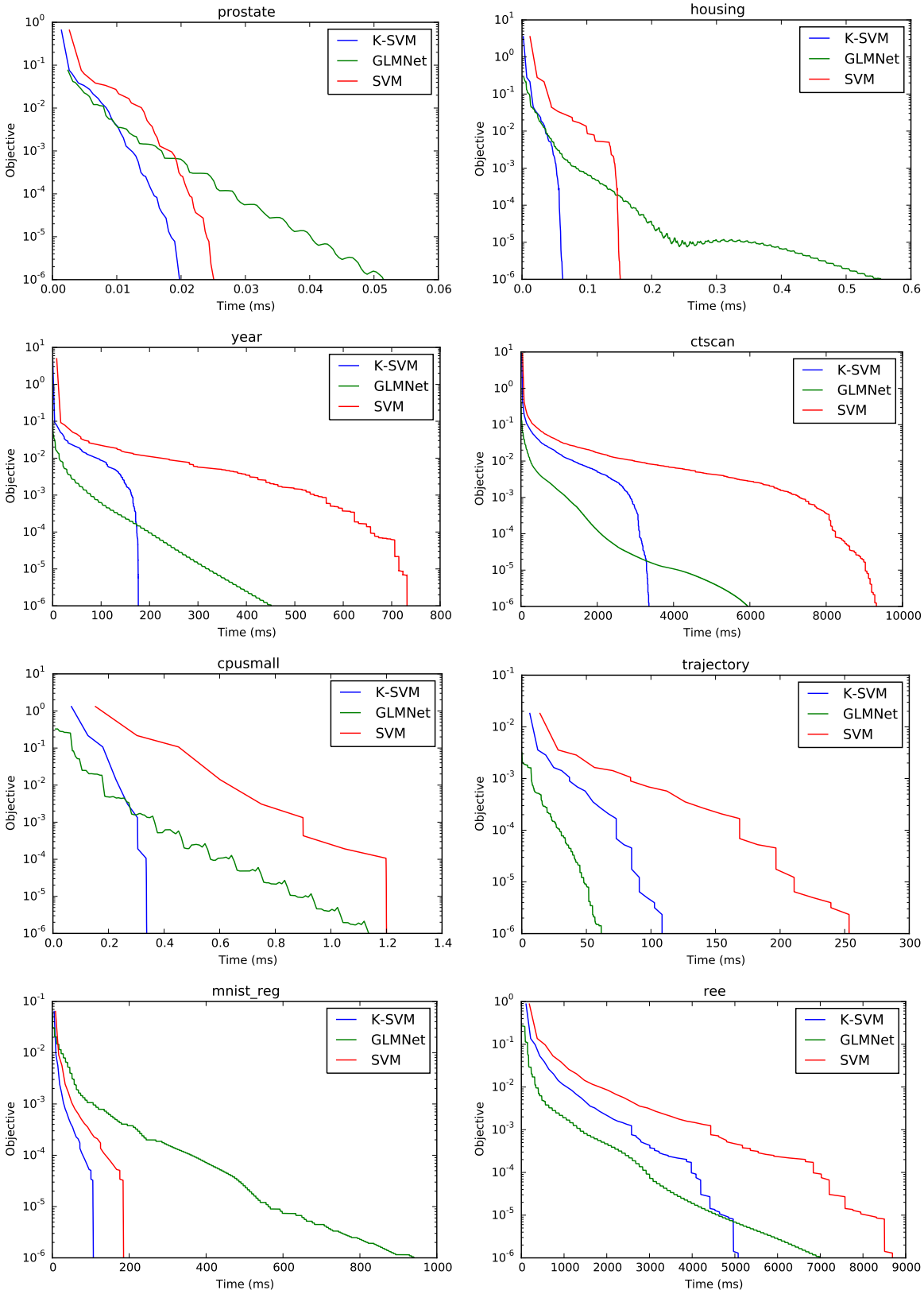

Figure 5.3: Time evolution of the objective function for the eight regression datasets with λ^* as the penalty factor

algorithm is still very high. These problems could probably benefit from some kind of feature screening which is not yet included in the CCD algorithm implemented by *scikit-learn*. However, as discussed in Section 3.4, in theory the strong rules (Tibshirani et al., 2012) only produce significant gain when $\lambda/\lambda_{\max} > 0.5$ which implies no gain in the case of the classification datasets considered here. These are the rules implemented in GLMNet R package, but not in the current *scikit-learn* version.

This convergence behaviour is further illustrated in Fig. 5.3 that depicts for the λ^* penalty the evolution of running times until the ϵ threshold is reached. In the small sample regression problems of `housing`, `prostate` and `mnist_reg`, where kernel operations are less costly, the running time of SVM is smaller than that of GLMNet even for rather modest values of ϵ (about 10^{-3}), but consistently larger in all the others. Futhermore, K-SVM obtains an ϵ-optimal solution faster than GLMNet for $\epsilon \geq 10^{-3}$ in `prostate`, `housing`, `mnist_reg` and `cpusmall`, for $\epsilon \geq 10^{-4}$ in `year`, and for $\epsilon \geq 10^{-5}$ in `ctscan` and `ree`. On the other hand, GLMNet is the clear winner in `trajectory` no matter the ϵ value. The conclusion is that the speed of the algorithm is always relative to the precision up to which the solution is computed. Note that this precision is arbitrary although it is very rare to see a value for $\epsilon < 1 \times 10^{-2}$ in optimization software, and one could argue that $\epsilon > 1 \times 10^{-6}$ is unnecessarily accurate for most applications. As an example, Chang and Lin, 2011 lets $\epsilon = 10^{-3}$ by default.

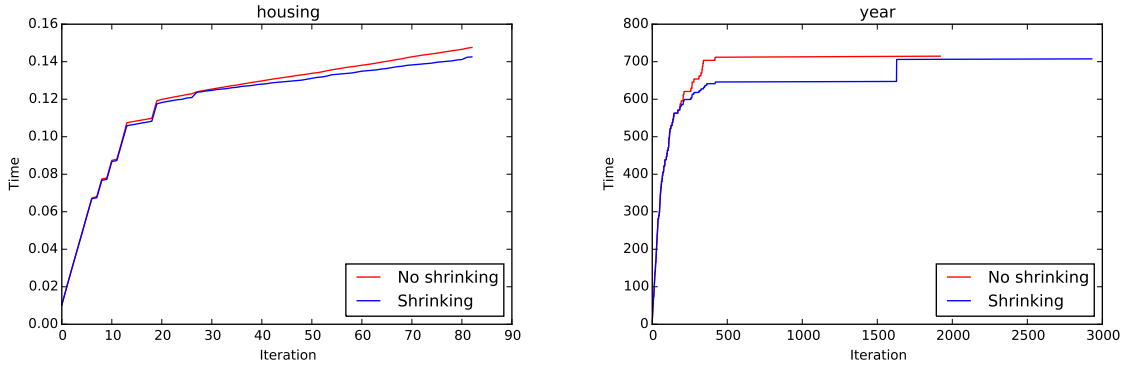

Figure 5.4: Time versus iterations for the `housing` and `year` datasets with penalty $2\lambda^*$ with and without shrinking

Besides, it is important to emphasize that **the shrinking feature of LIBSVM was also disabled** for all the regression and classification experiments, so we believe the comparisons reported to be fair. Although it was disabled, recall from Section 4.3.3 that LIBSVM only applies shrinking after the first $\min\{2d + 1, 1\,000\}$ iterations. Comparing the dimensions in Table 5.1 and the number of iterations in Table 5.2, this implies that it will have no effect on the large dimensional `ctscan`, `trajectory`, `mnist_reg`, `ree` and classification datasets. In the remaining datasets, we believe shrinking would have little to no effect in `prostate` and `cpusmall`, most likely because the gradient conditions for shrinking are not met, and a very small one in `housing` and `year`. The evolution of training times versus iterations in these problems with and without shrinking is given in Fig. 5.4 for the stronger sparsity inducing $2\lambda^*$ case. Note that we have used here the standard version of LIBSVM. As it can be seen, the effect is very modest, something to be partially expected given the small sparsity of the optimal solutions.

5.5 Discussion and further work

Following Jaggi (2014), in this chapter we have proposed a reduction from the Lasso problem to the Nearest Point Problem which, in turn, can be further transformed into a ν-SVC instance. In particular, the reduction starts from the Constrained formulation of the problem, although we have shown it is equivalent to the more usual Unconstrained one. Besides, the ν-SVC is also equivalent to the standard C-SVC (Section 4.1), although similarly to the Constrained and Unconstrained Lasso one can only go from one to the other once we get an optimal solution. Jaggi (2014) discusses many implications of this equivalence:

1. Algorithmic advances on one problem can be translated to the other For instance solvers for the SVM can now compete with well-established Lasso solvers, such as, GLMNet.

2. As we discussed in Chapter 4, SVMs can be extended to the non-linear setting through kernels. Thus, this equivalence could also provide a new kernelized Lasso formulation.

3. Theoretical results, such as theorems characterizing the sparsity of the solutions of one of the problems can also be useful to the other one.

4. Homotopy methods, which exploit the regularization path to compute faster solutions, are very well studied for the Lasso (Mairal and Yu, 2012; Giesen et al., 2012; Gärtner et al., 2012). In this sense maybe some new homotopy methods can de derived for the SVM using this equivalence.

We have already mentioned several works that exploit Item 1. The approach in Zhou et al. (2014) has the considerable advantage of working on GPU architectures. In addition, it can solve either a primal or a dual SVM problem, while LIBSVM only solves the dual one. On the other hand, they consider the Constrained Elastic Net, which is then transformed into a ℓ_2-regularized, ℓ_2-loss SVM, which is equivalent to Constrained Elastic Net. Although it is trivial to obtain the Lasso problem just by setting $\lambda_2 = 0$, they do not consider it in the numerical experiments. Also note that using the transformation from Section 3.2 any Elastic Net instance can be transformed into a Lasso problem using an extended dataset. The Lasso is addressed directly in Zhou et al. (2015) but we believe that the Wolfe solver proposed there should be less efficient than SMO, and no GLMNet comparisons are reported.

Besides making the reduction more precise, we have also demonstrated empirically that the straight use of the SMO algorithm, as implemented by LIBSVM, is competitive with GLMNet in six out of eleven problems. In four out of the five remaining ones, its running times remain close. We have also suggested how to exploit in practice the particular structure of the kernel matrix of the equivalent ν-SVC problem to achieve further time gains. Following this path we have experimentally shown that this small modification of LIBSVM results in a procedure that is faster than GLMNet in all of the problems considered except `trajectory`.

Recently several screening procedures have been proposed to speed up solving Lasso by reducing the size of the active feature set. While very elegant and powerful on concrete setups, screening procedures face two drawbacks. First, one usually needs to know the solution of Lasso $\mathbf{w}(\lambda_0)$ on a previous λ_0 penalty when applying screening over a new $\lambda < \lambda_0$; this can be done advantageously on a sequential, regularization path setting while looking for an optimal λ but less so when that λ is already fixed and Lasso solutions have to be repeatedly computed for that λ. The second drawback is that screening will reduce running times more effectively the sparser a problem is. In fact, for the datasets considered the screening procedure using the strong rules would only reduce training times for penalty ratios $\lambda/\lambda_{\max}$ larger than 0.5. Other

sequential methods in Xiang et al. (2017) produce, in some problems, runtime reductions for ratios as small as 0.05. In all our experiments this ratio (computed for a λ^* optimal obtained by cross-validation) was below 0.165, in six of them below 0.05 and in four below 0.02. Besides, high sparsity usually corresponds to large λ penalties but, on the other hand, large penalties may result in poorer models; again, in our problems the optimal λ values derived using the regularization path procedure of Friedman et al. (2010) yielded quite small penalty ratios.

Although different in many aspects, shrinking is a natural counterpart to screening when training SVMs. We have not used it in our comparisons with GLMNet but have independently observed that it would have had only slight effects on just two datasets. This may be partially due to the conservative way shrinking is applied in LIBSVM. A conservative approach is certain sensible when SVMs are to be trained using non-linear kernels such as Gaussian ones, since shrinking is not a safe procedure and it may initially lead to wrong solutions and a very costly reconstruction of the entire SVM gradient afterwards. On the other hand, a more aggressive shrinking could be safer in the linear SVM problems that arise from the Lasso reduction. In fact, another way to speed up solving the related ν-SVC instance is to consider specialized solvers, either for the primal or dual formulation, that only work on the linear problem. However, as of now, ν-SVC is not included in the most popular linear SVM solver, LIBLINEAR (Fan et al., 2008).

As we discussed earlier, although we only consider here Cyclic Coordinate Descent, we also point out that there has been lately a large research effort dealing with algorithms to solve the Lasso problem. We have already discussed some of them in Section 3.3. Among the many relevant papers, (see Yuan et al., 2012, for a review up to 2012) we can mention Tomioka et al. (2011) as an example of the broad range of techniques applied and that we have not covered.

One specific example that stands out is Stochastic Coordinate Descent, where the next coefficient to update is selected uniformly at random instead of cyclically. This version enjoys better theoretical convergence rates (Richtárik and Takáč, 2014), although empirically it was found in Shalev-Shwartz and Tewari (2011) to perform similarly to the cyclic one. Among recent algorithms we highlight Shalev-Shwartz and Zhang (2014), which is closely related to SMO. Another trend in coordinate descent algorithms is to scale them by paralellizing and/or distributing the computations. A prominent example is Richtárik and Takáč (2016). In contrast to single CPU algorithms, this paper proposes distributed multicore coordinate descent methods which rely on very clever sub-block sampling techniques and can handle very efficiently huge data matrices with 2×10^9 rows, 10^9 features and up to 20×10^9 non-zero entries. Clearly, they will beat any single CPU algorithm.

Increasing amount of work is also being done trying to extend SVM solvers to the parallel setting, so they can take advantage of the many recent hardware advances in multiple core CPU and general purpose graphics processing units (GPUs). An empirical analysis of SVM parallelization for multi-core CPUs and GPU architectures is in Tyree et al. (*Parallel Support Vector Machines in Practice*) and such settings are exploited in Zhou et al. (2014). Parallelization requirements often entail working with specific solvers that may introduce problem simplifications (such as the homogeneous model assumption of LIBLINEAR). However, the analysis in Tyree et al. (*Parallel Support Vector Machines in Practice*) suggests that simpler, implicit approaches to SVM parallelism may result in substantial computational gains with a less costly programming effort. We can directly take advantages of this new parallel algorithms to solve in parallel, not only the ν-SVC, but also the equivalent Lasso instance thanks to the reduction presented here. It would be interesting to see, from a practical point of view, if solving the Lasso this way is faster than doing it directly through some of the specific parallel solvers available.

Finally, we observe that the Lasso and related problems receive a constant attention in many

application areas (Vidaurre et al., 2013; Xu et al., 2014; Ren et al., 2016) and, moreover, they are at the core of many other problems in convex regularized learning, such as Fused Lasso, wavelet smoothing or trend filtering, currently solved using specialized algorithms. A ν-SVC approach could provide for them the same faster convergence that we have illustrated here for the standard Lasso.

Chapter 6

Accelerating SVM training

In this chapter we are going to revisit two classical acceleration techniques, the Heavy Ball method from Polyak (1964) and Nesterov's Acceleration (Nesterov, 1983). Both of these algorithms involve adding some kind of momentum term to the basic gradient step, although they differ in the amount of information used. Heavy Ball combines the previous two iterates and the gradient at the current step, while Nesterov's method also adds the gradient at the last step. Another important difference that we will discuss in detail later is that the convergence analysis of Heavy Ball requires the objective function to be twice differentiable. On the other hand, Nesterov's Acceleration works for any convex function but its iterations are not monotone, rendering it unusable in some applications. However, this can be solved by checking the monotonicity of the function at every iteration.

Despite these two algorithms being around for quite some time, recently there has been a renewed research interest in the application of similar acceleration techniques to more modern models. In particular novel advances in deep neural networks suggest that adding momentum to the gradient step is beneficial for speeding up training. This also suggest a recent trend where old methods that were not very popular when they came out (probably because they were much ahead of their time) are being re-discovered and successfully applied to many problems.

Following the previous trend, we will explore the application of both regular momentum and Nesterov's momentum to the descent direction provided by the SMO algorithm. This procedure, as discussed earlier in Section 4.3.3, is probably still the state-of-the-art algorithm to solve non-linear SVM. Furthermore, recent research seems more focused on adapting the algorithm to modern hardware, such as parallelization and distributed computing, rather than in new algorithmic advances.

The dual objective function of the C-SVC is an instance of the more general convex quadratic optimization problem but with additional constraints. Furthermore, when using the Gaussian kernel the objective function is also strongly convex, since the kernel matrix is positive definite if all the data points are distinct. Thus this momentum could provide for SMO the same acceleration than for Gradient Descent. We will discuss in this chapter two new proposals.

The first option is to apply Nesterov's Accelerated Gradient (NAG) ideas to SMO. In section 6.1.1 we will consider the use of NAG for solving the dual SVM problem, and check whether it reduces the number of iterations of standard SMO. The second one tries to improve SMO's descent direction by adding a momentum term $\mathbf{m}^k = \boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}$. This is known in the classical convex optimization literature as the Heavy Ball algorithm, and we will show that it is equivalent to Conjugate Gradient Descent. As we will discuss in section 6.2.2 this momentum term implies some extra computation per iteration and, potentially, a projection step to ensure that the new coefficients satisfy the box constrains. We will derive a new conjugate version of SMO that,

1. only needs $\frac{4}{3}n$ extra float products per iteration and,
2. reduces the projection step to a simple clipping of the ρ coefficient.

Finally, we will empirically evaluate all the approaches to see if they perform less iterations than SMO and, as a final goal, also exhibit faster execution time.

6.1 Nesterov Accelerated Gradient

Previously we have seen in Section 3.3.2 that the plain Gradient Descent algorithm has a rate of convergence of order $1/k$ when minimizing a L-smooth convex function f. Recall that Gradient Descent is just a particular case of the Iterative Soft-Thresholding Algorithm, also called Proximal Point Algorithm or Projected Gradient Descent. Nesterov's Accelerated Gradient (NAG) (Nesterov, 1983) is an improved version which attains a rate of $1/k^2$. First define the sequences:

$$t_0 = 1, \quad t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}, \quad \text{and} \quad \mu_k = \frac{t_k - 1}{t_{k+1}}. \tag{6.1.1}$$

Then, the algorithm is given by the equations

$$\mathbf{x}^{k+1} = \mathbf{y}^k - \frac{1}{L}\nabla f(\mathbf{y}^k), \tag{6.1.2}$$

$$\mathbf{y}^{k+1} = \mathbf{x}^{k+1} + \mu_k(\mathbf{x}^{k+1} - \mathbf{x}^k), \tag{6.1.3}$$

starting from an arbitrary initial point $\mathbf{x}^0 = \mathbf{y}^0$. In other words, NAG performs a simple step of gradient descent to go from $\mathbf{y}^k$ to $\mathbf{x}^{k+1}$ and then brings $\mathbf{x}^{k+1}$ a little bit further in the direction of the previous point $\mathbf{x}^k$. The following theorem shows that NAG achieves the $\mathcal{O}(1/k^2)$ converge rate:

Theorem 6.1 (Theorem 1 in Nesterov, 1983)**.** *Let f be a convex an L-smooth function, then the sequence $\{x^k\}$ generated by Eqs.* (6.1.1) *to* (6.1.3) *satisfies*

$$f(\mathbf{x}^k) - f(\mathbf{x}^*) \leq \frac{4L\|\mathbf{y}^0 - \mathbf{x}^*\|_2^2}{(k+2)^2}. \tag{6.1.4}$$

The intuition behind the previous choices for t_k and μ_k is quite difficult to grasp. Recently, there has been a large research effort trying to understand why and when the acceleration is possible. These works involve re-interpreting NAG from a different point of view, either as a linear coupling of Gradient Descent and Mirror Descent (Allen-Zhu and Orecchia, 2014), as a discretization of a second-order ordinary differential equation (Su et al., 2014; Flammarion and Bach, 2015; Arjevani et al., 2015), from a geometrical perspective (Bubeck et al., 2015) or others (Lessard et al., 2016; Wibisono et al., 2016). There have also been some practical applications of Nesterov's Accelerated Gradient such as the already mentioned FISTA algorithm (Beck and Teboulle, 2009a), deep network training (Sutskever et al., 2013) and, more recently, adaptive restarts (O'Donoghue and Candès, 2015).

In practice, it is often very expensive or even impossible to compute the Lipschitz constant L. In those cases, we can rewrite Eq. (6.1.2) as

$$\mathbf{x}^{k+1} = \mathbf{y}^k - \eta_k \nabla f(\mathbf{y}^k),$$

where η_k may be interpreted as a learning rate that can change during the optimization, or even be dynamically estimated (Sutskever et al., 2013). As an example, Beck and Teboulle

(2009a) suggest a backtracking rule to estimate the value of η_k at every iteration. It is also worh mentioning that there have been other proposals for the sequence $\{t_k\}$. Chambolle and Dossal (2014) analyze the standard one, used by FISTA, (Eq. (6.1.1)) and conclude that, even though it is nearly optimal from a theoretical point of view, other sequences of the form

$$t_k = \frac{k+a+1}{a}$$

for $a \geq 2$ may have some desirable properties in practice.

The previous analysis can also be extended to strongly convex functions, obtaining a linear convergence rate for Nesterov's Accelerated Gradient. Recall that a function f is ℓ-strongly convex if for some $\ell > 0$ and all $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$,

$$f(\mathbf{z}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{z} - \mathbf{x}) + \frac{\ell}{2}\|\mathbf{z} - \mathbf{x}\|_2^2. \tag{6.1.5}$$

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable ℓ-strongly convex and L-smooth quadratic function, for instance

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x},$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a positive definite matrix and $\mathbf{b} \in \mathbb{R}^n$ is a vector. For such f, the above conditions are equivalent to the following spectral condition

$$\ell \mathbf{I} \preceq \mathbf{A} \preceq L\mathbf{I},$$

that is, ℓ and L are a lower and upper bound on the smallest and largest eigenvalues of $\mathbf{A}$ for all $\mathbf{x}$. The gradient of f can be computed as

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b},$$

so clearly f as an unique minimizer $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$. Let $\kappa = \frac{L}{\ell}$ be the condition number of the matrix $\mathbf{A}$, then the following theorem shows that Nesterov's Accelerated Gradient (Eqs. (6.1.2) and (6.1.3)) obtains a linear convergence rate for

$$\mu_k = \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}.$$

Theorem 6.2 (Theorem 3.18 in Bubeck, 2015)**.** *Let f be ℓ-strongly convex and L-smooth; then Nesterov's Accelerated Gradient satisfies*

$$f(\mathbf{x}^k) - f(\mathbf{x}^*) \leq \frac{\ell + L}{2}\|\mathbf{y}^0 - \mathbf{x}^*\|_2^2 \exp\left(-\frac{k-1}{\sqrt{\kappa}}\right). \tag{6.1.6}$$

Note that the previous theorem does not need f to be of the form described above. However, if f is a quadratic function the strong convexity assumption boils down to the matrix $\mathbf{A}$ being positive definite, and this will be useful in the following section. Again, this rate can only be obtained if we are able to compute both L and ℓ and, in practice, this is rarely the case. In the following sections we will:

- Present a basic set up for the application of NAG to the SMO algorithm.
- Propose a naïve version of Nesterov's Acceleration for SMO and a strictly monotone one, where the μ parameter in Nesterov's method is analytically computed.
- Numerically study both approaches and show that they do indeed reduce the number of iterations of standard SMO.

6.1.1 Naïve Nesterov Accelerated SMO

Recall that the C-SVC formulation solves the following dual problem (4.1.23),

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top \mathbf{1} \right\} \quad \text{s.t.} \quad \boldsymbol{\alpha}^\top \mathbf{y} = 0,\ 0 \leq \alpha_i \leq C,\ i = 1, \dots, n.$$

One of the main algorithms to solve the previous problem is Sequential Minimal Optimization (SMO) (Section 4.3.3). At every iteration SMO selects two coefficients and updates them by computing the gradient of the resulting quadratic problem analytically.

If one uses the Gaussian kernel and there is no pattern $\mathbf{x}$ that appears in the sample with both $y = 1$ and $y = -1$, the kernel matrix $\mathbf{Q}$ is positive definite (Chen et al., 2006), i.e. we have $\ell \mathbf{I} \preceq \mathbf{Q} \preceq L\mathbf{I}$, $0 \leq \ell \leq L$ or, in other words, $f(\boldsymbol{\alpha})$ is L-smooth and ℓ-strongly convex. Then, the sequence generated by the SMO algorithm $\{\boldsymbol{\alpha}^k\}$ globally converges to the unique minimum of (4.1.23) with a linear rate, i.e. $f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^*) \leq cr$ for some $r < 1$ at iteration k (Theorem 4.1). For such $\mathbf{Q}$, we have also seen in the previous section that Nesterov's Accelerated Gradient achieves a linear convergence rate. Since ℓ and L are not known, we cannot apply that version in practice but, in any case, we consider worth exploring the potential effectiveness of Nesterov's Acceleration for the SMO algorithm.

The first idea in order to apply Nesterov's procedure to SMO would be to first compute the intermediate point $\mathbf{x}$, then select the two SMO coefficients according to the gradient $\nabla f(\mathbf{y})$ and finally update the original point $\boldsymbol{\alpha}$. Assuming we are at iteration k, first we compute and intermediate point $\mathbf{x}$ using Eq. (6.1.3),

$$\mathbf{x}^{k+1} = \boldsymbol{\alpha}^k + \mu_k(\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}). \tag{6.1.7}$$

Then, the gradient at that point is

$$\mathbf{g}^{k+1} = \nabla f(\mathbf{x}^{k+1}) = \nabla f(\boldsymbol{\alpha}^k) + \mu_k(\nabla f(\boldsymbol{\alpha}^k) - \nabla f(\boldsymbol{\alpha}^{k-1})).$$

Next we select the working set (i, j) as in SMO (Eqs. (4.3.21) and (4.3.22)) using the previous gradient, to define the following descent direction

$$\mathbf{d}^{k+1} = \mathbf{y}_i \mathbf{e}_i - \mathbf{y}_j \mathbf{e}_j.$$

Equation (6.1.2) in Nesterov's Accelerated Gradient would be thus replaced by the update

$$\boldsymbol{\alpha}^{k+1} = \mathbf{x}^{k+1} + \rho_k \mathbf{d}^{k+1} \tag{6.1.8}$$

where

$$\rho_k = -\frac{\mathbf{d}^{k+1} \cdot \mathbf{g}^{k+1}}{\mathbf{d}^{k+1} \cdot \mathbf{Q}\mathbf{d}^{k+1}} \tag{6.1.9}$$

is the unconstrained step (Eq. (4.3.14)). To perform the previous computations efficiently, we can store the gradient and update it every iteration as in Eq. (4.3.23),

$$\nabla f(\boldsymbol{\alpha}^{k+1}) = \nabla f(\boldsymbol{\alpha}^k) + \rho \left(\mathbf{y}_i \mathbf{Q}_i - \mathbf{y}_j \mathbf{Q}_j \right).$$

Finally, we still need to show the feasibility of $\mathbf{x}^{k+1}$. First, it is easy to see that the constraint $\mathbf{x}^{k+1} \cdot \mathbf{y} = \sum x_i^{k+1} y_i = 0$ is met as long as both $\boldsymbol{\alpha}^k$ and $\boldsymbol{\alpha}^{k-1}$ are feasible. Let $\gamma = -\mu$; then we can rewrite Eq. (6.1.7) as

$$\mathbf{x}^{k+1} = (1 - \gamma)\boldsymbol{\alpha}^k + \gamma \boldsymbol{\alpha}^{k-1}.$$

Algorithm 14: Naïve Nesterov's Accelerated SMO

Input: $C > 0$
Initialize: $\boldsymbol{\alpha}^0 = \mathbf{0}$, $t_0 = 1$.
while *stopping condition not met* **do**
  Compute μ_k as in Eq. (6.1.1) and clip it
  $\mathbf{x}^{k+1} \leftarrow \boldsymbol{\alpha}^k + \mu_k(\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1})$
  $\mathbf{g}^{k+1} \leftarrow \nabla f(\boldsymbol{\alpha}^k) + \mu_k(\nabla f(\boldsymbol{\alpha}^k) - \nabla f(\boldsymbol{\alpha}^{k-1}))$
  Select working set (i, j)
  $\mathbf{d}^{k+1} = \mathbf{y}_i \mathbf{e}_i - \mathbf{y}_j \mathbf{e}_j$
  Compute ρ_k as in Eq. (6.1.9) and clip it
  $\boldsymbol{\alpha}^{k+1} \leftarrow \mathbf{x}^{k+1} + \rho_k \mathbf{d}^{k+1}$
  $\nabla f(\boldsymbol{\alpha}^{k+1}) \leftarrow \nabla f(\boldsymbol{\alpha}^k) + \rho\,(\mathbf{y}_i \mathbf{Q}_i - \mathbf{y}_j \mathbf{Q}_j)$
end

Multiplying both sides by $\mathbf{y}$

$$\mathbf{x}^{k+1} \cdot \mathbf{y} = (1-\gamma)\boldsymbol{\alpha}^k \cdot \mathbf{y} + \gamma \boldsymbol{\alpha}^{k-1} \cdot \mathbf{y}$$

and using the feasibility of both $\boldsymbol{\alpha}^k$ and $\boldsymbol{\alpha}^{k-1}$ we get the desired result. The other constraint $0 \leq \mathbf{x}^{k+1} \leq C$ is not immediately satisfied but it can be easily achievable, relying again on the feasibility of $\boldsymbol{\alpha}^k$ and clipping accordingly.

Suppose we have an update of the form $\mathbf{a}' = \mathbf{a} + \mu \mathbf{b}$ with $0 \leq a_i \leq C$ for all i and $\mu > 0$. We start by defining the index sets

$$\mathcal{P}^+ = \{i \mid b_i > 0\} \quad \text{and} \quad \mathcal{P}^- = \{i \mid b_i < 0\}.$$

Then, we want to make sure that the following inequalities holds

$$\begin{aligned} 0 < \mu \leq \frac{C - a_i}{b_i} \quad &\text{if} \quad i \in \mathcal{P}^+, \\ 0 < \mu \leq -\frac{a_i}{b_i} \quad &\text{if} \quad i \in \mathcal{P}^-. \end{aligned}$$

Thus, we take $\mu = \min\{\mu, \mu^+, \mu^-\}$ where

$$\mu^+ = \min\left\{ \frac{C - a_i}{b_i} \;\middle|\; i \in \mathcal{P}^+ \right\},$$

and

$$\mu^- = \min\left\{ -\frac{a_i}{b_i} \;\middle|\; i \in \mathcal{P}^- \right\}.$$

Note that Eq. (6.1.7) is of the previous form with $\mathbf{a} = \boldsymbol{\alpha}^k$, $\mathbf{b} = \boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}$ and $\mu = \mu_k$. Note also that if $\mu < 1$, $\mathbf{x}^{k+1}$ will automatically verify the box constraints. If $\mu > 1$ we can also simply clip it to $\mu = 1$, i.e. take $\mathbf{x}^{k+1} = \boldsymbol{\alpha}^k$ and perform a standard SMO update.

Once μ_k is clipped then $\mathbf{x}^{k+1}$ satisfies the box constraints and thus we can apply the same procedure to make sure that $\boldsymbol{\alpha}^{k+1}$ is also feasible, letting $\mathbf{a} = \mathbf{x}^{k+1}$, $\mathbf{b} = \mathbf{d}^{k+1}$ and $\mu = \rho_k$. In this case the clipping is performed in ρ_k instead of μ_k and it is easier since $\mathbf{b}$ only has two non-zero components.

The full pseudocode is given in Algorithm 14. The iteration complexity of the previous algorithm is:

- n operations to compute $\mathbf{x}^{k+1}$,
- n operations to compute $\mathbf{g}^{k+1}$,
- $2n$ operations to select the working set,
- n operations to update the gradient,

for a total cost of $\mathcal{O}(5n)$ operations, in contrast to the $\mathcal{O}(3n)$ needed by standard SMO. However, the convergence of Algorithm 14 is not guaranteed, since the NAG step is not monotone and SMO convergence relies upon the fact that the value of the objective function decreases at every iteration, i.e. $f(\boldsymbol{\alpha}^{k+1}) < f(\boldsymbol{\alpha}^k)$. We will solve this in the next section by deriving a monotone version of the previous procedure.

6.1.2 Monotone Nesterov's Accelerated SMO

A possible solution is to use the ideas of the monotone FISTA algorithm suggested by Beck and Teboulle (2009b). This version works by computing the new coefficient $\boldsymbol{\alpha}'$ as in equation (6.1.8) and checking whether $f(\boldsymbol{\alpha}') < f(\boldsymbol{\alpha}^k)$. If that is the case, $\boldsymbol{\alpha}'$ becomes the new $\boldsymbol{\alpha}^{k+1}$. Otherwise we keep the old $\boldsymbol{\alpha}^k$ and compute a new intermediate point $\mathbf{y}^{k+1}$ as

$$\mathbf{z}^{k+1} = \gamma\boldsymbol{\alpha}' + (1-\gamma)\boldsymbol{\alpha}^k,$$

where $\gamma = t_k/t_{k+1}$ and

$$\boldsymbol{\alpha}' = \mathbf{x}^{k+1} + \rho_k\mathbf{d}^{k+1}.$$

Then, the new coefficient is computed as

$$\boldsymbol{\alpha}^{k+1} = \mathbf{z}^{k+1} + \rho_k\mathbf{d}^{k+1}.$$

Since $0 \leq \gamma \leq 1$, $\mathbf{z}^{k+1}$ can be seen as a convex combination of $\boldsymbol{\alpha}'$ and $\boldsymbol{\alpha}^k$ and thus it is automatically feasible. However, it is possible that even after using $\mathbf{z}^{k+1}$ to compute the new $\boldsymbol{\alpha}^{k+1}$ we end up with a possibly long sequence of coefficients where the value of the function remains constant. It is also worth mentioning that in order to check if the function decreases we have to compute its value, which is costly in the case of the SVC dual objective function.

We will take advantage of the quadratic structure of the function f to compute the exact value of γ that guarantees $f(\mathbf{z}^{k+1}) < f(\boldsymbol{\alpha}^k)$. Let

$$\mathbf{z}_\gamma = \gamma\boldsymbol{\alpha}' + (1-\gamma)\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^k + \gamma(\boldsymbol{\alpha}' - \boldsymbol{\alpha}^k) = \boldsymbol{\alpha}^k + \gamma\Delta,$$

where $\Delta = \boldsymbol{\alpha}' - \boldsymbol{\alpha}^k$. Then the SVM dual function evaluated at $\mathbf{z}_\gamma$ is

$$\begin{aligned} f(\mathbf{z}_\gamma) &= \frac{1}{2}(\boldsymbol{\alpha}^k + \gamma\Delta)\cdot\mathbf{Q}(\boldsymbol{\alpha}^k + \gamma\Delta) - \sum[\boldsymbol{\alpha}^k + \gamma\Delta]_i \\ &= \frac{1}{2}(\boldsymbol{\alpha}^k\cdot\mathbf{Q}\boldsymbol{\alpha}^k + 2\gamma\boldsymbol{\alpha}^k\cdot\mathbf{Q}\Delta + \gamma^2\Delta\cdot\mathbf{Q}\Delta) - \sum\alpha_i^k - \gamma\sum\Delta_i. \end{aligned}$$

The derivative with respect to γ is,

$$\begin{aligned} \frac{\partial f(\mathbf{z}_\gamma)}{\partial\gamma} &= \boldsymbol{\alpha}^k\cdot\mathbf{Q}\Delta + \gamma\Delta\cdot\mathbf{Q}\Delta - \sum\Delta_i \\ &= \gamma\Delta\cdot\mathbf{Q}\Delta + \Delta\cdot(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{1}) \\ &= \gamma\Delta\cdot\mathbf{Q}\Delta + \Delta\cdot\nabla f(\boldsymbol{\alpha}^k) \end{aligned}$$

and the minimum is reached at

$$\gamma^* = -\frac{\Delta \cdot \nabla f(\boldsymbol{\alpha}^k)}{\Delta \cdot \mathbf{Q}\Delta} = -\frac{(\boldsymbol{\alpha}' - \boldsymbol{\alpha}^k) \cdot \nabla f(\boldsymbol{\alpha}^k)}{(\boldsymbol{\alpha}' - \boldsymbol{\alpha}^k)\mathbf{Q}(\boldsymbol{\alpha}' - \boldsymbol{\alpha}^k)}. \tag{6.1.10}$$

To compute the previous value as efficiently as possible, note that we need the value of the function $f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}\cdot\mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}\cdot\mathbf{1}$ to verify the monotonicity, which implies $\boldsymbol{\alpha}\cdot\mathbf{Q}\boldsymbol{\alpha} = 2(f(\boldsymbol{\alpha}) + \boldsymbol{\alpha}\cdot\mathbf{1})$. The value of the function can also be written as a function of the gradient

$$\begin{aligned} f(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{\alpha}\cdot\mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}\cdot\mathbf{1} \\ &= \frac{1}{2}(\boldsymbol{\alpha}\cdot\mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}\cdot\mathbf{1}) - \frac{1}{2}\boldsymbol{\alpha}\cdot\mathbf{1} \\ &= \frac{1}{2}(\boldsymbol{\alpha}\cdot(\mathbf{Q}\boldsymbol{\alpha} - \mathbf{1})) - \frac{1}{2}\boldsymbol{\alpha}\cdot\mathbf{1} \\ &= \frac{1}{2}(\boldsymbol{\alpha}\cdot\nabla f(\boldsymbol{\alpha}) - \boldsymbol{\alpha}\cdot\mathbf{1}). \end{aligned}$$

Using the previous equivalences we can rewrite Eq. (6.1.10) as

$$\begin{aligned} \gamma^* &= -\frac{(\boldsymbol{\alpha}' - \boldsymbol{\alpha}^k)\cdot\nabla f(\boldsymbol{\alpha}^k)}{\boldsymbol{\alpha}'\cdot\mathbf{Q}\boldsymbol{\alpha}' + \boldsymbol{\alpha}^k\cdot\mathbf{Q}\boldsymbol{\alpha}^k - 2\boldsymbol{\alpha}'\cdot\mathbf{Q}\boldsymbol{\alpha}^k} \\ &= -\frac{\boldsymbol{\alpha}'\cdot\nabla f(\boldsymbol{\alpha}^k) - \boldsymbol{\alpha}^k\cdot\nabla f(\boldsymbol{\alpha}^k)}{2f(\boldsymbol{\alpha}') + 2\boldsymbol{\alpha}'\cdot\mathbf{1} + 2f(\boldsymbol{\alpha}^k) + 2\boldsymbol{\alpha}^k\cdot\mathbf{1} - 2\boldsymbol{\alpha}'\cdot\mathbf{Q}\boldsymbol{\alpha}^k} \\ &= -\frac{\boldsymbol{\alpha}'\cdot\nabla f(\boldsymbol{\alpha}^k) - 2f(\boldsymbol{\alpha}^k) - \boldsymbol{\alpha}^k\cdot\mathbf{1}}{2f(\boldsymbol{\alpha}') + 2f(\boldsymbol{\alpha}^k) + 2\boldsymbol{\alpha}^k\cdot\mathbf{1} - 2\boldsymbol{\alpha}'\cdot(\mathbf{Q}\boldsymbol{\alpha}^k + \mathbf{1})} \\ &= \frac{1}{2}\frac{2f(\boldsymbol{\alpha}^k) + \boldsymbol{\alpha}^k\cdot\mathbf{1} - \boldsymbol{\alpha}'\cdot\nabla f(\boldsymbol{\alpha}^k)}{f(\boldsymbol{\alpha}') + f(\boldsymbol{\alpha}^k) + \boldsymbol{\alpha}^k\cdot\mathbf{1} - \boldsymbol{\alpha}'\cdot\nabla f(\boldsymbol{\alpha}^k)} \\ &= \frac{1}{2}\frac{2f(\boldsymbol{\alpha}^k) + s_k - \mathbf{a}}{f(\boldsymbol{\alpha}') + f(\boldsymbol{\alpha}^k) + s_k - \mathbf{a}}, \end{aligned} \tag{6.1.11}$$

where $s_k = \boldsymbol{\alpha}^k\cdot\mathbf{1}$ and $\mathbf{a} = \boldsymbol{\alpha}'\cdot\nabla f(\boldsymbol{\alpha}^k)$. The pseudocode is shown in Algorithm 15.

Lines 2 to 9 are the same as Algorithm 14. After obtaining $\boldsymbol{\alpha}^{k+1}$, we then compute $f(\boldsymbol{\alpha}^{k+1})$ and check in Line 12 if the value of the objective function has decreased. If that is not the case, we compute the exact value $\mu_k = \gamma^*$ using Eq. (6.1.11) that satisfies the inequality and start again the iteration from Line 3.

To estimate the complexity of the previous steps, note that we need an extra n products to compute the value of the objective function in Line 11, on top of the $5n$ from the naïve procedure. If $f(\boldsymbol{\alpha}^{k+1}) > f(\boldsymbol{\alpha}^k)$, we need another n products to compute $\mathbf{a}$ and $6n$ to repeat the iteration, for a total of $\mathcal{O}(13n)$ products. Note that the values $f(\boldsymbol{\alpha}^k)$ and $\nabla f(\boldsymbol{\alpha}^k)$ are kept from the previous iterations, so they do not introduce any extra cost. Summing things up, a standard iteration of Algorithm 15 would cost $\mathcal{O}(6n)$ products, that is, twice of SMO's iteration, while a monotone iteration would be $\mathcal{O}(13n)$. Of course, we would expect non-monotone iterations to be much less frequent than monotone ones. In any case, the reduction of Algorithm 15 in the number of iterations would have to be quite large with respect to SMO's to make the extra cost per iteration worth it. For that reason we are going to explore in the next section a different algorithm, Conjugate Gradient Descent. This algorithm is another modification of gradient descent that also adds a momentum term but guarantees monotone descent at every iteration.

Algorithm 15: Monotone Nesterov's Accelerated SMO (MNAS)

Input: $C > 0$
Initialize: $\boldsymbol{\alpha}^0 = \mathbf{0}$, $t_0 = 1$.
1 **while** *stopping condition not met* **do**
2 Compute μ_k as in Eq. (6.1.1) and clip it
3 $\mathbf{x}^{k+1} \leftarrow \boldsymbol{\alpha}^k + \mu_k(\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1})$
4 $\mathbf{g}^{k+1} \leftarrow \nabla f(\boldsymbol{\alpha}^k) + \mu_k(\nabla f(\boldsymbol{\alpha}^k) - \nabla f(\boldsymbol{\alpha}^{k-1}))$
5 Select working set (i, j)
6 $\mathbf{d}^{k+1} = y_i \mathbf{e}_i - y_j \mathbf{e}_j$
7 Compute ρ_k as in Eq. (6.1.9) and clip it
8 $\boldsymbol{\alpha}^{k+1} \leftarrow \mathbf{x}^{k+1} + \rho_k \mathbf{d}^{k+1}$
9 $\nabla f(\boldsymbol{\alpha}^{k+1}) \leftarrow \nabla f(\boldsymbol{\alpha}^k) + \rho\,(y_i \mathbf{Q}_i - y_j \mathbf{Q}_j)$
10 $s_{k+1} = \boldsymbol{\alpha}^{k+1} \cdot \mathbf{1}$
11 $f(\boldsymbol{\alpha}^{k+1}) = \frac{1}{2}(\boldsymbol{\alpha}^{k+1} \cdot \nabla f(\boldsymbol{\alpha}^{k+1}) - s_{k+1})$
12 **if** $f(\boldsymbol{\alpha}^{k+1}) > f(\boldsymbol{\alpha}^k)$ **then**
13 $\mathbf{a} = \boldsymbol{\alpha}^{k+1} \cdot \nabla f(\boldsymbol{\alpha}^k)$
14 $\mu_k = \dfrac{1}{2} \dfrac{2f(\boldsymbol{\alpha}^k) + s_{ks} - \mathbf{a}}{f(\boldsymbol{\alpha}^{k+1}) + f(\boldsymbol{\alpha}^k) + s_k - \mathbf{a}}$
15 **go to** Line 3
16 **end**
17 **end**

6.2 Conjugate Gradient Descent

We start again with the simple gradient descent algorithm, where the weights are updated by making a small step in the gradient direction,

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \eta \nabla f(\mathbf{x}^k).$$

We have already seen that for step-size η small enough gradient descent has a monotone improvement at every iteration and it always converges to a minimum. Furthermore, if the function f is strongly convex gradient descent converges linearly, i.e. there exists $r \in (0, 1)$ such that

$$\|\mathbf{x}^k - \mathbf{x}^*\| \leq r^k \|\mathbf{x}^0 - \mathbf{x}^*\|.$$

The scalar r is known as convergence factor. For gradient descent the optimal convergence factor is

$$r = \frac{L - \ell}{L + \ell},$$

attained for

$$\eta = 2/(L + \ell). \tag{6.2.1}$$

This factor can also be written in terms of the condition number $\kappa = L/\ell$,

$$r = \frac{L - \ell}{L + \ell} = \frac{\kappa - 1}{\kappa + 1} = \left(1 - \frac{2}{\kappa + 1}\right). \tag{6.2.2}$$

However, this convergence can be improved by adding a momentum term,

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \eta \nabla f(\mathbf{x}^k) + \beta(\mathbf{x}^k - \mathbf{x}^{k-1}). \tag{6.2.3}$$

In the classic convex optimization literature this is known as the Heavy Ball method (Polyak, 1964) and belongs to the family of multi-step methods, where the next iterate depends not only on the current iterate but also on the previous ones. When the objective function is twice continuously differentiable, strongly convex and has Lipstchitz continuous gradient, the previous algorithm has a convergence factor

$$r = 1 - \frac{2}{\sqrt{\kappa} + 1},$$

attained for (Wright, 2013)

$$\eta = \frac{4}{L\left(1 + \frac{1}{\sqrt{\kappa}}\right)^2} \quad \text{and} \quad \beta = \left(1 - \frac{2}{\sqrt{\kappa} + 1}\right)^2. \tag{6.2.4}$$

The previous factor leads to a faster convergence than gradient descent (Eq. (6.2.2)). This difference increases with the condition number κ, so in practice the improvement is significant for poorly conditioned problems. As an example, to reduce $\|\mathbf{x}^k - \mathbf{x}^*\|_2$ by a factor ϵ we approximately need

$$k \geq \frac{\kappa}{2}|\log \epsilon|$$

for Gradient Descent and

$$k \geq \frac{\sqrt{\kappa}}{2}|\log \epsilon|$$

for the Heavy Ball method, which translates to $\sqrt{\kappa}$ times more iterations in the case of Gradient Descent. For a realistic value of $\kappa = 1000$ this is approximately 30 times less iterations when using momentum (Wright, 2013). Consider also the simple two variable convex quadratic model,

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x},$$

with

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 15 \\ 15 \end{pmatrix}.$$

Since $\mathbf{A}$ is positive definite this problem is strongly convex and has an unique minimum obtained at $\mathbf{x}^* = (15, 1.5)^\top$. We also know that $L = 10$ and $\ell = 1$ and thus we can compute the condition number $\kappa = L/\ell = 10$. Then, for the optimal choices for η and β given by Eqs. (6.2.1) and (6.2.4) both Gradient Descent and Heavy Ball obtain a linear convergence, but the latter should have a better convergence rate.

This is shown in Fig. 6.1, where we plot the level curves of the quadratic objective function, together with the sequences generated by Gradient Descent (black) and the Heavy Ball method (red). In the figure we can see the theoretical linear convergence (also called geometrical) of both methods, but the Heavy Ball method arrives faster to the optimum. As we mentioned before, the difference would have been even more significant for larger condition numbers.

The Heavy Ball method is strongly related to Nesterov's momentum,

$$\mathbf{x}^{k+1} = \mathbf{y}^k - \eta_k \nabla f(\mathbf{y}^k), \tag{6.2.5}$$

$$\mathbf{y}^{k+1} = \mathbf{x}^{k+1} + \mu_k(\mathbf{x}^{k+1} - \mathbf{x}^k). \tag{6.2.6}$$

Now let $\mathbf{m}^{k+1} = \mathbf{x}^{k+1} - \mathbf{x}^k$, then we can rewrite the second equation as

$$\mathbf{y}^{k+1} = \mathbf{x}^{k+1} + \mu_k \mathbf{m}^{k+1}$$

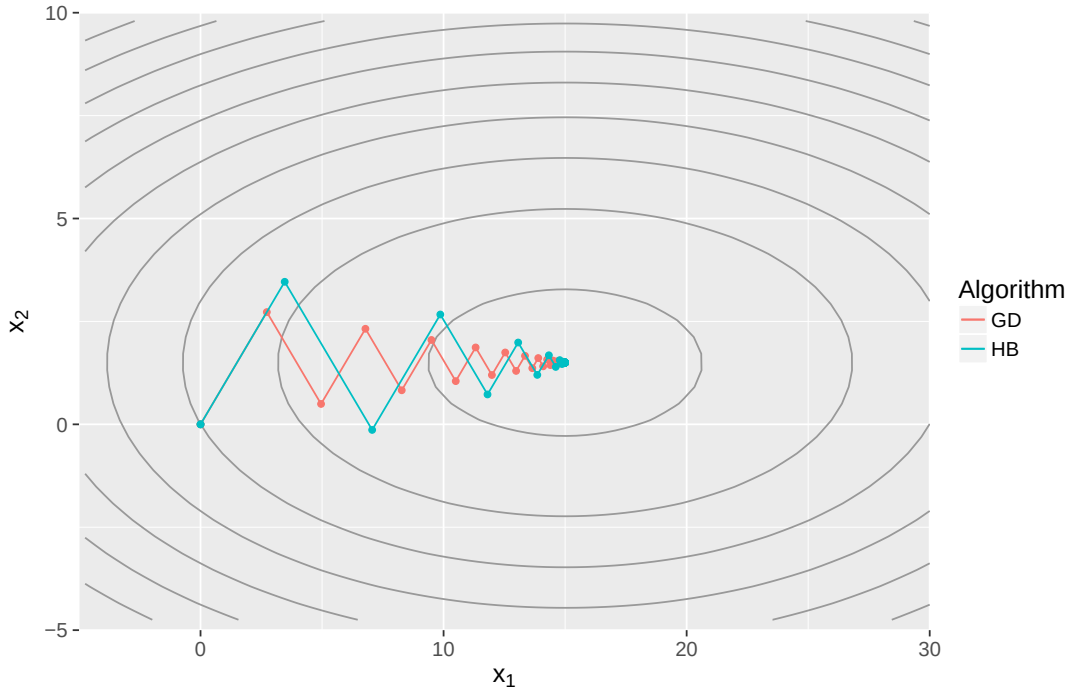

Figure 6.1: Convergence of Gradient Descent (GD) and the Heavy Ball (HB) methods for the optimal choices of η and β.

and thus $\mathbf{y}^k = \mathbf{x}^k + \mu_{k-1}\mathbf{m}^k$. Substituting into the first equation we get

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \eta_k \nabla f(\mathbf{x}^k + \mu_{k-1}\mathbf{m}^k) + \mu_{k-1}\mathbf{m}^k.$$

This is the same as the classical momentum but using $\nabla f(\mathbf{x}^k + \mu_{k-1}\mathbf{m}^k)$ instead of $\nabla f(\mathbf{x}^k)$,

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \eta_k \nabla f(\mathbf{x}^k) + \beta_k \mathbf{m}^k.$$

It is not surprising that Heavy Ball and Nesterov's Accelerated Gradient share the same fast convergence, although NAG factor is slightly worse,

$$r = 1 - \frac{1}{\sqrt{\kappa}}.$$

However, note that the analysis for the Heavy Ball method requires the function to be twice differentiable, while Nesterov's Acceleration only requires strong convexity. Another important difference is that the Heavy Ball method uses the information about the previous two iterates when computing the next, but in contrast to Nesterov's method it only includes the gradient at the current one.

Therefore, much like Nesterov's Acceleration, adding a momentum term to the SMO algorithm could help improving the descent direction. We face again the problem of computing the parameters η and β, since in practice L and ℓ are unknown. In this case, Fig. 6.2 shows, using the same simple example as before, that adding a momentum term is still beneficial as long as η and β are properly tuned.

Figure 6.2 illustrates the common "zig-zagging" behavior of Gradient Descent, which is partially corrected by the momentum term. Using the well-known analogy, we can interpret the momentum term as a heavy ball rolling down a hill, adding a certain inertia to the direction of

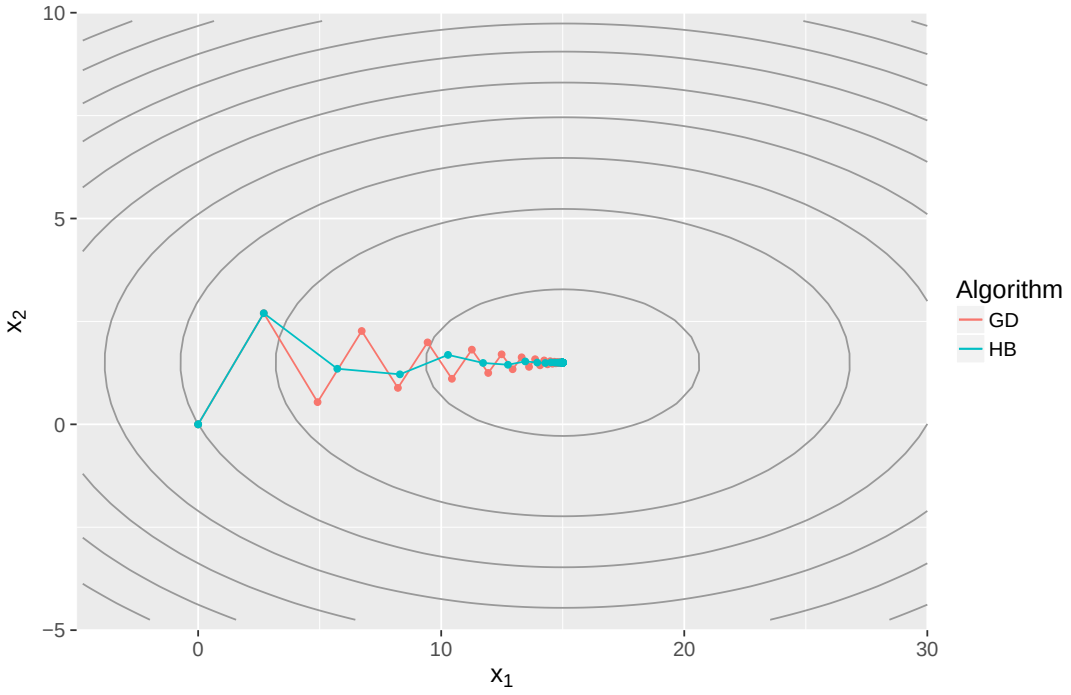

Figure 6.2: Convergence of Gradient Descent (GD) and the Heavy Ball (HB) methods for $\eta = 0.18$ and $\beta = 0.3$.

the gradient. This inertia has a "damping" effect, both smoothing and accelerating the path to the minimum (Goh, 2017). Here the key is how to select η and β to guarantee a fast convergence.

To partially solve the previous problem we are going to introduce a more practical version of the Heavy Ball method, known as Conjugate Gradient Descent (Nocedal and Wright, 2006). The basic step is

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \eta_k \mathbf{p}^k, \quad \mathbf{p}^k = -\nabla f(\mathbf{x}^k) + \gamma_k \mathbf{p}^{k-1}. \tag{6.2.7}$$

Although Conjugate Gradient looks in theory quite different from the Heavy Ball method, they both exploit the idea of using information not only from the current iteration but also the previous ones. In fact, it turns out that they are both equivalent for an appropriate choice of γ_k. Let $\mathbf{m} = \mathbf{x}^k - \mathbf{x}^{k-1}$ be the momentum term; then Eq. (6.2.3) can be rewritten as

$$\begin{aligned}
\mathbf{x}^{k+1} &= \mathbf{x}^k - \eta_k \nabla f(\mathbf{x}^k) + \beta_k (\mathbf{x}^k - \mathbf{x}^{k-1}) \\
&= \mathbf{x}^k + \eta_k \left(-\nabla f(\mathbf{x}^k) + \frac{\beta_k}{\eta_k} \mathbf{m}^k \right) \\
&= \mathbf{x}^k + \eta_k \mathbf{p}^k,
\end{aligned}$$

where $\mathbf{p}^k = -\nabla f(\mathbf{x}^k) + (\beta_k/\eta_k)\mathbf{m}^k$. Now, since $\mathbf{x}^k = \mathbf{x}^{k-1} + \eta_{k-1}\mathbf{p}^{k-1}$ we have

$$\mathbf{m}^k = \mathbf{x}^k - \mathbf{x}^{k-1} = \mathbf{x}^{k-1} + \eta_{k-1}\mathbf{p}^{k-1} - \mathbf{x}^{k-1} = \eta_{k-1}\mathbf{p}^{k-1}$$

and

$$\mathbf{p}^k = -\nabla f(\mathbf{x}^k) + \frac{\beta_k}{\eta_k} \eta_{k-1} \mathbf{p}^{k-1}.$$

Finally, letting

$$\gamma_k = \frac{\beta_k \eta_{k-1}}{\eta_k}$$

we get back Conjugate Gradient Descent.

Although harder to analyze, Conjugate Gradient Descent maintains the same fast convergence as the Heavy Ball method but it can be implemented without knowledge of L and ℓ. In practice η_k can be computed at every iteration by approximately minimizing the objective function f along the direction $\mathbf{p}^k$. In addition, if f is convex quadratic we can also choose (Nocedal and Wright, 2006)

$$\gamma_k = \frac{\|\nabla f(\mathbf{x}^k)\|_2^2}{\|\nabla f(\mathbf{x}^{k-1})\|_2^2}.$$

6.2.1 Conjugate MDM

Recall from Section 5.3 that given a set of n points in $\mathbb{R}^d$, $\mathcal{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$, the Minimum Norm Problem can be defined as finding the point $\mathbf{x}$ in the convex hull $\text{conv}(\mathcal{S})$ closest to the origin. More formally, the MNP can be written as the following minimization problem:

$$\min_{\boldsymbol{\alpha}} \{\boldsymbol{\alpha} \cdot \mathbf{Q}\boldsymbol{\alpha}\} \quad \text{s.t.} \quad 0 \leq \boldsymbol{\alpha} \leq 1, \sum \alpha_i = 1, \tag{6.2.8}$$

where $\mathbf{Q}$ denotes the kernel matrix, $\mathbf{Q}_{i,j} = \mathbf{x}_i \cdot \mathbf{x}_j$. Although this problem is one of the simplest convex constrained optimization problem, it has received more or less constant attention over the past 50 years. In fact, it was one of the problems considered by Frank and Wolfe in his seminal paper (Frank and Wolfe, 1956), it is intimately linked with the SVM problem (López and Dorronsoro, 2015), it has been recently revisited, accompanying the renewed interest in Frank-Wolfe (FW) optimization (Clarkson, 2010) and it has also been shown to be equivalent to the constrained version of the Lasso problem (Jaggi, 2014; Alaíz et al., 2015).

Starting from an initial guess $\boldsymbol{\alpha}^0$, MNP is usually solved by iterating

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \mathbf{d}^k,$$

where $\mathbf{d}^k$ is some descent direction (i.e., $\mathbf{d}^k \cdot \nabla f(\boldsymbol{\alpha}^k) < 0$). In the original Frank-Wolfe algorithm $\mathbf{d}^k$ is given by

$$\mathbf{d}^k_{\text{FW}} = \mathbf{e}_l - \boldsymbol{\alpha}^k$$

with $l = \text{argmin}_j \nabla f(\boldsymbol{\alpha}^k)_j$. Then, for $0 \leq \rho_k \leq 1$, we get $\boldsymbol{\alpha}^{k+1} = (1-\rho_k)\boldsymbol{\alpha}^k + \rho_k \mathbf{e}_l$, or in other words, the new value of the coefficient is a convex combination of the previous coefficient and $\mathbf{e}_l$. This means that it lies in Δ_n since both points belong also to the simplex, an thus it is automatically feasible. Therefore the main advantage of the original Frank-Wolfe algorithm is that its iterations are projection-free. On the other hand, its main drawback is that the $\mathbf{d}^k_{\text{FW}}$ direction may result in a slow sublinear convergence (Gilbert, 1966) that cannot be improved. Several variants of Frank-Wolfe, such as the away steps in Wolfe (1970) and GuéLat and Marcotte (1986) have been suggested; the better suited for the MNP is the Mitchell-Dem'yanov-Malozemov (MDM) algorithm (referred as the swap FW method in Ñanculef et al. (2014)).

The MDM descent direction is $\mathbf{d}^k = \mathbf{e}_l - \mathbf{e}_u$ with $u = \text{argmax}_j \left\{ \nabla f(\boldsymbol{\alpha}^k)_j \;\middle|\; \alpha^k_j > 0 \right\}$. Note that the update

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k(\mathbf{e}_l - \mathbf{e}_u)$$

may not lie in Δ_n, i.e., MDM is not projection–free, but, if needed, the projection turns out to be a simple clip of ρ_k. MDM is guaranteed to have linear convergence (López and Dorronsoro, 2015), that is, to have $\|\boldsymbol{\alpha}^{k+1} - \boldsymbol{\alpha}^*\| \leq r\|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^*\|$, with $r < 1$. In addition, its iterations only need n products to update the gradient

$$\nabla f(\boldsymbol{\alpha}^{k+1}) = \nabla f(\boldsymbol{\alpha}^k) + \rho_k(\mathbf{Q}_l - \mathbf{Q}_u),$$

since MDM's descent direction has only two non zero components (standard FW may need close to $2n$ products to update $\boldsymbol{\alpha}^{k+1}$ and $\nabla f(\boldsymbol{\alpha}^{k+1})$). However, MDM linear convergence is only guaranteed once the face of $\text{conv}(\mathcal{S})$ facing the origin is reached, and it is easy to find examples where r is very close to 1 and convergence is thus slow.

Now, following the Conjugate Gradient Descent algorithm, we are going to replace the descent direction $\mathbf{d}^k$ by a conjugate direction $\mathbf{p}^k$,

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \mathbf{p}^k,$$

where $\mathbf{p}^k = \mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}$. In the same fashion as Conjugate Descent, we choose the unconstrained $\tilde{\rho}_k$ to minimize the objective function $f(\boldsymbol{\alpha})$ along $\mathbf{p}^k$, i.e

$$\nabla f(\boldsymbol{\alpha}^k + \tilde{\rho}_k \mathbf{p}^k) = 2\boldsymbol{\alpha}^k \cdot \mathbf{Q}\mathbf{p}^k + \tilde{\rho}_k \mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k = 0.$$

This yields

$$\begin{aligned} \tilde{\rho}_k^* &= \frac{-\nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{p}^k}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k} \\ &= \frac{-\nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{d}^k - \gamma_k \nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{p}^{k-1}}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k} \end{aligned} \tag{6.2.9}$$

$$\tag{6.2.10}$$

If the previous line minimization along $\mathbf{p}^{k-1}$ has been unclipped, i.e. the following orthogonality condition must hold

$$\nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{p}^{k-1} = 0.$$

With that in mind we can simplify Eq. (6.2.9) to

$$\tilde{\rho}_k = \frac{-\nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{d}^k}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k}.$$

Note that the direction $\mathbf{p}^k$ is always a descent direction, $\nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{p}^k = \nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{d}^k < 0$. The unconstrained gain in f is now

$$f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^{k+1}) = \frac{1}{2} \frac{(\nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{d}^k)^2}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k} \tag{6.2.11}$$

and we can maximize that gain by choosing γ_k to minimize the denominator in Eq. (6.2.11), which is a function of γ_k, namely

$$\phi(\gamma_k) = \mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k = (\mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}) \cdot \mathbf{Q}(\mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}).$$

The derivative is

$$\phi'(\gamma_k) = 2\gamma_k \mathbf{p}^{k-1} \cdot \mathbf{Q}\mathbf{p}^{k-1} + 2\mathbf{d}^k \mathbf{Q}\mathbf{p}^{k-1},$$

and equating to zero we get the value for the minimum

$$\gamma_k^* = -\frac{\mathbf{d}^k \cdot \mathbf{Q}\mathbf{p}^{k-1}}{\mathbf{p}^{k-1} \cdot \mathbf{Q}\mathbf{p}^{k-1}}.$$

We can summarize now our conjugate MDM updates. If the iteration ending in $\boldsymbol{\alpha}^k$ along $\mathbf{p}^{k-1}$ has not been clipped, we

Algorithm 16: Conjugate MDM (CMDM)

Initialize: $\boldsymbol{\alpha}^0 = \mathbf{g}^0 = \mathbf{0}, \mathbf{p}^{-1} = \mathbf{q}^{-1} = \mathbf{0}$, $\delta^{-1} = 1$

while *stopping condition not met* **do**
- $l = \operatorname{argmin}_i \ g_i$
- $u = \operatorname{argmax}_{\alpha_i^k > 0} \ g_i$
- Compute the kernel matrix columns $\mathbf{Q}_l$ and $\mathbf{Q}_u$
- $\gamma_k = (\mathbf{q}_l^{k-1} - \mathbf{q}_u^{k-1})/\delta_{k-1}$
- $\mathbf{p}^k = \mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}$
- $\mathbf{q}^k = \mathbf{Q}_l - \mathbf{Q}_u + \gamma_k \mathbf{q}^{k-1}$
- $\delta_k = \mathbf{q}_l^k - \mathbf{q}_u^k$
- Compute $\tilde{\rho}_k$ using Eq. (6.2.12) and clip it to ρ_k if needed
- $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \mathbf{p}^k$
- $\mathbf{g}^{k+1} = \mathbf{g}^k + \rho_k \mathbf{q}^k$
- **if** $\tilde{\rho}_k$ *was clipped* **then**
  - $\mathbf{q}^k = \mathbf{p}^k = \mathbf{0}$
  - $\delta_k = 1$
- **end**

end

1. Compute γ_k and $\mathbf{p}^k$,
2. Compute $\tilde{\rho}_k$ and $\boldsymbol{\alpha}^{k+1}$ and, finally,
3. Check whether $\boldsymbol{\alpha}^{k+1}$ lies in the simplex Δ_n and, if not, clip $\tilde{\rho}_k$ accordingly.

On the other hand, if clipping has happened, i.e. the boundary of Δ_n has been hit, we simply reset $\mathbf{p}^k = \mathbf{d}^k$ as it may lead to further boundary hits. Then, we just perform a standard MDM update in the next iteration.

Working with MDM descent directions $\mathbf{d} = \mathbf{e}_l - \mathbf{e}_u$ greatly simplifies the previous computations. Let

$$\mathbf{q}^k = \mathbf{Q}\mathbf{p}^k = \mathbf{Q}(\mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}) = \mathbf{Q}_l - \mathbf{Q}_u + \gamma_k \mathbf{q}^{k-1}$$

and

$$\delta_k = \mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k = \mathbf{d}^k \cdot \mathbf{Q}\mathbf{p}^k = \mathbf{d}^k \cdot \mathbf{q}^k = \mathbf{q}_l^k - \mathbf{q}_u^k.$$

Then we have

$$\gamma_k = -\frac{\mathbf{d}^k \cdot \mathbf{Q}\mathbf{p}^{k-1}}{\mathbf{p}^{k-1} \cdot \mathbf{Q}\mathbf{p}^{k-1}} = -\frac{\mathbf{d}^k \cdot \mathbf{q}^{k-1}}{\delta_{k-1}} = \frac{\mathbf{q}_u^{k-1} - \mathbf{q}_l^{k-1}}{\delta_{k-1}},$$

where $\mathbf{Q}_j$ denotes $\mathbf{Q}$'s jth column and

$$\tilde{\rho}_k = \frac{-\nabla f(\boldsymbol{\alpha}^k) \cdot \mathbf{d}^k}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k} = \frac{\mathbf{g}_u - \mathbf{g}_l}{\delta_k} \tag{6.2.12}$$

where $\mathbf{g}^k = \nabla f(\boldsymbol{\alpha}^k)$ is the gradient of the objective function at $\boldsymbol{\alpha}^k$. Note that the gradient can be efficiently updated as

$$\mathbf{g}^{k+1} = \nabla f(\boldsymbol{\alpha}^k + \rho_k \mathbf{p}^k) = \nabla f(\boldsymbol{\alpha}^k) + \rho_k \mathbf{Q}\mathbf{p}^k = \mathbf{g}^k + \rho_k \mathbf{q}^k.$$

This results in the general conjugate MDM precedure shown in Algorithm 16. We can recover standard MDM from the previous algorithm by setting $\gamma = 0$, $\mathbf{p} = \mathbf{d}$ and $\mathbf{q} = \mathbf{Q}_l - \mathbf{Q}_u$. If n_p and

n_q denote the number of non-zero components of $\mathbf{p}$ and $\mathbf{q}$, the cost in products of each iteration is

1. n_p to update $\mathbf{p}$ in Line 6 and $\boldsymbol{\alpha}$ in Line 10,
2. n_q to update $\mathbf{q}$ in Line 7 and
3. n to update the gradient $\mathbf{g}$ in Line 11.

We expect $n_q \simeq n$ but $n_p \ll n$ and similarly the number of non-zero $\boldsymbol{\alpha}_i$ should also be $\ll n$. Thus a conjugate iteration should have a theoretical cost about twice as large as that of a standard MDM iteration and should lead to faster training if the number of MDM iterations is more than twice the number of Conjugate MDM (CMDM) ones. In any case, note that the cost of the iterations in which a non-cached kernel column matrix has to be computed will require a much larger $n \times d$ number of products when working with patterns in a d dimensional space or even more in a kernel setting.

We finish this section by briefly discussing the clipping step. First, note that if $\sum_j p_j^{k-1} = 0$, we also have $\sum_j p_j^k = 0$ and, hence, $\sum_j \alpha_j^{k+1} = 1$. Thus, to ensure $\boldsymbol{\alpha} \in \Delta_n$, we only have to use a ρ_k value so that $0 \leq \alpha_j^k + \rho_k p_j^k \leq 1$. Let $\mathcal{P}_+^k = \left\{ i \;\middle|\; p_i^k > 0 \right\}$ and $\mathcal{P}_-^k = \left\{ i \;\middle|\; p_i^k < 0 \right\}$; we want

$$0 < \rho_k \leq \frac{1 - \alpha_i^k}{p_i^k} \quad \text{if} \quad i \in \mathcal{P}_+^k$$

and

$$0 < \rho_k \leq -\frac{\alpha_i^k}{p_i^k} \quad \text{if} \quad i \in \mathcal{P}_-^k.$$

Thus, setting $\rho_+ = \min\left\{ \frac{1-\alpha_i^k}{p_i^k} \;\middle|\; i \in \mathcal{P}_+^k \right\}$ and $\rho_- = \min\left\{ -\frac{\alpha_i^k}{p_i^k} \;\middle|\; i \in \mathcal{P}_-^k \right\}$, we just take $\rho_k = \min\{\tilde{\rho}_k, \rho_+, \rho_-\}$. In summary, we have derived a conjugate version of the MDM algorithm that

1. only needs $2 \times n$ extra float products per iteration and,
2. reduces the projection step to a simple clipping of the ρ coefficient.

We will also numerically compare standard and conjugate MDM in Section 6.3.1.

6.2.2 Conjugate SMO

We have already mentioned that the MDM algorithm is just a special case of SMO (López and Dorronsoro, 2015). In fact, if we take at look at the dual problem of the C-SVC,

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} - \boldsymbol{\alpha}^\top \mathbf{1} \right\} \quad \text{s.t.} \quad \boldsymbol{\alpha}^\top \mathbf{y} = 0,\ 0 \leq \alpha_i \leq C,\ i = 1, \ldots, n \tag{6.2.13}$$

and the Minimum Norm Problem (6.2.8) they both look very similar. Thus, it seems sensible to derive a conjugate version of the SMO algorithm, much like the conjugate version of the MDM algorithm from the previous section.

Recall that SMO updates are of the form (Eq. (4.3.13))

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \mathbf{d}^k$$

where $\mathbf{d}^k = y_i\mathbf{e}_i - y_j\mathbf{e}_j$ and (i, j) are the indexes selected by the procedure described in Section 4.3.3. Thus, we can replace the descent direction $\mathbf{d}$ by a conjugate direction $\mathbf{p}$,

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \mathbf{p}^k \tag{6.2.14}$$

where

$$\mathbf{p}^k = \mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}. \tag{6.2.15}$$

Note that this conjugate direction is equivalent to adding a momentum term $\mathbf{m}^k = \boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}$ that takes into account the previous two iterates, as we have already shown in Section 6.2.

As before, once this conjugate direction $\mathbf{p}^k$ has been chosen, we find the unconstrained $\tilde{\rho}_k$ factor by minimizing f along the line of $\mathbf{p}^k$. Let $\mathbf{g}^k = \nabla f(\boldsymbol{\alpha}^k) = \mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{1}$ be the gradient of the SVC objective function at $\boldsymbol{\alpha}^k$; then we solve

$$\begin{aligned}\nabla f(\boldsymbol{\alpha}^k + \tilde{\rho}_k \mathbf{p}^k) &= \tilde{\rho}_k \mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k + \mathbf{p}^k \mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{p}^k \cdot \mathbf{1} \\ &= \tilde{\rho}_k \mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k + \mathbf{p}^k(\mathbf{Q}\boldsymbol{\alpha}^k - \mathbf{1}) \\ &= \tilde{\rho}_k \mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k + \mathbf{g}^k \cdot \mathbf{p}^k = 0.\end{aligned}$$

Writing $\mathbf{p}^k$ in terms of $\mathbf{p}^{k-1}$ this yields

$$\tilde{\rho}_k^* = \frac{-\mathbf{g}^k \cdot \mathbf{p}^k}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k} = \frac{-\mathbf{g}^k \cdot (\mathbf{d}^k + \gamma_k \cdot \mathbf{p}^{k-1})}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k} = \frac{-\mathbf{g}^k \cdot \mathbf{d}^k - \gamma_k \mathbf{g}^k \cdot \mathbf{p}^{k-1}}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k}. \tag{6.2.16}$$

If the previous line minimization along $\mathbf{p}^{k-1}$ has been unclipped, i.e. we have $\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^{k-1} + \rho_{k-1}\mathbf{p}^{k-1}$, the following orthogonality condition must hold

$$\mathbf{g}^k \cdot \mathbf{p}^{k-1} = 0. \tag{6.2.17}$$

We will call Eq. (6.2.17) the **first orthogonality condition**. As a consequence,

$$\mathbf{g}^k \cdot \mathbf{p}^k = \mathbf{g}^k \cdot \mathbf{d}^k + \gamma_k \mathbf{g}^k \cdot \mathbf{p}^{k-1} = \mathbf{g}^k \cdot \mathbf{d}^k < 0,$$

i.e. $\mathbf{p}^k$ is also a descent direction. In addition Eq. (6.2.16) simplifies to

$$\tilde{\rho}_k^* = \frac{-\mathbf{g}^k \cdot \mathbf{d}^k}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k}, \tag{6.2.18}$$

and is easy to see that the unconstrained gain in f is now

$$f(\boldsymbol{\alpha}^k) - f(\boldsymbol{\alpha}^{k+1}) = \frac{1}{2}\frac{(\mathbf{g}^k \cdot \mathbf{d}^k)^2}{\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k}, \tag{6.2.19}$$

where we have used our previous choice of the unconstrained $\tilde{\rho}_k^*$. Now we can approximately maximize this gain by choosing γ_k to minimize the denominator $\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k$. Writing it as a function of γ, we have

$$\phi(\gamma_k) = (\mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}) \cdot \mathbf{Q}(\mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}),$$

and it follows that

$$\phi'(\gamma_k) = 2(\mathbf{d}^k \cdot \mathbf{Q}\mathbf{p}^{k-1} + \gamma_k \mathbf{p}^{k-1} \cdot \mathbf{Q}\mathbf{p}^{k-1}) = 0.$$

Finally, this implies

$$\gamma_k^* = -\frac{\mathbf{d}^k \cdot \mathbf{Q}\mathbf{p}^{k-1}}{\mathbf{p}^{k-1} \cdot \mathbf{Q}\mathbf{p}^{k-1}}. \tag{6.2.20}$$

Now it is easy to see that this choice results in

$$\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^{k-1} = 0. \tag{6.2.21}$$

We will call Eq. (6.2.21) the **second orthogonality condition**. We can summarize now our conjugate SMO updates. If the iteration ending in $\boldsymbol{\alpha}^k$ along $\mathbf{p}^{k-1}$ has not been clipped, we

1. compute γ_k and $\mathbf{p}^k$ using Eqs. (6.2.15) and (6.2.20),
2. compute $\tilde{\rho}_k$ and $\boldsymbol{\alpha}^{k+1}$ using Eqs. (6.2.14) and (6.2.18) and, finally,
3. check whether $\boldsymbol{\alpha}^{k+1}$ satisfies the box constraints

$$0 \leq \boldsymbol{\alpha}^{k+1} \leq C$$

and, if not, clip $\tilde{\rho}_k$ accordingly to get ρ_k and compute again $\boldsymbol{\alpha}^{k+1}$.

Note that if the preceding $\mathbf{p}^{k-1}$ verifies $\sum y_i \mathbf{p}_i^{k-1} = 0$, then

$$\sum y_i \mathbf{p}_i^k = \sum y_i (\mathbf{d}^k + \gamma_k \mathbf{p}_i^{k-1}) = 0$$

and the new $\boldsymbol{\alpha}^k$ verifies the linear constraint, i.e.,

$$\sum y_i \alpha_i^{k+1} = \sum y_i \alpha_i^k + \rho_k \sum y_i p_i^k = 0.$$

On the other hand, if we have had to apply clipping to arrive at $\boldsymbol{\alpha}^{k+1}$, i.e., we have hit the boundary of the box region, we will simply reset $\mathbf{p}^k$ to 0 after the update, as keeping the current conjugate direction may lead to further boundary hits. We will then have $\mathbf{p}^{k+1} = \mathbf{d}^{k+1}$ at the new iteration, which becomes then a standard SMO update.

At first sight, the possible advantages of working with the conjugate directions may be offset by the higher cost that deriving them imposes on the standard SMO iterations, particularly that of computing the values $\mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k$ required in the different denominators. However, working with the SMO descent directions $\mathbf{d} = \mathbf{y}_i \mathbf{e}_i - \mathbf{y}_j \mathbf{e}_j$ greatly simplifies these computations. For this, we will keep an auxiliary vector $\mathbf{q} = \mathbf{Q}\mathbf{p}$ and constant $\delta = \mathbf{p} \cdot \mathbf{Q}\mathbf{p}$ that we will update at each iteration and use to simplify the computation of the other elements as follows:

$$\begin{aligned} \gamma_k &= -\frac{\mathbf{d}^k \cdot \mathbf{Q}\mathbf{p}^{k-1}}{\mathbf{p}^{k-1} \cdot \mathbf{Q}\mathbf{p}^{k-1}} = -\frac{\mathbf{d}^k \cdot \mathbf{q}^{k-1}}{\delta^{k-1}} \\ &= \frac{\mathbf{y}_j \mathbf{q}_j^{k-1} - \mathbf{y}_i \mathbf{q}_i^{k-1}}{\delta^{k-1}}, \end{aligned} \tag{6.2.22}$$

$$\begin{aligned} \mathbf{q}^k &= \mathbf{Q}(\mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}) \\ &= y_i \mathbf{Q}_i - y_j \mathbf{Q}_j + \gamma_k \mathbf{q}^{k-1}, \end{aligned} \tag{6.2.23}$$

$$\begin{aligned} \delta_k &= \mathbf{p}^k \cdot \mathbf{Q}\mathbf{p}^k = \mathbf{d}^k \cdot \mathbf{Q}\mathbf{p}^k = \mathbf{d}^k \cdot \mathbf{q}^k \\ &= y_i \mathbf{q}_i^k - y_j \mathbf{q}_j^k, \end{aligned} \tag{6.2.24}$$

$$\tilde{\rho}_k = \frac{-\mathbf{g}^k \cdot \mathbf{d}^k}{\delta_k} = \frac{\mathbf{y}_j \mathbf{g}_j^k - \mathbf{y}_i \mathbf{g}_i^k}{\delta_k}, \tag{6.2.25}$$

where $\mathbf{Q}_j$ denotes $\mathbf{Q}$'s jth column. Note that using Eq. (6.2.23) the gradient can be efficiently updated as

$$\mathbf{g}^{k+1} = \nabla f(\boldsymbol{\alpha}^k + \rho_k \mathbf{p}^k) = \mathbf{Q}\boldsymbol{\alpha}^k + \rho_k \mathbf{Q}\mathbf{p}^k - \mathbf{1} = \mathbf{g}^k + \rho_k \mathbf{q}^k. \tag{6.2.26}$$

Algorithm 17: Conjugate SMO (CSMO)

Initialize: $\boldsymbol{\alpha}^0 = \mathbf{g}^0 = \mathbf{0}, \mathbf{p}^{-1} = \mathbf{q}^{-1} = \mathbf{0}$, $\delta^{-1} = 1$

1 **while** *stopping condition not met* **do**
2 Select working set (i, j) using Eqs. (4.3.21) and (4.3.22)
3 Compute the kernel matrix columns $\mathbf{Q}_i$ and $\mathbf{Q}_j$, if not previously cached
4 $\gamma_k = (\mathbf{y}_j \mathbf{q}_j^{k-1} - \mathbf{y}_i \mathbf{q}_i^{k-1})/\delta^{k-1}$
5 $\mathbf{p}^k = \mathbf{d}^k + \gamma_k \mathbf{p}^{k-1}$
6 $\mathbf{q}^k = \mathbf{y}_i \mathbf{Q}_i - \mathbf{y}_j \mathbf{Q}_j + \gamma_k \mathbf{q}^{k-1}$
7 $\delta^k \mathbf{y}_i \mathbf{q}_i^k - \mathbf{y}_j \mathbf{q}_j^k$
8 Compute $\tilde{\rho}_k$ as in (6.2.25) and clip it accordingly to get ρ_k
9 $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \mathbf{p}^k$
10 $\mathbf{g}^{k+1} = \mathbf{g}^k + \rho_k \mathbf{q}^k$
11 **if** $\tilde{\rho}_k$ *was clipped* **then**
12 $\mathbf{q}^k = \mathbf{p}^k = \mathbf{0}$
13 $\delta_k = 1$
14 **end**
15 **end**

The pseudocode for the conjugate version of SMO is shown in Algorithm 17. Regarding the clipping of $\tilde{\rho}_k$, we need a value that ensures for all j

$$0 \leq \alpha_j^k + \rho_k p_j^k \leq C.$$

As before, let us define the index sets

$$\mathcal{P}_+^k = \left\{ i \;\middle|\; p_i^k > 0 \right\} \quad \text{and} \quad \mathcal{P}_-^k = \left\{ i \;\middle|\; p_i^k < 0 \right\}.$$

Thus, to make sure that the following inequalities hold

$$0 < \rho_k \leq \frac{C - \alpha_i^k}{p_i^k} \quad \text{if } i \in \mathcal{P}_+^k,$$
$$0 < \rho_k \leq -\frac{\alpha_i^k}{p_i^k} \quad \text{if } i \in \mathcal{P}_-^k,$$

we take $\rho_k = \min\{\tilde{\rho}_k, \rho^+, \rho^-\}$, where

$$\rho^+ = \min\left\{ \frac{C - \alpha_i^k}{p_i^k} \;\middle|\; i \in \mathcal{P}_+^k \right\}$$

and

$$\rho^- = \min\left\{ -\frac{\alpha_i^k}{h_i^k} \;\middle|\; i \in \mathcal{P}_-^k \right\}.$$

We finish this section discussing the computational cost of the conjugate SMO updates. If n_p and n_q denote the number of non-zero components of $\mathbf{p}$ and $\mathbf{q}$ respectively, the cost in products of each iteration is

1. $2n$ floating point operations in Line 2 when selecting i and j;

2. n_p operations to update $\mathbf{p}$ in Line 5, to update $\boldsymbol{\alpha}$ in Line 9, and to compute a clipped ρ_k in Line 8;

3. n_q operations to update $\mathbf{q}$ in Line 6 and

4. n operations to update the gradient $\mathbf{g}$ in Line 10.

We expect $n_q \simeq n$ but n_p should coincide with the number of non-zero components in $\boldsymbol{\alpha}$; this number should be $\ll n$ and, similarly, we should have $n_p \ll n$. Thus a conjugate iteration should theoretically add a cost of

$$3n + n_q + 2n_p \simeq 3n + n_q \simeq 4n,$$

in contrast with $3n$ for a standard SMO iteration. Therefore, it should lead to a faster training if the number of SMO iterations is more than $4/3$ the number of CSMO ones. In any case, note that the cost of the iterations in which a non-cached kernel column matrix has to be computed will require a much larger $n \times d$ number of products when working with patterns in a d dimensional space or even more in a kernel setting. Thus, in the starting iterations the cost of the SMO and conjugate SMO would be dominated by the much larger cost of computing the required $\mathbf{Q}$ columns.

6.3 Numerical experiments

Since this section contains a lot of experiments, we have divided it into four subsections, organized as follows:

- **MDM**. Section 6.3.1. As a first step towards a Conjugate SMO version we have derived a Conjugate MDM algorithm, since they are closely related. In this section we will compare both standard MDM and its conjugate variant in terms of the number of iterations and running time for several regression datasets. These datasets will be converted to a Minimum Norm problem using the transformation from Chapter 5.

- **Algorithm correctness**. Section 6.3.2. We have implemented SMO, Conjugate SMO and Monotone Nesterov's Accelerated SMO in Python and compare the solutions obtained. Since they all solve the dual of the SVC, we would expect that all of them converge to the same model, up to numerical precision.

- **Iteration comparison**. Section 6.3.3. Once we know all the algorithms are correctly implemented in Python, we measure the number of iterations until convergence for each one. We would expect that our accelerated variants, CS and MNAS, perform significantly less iterations than SMO. However, less iteration do not mean faster running times, since the cost per iteration is increased. Thus, we also measure running times and check if our theoretical iteration complexities, measured as the number of floating point operations, also translate into practice.

- **Comparison versus LIBSVM**. Section 6.3.4. Given the results of the previous subsection, we conclude that the CS variant is more promising than the MNAS one. We also move away from Python and implement Conjugate SMO inside the LIBSVM framework. LIBSVM is one of the most widely used SVM libraries, which contains a state-of-the-art implementation of SMO in the C language. Finally, we compare the running times of both SMO and CS for different hyper-parameter values. We also discuss extensively the effect of the cache, which is one of the most important features of LIBSVM and crucial to any efficient SMO implementation.

6.3.1 MDM

Table 6.1: Sample sizes and input dimensions of the datasets considered.

Dataset	Size	Dim.
`prostate`	67	8
`cpusmall`	6 143	12
`housing`	378	13
`year`	46 215	90
`trajectory`	20 000	298
`ctscan`	53 500	385
`mnist_reg`	784	5 000
`ree`	5 698	15 960

In this section we are going to validate the theoretical findings and compare both the standard MDM and its conjugate version (CMDM) empirically. Both algorithms were implemented in the C language so we expect the timing results to be broadly homogeneous. Since it is difficult to find interesting geometrical problems to benchmark these algorithms, we are going to take advantage of the equivalence presented in Chapter 5 and consider instead regression problems. The datasets used for the numerical experiments are `prostate`, from Tibshirani (1994), `housing`, `year`, `ctscan` and `cpusmall` from the UCI repository, `trajectory`, from the Machine Learning Dataset Repository, `ree`, the problem of predicting wind energy production over peninsular Spain using numerical weather forecasts, and `mnist_reg`, a regression problem built from the MNIST dataset. Note that these datasets are the same as the ones in Section 5.4.2, but for convenience we give here again the number of samples and dimensionality in Table 6.1.

As mentioned, we apply first the transformation from Chapter 5 of the constrained Lasso problem with data matrix $\mathbf{X}$, target $\mathbf{y}$ and constraint ρ to a Nearest Point Problem between $\mathbf{y}/\rho$ and the convex hull spanned by the columns of $\mathbf{X}$ and $-\mathbf{X}$, originally proposed by Jaggi (2014). This can be easily transformed to an MNP instance by subtracting $\mathbf{y}/\rho$ from the columns of the new enlarged data matrix (Alaíz et al., 2015). Thus, the new data matrix can be constructed as

$$\widetilde{\mathbf{X}} = (\mathbf{X} \,|\, -\mathbf{X}) - \frac{1}{\rho}\mathbf{y}\mathbf{1}_{2d}^{\top}.$$

Since MDM's running time is very sensitive on the starting point, we will run both MDM algorithm and its conjugate variant considering all sample patterns as starting points and counting the number of iterations until a convergence threshold is achieved. For this we use the KKT conditions of the MNP that imply, at an optimal $\boldsymbol{\alpha}^*$,

$$\Delta = \Delta(\alpha) = \max_{\alpha_i > 0} \nabla f(\alpha)_i - \min_j \nabla f(\alpha)_j \leq 0.$$

Accordingly, we will stop the iterations when $\Delta \leq \epsilon$ for some fixed ϵ. Finally we compute the median of the time and number of iterations. As it can be seen in Table 6.2 the conjugate version of MDM always perform less iterations in median compared to standard MDM. However, note also that this does not always implies faster execution time, since the CMDM iteration cost is approximately twice of a MDM iteration. The number of iterations ratio that also translates into faster execution times is roughly 2.5. Therefore, in general, for ratios smaller than 2.5 the lower number of iterations of CMDM does not compensate the higher computational cost per iteration.

Table 6.2: Median of the execution times and number of iterations for all possible values of the initial point for both standard MDM and conjugate MDM (CMDM).

		Iterations			Time (ms)		
Dataset	$\log_{10}\epsilon$	MDM	CMDM	Ratio	MDM	CMDM	Ratio
`prostate`	−1	66	23	2.89	0.020	0.018	1.11
	−2	74	30	2.41	0.020	0.021	0.95
	−3	86	36	2.42	0.022	0.023	0.96
	−4	98	42	2.35	0.024	0.025	0.96
	−5	111	48	2.34	0.026	0.027	0.96
	−6	124	54	2.28	0.028	0.029	0.97
`housing`	−1	108	46	2.35	0.093	0.097	0.96
	−2	144	61	2.36	0.101	0.102	0.99
	−3	182	72	2.52	0.108	0.109	0.99
	−4	219	85	2.58	0.115	0.116	0.99
	−5	258	100	2.58	0.123	0.122	1.01
	−6	296	112	2.63	0.130	0.130	1.00
`cpusmall`	−1	31	26	1.17	0.940	1.027	0.92
	−2	38	32	1.21	0.985	1.001	0.98
	−3	43	38	1.13	0.952	1.009	0.94
	−4	48	43	1.13	0.945	1.002	0.94
	−5	54	47	1.16	0.952	1.021	0.93
	−6	60	53	1.13	0.979	1.025	0.96
`year`	−1	1 184	628	1.88	622.768	624.971	1.00
	−2	1 483	760	1.95	622.873	625.039	1.00
	−3	1 804	895	2.02	622.697	624.606	1.00
	−4	2 136	1 038	2.06	623.493	624.842	1.00
	−5	2 479	1 178	2.11	632.417	637.002	0.99
	−6	2 816	1 322	2.13	623.666	626.001	1.00
`ctscan`	−1	2 869	1 111	2.58	10 244.815	10 090.548	1.02
	−2	4 316	1 461	2.95	9 845.849	9 710.350	1.01
	−3	5 854	1 807	3.24	9 575.900	9 447.781	1.01
	−4	7 436	2 160	3.44	9 028.077	8 902.731	1.01
	−5	9 025	2 510	3.60	8 878.653	8 758.532	1.01
	−6	10 000	2 855	3.50	7 947.531	7 843.266	1.01
`trajectory`	−1	83	60	1.38	193.118	193.120	1.00
	−2	105	73	1.44	193.156	193.279	1.00
	−3	127	86	1.48	192.252	192.227	1.00
	−4	150	99	1.52	191.045	187.127	1.02
	−5	173	111	1.56	190.115	186.730	1.02
	−6	196	124	1.58	193.075	193.292	1.00
`mnist_reg`	−1	87	67	1.30	3.906	7.093	0.55
	−2	146	105	1.39	6.445	10.746	0.60
	−3	207	143	1.45	9.121	14.524	0.63
	−4	271	181	1.50	11.864	18.259	0.65
	−5	335	220	1.52	14.664	21.996	0.67
	−6	402	259	1.55	17.536	25.801	0.68
`ree`	−1	724	263	2.75	2 554.903	2 172.459	1.18
	−2	1 153	366	3.15	4 211.407	3 189.228	1.32
	−3	1 590	470	3.38	5 795.421	3 978.882	1.46
	−4	2 032	576	3.53	6 830.010	4 423.721	1.54
	−5	2 478	682	3.63	7 699.295	4 836.374	1.59
	−6	2 927	789	3.71	9 175.000	5 620.889	1.63

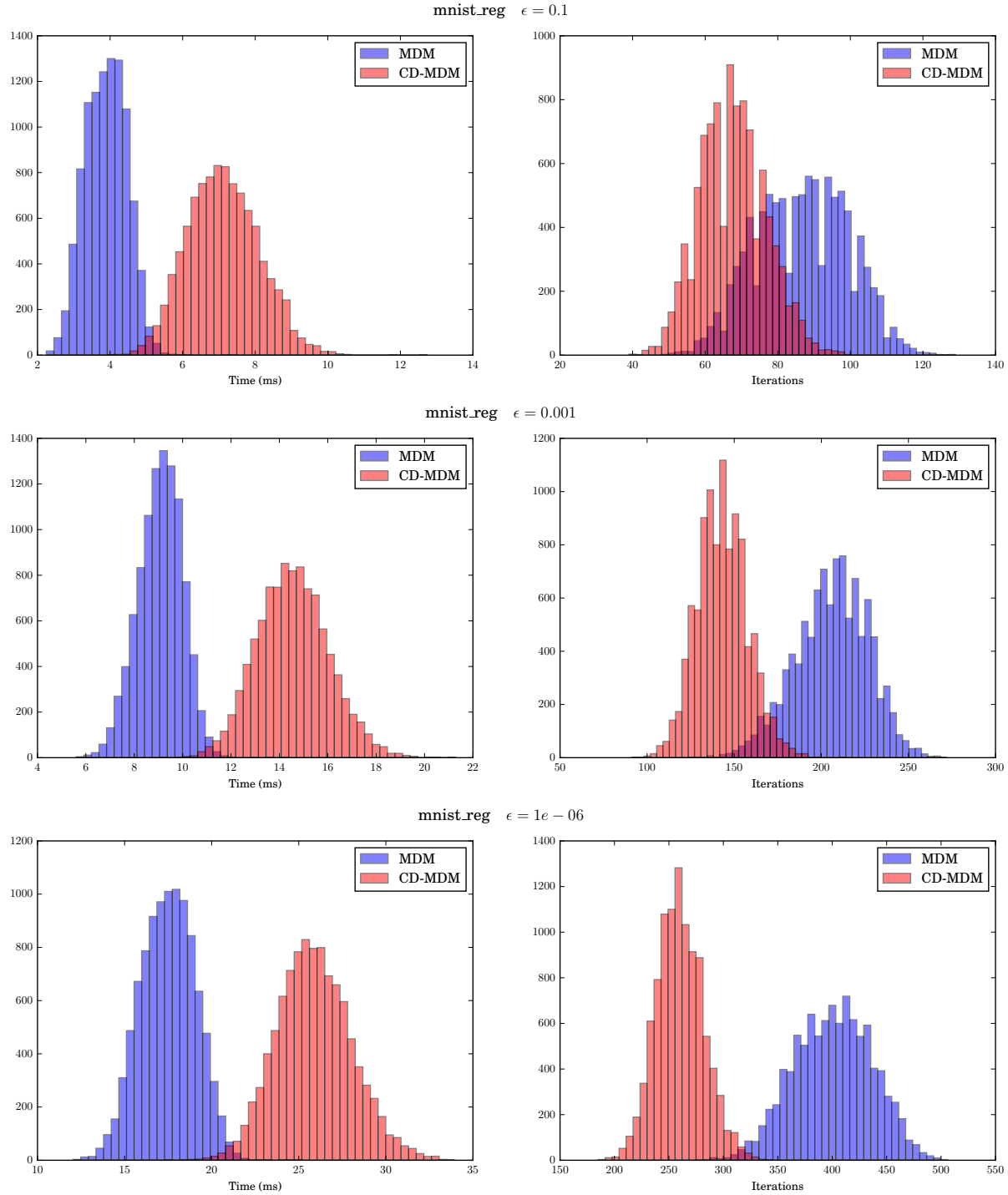

Figure 6.3: Execution times and iterations histograms for the dataset `mnist_reg` and tolerance $\epsilon = 0.1$, $\epsilon = 0.001$ and $\epsilon = 1e-06$.

Figures 6.3 and 6.4 show some example histograms of the execution time and number of iterations needed by standard and conjugate MDM for the `mnist_reg` and `ree` problems respectively and different ϵ values. Here we can see that the number of iterations distribution of the MDM algorithm are clearly to the right of the conjugate ones and in almost every case the

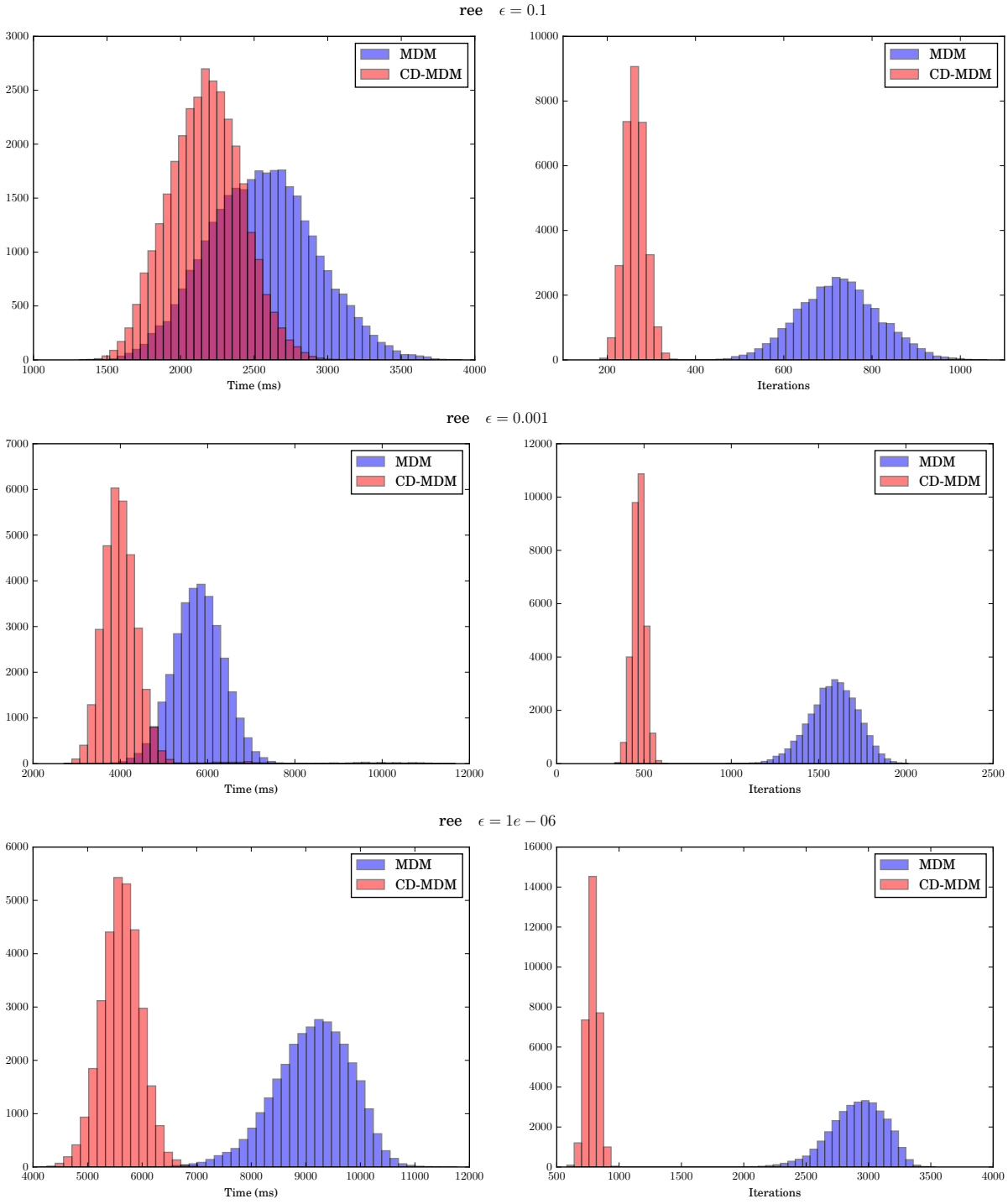

Figure 6.4: Execution times and iterations histograms for the dataset `ree` and tolerance $\epsilon = 0.1$, $\epsilon = 0.001$ and $\epsilon = 1e-06$.

ratio of the medians grows with the precision ϵ.

Finally, note that for the bigger problems `mnist_reg` and `ree` the number of points in the convex hull is quite big, around 10 000 and 30 000 respectively. Thus there are many possible starting points and each run of the algorithm is also quite expensive, so to make that computation

tractable we have pre-computed the kernel matrix $\mathbf{Q}$. In the rest of the algorithms the kernel rows are instead computed on-the-fly as needed. That is the main reason why the median execution time for a problem such as `ctscan` is larger than the bigger ones, when in reality if all of them were run without precomputing the kernel matrix both `ree` and `mnist_reg` would have been much slower. However, since we have pre-computed the matrix for both MDM and CMDM it should not affect the comparison very much or, at most, it should be detrimental for the conjugate variant since kernel computations are not taken into account in the theoretical complexity. In practice the cost of an iteration without pre-computing $\mathbf{Q}$ is dominated by the kernel computation and thus the extra cost of CMDM is barely noticeable. On the other hand, if the matrix is pre-computed no kernel operations are performed, which is beneficial for standard MDM.

In conclusion the conjugate variant of MDM achieves a substantial reduction in running time for most of the "big" problems except for `mnist_reg`, while having little to no effect in the small ones. In general the improvement grows with the inverse of the target precision ϵ, so the running time ratios are better for smaller ϵ values.

6.3.2 SMO: Algorithm correctness

Table 6.3: Dimensions, data sizes and class sizes of the datasets considered.

Dataset	d	n	n^+	n^-
heart	13	270	120	150
diabetes	8	768	500	268
australian	14	690	307	383
german	24	1 000	300	700
adult4	123	4 781	1 188	3 593
adult8	123	22 696	5 506	17 190
web7	300	24 692	740	23 952
web8	300	49 749	1 479	48 270

In this section we will put everything together and compare the behavior of second order SMO, Conjugate SMO (CS) and Monotone Nesterov's Accelerated SMO (MNAS) in 8 binary classification datasets, `heart`, `diabetes`, `australian`, `german` (in its numeric version), `adult4`, `adult8`, `web7` and `web8`, that are all available in LIBSVM's website. Table 6.3 shows for every dataset the number of samples for each class and dimension. We will work with the Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right). \tag{6.3.1}$$

Features for the first four problems are scaled to $[-1, 1]$ while the `adult` and `web` problems already have binary $(0-1)$ features.

Following the previous outline, we begin the SMO experiments by checking if the three algorithms, namely SMO, Conjugate SMO (CS) and Monotone Nesterov's Accelerated SMO (MNAS), yield the same final SVM model, in the sense that they arrive with high precision at the same value $f(\boldsymbol{\alpha}^*)$ of the SVC objective function and have very similar number of support vectors (SVs). We shall consider in these experiments two different values for the precision of the stopping criteria ϵ, 0.1, 0.001; three C values, 1, 10, 100; and two γ values, $1.0 \times 1/d$, $0.1 \times 1/d$.

For each one of those hyper-parameters sets we are going to measure the final value of the coefficients $\boldsymbol{\alpha}^*$, objective function $f(\boldsymbol{\alpha}^*)$ and number of support vectors. These values for the

Table 6.4: Final value of SMO's objective function and number of support vectors (nSV) for $\epsilon = 0.1$ and $\epsilon = 0.001$

			$\epsilon = 0.1$		$\epsilon = 0.001$	
Dataset	C	γ	$f(\boldsymbol{\alpha}^*)$	nSV	$f(\boldsymbol{\alpha}^*)$	nSV
heart	1	1	−100.824	128	−100.877	132
		0.1	−135.337	160	−135.428	163
	10	1	−660.276	113	−660.429	115
		0.1	−998.592	116	−999.108	117
	100	1	−2 526.000	109	−2 526.925	107
		0.1	−8 338.725	107	−8 340.956	106
diabetes	1	1	−413.466	446	−413.564	447
		0.1	−498.245	536	−498.448	538
	10	1	−3 724.647	404	−3 725.665	400
		0.1	−4 182.714	441	−4 183.452	442
	100	1	−34 129.822	392	−34 138.208	383
		0.1	−39 066.799	407	−39 074.251	408
australian	1	1	−201.436	238	−201.643	244
		0.1	−236.795	276	−237.271	289
	10	1	−1 710.259	238	−1 711.443	231
		0.1	−2 021.438	230	−2 025.824	226
	100	1	−12 795.859	236	−12 801.354	226
		0.1	−19 244.482	228	−19 252.758	220
german	1	1	−502.638	591	−502.770	599
		0.1	−584.171	604	−584.418	611
	10	1	−3 527.235	563	−3 528.445	561
		0.1	−5 259.519	566	−5 260.384	563
	100	1	−16 754.209	547	−16 760.502	539
		0.1	−46 325.338	541	−46 333.257	538
adult4	1	1	−1 695.480	1 863	−1 696.051	1 876
		0.1	−1 988.261	2 163	−1 989.040	2 185
	10	1	−14 034.475	1 844	−14 039.028	1 847
		0.1	−17 368.956	1 835	−17 373.817	1 835
	100	1	−91 469.480	1 910	−91 501.068	1 880
		0.1	−160 535.300	1 780	−160 568.229	1 758
adult8	1	1	−7 771.713	8 203	−7 774.383	8 242
		0.1	−8 543.447	8 898	−8 546.102	8 935
	10	1	−70 155.050	8 165	−70 180.534	8 102
		0.1	−80 498.816	8 261	−80 519.906	8 254
	100	1	−563 143.596	8 530	−563 347.241	8 394
		0.1	−766 082.352	8 164	−766 313.332	8 086
web7	1	1	−1 182.406	1 433	−1 184.225	1 595
		0.1	−1 431.004	1 481	−1 437.244	1 548
	10	1	−7 663.946	1 235	−7 672.213	1 474
		0.1	−11 944.053	1 396	−11 967.272	1 435
	100	1	−46 928.113	1 189	−46 965.997	1 424
		0.1	−85 132.550	1 120	−85 258.038	1 123
web8	1	1	−2 135.394	2 625	−2 137.950	2 834
		0.1	−2 787.795	2 962	−2 795.051	3 016
	10	1	−13 792.669	2 100	−13 804.975	2 416
		0.1	−21 729.045	2 598	−21 760.912	2 617
	100	1	−92 659.990	2 056	−92 720.179	2 335
		0.1	−155 840.796	1 950	−156 088.417	2 089

SMO algorithm, excluding the coefficients, can be found in Table 6.4. Then, we are going to compute the relative error of the final objective function and number of support vectors between SMO and CS/MNAS, defined as,

$$\text{Error}_f = \left| \frac{f(\boldsymbol{\alpha}^*_{\text{SMO}}) - f(\boldsymbol{\alpha}^*)}{f(\boldsymbol{\alpha}^*_{\text{SMO}})} \right|,$$

and

$$\text{Error}_{\text{nSV}} = \left| \frac{\text{nSV}_{\text{SMO}} - \text{nSV}}{\text{nSV}_{\text{SMO}}} \right|.$$

The previous measure can also be extended to vectors by replacing the absolute value with any norm. In our case we are going to use the ℓ_1-norm to compute the relative error of the coefficients,

$$\text{Error}_{\boldsymbol{\alpha}^*} = \frac{\|\boldsymbol{\alpha}^*_{\text{SMO}} - \boldsymbol{\alpha}^*\|_1}{\|\boldsymbol{\alpha}^*_{\text{SMO}}\|_1}.$$

Tables 6.5 and 6.6 show for $\epsilon = 0.1$ and $\epsilon = 0.001$ respectively the relative error of the final coefficients, objective function and number of support vectors obtained by each of the three algorithms. The conclusion is that the three methods obtain the same model up to numerical precision.

6.3.3 SMO: Iteration comparison

We report in Tables 6.7 and 6.8 for each dataset, C and γ values the number of iterations (columns 4 and 5) and running times (columns 7 and 8) of plain SMO and Monotone Nesterov's Accelerated SMO (MNAS), together with their respective ratios, SMO/MNAS. Thus, ratios above 1 mean that MNAS is faster than SMO while ratios below 1 mean the opposite. The first table corresponds to an $\epsilon = 0.1$ and the second table shows the results for $\epsilon = 0.001$. All the experiments were run in an Intel(R) Xeon(R) server with 16 E5-2680 2.70GHz CPUs and 128 Gb of RAM.

In general, MNAS only performs less iterations than SMO, and thus the ratio is bigger than 1, for large C values (10, 100). However, due to MNAS increased iteration complexity these iteration gains do not always translate into faster execution times. The iteration and time gains are generally better for larger C and γ and smaller ϵ.

The same results comparing SMO and Conjugate SMO are given in Tables 6.9 and 6.10, again for $\epsilon = 0.1$ and $\epsilon = 0.001$. As it can be seen in the previous tables CS always performs less iterations than SMO except for the following cases:

- $\epsilon = 0.1$; datasets `australian`, `web7`; $C = 1, 10$ and $\gamma = 0.1$.
- $\epsilon = 0.001$, dataset `web7`, $C = 1$ and $\gamma = 0.1$.

As an example we show in Fig. 6.5 the number of iterations needed by plain SMO and Conjugate SMO for different C values, $\gamma = 1$ and $\epsilon = 0.001$ (LIBSVM's defaults). Here we can see how the problem becomes more difficult as C is increased, which is beneficial to our conjugate implementation.

A similar behaviour is illustrated in Fig. 6.6, where we plot the evolution of the objective function for SMO, CS and MNAS. For the `adult4` dataset the convergence starts improving much earlier for $C = 100$, while in `web7` CS performs less iterations than SMO but not MNAS. Furthermore, these iteration gains in CS almost always correspond to running times that are

Table 6.5: Relative error with respect to SMO in the coefficients, objective value and number of support vectors for CS and MNAS ($\epsilon = 0.1$)

			$\boldsymbol{\alpha}^*$ ($\times 10^{-4}$)		$f(\boldsymbol{\alpha}^*)$ ($\times 10^{-4}$)		nSV ($\times 10^{-2}$)	
Dataset	C	γ	MNAS	CS	MNAS	CS	MNAS	CS
heart	1	1	1.169	1.429	2.710	2.255	2.344	0.000
		0.1	0.000	0.003	0.000	0.037	0.000	0.000
	10	1	1.086	1.469	0.158	0.938	0.885	0.000
		0.1	0.732	0.715	2.039	2.580	0.000	0.862
	100	1	0.878	0.997	1.992	2.929	1.835	1.835
		0.1	0.650	0.969	1.255	1.151	0.935	0.935
diabetes	1	1	0.188	0.229	0.741	1.615	0.673	0.000
		0.1	0.000	0.146	0.000	2.221	0.000	0.000
	10	1	0.199	0.309	0.462	1.902	0.248	0.990
		0.1	0.124	0.077	0.398	0.259	0.000	0.680
	100	1	0.273	0.273	1.101	1.158	0.765	0.510
		0.1	0.150	0.158	0.569	0.861	0.491	0.983
australian	1	1	1.052	1.290	2.539	5.713	1.681	0.840
		0.1	0.257	2.368	2.863	4.867	0.000	0.000
	10	1	0.928	1.021	1.468	4.417	0.420	2.941
		0.1	1.227	2.793	10.283	7.142	0.870	0.435
	100	1	0.500	0.571	2.241	3.165	0.848	2.119
		0.1	0.761	0.792	0.173	1.052	0.877	1.316
german	1	1	0.203	0.235	0.775	0.687	0.000	0.846
		0.1	0.057	0.198	0.048	2.463	0.331	0.331
	10	1	0.252	0.260	1.499	2.477	0.533	0.355
		0.1	0.165	0.167	0.371	1.099	0.000	0.530
	100	1	0.161	0.229	0.977	3.001	0.000	1.097
		0.1	0.148	0.139	0.459	0.862	0.185	0.185
adult4	1	1	0.025	0.056	0.983	2.351	0.268	0.215
		0.1	0.000	0.039	0.008	0.213	0.000	0.000
	10	1	0.061	0.057	0.813	2.487	0.651	0.325
		0.1	0.027	0.035	0.845	1.436	0.000	0.000
	100	1	0.031	0.042	1.358	2.760	0.157	1.047
		0.1	0.045	0.034	0.178	1.367	0.112	0.618
adult8	1	1	0.006	0.011	0.511	1.494	0.159	0.012
		0.1	0.000	0.005	0.000	0.237	0.000	0.000
	10	1	0.014	0.014	0.632	2.402	0.184	0.318
		0.1	0.003	0.007	0.377	0.355	0.121	0.012
	100	1	0.010	0.012	0.878	2.542	0.117	1.079
		0.1	0.010	0.010	0.990	1.843	0.196	0.233
web7	1	1	0.016	0.022	1.085	3.280	0.279	0.977
		0.1	0.011	0.024	3.242	3.093	0.135	0.000
	10	1	0.020	0.023	1.677	3.807	0.081	0.405
		0.1	0.022	0.027	5.037	6.157	0.716	0.215
	100	1	0.012	0.018	2.274	4.481	0.421	0.084
		0.1	0.019	0.022	2.475	5.786	0.893	1.607
web8	1	1	0.009	0.009	5.110	0.031	0.038	0.229
		0.1	0.004	0.010	4.119	6.017	0.034	0.135
	10	1	0.009	0.011	0.393	2.087	0.000	0.429
		0.1	0.010	0.010	3.448	1.561	0.039	0.308
	100	1	0.006	0.008	1.235	2.538	0.973	0.632
		0.1	0.010	0.013	0.836	3.349	0.051	0.769

Table 6.6: Relative error with respect to SMO in the coefficients, objective value and number of support vectors for CS and MNAS ($\epsilon = 0.001$)

			$\boldsymbol{\alpha}^*$ ($\times 10^{-6}$)		$f(\boldsymbol{\alpha}^*)$ ($\times 10^{-6}$)		nSV ($\times 10^{-3}$)	
Dataset	C	γ	MNAS	CS	MNAS	CS	MNAS	CS
heart	1	1	0.895	1.003	0.022	0.031	0.000	0.000
		0.1	2.465	1.269	0.063	0.041	6.135	0.000
	10	1	1.515	1.895	0.008	0.029	8.696	0.000
		0.1	0.725	1.458	0.036	0.041	0.000	0.000
	100	1	1.358	1.644	0.051	0.072	0.000	0.000
		0.1	2.977	2.532	0.036	0.061	0.000	0.000
diabetes	1	1	0.341	0.331	0.038	0.029	0.000	0.000
		0.1	0.072	0.000	0.012	0.000	0.000	0.000
	10	1	0.368	0.316	0.021	0.020	0.000	0.000
		0.1	0.281	0.122	0.020	0.011	0.000	6.787
	100	1	0.386	0.391	0.025	0.038	0.000	2.611
		0.1	0.117	0.219	0.010	0.028	0.000	2.451
australian	1	1	3.225	3.516	0.148	0.214	4.098	4.098
		0.1	12.081	17.888	0.190	0.814	3.460	6.920
	10	1	1.120	1.461	0.052	0.079	0.000	0.000
		0.1	5.155	5.452	0.293	0.474	8.850	0.000
	100	1	0.746	0.830	0.002	0.026	8.850	4.425
		0.1	5.968	3.446	0.318	0.347	18.182	4.545
german	1	1	0.481	0.430	0.014	0.036	0.000	0.000
		0.1	0.281	0.880	0.007	0.019	1.637	1.637
	10	1	0.294	0.338	0.014	0.037	0.000	0.000
		0.1	0.368	0.360	0.011	0.035	5.329	0.000
	100	1	1.130	1.173	0.108	0.132	1.855	0.000
		0.1	0.404	0.339	0.032	0.047	0.000	1.859
adult4	1	1	0.066	0.110	0.021	0.043	2.132	1.599
		0.1	0.116	0.150	0.006	0.025	0.458	0.458
	10	1	0.306	0.599	0.010	0.036	2.166	1.083
		0.1	0.041	0.076	0.015	0.049	0.545	0.000
	100	1	0.206	0.282	0.016	0.040	1.596	1.064
		0.1	0.097	0.143	0.019	0.036	5.119	1.706
adult8	1	1	0.044	0.131	0.029	0.054	0.728	1.699
		0.1	0.026	0.068	0.041	0.085	0.112	0.336
	10	1	0.216	0.236	0.022	0.049	2.469	2.222
		0.1	0.044	0.072	0.013	0.036	0.485	0.363
	100	1	0.221	0.261	0.018	0.048	2.264	0.715
		0.1	0.153	0.121	0.019	0.063	1.360	1.237
web7	1	1	0.205	0.349	0.070	0.197	0.000	2.508
		0.1	0.265	0.354	0.583	0.706	0.646	0.646
	10	1	0.424	0.452	0.040	0.144	2.035	4.749
		0.1	0.372	0.316	0.237	0.059	2.787	2.091
	100	1	0.171	0.341	0.033	0.070	1.404	0.000
		0.1	0.214	0.245	0.122	0.115	6.233	1.781
web8	1	1	0.109	0.133	0.208	0.255	1.764	2.823
		0.1	0.096	0.103	0.070	0.284	0.995	0.000
	10	1	0.239	0.205	0.086	0.161	2.897	8.278
		0.1	0.102	0.120	0.072	0.139	1.146	1.528
	100	1	0.128	0.203	0.028	0.086	3.854	4.711
		0.1	0.219	0.271	0.131	0.238	10.053	36.381

Table 6.7: Number of iterations and time for SMO and Monotone Nesterov Accelerated SMO (MNAS), together with their ratios SMO/MNAS ($\epsilon = 0.1$)

			Iterations			Time ($\times 10s$)		
Dataset	C	γ	SMO	MNAS	Ratio	SMO	MNAS	Ratio
heart	1	1	91	92	0.99	0.223	0.222	1.00
		0.1	86	91	0.95	0.224	0.224	1.00
	10	1	257	203	1.27	0.229	0.228	1.01
		0.1	108	97	1.11	0.221	0.220	1.00
	100	1	627	581	1.08	0.249	0.250	0.99
		0.1	336	287	1.17	0.229	0.230	1.00
diabetes	1	1	256	274	0.93	0.315	0.318	0.99
		0.1	277	284	0.98	0.320	0.323	0.99
	10	1	407	362	1.12	0.322	0.322	1.00
		0.1	264	273	0.97	0.314	0.316	1.00
	100	1	2 196	1 343	1.64	0.479	0.426	1.12
		0.1	476	393	1.21	0.325	0.319	1.02
australian	1	1	166	178	0.93	0.289	0.292	0.99
		0.1	142	143	0.99	0.288	0.289	0.99
	10	1	437	364	1.20	0.315	0.312	1.01
		0.1	158	159	0.99	0.286	0.289	0.99
	100	1	1 532	1 331	1.15	0.435	0.427	1.02
		0.1	553	414	1.34	0.324	0.319	1.02
german	1	1	402	406	0.99	0.408	0.412	0.99
		0.1	365	369	0.99	0.406	0.410	0.99
	10	1	1 259	1 061	1.19	0.576	0.549	1.05
		0.1	454	430	1.06	0.404	0.406	1.00
	100	1	4 596	3 151	1.46	1.405	1.093	1.29
		0.1	1 729	1 106	1.56	0.619	0.511	1.21
adult4	1	1	1 237	1 224	1.01	2.996	2.982	1.00
		0.1	1 209	1 219	0.99	3.227	3.290	0.98
	10	1	3 537	2 717	1.30	4.711	4.377	1.08
		0.1	1 231	1 202	1.02	3.771	4.119	0.92
	100	1	16 732	9 864	1.70	21.029	15.384	1.37
		0.1	4 248	2 967	1.43	7.685	7.005	1.10
adult8	1	1	5 407	5 250	1.03	55.951	75.598	0.74
		0.1	5 147	5 166	1.00	68.013	64.519	1.05
	10	1	15 542	11 203	1.39	117.635	87.811	1.34
		0.1	5 578	5 258	1.06	64.063	66.485	0.96
	100	1	86 525	48 306	1.79	474.306	383.019	1.24
		0.1	17 450	11 912	1.46	90.786	87.884	1.03
web7	1	1	1 090	1 085	1.00	20.873	24.657	0.85
		0.1	920	923	1.00	24.229	33.725	0.72
	10	1	2 195	2 312	0.95	32.190	34.375	0.94
		0.1	1 120	1 103	1.02	21.771	28.157	0.77
	100	1	5 651	6 750	0.84	41.168	61.798	0.67
		0.1	2 864	2 906	0.99	30.042	38.724	0.78
web8	1	1	2 029	2 014	1.01	82.362	100.968	0.82
		0.1	2 004	1 975	1.01	93.546	104.719	0.89
	10	1	3 666	3 762	0.97	84.536	102.106	0.83
		0.1	2 209	2 098	1.05	71.077	92.176	0.77
	100	1	9 543	9 833	0.97	121.137	193.014	0.63
		0.1	4 921	4 813	1.02	103.120	155.713	0.66

Table 6.8: Number of iterations and time for SMO and Monotone Nesterov Accelerated SMO (MNAS), together with their ratios SMO/MNAS ($\epsilon = 0.001$)

			Iterations			Time ($\times 10s$)		
Dataset	C	γ	SMO	MNAS	Ratio	SMO	MNAS	Ratio
heart	1	1	140	158	0.89	0.219	0.223	0.99
		0.1	127	114	1.11	0.226	0.259	0.87
	10	1	578	435	1.33	0.295	0.263	1.12
		0.1	191	144	1.33	0.234	0.227	1.03
	100	1	1 691	1 239	1.36	0.334	0.341	0.98
		0.1	1 016	525	1.94	0.287	0.268	1.07
diabetes	1	1	317	323	0.98	0.316	0.322	0.98
		0.1	283	294	0.96	0.362	0.381	0.95
	10	1	893	601	1.49	0.475	0.453	1.05
		0.1	334	308	1.08	0.432	0.398	1.09
	100	1	6 379	3 877	1.65	0.922	0.749	1.23
		0.1	990	586	1.69	0.392	0.381	1.03
australian	1	1	404	401	1.01	0.303	0.309	0.98
		0.1	594	383	1.55	0.375	0.412	0.91
	10	1	1 403	795	1.76	0.495	0.432	1.15
		0.1	1 129	613	1.84	0.500	0.400	1.25
	100	1	4 586	3 668	1.25	0.928	0.755	1.23
		0.1	3 090	1 419	2.18	0.456	0.373	1.22
german	1	1	672	672	1.00	0.442	0.510	0.87
		0.1	444	423	1.05	0.536	0.547	0.98
	10	1	3 189	2 046	1.56	1.265	0.983	1.29
		0.1	849	609	1.39	0.510	0.436	1.17
	100	1	12 560	6 868	1.83	3.165	2.384	1.33
		0.1	4 565	2 110	2.16	1.123	0.904	1.24
adult4	1	1	1 898	1 681	1.13	3.425	4.015	0.85
		0.1	1 404	1 349	1.04	5.342	4.269	1.25
	10	1	8 965	5 193	1.73	7.916	5.273	1.50
		0.1	2 102	1 669	1.26	3.415	3.641	0.94
	100	1	47 162	22 049	2.14	47.067	33.673	1.40
		0.1	10 099	5 732	1.76	11.725	9.777	1.20
adult8	1	1	8 316	7 200	1.16	63.417	78.300	0.81
		0.1	5 520	5 458	1.01	69.731	74.364	0.94
	10	1	43 727	24 654	1.77	151.736	163.908	0.93
		0.1	8 076	6 945	1.16	58.124	78.874	0.74
	100	1	284 637	137 391	2.07	917.887	654.244	1.40
		0.1	52 340	26 364	1.99	155.788	98.025	1.59
web7	1	1	2 458	2 411	1.02	29.930	32.162	0.93
		0.1	1 544	1 586	0.97	28.078	37.567	0.75
	10	1	8 047	8 333	0.97	51.505	79.330	0.65
		0.1	2 396	2 209	1.08	24.005	29.680	0.81
	100	1	23 507	24 361	0.96	99.852	176.081	0.57
		0.1	10 381	9 083	1.14	41.799	47.612	0.88
web8	1	1	3 951	4 014	0.98	110.758	126.512	0.88
		0.1	2 685	2 640	1.02	100.198	117.453	0.85
	10	1	14 127	15 889	0.89	179.241	226.285	0.79
		0.1	4 821	4 342	1.11	82.783	94.568	0.88
	100	1	41 112	43 336	0.95	296.160	469.777	0.63
		0.1	25 492	27 172	0.94	212.816	392.636	0.54

Table 6.9: Number of iterations and time for SMO and Conjugate SMO (CS), together with their ratios SMO/CS ($\epsilon = 0.1$)

			Iterations			Time ($\times 10s$)		
Dataset	C	γ	SMO	CS	Ratio	SMO	CS	Ratio
heart	1	1	91	90	1.01	0.238	0.236	1.01
		0.1	86	86	1.00	0.239	0.240	1.00
	10	1	257	186	1.38	0.244	0.241	1.02
		0.1	108	93	1.16	0.236	0.235	1.01
	100	1	627	502	1.25	0.265	0.259	1.02
		0.1	336	246	1.37	0.243	0.238	1.02
diabetes	1	1	256	252	1.02	0.340	0.339	1.00
		0.1	277	277	1.00	0.347	0.350	0.99
	10	1	407	366	1.11	0.355	0.349	1.02
		0.1	264	265	1.00	0.338	0.338	1.00
	100	1	2 196	1 387	1.58	0.597	0.501	1.19
		0.1	476	404	1.18	0.360	0.355	1.01
australian	1	1	166	154	1.08	0.302	0.306	0.99
		0.1	142	144	0.99	0.299	0.301	0.99
	10	1	437	352	1.24	0.328	0.321	1.02
		0.1	158	164	0.96	0.297	0.299	0.99
	100	1	1 532	1 137	1.35	0.448	0.412	1.09
		0.1	553	434	1.27	0.338	0.327	1.03
german	1	1	402	392	1.03	0.433	0.434	1.00
		0.1	365	365	1.00	0.431	0.431	1.00
	10	1	1 259	1 100	1.14	0.600	0.581	1.03
		0.1	454	414	1.10	0.430	0.433	1.00
	100	1	4 596	3 221	1.43	1.436	1.128	1.27
		0.1	1 729	1 276	1.36	0.649	0.586	1.11
adult4	1	1	1 237	1 221	1.01	30.090	30.044	1.00
		0.1	1 209	1 209	1.00	32.143	32.374	0.99
	10	1	3 537	3 095	1.14	47.185	44.524	1.06
		0.1	1 231	1 237	1.00	29.553	29.659	1.00
	100	1	16 732	12 087	1.38	169.069	133.088	1.27
		0.1	4 248	3 383	1.26	50.759	46.420	1.09
adult8	1	1	5 407	5 369	1.01	509.799	531.405	0.96
		0.1	5 147	5 147	1.00	531.060	534.185	0.99
	10	1	15 542	12 983	1.20	1 103.060	1 017.529	1.08
		0.1	5 578	5 497	1.01	665.697	660.533	1.01
	100	1	86 525	62 217	1.39	4 499.463	3 769.622	1.19
		0.1	17 450	14 511	1.20	1 420.154	978.171	1.45
web7	1	1	1 090	1 078	1.01	214.144	203.638	1.05
		0.1	920	933	0.99	199.133	201.360	0.99
	10	1	2 195	2 101	1.04	235.153	221.263	1.06
		0.1	1 120	1 151	0.97	200.844	201.301	1.00
	100	1	5 651	5 053	1.12	364.014	406.679	0.90
		0.1	2 864	2 668	1.07	355.006	344.876	1.03
web8	1	1	2 029	1 971	1.03	700.775	703.616	1.00
		0.1	2 004	1 979	1.01	805.382	1 026.556	0.78
	10	1	3 666	3 393	1.08	918.371	887.179	1.04
		0.1	2 209	2 114	1.04	877.262	792.532	1.11
	100	1	9 543	8 206	1.16	1 485.928	1 360.474	1.09
		0.1	4 921	4 527	1.09	772.865	979.809	0.79

Table 6.10: Number of iterations and time for SMO and Conjugate SMO (CS), together with their ratios SMO/CS ($\epsilon = 0.001$)

			Iterations			Time ($\times 10s$)		
Dataset	C	γ	SMO	CS	Ratio	SMO	CS	Ratio
heart	1	1	140	129	1.09	0.234	0.236	0.99
		0.1	127	110	1.15	0.239	0.237	1.01
	10	1	578	388	1.49	0.251	0.246	1.02
		0.1	191	151	1.26	0.236	0.236	1.00
	100	1	1 691	1 119	1.51	0.314	0.290	1.08
		0.1	1 016	543	1.87	0.259	0.251	1.03
diabetes	1	1	317	297	1.07	0.345	0.341	1.01
		0.1	283	283	1.00	0.344	0.348	0.99
	10	1	893	642	1.39	0.398	0.375	1.06
		0.1	334	297	1.12	0.338	0.334	1.01
	100	1	6 379	2 984	2.14	1.073	0.708	1.51
		0.1	990	587	1.69	0.452	0.410	1.10
australian	1	1	404	368	1.10	0.310	0.312	0.99
		0.1	594	399	1.49	0.332	0.322	1.03
	10	1	1 403	885	1.59	0.387	0.355	1.09
		0.1	1 129	645	1.75	0.354	0.327	1.08
	100	1	4 586	3 276	1.40	0.683	0.583	1.17
		0.1	3 090	1 367	2.26	0.460	0.374	1.23
german	1	1	672	601	1.12	0.442	0.436	1.01
		0.1	444	412	1.08	0.424	0.423	1.00
	10	1	3 189	2 414	1.32	0.892	0.793	1.12
		0.1	849	654	1.30	0.489	0.476	1.03
	100	1	12 560	8 584	1.46	2.834	2.006	1.41
		0.1	4 565	2 454	1.86	0.829	0.654	1.27
adult4	1	1	1 898	1 694	1.12	3.445	3.352	1.03
		0.1	1 404	1 357	1.03	3.475	3.476	1.00
	10	1	8 965	6 743	1.33	7.469	6.276	1.19
		0.1	2 102	1 819	1.16	3.259	3.207	1.02
	100	1	47 162	31 328	1.51	35.349	24.621	1.44
		0.1	10 099	6 721	1.50	7.034	5.459	1.29
adult8	1	1	8 316	7 516	1.11	56.870	53.942	1.05
		0.1	5 520	5 486	1.01	60.916	51.440	1.18
	10	1	43 727	30 965	1.41	128.214	106.017	1.21
		0.1	8 076	7 086	1.14	51.818	50.165	1.03
	100	1	284 637	185 062	1.54	800.948	540.952	1.48
		0.1	52 340	32 686	1.60	133.235	100.693	1.32
web7	1	1	2 458	2 241	1.10	35.887	30.494	1.18
		0.1	1 544	1 607	0.96	31.853	26.562	1.20
	10	1	8 047	6 912	1.16	55.316	48.842	1.13
		0.1	2 396	2 366	1.01	34.098	27.217	1.25
	100	1	23 507	18 161	1.29	111.429	93.021	1.20
		0.1	10 381	8 508	1.22	54.440	54.864	0.99
web8	1	1	3 951	3 859	1.02	114.615	113.130	1.01
		0.1	2 685	2 687	1.00	133.199	108.624	1.23
	10	1	14 127	11 939	1.18	152.126	127.094	1.20
		0.1	4 821	4 408	1.09	77.032	76.533	1.01
	100	1	41 112	30 286	1.36	294.980	237.012	1.24
		0.1	25 492	21 416	1.19	193.592	167.208	1.16

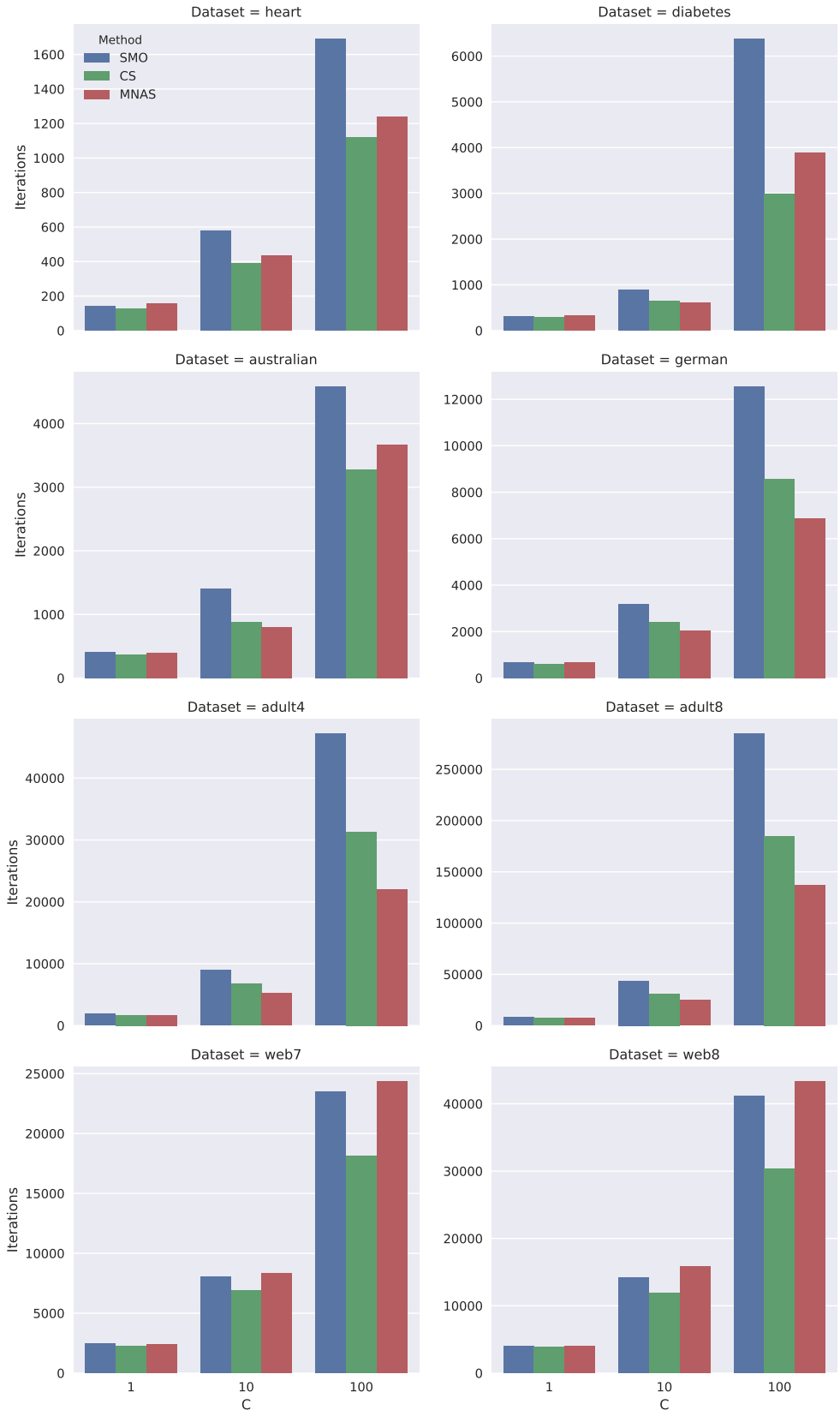

Figure 6.5: Evolution of the number of iterations with respect to C for the eight datasets

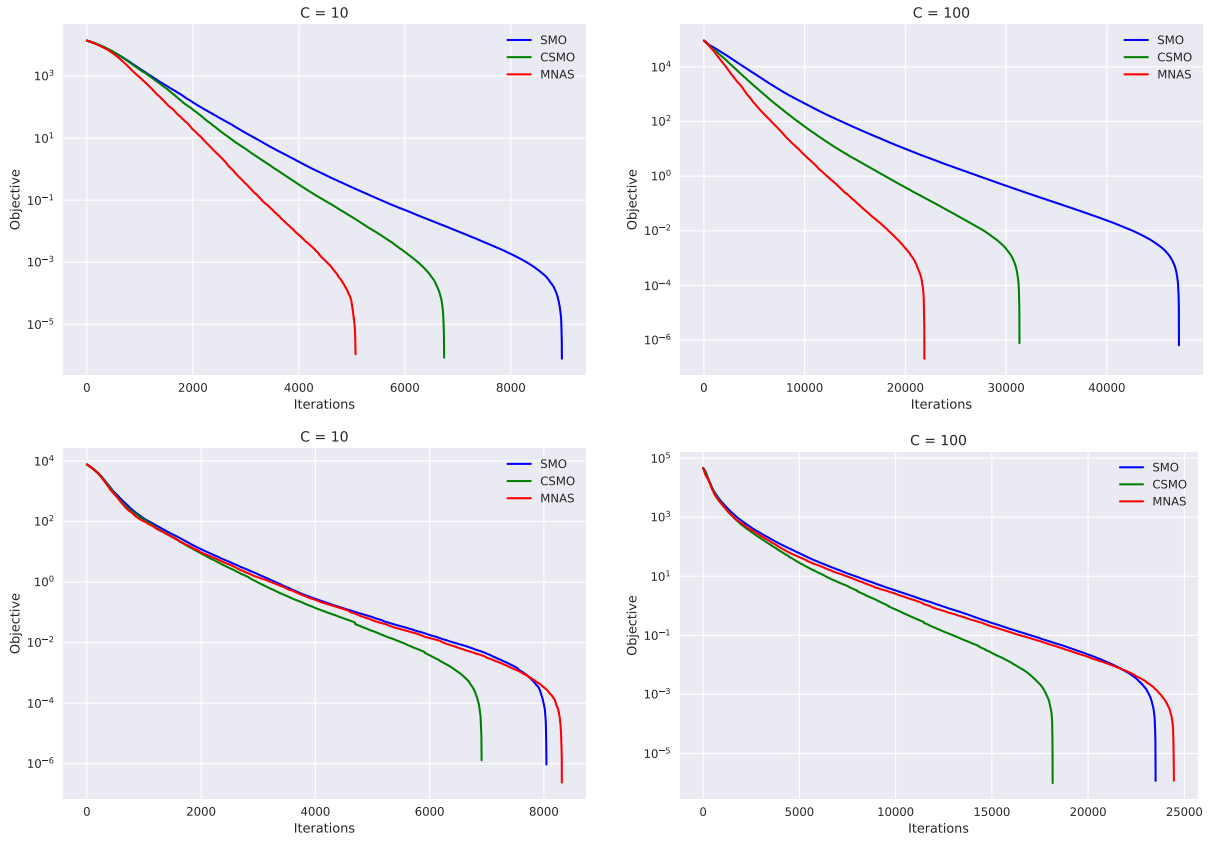

Figure 6.6: Evolution of the objective function with $\epsilon = 0.001$ for the `adult4` (top) and the `web7` datasets

at least equal or better than SMO's. One notable exception is the dataset `web7`, where CS performs much worse in running time for $\epsilon = 0.1$, $C = 1, 100$ and $\gamma = 0.1$. In general, similarly to the MNAS case, both the iterations and times ratios are better for $\epsilon = 0.001$ and the larger the C and γ.

Theoretically, these gains in iterations would correspond to faster execution times if the ratio is bigger than 4/3 in CS and 2 in MNAS, but in practice we observe it to happen for slightly smaller ratios. In particular, we have found that the average iteration ratio for which the time ratio is bigger than 1 in MNAS is 1.31 for $\epsilon = 0.1$ and 1.68 for $\epsilon = 0.001$. On the other hand CS average iteration ratios, 1.20 and 1.35 respectively, are closer to the theoretical ratio, $4/3 = 1.33$, and are better than their MNAS counterpart.

As we already mentioned, the previous results have been obtained using our own Python implementation of SMO, Monotone Nesterov's Accelerated SMO and Conjugate SMO. This implementation has an unlimited cache and thus the SMO algorithm has a slight advantage since its iterations are cheaper. In practice, when the kernel matrix is too big to fit in the cache, reducing the number of iterations also reduces the probability of a cache miss. Thus, since kernel computations take most of the iteration time, the extra computations performed by CS and MNAS should be negligible when compared with the time spent in the kernel operation.

In conclusion MNAS and CS would have an slight advantage in settings where the kernel matrix is very big and/or the size of the available cache is small. On the other hand, it is also important to note that there are better SMO implementations in low level languages, such as LIBSVM. For that reason we believe it is necessary to implement both methods inside the

LIBSVM framework to perform a runtime comparison closer to a real setting. However, we will only implement in C the Conjugate SMO variant, since the preliminary Python results for Nesterov's Acceleration are not very encouraging.

6.3.4 SMO: Comparison versus LIBSVM

The cache is one of the most important features of LIBSVM, which is also critical to an efficient SMO implementation. In the previous Python experiments we implemented a very simple unlimited cache. However, this is not realistic since it could be the case where all the necessary rows of the kernel matrix do not fit entirely into memory. Thus LIBSVM implements a simple yet effective "First In First Out" (FIFO) cache where the most recently used kernel rows are stored. Furthermore, the size of the cache can be specified in Mb by using a parameter. This is preferred to pre-computing the whole matrix since, in general, not all the samples are going to be support vectors and thus it is wasteful to compute their corresponding rows in advance.

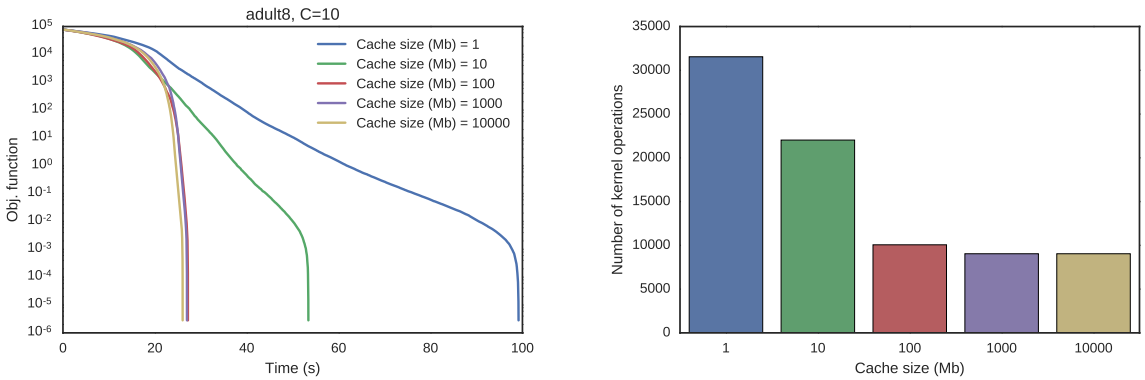

Figure 6.7: Time evolution of the objective function (left) and number of kernel operations (right) for different cache sizes.

First we show empirically how critical is the cache size when solving the SVC problem. For this we have selected the `adult8` dataset, $C = 10$ and $\gamma = 1/d$. Figure 6.7 shows how the execution time grows as a function of the cache size. In this example having no cache can be up to 4 times slower than an unlimited cache.

We investigate next the effect of the cache size in Conjugate SMO by implementing the algorithm inside the LIBSVM framework. The datasets used in this experiment are `adult9`, which is a bigger version of `adult8` with 32561 samples, and `web8`. The reason for this is that the rest of the datasets have a very low number of samples and thus a relatively small cache is already able to store all the necessary kernel rows.

We set the precision and γ to LIBSVM's default values, $\epsilon = 0.001$ and $\gamma = 1/d$. Then, for every dataset we measure the execution time for three different C values, 1, 10, 100, and six different cache sizes 1, 50, 100, 500, 1000, 5000 Mb. Every C and cache size combination was run 10 times and we repeated this procedure 3 times. At the end we computed the average of the 10 runs and for every value took the min of the 3 repetitions. Table 6.11 reports the number of iterations, final objective value and number of support vectors of SMO and Conjugate SMO (CS). As it can be seen, our C implementation of the CS algorithm is correct since it yields basically the same models as SMO. Besides, CS always performs less iterations than standard SMO in this two datasets, although as we discussed before each iteration is costlier.

The execution time in seconds is given in Table 6.12 for every C and cache size values and

Table 6.11: Iterations, objective value and number of support vectors for the `adult9` and `web8` datasets

		Iterations			$f(\boldsymbol{\alpha}^*)$		nSV	
Dataset	C	SMO	CS	Ratio	SMO	CS	SMO	CS
`adult9`	1	8 225	7 907	1.04	−11 596.356	−11 596.356	11 953	11 949
	10	21 256	16 736	1.27	−110 168.640	−110 168.643	11 465	11 473
	100	189 292	110 864	1.71	−1 036 086.460	−1 036 086.481	11 267	11 292
`web8`	1	3 089	2 976	1.04	−2 594.670	−2 594.670	2 951	2 952
	10	7 759	7 164	1.08	−18 638.315	−18 638.317	2 448	2 438
	100	32 556	26 909	1.21	−133 993.552	−133 993.563	2 035	2 043

Table 6.12: Running time in seconds for the `adult9` and `web8` datasets

		Time (s)					
		`adult9`			`web8`		
C	Cache (Mb)	SMO	CS	Ratio	SMO	CS	Ratio
1	1	10.843	10.660	1.02	2.847	2.781	1.02
	50	5.732	5.878	0.98	2.019	2.059	0.98
	100	5.723	5.788	0.99	1.970	2.017	0.98
	500	5.728	5.952	0.96	1.788	1.845	0.97
	1 000	5.653	5.760	0.98	1.767	1.808	0.98
	5 000	5.659	5.717	0.99	1.759	1.823	0.96
10	1	29.301	21.768	1.35	7.392	7.010	1.05
	50	6.891	6.966	0.99	2.575	2.751	0.94
	100	6.207	6.256	0.99	1.921	2.027	0.95
	500	6.080	6.203	0.98	1.655	1.778	0.93
	1 000	5.989	6.080	0.99	1.660	1.785	0.93
	5 000	5.826	5.953	0.98	1.671	1.778	0.94
100	1	155.368	88.948	1.75	30.597	27.013	1.13
	50	72.974	50.073	1.46	13.169	13.468	0.98
	100	29.823	24.959	1.19	3.725	4.183	0.89
	500	13.696	12.283	1.12	2.442	2.740	0.89
	1 000	12.518	11.367	1.10	2.445	2.763	0.88
	5 000	11.875	10.659	1.11	2.441	2.760	0.88

both datasets. Comparing the two tables we can see that better iteration ratios not always correspond to faster execution times. In general CS is faster than SMO the smaller the cache and the larger the C value. Also note that the number of iterations is independent of the cache size.

The same results are also depicted in Fig. 6.8. The conclusions are two fold. First, as it was expected, the execution time decreases the larger the cache since more kernel rows can be stored and reused. However, there is a critical value for the size of the cache from where improvement is almost non-existent. This value depends on both the data and C value. Second, CS is faster than plain SMO in settings with low-to-medium cache sizes for the `adult9` dataset and low cache sizes for the `web7` dataset. Note that LIBSVM default cache size is 100 Mb.

Table 6.13: Dimensions, data sizes and class sizes of the datasets considered.

Dataset	d	n	n^+	n^-
`adult8`	123	22 696	5 506	17 190
`web8`	300	49 749	1 479	48 270
`ijcnn1`	22	49 990	4 853	45 137
`cod-rna`	8	59 535	19 845	39 690
`mnist1`	784	60 000	6 742	53 258
`skin`	3	245 057	50 859	194 198

In the next experiment we are going to measure the execution time and number of iterations of the two biggest problems from Section 6.3.2, `adult8` and `web8`, together with `ijcnn1`, `cod-rna`, `mnist1` and `skin`. The dimensionality and number of samples for these problems is shown in Table 6.13. As it can be seen in the table, we have selected these problems since they are much bigger than the ones in the previous experiments. Thus, since the C implementation in LIBSVM is much faster we are looking to obtain more meaningful results. Furthermore LIBSVM could probably solve the `heart` or `diabetes` datasets in less than a millisecond. As before, these datasets correspond to binary classification problems and can be found in LIBSVM's data repository, except from `mnist1`. The MNIST database consists on 28×28 images of handwritten digits that have been preprocessed and normalized. Since it is a multi-class classification problem we use here the binary version that tries to distinguish the number 1 from the rest.

We are going to compare three different cache sizes (1, 100, 1000 Mb), C values (1, 10, 100) and γ values (1, 0.1, 0.01). As before, γ values are given as multiples of the default $1/d$. Thus, for instance, $\gamma = 0.1$ is actually $\gamma = 0.1 \times 1/d$. Note that in this experiment we have fixed the precision $\epsilon = 0.001$, since we have introduced the cache size as a new variable to the comparison.

The running times of SMO and Conjugate SMO (CS), together with the relative time difference can be found in Tables 6.14 to 6.19, columns 4, 5 and 7. The running times are the averages of 50 executions. We define the relative time difference as

$$\text{RTD} = \frac{\text{Time(SMO)} - \text{Time(CS/HS)}}{\text{Time(SMO)}} \times 100.$$

Note that, using the previous definition, if the relative time difference is positive then it means that either CS or HS are faster than SMO, if it is negative they are slower. For completeness iteration results are given in Appendix C, Tables C.1 to C.6. In general, Conjugate SMO is faster than standard SMO in the problems `skin`, `mnist1`, `adult8` and `cod-rna`, while it is slower in `web8` and `ijcnn1`. As we discussed before the conjugate version is usually better the smaller the cache size and the bigger the C and γ.

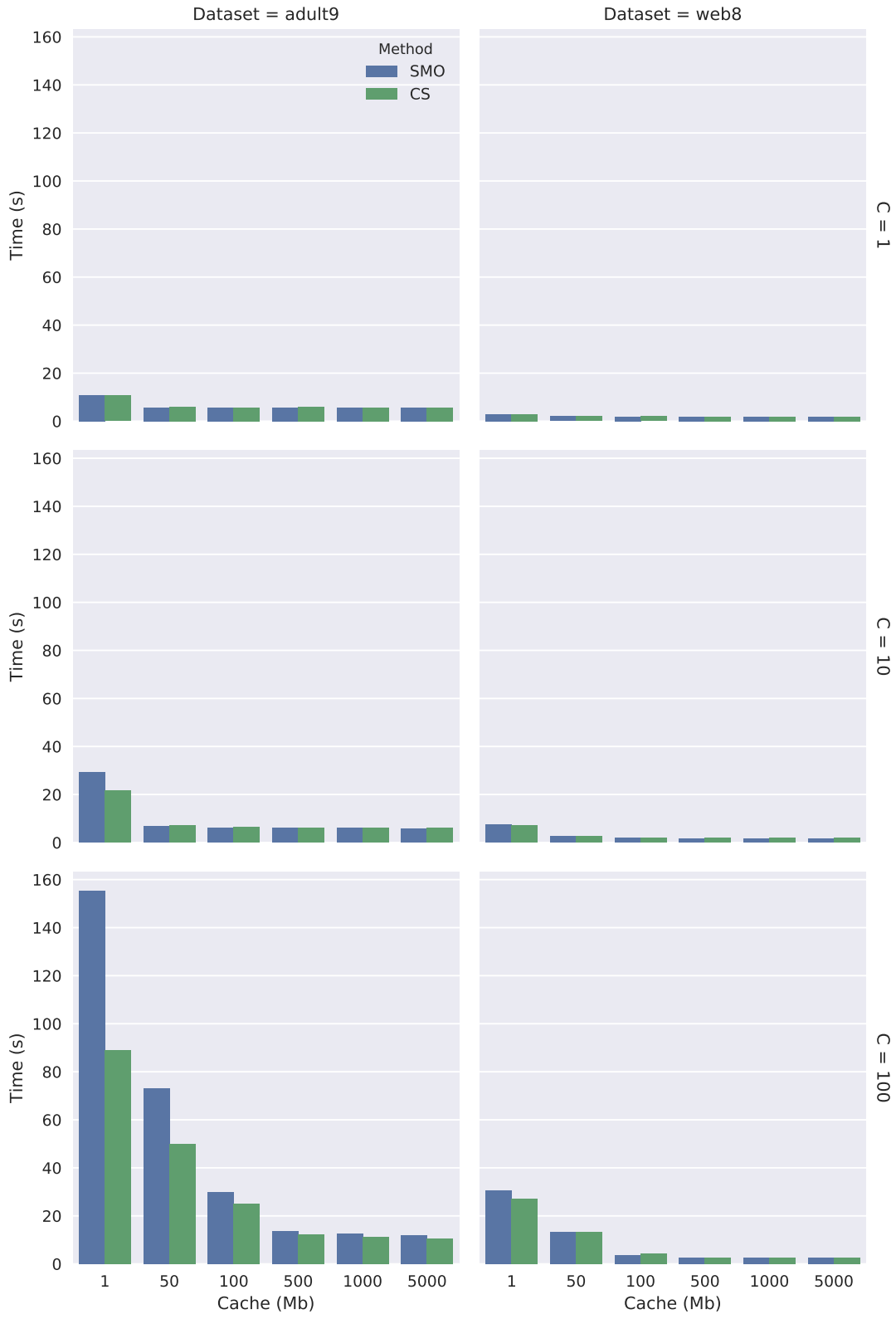

Figure 6.8: Comparison between SMO and CS of the running time as a function of the cache size for the `adult9` and `web7` datasets and different C values

Table 6.14: Comparison of the running time in seconds between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `adult8` dataset

			Time (s)			RTD (%)	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	2.16	2.16	2.13	0.14	1.71
		−1	1.97	2.01	1.97	−2.13	−0.15
		0	2.05	2.03	1.98	1.31	3.60
		1	4.36	3.85	3.86	11.81	11.49
	10	−2	2.31	2.45	2.39	−6.37	−3.60
		−1	2.45	2.50	2.37	−1.84	3.14
		0	5.44	4.32	4.37	20.50	19.62
		1	23.06	18.77	20.12	18.63	12.77
	100	−2	2.21	2.39	2.34	−8.09	−5.60
		−1	5.88	4.47	4.71	23.93	19.90
		0	46.29	30.01	34.47	35.18	25.53
		1	107.93	83.18	89.30	22.93	17.27
	1 000	−2	6.33	5.05	5.03	20.31	20.50
		−1	42.55	27.31	31.47	35.82	26.04
		0	384.45	202.98	248.07	47.20	35.47
		1	271.10	166.22	199.24	38.69	26.51
100	1	−2	2.39	2.51	2.44	−5.03	−2.14
		−1	2.03	2.02	1.97	0.74	2.95
		0	1.79	1.82	1.82	−1.51	−1.67
		1	2.29	2.40	2.29	−4.63	0.04
	10	−2	2.15	2.27	2.21	−5.59	−3.12
		−1	1.76	1.81	1.76	−2.96	−0.28
		0	1.87	1.94	1.87	−4.02	−0.11
		1	15.01	12.74	13.40	15.09	10.70
	100	−2	2.07	2.12	2.05	−2.56	1.11
		−1	2.02	2.11	2.04	−4.66	−1.09
		0	5.44	5.28	5.45	2.85	−0.22
		1	73.44	58.92	64.08	19.77	12.75
	1 000	−2	2.18	2.25	2.15	−3.40	1.33
		−1	5.25	4.82	5.01	8.20	4.50
		0	88.28	65.33	76.48	26.00	13.36
		1	147.08	109.23	120.86	25.74	17.83
1 000	1	−2	2.42	2.51	2.32	−3.47	4.37
		−1	2.20	2.27	2.14	−3.55	2.69
		0	1.96	2.02	1.96	−2.91	0.00
		1	2.04	2.11	2.03	−3.49	0.20
	10	−2	1.88	1.90	1.90	−0.96	−0.80
		−1	1.96	2.01	1.97	−2.55	−0.41
		0	2.03	2.11	2.03	−4.04	−0.15
		1	3.12	3.36	3.11	−7.86	0.13
	100	−2	1.75	1.84	1.74	−5.32	0.57
		−1	1.80	1.86	1.82	−3.39	−1.06
		0	3.45	3.32	3.34	3.68	3.25
		1	6.48	6.87	6.77	−6.08	−4.62
	1 000	−2	2.05	2.19	2.11	−6.99	−3.22
		−1	3.83	3.70	3.64	3.26	4.91
		0	19.17	17.17	18.23	10.48	4.95
		1	13.72	12.36	12.85	9.90	6.33

Table 6.15: Comparison of the running time in seconds between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `web8` dataset

			Time (s)			RTD (%)	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	1.45	1.51	1.46	−4.56	−0.76
		−1	1.51	1.51	1.50	−0.13	0.93
		0	1.83	1.79	1.73	1.75	5.37
		1	3.34	3.23	3.25	3.24	2.55
	10	−2	1.59	1.63	1.61	−2.26	−1.26
		−1	1.96	1.87	1.90	4.49	3.21
		0	4.97	4.77	4.60	3.99	7.45
		1	9.83	8.53	8.74	13.27	11.14
	100	−2	1.86	1.84	1.87	1.13	−0.59
		−1	4.51	4.53	4.46	−0.44	1.02
		0	20.54	18.82	19.59	8.40	4.65
		1	16.38	13.04	13.79	20.40	15.81
	1 000	−2	5.74	5.42	5.10	5.51	11.19
		−1	27.50	22.73	24.87	17.36	9.56
		0	75.33	64.17	66.95	14.82	11.13
		1	24.53	16.27	18.51	33.67	24.54
100	1	−2	1.38	1.41	1.38	−2.24	−0.22
		−1	1.45	1.50	1.47	−3.31	−1.45
		0	1.27	1.31	1.28	−3.39	−1.10
		1	1.94	2.02	2.03	−4.02	−4.44
	10	−2	1.40	1.45	1.43	−3.42	−1.99
		−1	1.32	1.37	1.33	−3.88	−0.99
		0	1.24	1.30	1.27	−5.59	−2.83
		1	6.08	5.71	5.69	6.02	6.29
	100	−2	1.31	1.37	1.32	−4.98	−1.07
		−1	1.24	1.40	1.30	−12.57	−4.92
		0	2.58	3.07	2.90	−19.16	−12.53
		1	9.33	8.23	8.33	11.84	10.75
	1 000	−2	1.24	1.40	1.33	−13.32	−7.67
		−1	2.19	2.55	2.47	−16.70	−13.23
		0	24.34	25.42	25.01	−4.44	−2.76
		1	11.20	9.63	9.79	13.94	12.52
1 000	1	−2	1.20	1.25	1.22	−4.34	−1.50
		−1	1.18	1.21	1.18	−2.64	−0.60
		0	1.14	1.19	1.15	−3.67	−0.26
		1	1.11	1.18	1.15	−5.57	−3.14
	10	−2	1.21	1.24	1.21	−2.56	0.08
		−1	1.16	1.24	1.21	−6.10	−3.52
		0	1.10	1.23	1.18	−12.11	−7.10
		1	1.39	1.57	1.50	−13.07	−7.97
	100	−2	1.21	1.27	1.21	−4.87	0.00
		−1	1.09	1.27	1.14	−16.57	−4.60
		0	1.65	1.99	2.02	−20.75	−22.63
		1	1.60	1.76	1.68	−9.80	−5.06
	1 000	−2	1.04	1.15	1.09	−10.72	−5.12
		−1	1.67	1.99	1.93	−19.57	−15.67
		0	4.42	5.76	5.37	−30.32	−21.54
		1	2.10	2.04	1.98	2.90	5.86

Table 6.16: Comparison of the running time in seconds between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `ijcnn1` dataset

			Time (s)			RTD (%)	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	4.58	4.84	4.66	−5.70	−1.81
		−1	4.29	4.53	4.43	−5.64	−3.31
		0	3.29	3.33	3.22	−1.12	2.37
		1	2.48	2.48	2.37	−0.08	4.16
	10	−2	4.64	5.03	4.75	−8.29	−2.28
		−1	3.52	3.63	3.52	−3.18	−0.03
		0	3.80	3.87	3.75	−1.90	1.21
		1	5.67	4.93	5.00	13.20	11.97
	100	−2	3.70	3.77	3.61	−1.70	2.56
		−1	5.44	4.83	4.61	11.30	15.27
		0	10.07	7.61	8.18	24.47	18.79
		1	25.50	18.63	22.88	26.94	10.25
	1 000	−2	7.13	5.56	5.74	22.04	19.55
		−1	15.64	10.30	11.40	34.12	27.09
		0	67.21	47.18	52.90	29.80	21.28
		1	110.51	79.89	87.64	27.71	20.69
100	1	−2	4.03	4.14	4.07	−2.71	−1.02
		−1	3.89	4.03	3.89	−3.62	−0.03
		0	2.87	2.99	2.92	−4.15	−1.81
		1	2.23	2.35	2.25	−5.29	−1.03
	10	−2	4.15	4.52	4.41	−8.99	−6.29
		−1	3.25	3.38	3.20	−4.28	1.33
		0	2.73	2.81	2.77	−2.78	−1.28
		1	1.61	1.69	1.65	−4.53	−2.23
	100	−2	3.59	3.87	3.76	−7.89	−4.99
		−1	3.09	3.28	3.12	−5.99	−1.00
		0	2.24	2.33	2.25	−4.07	−0.76
		1	4.94	4.99	5.07	−0.87	−2.59
	1 000	−2	3.75	3.79	3.67	−1.12	2.29
		−1	3.48	3.43	3.45	1.44	0.66
		0	5.96	6.09	6.17	−2.11	−3.57
		1	35.55	38.06	41.53	−7.06	−16.82
1 000	1	−2	4.00	4.16	4.08	−4.07	−1.90
		−1	3.70	3.97	3.72	−7.16	−0.51
		0	3.49	3.67	3.51	−5.33	−0.52
		1	2.21	2.26	2.22	−2.12	−0.09
	10	−2	3.77	3.97	3.86	−5.25	−2.36
		−1	3.65	3.78	3.70	−3.37	−1.29
		0	2.10	2.19	2.11	−4.13	−0.14
		1	1.29	1.38	1.33	−6.50	−2.55
	100	−2	3.59	3.77	3.63	−4.93	−1.03
		−1	2.81	2.91	2.82	−3.78	−0.64
		0	1.92	1.99	1.93	−3.70	−0.68
		1	2.06	2.38	2.19	−15.63	−6.07
	1 000	−2	3.45	3.75	3.56	−8.70	−3.19
		−1	2.98	2.98	2.88	−0.03	3.39
		0	4.60	5.62	5.63	−22.19	−22.43
		1	7.14	9.95	9.34	−39.36	−30.84

Table 6.17: Comparison of the running time in seconds between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `cod-rna` dataset

			Time (s)			RTD (%)	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	26.21	17.07	18.51	34.88	29.36
		−1	37.90	26.03	26.00	31.31	31.39
		0	39.69	25.62	25.88	35.45	34.80
		1	14.63	14.69	14.42	−0.46	1.38
	10	−2	108.47	53.59	56.65	50.59	47.77
		−1	154.73	77.41	82.37	49.97	46.76
		0	114.23	57.70	62.13	49.49	45.61
		1	15.11	15.30	15.06	−1.31	0.32
	100	−2	464.84	201.95	209.17	56.55	55.00
		−1	589.36	274.15	285.78	53.48	51.51
		0	324.78	143.22	174.44	55.90	46.29
		1	21.40	20.44	20.58	4.48	3.82
	1 000	−2	2 679.69	1 073.81	1 152.11	59.93	57.01
		−1	2 721.41	1 136.50	1 330.60	58.24	51.11
		0	1 080.81	447.39	595.28	58.61	44.92
		1	35.46	27.57	30.07	22.25	15.21
100	1	−2	10.27	10.79	10.65	−5.07	−3.69
		−1	16.77	16.58	16.28	1.16	2.93
		0	22.83	19.82	19.58	13.22	14.23
		1	12.47	12.99	12.86	−4.21	−3.18
	10	−2	18.15	16.57	16.80	8.73	7.46
		−1	37.72	30.73	32.13	18.53	14.82
		0	44.82	35.88	37.16	19.94	17.09
		1	14.29	14.77	14.54	−3.37	−1.72
	100	−2	49.91	40.20	40.42	19.44	19.01
		−1	109.66	83.25	84.41	24.08	23.03
		0	95.19	69.10	75.81	27.41	20.35
		1	20.14	21.02	20.39	−4.37	−1.26
	1 000	−2	245.58	184.08	190.81	25.04	22.30
		−1	467.06	319.20	350.41	31.66	24.98
		0	294.61	191.81	207.00	34.89	29.74
		1	27.85	26.43	26.48	5.07	4.90
1 000	1	−2	8.20	8.20	8.20	−0.02	0.05
		−1	9.64	10.44	10.12	−8.31	−4.99
		0	9.99	10.55	10.09	−5.61	−1.08
		1	11.84	12.46	12.21	−5.23	−3.15
	10	−2	14.28	12.99	13.10	9.04	8.27
		−1	18.12	17.52	16.12	3.27	11.01
		0	16.72	16.48	15.73	1.48	5.97
		1	12.87	13.52	13.08	−5.03	−1.62
	100	−2	39.52	29.87	29.29	24.42	25.88
		−1	49.95	43.82	41.77	12.29	16.38
		0	37.20	33.31	32.85	10.45	11.71
		1	16.72	17.89	17.76	−6.98	−6.20
	1 000	−2	200.06	148.61	146.89	25.71	26.58
		−1	185.88	139.70	146.65	24.84	21.10
		0	112.95	93.98	100.01	16.80	11.45
		1	19.60	19.72	19.61	−0.62	−0.04

Table 6.18: Comparison of the running time in seconds between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `mnist1` dataset

			Time (s)			RTD (%)	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	59.57	56.48	56.26	5.19	5.57
		−1	24.58	24.17	24.14	1.67	1.79
		0	44.21	42.55	46.66	3.75	−5.53
		1	58.35	55.59	55.92	4.75	4.16
	10	−2	25.64	25.31	25.23	1.30	1.61
		−1	55.92	55.36	57.33	1.00	−2.53
		0	143.71	111.79	104.75	22.21	27.11
		1	94.37	93.13	87.82	1.32	6.94
	100	−2	30.03	25.74	26.49	14.28	11.81
		−1	271.66	174.21	160.50	35.87	40.92
		0	154.94	118.63	128.48	23.43	17.08
		1	99.29	96.68	89.10	2.63	10.27
	1 000	−2	151.94	113.04	121.32	25.60	20.15
		−1	653.42	479.17	447.26	26.67	31.55
		0	168.23	116.85	126.87	30.54	24.59
		1	94.52	92.85	86.79	1.77	8.18
100	1	−2	131.12	118.44	96.93	9.67	26.07
		−1	38.86	37.89	39.43	2.51	−1.44
		0	17.00	16.56	17.54	2.59	−3.20
		1	40.12	40.02	39.57	0.25	1.37
	10	−2	51.02	51.50	51.54	−0.94	−1.02
		−1	14.89	14.95	14.83	−0.38	0.40
		0	33.81	32.56	33.83	3.71	−0.05
		1	51.58	50.01	49.12	3.05	4.77
	100	−2	36.42	38.52	38.12	−5.77	−4.68
		−1	34.78	33.69	34.88	3.14	−0.30
		0	117.64	91.86	83.18	21.91	29.29
		1	103.72	97.23	88.29	6.26	14.88
	1 000	−2	19.36	18.67	20.08	3.56	−3.76
		−1	229.43	161.16	162.76	29.75	29.06
		0	148.02	99.32	91.31	32.90	38.31
		1	47.27	45.62	45.17	3.49	4.43
1 000	1	−2	54.18	54.62	53.44	−0.81	1.36
		−1	23.22	23.08	23.09	0.60	0.56
		0	10.40	10.38	10.44	0.23	−0.41
		1	17.19	17.34	16.63	−0.87	3.28
	10	−2	49.17	46.51	45.92	5.43	6.63
		−1	19.59	19.68	19.77	−0.45	−0.89
		0	6.81	6.94	6.92	−1.91	−1.57
		1	30.20	30.09	27.85	0.35	7.79
	100	−2	16.12	16.19	15.88	−0.47	1.45
		−1	16.33	17.14	13.53	−4.94	17.17
		0	6.33	6.43	6.41	−1.52	−1.17
		1	13.78	13.76	13.94	0.14	−1.17
	1 000	−2	11.43	11.54	11.46	−0.94	−0.21
		−1	20.57	21.54	21.13	−4.71	−2.70
		0	13.39	14.04	13.44	−4.84	−0.42
		1	29.51	27.21	26.77	7.81	9.28

Table 6.19: Comparison of the running time in seconds between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `skin` dataset

			Time (s)			RTD (%)	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	11.03	10.50	10.38	4.84	5.90
		−1	123.77	109.23	109.82	11.75	11.27
		0	189.29	190.54	177.36	−0.66	6.30
		1	114.25	123.30	121.64	−7.92	−6.46
	10	−2	12.65	10.69	10.95	15.51	13.50
		−1	124.41	113.44	110.41	8.81	11.25
		0	181.57	181.26	176.78	0.17	2.64
		1	157.65	175.70	164.93	−11.45	−4.62
	100	−2	34.51	23.16	23.60	32.89	31.62
		−1	118.52	104.39	104.99	11.92	11.41
		0	186.75	189.37	174.73	−1.40	6.44
		1	153.79	168.18	159.77	−9.36	−3.89
	1 000	−2	172.00	88.92	108.53	48.30	36.90
		−1	115.65	101.33	101.10	12.38	12.58
		0	185.17	185.34	171.80	−0.10	7.22
		1	152.15	165.21	158.80	−8.59	−4.37
100	1	−2	10.13	10.11	10.00	0.23	1.24
		−1	110.49	98.79	98.63	10.59	10.73
		0	185.21	187.06	170.46	−1.00	7.96
		1	122.17	142.05	131.50	−16.27	−7.64
	10	−2	8.73	8.40	8.41	3.68	3.58
		−1	113.39	105.28	104.43	7.16	7.90
		0	178.33	170.59	167.36	4.34	6.15
		1	158.67	175.68	168.31	−10.72	−6.08
	100	−2	7.42	7.60	7.47	−2.44	−0.62
		−1	108.99	101.15	100.31	7.20	7.97
		0	186.96	184.09	182.90	1.53	2.17
		1	154.13	168.54	161.91	−9.35	−5.05
	1 000	−2	22.90	22.47	24.88	1.85	−8.64
		−1	106.83	100.08	98.94	6.32	7.39
		0	176.79	169.66	166.38	4.03	5.88
		1	154.30	170.44	162.68	−10.46	−5.43
1 000	1	−2	4.98	5.80	5.68	−16.45	−14.16
		−1	108.89	102.26	97.92	6.09	10.08
		0	178.25	174.17	172.24	2.29	3.37
		1	116.24	130.91	123.52	−12.62	−6.26
	10	−2	3.56	4.29	4.15	−20.29	−16.30
		−1	104.61	97.46	97.03	6.84	7.25
		0	178.76	172.96	169.18	3.25	5.36
		1	157.77	176.86	162.57	−12.10	−3.04
	100	−2	3.10	4.53	4.10	−46.14	−32.39
		−1	103.47	96.27	96.49	6.96	6.75
		0	184.46	172.18	166.91	6.66	9.51
		1	158.96	173.93	158.01	−9.42	0.60
	1 000	−2	14.02	16.61	17.54	−18.51	−25.12
		−1	99.89	92.02	91.40	7.88	8.50
		0	180.83	176.25	166.77	2.54	7.78
		1	157.46	167.73	157.80	−6.52	−0.22

We also experiment with an hybrid SMO/CS version, where we perform standard SMO iterations until we achieve a precision $\epsilon = 0.1$ and then we switch to the conjugate iterations until the final ϵ-optimal solution is reached. In our case we set the precision for this final solution to $\epsilon = 0.001$. The results are given in columns 6 and 8 of Tables 6.14 to 6.19. In general, when the CS is faster than SMO, HS is also faster but with lower ratio. On the other hand, we don't observe any noticeable improvements in HS over SMO when CS is slower.

To better visualize the dependence on C and γ we have plotted in Fig. 6.9 heatmaps of the relative time difference between SMO, Conjugate SMO and Hybrid SMO for LIBSVM default cache size, 100 Mb. Here we can see, for each of the datasets, combinations of C and γ where CS/HS is faster than SMO (in red) and where it is slower (in blue). Thus, for the datasets `cod-rna`, `mnist1` and `skin`, CS and HS are always faster or at least equally fast than SMO for every C and γ with the exception of $C = 10$, $\gamma = 0.01$ in `mnist1` and $C = 10$, $\gamma = 1.0$ in `skin`. For the dataset `adult8` CS/HS are better for the upper triangular part of the heatmap, that is, large C and γ. Finally, for the datasets `web8` and `ijcnn1` CS/HS are usually slower except for $C = 100$ and $\gamma = 1.0$ in `ijcnn1`.

Table 6.20: Relative time difference as a percentage between SMO and CS for a full hyper-parameter search with $\epsilon = 0.001$ and a 100 Mb cache

	Time (s)		
Dataset	SMO	CS	RTD (%)
heart	0.025	0.020	20.00
diabetes	0.419	0.386	7.88
australian	0.262	0.213	18.70
german	0.841	0.705	16.17
adult4	10.857	9.212	15.18
web7	153.561	155.992	−1.58
adult8	2 546.097	1 608.115	36.84
web8	748.557	752.478	−0.52
ijcnn1	495.470	439.572	11.28
mnist1	95 460.910	85 319.661	10.62
cod-rna	23 320.557	13 029.203	44.13
skin	2 541.449	2 484.472	2.24

Although in our experiments we have only checked a few values for C and γ, when tuning an SVM to build the final model it is common to search a bigger grid. As an example, in Fan et al. (2005b) they search from $C = 2^{-5} = 3.125 \times 10^{-2}$ to $C = 2^{15} = 32768$ with steps of 2 in the $\log_2$ scale, that is, $2^{-5}, 2^{-3}, \ldots, 2^{13}, 2^{15}$. For the optimal γ value a similar technique is used, starting in $\gamma = 2^{-15} = 3.052 \times 10^{-5}$ and ending up in $\gamma = 2^3 = 8$, again in the $\log_2$ scale and with steps of 2. For that reason, and given that CS seems to perform better than SMO in a fairly large section of the previous grid, it is interesting to compare the running time of both when performing a full hyper-parameter search. The results of this experiment are given in Table 6.20, where we show the total time of performing the search for the grid described above, with $\epsilon = 0.001$ and a cache size of 100 Mb (LIBSVM's defaults). As before, the running times are the average of 50 executions, but here we have also repeated the previous process 3 times, taking the minimum among them. CS significantly outperforms SMO in 9 out of the 12 datasets considered and it is also almost as fast as SMO in `skin`, `web7`, `web8`.

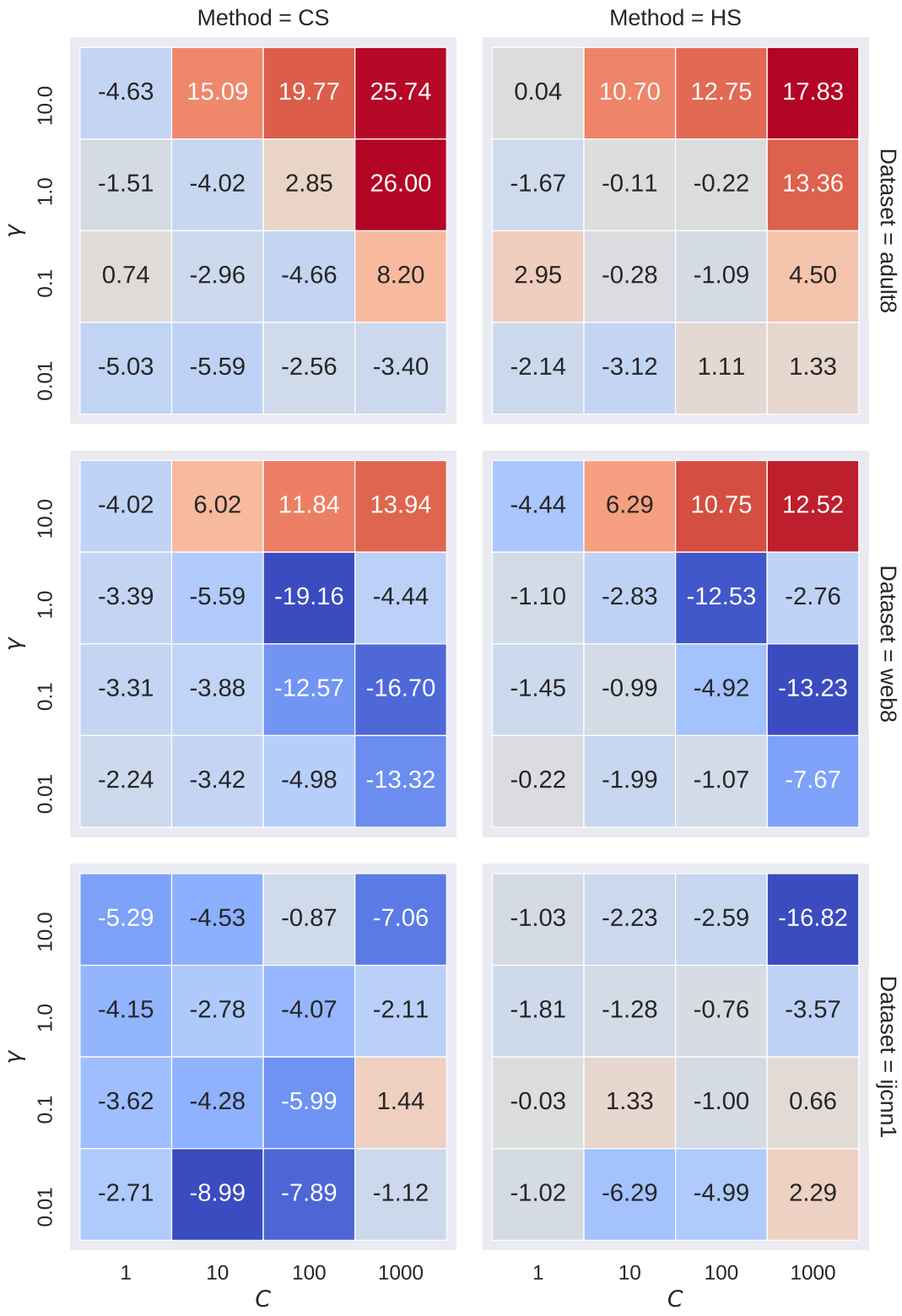

Figure 6.9: Relative time difference heatmap with a cache size of 100 Mb (a)

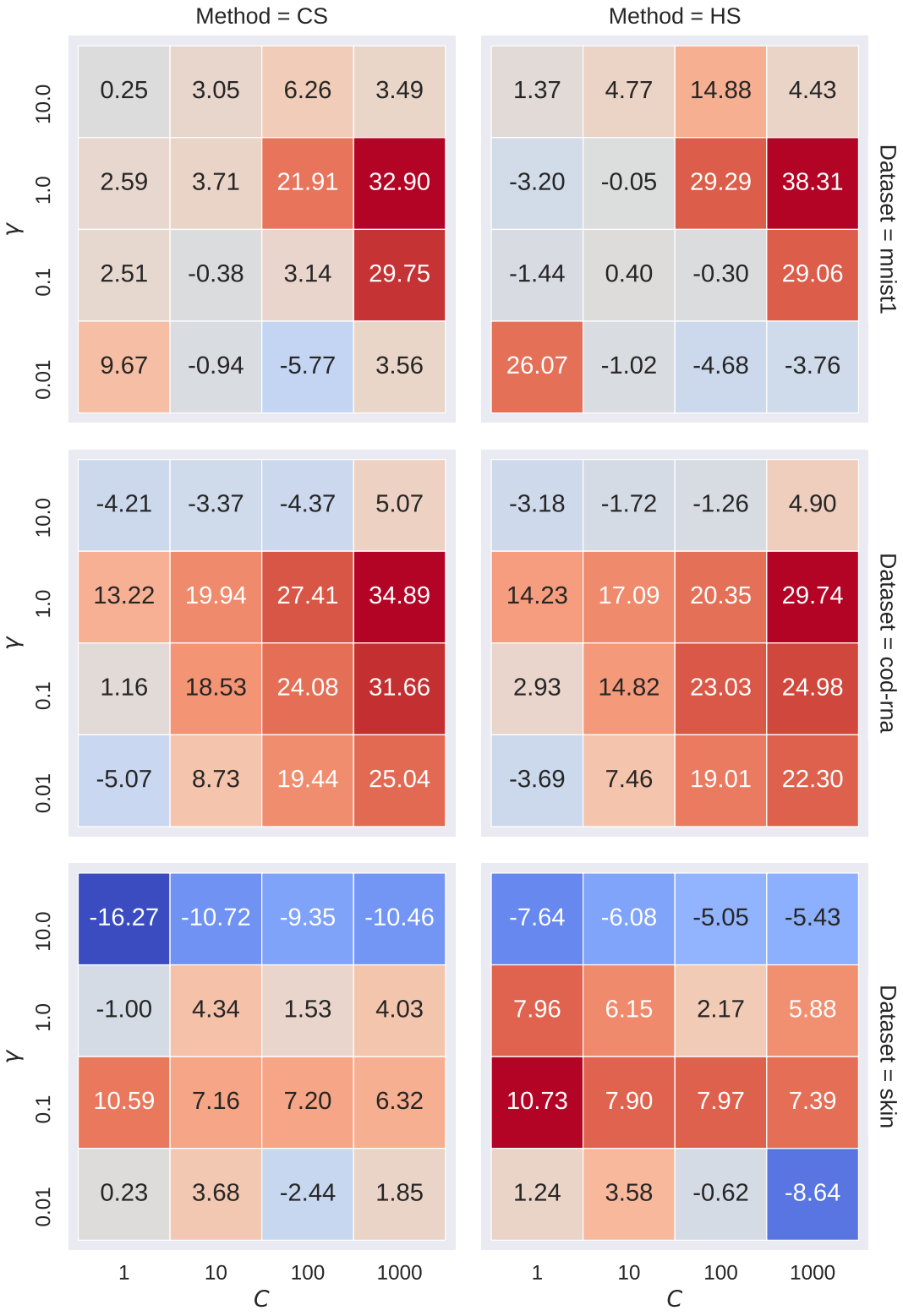

Figure 6.9: Relative time difference heatmap with a cache size of 100 Mb (b)

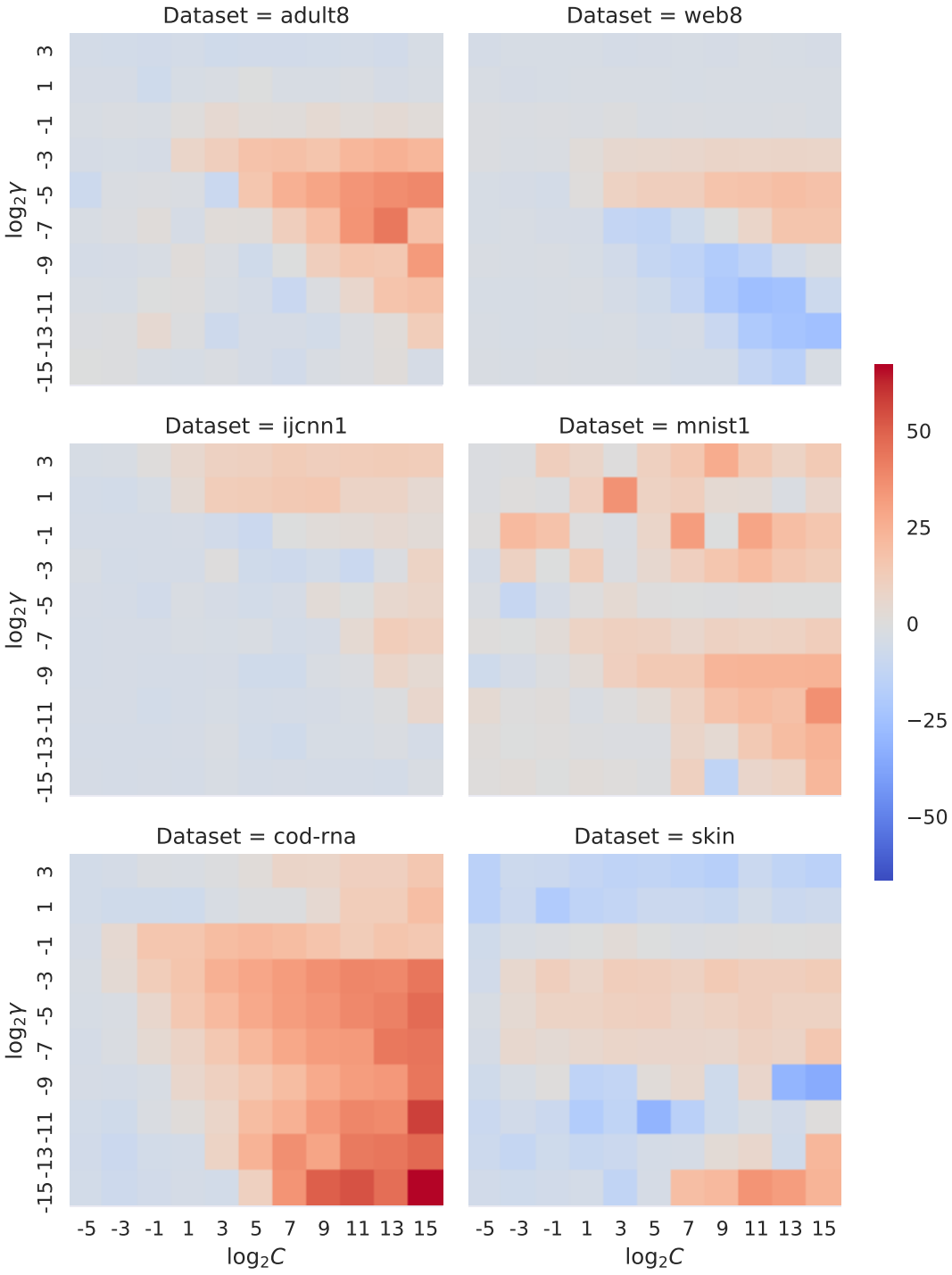

Figure 6.10: Relative time difference heatmap with a cache size of 100 Mb for the different C and γ values of a full hyper-parameter search

These results are also displayed in Fig. 6.10 for the six largest datasets. Here we can see how there is usually an upper-triangle in the C-γ grid where Conjugate SMO consistently outperforms standard SMO. It is also interesting to see that, in general, larger γ values benefit the Conjugate implementation but only up to a certain threshold, from where SMO starts to be better. On the other hand it is quite clear that Conjugate SMO outperforms SMO for large C values.

6.4 Discussion and further work

We have shown how a conjugate variant of the MDM algorithm (i.e., the Frank-Wolfe algorithm with swap steps) for the MNP achieves a substantial reduction of the number of iterations needed to arrive at a given convergence precision. We have also showed how this reduction in the number of iterations also translates in most cases to a reduction in the running time of the algorithm.

The Minimum Norm Problem is interesting not only on its own but also because it provides an alternative way of solving the constrained Lasso problem, using the equivalence described in Chapter 5. Another application of the conjugate directions lies with the similarity between the MDM and SMO algorithms. The basic structure of the SMO iterations is very similar to the MDM ones and so it is the construction of conjugate descent directions.

In this chapter we have also reviewed two classical variants of momentum, the Heavy Ball algorithm and Nesterov's Accelerated Gradient, and derived the corresponding SMO versions: Monotone Nesterov's Accelerated SMO and Conjugate SMO. This two algorithms achieve a substantial reduction of the number of iterations needed for SMO to arrive at a given convergence precision at the cost of some extra iteration complexity. They also obtain essentially the same final model in terms of the final number of SVs and the final value of the SVC cost function.

For Nesterov's Acceleration our complexity analysis in floating point operations suggests that the reduction in the number of iterations can be achieved with an iteration cost about twice as large as SMO's. On the other hand Conjugate SMO is a little bit cheaper, performing only 4/3 more floating point operations than SMO. In our experiments comparing our own Python implementation of the three algorithms we have seen that the empirical average iteration ratios that also yield faster execution times are 1.68 and 1.35 for MNAS and CS respectively, with $\epsilon = 0.001$. However, note that the these timing results were obtained using a possibly inefficient Python implementation of the SMO algorithm.

In general Conjugate SMO obtained relative good time ratios, specially for large C and γ. The running times are also better for lower values of the precision ϵ. On the other hand, the ratios for Monotone Nesterov SMO are not as promising except for the `adult` dataset, where MNAS works better than CS for $C = 100$. Thus, from a practical point of view, and as we have made clear, implementing MNAS inside the LIBSVM framework is not likely to result in training times smaller than those of standard SMO.

We have also implemented the Conjugate SMO algorithm inside the LIBSVM framework, the state of the art solver for non-linear SVMs. Using this implementation we have compared SMO and CSMO in terms of the number of iterations and running times for different variables like C value, kernel width and the size of the cache, which directly impacts the performance of these methods. As we have seen, in general the Conjugate variant is better than SMO for larger C and γ values. A discussion of the effects of these parameters for SVC and Gaussian kernels is made in Keerthi and Lin (2003) and in principle, large γ and C should lead to more difficult SVM problems.

An hybrid SMO/Conjugate SMO version was also implemented, where we start the optimization performing standard SMO iterations and then switch to the conjugate ones when a threshold precision ϵ is reached. This version performed similarly to Conjugate SMO empirically,

although it opens the way to explore another variants. First, in our experiments we have set the threshold precision to a fixed values $\epsilon = 0.1$. However, this precision could be estimated based on the problem we are solving, deciding on-the-fly at what threshold it is beneficial to start our conjugate iterations (if at all!). Thus, we could try to automatically determine if for the given data, C, γ and cache size is worth to use the conjugate or plain updates and where to make the switch.

Second, if there is a cache miss the iteration cost is dominated by the kernel operation. Thus, the extra cost of Conjugate SMO would be negligible in the initial iterations, since the cache of the kernel matrix is still being built. Following that idea we could also test an hybrid method that starts with the conjugate updates, which are more expensive, and then switches back to the standard ones. As a conclusion, SMO acceleration may be worthwhile if it initially achieves a substantial reduction of the objective function while the kernel matrix cache is built, even if one reverts to standard SMO afterwards. This should be more pronounced in large sample problems and also with large C values.

All these remarks also suggest that performance evaluation of an improved SVM procedure over fixed C, γ values should be complemented with more comprehensive comparisons over wider hyperparameter ranges, perhaps on a dynamic hyperparameter selection context. A particular example is Hyperopt (Bergstra and Bengio, 2012), that focuses on the evaluation of near-optimal hyper-parameters. In our experiments we have seen how Conjugate SMO is faster in ten out of the twelve datasets considered when performing a full hyper-parameter search, using the same grid as in Fan et al. (2005b). As a further work these experiments could also be extended to more datasets and also larger ones.

Another key factor in the effectiveness of an acceleration method is the condition number of SVM's kernel matrix $\mathbf{Q}$. For unconstrained quadratic problems over a positive definite $\mathbf{Q}$, gradient descent has linear convergence with an approximate rate of $1 - 2/\kappa$ (Nesterov, 2004), with κ being Q's condition number. Conjugate Gradient and Nesterov's Acceleration improve this rate to approximately $1 - 2/\sqrt{\kappa}$. This implies that, in principle, larger gains could be achieved in problems with kernel matrices having worse condition numbers. SMO also has linear convergence rate for positive definite kernel matrices $\mathbf{Q}$, which is also loosely related to the condition number of the matrix (Fan et al., 2005a).

Finally, understanding NAG behavior can still be considered as an open problem but in the past couple of years there has been a flurry of activity on the subject (Wibisono et al., 2016; Su et al., 2014; Allen-Zhu and Orecchia, 2014; Bubeck et al., 2015), particularly for the unconstrained, strongly convex case. As we have already mentioned, most of them are theoretical works based on the knowledge of the condition number κ of f, which makes it very difficult to exploit in practical algorithms. Of particular interest here is the work in O'Donoghue and Candès (2015), where they suggests to restart Nesterov's sequence μ_k when it yields a non-monotone step to control the over- or under-damping effect of a possibly too large momentum term. While very simple, in the experiments of O'Donoghue and Candès (2015) this results in a convergence for strongly convex f much faster than the one achieved working with the standard NAG μ_k schedule. In turn, if properly applied, this may help to improve the gains in the number of MNAS iterations reported here, since we could avoid having to compute the objective value in order to check the monotonicity.

In addition, while we only report results for SVC problems, the Conjugate SMO iterations immediately carry over to SVR problems and, in fact, they can also be extended to the ν-SVC and ν-SVR. These models are already implemented in LIBSVM.

Chapter 7

Conclusions

7.1 Discussion

The thesis has two different parts. In the first one, we have studied the Lasso model, proposed by Tibshirani in 1996. The main idea is to add a ℓ_1-penalty to the least squares loss function, achieving not only a good generalization error but also some sparsity in the final coefficients. In this sense the Lasso is not only a regression model but also a feature selection method, since the features that correspond to zeroed out coefficients can be eliminated from the final model. The underlying optimization problem has two main characteristics, both related to the ℓ_1-penalty, that make it more challenging to tackle than, say, Ridge Regression:

1. The objective function is not strongly convex.
2. The ℓ_1-norm is not differentiable.

Among the first algorithms to efficiently solve the Lasso we highlight FISTA, which is just Nesterov's Acceleration applied to Proximal Gradient Descent. Research on algorithms that scale to larger problems went in two main directions: Stochastic Coordinate Descent and Stochastic Gradient Descent.

In Stochastic Coordinate Descent, we randomly select one coefficient and optimize with respect to that coordinate using the full gradient. This algorithm is specially beneficial to the Lasso, since the ℓ_1-norm is separable and thus the inner optimization problem can be solved analytically. On the other hand, in Stochastic Gradient Descent we compute an approximation of the gradient with respect to all coordinates but using only a single data point. In general Coordinate Descent is faster in problems with moderately large sample sizes, up to the hundred thousands. Contributing to the popularity of the algorithm is also GLMNet, a very efficient Fortran/R implementation.

Building on a recent result by Jaggi, where he shows the equivalence between the constrained Lasso and SVMs, we have considered here SMO as a potential Lasso solver and show empirically that it is at least competitive. In particular, our contributions are:

- A refined version of the equivalence between the Lasso and SVMs problems. In particular we arrived at an instance of a geometrical problem known as Nearest Point Problem, which aims at finding the closest points between two convex hulls. The NPP is also equivalent to the ν-SVC formulation, and thus it can be solved directly by LIBSVM.
- A modified version of the SMO algorithm implemented by LIBSVM, especially tailored for this reduction. When solving the Lasso trough its equivalent ν-SVC instance we have

to double the number of variables, obtaining a new artificial data matrix $(\mathbf{X} \,|\, -\mathbf{X})$. Thus, when computing the kernel matrix there are many redundancies. We exploit this fact to suggest a new "kernel-aware" SVM algorithm (K-SVM), which solves the Lasso problem performing up to 8 times less kernel operations than the standard SVM.

- An empirical comparison between Cyclic Coordinate Descent, as implemented in the *scikit-learn* library, SVM and K-SVM. The results show that K-SVM obtains an ϵ-optimal solution faster than its SVM counterpart for $\epsilon = 10^{-6}$, as expected. K-SVM also outperforms Cyclic Coordinate Descent in all the datasets considered except from one.

- A software implementation of the reduction from the Lasso to an SVM instance, in Python, and an implementation of K-SVM inside the LIBSVM framework.

In the second part of this thesis we have focused on acceleration methods for the SMO algorithm. In particular we have developed two SMO variants, one based on the Conjugate Gradient Descent algorithm and another one inspired by Nesterov's Accelerated Gradient. Both acceleration techniques come from the classic convex literature and were known as early as 1963 and 1983, respectively. However, it wasn't until recently that these algorithms were applied to modern Machine Learning models such as deep networks, especially Nesterov's Acceleration. This is in part due to all the recent theoretical work trying to understand why and when the acceleration is possible.

Sequential Minimal Optimization was proposed by Platt in 1998 and since then, it is one of the most widely used algorithms to solve SVMs, mainly the non-linear formulation. This is partially due to the LIBSVM library, which appeared in 2001 and offers a fast and efficient implementation. LIBSVM recieves periodic updates but, apart from bugfixes and code optimizations, the algorithm is essentially SMO together with second order working set selection (Fan et al., 2005b). Furthermore, recent successful advances in SVM optimization try to make the algorithms scale by parallelizing the algorithm, distributing the computation, or both. Motivated by this ideas, we have followed here a different approach. Instead of focusing on the technological side, we tried to make SMO more efficient from an algorithmic point of view. As a result, we have obtained:

- A detailed analysis of the Heavy Ball method and Nesterov's Accelerated Gradient, showing how they improve on the basic Gradient Descent scheme.

- Two novel variants of the SMO algorithm, Conjugate SMO and Nesterov's Accelerated SMO. We have also analyzed the complexity of both algorithms in terms of floating point operations.

- Experimental results showing how, in general, our Nesterov's Accelerated and Conjugate versions reduce the number of iterations needed by SMO to converge to a given precision ϵ. In particular, we have studied the convergence for several values of the hyper-parameters C and γ, using our own Python implementation of the three methods.

- A software implementation of Conjugate SMO inside the LIBSVM framework.

- Experimental results comparing the running time of Conjugate SMO and standard SMO, as implemented by LIBSVM. In these experiments we have settled for a precision $\epsilon = 0.001$ and instead we have varied the cache size and SVC's hyper-parameters C and γ. As a conclusion Conjugate SMO is usually faster the larger the C and γ and the lower the cache size, that is, more difficult problems. This suggest that Conjugate Gradient is beneficial in

an hyper-parametrization setting, since the common practice is to explore a fairly big grid of values for C and γ.

7.2 Further work

This thesis is just a step in obtaining faster and more efficient algorithms to solve the Lasso and SVMs problems. Therefore, we finish our discussion by providing some ideas to extend the work presented here:

- As noted by Jaggi (2014), there are many implications of the equivalence between the Lasso and SVMs. We have exploited here one particular direction to solve the Lasso using essentially SMO. However, it may be profitable to apply in practice other results. As a particular example novel homotopy methods for the SVM could be developed.

- Although the Cyclic Coordinate Descent solver implemented in *scikit-learn* is of high quality, it is still not as fast as the original GLMNet package. The exact reasons for this discrepancy are very hard to know, but the scikit version could at least improve in two areas: implement a cache of recently computed dot products and implement screening rules. Thus, it would be of great interest to add these features to *scikit-learn* and then redo the comparison in Chapter 5.

- Stochastic Coordinate Descent offers an alternative to Cyclic Coordinate Descent that sometimes converges faster when the number of features is very large. Therefore it would also be interesting to include it in the experiments performed in Chapter 5.

- We have already mentioned screening as a potential technique to speed up solving the Lasso problem by discarding early unused coefficients. As we discussed, the effectiveness of the screening rules greatly depends on the problem. Similarly, the shrinking feature in LIBSVM discards coefficients during the optimization that are not going to be support vectors. Although we have disable shrinking when running K-SVM, it has the potential to further reduce training times. Thus, more experiments comparing shrinking and screening are needed.

- LIBSVM was the library chosen to perform the experiments in Chapter 5. However, LIBSVM is the state-of-the-art only for non-linear SVM models. In this case, we transform the Lasso problem into a linear ν-SVC instance and the question remains whether it can be solved more efficiently. As of now other popular libraries such as LIBLINEAR and Pegasos that contain more sophisticated algorithms for the linear case do not include the ν-SVC.

- Regarding the Conjugate and Nesterov's Accelerated SMO, it should be easy to extend them to work for the SVR model. In the same fashion, novel Conjugate and Nesterov Accelerated versions could also be derived for the ν-SVM. However, in this case more work is involved since this problem has an extra constraint.

- Currently, we believe our Monotone Nesterov's Accelerated SMO not to be competitive with either SMO or Conjugate SMO. One of the main reasons is that checking the monotonicity at every iteration involves computing the value of the objective function, which is quite expensive in the case of SVC's dual. However, in a recent work O'Donoghue and Candès (2015) suggest to test the monotonicity by using the gradient and to restart Nesterov's μ_k sequence when it does not hold. Since this information is already needed by SMO, it

could provide a faster convergence for Nesterov's Accelerated version. Furthermore, we also compute the exact coefficient in the convex combination that ensures monotone decrease. This could also be avoiding using the restarts mentioned above.

Chapter 8

Conclusiones

8.1 Discusión

Esta tesis tiene dos partes diferenciadas. En la primera, se ha estudiado el modelo Lasso, propuesto por Tibshirani en 1996. La idea principal es añadir una penalización ℓ_1 a la función de pérdida de mínimos cuadrados, obteniendo no solo un buen error de generalización sino también coeficientes finales dispersos. Bajo este punto de vista Lasso no es únicamente un modelo de regresión sino también un método de selección de características, ya que las variables que se corresponden con coeficientes iguales a cero se pueden eliminar del modelo final. El problema de optimización subyacente tiene dos características principales, ambas relacionadas con la penalización ℓ_1, y que hacen que este sea más complicado de abordar que otros problemas similares, como por ejemplo Ridge Regression:

1. La función objetivo no es fuertemente convexa.
2. La norma ℓ_1 no es diferenciable.

Entre los primeros algoritmos para resolver el Lasso de forma eficiente destaca FISTA, que consiste en la aceleración de Nesterov aplicada a Proximal Gradient Descent. La investigación en algoritmos que escalan a problemas más grandes se resume en dos direcciones principales: Stochastic Coordenate Descent y Stochastic Gradient Descent.

En Stochastic Coordinate Descent se selecciona un coeficiente de forma aleatoria y se optimiza con respecto a esa coordenada usando el gradiente completo. Este algoritmo es especialmente beneficioso para el problema Lasso, ya que la norma ℓ_1 es separable y por tanto el sub-problema de optimización tiene solución analítica. Por otra parte, en Stochastic Gradient Descent se calcula una aproximación del gradiente con respecto a todas las coordenadas pero usando un único ejemplo. En general Coordinate Descent es más rápido en problemas con tamaños muestrales moderadamente grandes, hasta los cientos de miles. Algo que también contribuye a la popularidad de este algoritmo es GLMNet, una implementación en Fortran/R muy eficiente.

Basándonos en un resultado reciente de Jaggi, donde muestra la equivalencia entre el Lasso con restricciones y las SVMs, hemos considerado en esta tesis el uso de SMO como un algoritmo para resolver el problema Lasso, mostrando empíricamente que es por lo menos competitivo. En particular, nuestras contribuciones son:

- Una versión refinada de la equivalencia entre el Lasso y las SVMs. En particular se construye una instancia de un problema geométrico conocido como el Nearest Point Problem, cuyo objetivo es encontrar los puntos más cercanos de dos envolventes convexas. El NPP es

también equivalente a la formulación ν-SVC, y por tanto se puede resolver de forma directa usando LIBSVM.

- Una versión modificada del algoritmo SMO implementado por LIBSVM especialmente diseñado para la reducción anterior. Cuando se resuelve el Lasso a través de la instancia ν-SVC equivalente tenemos que doblar el número de variables, obteniendo una nueva matriz de datos artificial $(\mathbf{X} \,|\, -\mathbf{X})$. Por tanto, cuando se calcula la matriz de núcleos esta contiene muchas redundancias. Aprovechándonos de ese hecho, sugerimos un nuevo algoritmo que tiene en cuenta esta estructura (K-SVM) y resuelve el problema Lasso realizando hasta 8 veces menos operaciones de kernel que la SVM estándar.

- Una comparación empírica entre Cyclic Coordinate Descent, tomando como referencia la implementación de la libreria *scikit-learn*, SVM y K-SVM. Los resultados muestran que K-SVM obtiene una solución ϵ-óptima más rápido que SVM para $\epsilon = 10^{-6}$, como era de esperar. K-SVM también supera a Cyclic Coordinate Descent en todos los conjuntos de datos considerados excepto en uno.

- Implementación en Python de la reducción del Lasso a una instancia de SVM y una implementación de K-SVM dentro de la libreria LIBSVM.

En la segunda parte de esta tesis nos hemos centrado en métodos de aceleración para el algoritmo SMO. En particular hemos desarrollado dos variantes de SMO, una basada en el algoritmo Conjugate Gradient Descent y otra inspirada por el gradiente acelerado de Nesterov. Ambas técnicas provienen de la literatura clásica de optimización convexa, y son conocidas desde 1963 y 1983, respectivamente. Sin embargo recientemente estos algoritmos, sobre todo el gradiente acelerado de Nesterov, han sido aplicados a modelos actuales de Aprendizaje Automático como por ejemplo redes neuronales profundas. Esto se debe en parte a todo el trabajo teórico que se está realizando para tratar de comprender por qué y cuándo la aceleración es posible.

Sequential Minimal Optimization fue propuesto por Platt en 1998 y, desde entonces, es uno de los algoritmos más usados para resolver las SVMs, principalmente su formulación no lineal. Esto se debe, parcialmente, a la libreria LIBSVM, que fue creada en 2001 y ofrece una implementación muy eficiente. LIBSVM recibe actualizaciones periódicas pero, aparte de correcciones menores y optimizaciones del código, el algoritmo es esencialmente SMO con selección de coeficientes de segundo orden (Fan et al., 2005b). Además, los avances recientes en la optimización del problema de las SMVs se centran en escalar los algoritmos usando paralelización, computación distribuida o ambas. Motivados por estas ideas, nosotros hemos considerado en esta tesis un enfoque distinto. En lugar de centrarnos en los avances tecnológicos, hemos intentado hacer que SMO sea más eficiente desde un punto de vista algorítmico. Como resultado, hemos obtenido:

- Un análisis detallado del método Heavy Ball y el gradiente acelerado de Nesterov, mostrando como mejoran el esquema básico de descenso por gradiente.

- Dos nuevas variantes del algorimo SMO, Conjugate SMO y Nesterov's Accelerated SMO. También hemos analizado la complejidad de ambos algoritmos en términos de operaciones de coma flotante.

- Resultados experimentales mostrando como, en general, los algoritmos Nesterov's Accelerated y Conjugate reducen el número de iteraciones de SMO necesarias para la convergencia con una precisión ϵ determinada. En particular, hemos estudiado la convergencia para múltiples valores de los hiper-parámetros C y γ, usando una implementación propia de los tres métodos en Python.

- Una implementación de Conjugate SMO en la librería LIBSVM.
- Resultados experimentales comparando el tiempo de ejecución de Conjugate SMO y SMO estándar, ambos implementados en LIBSVM. En estos experimentos hemos fijado una precisión $\epsilon = 0.001$ y hemos explorado en su lugar distintos valores para el tamaño de la cache y de los hiper-parámetros C y γ. La conclusión es que Conjugate SMO es más rápido cuanto más grande sean C y γ y cuanto más pequeño sea el tamaño de la cache, es decir, problemas de optimización más complejos. Esto sugiere que Conjugate SMO es beneficioso en un esquema de hiper-parametrización, ya que la práctica común consiste en explorar una rejilla más o menos grande de valores de C y γ.

8.2 Trabajo futuro

Esta tesis es solo un pequeño paso en el camino de obtener algorimos más rápidos y eficientes para resolver el Lasso y las SVMs. Por tanto, terminamos nuestra discusión proporcionando algunas ideas para extender el el trabajo aquí presentado:

- Como ya indicó Jaggi (2014), hay múltiples implicaciones de la equivalencia entre el Lasso y las SVMs. En esta tesis nos hemos aprovechado de una dirección en particular para resolver el Lasso usando SMO. Sin embargo, también puede ser beneficioso aplicar en la práctica otros resultados. Un ejemplo concreto podría ser desarrollar métodos de homotopía para las SVMs.
- A pesar de que el algoritmo Cyclic Coordinate Descent implementado en *scikit-learn* es de mucha calidad, todavía no es tan rápido como el paquete GLMNet original. Es complicado conocer las razones exactas de esta discrepancia, pero la versión de *scikit-learn* podría mejorar por lo menos en dos áreas: implementar una cache de productos escalares recientes e implementar reglas de *screening*. Por tanto, sería de gran interés añadir estas características a *scikit-learn* y rehacer la comparación en el Capítulo 5.
- Stochastic Coordinate Descent ofrece una alternativa a Cyclic Coordinate Descent que en ocasiones converge más rápido cuando el número de variables es muy grande. Por tanto también sería interesante incluir este algoritmo en los experimentos realizados en el Capítulo 5.
- Ya hemos mencionado el *screening* como una técnica que potencialmente puede acelerar la resolución del problema Lasso descartando coeficientes antes de que termine la optimización. Como ya hemos mencionado anteriormente, la efectividad de las reglas de *screening* depende en gran parte del problema en concreto. De forma similar, la función de *shrinking* de LIBSVM descarta coeficientes durante el proceso de optimización que no van a ser vectores de soporte. A pesar de que hemos deshabilitado *shrinking* en la realización los experimentos de K-SVM, existe la posibilidad de que reduzca todavía más el tiempo de ejecución. Por tanto, son necesarios más experimentos teniendo en cuenta tanto *shrinking* como *screening*.
- Hemos elegido la librería LIBSVM para realizar los experimentos del Capítulo 5. Sin embargo LIBSVM es únicamente el estado del arte para resolver SVMs no lineales. En este caso, Lasso se transforma en una instancia de la ν-SVC lineal y por tanto existe la duda de si este problema se puede resolver de manera más eficiente usando otro algoritmo distinto de SMO. Actualmente otras librerías populares como LIBLINEAR y Pegasos, que contienen algoritmos más sofisticados para resolver la formulación lineal, no incluyen la ν-SVC.

- Con respecto a Conjugate y Nesterov's Accelerated SMO, deberían de ser fácilmente aplicables al la SVR. De la misma forma, se podrían desarrollar nuevas versiones Conjugate y Nesterov' Accelerated para la ν-SVM. En este caso sería algo más complicado ya que el problema de optimización tiene una restricción extra.
- En este momento no consideramos que nuestra versión Monotone Nesterov's Accelerated SMO sea competitiva con SMO o Conjugate SMO. Uno de los motivos principales es que comprobar la monotonocidad conlleva calcular el valor de la función objetivo, que es muy costoso en el caso del dual de la SVC. Sin embargo, un trabajo reciente de O'Donoghue and Candès (2015) propone comprobar la monotonicidad usando el gradiente y reiniciar la secuencia de Nesterov μ_k cuando no se cumpla. Puesto que esta información ya es necesaria en SMO, podría proporcionar una convergencia más rápida para la versión Nesterov's Accelerated. Además también calculamos el valor exacto del coeficiente que controla la combinación convexa y asegura el descenso monótono. Esto también se podría evitar usando los reinicios mencionados anteriormente.

Appendices

Appendix A

Derivation of the soft-thresholding operator

Let $f(x) = \lambda\|x\|_1$, then the proximal mapping of f is defined as

$$\text{prox}_f(x) = \underset{z}{\text{argmin}}\{\|x - z\|_2^2 + \lambda\|z\|_1\}. \tag{A.1}$$

The optimality condition for the previous problem is (Proposition 2.2)

$$0 \in \nabla(\|x - z\|_2^2) + \partial(\lambda\|z\|_1) \;\Leftrightarrow\; 0 \in z - x + \lambda\partial\|z\|_1 \tag{A.2}$$

The l_1-norm is additive and thus we can consider each of its components separately. Let's examine first the case where $z_i \neq 0$. Then, $\partial\|z_i\| = \text{sign}(z_i)$ and the optimum z_i^* is obtained as

$$0 = z_i - x_i + \lambda\,\text{sign}(z_i) \;\Leftrightarrow\; z_i^* = x_i - \lambda\,\text{sign}(z_i^*) \tag{A.3}$$

Note also that if $z_i^* < 0$, then $x_i < -\lambda$ and equivalently if $z_i^* > 0 \Rightarrow x_i > \lambda$. Thus $\text{sign}(z_i^*) = \text{sign}(x_i)$ and $|x_i| > \lambda$. Substituting in Eq. (A.3) we get

$$z_i^* = x_i - \lambda\,\text{sign}(x_i). \tag{A.4}$$

In the case where $z_i = 0$, the subdifferential of the l_1-norm is the interval $[-1, 1]$ and the optimality condition is

$$0 \in -x_i + \lambda[-1, 1] \Leftrightarrow x_i \in [-\lambda, \lambda] \Leftrightarrow |x_i| \leq \lambda. \tag{A.5}$$

Putting all together we get

$$z_i^* = [\text{prox}_f(x)]_i = \begin{cases} 0 & \text{if } |x_i| \leq \lambda, \\ x_i - \lambda\,\text{sign}(x_i) & \text{if } |x_i| > \lambda. \end{cases} \tag{A.6}$$

Equation (A.6) can also be written as

$$\begin{aligned} [\text{prox}_f(x)]_i &= \text{sign}(x_i)\max(|x_i| - \lambda, 0) \\ &= \text{sign}(x_i)(|x_i| - \lambda)_+ \\ &= \frac{x_i}{|x_i|}(|x_i| - \lambda)_+ \\ &= x_i\left(1 - \frac{\lambda}{|x_i|}\right)_+ \end{aligned}$$

where $(\cdot)_+$ denotes the *positive part*.

Appendix B

Convergence rates

Suppose that $\{\mathbf{x}^k\}$ is a sequence of points in $\mathbb{R}^n$. The sequence converges to some point $\mathbf{x}^*$, written

$$\lim_{k\to\infty} \mathbf{x}^k = \mathbf{x}^*$$

if for any $\epsilon > 0$, there is an index k_0 such that

$$\|\mathbf{x}^k - \mathbf{x}^*\| \leq \epsilon,$$

for all $k > k_0$ (Nocedal and Wright, 2006).

Given two different algorithms that generate a sequence $\{\mathbf{x}^k\}$ approaching to the same limit, it is useful to characterize which one arrives to the limit faster. The speed at which a convergent sequence approaches the limit is called **rate of convergence**. Assume that the sequence $\{\mathbf{x}^k\}$ converges to $\mathbf{x}^*$. Then, we say that the sequence converges *linearly* to $\mathbf{x}^*$ if there exists a constant $r \in (0,\, 1)$ such that

$$\lim_{k\to\infty} \frac{\|\mathbf{x}^{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}^k - \mathbf{x}^*\|} = r \tag{B.1}$$

for all $k > k_0$. The number r is called the rate of convergence and it means that the distance to $\mathbf{x}^*$ decreases at each iteration by at least that factor,

$$\frac{\|\mathbf{x}^{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}^k - \mathbf{x}^*\|} \leq r.$$

Furthermore, if $r = 0$ we say that the sequence converges *superlinearly* and if $r = 1$ it converges *sublinearly*.

We can also distinguish different cases of superlinear convergence. A sequence $\{\mathbf{x}^k\}$ converges *superlinearly with order* q if there exists a constant $r \in (0,\, 1)$ such that

$$\lim_{k\to\infty} \frac{\|\mathbf{x}^{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}^k - \mathbf{x}^*\|} = r \tag{B.2}$$

for $k > k_0$. In particular, if $q = 2$ is called quadratic convergence, if $q = 3$ cubic convergence and so on. Figure B.1 shows the differences between sublinear, linear and quadratic convergences. Note that a linear convergence is often better than sublinear convergence but not always. For instance in this example, if we were to choose $\epsilon = 10^{-3}$ the sequence $1/k^2$ would have converge faster than r^k for $r = 0.85$, but maybe not for other rates.

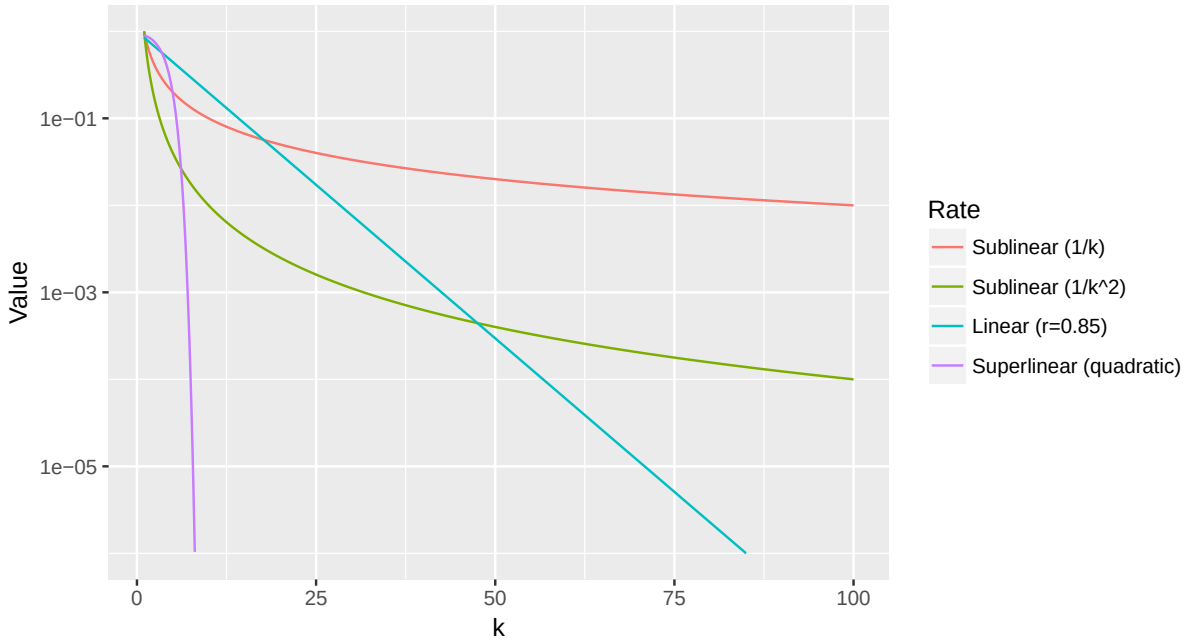

Figure B.1: Comparison of sublinear ($1/k$ and $1/k^2$), linear and superlinear (quadratic) convergences.

Appendix C

Iteration results

We include here for completeness the iteration results of the experiments in 6.3.4. Note that these do not necessarily coincide exactly with the ones in 6.3.3, since those were obtained using Python code. Thus, even though they implement the same algorithm there are many subtleties that may slightly change the results. One such example is the stopping criterion.

Table C.1: Comparison of the number of iterations between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `adult8` dataset

			Iterations			Ratio	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	6 250	6 251	6 251	1.00	1.00
		−1	5 786	5 774	5 774	1.00	1.00
		0	5 968	5 748	5 731	1.04	1.04
		1	12 746	10 939	11 240	1.17	1.13
	10	−2	5 779	5 788	5 788	1.00	1.00
		−1	6 168	5 923	5 823	1.04	1.06
		0	15 460	12 075	12 308	1.28	1.26
		1	65 787	51 253	55 034	1.28	1.20
	100	−2	6 136	5 830	5 863	1.05	1.05
		−1	14 886	11 110	11 997	1.34	1.24
		0	117 776	72 499	82 834	1.62	1.42
		1	269 486	191 220	215 867	1.41	1.25
	1 000	−2	16 341	12 761	12 795	1.28	1.28
		−1	110 050	66 836	75 228	1.65	1.46
		0	941 352	489 477	612 510	1.92	1.54
		1	661 900	397 725	486 109	1.66	1.36
100	1	−2	6 250	6 251	6 251	1.00	1.00
		−1	5 786	5 774	5 774	1.00	1.00
		0	5 968	5 748	5 731	1.04	1.04
		1	12 746	10 939	11 240	1.17	1.13
	10	−2	5 779	5 788	5 788	1.00	1.00
		−1	6 168	5 923	5 823	1.04	1.06
		0	15 460	12 075	12 308	1.28	1.26
		1	65 787	51 253	55 034	1.28	1.20
	100	−2	6 136	5 830	5 863	1.05	1.05
		−1	14 886	11 110	11 997	1.34	1.24
		0	117 776	72 499	82 834	1.62	1.42
		1	269 486	191 220	215 867	1.41	1.25
	1 000	−2	16 341	12 761	12 795	1.28	1.28
		−1	110 050	66 836	75 228	1.65	1.46
		0	941 352	489 477	612 510	1.92	1.54
		1	661 900	397 725	486 109	1.66	1.36
1 000	1	−2	6 250	6 251	6 251	1.00	1.00
		−1	5 786	5 774	5 774	1.00	1.00
		0	5 968	5 748	5 731	1.04	1.04
		1	12 746	10 939	11 240	1.17	1.13
	10	−2	5 779	5 788	5 788	1.00	1.00
		−1	6 168	5 923	5 823	1.04	1.06
		0	15 460	12 075	12 308	1.28	1.26
		1	65 787	51 253	55 034	1.28	1.20
	100	−2	6 136	5 830	5 863	1.05	1.05
		−1	14 886	11 110	11 997	1.34	1.24
		0	117 776	72 499	82 834	1.62	1.42
		1	269 486	191 220	215 867	1.41	1.25
	1 000	−2	16 341	12 761	12 795	1.28	1.28
		−1	110 050	66 836	75 228	1.65	1.46
		0	941 352	489 477	612 510	1.92	1.54
		1	661 900	397 725	486 109	1.66	1.36

Table C.2: Comparison of the number of iterations between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `web8` dataset

			Iterations			Ratio	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	2 449	2 501	2 501	0.98	0.98
		−1	2 679	2 643	2 643	1.01	1.01
		0	3 089	2 976	2 893	1.04	1.07
		1	5 378	5 041	5 092	1.07	1.06
	10	−2	2 637	2 643	2 643	1.00	1.00
		−1	3 144	2 960	3 018	1.06	1.04
		0	7 759	7 164	7 009	1.08	1.11
		1	16 344	12 997	13 345	1.26	1.22
	100	−2	3 027	2 921	3 001	1.04	1.01
		−1	7 147	6 887	6 831	1.04	1.05
		0	32 556	26 909	28 168	1.21	1.16
		1	28 525	20 717	22 262	1.38	1.28
	1 000	−2	8 846	8 063	7 670	1.10	1.15
		−1	41 168	32 089	34 422	1.28	1.20
		0	124 013	92 767	97 653	1.34	1.27
		1	45 755	26 271	30 442	1.74	1.50
100	1	−2	2 449	2 501	2 501	0.98	0.98
		−1	2 679	2 643	2 643	1.01	1.01
		0	3 089	2 976	2 893	1.04	1.07
		1	5 378	5 041	5 092	1.07	1.06
	10	−2	2 637	2 643	2 643	1.00	1.00
		−1	3 144	2 960	3 018	1.06	1.04
		0	7 759	7 164	7 009	1.08	1.11
		1	16 344	12 997	13 345	1.26	1.22
	100	−2	3 027	2 921	3 001	1.04	1.01
		−1	7 147	6 887	6 831	1.04	1.05
		0	32 556	26 909	28 168	1.21	1.16
		1	28 525	20 717	22 262	1.38	1.28
	1 000	−2	8 846	8 063	7 670	1.10	1.15
		−1	41 168	32 089	34 422	1.28	1.20
		0	124 013	92 767	97 653	1.34	1.27
		1	45 755	26 271	30 442	1.74	1.50
1 000	1	−2	2 449	2 501	2 501	0.98	0.98
		−1	2 679	2 643	2 643	1.01	1.01
		0	3 089	2 976	2 893	1.04	1.07
		1	5 378	5 041	5 092	1.07	1.06
	10	−2	2 637	2 643	2 643	1.00	1.00
		−1	3 144	2 960	3 018	1.06	1.04
		0	7 759	7 164	7 009	1.08	1.11
		1	16 344	12 997	13 345	1.26	1.22
	100	−2	3 027	2 921	3 001	1.04	1.01
		−1	7 147	6 887	6 831	1.04	1.05
		0	32 556	26 909	28 168	1.21	1.16
		1	28 525	20 717	22 262	1.38	1.28
	1 000	−2	8 846	8 063	7 670	1.10	1.15
		−1	41 168	32 089	34 422	1.28	1.20
		0	124 013	92 767	97 653	1.34	1.27
		1	45 755	26 271	30 442	1.74	1.50

Table C.3: Comparison of the number of iterations between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `ijcnn1` dataset

			Iterations			Ratio	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	9 006	9 006	9 006	1.00	1.00
		−1	8 402	8 369	8 369	1.00	1.00
		0	6 758	6 669	6 610	1.01	1.02
		1	5 657	5 386	5 341	1.05	1.06
	10	−2	8 509	8 495	8 495	1.00	1.00
		−1	7 611	7 399	7 376	1.03	1.03
		0	7 508	6 717	6 756	1.12	1.11
		1	11 146	8 647	9 062	1.29	1.23
	100	−2	7 857	7 686	7 768	1.02	1.01
		−1	10 651	8 846	8 786	1.20	1.21
		0	19 427	13 794	14 951	1.41	1.30
		1	48 591	32 657	37 825	1.49	1.28
	1 000	−2	14 105	10 621	11 144	1.33	1.27
		−1	34 956	21 978	24 741	1.59	1.41
		0	129 905	82 096	93 399	1.58	1.39
		1	238 126	161 662	183 527	1.47	1.30
100	1	−2	9 006	9 006	9 006	1.00	1.00
		−1	8 402	8 369	8 369	1.00	1.00
		0	6 758	6 669	6 610	1.01	1.02
		1	5 657	5 386	5 341	1.05	1.06
	10	−2	8 509	8 495	8 495	1.00	1.00
		−1	7 611	7 399	7 376	1.03	1.03
		0	7 508	6 717	6 756	1.12	1.11
		1	11 146	8 647	9 062	1.29	1.23
	100	−2	7 857	7 686	7 768	1.02	1.01
		−1	10 651	8 846	8 786	1.20	1.21
		0	19 427	13 794	14 951	1.41	1.30
		1	48 591	32 657	37 825	1.49	1.28
	1 000	−2	14 105	10 621	11 144	1.33	1.27
		−1	34 956	21 978	24 741	1.59	1.41
		0	129 905	82 096	93 399	1.58	1.39
		1	238 126	161 662	183 527	1.47	1.30
1 000	1	−2	9 006	9 006	9 006	1.00	1.00
		−1	8 402	8 369	8 369	1.00	1.00
		0	6 758	6 669	6 610	1.01	1.02
		1	5 657	5 386	5 341	1.05	1.06
	10	−2	8 509	8 495	8 495	1.00	1.00
		−1	7 611	7 399	7 376	1.03	1.03
		0	7 508	6 717	6 756	1.12	1.11
		1	11 146	8 647	9 062	1.29	1.23
	100	−2	7 857	7 686	7 768	1.02	1.01
		−1	10 651	8 846	8 786	1.20	1.21
		0	19 427	13 794	14 951	1.41	1.30
		1	48 591	32 657	37 825	1.49	1.28
	1 000	−2	14 105	10 621	11 144	1.33	1.27
		−1	34 956	21 978	24 741	1.59	1.41
		0	129 905	82 096	93 399	1.58	1.39
		1	238 126	161 662	183 527	1.47	1.30

Table C.4: Comparison of the number of iterations between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `cod-rna` dataset

			Iterations			Ratio	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	42 950	27 177	29 578	1.58	1.45
		−1	61 078	38 103	38 938	1.60	1.57
		0	91 229	54 369	55 372	1.68	1.65
		1	37 381	35 850	36 107	1.04	1.04
	10	−2	170 421	77 285	82 945	2.21	2.05
		−1	261 206	119 248	127 991	2.19	2.04
		0	241 711	115 265	126 190	2.10	1.92
		1	43 026	40 150	40 563	1.07	1.06
	100	−2	749 551	298 121	313 889	2.51	2.39
		−1	1 003 438	427 601	450 801	2.35	2.23
		0	707 542	285 249	364 116	2.48	1.94
		1	60 297	52 595	54 480	1.15	1.11
	1 000	−2	4 419 404	1 633 647	1 768 231	2.71	2.50
		−1	4 696 452	1 806 054	2 132 620	2.60	2.20
		0	2 508 529	948 047	1 315 501	2.65	1.91
		1	105 650	74 924	85 085	1.41	1.24
100	1	−2	42 950	27 177	29 578	1.58	1.45
		−1	61 078	38 103	38 938	1.60	1.57
		0	91 229	54 369	55 372	1.68	1.65
		1	37 381	35 850	36 107	1.04	1.04
	10	−2	170 421	77 285	82 945	2.21	2.05
		−1	261 206	119 248	127 991	2.19	2.04
		0	241 711	115 265	126 190	2.10	1.92
		1	43 026	40 150	40 563	1.07	1.06
	100	−2	749 551	298 121	313 889	2.51	2.39
		−1	1 003 438	427 601	450 801	2.35	2.23
		0	707 542	285 249	364 116	2.48	1.94
		1	60 297	52 595	54 480	1.15	1.11
	1 000	−2	4 419 404	1 633 647	1 768 231	2.71	2.50
		−1	4 696 452	1 806 054	2 132 620	2.60	2.20
		0	2 508 529	948 047	1 315 501	2.65	1.91
		1	105 650	74 924	85 085	1.41	1.24
1 000	1	−2	42 950	27 177	29 578	1.58	1.45
		−1	61 078	38 103	38 938	1.60	1.57
		0	91 229	54 369	55 372	1.68	1.65
		1	37 381	35 850	36 107	1.04	1.04
	10	−2	170 421	77 285	82 945	2.21	2.05
		−1	261 206	119 248	127 991	2.19	2.04
		0	241 711	115 265	126 190	2.10	1.92
		1	43 026	40 150	40 563	1.07	1.06
	100	−2	749 551	298 121	313 889	2.51	2.39
		−1	1 003 438	427 601	450 801	2.35	2.23
		0	707 542	285 249	364 116	2.48	1.94
		1	60 297	52 595	54 480	1.15	1.11
	1 000	−2	4 419 404	1 633 647	1 768 231	2.71	2.50
		−1	4 696 452	1 806 054	2 132 620	2.60	2.20
		0	2 508 529	948 047	1 315 501	2.65	1.91
		1	105 650	74 924	85 085	1.41	1.24

Table C.5: Comparison of the number of iterations between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `mnist1` dataset

			Iterations			Ratio	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	4 479	4 479	4 479	1.00	1.00
		−1	2 137	2 108	2 108	1.01	1.01
		0	1 761	1 549	1 619	1.14	1.09
		1	3 058	2 914	2 934	1.05	1.04
	10	−2	2 119	2 093	2 093	1.01	1.01
		−1	2 319	2 019	2 093	1.15	1.11
		0	5 848	4 631	4 890	1.26	1.20
		1	4 043	3 867	3 845	1.05	1.05
	100	−2	2 484	2 122	2 175	1.17	1.14
		−1	11 287	9 098	9 087	1.24	1.24
		0	11 516	8 808	9 555	1.31	1.21
		1	4 043	3 867	3 845	1.05	1.05
	1 000	−2	13 075	9 762	10 446	1.34	1.25
		−1	49 430	34 627	38 456	1.43	1.29
		0	12 303	9 084	9 941	1.35	1.24
		1	4 043	3 867	3 845	1.05	1.05
100	1	−2	4 479	4 479	4 479	1.00	1.00
		−1	2 137	2 108	2 108	1.01	1.01
		0	1 761	1 549	1 619	1.14	1.09
		1	3 058	2 914	2 934	1.05	1.04
	10	−2	2 119	2 093	2 093	1.01	1.01
		−1	2 319	2 019	2 093	1.15	1.11
		0	5 848	4 631	4 890	1.26	1.20
		1	4 043	3 867	3 845	1.05	1.05
	100	−2	2 484	2 122	2 175	1.17	1.14
		−1	11 287	9 098	9 087	1.24	1.24
		0	11 516	8 808	9 555	1.31	1.21
		1	4 043	3 867	3 845	1.05	1.05
	1 000	−2	13 075	9 762	10 446	1.34	1.25
		−1	49 430	34 627	38 456	1.43	1.29
		0	12 303	9 084	9 941	1.35	1.24
		1	4 043	3 867	3 845	1.05	1.05
1 000	1	−2	4 479	4 479	4 479	1.00	1.00
		−1	2 137	2 108	2 108	1.01	1.01
		0	1 761	1 549	1 619	1.14	1.09
		1	3 058	2 914	2 934	1.05	1.04
	10	−2	2 119	2 093	2 093	1.01	1.01
		−1	2 319	2 019	2 093	1.15	1.11
		0	5 848	4 631	4 890	1.26	1.20
		1	4 043	3 867	3 845	1.05	1.05
	100	−2	2 484	2 122	2 175	1.17	1.14
		−1	11 287	9 098	9 087	1.24	1.24
		0	11 516	8 808	9 555	1.31	1.21
		1	4 043	3 867	3 845	1.05	1.05
	1 000	−2	13 075	9 762	10 446	1.34	1.25
		−1	49 430	34 627	38 456	1.43	1.29
		0	12 303	9 084	9 941	1.35	1.24
		1	4 043	3 867	3 845	1.05	1.05

Table C.6: Comparison of the number of iterations between SMO, Conjugate SMO (CS) and Hybrid SMO (HS) for the `skin` dataset

			Iterations			Ratio	
Cache (Mb)	C	$\log_{10}\gamma$	SMO	CS	HS	CS	HS
1	1	−2	4 911	4 252	4 225	1.15	1.16
		−1	50 717	41 502	41 674	1.22	1.22
		0	137 848	121 036	122 190	1.14	1.13
		1	102 095	102 051	102 095	1.00	1.00
	10	−2	5 992	4 373	4 641	1.37	1.29
		−1	49 967	41 127	41 411	1.21	1.21
		0	140 022	122 164	123 916	1.15	1.13
		1	137 600	137 310	137 599	1.00	1.00
	100	−2	17 879	10 486	11 005	1.71	1.62
		−1	48 983	39 846	40 284	1.23	1.22
		0	140 197	122 229	123 949	1.15	1.13
		1	137 599	137 297	137 599	1.00	1.00
	1 000	−2	90 214	39 705	50 058	2.27	1.80
		−1	48 401	40 069	40 395	1.21	1.20
		0	139 763	122 435	123 992	1.14	1.13
		1	137 600	137 297	137 599	1.00	1.00
100	1	−2	4 911	4 252	4 225	1.15	1.16
		−1	50 717	41 502	41 674	1.22	1.22
		0	137 848	121 036	122 190	1.14	1.13
		1	102 095	102 051	102 095	1.00	1.00
	10	−2	5 992	4 373	4 641	1.37	1.29
		−1	49 967	41 127	41 411	1.21	1.21
		0	140 022	122 164	123 916	1.15	1.13
		1	137 600	137 310	137 599	1.00	1.00
	100	−2	17 879	10 486	11 005	1.71	1.62
		−1	48 983	39 846	40 284	1.23	1.22
		0	140 197	122 229	123 949	1.15	1.13
		1	137 599	137 297	137 599	1.00	1.00
	1 000	−2	90 214	39 705	50 058	2.27	1.80
		−1	48 401	40 069	40 395	1.21	1.20
		0	139 763	122 435	123 992	1.14	1.13
		1	137 600	137 297	137 599	1.00	1.00
1 000	1	−2	4 911	4 252	4 225	1.15	1.16
		−1	50 717	41 502	41 674	1.22	1.22
		0	137 848	121 036	122 190	1.14	1.13
		1	102 095	102 051	102 095	1.00	1.00
	10	−2	5 992	4 373	4 641	1.37	1.29
		−1	49 967	41 127	41 411	1.21	1.21
		0	140 022	122 164	123 916	1.15	1.13
		1	137 600	137 310	137 599	1.00	1.00
	100	−2	17 879	10 486	11 005	1.71	1.62
		−1	48 983	39 846	40 284	1.23	1.22
		0	140 197	122 229	123 949	1.15	1.13
		1	137 599	137 297	137 599	1.00	1.00
	1 000	−2	90 214	39 705	50 058	2.27	1.80
		−1	48 401	40 069	40 395	1.21	1.20
		0	139 763	122 435	123 992	1.14	1.13
		1	137 600	137 297	137 599	1.00	1.00

Appendix D

Published papers

Alaíz, C., Torres-Barrán, A., and Dorronsoro, J. R. "nu-SVM Solutions of Constrained Lasso and Elastic Net". Manuscript submitted for publication.

Alaíz, C., Torres, A., and Dorronsoro, J. (2012). "Sparse linear wind farm energy forecast". In: *Artificial Neural Networks and Machine Learning–ICANN 2012*, pp. 557–564.

Alaíz, C., Torres, A., and Dorronsoro, J. R. (2015). "Solving Constrained Lasso and Elastic Net Using ν–SVMs". In: *Proceedings of ESANN 2015, Bruges, Belgium, 22-24 April 2015*, pp. 1382–1390.

Alonso, Á., Torres, A., and Dorronsoro, J. R. (2015). "Random Forests and Gradient Boosting for Wind Energy Prediction". In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer, pp. 26–37.

Azegrouz, H., Karemore, G., Torres, A., Alaíz, C. M., Gonzalez, A. M., Nevado, P., Salmerón, A., Pellinen, T., Pozo, M. A. del, Dorronsoro, J. R., and Montoya, M. C. (2013). "Cell-based fuzzy metrics enhance high-content screening (HCS) assay robustness". In: *Journal of Biomolecular Screening* 18.10, pp. 1270–1283.

Catalina, A., Torres-Barrán, A., and Dorronsoro, J. R. (2016). "Machine Learning Prediction of Photovoltaic Energy from Satellite Sources". In: *ECML/PKDD 4th International Workshop on Data Analytics for Renewable Energy Integration (DARE)*.

Catalina, A., Torres-Barrán, A., and Dorronsoro, J. R. (2017). "Satellite Based Nowcasting of PV Energy over Peninsular Spain". In: *International Work-Conference on Artificial Neural Networks*. To appear. Springer.

Díaz-Vico, D., Omari, A., Torres-Barrán, A., and Dorronsoro, J. R. (2017a). "Deep Fisher Discriminant Analysis". In: *International Work-Conference on Artificial Neural Networks*. To appear. Springer.

Díaz-Vico, D., Torres-Barrán, A., Omari, A., and Dorronsoro, J. R. (2017b). "Deep Neural Networks for Wind and Solar Energy Prediction". In: *Neural Processing Letters*, pp. 1–16.

Díaz, D., Torres, A., and Dorronsoro, J. R. (2015). "Deep Neural Networks for Wind Energy Prediction". In: *International Work-Conference on Artificial Neural Networks*. Springer, pp. 430–443.

Torres-Barrán, A. and Dorronsoro, J. R. (2015). "Conjugate Descent for the Minimum Norm Problem". In: *NIPS Workshop on Optimization for Machine Learning (OPT)*.

Torres-Barrán, A. and Dorronsoro, J. R. (2016a). "Conjugate descent for the SMO algorithm". In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 3817–3824.

Torres-Barrán, A. and Dorronsoro, J. R. (2016b). "Nesterov Acceleration for the SMO Algorithm". In: *International Conference on Artificial Neural Networks*. Springer, pp. 243–250.

Torres-Barrán, A., Alonso, Á., and Dorronsoro, J. R. (2017). "Regression Tree Ensembles for Wind Energy and Solar Radiation Prediction". In: *Neurocomputing*. In press.

Torres, A., Prada, J., and Dorronsoro, J. R. (2014a). "Nowcasting Meteorological Readings for Wind Energy Prediction". In: *EWEA 2014*.

Torres, A., Díaz, D., and Dorronsoro, J. R. (2014b). "Sparse one hidden layer MLPs." In: *Proceedings of ESANN 2014, Bruges, Belgium, 23-25 April 2014*, pp. 655–660.

Bibliography

Alaíz, C., Torres-Barrán, A., and Dorronsoro, J. R. "nu-SVM Solutions of Constrained Lasso and Elastic Net". Manuscript submitted for publication.

Alaíz, C., Torres, A., and Dorronsoro, J. (2012). "Sparse linear wind farm energy forecast". In: *Artificial Neural Networks and Machine Learning–ICANN 2012*, pp. 557–564.

Alaíz, C., Torres, A., and Dorronsoro, J. R. (2015). "Solving Constrained Lasso and Elastic Net Using ν–SVMs". In: *Proceedings of ESANN 2015, Bruges, Belgium, 22-24 April 2015*, pp. 1382–1390.

Allen-Zhu, Z. and Orecchia, L. (2014). "Linear coupling: An ultimate unification of gradient and mirror descent". In: *Arxiv* 1407.1537.

Allen-Zhu, Z., Richtárik, P., Qu, Z., and Yuan, Y. (2016). "Even faster accelerated coordinate descent using non-uniform sampling". In: *Proceedings of the 33rd International Conference on Machine Learning*. Vol. 48. ICML'16. JMLR.org.

Alonso, Á., Torres, A., and Dorronsoro, J. R. (2015). "Random Forests and Gradient Boosting for Wind Energy Prediction". In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer, pp. 26–37.

Arjevani, Y., Shalev-Shwartz, S., and Shamir, O. (2015). "On lower and upper bounds for smooth and strongly convex optimization problems". In: *arXiv preprint arXiv:1503.06833*.

Azegrouz, H., Karemore, G., Torres, A., Alaíz, C. M., Gonzalez, A. M., Nevado, P., Salmerón, A., Pellinen, T., Pozo, M. A. del, Dorronsoro, J. R., and Montoya, M. C. (2013). "Cell-based fuzzy metrics enhance high-content screening (HCS) assay robustness". In: *Journal of Biomolecular Screening* 18.10, pp. 1270–1283.

Balder, E. J. (2008). "On subdifferential calculus". handout. Available as `http://www.staff.science.uu.nl/~balde101/cao10/cursus10_1.pdf`.

Bauschke, H. H. and Combettes, P. L. (2011). *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, p. 468. ISBN: 978-1-4419-9466-0.

Beck, A. and Teboulle, M. (2009a). "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems". In: *SIAM J. Img. Sci.* 2.1, pp. 183–202. ISSN: 1936-4954.

Beck, A. and Teboulle, M. (2009b). "Fast Gradient-Based Algorithms for Constrained Total Variation Image Denoising and Deblurring Problems". In: *Image Processing and IEEE Transactions on* 18.11, pp. 2419–2434. ISSN: 1057-7149.

Ben-Tal, A. and Nemirovski, A. (2001). *Lectures on modern convex optimization: analysis and algorithms and and engineering applications*. Philadelphia, PA, and USA: Society for Industrial and Applied Mathematics. ISBN: 0-89871-491-5.

Bergstra, J. and Bengio, Y. (2012). "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13, pp. 281–305.

Bishop, C. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1st ed. 2006. Corr. 2nd printing 2011. Springer. ISBN: 0387310738.

Bonnefoy, A., Emiya, V., Ralaivola, L., and Gribonval, R. (2015). "Dynamic screening: Accelerating first-order algorithms for the lasso and group-lasso". In: *IEEE Transactions on Signal Processing* 63.19, pp. 5121–5132.

Bottou, L. (2010). "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, pp. 177–186.

Bottou, L. and Lin, C.-J. (2007). "Support vector machine solvers". In: *Large scale kernel machines*, pp. 301–320.

Bousquet, O. and Bottou, L. (2008). "The tradeoffs of large scale learning". In: *Advances in neural information processing systems*, pp. 161–168.

Boyd, S. and Vandenberghe, L. (2008). *Subgradients*. Lecture notes for EE364b, Winter 2006-07.

Bradley, J. K., Kyrola, A., Bickson, D., and Guestrin, C. (2011). "Parallel coordinate descent for l1-regularized loss minimization". In: ICML'11.

Breiman, L. (1995). "Better Subset Regression Using the Nonnegative Garrote". English. In: *Technometrics* 37.4, pp. 373–384. ISSN: 00401706.

Bubeck, S. (2015). "Convex optimization: Algorithms and complexity". In: *Foundations and Trends® in Machine Learning* 8.3-4, pp. 231–357.

Bubeck, S., Lee, Y. T., and Singh, M. (2015). "A geometric alternative to Nesterov's accelerated gradient descent". In: *Arxiv* 1506.08187.

Catalina, A., Torres-Barrán, A., and Dorronsoro, J. R. (2016). "Machine Learning Prediction of Photovoltaic Energy from Satellite Sources". In: *ECML/PKDD 4th International Workshop on Data Analytics for Renewable Energy Integration (DARE)*.

Catalina, A., Torres-Barrán, A., and Dorronsoro, J. R. (2017). "Satellite Based Nowcasting of PV Energy over Peninsular Spain". In: *International Work-Conference on Artificial Neural Networks*. To appear. Springer.

Chambolle, A. and Dossal, C. (2014). "How to make sure the iterates of FISTA converge".

Chang, C.-C. and Lin, C.-J. (2011). "LIBSVM: a Library for Support Vector Machines". In: *ACM Trans. Intell. Syst. Technol.* 2.3, 27:1–27:27. ISSN: 2157-6904.

Chapelle, O. (2007). "Training a support vector machine in the primal". In: *Neural computation* 19.5, pp. 1155–1178.

Chen, P.-H., Fan, R.-E., and Lin, C.-J. (2006). "A study on SMO-type decomposition methods for support vector machines". In: *IEEE Trans. Neural Networks* 17.4, pp. 893–908.

Cherkassky, V. and Ma, Y. (2004). "Practical selection of SVM parameters and noise estimation for SVM regression". In: *Neural Networks* 17, pp. 113–126.

Clarkson, K. L. (2010). "Coresets, Sparse Greedy Approximation, and the Frank-Wolfe Algorithm". In: *ACM Trans. Algorithms* 6.4, 63:1–63:30. ISSN: 1549-6325.

Combettes, P. L. and Pesquet, J.-C. (2009). "Proximal Splitting Methods in Signal Processing". In: *ArXiv e-prints*.

Cristianini, N. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. 1st ed. Cambridge University Press. ISBN: 0521780195.

Csiba, D., Qu, Z., and Richtárik, P. (2015). "Stochastic Dual Coordinate Ascent with Adaptive Probabilities". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Vol. 37. ICML'15. Lille, France: JMLR.org, pp. 674–683.

Daubechies, I., Defrise, M., and De Mol, C. (2004). "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint". In: *Communications on pure and applied mathematics* 57.11, pp. 1413–1457.

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives". In: *Advances in Neural Information Processing Systems*, pp. 1646–1654.

Díaz-Vico, D., Omari, A., Torres-Barrán, A., and Dorronsoro, J. R. (2017a). "Deep Fisher Discriminant Analysis". In: *International Work-Conference on Artificial Neural Networks*. To appear. Springer.

Díaz-Vico, D., Torres-Barrán, A., Omari, A., and Dorronsoro, J. R. (2017b). "Deep Neural Networks for Wind and Solar Energy Prediction". In: *Neural Processing Letters*, pp. 1–16.

Díaz, D., Torres, A., and Dorronsoro, J. R. (2015). "Deep Neural Networks for Wind Energy Prediction". In: *International Work-Conference on Artificial Neural Networks*. Springer, pp. 430–443.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). "Efficient projections onto the l 1-ball for learning in high dimensions". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 272–279.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). "Least angle regression". In: *Annals of Statistics* 32, pp. 407–499.

El Ghaoui, L., Viallon, V., and Rabbani, T. (2010). "Safe feature elimination in sparse supervised learning". In: *EECS Department, University of California, Berkeley, Tech. Rep.*

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005a). "Working Set Selection using Second Order Information for Training Support Vector Machines". In: *Journal of Machine Learning Research* 6, pp. 1889–1918. ISSN: 1533-7928.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005b). "Working set selection using second order information for training support vector machines". In: *Journal of machine learning research* 6.Dec, pp. 1889–1918.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). "LIBLINEAR: A library for large linear classification". In: *Journal of machine learning research* 9.Aug, pp. 1871–1874.

Fercoq, O. and Richtárik, P. (2015). "Accelerated, parallel, and proximal coordinate descent". In: *SIAM Journal on Optimization* 25.4, pp. 1997–2023.

Fercoq, O., Gramfort, A., and Salmon, J. (2015). "Mind the duality gap: safer rules for the Lasso". In: *Proceedings of The 32nd International Conference on Machine Learning*, pp. 333–342.

Figueiredo, M. A. T., Nowak, R. D., and Wright, S. J. (2007). "Gradient Projection for Sparse Reconstruction: Application to Compressed Sensing and Other Inverse Problems". In: *Selected Topics in Signal Processing and IEEE Journal of* 1.4, pp. 586–597.

Flammarion, N. and Bach, F. R. (2015). "From Averaging to Acceleration, There is Only a Step-size." In: *COLT*, pp. 658–695.

Frandi, E., Ñanculef, R., and Suykens, J. A. K. (2015). "A PARTAN-accelerated Frank-Wolfe algorithm for large-scale SVM classification". In: *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pp. 1–8.

Frandi, E., Ñanculef, R., Lodi, S., Sartori, C., and Suykens, J. A. (2016). "Fast and scalable Lasso via stochastic Frank–Wolfe methods with a convergence guarantee". In: *Machine Learning* 104.2-3, pp. 195–221.

Frank, M. and Wolfe, P. (1956). "An algorithm for quadratic programming". In: *Naval Research Logistics Quarterly* 3.1-2, pp. 95–110. ISSN: 1931-9193.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1, pp. 1–22. ISSN: 1548-7660.

Friedman, J., Hastie, T., Höfling, H., Tibshirani, R., et al. (2007). "Pathwise coordinate optimization". In: *The Annals of Applied Statistics* 1.2, pp. 302–332.

Fu, W. J. (1998). "Penalized Regressions: The Bridge versus the Lasso". In: *Journal of Computational and Graphical Statistics* 7.3, pp. 397–416. ISSN: 10618600.

Gärtner, B., Jaggi, M., and Maria, C. (2012). "An exponential lower bound on the complexity of regularization paths". In: *Journal of Computational Geometry* 3.1, pp. 168–195.

Geman, S., Bienenstock, E., and Doursat, R. (1992). "Neural Networks and the Bias/Variance Dilemma". In: *Neural Computation* 4.1, pp. 1–58. ISSN: 0899-7667.

Genkin, A., Lewis, D. D., and Madigan, D. (2007). "Large-scale Bayesian logistic regression for text categorization". In: *Technometrics* 49.3, pp. 291–304.

Giesen, J., Jaggi, M., and Laue, S. (2012). "Approximating parameterized convex optimization problems". In: *ACM Transactions on Algorithms (TALG)* 9.1, p. 10.

Gilbert, E. (1966). "Minimizing the Quadratic Form on a Convex Set". In: *SIAM Journal on Control* 4, pp. 61–79.

Glasmachers, T. and Igel, C. (2006). "Maximum-gain working set selection for SVMs". In: *Journal of Machine Learning Research* 7.Jul, pp. 1437–1466.

Goh, G. (2017). "Why Momentum Really Works". In: *Distill*. URL: `http://distill.pub/2017/momentum`.

Grave, E., Obozinski, G., and Bach, F. R. (2011). "Trace Lasso: a trace norm regularization for correlated designs". In: *CoRR* abs/1109.1990.

GuéLat, J. and Marcotte, P. (1986). "Some comments on Wolfe's 'away step'". In: *Mathematical Programming* 35.1.

Hastie, T., Tibshirani, R., and Friedman, J. (2003). *The Elements of Statistical Learning: Data Mining and Inference and and Prediction*. Corrected. Springer. ISBN: 0387952845.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S. (2008). "A dual coordinate descent method for large-scale linear SVM". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 408–415.

Hsu, C.-W. and Lin, C.-J. (2002). "A simple decomposition method for support vector machines". In: *Machine Learning* 46.1-3, pp. 291–314.

Jaggi, M. (2014). "An equivalence between the lasso and support vector machines". In: *Regularization, optimization, kernels, and support vector machines*. Chapman and Hall/CRC, pp. 1–26.

Joachims, T. (1998). *Making large-scale SVM learning practical*. Tech. rep. Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund.

Joachims, T. (2006). "Training linear SVMs in linear time". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 217–226.

Joachims, T. and Yu, C.-N. J. (2009). "Sparse kernel SVMs via cutting-plane training". In: *Machine Learning* 76.2-3, pp. 179–193.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., and Murthy, K. R. K. (2001). "Improvements to Platt's SMO algorithm for SVM classifier design". In: *Neural computation* 13.3, pp. 637–649.

Keerthi, S. S. and Lin, C. J. (2003). "Asymptotic behaviors of support vector machines with Gaussian kernel". In: *Neural Comput.* 15.7, pp. 1667–1689. ISSN: 0899-7667.

Kim, S.-J., Koh, K., Lustig, M., Boyd, S., and Gorinevsky, D. (2007). "An Interior-Point Method for Large-Scale ℓ_1-Regularized Least Squares". In: *IEEE journal of selected topics in signal processing* 1.4, pp. 606–617.

Kimeldorf, G. and Wahba, G. (1971). "Some results on Tchebycheffian spline functions". In: *Journal of mathematical analysis and applications* 33.1, pp. 82–95.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). "Online learning with kernels". In: *IEEE transactions on signal processing* 52.8, pp. 2165–2176.

Koh, K., Kim, S.-J., and Boyd, S. (2007). "An interior-point method for large-scale l1-regularized logistic regression". In: *Journal of Machine learning research* 8.Jul, pp. 1519–1555.

Konečnỳ, J., Liu, J., Richtárik, P., and Takáč, M. (2016). "Mini-batch semi-stochastic gradient descent in the proximal setting". In: *IEEE Journal of Selected Topics in Signal Processing* 10.2, pp. 242–255.

Kooij, A. J. (2007). "Prediction accuracy and stability of regression with optimal scaling transformations". PhD thesis. Department of Data Theory, University of Leiden. URL: `https://openaccess.leidenuniv.nl/dspace/handle/1887/12096`.

Langford, J., Li, L., and Zhang, T. (2009). "Sparse online learning via truncated gradient". In: *Journal of Machine Learning Research* 10.Mar, pp. 777–801.

Lázaro, J. L. and Dorronsoro, J. R. (2012). "Simple Proof of Convergence of the SMO Algorithm for Different SVM Variants". In: *IEEE Trans. Neural Netw. Learning Syst.* 23.7, pp. 1142–1147.

Lessard, L., Recht, B., and Packard, A. (2016). "Analysis and design of optimization algorithms via integral quadratic constraints". In: *SIAM Journal on Optimization* 26.1, pp. 57–95.

List, N. and Simon, H. U. (2007). "General polynomial time decomposition algorithms". In: *Journal of Machine Learning Research* 8.Feb, pp. 303–321.

Liu, J. and Ye, J. (2009). "Efficient Euclidean projections in linear time". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, pp. 657–664.

Liu, J. and Ye, J. (2010). *Efficient l1/lq norm regularization*. Version 1. arXiv: `1009.4766v1 [cs.LG]`.

Liu, J., Ji, S., Ye, J., et al. (2009). "SLEP: Sparse learning with efficient projections". In: *Arizona State University* 6, p. 491.

Liu, J., Zhao, Z., Wang, J., and Ye, J. (2014). "Safe screening with variational inequalities and its application to lasso". In: *International Conference on Machine Learning*, pp. 289–297.

López, J. and Dorronsoro, J. R. (2015). "Linear convergence rate for the MDM algorithm for the Nearest Point Problem". In: *Pattern Recognition* 48.4, pp. 1510–1522.

Luo, Z.-Q. and Tseng, P. (1992). "On the convergence of the coordinate descent method for convex differentiable minimization". In: *Journal of Optimization Theory and Applications* 72.1, pp. 7–35.

Mairal, J. and Yu, B. (2012). "Complexity Analysis of the Lasso Regularization Path". In: *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 353–360.

Ñanculef, R., Frandi, E., Sartori, C., and Allende, H. (2014). "A novel Frank-Wolfe algorithm. Analysis and applications to large-scale SVM training". In: *Inf. Sci.* 285, pp. 66–99.

Nemirovskii, A., Yudin, D. B., and Dawson, E. R. (1983). *Problem complexity and method efficiency in optimization*. Wiley.

Nesterov, Y. (2012). "Efficiency of coordinate descent methods on huge-scale optimization problems". In: *SIAM Journal on Optimization* 22.2, pp. 341–362.

Nesterov, Y. (1983). "A method of solving a convex programming problem with convergence rate $O(1/k^2)$". In: *Soviet Mathematics Doklady* 27.2, pp. 372–376.

Nesterov, Y. (2004). *Introductory lectures on convex optimization : a basic course*. Applied optimization. Boston, Dordrecht, London: Kluwer Academic Publ. ISBN: 1-4020-7553-7.

Nocedal, J. and Wright, S. J. (2006). *Numerical optimization*. 2nd ed. Springer.

Nutini, J., Schmidt, M. W., Laradji, I. H., Friedlander, M. P., and Koepke, H. A. (2015). "Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection." In: *ICML*, pp. 1632–1641.

O'Donoghue, B. and Candès, E. J. (2015). "Adaptive Restart for Accelerated Gradient Schemes". In: *Foundations of Computational Mathematics* 15.3, pp. 715–732.

Ogawa, K., Suzuki, Y., and Takeuchi, I. (2013). "Safe Screening of Non-Support Vectors in Pathwise SVM Computation". In: *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1382–1390.

Osborne, M. R., Presnell, B., and Turlach, B. A. (2000). "On the lasso and its dual". In: *Journal of Computational and Graphical statistics* 9.2, pp. 319–337.

Osuna, E., Freund, R., and Girosi, F. (1997). *Support vector machines: Training and applications*. Tech. rep.

Palagi, L. and Sciandrone, M. (2005). "On the convergence of a modified version of SVM light algorithm". In: *Optimization methods and Software* 20.2-3, pp. 317–334.

Plackett, R. L. (1950). "Some Theorems in Least Squares". In: *Biometrika* 37.1-2, pp. 149–157.

Platt, J. (1998). *Sequential minimal optimization: A fast algorithm for training support vector machines*. Tech. rep.

Polyak, B. T. (1964). "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1–17.

Puig, A. T., Wiesel, A., and Hero, A. O. (2009). "A multidimensional shrinkage-thresholding operator". In: *Statistical Signal Processing, 2009. SSP'09. IEEE/SP 15th Workshop on*. IEEE, pp. 113–116.

Qu, Z. and Richtárik, P. (2016a). "Coordinate descent with arbitrary sampling I: Algorithms and complexity". In: *Optimization Methods and Software* 31.5, pp. 829–857.

Qu, Z. and Richtárik, P. (2016b). "Coordinate descent with arbitrary sampling II: Expected separable overapproximation". In: *Optimization Methods and Software* 31.5, pp. 858–884.

Qu, Z., Richtárik, P., and Zhang, T. (2015). "Quartz: Randomized dual coordinate ascent with arbitrary sampling". In: *Advances in neural information processing systems*, pp. 865–873.

Ren, Z., Yang, Y., Bao, F., Deng, Y., and Dai, Q. (2016). "Directed Adaptive Graphical Lasso for causality inference". In: *Neurocomputing* 173, pp. 1989–1994.

Richtárik, P. and Takáč, M. (2014). "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function". In: *Mathematical Programming* 144.1-2, pp. 1–38.

Richtárik, P. and Takáč, M. (2016). "Parallel coordinate descent methods for big data optimization". In: *Mathematical Programming* 156.1-2, pp. 433–484.

Roush, F. W. (1982). "Applied linear regression". In: *Mathematical Social Sciences* 3.1, pp. 92–93.

Saha, A. and Tewari, A. (2013). "On the nonasymptotic convergence of cyclic coordinate descent methods". In: *SIAM Journal on Optimization* 23.1, pp. 576–601.

Scherrer, C., Halappanavar, M., Tewari, A., and Haglin, D. (2012). "Scaling Up Coordinate Descent Algorithms for Large L1 Regularization Problems". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland, pp. 355–362.

Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). "New support vector algorithms". In: *Neural computation* 12.5, pp. 1207–1245.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). "Estimating the support of a high-dimensional distribution". In: *Neural computation* 13.7, pp. 1443–1471.

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

Shalev-Shwartz, S. and Tewari, A. (2011). "Stochastic methods for l1-regularized loss minimization". In: *Journal of Machine Learning Research* 12.Jun, pp. 1865–1892.

Shalev-Shwartz, S. and Zhang, T. (2012). "Proximal stochastic dual coordinate ascent". In: *arXiv preprint arXiv:1211.2717*.

Shalev-Shwartz, S. and Zhang, T. (2014). "Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization." In: *ICML*, pp. 64–72.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). "Pegasos: Primal estimated sub-gradient solver for svm". In: *Proceedings of the 24th international conference on Machine learning*. ACM, pp. 807–814.

Shevade, S. K. and Keerthi, S. S. (2003). "A simple and efficient algorithm for gene selection using sparse logistic regression". In: *Bioinformatics* 19.17, pp. 2246–2253.

Steinwart, I. (2003). "Sparseness of support vector machines". In: *Journal of Machine Learning Research* 4.Nov, pp. 1071–1105.

Su, W., Boyd, S., and Candes, E. (2014). "A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., pp. 2510–2518.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. Ed. by S. Dasgupta and D. Mcallester. Vol. 28. 3. JMLR Workshop and Conference Proceedings, pp. 1139–1147.

Tibshirani, R. (1994). "Regression Shrinkage and Selection Via the Lasso". In: *Journal of the Royal Statistical Society and Series B* 58, pp. 267–288.

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J., and Tibshirani, R. J. (2012). "Strong rules for discarding predictors in lasso-type problems". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 74.2, pp. 245–266.

Tomioka, R., Suzuki, T., and Sugiyama, M. (2011). "Super-linear convergence of dual augmented Lagrangian algorithm for sparsity regularized estimation". In: *Journal of Machine Learning Research* 12.May, pp. 1537–1586.

Torres-Barrán, A. and Dorronsoro, J. R. (2015). "Conjugate Descent for the Minimum Norm Problem". In: *NIPS Workshop on Optimization for Machine Learning (OPT)*.

Torres-Barrán, A. and Dorronsoro, J. R. (2016a). "Conjugate descent for the SMO algorithm". In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 3817–3824.

Torres-Barrán, A. and Dorronsoro, J. R. (2016b). "Nesterov Acceleration for the SMO Algorithm". In: *International Conference on Artificial Neural Networks*. Springer, pp. 243–250.

Torres-Barrán, A., Alonso, Á., and Dorronsoro, J. R. (2017). "Regression Tree Ensembles for Wind Energy and Solar Radiation Prediction". In: *Neurocomputing*. In press.

Torres, A., Prada, J., and Dorronsoro, J. R. (2014a). "Nowcasting Meteorological Readings for Wind Energy Prediction". In: *EWEA 2014*.

Torres, A., Díaz, D., and Dorronsoro, J. R. (2014b). "Sparse one hidden layer MLPs." In: *Proceedings of ESANN 2014, Bruges, Belgium, 23-25 April 2014*, pp. 655–660.

Tseng, P. (2001). "Convergence of a block coordinate descent method for nondifferentiable minimization". In: *Journal of optimization theory and applications* 109.3, pp. 475–494.

Tseng, P. and Yun, S. (2009a). "A coordinate gradient descent method for nonsmooth separable minimization". In: *Mathematical Programming* 117.1, pp. 387–423.

Tseng, P. and Yun, S. (2009b). "Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization". In: *Journal of optimization theory and applications* 140.3, p. 513.

Tyree, S., Gardner, J. R., Weinberger, K. Q., Agrawal, K., and Tran, J. *Parallel Support Vector Machines in Practice*. arXiv: `1404.1066v1 [cs.LG]`.

Vidaurre, D., Bielza, C., and Larrañaga, P. (2013). "Classification of neural signals from sparse autoregressive features". In: *Neurocomputing* 111, pp. 21–26.

Wang, J., Zhou, J., Wonka, P., and Ye, J. (2013). "Lasso screening rules via dual polytope projection". In: *Advances in Neural Information Processing Systems*, pp. 1070–1078.

Wang, P.-W. and Lin, C.-J. (2014). "Iteration complexity of feasible descent methods for convex optimization." In: *Journal of Machine Learning Research* 15.1, pp. 1523–1548.

Wibisono, A., Wilson, A., and Jordan, M. (2016). "A Variational Perspective on Accelerated Methods in Optimization". In: *Arxiv* 1603.04245.

Wolfe, P. (1970). "Convergence theory in nonlinear programming". In: *Integer and Nonlinear Programming*. Ed. by J. Abadie. North–Holland, pp. 1–36.

Wright, S. (2013). *Optimization*.

Wu, T. T. and Lange, K. (2008). "Coordinate descent algorithms for lasso penalized regression". In: *The Annals of Applied Statistics*, pp. 224–244.

Xiang, Z. J., Wang, Y., and Ramadge, P. J. (2017). "Screening tests for lasso problems". In: *IEEE transactions on pattern analysis and machine intelligence* 39.5, pp. 1008–1027.

Xiao, L. (2010). "Dual averaging methods for regularized stochastic learning and online optimization". In: *Journal of Machine Learning Research* 11.Oct, pp. 2543–2596.

Xiao, L. and Zhang, T. (2014). "A proximal stochastic gradient method with progressive variance reduction". In: *SIAM Journal on Optimization* 24.4, pp. 2057–2075.

Xu, B., Huang, K., King, I., Liu, C.-L., Sun, J., and Satoshi, N. (2014). "Graphical lasso quadratic discriminant function and its application to character recognition". In: *Neurocomputing* 129, pp. 33–40.

Yu, K., Leufen, G., Hunsche, M., Noga, G., Chen, X., and Bareth, G. (2014). "Investigation of Leaf Diseases and Estimation of Chlorophyll Concentration in Seven Barley Varieties Using Fluorescence and Hyperspectral Indices". In: *Remote Sensing* 6.1, pp. 64–86. ISSN: 2072-4292.

Yuan, G.-X., Ho, C.-H., and Lin, C.-J. (2012). "An improved glmnet for l1-regularized logistic regression". In: *Journal of Machine Learning Research* 13.Jun, pp. 1999–2030.

Zhang, T. and Oles, F. J. (2001). "Text categorization based on regularized linear classification methods". In: *Information retrieval* 4.1, pp. 5–31.

Zhou, Q., Chen, W., Song, S., Gardner, J. R., Weinberger, K. Q., and Chen, Y. (2014). "A reduction of the elastic net to support vector machines with an application to gpu computing". In: *arXiv preprint arXiv:1409.1976*.

Zhou, Q., Song, S., Huang, G., and Wu, C. (2015). "Efficient Lasso training from a geometrical perspective". In: *Neurocomputing* 168, pp. 234–239.

Zou, H. and Hastie, T. (2005). "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2, pp. 301–320. ISSN: 1467-9868.