UNIVERSIDAD AUTÓNOMA DE MADRID

Efficient Optimization Methods for Regularized Learning: Support Vector Machines and Total–Variation Regularization

by

Álvaro Barbero Jiménez

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

in the
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Under the supervision of José Ramón Dorronsoro Ibero

May 2011

Abstract

In the context of machine learning methods, regularization has become an established practice to control overfitting in the modeling process and to induce structure into the resultant models. At the same time, the flexibility of the regularization framework has provided a common point of view embracing classical and established learning models, as well as recent proposals in the topic. This richness comes from its appealing simplicity, which casts the learning process into a composite optimization problem formed by a loss function and a regularizer; different models are obtained through the selection of appropriate loss and regularizer functions.

This elegant modularity, however, does not come at no cost, as an adequate optimization algorithm must be applied or devised in order to solve the resultant problem. While general purpose solvers are directly applicable out–of–the–box in some settings, they usually produce poor results in terms of efficiency and scalability. Further, in more complex models featuring non-smooth or even non-convex loss or regularizer functions, such approaches easily become inapplicable. Consequently, the design of appropriate optimization methods becomes a key task for the success of a regularized learning process.

In this thesis two particular cases of regularization are studied in depth. On the one hand, the well established and successful Support Vector Machine model is presented in its different forms. A careful observation at the current algorithmic solutions to this problem shows that correcting hidden deficiencies and making a better use of the gathered information can lead to significant improvements in running times, surpassing state of the art methods. On the other hand, a class of sparsity-inducing regularizers known as Total–Variation is studied, with wide application in the fields of signal and image processing. While a variety of approaches have been applied to solve this class of problems, it is shown here that by taking advantage of their strong structural properties and adapting suitable optimization algorithms, relevant improvements in efficiency and scalability can be obtained as well. Software implementing the developed methods is also made available as part of this thesis.

Resumen

En el ámbito de los métodos de aprendizaje automático, la regularización se ha convertido en una práctica habitual para controlar el sobreajuste en el proceso de modelización, así como para inducir una estructura en los modelos resultantes. Al mismo tiempo, la flexibilidad otorgada por este marco de regularización permite abarcar bajo un punto de vista único tanto modelos de aprendizaje estándar en el área como propuestas recientes. Esta riqueza está fundamentada en su simplicidad, que convierte el problema del aprendizaje en un problema de optimización compuesto por una función de pérdida más un regularizador. La selección apropiada de diferentes funciones de pérdida y regularizadores permite, por tanto, obtener toda una serie de modelos diferentes para cada situación.

Esta elegante modularidad, no obstante, no está exenta de un coste, puesto que debe aplicarse o diseñarse un algoritmo adecuado para resolver el problema de optimización resultante. A pesar de que existen métodos estándar de optimización capaces de resolver directamente tal problema en muchos de los casos, generalmente estos métodos tienen malos resultados en términos de eficiencia y escalabilidad. O lo que es peor, si la función de pérdida o el regularizador resultan ser funciones no diferenciables o no convexas, dando lugar a modelos más complejos, tales métodos suelen ser completamente inaplicables. Por tanto, el diseño de algoritmos de optimización adecuados a los modelos elegidos resulta ser una pieza clave en el éxito del proceso de aprendizaje regularizado.

En esta tesis se estudian en profundidad dos casos particulares de regularización. En primer lugar, las Máquinas de Vectores de Soporte se presentan en sus diferentes formas, las cuales son consideradas como modelos bien establecidos en el área. Mediante una observación detallada de los algoritmos que resuelven este problema se detectan deficiencias ocultas en los mismos, y se proponen formas de corregirlas mediante un mejor uso de la información manejada por tales algoritmos, obteniendo mejoras significativas en tiempos de ejecución. En segundo lugar se estudia la clase de regularizadores conocida como de Variación Total, los cuales han sido ampliamente utilizados en los campos de procesado de señal y de imagen para producir modelos de parámetros dispersos. A pesar de que ya se han analizado múltiples formas de abordar esta clase de problemas, en esta tesis se demuestra que haciendo uso explícito de sus propiedades estructurales y adaptando algoritmos de optimización apropiados pueden conseguirse mejoras relevantes en eficiencia y escalabilidad. Como parte de esta tesis se incluyen también programas implementando las soluciones algorítmicas aquí desarrolladas.

Acknowledgements

This thesis is the outcome of three years devoted to learning, studying, theorizing, experimenting, reading, writing, travelling and teaching about the topics here addressed. This final product, which concludes a life stage and opens a new one, is the most visible result of the long process required to achieve the degree of Doctor of Philosophy. It is therefore righteous to acknowledge here every support received during this period, supports that have been essential for the fulfillment of this work.

In the first place I would like to sincerely thank my parents and brother for all the confidence they have placed on me during these years, as well as their moral and personal support. The same can be said for my close friends, which I shall name in strict alphabetical order to avoid favoritisms: Antonio Almazán, Ana María Arroyo, Jorge Barriocanal, Elena Benitez, Cristina & Enrique Clemente, Sandra Fresnillo, Abril Pinero, and Álvaro Valera. My colleagues at IIC deserve equal credit, specially my close companions in this PhD journey: Jorge López and Irene Rodríguez.

Special mention must be made to my PhD director and professor José Ramón Dorronsoro, whose guidance and advices have been of invaluable worth in my struggle against the devious intricacies of theoretical and practical science, publishing, and even bureaucracy.

I should also mention the grants and institutions that have provided the fundings for the realization of all the activities involved in this PhD: the FPU grant kindly funded by the Spanish ministry of education and science (reference AP2006-02285), the pre-doctoral grants and afterwards permanent position offered by the Instituto de Ingeniería del Conocimiento, and the fundings for travel, conference and publication expenses covered by program Spain's TIN 2007-66862 and "Cátedra IIC Modelado y Predicción". I would like to thank as well the institutions where I had the opportunity to develop this work, the Universidad Autónoma de Madrid and the Max Planck Institute for Intelligent Systems. From my stays at the latter I must thank my supervisor Moritz Grosse-Wentrup and also Stefan Harmeling for kindly reviewing this manuscript. Special mention also goes for Suvrit Sra for all his help and guidance during the last year of this work.

And finally, in a less formal tone, I wish to say thanks to all the staff and supporters of the Albarji Productions web community, and specially Felipe, Mati and Nus. Keep up the good work! :)

Contents

Abbreviations

CB	**C**ycle **B**reaking
CH	**C**onvex **H**ulls
FLSA	**F**used **L**asso **S**ignal **A**pproximator
GP	**G**radient **P**rojection
i.i.d.	**i**ndependent and **i**dentically **d**istributed
ISNR	**I**mproved **S**ignal to **N**oise **R**atio
KKT	**K**arush–**K**uhn–**T**ucker
LS–SVM	**L**east **S**quares **S**upport **V**ector **M**achine
MDM	**M**itchel–**D**em'yanov–**M**alozemov
MSMO	**M**omentum **S**equential **M**inimal **O**ptimization
MSN	**M**oré **S**orensen **N**ewton
PD	**P**roximal **D**ykstra
PDHG	**P**rimal **D**ual **H**ybrid **G**radient
PN	**P**rojected **N**ewton
QP	**Q**uadratic **P**roblem
RCH	**R**educed **C**onvex **H**ulls
ROF	**R**udin–**O**sher–**F**atemi
SMO	**S**equential **M**inimal **O**ptimization
SV	**S**upport **V**ector
SVC	**S**upport **V**ector **C**lassification
SVR	**S**upport **V**ector **R**egression
SVM	**S**upport **V**ector **M**achine
TV	**T**otal–**V**ariation
VC	**V**apnik **C**hervonenkis
WSS	**W**orking **S**et **S**election

To my family and friends

Chapter 1

Introduction

"I almost wish I hadn't gone down that rabbit-hole
and yet – and yet – it's rather curious, you know, this sort of life!"
Alice

During the last decades computer technology has experienced extraordinary advances. Computers, initially following their name in the most literal sense, consisted in just machines to perform mathematical computations. The complexity of their components and use made them only reachable to devoted experts and for very specific calculating purposes, thus being not so different to industrial machinery. This early use of computers is utterly striking when compared to their situation nowadays: computers are ubiquitous, being present in almost every office and home. Current mobile phones, television, music and video players, and even credit cards also make use of digital computation as their keystone component. And even though the complexity of the circuitry forming this technology has increased exponentially, their use is now widespread, having produced an undeniable change in societies all over the world.

Two main factors can explain this tremendous change in the field. The first one are the incredible improvements realized in microchip technologies, which have restlessly augmented the computational power of hardware at an exponential rate year after year since the early 70's by incrementing the number of transistors contained in a Central Processing Unit (CPU). This trend was already predicted in 1965 by Intel co-founder Gordon E. Moore in what came to be known as **Moore's law** [1]. This empirical law states that every two years, the number of transistors in a CPU approximately doubles. Figure 1.1 [1] presents a chart of this predicted trend together with factual evolution of microprocessors during the last decades: up to now the law has been surprisingly

[1] Image extracted from Wikipedia: `http://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2008.svg`

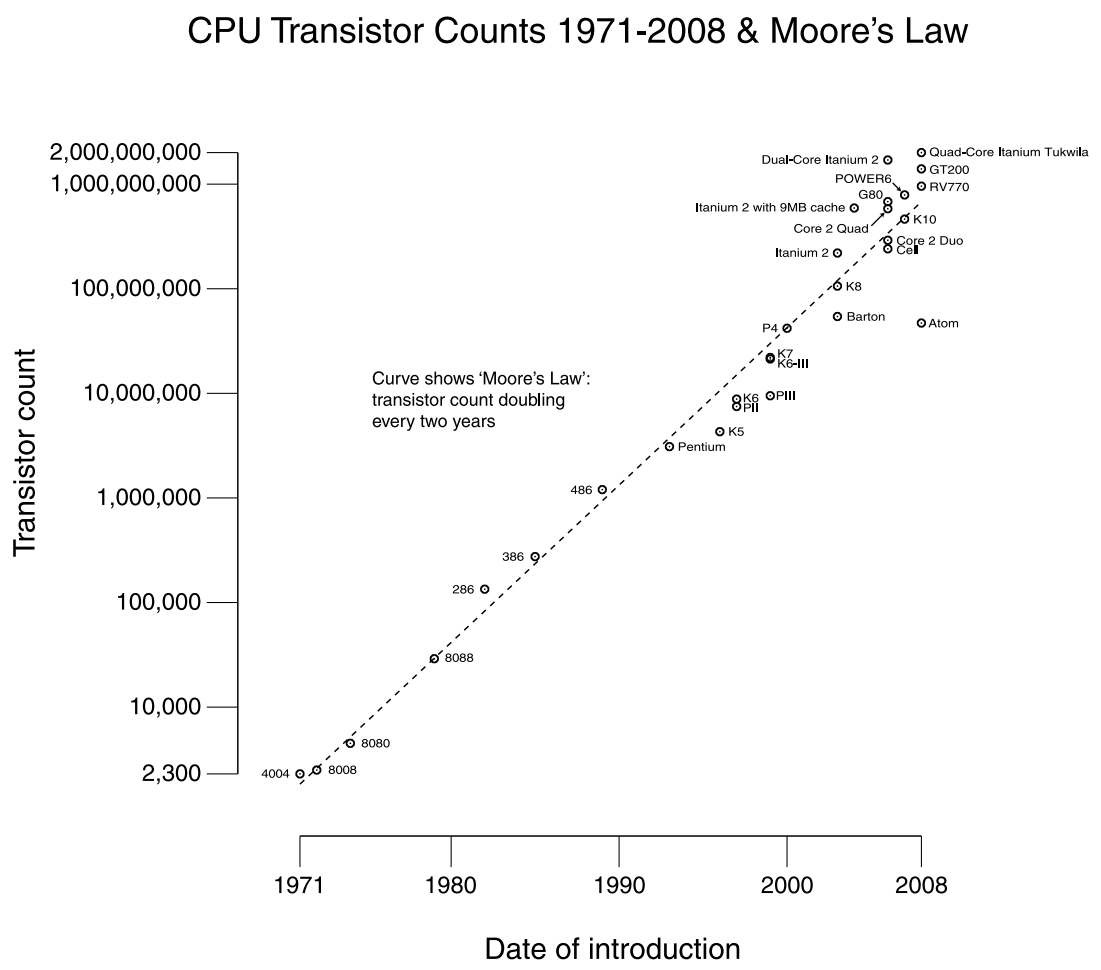


FIGURE 1.1: Number of transistors for a series of CPU models during the last decades. The dashed line represents the predictions by Moore's law.

accurate. Surprisingly, not only because of it being a mere empirical law, but also because such growth in technology is completely unprecedented.

The second factor of the generalization of computers corresponds to the more theoretical side of computation, which has had strong implications in the applications of computers, and ultimately, in the way they are used. **Computer science** has produced methods and algorithms to make use of the available hardware technologies in order to solve complex computing problems, devising ways to store, manage, and process data, and studying human-computer interaction aspects to improve their usability and their ability to aid humans in technical tasks. This has, thus, extended the use of computers to countless applications, including word processing, accounting, multimedia, gaming, etc. Probably the most outstanding example of a categorical jump in the usage possibilities of computers was made by the invention of the Internet, which unleashed a genuine technological revolution and completely changed the concepts of information search and telecommunications.

Undoubtedly, the recent advances in computer technology have given a wondrous new shape to the world. However, and in spite of this, all the processing power of the most recent processor fades when compared to the abilities of the best computing device created by nature: the human brain. Counting up to around 85 billion adaptive processing units in the form of neurons, the brain is able to address easily extremely complex tasks such as learning, motion planning, language processing and image recognition, tasks only clumsily approachable by state–of–the–art methods in computer science. All of these problems share a common attribute: they cannot be easily defined in an exact, formal mathematical way. Formally defining the complete structure of a language is a daunting task, hardly able to take in consideration subtleties such as ironies. Computer-controlled robotic arms and cameras have already been developed, but making such contraption learn to play ping–pong by sheer observation seems an insurmountable challenge. The reason of this is simple: regardless of how much microprocessor technology or computer science have evolved, computers remain being machines to perform mathematical computations, making them perfect candidates to solve tasks reducible to mathematical problems, but poor approaches to work on more fuzzily defined problems.

Because of this, the field of computer science known as **artificial intelligence** has pursued ways to imitate human intelligence through the use of computers. The question is, however, how to define human intelligence. The famous **Turing test** [2] proposed a way to determine whether a computer shows true intelligence: a human maintains a conversation with a computer program, and if the human cannot determine whether he is actually talking to another human or to the computer program itself, then that program can be regarded as intelligent. Still, this definition of intelligence is also ambiguous and controversial, and so up to now a formal satisfactory definition of intelligence has not been attained. Because of this and the limitations of computers, most of the successful research in the field of artificial intelligence has been focused on solving more specific and well-defined problems, such as finding the shortest path between two points in a predefined environment, or planning strategies for games with set rules – for instance chess. Therefore, artificial intelligence algorithms generally address such problems by analyzing the rules or properties defining the problem, and then planning strategies aimed at solving them effectively, so that they might appear as "intelligent" solutions for an observer.

An alternative approach to such problems is given by **machine learning**, which can be regarded as a branch of artificial intelligence. Instead of having a predefined set of rules defining the problem, what machine learning proposes is to collect data on different strategies or inputs tried on the problem and the outcomes obtained, intending then to discover what characterizes a good solution. More specifically, the problem is treated as a **black–box system** producing an output y depending on an input x, following

unknown and not necessarily deterministic rules. What machine learning does then is trying to build a **model** which approximates the behavior of the system by just observing a number of input–output examples. For instance, in the tic–tac–toe game a machine learning approach would be oblivious to the actual rules of the game, just requiring to collect data on a number of plays and their outcome (player's choices and winner), building then a model that captures the knowledge on when a set of player's choices leads to victory or defeat. This knowledge could then be used to produce an artificial tic–tac–toe player performing winning strategies similar to the ones observed. Because of this, what the computer actually does is **learning** how to solve a particular problem by observation. This approach to problem-solving certainly seems more natural when compared to the way humans or other living beings deal with their everyday life, in contrast to a classic artificial intelligence approach. It also tries to imitate the adaptive process of natural learning, which is performed in the brain owing to the plasticity of its neurons.

Of course, collecting data on the problem is only part of the solution; the rest of the responsibility is placed on the **learning algorithm**. Since computer hardware completely lacks the adaptability of brain neurons, the algorithm must be the adaptive component performing the learning task, adjusting itself to the system being studied and producing an appropriate model. Such algorithm should be able to infer (approximately), from the limited set of gathered data, how the inputs to the system produce their outputs. As detailed later on in the text, this can be done by selecting a general, flexible model, which is then adjusted using the available data to produce similar outputs as the system under study for the same set of inputs. The relevant point is that this adjustment procedure, known as the **training** of the model, can be expressed in formal mathematical terms, and is therefore computable. Or, in other words, the abstract problem of "learning" can be put down into a form appropriate to be solved by computers, hence allowing machines to learn.

This thesis presents the results obtained in the research of a series of topics in the field of machine learning, more specifically, on some of the models and methods in **regularized learning**. This first chapter provides an overview of the thesis, beginning with a more detailed introduction to machine learning and regularized learning, following then with the Support Vector Machine and Total–Variation regularization models, on which this thesis has focused. At the end of the chapter the contributions of this thesis are enumerated together with some indications to navigate through the rest of the chapters, where the work done is presented in full detail.

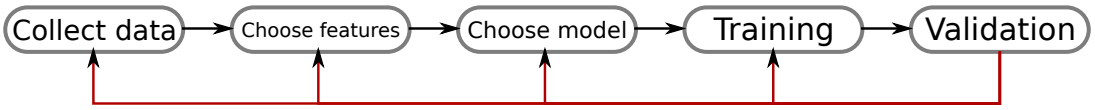


FIGURE 1.2: Design cycle of a machine learning solution.

1.1 Machine learning at a glance

Most of the times a machine learning approach is used to address a particular problem, a design cycle such as the one shown in Figure 1.2 is adopted [3]. Since machine learning is data-driven, the first step is always to **collect data** from the system under study, in the form of a set of **input patterns** x_i, $i \in [1, \dots, N]$, with their associated system outcomes or **targets** y_i, $i \in [1, \dots, N]$. The particular form of these patterns and targets depends on the system, as they might have been collected as numerical data, text, graphs, or any kind of structure. Because of this, most of the times a preprocessing of such data is performed to cast them into vectors, such that $x_i \in \mathbb{R}^D$. This is done by **choosing relevant features** that could explain the behavior of the system and can be expressed as scalars. This representation eases the following steps, although models using other forms of input patterns abound as well. How to choose relevant features is an extensive research topic, depending both on knowledge of the system being modeled and also on numerical approaches to find well-performing features.

A common and key assumption about the data is that they are **independent and identically distributed** (i.i.d) random variables. This implies that all the data patterns gathered have been generated from the same probability distribution and, in particular, that other possible patterns in the system to model also follow this distribution. If this assumption is met, adapting the model to the available data provides some guarantees on its performance over new data patterns. More details on this are given later on in Section 1.2.

After processing the data, the next step is selecting an appropriate model. One simple example of such would be a **linear model**, which assumes that the system behaves linearly, and takes the form

$$\hat{y} = f(x) = w \cdot x + b,$$

that is, for a particular input x the model predicts that the output of the system will be $\hat{y}$, and this estimate of the output is computed by weighing each of the input variables x_j, $j \in [1, \dots, D]$ with a **weight vector** w, and then adding an independent **bias** term b. $\theta = \{w, b\}$ represents the parameters of this model, which should be adjusted so that the output estimates result similar to the actual system outputs, i.e. $\hat{y}_i \simeq y_i \ \forall i$. To

Loss function	Formula		
Squared Error	$\sum_i \left(f(x_i) - y_i\right)^2$	$=$	$\|f(x) - y\|_2^2$
Absolute Error	$\sum_i \|f(x_i) - y_i\|$	$=$	$\|f(x) - y\|_1$
Maximum Error	$\max_i \{\|f(x_i) - y_i\|\}$	$=$	$\|f(x) - y\|_\infty$
Classification Error	$\text{card}\left(\{\text{sign}(f(x_i)) \neq y_i\}\right)$	$=$	$\|\text{sign}(f(x)) - y\|_0$
Logistic loss	$\sum_i \log\left(1 + e^{-y_i f(x_i)}\right)$		
Hinge loss	$\sum_i \max\{1 - y_i(w \cdot x_i + b), 0\}$	$=$	$\sum_i \|1 - y_i(w \cdot x_i + b)\|_+$

TABLE 1.1: A sample of popular choices for the loss function. $f(x)$ represents the vector resulting from applying the model function $f(\cdot)$ over each input pattern x_i, and $\text{sign}(z)$ returns -1 if $z \leq 0$, $+1$ else. Compact formulations using a norm are also shown when possible.

do so, as part of the model selection a choice of a **loss function** $L(\theta)$ must be made as well. The purpose of this function is to measure how accurately a particular choice of the model parameters θ represents the system under study. A classical example of loss function is given by the **squared error** loss, presented in Table 1.1. Since the loss grows when the model produces inaccurate outputs, the problem of learning the best model parameters can be written as the **optimization problem**

$$\min_\theta L(\theta),$$

that is, finding the model parameters θ producing a loss as small as possible. This procedure is known as **training** the model, since it is at this step where the model learns and adapts itself to the data. Following the previous example of a linear model and employing the squared error loss produces the optimization problem

$$\min_{w,b} \sum_i \left(w \cdot x_i + b - y_i\right)^2,$$

which is widely known as the **linear least squares method** for fitting an unknown function g only known through a set of observations $\{x_i, g(x_i)\}$ $\forall i \in [1, \ldots, N]$. That is, this popular method is nothing but learning a linear model $f(\cdot)$ for the system $g(\cdot)$ by using a squared error loss.

The choice of the loss function to use depends on the problem or system to model. Table 1.1 presents a selection of representative loss functions. This variety comes from

either functions adequate for specific tasks or from different assumptions of the noise present in the system. For instance, the already mentioned squared loss assumes that such noise follows a gaussian distribution, that is, the observed outputs y_i relate to the real outputs of the system $g(x_i)$ as $y_i = g(x_i) + n_i$, where n_i is a random draw from a gaussian distribution with unknown mean and variance. If such assumption is met, minimizing the squared loss results in the well-known **maximum likelihood estimator** (MLE) [4], hence handling noise appropriately. Other loss functions make different assumptions on the noise, and so their suitability depends on the actual noise distribution of the system being studied.

What should be stressed now is that the choice of loss function modifies the optimization problem to solve in order to learn the model parameters. While in the case of the squared loss a solution can be found analytically, for instance the optimization of the hinge loss requires to make use of a numerical **optimization algorithm**. This in turn implies that the training phase of the model will require more or less computational effort depending on the loss choice made, but also on the way the resulting optimization problem is solved. More efficient optimization algorithms will result in reduced training times. This is relevant because the size of the optimization problem scales with the size of the available data, and in **large-scale** settings where hundred of thousands or millions of data patterns are available it might be unfeasible to complete the training when a poor optimization algorithm is used. Also in **on–line** settings where a model must be trained on–demand with recently received data in a predefined amount of time, efficiency of the training procedure is paramount to ensure applicability of the model.

It must be mentioned that **non-linear models**, in contrast to the already introduced linear model, are also possible. Non-linear models are characterized for performing non-linear operations on their input variables, thus producing more complex and richer approximations to the system being modeled. However the way these non-linear operations are made is not predefined, hence giving rise to a wide variety of strategies to approach non-linearity. In this thesis non-linear models are approached using the **kernel trick** technique, which is presented later on in Section 2.1.2.

The final step in the design cycle is **validation**. The model is tried out on a different set of data than the one used for training, and its quality is measured by again making use of a loss function. If poor results are obtained, this might be due to a bad data collection, a failure to identify relevant features, a bad choice of the model or even a poor solution to the optimization problem posed by the training. Therefore these previous steps should be revised and repeated again as many times as necessary until good enough results are obtained in validation. How long it takes to obtain good results

depends on the particular application, but a good practice is to compare against other approaches solving the same or a similar problem.

Up to this point machine learning has been discussed from a general perspective. However, in practice several subfields of machine learning address different kinds of problems or **tasks**, adapting the design steps above to the particular task being addressed. Probably the most common task in machine learning is that of **pattern classification** [3]. This assumes that each pattern can be given a **label**, so that the pattern can be thought as belonging to a particular group or **class**. For example, an experienced medical doctor is able to classify his patients as healthy or ill by looking at the numerical results from a blood test. It would be desirable then to have an automatic method able to perform this intelligent decisions as well, e.g. for supporting decisions of less-experienced doctors. From a machine learning point of view, the experienced medical doctor is the system to be imitated, its inputs being the blood tests reports and its outputs the decisions on how to classify the patients. The key difference with the previous general setting is that the outputs are limited to $y_i \in \{\text{healthy}, \text{ill}\}$, which could be represented numerically as $y_i \in \{-1, 1\}$. That is, outputs are limited to belong to a fixed set.

While in principle any kind of model, and in particular of loss can be used for the classification setting, some of them work better. For instance, using a squared error loss is probably not the best choice for a binary classification problem like the one above, as a model producing $\hat{y}_i \simeq -0.8$ and $\hat{y}_i \simeq 0.7$ for every healthy and ill patient is not really worse than another producing exactly $\hat{y}_i = -1$ and $\hat{y}_i = 1$ for healthy and ill, since in both cases the values of $\hat{y}$ already give a perfect guess about the class of each patient. Because of this, using the **classification error** as shown in Table 1.1 would be a better loss choice. Unfortunately, this loss is extremely difficult to optimize since it reduces to the hard L0 norm (more details about norms in Section 1.2). Therefore, this loss is usually only employed for validation, where there is no need to solve an optimization problem. For training, a surrogate loss approximating the classification error is preferred, such as the **logistic** or the **hinge** losses.

Figure 1.3 shows an example of a classification problem. The resulting model constitutes a **separating hyperplane**, since when validating the model every data point x such that $w \cdot x + b > 0$ is assigned to the positive $(+1)$ class, whereas any data point such that $w \cdot x + b < 0$ is assigned to the negative (-1) class. Since the problem only has two inputs, this is visually appreciated as a line splitting the inputs space $\mathbb{R}^2$ into two parts, one for each class.

Another frequent task addressed in machine learning is **regression** or function approximation, where the outputs y_i might take any value in $\mathbb{R}$. In this setting those loss

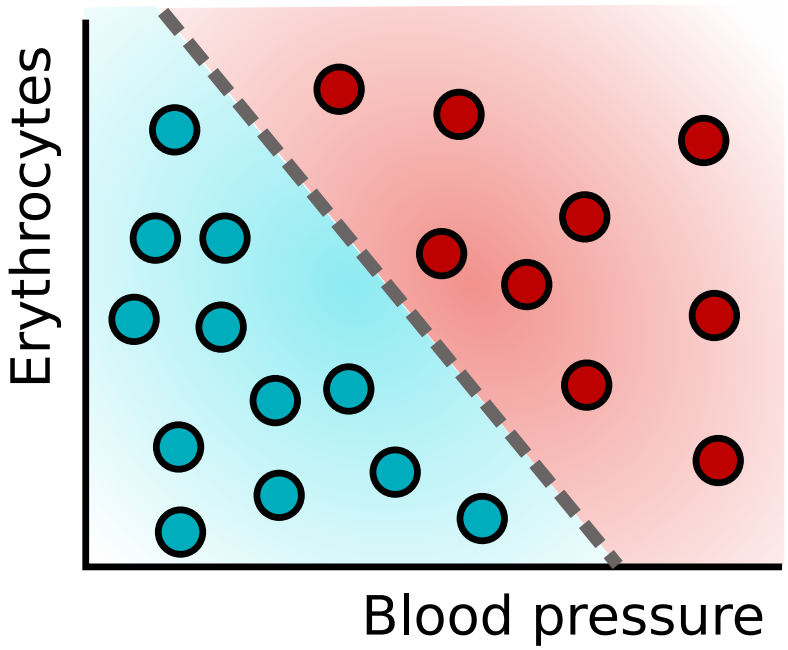


FIGURE 1.3: An example of a classification problem. Input variables are blood pressure an erythrocytes density, while the output is whether the patient is healthy or ill. Blue dots stand for data from healthy patients, red for ill ones. A linear model producing the separating hyperplane marked in gray is able to correctly classify patients.

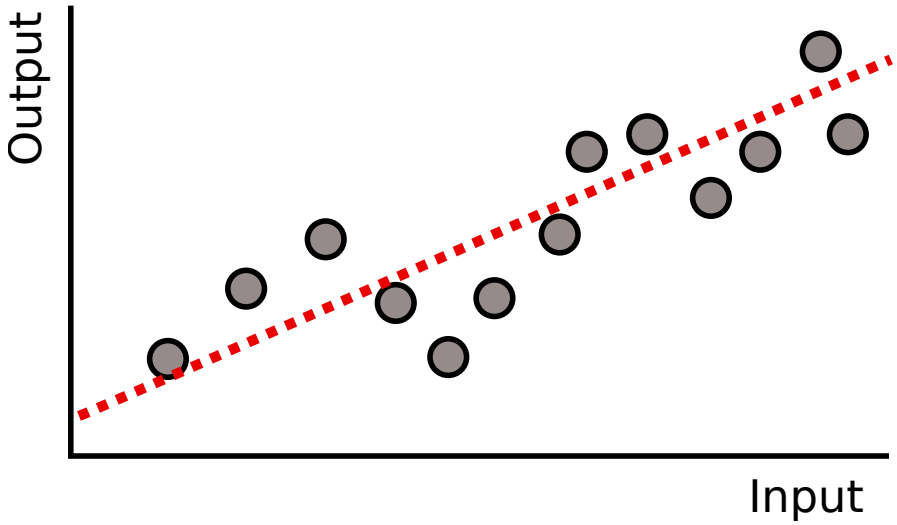


FIGURE 1.4: An example of a regression problem. A linear model finds a linear function minimizing the squared error of the data.

functions designed for classification are no longer useful, and thus squared error, **absolute error** or **maximum error** losses (Table 1.1) are put into use. When using a linear model, the linear function minimizing such loss function over the data will be the result of the training procedure. An example is depicted in Figure 1.4.

Other tasks of application in machine learning depend on the way the targets y_i are provided. To name a few: in **semi–supervised learning** [5] the targets are only

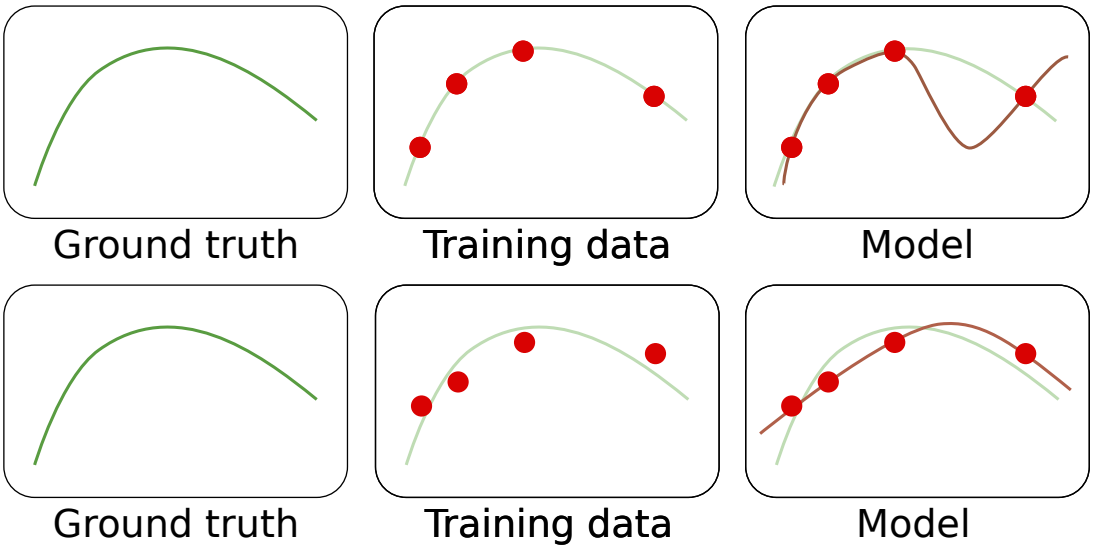


FIGURE 1.5: Example of overfitting due to excessive model complexity (first row) and noisy data (second row).

known for a subset of the input patterns. **Clustering** [3] tries to construct clusters or groups without any kind of target information, using just input values and similarity measures between patterns such as distances. **Multi–task learning** [6] solves a number of problems at once, making use of hidden relationships to improve each of the individual models. In spite of this diversity, all of them follow the design cycle above, and a correct choice of model and loss function is critical.

This thesis focuses on the tasks of classification and regression when approached with a particular family of models. Before introducing them, though, a brief review of the topic of regularization is mandatory.

1.2 Regularization

While the goal in any machine learning application is to build a model as accurate as possible, quite often the available data are scarce, not providing a thorough representation of every aspect of the system under study. Even more, the data are usually affected by **noise**, its cause being inaccuracies in the data collection process or a non-deterministic behavior of the system. These effects, present in any real-world machine learning problem, contribute to the difficulty of building an appropriate model, and what is more, can lead to an undesirable effect: **overfitting**.

Overfitting is better understood by taking a look at the simple but revealing example in Figure 1.5. The real function of inputs and outputs followed by the system (ground truth) is a simple parabola, nevertheless a regression model fails to fit it properly. In

the first case (top line) the constructed model is providing a much more complex approximation – in terms of non-linearity – of the system than the real ground truth, thus producing inaccurate predictions for regions where training data were unavailable. Conversely, in the second case (bottom line) the model is also fitting a parabola, though the noise in the measurements of training data leads to an inaccurate model.

Two lessons can be learned from this situation. First, striving to find a perfect fit of noisy training data might produce undesired results. Noise can blur the ground truth of a system in a way that the relationship between explanatory features and targets might seem different than what it really is. To place complete trust in the data under this situation can therefore lead to unrealistic models. Second, the training process will ignore those areas of the input space devoid of or lacking enough data, as any changes of the model in them will produce none or little impact in its performance over the training set. The model is hence free to roam in these unpopulated areas, and thus will tend to generate well-behaved solutions near the training points, but any kind of behavior away from them, which can result in more complex models, especially if non-linearities in the model are allowed.

To understand the perils of this complexity, it must be realized that while simple models such as straight lines or parabolas are governed by a reduced set of parameters, the behavior of complex models such as higher order polynomials is ruled by a larger parameter set. Intuitively, a larger number of parameters to adjust involves a harder training task as well as a higher probability of producing a wrong fit. This intuition popularly takes form as the well known **Occam's Razor**, which states that when a phenomenon can be explained by several hypothesis, the simplest of them is more likely to be correct. Though this statement lacks of formal justification in general, it finds a realization in the field of classification models through **statistical learning theory**, also named after its authors as Vapnik–Chervonenkis (VC) theory [7–9]. This field of knowledge studies the theoretical properties of machine learning methods for pattern classification, formalizing the concept of classifier complexity as the **Vapnik–Chervonenkis dimension** (VC dimension).

Definition 1.1 (VC dimension)**.** Given a binary classification model f ruled by a set of parameters θ, f is said to shatter a set of points $(x_1, \cdots, x_N)$ if for every possible assignment of labels to these points there exists a choice of θ such that f incurs in no classification error. Following this, the VC dimension h of f is given as the maximum number of data points that can be shattered by f.

The usefulness of the VC dimension shows up when supposing that the true relationship between inputs x and outputs y is given by some **probability distribution** $P(x, y)$,

so that the available training data are in fact a random draw from this distribution. It would be desirable, thus, to minimize the expected error over all possible outcomes from this distribution. This notion of expected error corresponds to the **risk** of the model [10], and is defined as

$$R[f] = \int \tfrac{1}{2} \left|\text{sign}(f(x)) - y\right| dP(x, y).$$

For a binary classification problem, i.e. $y \in \{-1, +1\}$, the loss employed here equals the classification error in Table 1.1. The problem is that, of course, the distribution P is not known (otherwise the problem would be already solved), and so only the **empirical risk** can be evaluated

$$R_{\text{emp}}[f] = \frac{1}{N} \sum_{i=1}^{N} \tfrac{1}{2} \left|\text{sign}(f(x_i)) - y_i\right|,$$

which is the classification error in the training data set. With these definitions and the measure of complexity given by the VC dimension, the following theorem results [10, 11]

Theorem 1.2. *Suppose a probability distribution $P(x, y)$ producing the data, and a training set of N patterns identically and independently distributed (i.i.d) drawn from P. Then, if the model has VC dimension $h < N$, with probability of at least $1 - \delta$ the following holds* [2]*:*

$$R[f] \leq R_{emp}[f] + \sqrt{\frac{h \ln(\frac{2N}{h} + 1) - \ln(\frac{4}{\delta})}{N}}.$$

In other words, a probabilistic upper bound on the error of the model for unseen data can be given, which depends on the error over the training data but also on the complexity of the model. The more complex the model is, the larger this upper bound results. Therefore, better guarantees on the error for unseen data (i.e. a test set) are obtained by refraining from producing complex models.

Regularization appears as a technique to control the complexity of the resulting model during the training process, as well as to adjust the amount of confidence placed on the training data and thus reduce the impact of noise. Regularization can be formally defined by modifying the training optimization problem as

$$\min_{\theta} L(\theta) + \lambda r(\theta) \tag{1.1}$$

[2] The probabilistic uncertainty comes from the fact that the training set is subject to randomness, since it is a random draw of N patterns from the $P(x, y)$ distribution.

Regularizer	Notation	Formula
L_p or l_p norm	$\|\theta\|_p$	$\left(\sum_i \|\theta_i\|^p\right)^{1/p}$
l_p^p	$\|\theta\|_p^p$	$\left(\sum_i \|\theta_i\|^p\right)$
Block Mixed p, q norm	$\|\|\theta\|\|_{p,q}$	$\left(\sum_a \left(\sum_b \|\theta_{ab}\|^p\right)^{q/p}\right)^{1/q}$
Total–Variation	$\mathrm{TV}_p(\theta)$	$\left(\sum_{i=2}^N \|\theta_i - \theta_{i-1}\|^p\right)^{1/p}$
Indicator function	$\iota_{\mathcal{C}}(\theta)$	$\begin{cases} 0 & \text{if} \quad \theta \in \mathcal{C} \\ +\infty & \text{otherwise} \end{cases}$

TABLE 1.2: A sample of popular choices for the regularizer function.

where the new term stands r for a **regularizer** which penalizes the complexity of the model, and λ is the **regularization parameter** whose value determines whether more importance is given to the regularizer or to the loss function. Similarly to the loss function, a wide range of options have been presented in the literature for the regularizer, a number of them with remarkable success. Some of these regularizers are briefly presented in Table 1.2. The most common of them is recognized as the L_p norm, even more for the specific choices of $p = 1$ or $p = 2$, which imposes a complexity penalty for every model parameter different from zero. In fact, by selecting the squared error loss and the L_1 regularizer for a linear model, the resulting expression

$$\min_{\theta=(w,b)} \|Xw + b - y\|_2^2 + \lambda\|w\|_1. \tag{1.2}$$

can be recognized as the well-established **Lasso** model [12], which modifies the complexity by reducing the number of features used by the model. Note how the bias b is not included in the regularizer, since it does not make use of the input features.

Another extended model is produced by applying an l_2^2 regularizer to the same loss,

$$\min_{\theta=(w,b)} \|Xw + b - y\|_2^2 + \lambda\|w\|_2^2,$$

which is known as the **Regularized Least Squares** model [4].

More complex forms of regularization like **block mixed norms** or **Total–Variation** are employed to impose certain structural properties in the resulting model parameters θ, and find success in those situations where exploitable prior information about the

problem is available. More details about these are presented in Section 1.3. Also, in some settings optimizing an l_p^p regularizer is easier than an L_p norm, and so such variants are sometimes preferred.

Special attention must be paid to the indicator function regularizer, which provides a mean to cast constrained optimization problems as unconstrained ones and vice versa, as

$$\min_{\theta} \quad L(\theta) + \lambda\, \iota_{\mathcal{C}}(\theta) \equiv \begin{cases} \min_{\theta} & L(\theta) \\ \text{s.t.} & \theta \in \mathcal{C} \end{cases} .$$

The rationale behind this transformation is simple: as long as $L(\theta)$ takes a finite value for some choice of θ, the minimum of the problem cannot be any $\theta \notin \mathcal{C}$, as for those the value of the objective function turns out to be $+\infty$. Therefore, constraining the search space to $\theta \in \mathcal{C}$ leaves the minimum unchanged. Similarly, removing then the indicator function from the objective doest not change the minimum, since it only takes the value 0 in the constrained region. Combining the indicator function with other regularizers such as the L_p norm produces constrained versions of them which, depending on the context, might result in more approachable optimization problems. For example, applying the indicator function over the L_2 norm produces

$$\begin{aligned} \min_{\theta} \quad & L(\theta) \\ \text{s.t.} \quad & \|\theta\|_2 \leq \lambda, \end{aligned}$$

i.e. finding the parameters minimizing the loss function constrained to an euclidean ball of radius λ. This particular problem frequently appears as an intermediate step in the context of **Trust Region methods** [13], and so efficient solvers are available for it. More insights into this problem are given in Chapter 4.

1.3 Inducing structure through regularizers

While the basic reason for regularization is to avoid the dangerous overfitting effect, regularization can also be used to impose **structure** into the parameters of the resulting model. By structure it is understood that the model parameters are influenced, either in a direct or indirect way, to present certain properties of interest in the resulting model. Guaranteeing these properties might benefit the exploitation of the model or

even its accuracy if prior information about the system to be modeled is available. Thus, regularization can also be interpreted as including in the model a **prior** over the parameters [4], that is, assuming a particular distribution of the model parameters before taking the data into account, and modifying it only if such data suggests a different distribution. In regularization the strength of such modification is ruled by the value of the regularization parameter λ.

Probably the most well-known example of structure is **sparsity**, and can be defined as the number of model parameters with a value of zero after the training procedure, $\text{card}(\{\theta_i = 0\})$. Sparsity is a desirable property in on–line settings, where for a given entry the model must be able to produce an output under strict time constraints. Supposing, for example, a linear model $\hat{y} = w \cdot x + b$, a sparse w allows to compute the same output as $\hat{y} = \sum_{i|w_i \neq 0} w_i x_i + b$, hence reducing the number of floating point operations needed and therefore the computational time required by the model.

The aim of the already introduced Lasso model (Equation 1.2) [12] is precisely the construction of a sparse linear model, and for this purpose an L_1 regularizer is employed. Indeed, $\mathbf{L}_p$ **regularization** is able to induce sparsity for some choices of p. A clear intuition about this fact can be obtained by taking a glance at the contour lines for different L_p regularizations shown in Figure 1.6. For any choice of p, the L_p regularization can be seen as penalizing the distance from the parameter vector to the origin, measured using an L_p norm. In this way, L_2 regularization penalizes the euclidean distance to the origin, thus producing spherical contour lines.

The shape of these contour lines is notably changed as the value of p tends to 1 or ∞. On the one hand for the $\mathbf{L}_\infty$ case the regularizer turns out to be $\|\theta\|_\infty = \lim_{p\to\infty} \left(\sum_i |\theta_i|^p\right)^{1/p} = \max_i(|\theta_i|)$, implying that only the largest parameter is penalized, as is seen in Figure 1.6(d). This regularization approach is severely prone to produce non-sparse models, as the rest of parameters are allowed to take any value in the range $[0, \max_i(|\theta_i|)]$ without modifying the value of the regularizer; the values minimizing the loss function will then be chosen.

On the other hand, the $\mathbf{L}_1$ **regularizer** produces larger output values in those situations where a number of parameters deviate from zero. This fact is observed in the contour lines of Figure 1.6, where the red point results in larger penalties for smaller p. Conversely, the green point shows exactly the same penalty under any norm. The reason is that the smaller the norm the more contracted the regions become where more than one parameter differ from zero. Therefore, an L_1 regularization fosters the sparsity of the final model.

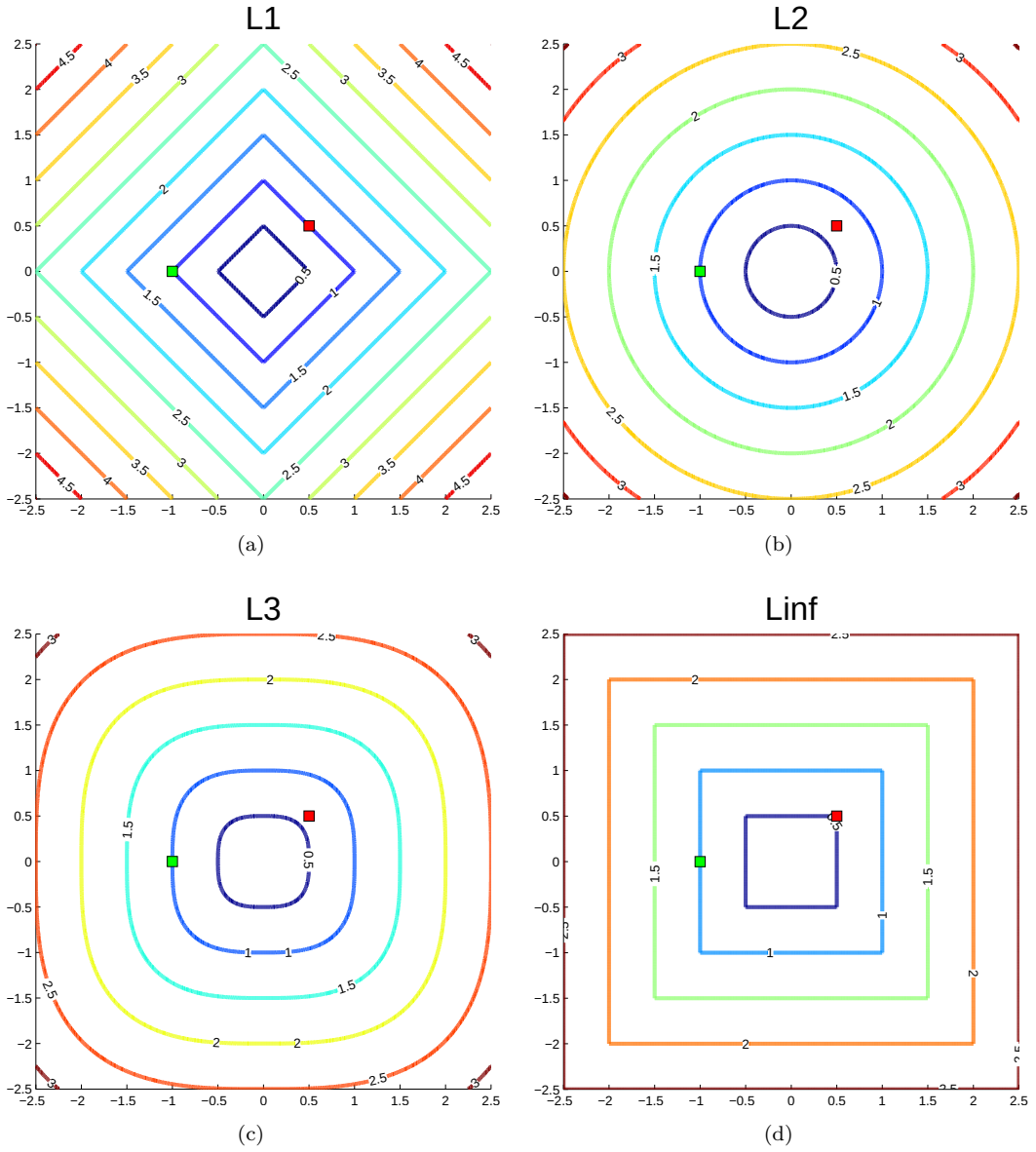


FIGURE 1.6: Contour plots for different choices of the L_p norm regularizer. Reference points $(-1, 0)$ (green) and $(0.5, 0.5)$ (red) are shown as well to stress the different amount of penalty imposed by each norm.

It is also worth noting that $p \in [0, 1)$ choices are also possible, producing contour plots such as the one shown in Figure 1.7 and thus promoting even sparser models. In the limit case known as $\mathbf{L_0}$ **norm** where $p \to 0$ the regularizer turns out to be $\|\theta\|_0 = \lim_{p\to 0} \left(\sum_i |\theta_i|^p\right)^{1/p} = \text{card}(\{\theta_i = 0\})$, which is nothing but a measure of sparsity. Unfortunately this class of regularizers no longer fulfills the properties of a norm [3], and furthermore, produces non-convex optimization problems. Because of this, the resulting problems are significantly harder to optimize, and so regularizers in the range

[3] In spite of this, the L_0 regularizer is still generally named as L_0 norm as a particular case of the L_p norm.

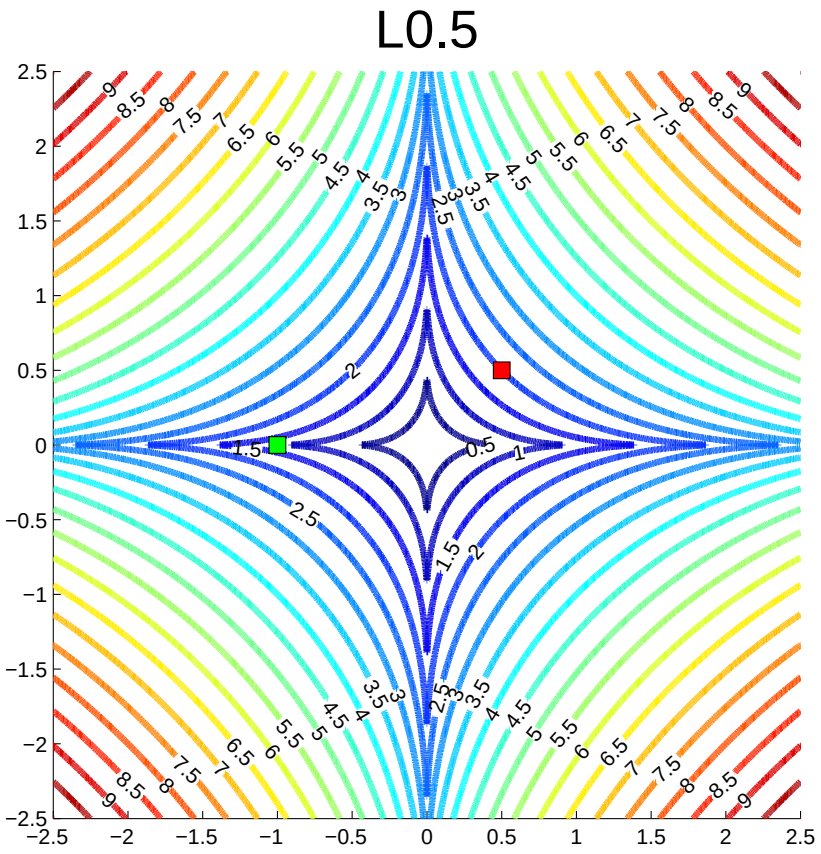


FIGURE 1.7: Contour plot for $L_{0.5}$ regularizer. Reference points coincide with those of Figure 1.6.

$p \in [1, \infty)$ are preferred. It should be noted in passing, though, that some works in approximate L_0 optimization have shown promising results [14].

More complex structural properties can be induced into the model parameters when prior information is available. A good example of this is the **block mixed p, q norm** (Table 1.2), in which the parameters of the model are partitioned in blocks, applying an L_p norm over each block and then an L_q norm over the resulting values. Once again, depending on the choices for the particular p and q norms being used a different flavor of the regularizer is obtained, the most popular being **group lasso** [15], taking $p = 2$, $q = 1$. This choice enforces sparsity among blocks while allowing non-sparse coefficients inside blocks. As a result, full blocks of parameters tend to be zero or non-zero as a whole, thus obtaining group-sparsity in the final model. This setting is of special interest in, for instance, remote sensing applications where the available data is gathered through a number of sensors, each of them providing several measurements. Grouping these measurements by sensor will rule out from the model those sensors producing non-essential information, hence reducing the number of data sources needed by the model, which in turn can produce savings in hardware or communication costs.

A regularizer of special interest for this thesis is the **Total–Variation regularizer**, which is widely used in the context of DNA microarray data classification, and signal and image processing. Instead of penalizing the value of each model parameter, TV penalizes the differences in magnitude of consecutive coefficients. The reasons for using this kind of penalty are given in full detail in Chapter 4, but for now it suffices to say that this allows to encode prior knowledge about the problem, hence allowing to obtain superior models in some areas of application. However, optimizing a model with Total–Variation regularization is not straightforward, and so it is of interest to design efficient optimization algorithms to do so. While proposals abound in the literature, in this thesis methods outperforming the state–of–the–art are developed by making a careful and explicit use of the structure of the resulting optimization problem.

Finally it must also be mentioned that in many applications several regularizers are combined into the same model so as to join their strengths, defining then an optimization problem in the form

$$\min_{\theta} L(\theta) + \sum_{i} \lambda_i r_i(\theta).$$

The tuning of the penalty parameters λ_i allows to configure the stress placed on each regularizer, consequently adapting the structure of the resulting model. Successful models making use of this strategy are the **elastic net** [16], which makes use of an $\mathrm{L}_1 + \mathrm{L}_2$ regularizer, or the **fused lasso** [17], employing a $\mathrm{TV}_1 + \mathrm{L}_1$ regularization. Insights into this model are given later on in Section 4.2.

1.4 Support Vector Machines

One of the most successfully applied regularization instances, as well as another point of interest in this thesis, are the **Support Vector Machines** (SVM) [9, 10, 18]. From the point of view of regularization the SVM model is the conjunction of a hinge loss and a l_2^2 regularizer applied to a linear model, resulting in

$$\min_{w,b} \sum_{i} |1 - y_i(w \cdot x_i + b)|_+ + \frac{\lambda}{2}\|w\|_2^2.$$

It is, however, more common to find its equivalent form

$$\min_{w,b} \tfrac{1}{2}\|w\|_2^2 + C \sum_{i} |1 - y_i(w \cdot x_i + b)|_+, \tag{1.3}$$

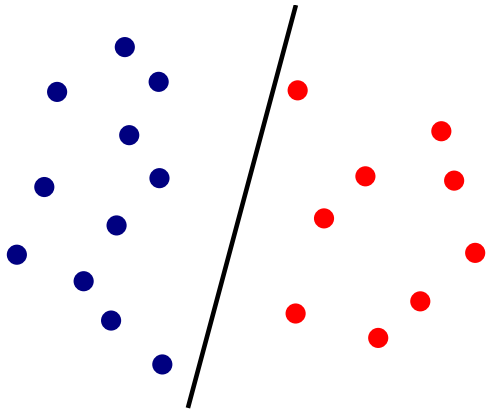


FIGURE 1.8: An example of a separating hyperplane obtained by the perceptron model.

with $C = \frac{1}{\lambda}$ the penalty parameter.

This particular choice of the loss and regularizer functions is not fortuitous. On the contrary, its motivation can be explained by following a geometric, intuitive reasoning. To do so, it is needed to travel back to one of the classical machine learning models: the **perceptron**.

First proposed by Rosenblatt [3, 19], the perceptron is an attempt to emulate the synaptic behavior of a neuron, able to generate an electric pulse when receiving a certain pattern of inputs. This is modeled by assuming that the response of the neuron is a simple linear combination of its inputs, $\hat{y}_i = w \cdot x_i + b$. By means of a training dataset, the coefficients (w, b) are adjusted so that the neuron activates ($\hat{y}_i > 0$) for the desired samples. From the viewpoint of machine learning the perceptron is a model aiming to solve a classification problem by finding a separating hyperplane able to attain zero classification error, i.e.

$$\text{find}_{w,b} \quad \text{s.t.} \quad (w \cdot X_i + b) y_i \ \geq \ 0 \quad \forall\, i.$$

Of course, and as pointed out by the famous work of Minsky and Papert in [20], such thing is only feasible for those problems which are linearly separable in nature, an example of such being shown in Figure 1.8. For problems requiring a non-linear model the perceptron fails to find an acceptable solution, or even a solution at all due to the non-convergent behavior of its associated training algorithm. In order to solve these drawbacks the perceptron was extended to emulate several layers of neuronal activity, each neuron able to generate a non-linear combination of its inputs. The result of this effort was the **Multilayer Perceptron** (MLP) model [3], arguably one of the most applied methods in the history of machine learning. It is, however, a different extension of the perceptron that leads to the SVM model.

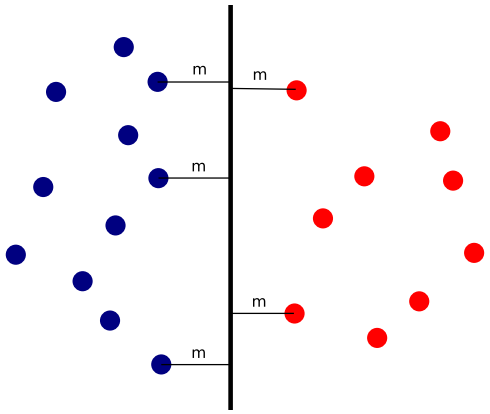


FIGURE 1.9: An example of a maximum margin separating hyperplane.

A single look at the perceptron model is enough to realize that, if a solution to the problem exists, it is probably not unique. Indeed, any hyperplane able to correctly separate both classes will be an optimal solution, and as such, as good as any other "separating" choice of parameters. This assertion, though, proves to be wrong in practice. A clear intuition of this fact is obtained by taking a look at another, also optimal, solution of the perceptron model presented in Figure 1.9: this hyperplane seems to do a better job at separating the classes, even though it achieves the same misclassification error as the previous example. The reason for this is that in the first example the chosen hyperplane lies too close to some of the data points. If the data were noisy – and thus not fully reliable – or a small disturbance in the model parameters took place, this choice of parameters would be prone to errors in out-of-training data. Conversely, the second example would present smaller sensitivity to these issues.

This intuitive idea is formalized under the concept of **large margin classifiers** [21]. Instead of finding any separating hyperplane, the model is modified to seek the classifier with largest margin among those solving the problem. The margin is defined as the distance from the separating hyperplane to the nearest data point, which through geometric arguments turns out to be

$$m = \min_i \quad \frac{y_i(w \cdot x_i + b)}{\|w\|_2}.$$

Seeking a classifier with large margin is desirable, since it is an effective way of controlling the VC dimension of the model [10], and thus, avoiding overfitting (Section 1.2). Following these ideas the perceptron can be extended to seek the largest margin separating hyperplane, as

$$\max_{w,b,m} \quad m \quad \text{s.t.} \quad \frac{y_i(w \cdot x_i + b)}{\|w\|_2} \geq m \quad \forall i,$$

i.e. it is not only required that every point is correctly classified, but also to have a distance to the separating hyperplane larger or equal to the margin. An equivalent and more approachable form of this problem [18] is given by

$$\min_{w,b} \quad \tfrac{1}{2}\|w\|_2^2 \quad \text{s.t. } y_i(w \cdot x_i + b) \geq 1 \quad \forall i,$$

where the margin does not appear explicitly, but is nevertheless maximized owing to the minimization of $\|w\|_2^2$ while keeping every data point at an (unnormalized) distance of 1 from the hyperplane.

Only one more step is needed to arrive to the SVM model, and this step is once again motivated by the possible presence of noise. Note that the present model requires for every data point to be correctly classified as well as to meet the margin requirement. These kind of models are sometimes known as **hard margin classifiers** as an opposite to **soft margin classifiers**, which tolerate violations of the margin condition up to a degree as a mean to produce a better model. Intuition again leads to think that a soft margin classifier could behave better in some situations, e.g. it might well happen that, even if the classification problem is linearly separable, this property is lost in the training set due to noisy data. It is clear, though, that the ground truth is still linear, and so must be the model. Following these ideas the previous model is adapted as

$$\min_{w,b,\xi} \quad \tfrac{1}{2}\|w\|_2^2 + C\sum_i \xi_i \quad \text{s.t. } y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad , \quad \xi_i \geq 0 \quad \forall i, \tag{1.4}$$

where a violation ξ of the margin constraints is allowed. Attention must be paid to the fact that two different objectives are being pursued: obtaining a maximum margin classifier while keeping these violations at bay. These objectives are balanced through the penalty parameter C, which will be needed to be tuned according to the nature of the problem at hand. Large values of C will place most of the weight of the model in the shrinkage of the violations, thus approximating a hard margin classifier. On the other hand small values of C will allow larger violations of the classification constraints for the sake of a larger margin. An example of a solution obtained by this soft margin perceptron is shown in Figure 1.10.

Equivalence between this last formulation and the SVM model in Equation 1.3 can now be established by noting that, at the optimum of Equation 1.4, $\xi_i = 0$ if x_i fulfills the margin condition, while $\xi_i = 1 - y_i(w \cdot x^i + b)$ if it does not. Consequently the substitution $\xi_i = |1 - y_i(w \cdot x^i + b)|_+$ can be applied, thus obtaining Equation 1.3.

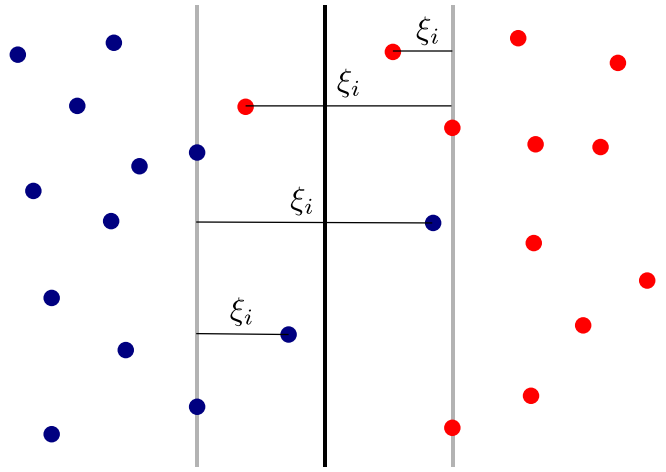


FIGURE 1.10: The soft margin perceptron is able to find a good solution in a situation where noise blurs the linearity of the data.

What can be extracted from this evolutionary trip from the perceptron to the SVM is a collection of sound ideas that justify the use of a hinge loss + squared L_2 regularizer as a reasonable classification model: adaptability, margin maximization and noise tolerance. These ideas, together with a number of useful extensions to be discussed later make the SVM a solid choice for modeling classification, regression and novelty detection problems, either for linear or non-linear data. A more thorough discussion of these topics is presented in Chapter 2. Although a vast number of algorithms have already been proposed in the literature to efficiently solve the optimization problems posed by the SVM in each one of its forms, active research still continues to seek better methods and approaches; this thesis also contributes to this field, as shown in Chapter 3.

1.5 Thesis contributions

As seen in the preceding sections, regularization presents a general framework for learning that prevents overfitting and allows to induce structure in the resulting model parameters. Under this general framework the particular models of the Support Vector Machine and the Total-Variation regularization take place, both of them being well established in their areas of application. It is, however, from the point of view of the optimization problems posed by these models where research interests remain active, seeking more efficient, scalable or practical algorithms to solve them. Following these interests, the contributions of this thesis can be summarized as:

- A thorough analysis of the strong points and weaknesses of the available algorithms for SVM training, in particular the current state–of–the–art Sequential Minimal Optimization (SMO) method for non–linear SVM solving.

- The proposal and analysis of a number of modifications to the SMO algorithm seeking to improve its efficiency.
- An analysis of the structure of the Total-Variation regularization problem from the point of view of convex optimization and proximity operators theories.
- The development of new, efficient algorithms for Total-Variation regularization and some of its applications making use of this analysis.
- Experimental results supporting the improvements in performance of the new methods for SVM and Total-Variation solving.
- Software implementations of such methods.

The rest of this thesis is organized as follows. Chapter 2 presents a detailed review of the theory behind the Support Vector Machine model in its different variants and their associated optimization algorithms. Following this, in Chapter 3 a new series of algorithms to improve SVM solving is presented. Addressing the topic of Total-Variation regularization, Chapter 4 presents the insights and results obtained through the work in this area. Finally, Chapter 5 states the conclusions of this thesis and suggests possible lines of further work. In addition to all of this, Appendix A presents a brief review of the journal publications and conference contributions produced as a result of the work for this thesis.

Chapter 2

Theory and algorithms for Support Vector Machines

"If you only read the books that everyone else is reading,
you can only think what everyone else is thinking."
Haruki Murakami

This chapter consists of a review of a number of advanced though essential concepts in the field of Support Vector Machines, which form the basis needed for the contributions presented in Chapter 3 of this thesis. First, the notions of convexity, duality and kernelization in SVMs are introduced. These properties provide base elements for a deep understanding of the structure and characteristics of the SVM optimization problem. Next, a number of SVM-like models developed in the literature to address other machine learning tasks are introduced, together with a general SVM formulation containing all of them. All the following discussions are made in terms of this generalized formulation. Third, a geometric approach to SVM models is presented, providing further insights into this problem. Finally, a review of SVM training algorithms is included, placing special stress in the Sequential Minimal Optimization method, which is regarded as the current state-of-the-art for non-linear SVM training.

2.1 Properties of the SVM model

As stated in Section 1.4, the Support Vector Machine can be understood as a regularization problem in which the hinge loss is selected for the loss function and a l_2^2 regularizer is applied,

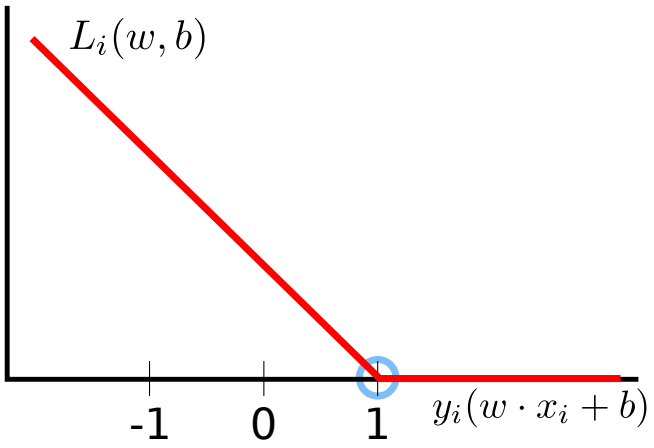


FIGURE 2.1: Depiction of hinge loss function $L_i(w,b)$ for a single pattern x_i. A linear penalty is applied to misclassified patterns, but also to correctly classified ones if the classifier is not "sure enough" (output ≥ 1) of its prediction. A point of non-differentiability is present at $y_i(w \cdot x_i + b) = 1$.

$$\min_{w,b} \tfrac{1}{2}\|w\|_2^2 + C\sum_i |1 - y_i(w \cdot x_i + b)|_+. \tag{2.1}$$

The motivations behind the choice of this particular model have already been discussed. It should be realized now that selecting a model for the machine learning task at hand is only the first step of the model construction process. Once the model to use is expressed as an optimization problem (as is Equation 2.1) an appropriate optimization algorithm should be devised to solve this problem and produce the model parameters. Failing to find an adequate algorithm might well result in inefficient or unstable methods unfit for application in any real purpose scenario. It is therefore mandatory to apply or design proper optimization algorithms in order to obtain practical models.

From the point of view of optimization theory, the SVM model can be regarded as a **quadratic problem** (QP), as the function to minimize (or **objective function**, $\frac{1}{2}\|w\|_2^2 + C\sum_i |1 - y_i(w \cdot x_i + b)|_+$) is a quadratic function w.r.t. any of the variables to optimize (w,b). This class of problems is, in general, relatively easy to solve through the theory of convex optimization, to be addressed later. Unfortunately, the objective function here also features the **non-smooth** property, or in other words, it is not differentiable (smooth) at every point of its domain. This lack of smoothness originates from the hinge loss, which as shown in Figure 2.1 is non–differentiable at the point $y_i(w \cdot x_i + b) = 1$. Non-smooth problems are, in general, harder to solve than smooth problems, though as shown later this can be circumvented easily by rewriting the problem in a more suitable form.

In what follows it is shown how the convexity of the SVM problem allows to make use of **duality**, and so exploit the structure of the problem. The **kernelization** strategy

is also introduced, which allows to produce non-linear models out of the very same SVM problem. All of these concepts prove to be essential for the proper design of SVM optimization algorithms.

2.1.1 Convexity and duality

As commented before, the SVM optimization problem falls into the category of the well studied **convex optimization** problems [22, 23]. A convex optimization problem can be expressed in the form

$$\begin{aligned} \min_{x} \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq b_i, \quad i = 1, \ldots, m, \end{aligned} \tag{2.2}$$

where the objective function f_0 and every **constraint** f_i is a convex function, i.e. meets the convexity property $f_i(\alpha x+\beta y) \leq \alpha f_i(x)+\beta f_i(y)$ for any $\alpha, \beta \geq 0$, $\alpha+\beta = 1$. Equality constraints in the form $f_i(x) = b_i$ for $f_i(x)$ a linear function can also be managed by transforming them into two inequality constraints as $f_i(x) \leq b_i$ and $-f_i(x) \leq -b_i$. Additionally, the domain of x needs to be convex (this is met for the general $x \in \mathbb{R}^N$).

This class of problems is known to have nice properties which are useful for their optimization; for instance, any extreme point is necessarily also a global minimum. Another key property of convex problems is **duality**. For every convex **primal problem** $\mathcal{P}(x)$ an alternative **dual problem** $\mathcal{D}(\lambda)$ can be defined such that at the optimum both attain the same objective value, that is, $\mathcal{P}(x^*) = \mathcal{D}(\lambda^*)$. What is more, means to obtain x^* from λ^* can be derived. Hence, duality effectively allows to solve an alternative problem instead of the original one, also obtaining the desired solution.

Duality can be easily understood by approaching it from the point of view of the **Lagrange coefficients** or multipliers. Given a convex problem in the form of Equation 2.2, its **Lagrangian** is defined as

$$L(x, \lambda) = f_0(x) + \sum_i \lambda_i (f_i(x) - b_i),$$

where the introduced λ variables are the Lagrange coefficients. Using the Lagrangian, an equivalent optimization problem to Equation 2.2 can be written as

$$\left\{ \begin{array}{cc} \min_x \max_\lambda & L(x,\lambda) \\ \text{s.t.} & \lambda \geq 0 \end{array} \right\} \equiv \left\{ \begin{array}{cc} \min_x \max_\lambda & f_0(x) + \sum_i \lambda_i (f_i(x) - b_i) \\ \text{s.t.} & \lambda \geq 0 \end{array} \right\}. \quad (2.3)$$

The equivalence follows by noticing that:

- If some $f_i(x) > b_i$, then $\max_\lambda \{f_0(x) + \sum_i \lambda_i (f_i(x) - b_i)\} \to \infty$ by selecting $\lambda_i \to +\infty$. Therefore $\max_\lambda L(x,\lambda) = +\infty$.
- If every $f_i(x) \leq b_i$, then $\max_\lambda \{f_0(x) + \sum_i \lambda_i (f_i(x) - b_i)\} = f_0(x)$ with $\lambda_i = 0 \; \forall \; i$. Thus, $\max_\lambda L(x,\lambda) = f_0(x)$.

It is then clear that at the optimum Equation 2.2 and Equation 2.3 share the same solution, as infeasible x choices for Equation 2.2 result in infinity (and thus suboptimal) objective values for Equation 2.3.

The resultant problem from the inclusion of the Lagrange coefficients is a **saddle point problem**, in which the solution consists in an equilibrium point of the maximization and minimization problems. A great advantage of this equivalent formulation of the primal problem is the disappearance of every primal constraint, presenting only very simple positivity constraints instead. However, the max/min objective is in general difficult to deal with. Fortunately, a further transforming step leads to the more approachable dual problem, this step being given by **Sion's minimax theorem** [24], which can be summarized as follows.

Theorem 2.1 (Sion's minimax theorem)**.** *Let X, Y be compact convex subsets of a linear topological space, f a real-valued function in $X \times Y$ with*

- *$f(x,\cdot)$ upper semicontinuous and quasiconcave on $Y \; \forall \; x \in X$,*
- *$f(\cdot,y)$ lower semicontinuous and quasiconvex on $X \; \forall \; y \in Y$,*

then

$$\min_{x \in X} \max_{y \in Y} f(x,y) = \max_{y \in Y} \min_{x \in X} f(x,y),$$

i.e. the minimum and maximum operators can be swapped.

A function f is quasiconvex if $f(\lambda x + (1-\lambda)y) \leq \max\{f(x), f(y)\}\ \forall\ x, y,\ \lambda \in [0,1]$, and quasiconcave if $f(\lambda x + (1-\lambda)y) \geq \min\{f(x), f(y)\}$. These concepts can be regarded as generalizations of the convexity and concavity properties, and thus they are immediately met for a convex problem. Lower (or upper) semicontinuity is a weaker assumption than continuity, as it only requires, roughly speaking, that for any choice $x \in \mathbb{R}$ and every point x_0 in a neighborhood of x, $f(x_0) \geq f(x)$ (or $f(x_0) \leq f(x)$). Lower semicontinuity is also always met for convex functions.

Besides this, the compactness requirements reduce to the feasible sets being closed and bounded for the case of an euclidean space, like the one addressed here. The space of the x for a general convex problem, though, is not necessarily bounded, but this technicality is just needed to avoid unbounded solutions (tending to $x, y \to \infty$) to the minimization / maximization optimization problems above. Since in a convex problem a finite optimum is guaranteed, this is not an issue. Consequently the theorem can be applied in the saddle point problem, producing an equivalent formulation of Equation 2.2

$$\begin{aligned} \max_{\lambda} \min_{x} \quad & f_0(x) + \sum_i \lambda_i (f_i(x) - b), \\ \text{s.t.} \quad & \lambda \geq 0, \end{aligned} \tag{2.4}$$

which is the dual problem. While at first glance this problem may appear to pose a similar difficulty to the previous saddle point problem, it might well happen that the inner minimization problem can be solved in closed form, greatly simplifying the task at hand. This is just the situation arising in the SVM problem, as it is shown in the following. Before that, though, some additional observations of interest should be made about duality.

The applicability of Sion's minimax theorem ensures that at the optimum both primal and dual formulations present identical objective values, $\mathcal{P}(x^*) = \mathcal{D}(\lambda^*)$. This situation is commonly known as primal-dual problems presenting no **duality gap**, or having **strong duality**. However, for more general primal optimization problems in which, for instance, the quasiconvexity assumptions are not met, a duality gap is bound to appear. Nevertheless even in these defective situations a useful **weak duality** property can be stated

Proposition 2.2 (Weak duality)**.** *Given a feasible (not necessarily optimal) solution $\hat{x}$ of the primal problem and a feasible (not necessarily optimal) solution $\hat{\lambda}$ of the dual problem,*

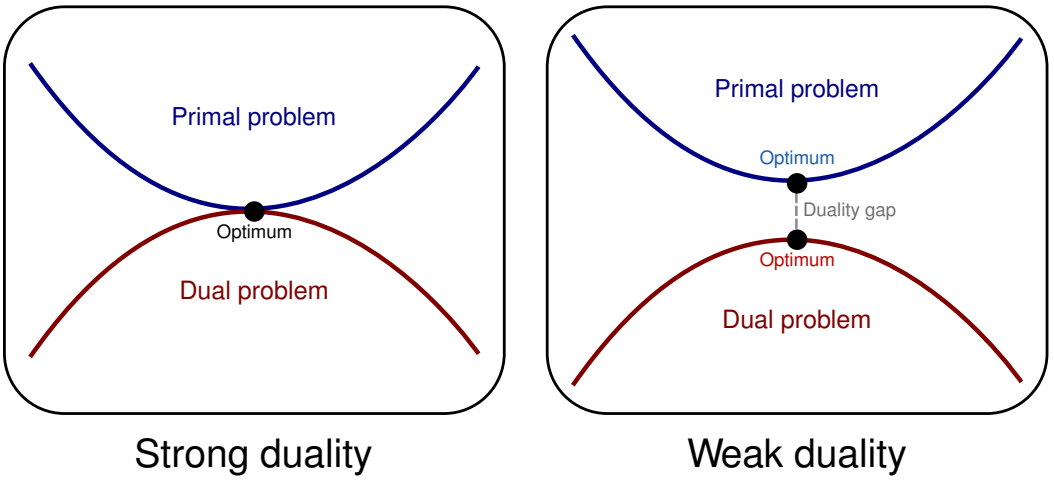


FIGURE 2.2: Depiction of the relationship between primal and dual problems. By the weak duality theorem, the dual problem is always upper bounded by the primal problem. Only under strong duality the objective value of both problems coincides at the optimum.

$$\mathcal{D}(\hat{\lambda}) \leq \mathcal{P}(\hat{x}).$$

Proof. The value of the dual problem (Equation 2.4) for a given feasible $\hat{\lambda}$ turns out to be $\mathcal{D}(\hat{\lambda}) = \min_x f_0(x) + \sum_i \hat{\lambda}_i(f_i(x) - b) = \min_x L(x, \hat{\lambda})$. It is clear that, whatever the feasible primal $\hat{x}$ given, $\min_x L(x, \hat{\lambda}) \leq L(\hat{x}, \hat{\lambda})$. Now, the value of the primal problem for this given $\hat{x}$ would be $\mathcal{P}(\hat{x}) = \max_{\lambda \geq 0} L(\lambda, \hat{x})$, and it is immediate that $L(\hat{x}, \hat{\lambda}) \leq \max_{\lambda \geq 0} L(\lambda, \hat{x})$. Therefore

$$\mathcal{D}(\hat{\lambda}) = \min_x L(x, \hat{\lambda}) \leq L(\hat{x}, \hat{\lambda}) \leq \max_{\lambda \geq 0} L(\lambda, \hat{x}) = \mathcal{P}(\hat{x}).$$

□

By noting that the optimal solutions λ^* and x^* must be feasible solutions as well, the following useful corollary is obtained:

Corollary 2.3. *The optimal value of the dual is upper bounded by the value of the primal, that is*

$$\mathcal{D}(\lambda^*) \leq \mathcal{P}(x^*).$$

Therefore, for any primal optimization problem there is a guarantee that its corresponding dual is a lower bound on the primal's optimal value, and vice versa. A depiction of this effect is shown in Figure 2.2. If only weak duality is met, the difference $\mathcal{P}(x^*) - \mathcal{D}(\lambda^*)$

is the amount of dual gap, which can be used as a measurement of the distance between both problems, and so of the approximation quality of a solution obtained through the dual problem. Even if strong duality is present the amount of duality gap can be used during the optimization procedure to obtain a quality measure of the solution found by the optimization algorithm so far. As shown later in the text, this turns out to be an useful stopping criterion in SVM optimization methods.

Addressing now the SVM problem under the point of view of these convex optimization tools, consider it in its constrained form (as seen in Equation 1.4)

$$\begin{aligned} \min_{w,b,\xi} \quad & \tfrac{1}{2}\|w\|_2^2 + C\sum_i \xi_i, \\ \text{s.t.} \quad & \begin{cases} y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall\, i, \\ \xi_i \geq 0 \quad \forall\, i. \end{cases} \end{aligned}$$

It is easy to see that the objective function is convex, while the constraints are linear, thus fulfilling the properties of a convex optimization problem. Following the previous derivation for a general convex problem, the Lagrangian of the SVM takes the form

$$L(w, b, \xi, \alpha, \nu) = \tfrac{1}{2}\|w\|_2^2 + C\sum_i \xi_i + \sum_i \alpha_i(1 - \xi_1 - y_i(w \cdot x_i + b)) - \sum_i \nu_i \xi_i,$$

and so the dual problem can be stated as

$$\begin{aligned} \max_{\alpha,\nu} \min_{w,b,\xi} \quad & \tfrac{1}{2}\|w\|_2^2 + C\sum_i \xi_i + \sum_i \alpha_i(1 - \xi_i - y_i(w \cdot x_i + b)) - \sum_i \nu_i \xi_i, \\ \text{s.t.} \quad & \alpha_i, \nu_i \geq 0 \quad \forall\, i. \end{aligned} \tag{2.5}$$

The inner problem becomes an unconstrained problem, and so it can be solved by finding a choice of variables (w^*, b^*, ξ^*) for which the gradient becomes 0, i.e.

$$\nabla_{w,b,\xi} L(w^*, b^*, \xi^*, \cdot, \cdot) = 0.$$

This is done trivially as

$$\begin{aligned} \nabla_w L = w^* - \sum_i \alpha_i y_i x_i = 0 \quad &\rightarrow \quad w^* = \sum_i \alpha_i y_i x_i, \\ \frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0 \quad &\rightarrow \quad \sum_i \alpha_i y_i = 0, \\ \frac{\partial L}{\partial \xi_i} = C - \alpha_i - \nu_i = 0 \quad &\rightarrow \quad \nu_i = C - \alpha_i. \end{aligned} \tag{2.6}$$

Substituting these expressions into Equation 2.5 produces

$$\begin{aligned} \max_{\alpha} \quad & -\tfrac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j + \sum_i \alpha_i, \\ \text{s.t.} \quad & \begin{cases} 0 \leq \alpha_i \leq C \quad \forall i, \\ \sum_i \alpha_i y_i = 0, \end{cases} \end{aligned}$$

where the additional constraints $\alpha_i \leq C \ \forall i$ come from the fact that $\nu_i = C - \alpha_i$ and $\alpha_i, \nu_i \geq 0 \, \forall i$. This simplified dual problem can be expressed more compactly by adapting a vectorial notation in the form

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2} \alpha^T Q \alpha - e^T \alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = 0, \\ 0 \leq \alpha \leq C, \end{cases} \end{aligned} \tag{2.7}$$

where e is a vector of ones and the matrix Q is formed by $Q_{ij} = y_i y_j K_{ij}$ with $K_{ij} = x_i \cdot x_j$, or alternatively as $Q = I_y K I_y$, with I_y a diagonal matrix with the labels vector y as its main diagonal. As will be discussed later, the matrix K plays a crucial role in the non–linearization of the SVM model, though for now it is simply formed by the dot products of every pair of training patterns.

The resultant SVM dual problem is also a convex optimization problem; however, it features simpler constraints in the form of a single equality constraint and a set of **box constraints** over the α coefficients. Conversely, the quadratic term in the objective function is more complex because of the presence of the matrix Q. Depending on the situation solving the primal or the dual is preferred, and a large number of algorithms following either of these approaches can be found in the literature: a description of the most representative methods is presented in Section 2.4.

If the dual approach is followed, a procedure to recover the primal variables w^*, b^* from the dual solution α^* is needed. w^* can be computed straightforwardly as a function of α^* by noticing that by Equation 2.6, $w^* = \sum_i \alpha_i^* y_i x_i$. Equivalently this formula can be plugged into the classification function itself as $f(x) = w^* \cdot x + b^* = \sum_i \alpha_i^* y_i x_i \cdot x + b^*$, thus not needing an explicit representation of w^*. Also, it is of special relevance that in practice only a subset of the α^* coefficients turn out to be $\alpha_i^* > 0$, i.e. α^* is usually sparse. This allows to rewrite w^* just in terms of these so-called **support vectors** (SV), as $w^* = \sum_{\alpha_i > 0} \alpha_i^* y_i x_i$, as well as the classification function as $f(x) = w^* \cdot x + b^* = \sum_{\alpha_i > 0} \alpha_i^* y_i x_i \cdot x + b^*$. As shown later in the text, these facts prove to be essential in order to allow the SVM to obtain nonlinear models.

Obtaining b^*, though, requires invoking additional arguments establishing links between the primal and dual formulations. The links to be used here are the **Karush-Kuhn-Tucker conditions** (or KKT conditions) [23, 25], which provide necessary conditions of optimality for nonlinear constrained programs, namely

Theorem 2.4 (Karush-Kuhn-Tucker conditions)**.** *Given a nonlinear optimization problem in the form*

$$\begin{aligned} \min_x \quad & f(x), \\ s.t. \quad & \begin{cases} g_i(x) \leq 0 \ , \ i = 1, \ldots, m, \\ h_j(x) = 0 \ , \ j = 1, \ldots, l, \end{cases} \end{aligned}$$

with $f, g_i, h_j \ \forall \ i, j$ functions from $\mathbb{R}^n$ to $\mathbb{R}$ and continuously differentiable at a point x^* satisfying some regularity conditions (see below). Then x^* is a local minimum if $\exists$ $\mu_i (i = 1, \ldots, m), \lambda_j (j = 1, \ldots, l)$ KKT multipliers such that the following conditions are met:

Stationarity

- $\nabla f(x^*) + \sum_{i=1}^m \mu_i \nabla g_i(x^*) + \sum_{j=1}^l \lambda_j \nabla h_j(x^*) = 0.$

Primal feasibility

- $g_i(x^*) \leq 0 \ \forall \ i = 1, \ldots, m,$
- $h_j(x^*) = 0 \ \forall \ j = 1, \ldots, l.$

Dual feasibility

- $\mu_i \geq 0 \ \forall \ i = 1, \ldots, m.$

Complementary slackness

- $\mu_i g_i(x^*) = 0 \ \forall \ i = 1, \ldots, m.$

Several **regularity conditions** can be tested on the point x^* to qualify for the application of the KKT conditions, depending on the particular kind of problem at hand. For a convex problem the most commonly applied is the **Slater condition**, which requires

the existence of at least one point x such that $h_j(x) = 0 \ \forall \ j$ and $g_i(x) < 0 \ \forall \ i$ active in x^* (i.e., $g_i(x^*) = 0$). This generally reduces to checking that the interior of the primal feasible set is not empty, i.e. there are strictly feasible points ($g_i(x) < 0 \ \forall \ i$).

Note how the so-called KKT multipliers are in fact the variables of the dual problem, and so these conditions establish relationships that must hold at any local minimum / maximum of the primal / dual problem. Even more, the following corollary strengthens these relationships in the case of a class of convex problems.

Corollary 2.5. *If the objective function f and the inequality constraints g_i are continuously differentiable convex functions and the equality constraints h_j are affine functions, the KKT conditions are sufficient for optimality.*

The SVM primal problem fully meets these requirements, and so the KKT conditions provide a reliable link between primal and dual formulations, as well as a guarantee of global optimality. In fact, rewriting these conditions for the SVM problem results in

Stationarity	**Primal feasibility**
• $w = \sum_i \alpha_i y_i x_i$,	• $y_i(w \cdot x_i + b) \geq 1 - \xi_i \ \forall \ i$,
• $\sum_i \alpha_i y_i = 0$,	• $\xi_i \geq 0 \ \forall \ i$.
• $\nu_i = C - \alpha_i \ \forall \ i$.	
Dual feasibility	**Complementary slackness**
• $\alpha_i \geq 0 \ \forall \ i$,	• $\alpha_i(1 - \xi_i - y_i(w \cdot x + b)) = 0 \ \forall \ i$,
• $\nu_i \geq 0 \ \forall \ i$.	• $\xi_i \nu_i = 0 \ \forall \ i$.

Once the optimal α_i^* have been obtained, the primal b^* can be recovered by realizing that for those $\alpha_i^* \in (0, C)$ we have

- As $\alpha_i^* < C$, $\nu_i^* > 0$, and so $\xi_i^* = 0$.
- As $\alpha_i^* > 0$, $1 - \xi_i^* - y_i(w \cdot x + b^*) = 0$.
- By joining these facts, $1 - y_i(w^* \cdot x + b^*) = 0$, and so $b^* = y_i - w^* \cdot x_i$.

Thus, if any of the α_i in the dual solution lies in the range $(0, C)$ (something that happens often in practice), the primal bias b can be recovered using this strategy. It is however generally suggested to compute b as an average of this estimation for every $\alpha_i \in (0, C)$.

Algorithm 1 Generic SVM dual solver

Inputs: training set (x, y), penalty parameter C.
Solve dual problem 2.7 to obtain optimal α^*:

$$\begin{array}{ll} \min_\alpha & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha, \\ \text{s.t.} & \begin{cases} 0 \leq \alpha \leq C \\ \alpha \cdot y = 0 \end{cases} . \end{array}$$

Recover primal solutions as

$w^* = \sum_i \alpha_i^* y_i x_i$,
$b^* = \frac{1}{\text{card}(\{\alpha_i \in (0,C)\})} \sum_{\alpha_i \in (0,C)} (y_i - w^* \cdot x_i)$.

Return (w^*, b^*).

As a wrap-up of this section, Algorithm 1 briefly sketches the procedure to obtain a solution for the SVM model through its dual problem. The resulting procedure turns out to be remarkably simple in spite of the seemingly involved theory used. This fact makes the SVM dual problem an appealing reformulation of the "standard" SVM problem, and as such a number of successful algorithms make use of it.

2.1.2 Kernelization

As presented, the Support Vector Machine is supported by solid foundations from the field of regularization, making it an attractive model for classification. However, a serious drawback of this model (as presented so far) is its **linearity**. Recall that the decision function of the SVM is given by the expression $f(x) = w \cdot x + b$, which is nothing but a hyperplane bisecting the space of the inputs x. This kind of decision function, shared by its predecessor the perceptron [19], completely fails to produce an accurate model in those situations where the data is **nonlinear**. For the perceptron model this drawback was overcome by the invention of the Multilayer Perceptron (MLP) model [3], which introduced nonlinear capabilities into the model by stacking several layers of perceptrons with nonlinear transformations in-between. Unfortunately this approach is not applicable in the SVM model, as nice properties such as margin maximization would be partially lost. However, an alternative way to introduce nonlinearity is possible.

Suppose the training data is given as a set of patterns with d explanatory features, so that each pattern x_i lies in the **input space** $\mathbb{R}^d$. Now define a **mapping function** $\Phi : \mathbb{R}^d \to \mathbb{F}$ such that it maps the input patterns to an enlarged **feature space** $\mathbb{F}$ (for instance, $\mathbb{F} \equiv \mathbb{R}^D$ with $D \gg d$). If the mapping function is nonlinear and, instead of using the original x_i as the training set, the mapped $\Phi(x_i)$ are used to build a linear model, the model will still remain linear with respect to the $\Phi(x_i)$, but nonlinear with respect to the input data x_i. Or, in other words, the model will be linear in the feature space, though nonlinear in the input space. An example of this effect is shown in

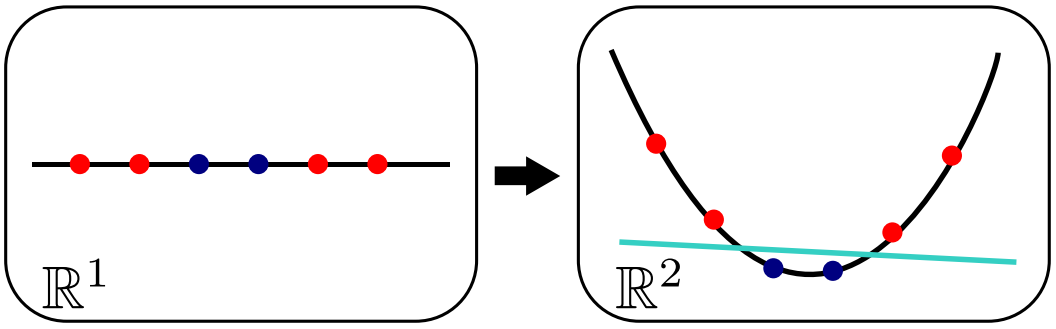


FIGURE 2.3: A simple example of how embedding the input data into a larger space can effectively cast a nonlinear problem into a linear one. Although the data is nonlinear in their original input space ($\mathbb{R}^1$), adding the nonlinear feature x^2 produces a linearly separable problem in the feature space $\mathbb{R}^2$.

Figure 2.3, where a linear classifier is able to solve a nonlinear model by following this approach.

Of course, the effectiveness of this strategy is completely determined by the choice of an appropriate mapping function. Furthermore, the mapped data should be stored in memory or recomputed every time it is needed. If the feature space is very large or the evaluation of the mapping function is costly, this might raise memory or computational issues. These problems can be partially solved by observing that in the dual formulation of the SVM

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T Q\alpha + e^T\alpha, \\ \text{s.t.} \quad & \begin{cases} 0 \leq \alpha \leq C \\ \alpha \cdot y = 0 \end{cases}, \end{aligned}$$

the training data does not show up explicitly, but as part of the computation of Q (recall $Q_{ij} = y_i y_j K_{ij}$ with $K_{ij} = x_i \cdot x_j$). Introducing the mapping Φ, thus, only involves modifying the computation of K as $K_{ij} = \Phi(x_i) \cdot \Phi(x_j)$. By further defining the **kernel function** $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$ it is realized that the matrix K is just $K_{ij} = k(x_i, x_j)$. Owing to this, this matrix is commonly named as the **kernel matrix**. It should be realized that because of this, there is no need to compute explicitly the mapping Φ as long as it is possible to calculate the kernel function k or the kernel matrix K directly. This removes the necessity of storing the mappings $\Phi(x_i)$ explicitly, and may also lead to savings in computation times if given a mapping Φ the inner product $\Phi(x_i) \cdot \Phi(x_j)$ can be simplified down.

Needless to say, the problem of finding a suitable mapping function has only been replaced by the problem of finding an appropriate kernel function. What is more, the

kernel function must be decomposable into an inner product of mappings ($k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$) in order to make sense. This task, seemingly daunting at first glance, can be approached by using **Mercer's theorem** [10, 26]. Before introducing it, the definition of positive semi-definiteness for a kernel function is presented:

Definition 2.6. (Positive semi-definite kernel) A kernel function $k(x_i, x_j)$ is positive semi-definite if for any other function f at least square integrable

$$\int_{\mathcal{X}\times\mathcal{X}} k(x_i, x_j) f(x_i) f(x_j) dx_i dx_j \geq 0,$$

where $\mathcal{X}$ is the set of all possible x.

Alternatively and considering a finite set $\mathcal{X}$ (as in a training set), a kernel function is positive semi-definite if its associated kernel matrix K for any $\mathcal{X}$ is positive semi-definite ($K \succcurlyeq 0$), i.e. for every $x \neq 0$

$$x^T K x \geq 0.$$

With this in mind, Mercer's theorem can be stated as follows

Theorem 2.7 (Mercer's theorem)**.** *If a scalar function $k(x_i, x_j)$ is positive semi-definite, then it can always be decomposed as an inner product $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$ where $\Phi(x) \in \mathbb{F}$, $\mathbb{F}$ a Hilbert space.*

Although the full theory behind this theorem and Hilbert spaces is not straightforward, in practice there is no need to resort to it. It suffices to say that any Hilbert space is characterized by having a well–defined inner product operation. Mercer's theorem, hence, can be used as a tool that allows to define a kernel function with guarantees of fulfilling the necessary properties. Several examples of kernel functions following this theorem can be found in the literature; some of the most popular ones are presented in Table 2.1. In practice and for general applications, the **gaussian kernel** is preferred for its versatility. To obtain a better fit of the kernel function to the modeling problem at hand, the parameters of the function are usually optimized to maximize the model accuracy in a validation set. More details about this procedure are given in the experimental sections of this thesis.

It is therefore possible to introduce nonlinearity into the SVM model by just using a kernel matrix K computed through a kernel function, as the structure of the optimization problem to solve remains identical. This method to transform a linear model into a nonlinear one through the use of kernels is known as the **kernel trick**, and has also been applied to other models and techniques in the field of machine learning, such as the Perceptron [27], Nearest Neighbors [28] or Principal Component Analysis [29] to name a few.

Kernel function	Formula	Parameters
Linear	$x_i \cdot x_j$	None
Homogeneous polynomial	$(x_i \cdot x_j)^p$	p polynomial degree
Inhomogeneous polynomial	$(x_i \cdot x_j + 1)^p$	p polynomial degree
Gaussian	$e^{\left(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}\right)}$	σ kernel width

Table 2.1: Some examples of popular kernel functions.

A word of warning about using the kernel trick under the primal formulation of the SVM is needed. Recall that the primal solution w relates to the dual solution α through the formula $w = \sum_i \alpha_i y_i x_i$. When introducing the mapping function this equivalence turns out be $w = \sum_i \alpha_i y_i \Phi(x_i)$ and so w lies in the feature space generated by the mapping Φ (as expected). This fact implies that while solving the dual formulation under the kernel trick is straightforward, addressing the primal formulation can lead to complications derived from the possibly very large dimensionality of w. Even worse, the mapping Φ might be unknown, as Mercer's theorem does not provide of a formula for Φ (only proves its existence). Therefore it is sensible to consider only the dual formulation for the non–linearized SVM model, and as will be presented later in the text, optimization algorithms for this model generally do so.

2.2 Modifications of the SVM model

While the Support Vector Machine model was originally intended for classification tasks, analog models following similar principles have also been developed to address other machine learning tasks, such as regression or density estimation. Nice properties like margin maximization, convexity and duality are shared among these models, and allow to study all of them under a common framework. In fact, they all can be regarded as particular cases of a generalized SVM model. This observation is crucial for the design of optimization algorithms addressing the SVM problem, as being able to solve this generalized formulation immediately provides effective ways to deal with all the underlying particular models.

This section contains a review of the SVM model extensions of interest for this thesis, concluding with the presentation of the general SVM formulation by Chang et al. [30]. Additionally, and as a first minor contribution of this thesis, this formulation is further generalized to include as well the Least-Squares Support Vector Machine model for classification and regression [31].

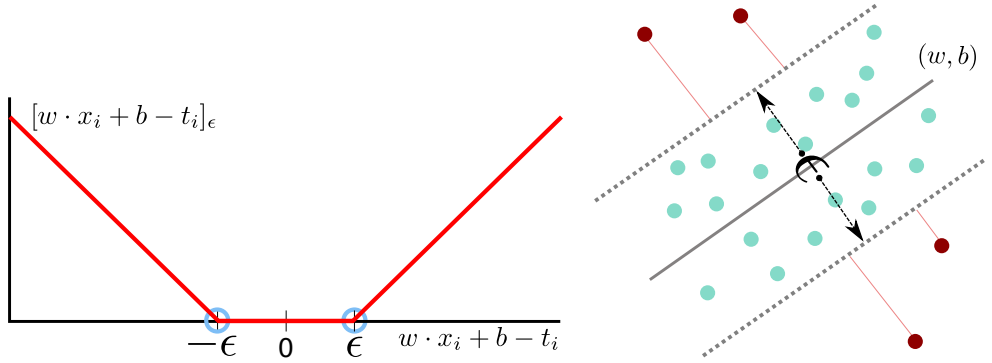


FIGURE 2.4: a) Depiction of ϵ-insensitive loss function for a single pattern x_i. b) Depiction of the ϵ-tube formed by the SVR model.

Note that when discussing these alternative formulations the original SVM model is sometimes referred in the literature as **Support Vector Classification** (SVC).

2.2.1 Support Vector Regression

One of the most popular SVM model modifications strives to solve regression tasks as a natural extension of the classification problem [32, 33]. This model, known as **Support Vector Regression** (SVR), is another instance of regularization, showing the loss plus regularizer objective function

$$\min_{w,b} \tfrac{1}{2}\|w\|_2^2 + C\sum_i [w \cdot x_i + b - t_i]_\epsilon,$$

for explanatory features and targets (x_i, t_i). The loss function of choice here is the **ϵ-insensitive loss**, which takes the form $[z]_\epsilon = \max\{0, |z| - \epsilon\}$. A depiction of it is shown in Figure 2.4(a). The insensitivity parameter ϵ acts together with C as a safeguard against overfitting: not penalizing small errors avoids unnecessarily increasing the model complexity in order to obtain a perfect fit over the training set. This also leads the model to produce a solution (w^*, b^*) around which training samples gather at a distance no larger than ϵ: the so-called **ϵ-tube** (see Figure 2.4(b)).

In analog with the hinge loss used in SVM for classification, the ϵ-insensitive loss presents points of non-differentiability. To handle the difficulties produced by this, an alternative constrained formulation of SVR is preferred,

$$\begin{array}{ll} \min\limits_{w,b,\xi,\xi^*} & \frac{1}{2}\|w\|_2^2 + C\sum\limits_i(\xi_i + \xi_i^*), \\ \text{s.t.} & \left\{ \begin{array}{l} t_i - w\cdot x_i - b \leq \epsilon + \xi_i \ \forall\ i \\ w\cdot x_i + b - t_i \leq \epsilon + \xi_i^* \ \forall\ i \\ \xi, \xi^* \geq 0 \end{array} \right. , \end{array}$$

where the slack variables ξ, ξ^* stand for the loss produced by training points outside the ϵ-tube. As in SVM, this optimization problem is convex, and so a dual formulation allowing the application of the kernel trick is also possible, resulting in the dual problem

$$\begin{array}{ll} \min\limits_{\alpha,\alpha^*} & \frac{1}{2}(\alpha - \alpha^*)^T K(\alpha - \alpha^*) + \epsilon(\alpha + \alpha^*)\cdot e - (\alpha - \alpha^*)\cdot t, \\ \text{s.t.} & \left\{ \begin{array}{l} \alpha\cdot e = \alpha^*\cdot e \\ 0 \leq \alpha, \alpha^* \leq C \end{array} \right. . \end{array}$$

After solving the dual problem the primal solution can be recovered through the use of duality and the KKT conditions as

$$\begin{aligned} w &= \sum_i(\alpha_i - \alpha_i^*)x_i, \\ b &= t_i - w\cdot x_i - \epsilon\ \forall\ \alpha_i \in (0, C), \\ b &= t_i - w\cdot x_i + \epsilon\ \forall\ \alpha_i^* \in (0, C). \end{aligned}$$

Again, several formulæ allow to recover b, and so in practice an average of them is used.

2.2.2 One-Class SVM

Yet another modification of the Support Vector Machine model allows to address the task of density estimation: the so-called **One-Class Support Vector Machine**. First proposed by Schölkopf et al. [34], the problem to be solved can be written in loss plus regularizer form as

$$\min_{w,\rho} \frac{1}{2}\|w\|_2^2 - \rho + \frac{1}{\nu N}\sum_i[\rho - w\cdot x_i]_+$$

for a training set $x_1, \ldots, x_N$. This corresponds to looking for a hyperplane w such that every data point lies on its positive side with a margin of at least ρ. It should be noted how deviations from this hyperplane are penalized following a hinge loss, this time weighed by the parameter $\nu \in (0, 1)$. A constrained formulation of the problem is also possible, resulting in

$$\begin{aligned} \min_{w,\xi,\rho} \quad & \tfrac{1}{2}\|w\|_2^2 - \rho + \frac{1}{\nu N}\sum_i \xi_i, \\ \text{s.t.} \quad & \begin{cases} w \cdot x_i \geq \rho - \xi_i \ \forall\, i, \\ \xi \geq 0. \end{cases} \end{aligned}$$

Once the optimization problem is solved new data points can be classified as belonging to the data distribution or not by evaluating $f(x) = sgn(w \cdot x - \rho)$. If $f(x) \geq 0$, x is considered to have been generated by the same distribution, else x is regarded to have been generated by a different distribution or to be an outlier.

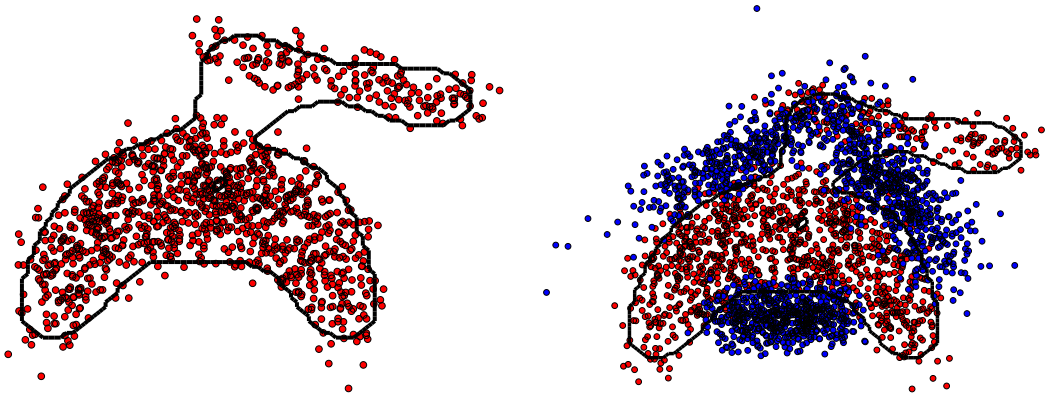
Once again this SVM-like model can be transformed into a dual form where kernelization is feasible. The dual one-class SVM problem takes the form

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T K \alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot e = 1, \\ 0 \leq \alpha \leq \frac{1}{\nu N}, \end{cases} \end{aligned}$$

and the primal solution (w, ρ) can be recovered as

$$\begin{aligned} w &= \sum_i \alpha_i x_i, \\ \rho &= \sum_j \alpha_j w \cdot x_i \quad \forall\, \alpha_i \in \left(0, \frac{1}{\nu N}\right). \end{aligned}$$

Figure 2.5 presents and example of application of the One-Class SVM model. Using the bidimensional dataset *banana* [35], training and test splits are made, and all instances from the negative (blue) class are removed from the training set. By training the One-Class SVM with appropriate kernel and ν parameters, the boundary shown in



(a) (b)

FIGURE 2.5: An example of application of the One-Class SVM on the *banana* dataset. a) Training set using only data from one of the classes, along with the classification boundary learned. b) Application of the model to test data presenting the two classes.

Figure 2.5(a) is obtained. When this trained model is then applied to the test set, containing patterns from both classes, most of the patterns from the new class are correctly identified as not belonging to the distribution observed in training.

2.2.3 LS-SVM

The **Least-Squares Support Vector Machine** (LS-SVM) was introduced by Suykens and Vandewalle [31] as a model approximating the standard SVM model. Featuring a simpler formulation, it also allows to tackle both classification and regression tasks, and the resulting optimization problem is also easier to solve. The primal formulation of the LS-SVM can be written in the form of loss plus regularizer as

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 + \frac{C}{2}\|(w \cdot X + be) - y\|_2^2,$$

with X matrix of input patterns, y vector of class labels or regression targets, depending on the task to perform, and e is an all–ones vector. It is readily noticed that the loss function selected for the model is nothing but the least-squares loss (hence the name). While this loss is better suited for regressions tasks, it can also deal with classification as a particular case of regression to values $\{-1, 1\}$. It must be remarked, though, that in opposition to the hinge loss or the ϵ-insensitive loss, the least-squares loss penalizes every deviation of the model from the target value, which can make the model prone to overfitting. In practice, however, LS-SVM usually produces similar results to SVM or SVR in spite of this.

A dual form of LS-SVM can be obtained by first rewriting its primal as a constrained problem, in the form

$$\begin{array}{ll} \min\limits_{w,b,\xi} & \frac{1}{2}\|w\|_2^2 + \frac{C}{2}\xi^T\xi, \\ s.t. & (w \cdot x_i + b) - y_i = \xi_i \; \forall \; i. \end{array}$$

Dualization can then be performed, obtaining the problem

$$\begin{array}{ll} \min\limits_{\alpha} & \frac{1}{2}\alpha^T Q \alpha - y^T \alpha, \\ s.t. & \alpha^T e = 0, \end{array}$$

where the matrix Q can be obtained as $Q = K + I_C$, with I_C a diagonal matrix with non-zero entries $1/C$. Once solved, optimal values for the primal can be recovered as

$$\begin{array}{rcll} w & = & \sum\limits_i \alpha_i y_i x_i, & \\ b & = & y_i - \frac{\alpha_i}{C} - w \cdot x_i & \forall \; i. \end{array}$$

One major drawback of LS-SVM is that because of the dual problem lacking box constraints, having $\alpha_i^* = 0$ is a rare situation, and so usually every training vector turns out to be a support vector. This results in high costs when evaluating the objective function $f(x) = \sum_{\alpha_i \neq 0} \alpha_i y_i x$, making the method impractical for on–line purposes when the training set is large.

On a positive note, the lack of box constraints allows to obtain an LS-SVM solution through the following linear equations system

$$\begin{bmatrix} 0 & e^T \\ e & Q \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix},$$

which has been shown to produce the optimal (α^*, b^*) choices [31]. This method for solving the LS–SVM is widely extended in practice, and in fact is the approach followed in the LS–SVM software by the original authors [36]. It must be realized, though, that solving this kind of linear system incurs in a cubic cost ($O(N^3)$) even if matrix

Support Vector Machine	**One–Class SVM**	**Support Vector Regression**
$Q = I_y\ K\ I_y$	$Q = K$	$\alpha = [\alpha, \alpha^*]$
$p = -e$	$y = e$	$Q = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix}$
$\Delta = 0$	$p = 0$	$p = [\epsilon e - t, \epsilon e + t]$
	$\Delta = 1$	$y = [e, -e]$
	$C = \frac{1}{\nu N}$	$\Delta = 0$

TABLE 2.2: Transformations of the generalized SVM formulation by Chang et al. (Equation 2.8) leading to particular SVM models

decomposition strategies are used, and thus its scalability is severely compromised. More efficient (though more complex) methods similar to the ones used in the classic SMO model have been developed as well [37, 38]. In this thesis the LS–SVM model is shown to be a particular instance of a more general SVM formulation (see Section 2.2.4), and so this latter approach is followed.

It is also notable that it has been shown [39, 40] how relationships exist between the LS–SVM model and the **Kernel Fisher Discriminant Analysis** (KFDA) [41]. In particular a solution for the KFDA model can be obtained by first solving the LS–SVM problem and then rescaling the resultant solution. Therefore, any algorithm solving LS–SVM is also immediately applicable for solving the KFDA model.

2.2.4 Generalized SVM formulation

While the standard Support Vector Machine, Support Vector Regression and One-Class SVM models solve different tasks, their dual forms present intriguing similarities. Indeed, and as observed by Chang et al. [30], the three models can be posed under a common, more general formulation in the form

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T Q \alpha + p^T \alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = \Delta \\ 0 \leq \alpha \leq C \end{cases}. \end{aligned} \tag{2.8}$$

This is equivalent to the cited models under the transformations shown in Table 2.2. Recall that I_y stands for a diagonal matrix with the labels vector y as its main diagonal.

The main benefit of this joint formulation is the ability to deal with classification, regression and density estimation tasks by just solving the single optimization problem

Least–Squares SVM	
$Q = K + I_C$	$\Delta = 0$
$p = -y$	$L = -\infty$
$y = e$	$C = \infty$

TABLE 2.3: Transformation of the generalized SVM formulation proposed in this thesis (Equation 2.9) leading to the LS–SVM model.

posed by Equation 2.8. Note that this general problem still keeps a similar structure to the specific models: quadratic objective function with box constraints and a single equality constraint. Therefore, the difficulty of the problem to solve is not increased by this generalization.

Due to its flexibility, this approach to SVM modeling is followed in the popular LIBSVM [30] software. In this thesis this formulation is further extended to include the LS-SVM model as well by modifying the lower bound on the α coefficients as

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T Q\alpha + p^T\alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = \Delta \\ L \leq \alpha \leq C \end{cases} . \end{aligned} \tag{2.9}$$

The new parameter L allows choosing a lower bound different than 0. SVM, SVR and One-class SVM models still fit in this framework by setting $L = 0$ and using the transformations above. For the LS-SVM model the adequate transformation is given by Table 2.3.

In this way the optimization problem 2.9 effectively handles all four SVM models. A diagram depicting its relationships with these models is shown in Figure 2.6. Owing to this, the rest of the discussion focuses on the optimization of this problem.

2.2.5 Other SVM formulations

For the sake of completeness, other SVM models of relevance are briefly discussed in this section. Note however they are outside the scope of this thesis.

- The **ν-SVM** [42] is a reformulation of SVM for classification, where the penalty parameter C is substituted by another parameter $\nu \in (0, 1]$ having a more direct interpretation: ν is an upper bound on the fraction of training errors and a lower

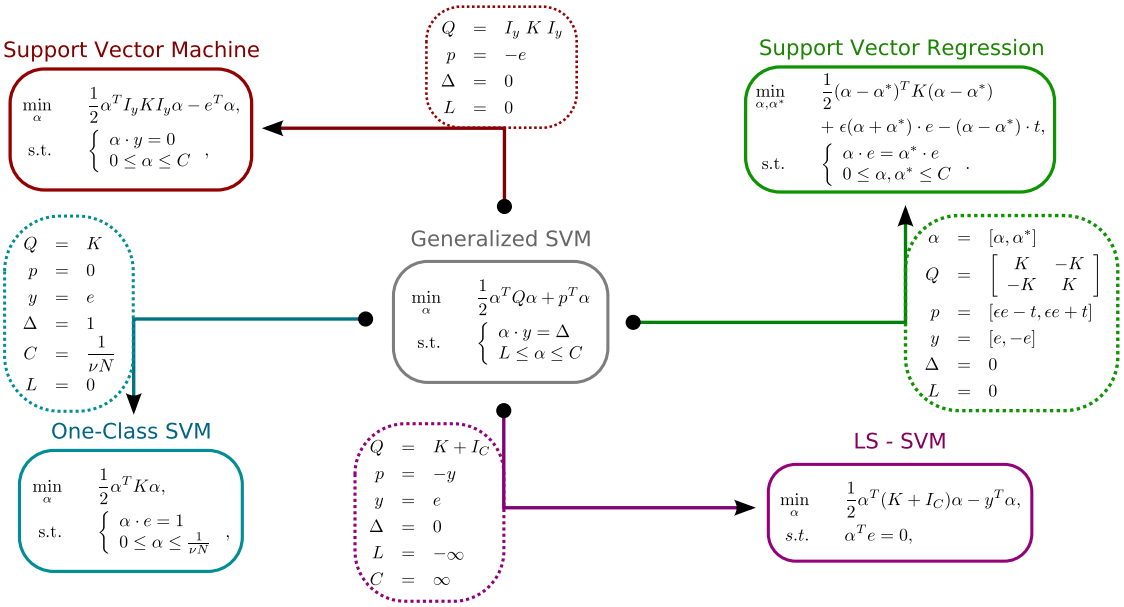


FIGURE 2.6: Relationships between the generalized SVM and the presented SVM models.

bound of the fraction of support vectors. Equivalence relationships between the C and ν parameters have been established, as well as with geometric SVM models [43].

- In analogy with the SVM and SVR models, the **ν-SVR** [42] stands as the regression counterpart for ν-SVM. Similar interpretations of the parameter ν can be made in this case as well.

- The **Extended ν-SVM** (Eν-SVM) [44, 45] model is a generalization of ν-SVM able to produce a wider range of solutions than the classic ν-SVM (and thus C-SVM). This is done by observing that there exist a certain ν_{max} above which the ν-SVM is only able to produce the trivial $w = 0$ solution. By generalizing the ν-SVM model it is possible to obtain non-trivial solutions for $\nu > \nu_{max}$, which can lead to better model accuracies in some situations. Unfortunately, the resultant optimization problem turns out to be non-convex for the new range $(\nu_{max}, 1)$, making it very hard to solve. Relationships of this model with other SVM formulations and efficient algorithms for Eν-SVM are still open areas of research.

2.3 Geometry of SVM

While the SVM models are meaningful from the point of view of regularization and margin maximization, their motivations are not always easy to grasp. The previous sections have already shown that, in particular, the SVM model for classification results

from the sum of a number of non-trivial ideas, and so gaining insights into this method is not straightforward. One example of this fact is the C penalty parameter: even though its purpose is well understood, there is not a clear interpretation of the effects of specific values of C on the model.

Geometric Support Vector Machines appear as an alternative yet equivalent model to the standard SVM, providing clearer concepts about the classifier through geometry. In this section a brief review of this particular approach is presented.

An equivalent geometric model for SVM for classification was first proposed in the work of Bennett and Bredensteiner [46]. Through the use of KKT conditions and duality arguments, the authors show that the solution found by the SVM model is equivalent to performing the following steps:

1. Consider two **convex hulls**: one formed by the points of the positive class, the other by the points of the negative class.
2. Find two points, each inside one of the convex hulls, such that distance between them is minimal.
3. The hyperplane bisecting the line joining such points is the SVM solution.

Recall from geometry basics that the convex hull $\mathcal{C}$ of a set of points $\mathcal{X} = \{x_1 \dots x_N\}$ is defined as the set

$$\mathcal{C}(\mathcal{X}) = \left\{ z \,\middle|\, z = \sum_i \mu_i x_i, \sum_i \mu_i = 1, \mu_i \geq 0 \; \forall \; i \right\},$$

that is to say, any point which can be expressed as a convex combination of the points in $\mathcal{X}$ belongs to $\mathcal{C}(\mathcal{X})$.

Figure 2.7 shows an example of this approach. A quick look suffices to realize that the hyperplane obtained is, in fact, the one maximizing the margin, as one would expect from the solution of a SVM.

Mathematically the geometric model can be simply written as

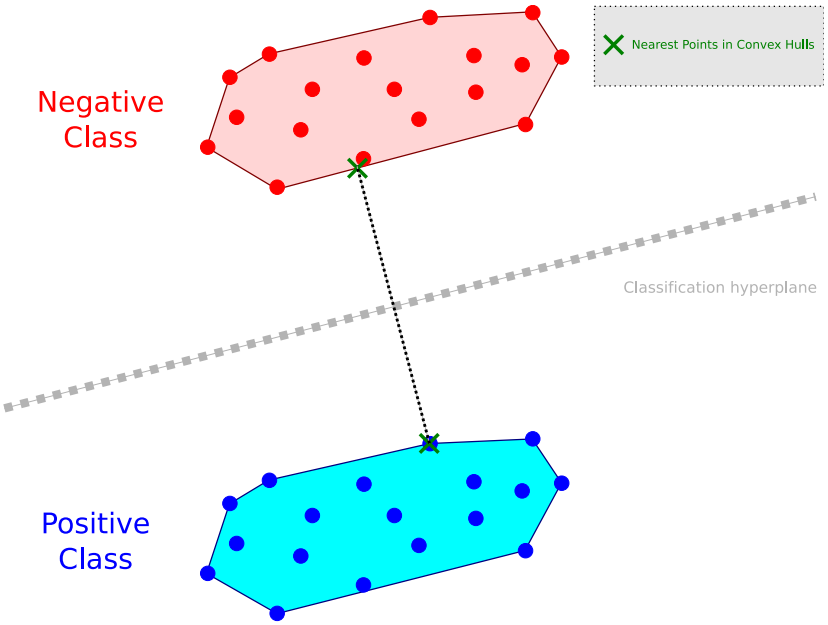


FIGURE 2.7: Example of the classification hyperplane obtained by following the bisection of the line joining the nearest points in the convex hulls of each class.

$$\begin{aligned} \min_{\alpha} \quad & \left\| \sum_{i+} \alpha_i x_i - \sum_{i-} \alpha_i x_i \right\|_2^2, \\ \text{s.t.} \quad & \begin{cases} \sum_{i+} \alpha_i = 1, \\ \sum_{i-} \alpha_i = 1, \\ \alpha \geq 0, \end{cases} \end{aligned} \tag{2.10}$$

where $i+$ notes the indexes corresponding to the patterns from the positive class and $i-$ to the negative one. This optimization problem is nothing but finding the nearest points (objective function) in the convex hulls of each class (constraints). The classification hyperplane (w, b) can then be obtained as the mentioned bisection:

$$\begin{aligned} w &= \sum_{i+} \alpha_i x_i - \sum_{i-} \alpha_i x_i, \\ b &= -\tfrac{1}{2} \left(w \cdot \sum_{i+} \alpha_i x_i + w \cdot \sum_{i-} \alpha_i x_i \right). \end{aligned}$$

By noting that $\sum_{i+} \alpha_i x_i - \sum_{i-} \alpha_i x_i = \sum_i y_i \alpha_i x_i$ and rewriting the norm in matrix form, it is easy to arrive at the equivalent problem

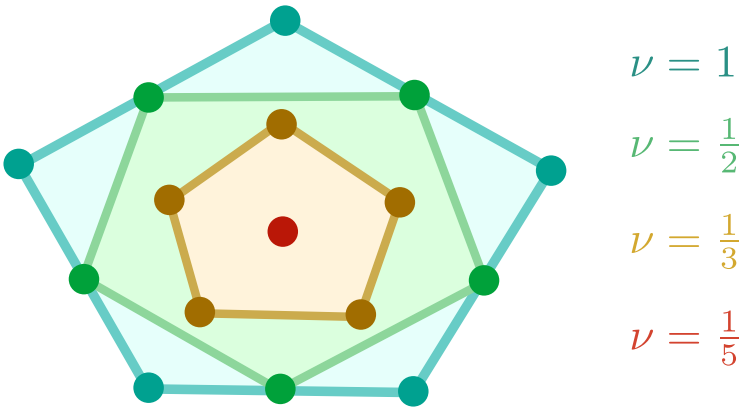


FIGURE 2.8: Example of convex hull reduction for different values of the ν parameter.

$$\begin{aligned} \min_{\alpha} \quad & \alpha^T Q \alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = 0, \\ 0 \leq \alpha \leq 1, \\ \alpha \cdot e = 2. \end{cases} \end{aligned} \tag{2.11}$$

In this form the similarities with the dual SVM formulation become evident. The only novelties are the appearance of the last constraint and the bound on the α coefficients by 1.

It must be drawn to attention now the fact that the described geometric model corresponds to a hard margin SVM. If the classes result not to be linearly separable, their convex hulls have a non-empty intersection, and thus the result of problem 2.10 is the trivial solution $w = 0$. To make up for this, again in [46] the concept of **Reduced Convex Hulls** (RCH) is presented, which are defined as

$$\mathcal{C}_\nu(\mathcal{X}) = \left\{ z \,\middle|\, z = \sum_i \mu_i x_i, \sum_i \mu_i = 1, 0 \leq \mu_i \leq \nu \; \forall \, i \right\},$$

that is, again a convex combination of points, though this time the maximum contribution of each point is limited to ν. A value $\nu = 1$ produces the standard convex hull definition, and as ν grows smaller the hulls are reduced towards their barycenters. The minimum feasible value of ν turns out to be $\nu_{min} = \frac{1}{|\mathcal{X}|}$, as then the RCH is reduced to a singleton: the barycenter $\sum_i \frac{1}{|\mathcal{X}|} x_i$. An illustration showing the resultant RCHs for different values of ν is shown in Figure 2.8.

By means of reducing the convex hulls a non-trivial classification hyperplane can be obtained, and so the following optimization problem arises

$$\begin{aligned} \min_{\alpha} \quad & \alpha^T Q \alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = 0, \\ 0 \leq \alpha \leq \nu, \\ \alpha \cdot e = 2, \end{cases} \end{aligned}$$

which results to be equivalent to the soft margin SVM. As expected, ν acts as a substitute for the penalty parameter C, allowing for larger margin errors when ν is small. However, the range of this alternative parameter is limited as $\nu \in \left[\frac{1}{|\mathcal{X}|}, 1\right]$, and furthermore its effect in the model is clear through geometry concepts.

Besides classification, an equivalent geometric model has been proposed as well for **Support Vector Regression** by Bi and Bennet [47]. It consists in casting the regression problem as a classification one by creating two synthetic classes above and below the ϵ-tube, and then finding the SVM classification hyperplane, which turns out to be also the regression function. This approach could be useful in those situations where a good SVM solver is already available, thus allowing to apply it to regression as well. However, in practice the generalized SVM formulation presented in Section 2.2.4 is preferred, since it also allows to address other problems apart from classification and regression. Nevertheless, this geometric regression approach is of use to gain further insights into the SVR model.

It is also worth mentioning that other approaches to geometric soft margin SVM have appeared in the literature, such as the **Scaled Convex Hulls** (SCH) method by Liu et al. [48, 49]. This particular approach reduces the class convex hulls by scaling them down towards their barycenters, therefore not altering the shape of the hull. While this provides some advantages such as an easy computation of a value for the scaling parameter ν ensuring hulls separability, the scaled hulls are only an approximation of the reduced hulls, producing a similar model only when the number of training patterns is large.

Considering geometric SVMs as a whole, it can be stated that they provide the same functionality as standard SVMs, as well as further insights into the workings of these models. A number of algorithms are available to solve these alternative optimization problems (see Section 2.4.3), some of them having strong connections with solver algorithms for the standard SVM. These connections result useful in practice to transfer advances in one field to the other one, and in fact some of the novelties presented in this thesis make use of them (Section 3.1).

2.4 Standard solvers for SVM

After studying the key concepts of the Support Vector Machine model and its generalization to address several SVM-like models, this section presents a review of the algorithms available in the literature to solve the SVM problem. An overview of solvers for the geometric SVM formulation is included in the discussion as well. The methods here form the baseline from which the contributions in Chapter 3 are built up.

First, the primal formulation of the SVM is considered, as the majority of the recent advances in SVM methods have been focused on this variant. At first glance two approaches seem possible:

a) Solve the unconstrained loss + regularizer form of the problem, which suffers from non-differentiability (Equation 2.1),

$$\min_{w,b} \tfrac{1}{2}\|w\|_2^2 + C\sum_i |1 - y_i(w \cdot x_i + b)|_+.$$

b) Solve the smooth, convex but constrained version of the problem (Equation 1.4),

$$\begin{aligned} \min_{w,b,\xi} \quad & \tfrac{1}{2}\|w\|_2^2 + C\sum_i \xi_i, \\ \text{s.t.} \quad & \begin{cases} y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall\, i, \\ \xi_i \geq 0 \quad \forall\, i. \end{cases} \end{aligned}$$

The most straightforward approach is to address formulation (b). Being a standard convex problem, widely-available **Quadratic Program solvers** (QP solvers) such as `CVX` [50] or Matlab's Optimization Toolbox can be used practically out of the box to obtain a solution for the problem. This naive approach, though, does not make any use of the structure present in the problem, and thus is bound to produce poor results.

Solving formulation (a) turns out to be more profitable, and in fact the most successful algorithms addressing the primal SVM problem do so by employing non-differentiable optimization methods. One such example is the **Pegasos** [51] method by Shalev-Shwartz et al., which relies on a projected subgradient descent method. Other examples are given by the **OCAS** [52] method by Franc and Sonnenburg and the less recent **SVM-Perf** [53] method by Joachims, which follows a cutting planes strategy to find the SVM solution. All of these algorithms are able to solve the primal problem very efficiently by taking advantage of the fact that the hinge loss, although not being smooth, is a simple piecewise linear function; i.e. the structure of the problem is explicitly used.

Algorithm 2 Dual coordinate descent for linear SVM

1: **Inputs:** training set (x, y), penalty parameter C.
2: Initialize α, w to some feasible value.
3: **while** α not optimal **do**
4: **for** $i = 1, \ldots, N$ **do**
5: Compute gradient $G = y_i w \cdot x_i - 1$.
6: Project gradient $PG = \begin{cases} \min\{G, 0\} & \text{if } \alpha_i = 0, \\ \max\{G, 0\} & \text{if } \alpha_i = C, \\ G & \text{if } 0 < \alpha_i < C. \end{cases}$
7: Compute new unconstrained $\alpha_i' = \frac{\alpha_i - PG}{Q_{ii}}$.
8: Clip to box constraints $\alpha_i' = \min\{C, \max\{\alpha_i', 0\}\}$.
9: Update $w' = w + (\alpha_i' - \alpha_i) y_i x_i$.
10: **end for**
11: **end while**
12: **Return** (w^*, b^*).

Despite all these efforts to produce good solvers for the SVM primal, there is a procedure these algorithms can hardly go through: kernelization. Recall from 2.1.2 that when the kernel trick is used the SVM weight vector w lies in the feature space $\mathbb{F}$ induced by the mapping Φ. If this mapping is only known through its associated kernel function $k(\cdot, \cdot)$, no explicit representation of w can be made, making the primal problem unapproachable in principle.

A way of circumventing this problem is to try to extend the ideas behind these algorithms to the dual formulation. A remarkable example of this is the state-of-the-art SVM optimization method included in the **LIBLINEAR** library [54, 55] by Hsieh et al., which solves the modified dual problem

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T Q \alpha - e^T \alpha, \\ \text{s.t.} \quad & 0 \leq \alpha \leq C, \end{aligned}$$

that results from considering a primal SVM with no bias term b. The algorithm essentially performs coordinate descent in this dual problem, i.e. updates one α_i at a time, by also taking advantage from the structure of the primal. A pseudocode of this method is presented in Algorithm 2. The greatest virtue of this method comes from the fact that most expensive steps are the computation of the gradient (line 5) and the update of w (line 9): both of them can be performed in just $O(\bar{d})$ operations, with $\bar{d}$ the average number of non-zero features in the training patterns. However, it must be pointed out that this is possible thanks to the ability to maintain an explicit representation of w, which in turn eases the computation of the gradient. If the very same algorithm was

to be applied to a kernelized SVM, this shortcut would no longer be available, and the gradient would have to be computed as

$$G = Q_{i,:}\alpha - 1 = y_i \sum_{\alpha_j > 0} \alpha_j y_j k(x_i, x_j) - 1,$$

where $Q_{i,:}$ stands for the i-th row of Q. This requires up to $O(\bar{d}N)$ operations for N the size of the training set, which is much more than the previous computation, as generally $\bar{d} \ll N$. Furthermore these $O(\bar{d}N)$ operations involve calling the kernel function which, being a non-linear operation, usually incurs in even larger costs than "standard" operations like products or sums. Even if the full gradient is maintained in memory and updated at every iteration, the updates follow the expression

$$\nabla_\alpha f(\alpha') = \nabla_\alpha f(\alpha) + Q_{:,i}(\alpha'_i - \alpha_i),$$

for $Q_{:,i}$ the i-th column of Q, therefore requiring again up to $O(\bar{d}N)$ operations. In the view of this, the authors of the method conclude that when considering non-linear SVMs, using a decomposition method (presented later on in Section 2.4.1) is more efficient than their proposed algorithm.

The lesson learned from the analysis of the dual coordinate descent method is that even though great advantage can be gained from using the explicit primal representation, this advantage does not hold for the non-linear case. This effect is confirmed also by the authors of the Pegasos algorithm [51], in which kernelization can be achieved by replacing the explicit representation of the weight vector by $w = \sum_{\alpha_i > 0} \alpha_i \Phi(x_i)$. This, however, again increments the cost per iteration of the algorithm, hence not being satisfactory.

In conclusion, even though recent works in primal SVM optimization have resulted in extremely efficient and practical training algorithms for linear SVMs, these results do not apply for non-linear SVMs. Because of this, dual solvers based on the decomposition strategy are still the state-of-the-art approach in this area. Consequently, the rest of this section focuses on these dual solvers.

2.4.1 Dual SVM solvers: QP, chunking and decomposition methods

Considering now the optimization problem posed by the generalized dual formulation of the SVM, recall that it can be written as the quadratic (convex) program (Equation 2.9)

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T Q\alpha + p^T\alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = \Delta, \\ L \leq \alpha \leq C. \end{cases} \end{aligned}$$

As in the constrained primal formulation, this problem can also be tackled by using a general QP solver, though again this is prone to perform badly. Methods tailored to take advantage of the specific structure of the problem, such as **interior point** methods [56], also fail to produce good running times because

a) They require storing the full Q matrix into memory ($O(N^2)$ storage), or at least making use of all of its entries at every iteration.

b) Their computational costs are of order $O(N^3)$.

c) Even though they provide very strong guarantees on convergence and on the accuracy of the solution ($\approx 10^{-8}$ dual gap), in machine learning applications generally mild guarantees suffice ($\approx 10^{-3}$ dual gap).

Because of (a), large-scale learning is unfeasible, as for instance a dataset of 100000 training patterns would requiring storing a 100000 $\times$ 100000 matrix of floats. Even if only 4 bytes are used per float, this already requires 40 GB of RAM memory. If instead of storing Q in memory its entries are computed on the fly every time they are needed, the running time of (b) would further increase to $O((\bar{d}N)^3)$ kernel evaluations per iteration. Even a computational cost of $O(N^3)$ is impractical for medium to large size datasets. Finally, (c) hints that using such methods is overshooting, and so simpler methods should be considered instead.

To overcome these difficulties, the first working idea that appeared in the SVM community was **chunking**. Originally, Vapnik [8] realized that by discarding from the training set those elements that result in $\alpha_i^* = 0$ at the optimum (i.e. not support vectors) the solution of the problem remains the same. To take advantage of this, the algorithm sequentially applies a QP solver over a series of subproblems involving only a subset B of the training patterns, known as the **working set**, in the form

$$\begin{aligned} \min_{\alpha_B} \quad & \tfrac{1}{2}\begin{bmatrix}\alpha_B^T & \alpha_N^T\end{bmatrix}\begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN}\end{bmatrix}\begin{bmatrix}\alpha_B \\ \alpha_N\end{bmatrix} + \begin{bmatrix}p_B^T & p_N^T\end{bmatrix}\begin{bmatrix}\alpha_B \\ \alpha_N\end{bmatrix}, \\ \text{s.t.} \quad & \begin{cases} \begin{bmatrix}\alpha_B^T & \alpha_N^T\end{bmatrix}\begin{bmatrix} y_B \\ y_N\end{bmatrix} = \Delta, \\ L \leq \alpha_B, \alpha_N \leq C. \end{cases} \end{aligned}$$

The training patterns N not in B are kept fixed, and so the problem simplifies down to

$$\begin{aligned} \min_{\alpha_B} \quad & \tfrac{1}{2}\alpha_B^T\, Q_{BB}\, \alpha_B + \alpha_B^T\,(Q_{BN}\,\alpha_N + p_B) \\ \text{s.t.} \quad & \begin{cases} \alpha_B \cdot y_B = \Delta, \\ L \leq \alpha_B \leq C, \end{cases} \end{aligned}$$

which is a much easier problem if the working set B is small and, what is more, only requires storing in memory the parts Q_{BB} and $Q_{BN}\,\alpha_N$ of the kernel matrix. The main concern is, though, how to decide which working set to choose, and furthermore, whether there is a guarantee that after sequentially optimizing over a series of subsets $B_1, B_2, \ldots$ the solution of the full problem is obtained. Vapnik connected this problem with his observation regarding the $\alpha_i^* = 0$ coefficients and proposed the following: start with some small working set B, and at each iteration remove from this subset those $\alpha_i = 0$, and add to the set a fixed number M of α_i coefficients among those that most violate the KKT optimality conditions (details on this criterion are given later in Section 2.4.2). Eventually B is guaranteed to contain every support vector, and thus the solution of the full problem is attained.

While practical at first, the chunking approach suffers from a severe drawback: if the number of support vectors of the final solution is large, the last steps of the method require solving problems of almost the same size as the full problem itself. To address this, Osuna et al. [57] proved that the solution to the SVM problem can be obtained by solving a series of subproblems even if none of them optimize over the whole set of support vectors simultaneously; this new approach was termed the **decomposition method**. The only requirement for this to happen is that at each step at least one example violating the KKT conditions is added to the working set B. Therefore, using this fact, Osuna et al. propose a chunking algorithm in which a fixed-size B is used, and at each step drops one element from B to add a violating one. This approach solves the memory issues raised by the storage of $(Q_{BB}, Q_{BN}\,\alpha_N)$ regardless of the size of the

training set used thanks to the fixed size of the working set, while at the same time guaranteeing convergence.

In spite of this, the method of Osuna et al. is not free from problems. The selection of which element to drop from the working set and which one to add among the KKT violating coefficients is not trivial, and different authors implementing the method have used different heuristics. A clear winner among all these approaches to decomposition did not appear until the introduction of the **SVM$^{\text{light}}$** [58] algorithm by Joachims. SVM^{light} builds over the method of Osuna et al. by introducing a number of key novelties, the most prominent of them being a strategy to select the working set at each step based on Zoutendijk's method [59]. This finds the feasible updating direction providing greatest descent in the objective function (in a first order sense) such that only q coefficients are updated, q being the size of the working set B. This can be written as the optimization problem

$$\begin{aligned} \min_{d} \quad & \nabla f(\alpha) \cdot d \\ \text{s.t.} \quad & \begin{cases} d \cdot y = 0, \\ d_i \geq 0 \quad \forall\, i : \alpha_i = 0, \\ d_i \leq 0 \quad \forall\, i : \alpha_i = C, \\ -1 \leq d \leq 1, \\ \|d\|_0 = q. \end{cases} \end{aligned}$$

Note the "first order sense" comes from the fact that $\nabla f(\alpha) \cdot d$ approximates the change in the objective function when performing a step d under a first order Taylor expansion approximation, i.e.

$$f(\alpha + d) \simeq f(\alpha) + \nabla f(\alpha) \cdot d.$$

The optimization problem posed above would be, in principle, very hard to solve because of the L0 constraint $\|d\|_0 = q$. In spite of this, Joachims notes that in fact the problem is solvable by following the steps:

1. Restrict q to an even number.
2. Sort the α_i coefficients by the values $\omega_i = y_i[\nabla f(\alpha)]_i$ in decreasing order.
3. Select the top $\frac{q}{2}$ elements from the sorted list such that $0 < \alpha_i < C$, or $d_i = -y_i$ obeys the constraints above.

4. Select the bottom $\frac{q}{2}$ elements from the sorted list such that $0 < \alpha_i < C$, or $d_i = y_i$ obeys the constraints above.

5. The selected coefficients define the working set.

Following this, the working set selection is performed just at a cost of $O(N \log N)$ operations, as the most expensive step is sorting (supposing the gradient has already been computed). Also, using this working set selection strategy is shown to produce much better results than other heuristics.

Nevertheless, and even after the improvements of SVM^{light}, the decomposition method still needs to invoke a QP solver at each step of the algorithm. Even if the size of the subproblems to solve is smaller than the full problem, the performance issues pointed out at the beginning of the section still arise. It was not until the introduction of the SMO algorithm that the need of a QP solver was dropped off and these drawbacks overcame. Due to the relevance of this algorithm for the field of SVMs as well as for this thesis, the next subsection is entirely devoted to this method.

2.4.2 Sequential Minimal Optimization

2.4.2.1 Platt's SMO

The **Sequential Minimal Optimization** (SMO) algorithm is the result of taking the decomposition strategy by Osuna et al. [57] to the extreme. First proposed by Platt [60], the algorithm sequentially optimizes minimum-sized working sets; hence the name. As in the SVM dual problem the equality constraint $\alpha \cdot y = \Delta$ is present, the minimum size working set must consist of two α_i coefficients for the α to remain feasible. Traditionally, the indexes of these two coefficients are usually noted as l and u. The greatest advantage of this approach boils down to the fact that with such a small working set, a closed form solution of the resultant quadratic problem is possible, thus overriding the need of a QP solver. In what follows the way to achieve this is explained.

Considering that the working set consists in only the l and u indexes, the value of the α coefficients after an SMO step would in principle be

$$\begin{aligned} \alpha'_u &= \alpha_u + \delta_u, \\ \alpha'_l &= \alpha_l + \delta_l, \\ \alpha'_i &= \alpha_i \quad \forall\, i \neq l, u. \end{aligned}$$

However, this update is not valid for every pair of (δ_u, δ_l) values, as the constraints of the problem should be taken into account. Considering first the $\alpha \cdot y = \Delta$ constraint it is realized that

$$\begin{aligned} & \alpha' \cdot y &&= \Delta, \\ \rightarrow \quad & \alpha \cdot y + \delta_u y_u + \delta_l y_l &&= \Delta, \\ \rightarrow \quad & \delta_u y_u + \delta_l y_l &&= 0, \\ \rightarrow \quad & \delta_l &&= -y_u y_l \delta_u, \end{aligned}$$

that is, the update really depends only on δ_u, as δ_l depends on it to ensure feasibility. Because of this, the SMO's update can be written as

$$\alpha' = \alpha + \delta(e_u - y_u y_l e_l) = \alpha + \delta s,$$

where e_i is an all-zeroed vector but for a 1 at the i-th entry, δ is a stepsize and s is the updating direction $e_u - y_u y_l e_l$. This update guarantees that the constraint $\alpha \cdot y = \Delta$ is met, and so the decomposition subproblem associated with the working set (l, u) can be written as

$$\begin{aligned} \min_{\delta} \quad & \tfrac{1}{2}(\alpha + \delta s)^T Q (\alpha + \delta s) + (\alpha + \delta s)^T p \\ \text{s.t.} \quad & \begin{cases} L \leq \alpha_u + \delta \leq C, \\ L \leq \alpha_l - y_u y_l \delta \leq C. \end{cases} \end{aligned} \tag{2.12}$$

The subproblem depends on a single variable δ, the objective function is still quadratic convex (a parabola, in fact), and the box constraints have been reduced to two two-sided simple inequality constraints, thanks to the fact that every $\alpha_i \; \forall \; i \neq l, u$ is not updated. This kind of problem is straightforward to solve upon realization that, because the problem is convex and one-dimensional, the optimal δ^* is either at the unconstrained optimum $\hat{\delta}$ or at boundary of the feasible region (in fact, an interval) nearest to $\hat{\delta}$. Therefore, the solution is obtained by first computing the unconstrained optimum as

$$\begin{aligned} & 0 &&= \tfrac{\partial f(\alpha+\delta s)}{\partial \delta}, \\ \rightarrow \quad & 0 &&= s^T Q(\alpha + \delta s) + s^T p, \\ \rightarrow \quad & \hat{\delta} &&= \tfrac{-s^T Q \alpha - s^T p}{s^T Q s}, \end{aligned}$$

and then clipping it down to the feasible region (if needed) as

Algorithm 3 Sequential Minimal Optimization

Inputs: x, y, p, L, C, Q.
while stopping criterion not met **do**
  Select working set (l, u).
  Compute unconstrained stepsize $\hat{\delta}$ using Equation 2.14.
  Clip down $\hat{\delta}$ using Equation 2.13 to obtain δ^*.
  Update: $\alpha_u \leftarrow \alpha_u + \delta^*$, $\alpha_l \leftarrow \alpha_l - y_u y_l \delta^*$.
end while
Compute bias b^*.
Return (α^*, b^*).

$$\delta^* = \max\left\{L - \alpha_u, -y_u y_l (L - \alpha_u), \min\left\{C - \alpha_u, -y_u y_l (C - \alpha_l), \hat{\delta}\right\}\right\}. \tag{2.13}$$

As complex this last expression might seem to be, it just checks whether $\hat{\delta}$ makes α_u or α_l move out of the $L \leq \alpha_i \leq C$ bounds and, if this is the case, adjusts δ so that the update moves α no farther than the bounds.

Regarding the efficiency of these computations, it is clear that the clipping operation can be done in constant ($O(1)$) time. Addressing now the calculation of $\hat{\delta}$, because of the sparsity of s (only entries u and l differ from zero) and realizing that the gradient takes the form $\nabla f(\alpha) = Q\alpha + p$, its expression can be rewritten as

$$\hat{\delta} = \frac{-s \cdot \nabla f(\alpha)}{s^T Q s} = \frac{-[\nabla f(\alpha)]_u + y_u y_l [\nabla f(\alpha)]_l}{Q_{uu} + Q_{ll} - 2Q_{ul}}, \tag{2.14}$$

which turns out to be doable with cost $O(\bar{d})$ for the calculation of the required Q entries ($\bar{d}$ the average number of non-zero features), provided the gradient has already been computed.

A sketch of the SMO algorithm is shown as Algorithm 3. Recall the bias of the solution b^* could be computed by means of the KKT conditions (Section 2.1.1). The only loose ends in the definition of the algorithm are the procedure to select the working set and the stopping criterion to use. In Platt's original proposal [60] an involved heuristic checking the degree of violation of the KKT conditions for each pattern is used to select the working set. If no pattern with a minimum quantity of violation is found, the algorithm stops. Though the ideas behind this proposal are sound, their implementation turns out to be somewhat messy.

2.4.2.2 Keerthi et al.'s SMO

A few years later after the publication of Platt's SMO method, Keerthi et al. [37] proposed an improvement where a more simplistic yet more efficient working set selection is used. Their method is also based on measuring the quantity of violation for each pattern, and to do so they proceed as follows.

To begin with, the Lagrangian of the dual problem can be computed as

$$L(\alpha, \nu, \tau, b) = \tfrac{1}{2}\alpha^T Q\alpha + \alpha \cdot p - \nu \cdot (\alpha - L) + \tau \cdot (\alpha - C) + \beta(\alpha \cdot y - \Delta).$$

The "dual of the dual" Lagrange coefficient β actually coincides with the primal variable b. This is because of strong duality: dualization of the dual results in the original primal problem. Note though that in the dual of the SVM some transformations have been carried out in order to simplify the problem, and thus the Lagrange coefficients ν and τ appear instead of the primal ξ. Regardless of this, the KKT conditions for this dual-primal pair can be obtained as in the dual SVM derivation, resulting in

Stationarity	**Dual feasibility**
• $\nabla f(\alpha) - \nu + \tau + by = 0.$	• $L \leq \alpha_i \leq C \ \forall \ i,$
	• $\alpha \cdot y = \Delta.$
Primal feasibility	**Complementary slackness**
• $\nu_i, \tau_i \geq 0 \ \forall \ i.$	• $\nu_i(\alpha_i - L) = 0 \ \forall \ i,$
	• $\tau_i(\alpha_i - C) = 0 \ \forall \ i.$

Recall that $\nabla f(\alpha) = Q\alpha + p$. Supposing an already feasible α, the primal feasibility conditions are already met. Regarding the rest of conditions, each α_i multiplier can be assigned to one of the following groups

- $\alpha_i < C$: by the complementary slackness conditions $\tau_i = 0$, and by dual feasibility $\nu_i \geq 0$. Therefore, by stationarity $[\nabla f(\alpha)]_i \geq -by_i$.
- $\alpha_i > L$: by the complementary slackness conditions $\nu_i = 0$, and by dual feasibility $\tau_i \geq 0$. Therefore, by stationarity $[\nabla f(\alpha)]_i \leq -by_i$.

By considering that $y_i \in \{-1, 1\}$, these two cases can be summarized into the KKT conditions

$$\begin{cases} -[\nabla f(\alpha)]_i \leq b \text{ if } i \in I \equiv \{y_i = 1, \alpha_i < C \text{ or } y_i = -1, \alpha_i > L\}, \\ -[\nabla f(\alpha)]_i \geq b \text{ if } i \in J \equiv \{y_i = -1, \alpha_i < C \text{ or } y_i = 1, \alpha_i > L\}. \end{cases}$$

Using this, the KKT violation for a pattern x_i can be measured as the absolute value in which the corresponding $[\nabla f(\alpha)]_i$ deviates from fulfilling these conditions. As already shown in the work of Osuna et al. [57], violating patterns are a good choice for the working set, so choosing (l, u) from the above sets is a sensible idea. It should be noted, however, that the optimal b is unknown until the end of the algorithm. Because of this, the violation can only be measured as pairs of patterns, i.e. a **violating pair** is a pair (i, j) such that $i \in I$, $j \in J$ and $-[\nabla f(\alpha)]_i > -[\nabla f(\alpha)]_j$, leaving no possible b value meeting the KKT conditions above. Therefore, such a pair is eligible as a working set. Also, making use of this concept of violating pairs, the KKT conditions can be even further summarized by means of the **most violating pair** (MVP), defined through the indexes (low, up) as

$$\begin{aligned} low &= \text{argmax}_i \{-[\nabla f(\alpha)]_i \text{ s.t. } i \in I\}, \\ up &= \text{argmin}_j \{-[\nabla f(\alpha)]_j \text{ s.t. } j \in I\}. \end{aligned}$$

Using this pair it is clear that the KKT conditions simplify down to

$$-[\nabla f(\alpha)]_{low} \leq -[\nabla f(\alpha)]_{up}.$$

Given this, Keerthi et al. propose two selection procedures:

- SMO Modification 1: find an i in the set $L < \alpha_i < C$ such that when paired with low or up (depending on whether $i \in J$ or $i \in I$, respectively), the resulting pair has a violation of at least 2ε. Use that pair as the working set.
- SMO Modification 2: use the most violating pair (low, up) as the working set if it has a violation of at least 2ε.

Modification 1 follows the spirit of the heuristics of Platt, which focused on the coefficients not at bound $L < \alpha_i < C$. Modification 2, in turn, pursues reducing the total violation of the KKT conditions by adjusting at each step the coefficients with largest

violation. Violations under a minimum of ε are not considered, as an approximate fulfillment of the optimality conditions is enough in practice. If at some step of the algorithm the rules above cannot find any suitable violating pair, the algorithm stops. Therefore, implicitly, the algorithm is using the quantity of violation of the MVP as a stopping criterion. Also, whatever the modification used, the working set selection requires linear ($O(N)$) time provided the gradient $\nabla f(\alpha)$ has already been computed. This one is maintained in memory and updated efficiently at every iteration by noting that

$$\begin{aligned} \nabla f(\alpha + \delta s) &= Q(\alpha + \delta s) + p, \\ &= \nabla f(\alpha) + \delta Q s, \\ &= \nabla f(\alpha) + \delta(Q_{:,u} - y_u y_l Q_{:,l}), \end{aligned}$$

at a cost of $O(\bar{d}N)$ dominated by the computation of the columns $Q_{:,u}$ and $Q_{:,l}$. Consequently, the overall cost of the algorithm per iteration is $O(\bar{d}N)$.

In practice, Keerthi et al. show that Modification 2 performs better, and so this is the strategy followed in most SMO implementations. Furthermore, this modification has been proved to be convergent [61]. It has also been shown that choosing $q = 2$ in SVM^{light} results in exactly the same algorithm as Modification 2 [62]. As this approach usually obtains better results than SVM^{light} when other choices of q are used, it is clear that avoiding the use of a QP solver effectively improves the efficiency of the method.

2.4.2.3 WSS2

Modification 2 is also termed as **First Order Working Set Selection** or WSS1, in contrast to the **Second Order Working Set Selection** (WSS2) [63] by Fan et al. This refinement further improves the ability of the selected working sets to produce a decrease in the objective function (the function being minimized) by observing that the new value of the objective function after an update, assuming no clipping to constraints is needed, is given by

$$\begin{aligned} f(\alpha+\hat{\delta}s) &= \tfrac{1}{2}(\alpha+\hat{\delta}s)^T\, Q\, (\alpha+\hat{\delta}s) + (\alpha+\hat{\delta}s)^T p, \\ &= f(\alpha) + \tfrac{1}{2}\hat{\delta}^2 s^T Q s + \hat{\delta}\alpha^T Q s + \hat{\delta}s\cdot p, \\ &= f(\alpha) + \tfrac{1}{2}\hat{\delta}^2 s^T Q s + \hat{\delta}s\cdot\nabla f(\alpha), \\ &= f(\alpha) - \tfrac{1}{2}\frac{(s\cdot\nabla f(\alpha))^2}{s^T Q s}, \\ &= f(\alpha) - \tfrac{1}{2}\frac{(-[\nabla f(\alpha)]_u - y_u y_l [\nabla f(\alpha)]_l)^2}{Q_{uu}+Q_{ll}-2Q_{ul}}, \end{aligned}$$

where it is made use of the fact that the unbounded optimal stepsize is $\hat{\delta} = \frac{-s\cdot\nabla f(\alpha)}{s^T Q s}$. What Fan et al. propose is to select one element of the working set using WSS1 (for instance, u) and then choose the other in order to maximize the decrease in the objective function. This results in the rules

$$\begin{cases} u &= low \equiv \operatorname{argmax}_i \{-[\nabla f(\alpha)]_i \quad \text{s.t.} \quad i\in I\}, \\ l &= \operatorname{argmax}_j \left\{ \frac{(-[\nabla f(\alpha)]_u - y_u y_j [\nabla f(\alpha)]_l)^2}{Q_{uu}+Q_{jj}-2Q_{uj}} \quad \text{s.t.} \quad j\in J, -[\nabla f(\alpha)]_j < -[\nabla f(\alpha)]_u \right\}. \end{cases}$$

The new rule for l is also doable in almost linear time, as it just requires evaluating the decrease obtained for each possible $j = 1,\ldots,N$, and each such evaluation has cost $O(\bar{d})$ because of the requirement to compute entries Q_{uj} which, in turn, call the kernel function. The resultant cost for WSS2 is $O(\bar{d}N)$, though its faster convergence speed in terms of iterations more than justifies this additional burden over the $O(N)$ time required by WSS1. It should also be noted that the cost of the SMO algorithm is still dominated by the update of gradient $O(\bar{d}N)$, so WSS2 does not really increase the algorithm's overall complexity. Convergence for this kind of WSS has also been proved [63], and nowadays it is widely regarded as the best performing strategy in SMO implementations.

2.4.2.4 LIBSVM: caching and shrinking

To close this section two improvements that further boost the performance of SMO algorithms are reviewed: caching and shrinking, which were first introduced by Joachims [58]. The use of these heuristics together with WSS2 form the popular **LIBSVM** method [30] by Chang and Lin, which can be considered as the *de facto* standard in non-linear SVM training software.

On the one hand, **caching** is a memory management strategy that successfully reduces the overhead caused by the evaluations of the kernel function by storing in memory a subset of the rows of the matrix Q. It is based on the observation that at each iteration of SMO, only the rows/columns $Q_{l,:}$ and $Q_{u,:}$ corresponding to the working set are required (Q is symmetric). If these rows need to be computed every time they are needed, the cost of SMO per iteration is in the order of $O(\bar{d}N)$, with kernel evaluations dominating the cost. If, however, these rows were already precomputed and available in memory, the cost would drop down to $O(N)$. However, and as already stated at the beginning of this section, storing the full matrix Q into memory is infeasible. What caching proposes, thus, is an intelligent way to maintain in memory a subset of the rows of Q so that the cost per iteration is reduced.

Caching works as follows. At each iteration of SMO, the two rows corresponding to the selected (l, u) working set are requested from a kernel cache. If these rows happen to be available in this cache their values can be used immediately without further computations. Else, the non-available rows are computed from scratch and added to the cache. When the size of the cache grows above a used-defined memory limit, rows are dropped following a Least Recently Used (LRU) policy to reduce memory usage. This policy ensures that the removed rows have not been accessed recently, thus avoiding repeated computations of highly demanded rows. In this way the kernel cache effectively helps to reduce the number of kernel function evaluations throughout the whole algorithm. As expected, this effectiveness greatly depends on the amount of memory available for the cache; more details on this are given in the experimental sections of Chapter 3.

Shrinking, on the other hand, is based on more theoretical properties of the SMO algorithm. As shown by Chen et al. [64], there is a point in the algorithm after which the coefficients at bound $\alpha_i \in \{L, C\}$ that are not at a support hyperplane in the optimum ($w^* \cdot x_i + b^* \neq \pm 1$) remain at that bound until the end of the algorithm. In other words, those coefficients have already attained their optimal α_i^* value, and therefore, could be held fixed during the rest of the algorithm, guaranteeing the same final solution. This property can be regarded as an extension of Vapnik's motivations for the chunking algorithm [8]. The shrinking strategy takes advantage of this by "freezing" those α coefficients that are located at the boundaries of the problem $\{L, C\}$ and are likely to remain there, so that the optimization can continue without considering them (the hyperplane condition $w^* \cdot x_i + b^* \neq \pm 1$ is ignored). Different implementations of the shrinking strategy depend on the criterion used to decide which coefficients should be shrunk. In LIBSVM, after a certain number of iterations a shrinking step is performed, in which the gradient of the α_i at bound is checked. If the gradient indicates the coefficient is likely to remain there (see [64] for details), that α_i is "shrunk", and the algorithm continues working on the non-shrunk coefficients.

To prevent bad quality solutions arising from too early shrunk coefficients, once a solution satisfying the stopping criterion is attained, the problem is reconstructed, i.e. every coefficient is activated again, and once again the algorithm continues considering every coefficient. This reconstruction step involves recomputing the gradient for every shrunk coefficient, which can be very expensive. In LIBSVM this step is performed in an efficient way by noting that each entry of gradient can be written as

$$\nabla[f(\alpha)]_i = p_i + L \sum_{\alpha_j = L} Q_{ij} + \sum_{\alpha_j \in (L,C)} \alpha_j Q_{ij} + C \sum_{\alpha_j = C} Q_{ij}.$$

The second term has no effect in the computation if $L = 0$ (which is the case for SVM classification, SVR, and One-class), and the computation of the last one can be done efficiently by maintaining in memory the sums $\bar{G}_i = \sum_{\alpha_j = C} Q_{ij}$, which only need to be updated when a coefficient hits or leaves the C boundary. Note however that there is no fast way to compute the third term, which must be recomputed from scratch.

It must be noted that because shrinking only identifies those coefficients that with some probability are at their optimal value, one can find cases where this identification fails and no faster or even slower convergence is attained. Therefore, shrinking is regarded as a heuristic procedure that may or may not work. Despite this, its use is generally recommended, and furthermore is used jointly with caching: shrunk coefficients do not need to appear anymore in cache, and thus neither their Q rows nor their columns are stored, making room for other non-shrunk rows.

Joining WSS2, caching and shrinking, LIBSVM's implementation of SMO, can be considered the state-of-the-art in non-linear SVM training. Nevertheless, there is still room for improvement, and as shown in the contributions of this thesis in Chapter 3, LIBSVM can still benefit from new strategies.

2.4.3 Geometric SVM solvers

In parallel with the development of primal and dual algorithms for the SVM problem, a series of methods from the field of geometry have been adapted to address the alternative, geometric formulation of the SVMs for classification. A brief overview of the most relevant methods of this kind is given in this section.

The first successful attempt to solve the geometric SVM problem was made by Franc and Hlavăc [65] as the **Schlesinger–Kozinec** (SK) algorithm, sometimes also referred as the **Gilbert–Schlesinger–Kozinec** (GSK) algorithm, as its motivation can be traced back to the work of Gilbert [66]. The algorithm originates in a procedure designed to

Algorithm 4 Schlesinger–Kozinec (simplified)

Inputs: convexs set $\mathcal{X}_+, \mathcal{X}_-$.
Initialize $w_+ \in \mathcal{X}_+$, $w_- \in \mathcal{X}_-$.
while not at optimum **do**
  Find point m with least margin to the hyperplane $w = w_+ - w_-$.
  Move $w_\pm$ in the same hull as m so as to minimize distance to the opposite $w_\mp$.
end while
Return (w_+, w_-).

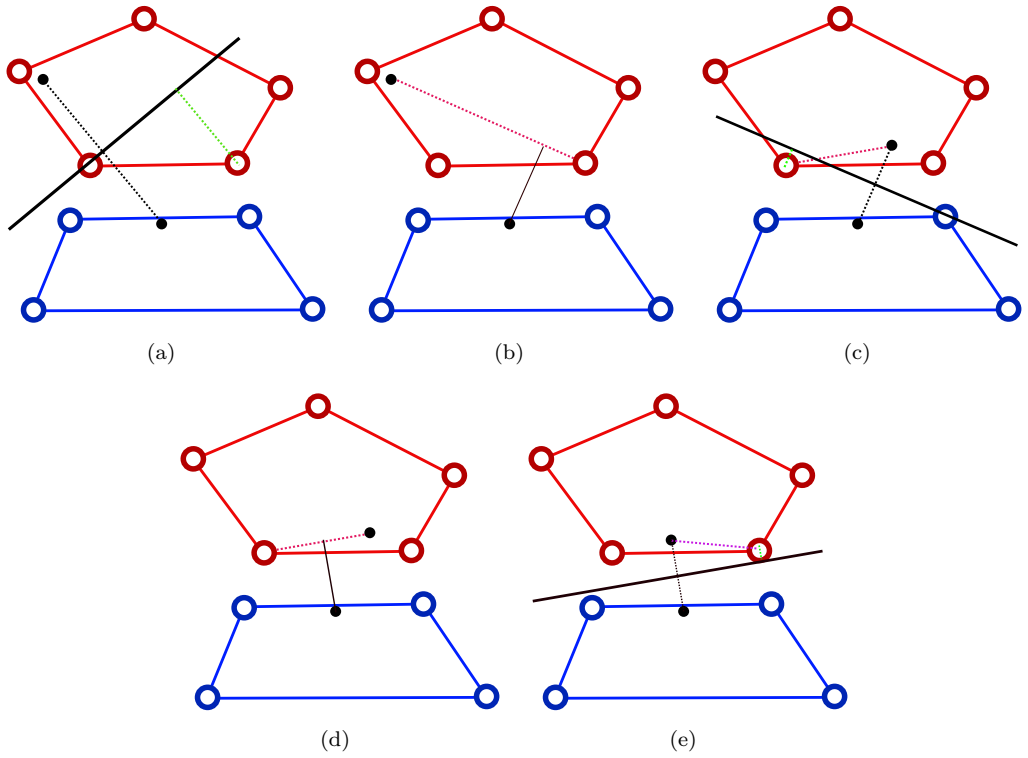


FIGURE 2.9: Example of iterations performed by the GSK algorithm. a) The point with least margin is identified, which turns out to belong to the set $\mathcal{X}_+$ (red). b) w_+ is displaced towards this least-margin point to minimize distance to w_-. c-e) This procedure is repeated until convergence.

solve the **Minimum Norm Problem** (MNP), which pursues finding the point x^* inside convex set such that the norm $\|x\|_2$ is minimized. This problem can be easily generalized to finding the closest points in two convex sets by alternatively optimizing for each one of the sets, minimizing at each step the norm of the point in that set and considering as the origin the point in the other set. The resulting algorithm is given as Algorithm 4, in which at each step the data point with least margin is selected for update. It should be noted that a point at the wrong side of the hyperplane is considered to have less margin than a correctly classified one, even if its actual distance to the hyperplane is smaller [1]. An example showing some iterations of the algorithm is depicted in Figure 2.9.

[1] This is sometimes referred as **signed margin**

Adapting this algorithm to work when the convex sets are defined as the convex hull of a set of points and adding kernelization results in the algorithm proposed by Franc and Hlavăc [65]. This is done by expressing the points $w_\pm$ as $w_\pm = \sum_{i\pm} \alpha_i \Phi(x_i)$, that is, making explicit use of the convex hull structure for convex combination weights α. With this, the method is able to solve the problem

$$\begin{aligned} \min_{\alpha} \quad & \left\| \sum_{i+} \alpha_i \Phi(x_i) - \sum_{i-} \alpha_i \Phi(x_i) \right\|_2^2, \\ \text{s.t.} \quad & \begin{cases} \sum_{i+} \alpha_i = 1, \\ \sum_{i-} \alpha_i = 1, \\ \alpha \geq 0, \end{cases} \end{aligned}$$

which after some algebra can be rewritten as

$$\begin{aligned} \min_{\alpha} \quad & \alpha^T Q \alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = 0, \\ 0 \leq \alpha \leq 1, \\ \alpha \cdot e = 2, \end{cases} \end{aligned}$$

i.e. the hard–margin geometric SVM model (Equation 2.11). The method presents costs per iteration of the same order as the SMO algorithm ($O(\bar{d}N)$), though its convergence rate is much poorer due to the fact that a large number of iterations are needed to achieve a good enough solution. This is because even if the solution of the problem is always at a vertex or a face of the convex sets, SK can only approach the boundary of the sets asymptotically (as seen in the example).

A further SK extension by Mavroforakis and Theodoridis [67] addresses the soft-margin case

$$\begin{aligned} \min_{\alpha} \quad & \alpha^T Q \alpha, \\ \text{s.t.} \quad & \begin{cases} \alpha \cdot y = 0, \\ 0 \leq \alpha \leq \nu, \\ \alpha \cdot e = 2, \end{cases} \end{aligned}$$

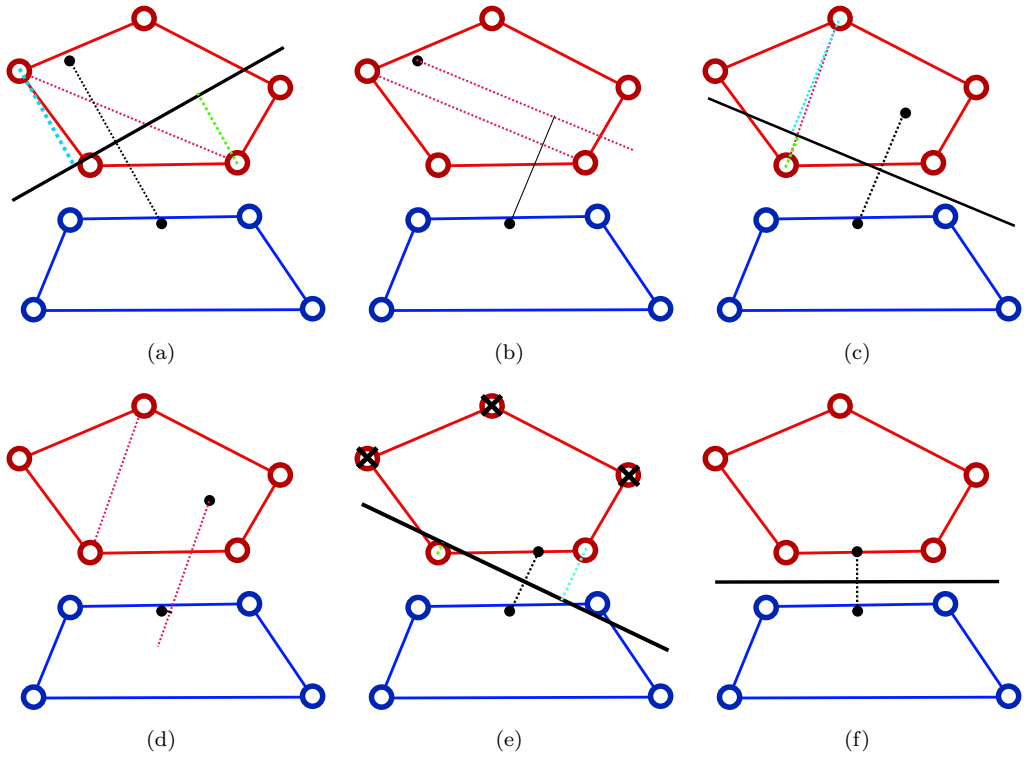


FIGURE 2.10: Example of iterations performed by the MDM algorithm. a) The points with least and most margin are identified. b) w_+ is displaced the direction given by the selected points so as to minimize distance to w_-. c-e) This procedure is repeated. f) The optimal solution is found. Note how in e) the crossed points are not eligible, as they no longer contribute to w_+ ($\alpha_i = 0$).

in which the hulls are replaced by reduced convex hulls with a reduction parameter ν. This approach, though, increases the cost per iteration to $O(N \log(N))$ due to some required sorting operations.

Improving on SK, the **Mitchel–Dem'yanov–Malozemov** (MDM) algorithm [68] manages to solve SK convergence issues. To do so, the update is performed in the direction given by the point with least margin and the one with largest margin contributing to $w_\pm$ (i.e. $\alpha_i > 0$), with the requirement that both of them must belong to the same convex set. This avoids asymptotic approximations to the boundary of the convex sets, hence requiring less iterations for convergence. Even though MDM roughly requires twice as many operations than SK per iterate, most of the times it pays off, resulting in faster training times. Figure 2.10 shows an example of some iterations of the algorithm. As in SK, the original method could only solve the hard-margin version of the problem, though it was later extended to the soft-margin case first by Tao et al. [69] and then in a more effective way by López, Barbero and Dorronsoro [70, 71].

As shown in [70], MDM turns out to be a restricted version of the SMO algorithm when WSS1 is used. Indeed, l coincides with the point with least margin, u with the point with

largest margin and $\alpha_i > 0$, and the restriction comes in the form of l and u belonging to the same convex set, that is, to the same class. This observation is backed up by the fact that the ν-SVM formulation (see Section 2.2.5) can be shown to be equivalent to the geometric SVM [43], and in the LIBSVM software [30] this SVM variant is solved by an adapted SMO procedure in which both l and u are forced to belong to the same class. This equivalence between algorithms is relevant because any improvement in one of the methods could be, in principle, applied to the other one and, in fact, in Chapter 3 an effective use is made of this fact.

Chapter 3

Accelerating SVM training

"The mind that opens to a new idea
never comes back to its original size."
Albert Einstein

In this chapter the first major contribution of this thesis is presented: a method to reduce the number of iterations required by Sequential Minimal Optimization for convergence, while maintaining its computational cost per iteration. The method originates in observations of the behavior of the Mitchel–Dem'yanov–Malozemov algorithm for geometric SVMs, which because of its way of constructing updating directions suffers from slow convergence towards the optimum in some situations. A geometry-inspired algorithm termed as the **Cycle-Breaking** method is proposed to address these flaws, which is then applied to the SMO algorithm making use of the links between both methods. Once in SMO, Cycle-Breaking is further improved to form the **Momentum Sequential Minimal Optimization** (MSMO) algorithm, which once adapted to the generalized SVM formulation of Section 2.2.4 is effectively applied for SVM classification, Support Vector Regression, One-class SVM and Least–Squares SVM. The chapter closes with a thorough analysis on the compatibility of MSMO with the alternative accelerating strategies of caching and shrinking, providing notions about when MSMO can provide useful improvements in SVM training times.

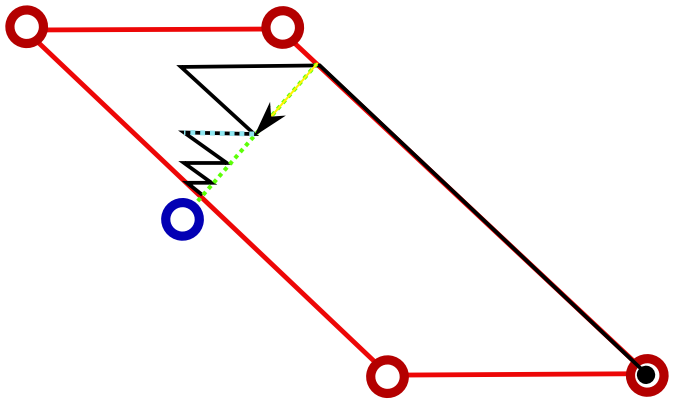


FIGURE 3.1: An example where the MDM method zigzags towards the optimum. One class consists in a single point where the other one is made up of four.

3.1 Cycle-Breaking

3.1.1 Deficiencies of the MDM algorithm

The Mitchel–Dem'yanov–Malozemov algorithm (Section 2.4.3) can be regarded as a the best-performing method for solving the optimization problem posed by geometric SVMs. It is, as well, a constrained version of the Sequential Minimal Optimization (Section 2.4.2) method applied to this particular problem where the two coefficients (l, u) of the working set are forced to belong to the same class. While because of this constraint MDM results in a slower algorithm than SMO (it solves a harder problem) [70], its intrinsic geometric interpretation allows a more careful study of the behavior of this method. For instance, when not employing the kernel trick and using 2-dimensional data, regardless on the number of training patters, it is possible to produce a 2-dimensional plot representing the two convex hulls and the evolution the points (w_+, w_-) follow until the solution of the problem is found.

Because of this and thanks to observation it was realized that situations like the one depicted in Figure 3.1 could take place, in which the algorithm zigzags towards the optimum by repeating several times the same choices of working set selections, i.e. $(l_1, u_1), (l_2, u_2), \ldots, (l_M, u_M), (l_1, u_1), (l_2, u_2), \ldots, (l_M, u_M), (l_1, u_1), \ldots$ and so on until a boundary of the convex set is hit, making one of the u_i in the repetition no longer available (because $\alpha_i \to 0$) or a pair (l_j, u_j) out of the repetition becomes a better working pair (because of the changes in $w_\pm$). Until either of these events takes place the algorithm is optimizing over and over using the same $(l_1, u_1), (l_2, u_2), \ldots, (l_M, u_M)$ working sets, that is, repeating several times the same **cycle**. While this situation might appear to be a degeneracy of the method, it actually turns out to be reproduced also in high-dimensional situations (as shown later on experimentally).

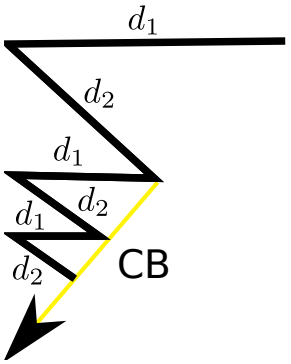


FIGURE 3.2: A cycle made up by the two directions d_1 and d_2 can be broke down by building the cycle-breaking direction shown.

An explanation to this effect from the point of view of optimization can be given. As stated before, SMO with WSS1 (and thus MDM) is nothing but the SVM^{light} algorithm when a working set of size $q = 2$ is used [62]. As SVM^{light} working set selection follows a modified Zoutendijk's method (see Section 2.4.1), in essence it is selecting the updating direction s with only 2 non-zero entries producing the largest decrease in the objective function, in a first–order sense. Methods using only first–order information (gradient) to select their updating directions are known to show **zigzagging** effects [25]: the gradient descent method is the best example of this. Conversely, second–order methods like Newton descent or Conjugate Gradient [72] get rid of this inefficient zigzagging by making use of Hessian information, though at a higher cost per iteration. MDM and SMO not only use just first–order information, but further limit themselves to update the minimum possible number of the problem's variables per iteration. This is in line with classic methods such as **coordinate optimization**, where a simple line optimization along each one of the variables is done at a time. These methods, though requiring minimal computational costs per iteration, further suffer these zigzagging effects, resulting in a large number of iterations till convergence. Using a "second-order" working set selection (WSS2) only solves the problem partially, since just two rows from the full Hessian are employed and still only two variables are updated per iteration. Hence, zigzagging is still bound to appear, though with less frequency than in WSS1.

Back to MDM, the appearance of cycles is an inefficiency of the method, because usually little improvement in the objective function is gained at each step and, furthermore, the cycle repetition can be completely overcome by performing an optimization step following a suitable updating direction, different than the standard one chosen by MDM as a function of (l, u). In the simple example of Figure 3.2 the cycle being repeated is made up of just two working sets (l_1, u_1) and (l_2, u_2). By denoting the updating directions defined by these sets as d_1 and d_2, it is clear that by optimizing along the combined direction $d = s_1 + s_2$, the end of the cycle repetitions is reached in a single iteration. Therefore, looking for these **cycle-breaking directions** seems an appealing way to

solve this problem and hence to reduce MDM's number of iterations for convergence. The question is how these directions can be constructed and how the algorithm can realize that a cycle is taking place.

3.1.2 The Cycle-Breaking method

The first instantiation of the cycle-breaking idea was made as the **Cycle-Breaking method** applied to the MDM algorithm [73], and later on extended to WSS1 SMO [74], WSS2 SMO [75] and Support Vector Regression [76]. This was before the generalized SVM formulation used in this thesis was considered, though for the ease of presentation, the description of the algorithm here is given in terms of an SMO solving this formulation, considering MDM is a particular case of it where $y_u y_l = 1$ since (u, l) must belong to the same class.

First of all, the SMO algorithm was modified to maintain an updates queue $\mathcal{Q}$ containing information about the working sets (l, u) and the updating step δ used during the past τ iterations (τ being a queue memory parameter). This allows to remember the decisions taken by SMO during the last τ steps, and to decide whether a cycle is taking place using this information. Specifically, at each iteration and after SMO has issued a new working pair (l, u) to use in the current iteration, it is checked whether the same pair already appears in $\mathcal{Q}$. If so, it is decided that a cycle is taking place, it being formed by the updates performed between the repetition found in $\mathcal{Q}$ and the most recent one, and a cycle-breaking step is applied. Otherwise a standard SMO step is performed, adding the current working set and resultant stepsize (l, u, δ) as the most recent element of $\mathcal{Q}$. The oldest element from $\mathcal{Q}$ is dropped if $\mathcal{Q}$ already contains τ elements.

To perform a cycle-breaking step, suppose the detected cycle is given by the previous iterations $(l_1, u_1, \delta_1), \ldots, (l_M, u_M, \delta_M)$. Following the intuitions presented before, a cycle-breaking direction d is built as

$$d = \sum_{i=1}^{M} \delta_i (e_{u_i} - y_{u_i} y_{l_i} e_{l_i}), \tag{3.1}$$

that is, summing up the directions making the detected cycle. Then the subproblem of optimizing the objective function following this direction is solved, which takes the form

$$\begin{aligned} \min_{\delta} \quad & \tfrac{1}{2}(\alpha + \delta d)^T Q\, (\alpha + \delta d) + (\alpha + \delta d)^T p, \\ \text{s.t.} \quad & L \leq \alpha + \delta d_i \leq C \quad \forall\, d_i \neq 0. \end{aligned}$$

Because each of the basic directions $s_i = \delta_i(e_{u_i} - y_{u_i} y_{l_i} e_{l_i})$ meets the constraint $(\alpha + \delta_i s_i) \cdot y$, this constraint is also met for d. Consequently, there is no need to consider it here. Despite the direction d being more complex than the usual s_i directions, the resulting subproblem presents the same structure as the problem to solve at each step of SMO (Equation 2.12), but for the larger number of box-constraints, which depend on the number of non-zero entries of d. Therefore, the unconstrained optimum of the problem can be computed similarly as

$$\hat{\delta} = \frac{-d \cdot \nabla f(\alpha)}{d^T Q d}. \tag{3.2}$$

The computation of $d \cdot \nabla f(\alpha)$ can be made fast by exploiting the sparsity of d, i.e.

$$d \cdot \nabla f(\alpha) = \sum_{d_i \neq 0} d_i [\nabla f(\alpha)]_i.$$

Since d is made up of at most τ previous updates (because of the updates queue $\mathcal{Q}$ limitation) and each of this updates modifies exactly two entries, at most 2τ entries of d are non-zero. Therefore, $d \cdot \nabla f(\alpha)$ can be computed in $O(2\tau)$ time (recall the gradient is always maintained in memory in SMO). Similarly, the calculation of the denominator $d^T Q d$ can also take advantage of this sparsity by defining first $U = Qd$, i.e.,

$$U_i = \sum_{d_j \neq 0} Q_{i,j} d_j, \tag{3.3}$$

and then using

$$d^T Q d = \sum_{d_i \neq 0} d_i U_i.$$

The computation of U requires $O(2\tau N \bar{d})$ operations, $O(2\tau \bar{d})$ per each U_i entry, $\bar{d}$ because of the evaluation of the kernel function for Q ($\bar{d}$ average number of non–zero features). After that, only $O(2\tau)$ operations are needed to obtain the denominator $d^T Q d$. Overall, the cost for computing the unconstrained stepsize $\hat{\delta}$ is dominated by the computation of U, i.e. $O(2\tau N \bar{d})$.

Replicating the arguments given for SMO (Section 2.4.2), the constrained optimum δ^* is obtained by clipping down $\hat{\delta}$. To do so, each one of the box constraints is checked, clipping the value of δ if needed, as

Algorithm 5 Cycle-Breaking method

Inputs: x, y, p, L, C, Q, τ.
Initialize $\mathcal{Q} = \varnothing$, $\alpha = \nabla f(\alpha) = 0$.
while stopping criterion not met **do**
  Select working set (l, u).
  if $(l, u) \in \mathcal{Q}$ **then**
    Build Cycle-Breaking direction d using Equation 3.1.
    Compute U and unconstrained stepsize $\hat{\delta}$ using Equation 3.3 and Equation 3.2.
    Clip down $\hat{\delta}$ using Equation 3.4 to obtain δ^*.
    Update: $\alpha \leftarrow \alpha + \delta^* d$, $\nabla f(\alpha) \leftarrow \nabla f(\alpha) + \delta^* U$.
    Reset $\mathcal{Q} = \varnothing$.
  else
    Perform standard SMO step.
    Add (l, u, δ^*) to $\mathcal{Q}$, drop oldest from $\mathcal{Q}$ if more than τ elements exist.
  end if
end while
Compute bias b^*.
Return (α^*, b^*).

$$\delta = \begin{cases} \min\left\{\delta, \frac{C-\alpha_i}{d_i}\right\} & \text{if} \quad d_i > 0, \\ \min\left\{\delta, \frac{L-\alpha_i}{d_i}\right\} & \text{if} \quad d_i < 0, \end{cases} \qquad \forall\, d_i \neq 0. \tag{3.4}$$

After applying this clipping procedure, the resulting δ is δ^*, and so the update is performed as $\alpha \leftarrow \alpha + \delta^* d$. The only point left to address is the update of the gradient $\nabla f(\alpha)$, a crucial step to maintain SMO's performance. This is straightforward to do upon realization that

$$\begin{aligned} \nabla f(\alpha + \delta^* d) &= Q(\alpha + \delta^* d) + p, \\ &= \nabla f(\alpha) + \delta^* Q d, \\ &= \nabla f(\alpha) + \delta^* U, \end{aligned}$$

that is, the update in the gradient is immediately given by the stepsize δ^* and the vector U already computed to obtain δ^*. Therefore the update of the gradient just requires $O(N)$ operations.

A pseudocode of the algorithm is shown as Algorithm 5. In practice some additional measures are taken to ensure good performance of the cycle-breaking steps:

- The slope of the standard direction suggested by SMO $\nabla f(\alpha) \cdot s$ is compared against the slope of the cycle-breaking direction $\nabla f(\alpha) \cdot d$. The cycle-breaking

update is only performed if $\nabla f(\alpha) \cdot d < \nabla f(\alpha) \cdot s$, that is, if the improvement in a first–order sense is larger than the one provided by a standard update. Else, a standard update is performed and (l, u) and every older entry in $\mathcal{Q}$ is removed to prevent trying to break this cycle again. This avoids running cycle-breaking steps that provide little improvement. While when τ is well–tuned such situations do not take place very frequently, still some savings can be obtained following this rule.

- If $\hat{\delta} \neq \delta^*$ in a standard step, i.e. a boundary of the problem is hit, $\mathcal{Q}$ is reset to $\varnothing$. This prevents generating infeasible cycle-breaking directions, as d would try to push α further beyond the boundary.

A value for the new parameter τ has to be chosen to run the algorithm. τ constitutes a trade-off between a low cost per cycle-breaking step and the ability to detect and break long cycles (though large values of τ might be prone to detect spurious cycles). In practice, therefore, small values of $\tau \in [10, 30]$ generally work best. In any case, usually $\tau \ll N$. Further discussion on this parameter is given later in the experimental results (Section 3.1.3).

Cost per iteration depends on whether a standard SMO step or a cycle-breaking step is performed. The standard step retains its original cost ($O(N\bar{d})$) while the cost of the cycle-breaking step is dominated by its most expensive computation, the calculation of U, which involves at most $O(2\tau N\bar{d})$ operations. This means a cycle-breaking step could take up to the cost of 2τ steps; fortunately in practice the number of non-zero entries in the cycle-breaking direction d is usually very small, making this cost smaller, and so performing cycle-breaking steps pays off.

As a final remark, it should be noted how this cycle-breaking strategy is reminiscent of the Hooke–Jeeves algorithm [77] for accelerating the cyclic coordinate descent method. This method is a special case of coordinate optimization where the variable over which to optimize at each step is selected using a policy such that, after a number T of iterations, each one of the problem's variables has been optimized over at least once. The simplest form of this method involves optimizing each variable sequentially $(\alpha_1, \alpha_2, \ldots, \alpha_N, \alpha_1, \ldots)$ and so $T = N$. The Hooke–Jeeves algorithm improves over the cyclic coordinate descent method by building an accelerating direction d after each optimization cycle, summing up in d the previous T directions and then performing a line search along d. In this respect, the Cycle-Breaking method can be regarded as a modification of the Hooke–Jeeves algorithm in which accelerating directions are only build when there is a reasonable expectation of the utility of those directions, and the

Dataset	N	d	Splits	$\log_{10}(C)$	$\log_{10}(2\sigma^2)$
thyroid	140	5	100	1	0.5
heart	170	13	100	1	2.3
titanic	150	3	100	0.7	0.3
breastcancer	200	9	100	1.2	1.7
diabetes	468	8	100	0	1.3
german	700	20	100	0.5	1.7
flare	666	9	100	0	1.5
image	1300	18	20	2.7	1.5
splice	1000	60	20	3	1.8
banana	400	2	100	2.5	0.0
ringnorm	400	20	100	9	1
twonorm	400	20	100	0.5	1.6
waveform	400	21	100	0	1.3

TABLE 3.1: Training set sizes N, input dimensions d and training-test splits of the datasets used in the Cycle-Breaking experiments, along the recommended C and σ parameters.

underlying algorithm (SMO) selects the variables to update as a function of gradient information instead of following a cyclic policy.

3.1.3 Experimental results

Experimental results supporting the cycle-breaking method had been already carried out in references [73–76]. For the sake of clarity of presentation and because the main contribution of this chapter is the MSMO algorithm (presented in Section 3.2), here only the most representative results for SVM classification using SMO with WSS2 are presented. It should be noted that these results apply as well for Support Vector Regression and the MDM algorithm.

The experimental setup was as follows. Benchmark datasets were taken from Gunnar Rätsch's [35] online repository. These are distributed in the form of 100 training / test splits per dataset (but for two datasets which only present 20 splits), and recommended values for the SVM parameters are provided as well. Table 3.1 presents the size, number of splits and parameters for these datasets. As a stopping criterion for SMO, a KKT

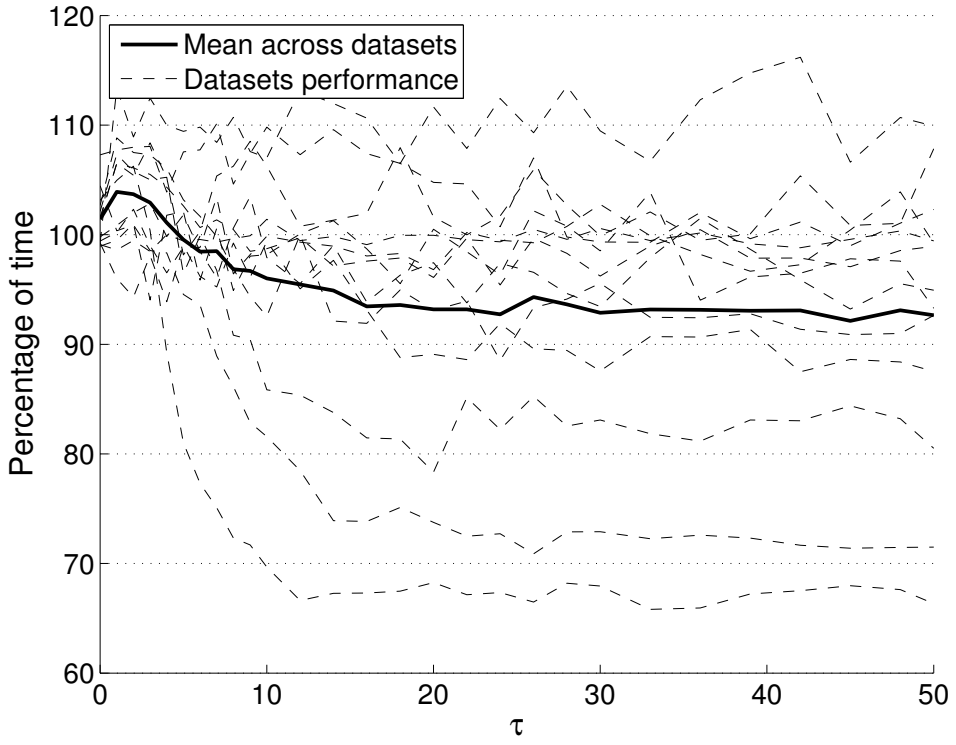


FIGURE 3.3: Percentage of time required by Cycle-Breaking w.r.t SMO for the benchmark datasets. Dashed lines present results for individual datasets, while the bold line is an average across datasets.

violation smaller than 10^{-5} in the most violating pair must be achieved. The kernel function used was always the gaussian kernel $k(x_i, x_j) = e^{-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}}$.

The SMO algorithm as well as the Cycle-Breaking method were implemented from scratch in the C programming language. A simple caching strategy that stores up to 100 rows of Q was included in the implementation as well. Shrinking was not used.

In the first place, the influence of the τ parameter over training times was studied. Figure 3.3 presents reductions in training times attained by the Cycle-Breaking method w.r.t standard SMO times, for a range of values of the τ parameter. Small values of τ slightly degrade the performance of the algorithm, while from $\tau = 5$ on improvements are observed, up to the point $\tau \simeq 20$ beyond which the behavior of the method does not change significantly. The interpretation of this effect is the following: small τ values can only detect very small cycles, and thus the larger, probably more inefficient cycles are left unchanged. However as the Cycle-Breaking strategy imposes additional burdens on computational times, applying it only to such small cycles does not seem to pay off. Regarding the constant behavior of the method for large τ values, it should be realized that whatever the size $\mathcal{Q}$ is allowed to grow to, if no cycles of such large sizes take place during the algorithm the result will be the same. Therefore, it seems that in practice large cycles do not take place, or are marginal when compared with the occurrence of

Dataset	Times SMO	Times CB	Reduction
thyroid	3.02	2.90	96.10 %
heart	8.07	7.52	93.11 %
titanic	3.67	3.73	101.62 %
breastcancer	31.90	24.99	78.33 %
diabetes	40.50	40.70	100.50 %
german	155.83	138.83	89.09 %
flare	83.63	81.49	97.44 %
image	1818.10	1340.80	73.75 %
splice	1470.00	1469.30	99.95 %
banana	406.34	277.33	68.25 %
ringnorm	44.17	49.32	111.65 %
twonorm	16.29	17.07	104.78 %
waveform	22.57	21.85	96.81 %
Average			93.18 %

TABLE 3.2: Average running times (in milliseconds) for SMO WSS2 and Cycle-Breaking (CB); the reduction in % is also given.

smaller cycles. Under the light of these results, it seems that an adequate value for this parameter would be $\tau = 20$.

Table 3.2 presents detailed running times for every dataset and $\tau = 20$. An average reduction factor of 93.18 % is obtained, with large variations across datasets: while for *banana* Cycle-Breaking requires only 68.25 % of the time of SMO, in *diabetes* both algorithms perform similarly, and in *ringnorm* Cycle-Breaking takes even more time than SMO. The causes of these variations will be clear when analyzing them in the context of the MSMO method (Section 3.2), and so they are not addressed here.

As a conclusion of these experiments, it can be stated that the basic Cycle-Breaking idea works, breaking cycles even in high-dimensional (kernel) spaces, though regrettably, the improvements obtained are not significant enough to be of practical use. Fortunately, the method can be further enhanced to augment its effectiveness; the next section presents how this can be realized.

3.2 Momentum Sequential Minimal Optimization

In this section another SMO accelerating method is built upon the Cycle-Breaking ideas, resulting in the **Momentum Sequential Minimal Optimization** (MSMO) algorithm. This improved method is able to attain significant reductions in the number of iterations required by SMO for convergence, while not increasing noticeably the cost per iteration. Depending on the situation and its combination with other accelerating methods such as caching and shrinking, the method is also able to attain relevant savings in training times.

3.2.1 Momentum motivation

As already discussed in Section 3.1, the SMO algorithm builds its updating directions by only making use of first–order information, and, furthermore, restricting them to make use of the minimal number of α_i coefficients, which turn out to be 2. While this approach has large advantages like getting rid of the necessity of using a QP solver (Section 2.4.2), the excessive simplicity in the updates leads to zigzagging effects towards the optimum, resulting in slow convergence in terms of number of iterates. Because of this, a possible way to improve SMO performance would be to allow the use of more complex and informative updating directions, while preserving the ability to solve the resulting subproblem in closed form.

A first naive idea would be to apply a **Gradient Projected** method [25]. This implies using the updating direction $d = -\nabla f(\alpha)$ at each step and performing line optimization along d, projecting the result back to the feasible set afterwards. Since the constraints of the SVM problem are simple, this projection would reduce to a simple clipping procedure like in SMO or Cycle-Breaking. However, a severe drawback becomes evident when the optimal unconstrained stepsize is computed, as it has the form

$$\hat{\delta} = \frac{-d \cdot \nabla f(\alpha)}{d^T Q d},$$

and since d may be an all non-zeros vector, the calculation of the denominator involves $O(\bar{d}N^2)$ operations because of the entries of Q. Furthermore, caching a fraction of the rows of Q is of no use in this setting, as the whole matrix would be requested at every iteration.

The moral of this attempt is that not only the subproblem must be solvable in closed form, but also the updating direction should be sparse. The Cycle-Breaking method fulfills both conditions, but the construction of its directions is based on heuristics that

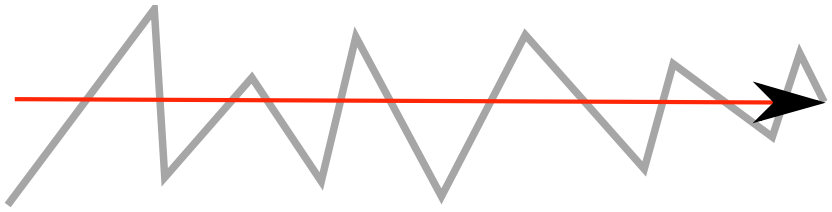


FIGURE 3.4: Depiction of the concept of general direction. While individual directions show zigzagging patterns, the general direction of movement has a clearer orientation towards the objective.

leave considerable room for improvement. The method proposed here seeks to obtain better updating directions while keeping these good properties.

Looking at the Cycle-Breaking method from a different point of view, it could be said that what the method is trying to do is to capture the **general direction of movement** of the SMO algorithm, i.e., the direction SMO follows when batches of several consecutive steps are considered. An example of this idea is depicted in Figure 3.4. While each step of the algorithm produces an update that follows a zigzag pattern when compared with the preceding and subsequent steps, if a sufficiently large number of steps are considered as a whole, it is clear that the algorithm is moving following a certain direction. Extending this concept to the limit, that is, considering every step since the beginning of the algorithm till convergence, it is obvious that the direction followed is the one taking the starting point to the optimum. The point here is that taking a reduced number of steps the general direction could still be an useful estimate of the direction towards the optimum.

The concept of general direction is broader than that of cycle repetitions: while both of them try to break zigzagging to save iterations, these zigzags might not be necessarily caused by the repetition of a cycle, i.e. the same sequence of working sets. In fact, in large training sets it is not strange to find patterns such that $\Phi(x_i) \simeq \Phi(x_j)$ but $i \neq j$, so that the working set (i, u) is as suitable as (j, u). Because of this, Cycle-Breaking might fail to identify cycles taking place, as only the indexes of the working sets are checked. Checking directly the $(\Phi(x_i), \Phi(x_j))$ values instead is unfeasible or impractical depending on the choice of kernel function, whereas using the general direction of movement could solve this issues.

In fact, the concept of general direction of movement is not new, and it has already been used in other contexts in the field of optimization and also machine learning under the idea of **momentum**. It involves modifying the updating rule of a given algorithm so that it takes the form

$$x \leftarrow x + (1 - \lambda)\delta s + \lambda m,$$

for (s, δ) the standard updating direction and stepsize issued by the optimization algorithm, m a momentum term and $\lambda \in [0, 1)$ a parameter to weigh the influence of the momentum. The momentum is usually defined as $m^t = x^t - x^{t-1}$ for iteration t of the algorithm, that is to say, it is the movement performed in the previous iteration. Since each update of x involves a momentum term, the definition of m is recursive,

$$\begin{aligned} m^t &= x^t - x^{t-1}, \\ &= (1 - \lambda^{t-1})\delta^{t-1}s^{t-1} + \lambda^{t-1}m^{t-1}, \end{aligned}$$

and so implicitly all the steps performed during the algorithm build up into m. Or, in other words, the momentum is capturing the intended general direction of movement.

Momentum terms have been widely used in the training of classic feedforward networks [3], but have also become popular recently in the spirit of cleverly combining the present x^t and previous x^{t-1} points in the optimization iterations to make a better choice for the next one x^{t+1}. Examples of this can be found in the field of image and signal processing in methods such as FISTA [78] or TwIST [79], as well as in the context of proximal splitting methods [80]. Also it has been proved by Bhaya and Kaszkurewicz [81] that when optimizing unconstrained problems with quadratic objective functions, gradient descent with momentum coincides with the second–order method of Conjugate Gradients [72] if particular choices of the weighting parameter λ are used. All these facts support the idea that the use of momentum effectively helps to improve the efficiency of the algorithm, and also that second-order information is gathered and put into use by this method.

Because of this, trying out this approach over the SMO algorithm seems a reasonable idea. However, and as stated in the beginning of this section, care must be take so as not to use non-sparse updating directions; otherwise the performance of the algorithm would degrade beyond repair. Unfortunately and due to the recursive definition of the momentum term, even if the base directions s are sparse, after a certain number of iterations m would become non-sparse, as all the preceding s accumulate into m. Therefore, momentum cannot be directly applied over SMO, and so some amendments are required.

3.2.2 The MSMO algorithm

This section details the **Momentum Sequential Minimal Optimization** (MSMO) algorithm, which puts into practice the aforementioned ideas regarding acceleration

through a momentum term. As stated before, a direct application of momentum is not practical in SMO because of the density of the resultant updating direction, and so here a **memory–limited momentum** is proposed, which takes the form

$$m^t = \sum_{r=t-\tau}^{t-1} (1-\lambda^r)\delta^r s^r,$$

in which only the s terms from the most recent τ updates are taken into account, much in the spirit of the Cycle–Breaking method. Even though this might appear as a rough approximation to the classic momentum term, it features useful properties for the problem at hand, namely, the ability to control the sparseness of the generated direction through the τ parameter (since each basic direction s^r only has two non–zero entries) and the independence of each contributing term s, which facilitates bookkeeping operations, as detailed later in the text. Also, and even though generally in this class of methods λ is a predefined, fixed parameter of the algorithm in the range $\lambda \in [0, 1)$, in this formulation it is possible to find (δ^*, λ^*) pairs that maximize the decrease in the objective function at each iteration, the range of λ being augmented to $\lambda \in \mathbb{R}$. The way the updating direction with momentum is constructed is also modified to be

$$d = (1-\lambda)s + \lambda m = \lambda(m-s) + s,$$

so that the update becomes

$$\alpha \leftarrow \alpha + \delta d,$$

that is, it can be expressed in the usual form of an updating direction together with a stepsize. A depiction of the effect of λ in the resultant direction d is shown in Figure 3.5.

The subproblem to solve at each iteration thus takes the form

$$\begin{array}{ll} \min\limits_{\delta,\lambda} & \frac{1}{2}(\alpha+\delta d)^T\, Q\, (\alpha+\delta d) + (\alpha+\delta d)^T p, \\ \text{s.t.} & L \leq \alpha + \delta d_i \leq C \quad \forall\, d_i \neq 0, \end{array}$$

which has the same structure as the subproblem appearing in a cycle–breaking step but for the need to optimize over λ as well, which is included in the problem implicitly through d. The unconstrained optimal values $\hat{\delta}$ and $\hat{\lambda}$ can be obtained by computing

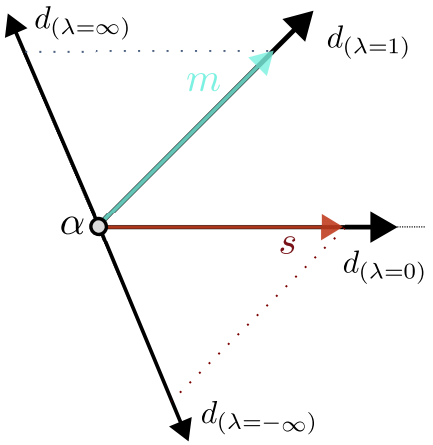


FIGURE 3.5: Effect of the weighing parameter λ on the resulting updating direction d. $\lambda = 0$ coincides with the standard SMO update.

the partial derivatives of the objective function w.r.t. to δ and λ and equaling them to zero, obtaining the system of equations

$$\begin{cases} \delta &= \frac{-\nabla f(\alpha)\cdot(\lambda(m-s)+s)}{(\lambda(m-s)+s)^T Q(\lambda(m-s)+s)}, \\ \lambda &= \frac{-\delta\nabla f(\alpha)\cdot(m-s)-\delta^2 s^T Q(m-s)}{\delta^2(m-s)^T Q(m-s)}. \end{cases}$$

These expressions can be simplified by defining the notation

$$\begin{aligned} Z &= s^T Qs = Q_{uu} + Q_{ll} - 2Q_{ul}, \\ M &= m^T Qm, \\ R &= m^T Qs, \\ H &= (m-s)^T Q(m-s) = M + Z - 2R, \\ \nabla_s &= \nabla f(\alpha)\cdot s, \\ \nabla_m &= \nabla f(\alpha)\cdot m. \end{aligned}$$

By realizing that the SMO update is included in this more general update when $\lambda = 0$, it is clear that $\hat{\delta} \neq 0$, since the SMO updating direction is guaranteed to provide a decrease in the objective function. Therefore both terms of the fraction in the equation for λ can be divided by δ^2, obtaining

$$\begin{cases} \delta &= \frac{-\lambda(\nabla_m - \nabla_s) - \nabla_s}{\lambda^2 H + Z + 2\lambda(R-Z)}, \\ \\ \lambda &= \frac{-\frac{1}{\delta}(\nabla_m - \nabla_s) - R + Z}{H}. \end{cases}$$

Plugging the expression for δ into the one for λ and working out the computations results in

$$\hat{\lambda} = \frac{-Z\nabla_m + R\nabla_s}{-H\nabla_s + (R-Z)(\nabla_m - \nabla_s)}, \tag{3.5}$$

and, consequently,

$$\hat{\delta} = -\frac{(\nabla_m - \nabla_s)(Z-R) + H\nabla_s}{-(R+Z)^2 + HZ}. \tag{3.6}$$

All the terms needed in these computations can be easily obtained by defining $U = Qm$ (similarly to Cycle-Breaking), for

$$\begin{aligned} \nabla_m = \sum_{m_i \neq 0} m_i [\nabla f(\alpha)]_i \quad &, \quad \nabla_s = [\nabla f(\alpha)]_u - y_u y_l [\nabla f(\alpha)]_l, \\ M = m \cdot U = \sum_{m_i \neq 0} m_i U_i \quad &, \quad R = U_u - y_u y_l U_l, \end{aligned} \tag{3.7}$$

where the sparse structure of m has been exploited. Details on how to compute U efficiently are given later.

These $(\hat{\delta}, \hat{\lambda})$ choices must be projected back into the feasible region. Unfortunately the subproblem is now two–dimensional, as opposed to the one–dimensional subproblems in SMO and Cycle-Breaking. Also while the box–constraints remain simple w.r.t. δ, this is not case for λ, which modifies the updating direction itself (through its angle). Because of this, a clipping strategy like the ones used before cannot guarantee an optimal projection, though an approximate method can be devised. Considering $\hat{\lambda}$ fixed, the resultant updating direction $d = (1-\hat{\lambda})s + \hat{\lambda}m$ is constructed and the stepsize $\delta = \hat{\delta}$ is clipped down as

$$\delta = \begin{cases} \min\left\{\delta, \frac{C-\alpha_i}{d_i}\right\} & \text{if} \quad d_i > 0, \\ \min\left\{\delta, \frac{L-\alpha_i}{d_i}\right\} & \text{if} \quad d_i < 0, \end{cases} \qquad \forall\, d_i \neq 0 \tag{3.8}$$

in a similar fashion to what is done in Cycle-Breaking. Nevertheless, it might be the case that the direction d produced by the angle $\hat{\lambda}$ is infeasible regardless of the updating step δ selected, and so the clipping returns $\delta^* = 0$. If this situation arises a more complex projection procedure would be needed so, to avoid these extra computations, a standard SMO update is performed instead.

Once the updating direction and stepsize have been decided, a way to update the gradient $\nabla f(\alpha)$ is needed as well. As in SMO, this can be done efficiently as

$$\begin{aligned} \nabla f(\alpha + \delta d) &= \nabla f(\alpha) + \delta Q d, \\ &= \nabla f(\alpha) + \delta Q((1-\lambda)s + \lambda m), \\ &= \nabla f(\alpha) + \delta(1-\lambda)w + \delta\lambda u. \end{aligned} \tag{3.9}$$

where $w = Qs = Q_{:,u} - y_u y_k Q_{:,l}$, with $Q_{:,i}$ the i-th column of Q. What is more, the vector U used for some of the previous calculations can be maintained in memory and updated at each iteration as well by noticing that

$$\begin{aligned} U' &= Qm', \\ &= Q(m + (1-\lambda)\delta s - (1-\tilde{\lambda})\tilde{\delta}\tilde{s}), \\ &= U + (1-\lambda)\delta w - (1-\tilde{\lambda})\tilde{\delta}\tilde{w}, \end{aligned} \tag{3.10}$$

where $(\tilde{\delta}, \tilde{\lambda}, \tilde{w})$ stand for the corresponding quantities of the oldest term contributing to m (update at time $t - \tau$), which is to be removed due to the limited memory of the momentum, unless m still does not contain τ terms. If that is the case, no removal is needed, and thus $U' = U + \delta(1-\lambda)w$. It should be noted that by storing in memory the quantities (δ, λ, w) for each term building up m this update requires no additional Q entries with respect to the ones used in a standard SMO update. This only incurs in a memory requirement of order $O(\tau N)$.

A simplified pseudocode of the whole Momentum Sequential Minimal Optimization algorithm, is presented as Algorithm 6. Similarly to SMO, an updates queue $\mathcal{Q}$ is used to store the (δ, λ, w) information from previous iterations. m is updated at each iteration, adding and removing the required values at its entries as terms enter and leave $\mathcal{Q}$. Also, in practice this simple pseudocode is implemented with the following enhancements:

Algorithm 6 Momentum Sequential Minimal Optimization

Inputs: x, y, p, L, C, Q, τ.
Initialize $\alpha = \nabla f(\alpha) = m = U = 0$, $\mathcal{Q} = \varnothing$.
while stopping criterion not met **do**
  Select working set (l, u).
  Compute MSMO variables using Equation 3.7.
  Compute unconstrained $\hat{\lambda}$ and $\hat{\delta}$ using Equation 3.5 and Equation 3.6.
  Build direction $d = (1 - \hat{\lambda})s + \hat{\lambda}m$.
  Clip down $\hat{\delta}$ using Equation 3.8 to obtain δ^*.
  if $\delta^* = 0$ **then**
    Perform standard SMO step ($\lambda = 0$).
    Reset momentum: $m = U = 0$, $\mathcal{Q} = \varnothing$.
  else
    Update $\alpha \leftarrow \alpha + \delta^* d$, $\nabla f(\alpha)$ using Equation 3.9.
  end if
  Update m, U using Equation 3.10 and the information in $\mathcal{Q}$.
  Add $(\delta^*, \hat{\lambda}, w)$ to $\mathcal{Q}$, drop oldest term if $\mathcal{Q}$ contains more than τ elements.
end while
Compute bias b^*.
Return (α^*, b^*).

- At the beginning of each iteration, the feasibility of m on its own is checked, i.e., whether there is any $m_i > 0$ and $\alpha_i = C$, or any $m_i < 0$ and $\alpha_i = L$. If this situation takes place, the momentum is pushing towards an out–of–bounds region, and so an unfeasible updating direction d is likely to appear. To avoid this, the momentum is reset and an standard SMO step is applied instead.

- If $\mathcal{Q} = \varnothing$ (algorithm starts or momentum has been reset), an SMO step is directly performed at that iteration. If as a result of this step the updated coefficients hit a boundary, the momentum is not updated, so $\mathcal{Q}$ is left as $\varnothing$. This avoids wasting time in full-fledged MSMO iterations when no momentum info is available or the momentum term in the next iteration is bound to be infeasible.

- Similarly, if a full momentum update hits a boundary and in that boundary m turns out to be an infeasible direction, the update is carried out but afterwards the momentum is reset to avoid an infeasible direction in the next iteration.

- Care is taken during the updates of the α coefficients, so that coefficients at bound ($\alpha_i \in \{L, C\}$) can be identified without distortions coming from numerical errors. While this is straightforward to do in SMO where the updating equations are simple, MSMO requires careful handling. This is of special relevance for infeasible directions tests and for the application of the shrinking algorithm.

Addressing now the computational costs of the algorithm, the quantities in Equation 3.7 only require $O(\tau)$ operations thanks to the sparsity of m. The clipping of δ and the α

update can both be done with cost $O(\tau)$. All these costs are marginal when compared with the gradient update, needing $O(\bar{d}N)$ operations for the calculation of w. This is also the order of cost of the working set selection function if WSS2 is used. It is of special significance that the new elements appearing in the gradient update only amount to $O(N)$ costs, similarly to the update of U. Therefore, it turns out that the complexity order of MSMO is also $O(\bar{d}N)$ for every iteration, as in SMO.

Of course, this does not mean that both algorithms have the same computational demands, since MSMO does require more computations to update U, obtain R, M, etc., but at costs in the $O(N)$ or $O(\tau)$ orders. Therefore, both algorithms have the same cost per iteration only for asymptotically large $\bar{d}$, or to be more precise, for asymptotically large costs of evaluation of the kernel function, which is the source of those $O(\bar{d})$ costs. These observations are essential to understand the performance of the algorithm in practice, as shown later.

3.2.3 Experimental results

Thorough experimental results analyzing the performance of the Momentum Sequential Optimization algorithm are presented here. For organization purposes, the description of the experiments here has been divided in several subsections comprising results under a variety of situations.

3.2.3.1 Comparison with Cycle-Breaking

To begin with, a comparison with the Cycle-Breaking method is mandatory, since the major claim of the MSMO algorithm is to be a generalized and improved version of this method. To do so, MSMO was run under the same experimental setting as Cycle-Breaking (Section 3.1.3), in particular using the same datasets and parameters. For this test, the MSMO method was programmed in C from scratch as well. It must be noted that this was a rough implementation just for the sake of this experiment and to be able to compare adequately against Cycle-Breaking; a better, more efficient implementation of MSMO is presented in the next subsection, which was used for the rest of the experiments.

First, the study of the influence of the τ parameter was repeated for MSMO. While the original motivations for τ in Cycle-Breaking are different than in MSMO, in essence both control the amount of information accumulating into the updating directions, and so it seems sensible to study them together. Figure 3.6 presents the degrees of reduction achieved by both methods for a range of τ values. The results for Cycle-Breaking are the

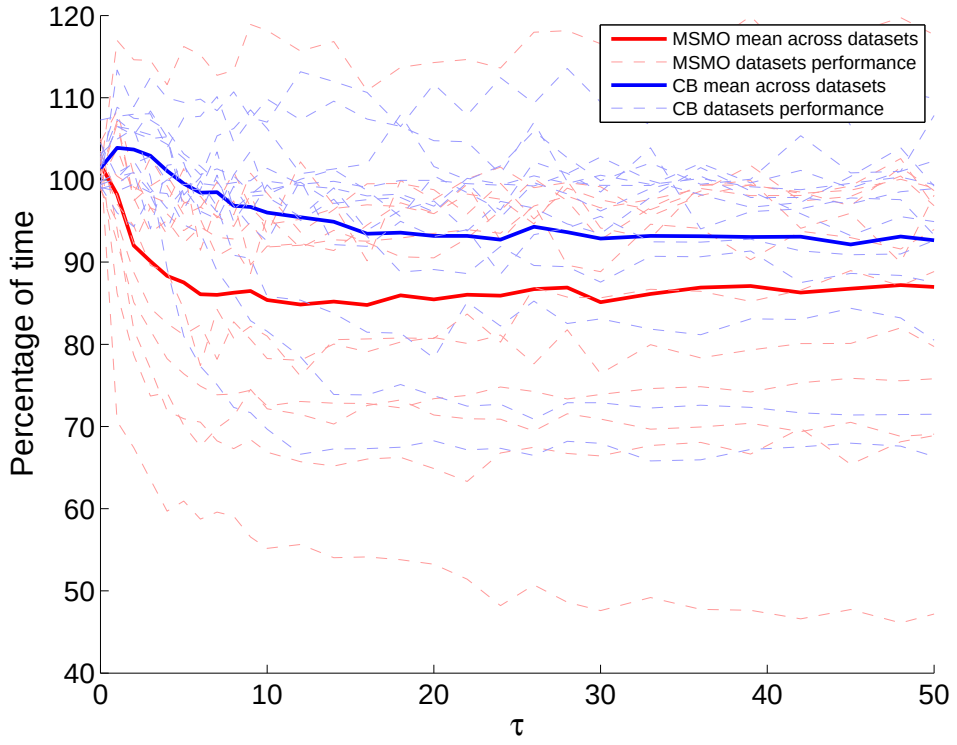


FIGURE 3.6: Percentage of time required by Cycle-Breaking and MSMO w.r.t SMO for the Cycle-Breaking benchmark datasets. Dashed lines present results for individual datasets, while the bold line is an average across datasets.

same as the ones presented in Section 3.1.3. It is immediate that MSMO performs much better than Cycle-Breaking for the full range of τ, but other facts are also remarkable. For instance, while Cycle-Breaking performs worse than SMO for a number of datasets, in MSMO this only happens for one of them. Even more, MSMO attains its best parameter value around $\tau \simeq 10$, and before that it already obtains improvements for almost every choice of τ.

Detailed time values for MSMO with $\tau = 10$ are presented in Table 3.3, and compared with the results obtained in Section 3.1.3 for the Cycle-Breaking algorithm. It is confirmed that MSMO is a superior algorithm to Cycle-Breaking, achieving an average time reduction factor of 85.37 %. Also, in the *banana* dataset MSMO is able to save almost half of the running time, while bad performance only shows up for *ringnorm*. Like in Cycle-Breaking, large variance in the results is present across datasets: an explanation for this is given later in this experimental section.

3.2.3.2 Efficient implementation of MSMO

As stated before, a better, more efficient implementation of MSMO than the one used for the previous experiments was produced. The main reason this implementation has

Method	SMO	Cycle-Breaking		MSMO	
Dataset	Times	Times	Reduction	Times	Reduction
thyroid	3.02	2.90	96.10 %	2.77	91.85 %
heart	8.07	7.52	93.11 %	6.54	81.00 %
titanic	3.67	3.73	101.62 %	3.43	93.43 %
breastcancer	31.90	24.99	78.33 %	21.33	66.87 %
diabetes	40.50	40.70	100.50 %	38.30	94.59 %
german	155.83	138.83	89.09 %	121.96	78.26 %
flare	83.63	81.49	97.44 %	76.89	91.94 %
image	1818.10	1340.80	73.75 %	1311.80	72.15 %
splice	1470.00	1469.30	99.95 %	1065.70	72.50 %
banana	406.34	277.33	68.25 %	224.19	55.17 %
ringnorm	44.17	49.32	111.65 %	52.20	118.17 %
twonorm	16.29	17.07	104.78 %	15.58	95.63 %
waveform	22.57	21.85	96.81 %	22.20	98.36 %
Average			93.18 %		85.37 %

TABLE 3.3: Average running times (in milliseconds) for SMO WSS2, Cycle-Breaking with $\tau = 20$ and MSMO with $\tau = 10$; the reductions in % are also given.

such virtues is due to the fact that it is implemented within the framework of LIBSVM [30], a popular SVM software by Chang and Lin. Such popularity is not only based on LIBSVM's ability to address different SVM models through its general SVM formulation (classification, SVR and one-class, among others) but also thanks to its careful implementation, especially with regard to its integrated caching and shrinking methods. For instance, caching seems deceivingly straightforward to implement; however a naive instantiation of this technique results in poor gains. An adequate Least–Recently–Used strategy, an efficient implementation of every cache management operation and the ability to retrieve a full cache row in a single operation are all essential ingredients for taking full advantage of a kernel cache. Similar comments could be made about the shrinking and the required bookkeeping operations during the SMO algorithm. Because of these reasons and also to able to properly compare the MSMO algorithm to a state–of–the–art method, an implementation in LIBSVM seemed desirable.

In the first place, the general SVM model solver by LIBSVM (Equation 2.8) was extended to the further generalized formulation proposed in this thesis (Equation 2.9). This merely involved adding the new parameter L, whose particular value depends on the model to solve (see Section 2.2.4) and adapting the code to consider L as the α lower bound instead of the usual 0. Making use of this extended formulation, the LS–SVM model for classification and regression was implemented as well, adding up to the list of already available models in LIBSVM.

Dataset	Task	N	d	$C \mid \nu$	γ	ϵ
phw	10-class	7227	16	10	1	∅
news	20-class	13281	62060	10	0.1	∅
letter	26-class	13294	16	10	10	∅
poker	10-class	16674	10	1	0.1	∅
census	2-class	30148	14	10	1	∅
a9a	2-class	32561	123	1000	0.001	∅
ijcnn1	2-class	35000	22	1	1	∅
shuttle	7-class	38664	9	10000	100	∅
connect-4	3-class	45038	126	10	0.1	∅
w8a	2-class	49749	300	1	1000	∅
mnist2	2-class	60000	780	100	0.01	∅
acoustic	3-class	78823	50	10	1	∅
seismic	3-class	78823	50	10	1	∅
combined	3-class	78823	100	10	1	∅
mg	regression	922	6	10	1	0.1
space	regression	2070	6	1	10	0.1
abalone	regression	2784	8	100000	0.01	0.1
cpusmall	regression	5460	12	1	1	0.01
cadata	regression	13760	8	0.1	0.0001	0.1
census-one	1-class	22676	14	0.001	0.0001	∅
a9a-one	1-class	24720	123	0.0001	0.001	∅
ijcnn1-one	1-class	31585	22	0.0001	0.01	∅

TABLE 3.4: Datasets used in the MSMO experiments along with their dimensions and selected SVM, SVR and One-class SVM parameters. While SVM and SVR use a C penalty parameter, One-class uses ν. The ϵ tube width parameter is only used by SVR.

Secondly, the MSMO algorithm was implemented following the pseudocode and recommended enhancements in Section 3.2.2. Thanks to the general SVM formulation used in LIBSVM, this immediately allows its application for SVM for classification, SVR, One-class SVM and LS–SVM.

A publicly available version of this code can be found at `http://arantxa.ii.uam.es/~gaa/software.html`. The new parameter τ has been added as a new LIBSVM command line argument; standard SMO is run when $\tau = 0$. This is the code that was used in all the experiments to follow.

3.2.3.3 "Plain" SMO comparison

Using the MSMO implementation within LIBSVM, a comparison of the improvements obtained by this new algorithm with respect to standard SMO is presented here. As it is of interest to check whether the method applies to large classification datasets as

well, for the following experiments the datasets shown in Table 3.4 were used, which were obtained from the online repositories in [30, 82]. Datasets for regression and one-class classification are included in the mix as well. The latter were built by taking the corresponding classification datasets and keeping only the patterns belonging to the majority class. For those datasets provided as a single file, i.e. with no predefined training/test division, the first $\frac{2}{3}$ of the data were used as training set, the rest as test set. Classification datasets were used as provided, with the exception of *mnist*, which was casted to a two-class problem by seeking to classify odd versus even digits. Target values in regression datasets were normalized to lie in the range $[0, 1]$.

The SVM parameters were selected by a grid search aiming to maximize accuracy in the corresponding test set.[1] Table 3.4 shows the optimal parameters for SVM for classification datasets, SVR for regression datasets, and One-class SVM for one-class datasets. Results for LS–SVM both for classification and regression are presented later on. In every experiment the gaussian kernel $k(x_i, x_j) = e^{-\gamma||x_i - x_j||_2^2}$ was used; it must be noticed that a parameter γ appears in the place of the previously used σ. Both are equivalent as $\gamma = \frac{1}{2\sigma^2}$, nevertheless here γ is preferred since it is the standard one adopted in LIBSVM.

As a stopping criterion for the training algorithms a KKT violation smaller then 10^{-3} in the most violating pair was used, which is the default in LIBSVM.

Results for momentum memories $\tau = 1$ and $\tau = 10$ are presented in Table 3.5. The number of iterations to achieve convergence and running times were measured for both algorithms; test accuracies were exactly the same in all cases. But for the *w8a* and *cadata* datasets, it is clear that MSMO succeeds in constructing more informative updating directions for the classification and regression problems, as the number of iterations to achieve convergence is always reduced. Furthermore, this reduction in iterations is reflected in a reduction in training times, which are only slightly higher than their respective iterations reductions. This fact support the previous claims (Section 3.2.2) about how MSMO's extra computations ($O(N)$) are marginal with respect to the base cost of SMO ($O(\bar{d}N)$). Consequently, any reduction in iterations practically produces direct savings in training times. In contrast to these results, for the one-class problems, MSMO has little impact on performance. This is, however, readily explained by the fact that training the SVM model for these datasets requires a very low number of iterations ($\simeq 30$), regardless of the number of training patterns, and so MSMO has a thin chance to produce any savings.

[1] This practice generally leads to overfitting over the test sets, providing inaccurate estimates of the generalization ability of the model. It must be realized, however, that model accuracy is not of interest in these experiments, since both SMO and MSMO solve exactly the same model, but rather how fast these methods can find the solution under an ideal choice of the SVM parameters.

	$\tau = 1$		$\tau = 10$	
Dataset	Iters. %	Time %	Iters. %	Time %
phw	81.6%	85.6%	75.7%	80.7%
news	94.3%	96.8%	87.1%	90.3%
letter	88.6%	96.8%	83.3%	91.9%
poker	94.8%	95.2%	90.4%	89.9%
census	77.1%	77.9%	69.4%	69.9%
a9a	73.1%	73.8%	47.7%	47.8%
ijcnn1	92.2%	91.6%	89.7%	89.9%
shuttle	49.7%	42.6%	62.6%	60.4%
connect-4	86.4%	82.9%	69.2%	74.6%
w8a	101.4%	103.0%	101.4%	102.1%
mnist2	86.5 %	85.2 %	55.3 %	56.0 %
acoustic	84.9%	85.1%	72.4 %	72.7 %
seismic	79.6 %	80.2 %	75.8 %	75.7 %
combined	86.3 %	87.8 %	77.9 %	77.3 %
Average	84.0 %	84.6 %	75.5 %	77.0 %
mg	63.6 %	63.5 %	53.3 %	61.4 %
space	89.8 %	93.5 %	71.3 %	56.8 %
abalone	31.8 %	33.3 %	35.0 %	36.5 %
cpusmall	75.4 %	77.0 %	68.2 %	69.6 %
cadata	96.7 %	102.2 %	97.7 %	101.1 %
Average	71.4 %	73.9 %	65.1 %	65.08 %
census-one	100 %	101.7 %	100 %	97.1 %
a9a-one	115.7 %	108.8 %	100 %	100.5 %
ijcnn1-one	73.9 %	92.7 %	86.9 %	99.0 %
Average	96.5 %	101.0 %	95.6 %	98.8 %

TABLE 3.5: Percentage of iterations and running time of MSMO w.r.t SMO to achieve convergence, grouped by model (classification, regression, one-class).

Another point worth mentioning is that although a choice of $\tau = 10$ produces larger savings than $\tau = 1$, the fact that a single term building up m already produces a significant speedup is of interest. Recall that each term requires storing its corresponding updating info (δ, λ, w) in memory. If $\tau = 1$, additional memory space for only $N+2$ floats is needed, thus making this choice attractive for those situations where strict memory limitations must be met.

Addressing now the LS–SVM model, Table 3.6 presents a listing of some of the classification and regression datasets from Table 3.4 for which the LS–SVM model was tested. Since this model is different from the SVM classification and SVR formulations, a search for optimal parameters was run again, yielding the values also shown in the table.

Dataset	Task	N	d	C	γ
phw	10-class	7227	16	10	1
letter	26-class	13294	16	10	10
census	2-class	30148	14	10	1
a9a	2-class	32561	123	0.1	0.1
ijcnn1	2-class	35000	22	1	1
w8a	2-class	49749	300	1000	0.1
acoustic	3-class	78823	50	10	1
mg	regression	922	6	100	1
abalone	regression	2784	8	100	1
cadata	regression	13760	8	1	0.0001

TABLE 3.6: Datasets used in the MSMO experiments with LS–SVM model, along with their dimensions and selected parameters.

As well as including results for SMO and MSMO with $\tau = 10$, Table 3.7 shows also training times for the **LS–SVMlab** toolbox [36], which is the standard software for LS–SVM training implemented by the authors of the model. This method finds a solution to the LS–SVM problem just by solving a system of linear equations (details in Section 2.2.3), thus incurring in cost of order $O(N^3)$. This should be compared against a (roughly) $O(N^2)$ cost of SMO [60]. Furthermore, LS–SVMlab requires storing the full matrix Q in memory, which makes it ineffective for large training sets.

By analyzing first the SMO–MSMO pair, similarly to the previous models MSMO manages to produce gains in training times in most of the cases. Comparing now MSMO running times against LS–SVMlab, for the smallest datasets (*mg* and *abalone*) the latter is clearly faster, despite the difference in complexity orders. However, for the larger *phw* dataset MSMO produces faster convergence. This can be explained as follows: even though (M)SMO has lower computational complexity ($O(N^2)$), it requires far more operations than the LS–SVMlab algorithm, which directly solves the problem through the solving of a single linear system ($O(N^3)$). For small datasets solving such system results to be cheaper than running SMO. However, as the size of the problem grows, LS–SVMlab running times grow cubicly, whereas SMO quadratically. This explains why LS–SVMlab works better only for small inputs.

Additionally, it is relevant that LS-SVMlab could not be run for most of the datasets (denoted in Table 3.7 as $\varnothing$), as because of their large size, trying to load the full Q matrix produced a RAM memory overflow. Though strictly speaking the algorithm is still runnable through virtual memory, this approach is impractical, and thus was not considered here.

	SMO		MSMO			LS–SVMlab
Dataset	Iters.	Time	Iters.	Time	% Red.	Time
phw	143462	69.4	106226	54.9	79.10 %	168.51
letter	915659	300.8	699596	242.9	80.74 %	∅
census	88870	1098.2	69406	905.5	82.45 %	∅
a9a	48922	1088.2	48840	1093.4	100.48 %	∅
ijcnn1	49533	751.7	47875	739.3	98.35 %	∅
w8a	601716	10219	380058	7417	72.58 %	∅
acoustic	545908	29762	401490	23142	77.75 %	∅
Average					84.49 %	
mg	8640	1.87	4869	1.04	55.9 %	0.33
abalone	17251	12.74	9927	7.90	62.0 %	4.38
cadata	21863	266.7	20553	263.0	98.6 %	∅
Average					72.16 %	

TABLE 3.7: Iterations and running times of SMO, MSMO and LS–SVMlab for solving the LS–SVM model. Percentage of time required by MSMO w.r.t. SMO is also displayed. A ∅ appears whenever LS–SVMlab could not solve the problem.

3.2.3.4 Effects of caching and shrinking

After showing how "plain" MSMO obtains significant speedups for a variety of models and datasets, now its performance is analyzed when combined with the caching and shrinking strategies. Four scenarios are tested: using kernel caches of 100 MB and 1 GB, and using shrinking or not for each of these cache sizes. Repeating the experiments in the previous section with these settings produces the results summarized in Table 3.8 for SVM classification, SVR and One-class SVM. Again, accuracies obtained for both models were exactly the same.

Several facts must be pointed out from these results. First, and as expected, the reduction in number of iterations does not depend on the cache size, since this strategy just saves time in the evaluations of the kernel function. Shrinking, conversely, does have some influence in the iterations savings, though negligible in the classification datasets. In the regression datasets only *abalone* is seriously affected.

Second, and in spite of the savings in iterations remaining similar, reductions in training times are greatly modified by the inclusion of these strategies. As the cache size grows, reductions are diminished. The use of shrinking also has a great impact, weakening further the effects of MSMO. In spite of this, speed-ups can still be obtained for some of the datasets.

	Full			Shrinking		
Dataset	Iters.	100 MB	1 GB	Iters.	100 MB	1 GB
phw	75.79	108.66	102.98	75.80	**96.17**	97.08
news	87.19	100.69	100.48	87.21	101.97	100.94
letter	83.31	98.79	98.72	83.31	98.58	98.63
poker	90.41	**92.16**	98.64	90.60	100.61	100.80
census	69.40	**95.93**	**92.43**	63.83	101.69	98.78
a9a	47.70	**80.96**	**76.45**	48.25	**92.97**	97.16
ijcnn1	89.76	99.47	99.39	88.93	98.86	102.52
shuttle	62.66	**95.51**	**93.12**	64.38	108.84	108.49
connect-4	69.29	**70.14**	**90.70**	69.28	**87.66**	**92.65**
w8a	101.43	102.20	101.83	101.44	100.75	101.83
mnist2	55.35	**57.37**	**77.24**	55.96	**84.98**	**93.97**
acoustic	72.48	**75.64**	**92.40**	72.37	**94.10**	101.80
seismic	75.85	**87.95**	99.81	77.62	100.40	100.20
combined	77.96	**78.74**	**88.34**	77.30	**87.59**	98.78
Average	75.61	88.87	93.75	75.44	96.79	99.54
mg	53.32	**53.83**	**62.24**	50.28	**77.59**	**75.68**
space	71.33	**35.23**	**92.23**	71.33	99.89	126.50
abalone	35.09	**39.31**	**39.31**	78.34	**91.28**	**91.01**
cpusmall	68.22	**79.21**	**78.22**	72.30	**93.65**	**95.98**
cadata	97.77	105.49	105.59	97.51	104.73	104.74
Average	65.14	62.61	75.51	73.95	93.428	98.782
census-one	100	83.37	101.06	100	113.26	102.18
a9a-one	100	109.90	107.26	100	113.76	119.45
ijcnn1-one	86.95	**96.35**	**95.02**	86.95	104.89	100.10
Average	95.65	96.54	101.11	95.65	110.63	107.243

TABLE 3.8: Percentage of iterations and time required by MSMO w.r.t SMO to achieve convergence, for cache sizes of 100 MB and 1GB, and for full problem optimization (no shrinking) or shrinking. Situations where some reduction in times (at most 97 % of time) is maintained are marked in bold.

The causes of these results can be easily understood by recalling that most of the cost per iteration of SMO comes from the evaluations of the kernel function, producing the aforementioned $O(\bar{d}N)$ algorithm complexity. The caching technique allows reusing kernel evaluations from previous iterations and so the later iterations in the algorithm have a reduced average cost per kernel evaluation, depending on the actual size of the cache. If the cache is large enough it might well happen that at some point of the algorithm the full kernel matrix is in memory, and SMO no longer needs to invoke the kernel function, improving its cost per iteration to $O(N)$. This has two implications:

1. When the cost of SMO drops down to $O(N)$, so does the cost of MSMO. However,

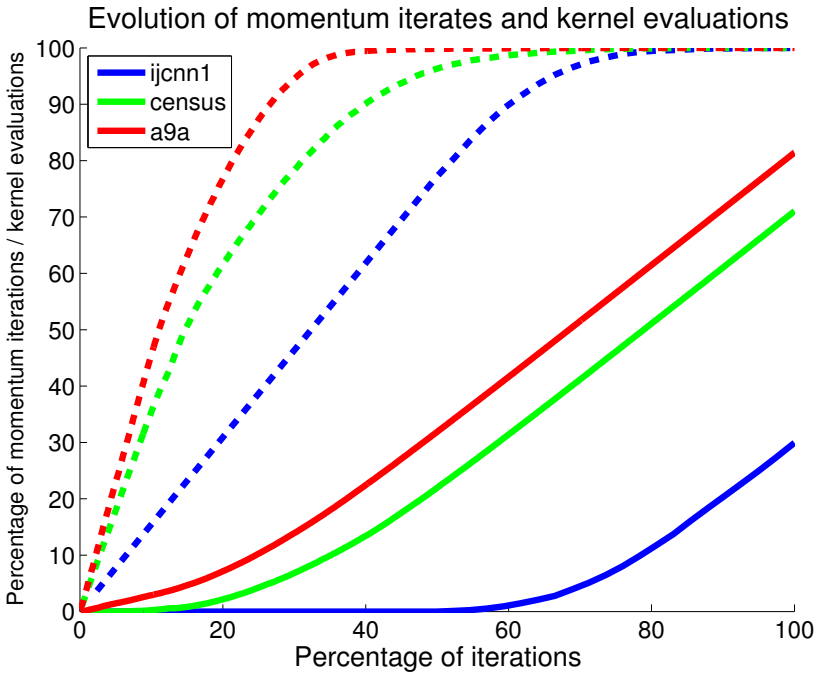


FIGURE 3.7: Evolution of the number of iterations where the momentum update was successfully applied (solid lines) and the number of kernel evaluations, i.e. Q matrix entries not found in cache (dashed lines).

MSMO requires a number of additional operations with also order $O(N)$ cost. While these were negligible for a cost per iteration of $O(\bar{d}N)$, they are no longer so in this situation. Therefore, the relative cost per iteration of MSMO in this setting should be noticeably higher than that of SMO.

2. The iterations with cost $O(\bar{d}N)$ take as much as $O(\bar{d})$ times the time of an $O(N)$ iteration. This observation, as obvious as it might be, makes a great difference in a procedure like MSMO whose aim is to reduce the number of iterations. Clearly, saving $O(\bar{d}N)$ cost iterations is by far more beneficial than saving $O(N)$ iterations.

Furthermore and as pointed out in Theorem II.3 in [83] and in [64], at the end of the algorithm only a subset of the α coefficients has to be updated, and thus only a reduced set of rows from the kernel matrix is needed. This effect is observed in Figure 3.7, where MSMO is run with a cache size of 100 MB, no shriking and $\tau = 10$ for the *ijcnn1*, *census* and *a9a* datasets. As observed, the dashed lines representing the accumulated percentage of kernel operations saturate early on, i.e., there is a point after which no additional kernel evaluations are needed, as they can be fetched directly from the cache. The plot also shows how the number of iterations where the momentum update is successfully applied grows with the total number of iterations. It is clear that momentum updates start to be applied consistently only after a certain number of iterations (this is specially true in the *ijcnn1* dataset). However, it is after these iterates that the number of kernel

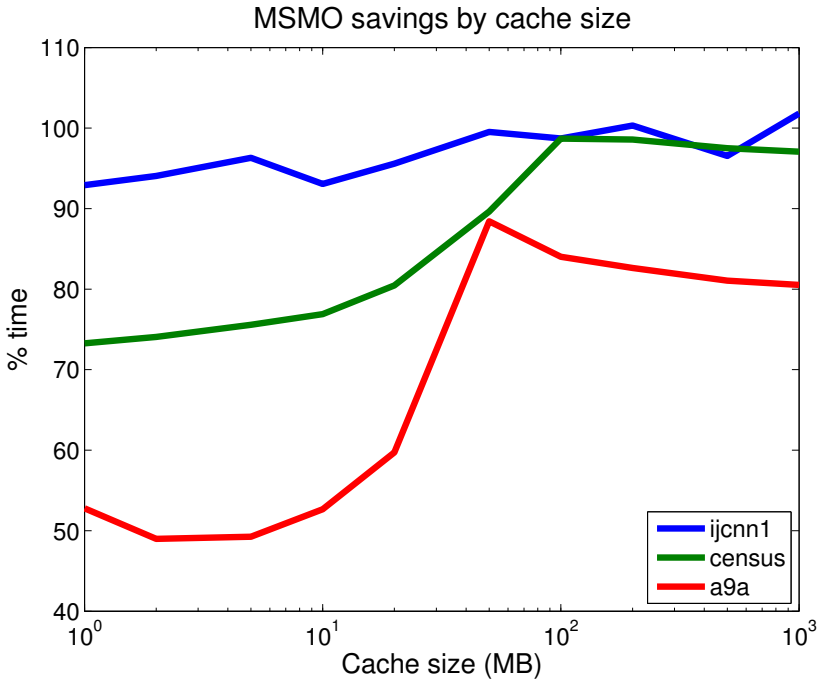


FIGURE 3.8: Percentage of time for convergence required by MSMO in comparison to SMO for increasing kernel cache sizes.

evaluations per iteration drops down due to kernel caching. Therefore, the acceleration induced by MSMO is less effective, as only iterations with cost $\simeq O(N)$ are being saved, while the costly $O(\bar{d}N)$ iterations at the beginning of the algorithm remain unchanged. Joining this with the relative higher costs of MSMO in iterates requiring just $O(N)$ operations explains the observed weakening in MSMO performance for increasing cache sizes.

This explanation is confirmed by taking a look at Figure 3.8, where the percentage of time required by MSMO w.r.t SMO is shown for increasing cache sizes and the same datasets as in Figure 3.7 (no shrinking). Larger cache sizes imply smaller time gains for MSMO, as stated.

The effects of shrinking, on the other hand, are two-fold. First, the effective number of training patterns n (active patterns after shrinkages) is to be much smaller than N, also reducing the cost per iteration. Additionally, when combined with caching, only the submatrix corresponding to the unshrunk elements is cached, thus further minimizing the average cost of kernel evaluations even if the number of training patterns is large or the available cache size small. These two effects combined contribute to reducing the cost per iteration to $O(n)$ in the later iterations of the algorithm, thus further limiting the potential time savings produced by MSMO. It must be noted, however, that improvement is still obtained when applying MSMO in some of the cases (as seen in Table 3.8).

	No cache		100 MB		1 GB		
Dataset	SMO	MSMO	SMO	MSMO	SMO	MSMO	LS–SVMlab
phw	69.4	79.1 %	18.2	95.6 %	18.3	94.5 %	168.5
letter	300.8	80.7 %	71.6	95.0 %	70.9	95.2 %	∅
census	1098.2	82.4 %	1105	82.1 %	976.7	84.9 %	∅
a9a	1088.2	100.4 %	1096.3	100.2 %	1066.3	101.4 %	∅
ijcnn1	751.7	98.3 %	759.9	98.1 %	738.9	98.9 %	∅
w8a	10219	72.5 %	9924.7	67.3 %	8275.9	68.4 %	∅
acoustic	29762	77.75 %	29948	76.6 %	29257	77.4 %	∅
Average		84.45 %		87.84 %		88.67 %	
mg	1.87	55.9 %	0.32	72.4 %	0.34	71.3 %	0.33
abalone	12.7	62.0 %	2.2	78.7 %	2.1	80.3 %	4.3
cadata	266.7	98.6 %	265.7	98.7 %	83.3	111.8 %	∅
Average		72.16 %		83.26 %		87.8 %	

TABLE 3.9: Running times for SMO and reduction factors attained by MSMO when solving the LS–SVM model for different cache sizes. LS–SVMlab running times from Table 3.7 are also included for reference.

For completeness, results for LS–SVM are included as well in Table 3.9 for different cache sizes. Shrinking cannot be applied in this model, as box constraints no longer exist. The decay of MSMO's performance as the cache size grows takes place again, as expected, although this time the lack of an shrinking strategy leaves more room for improvement to MSMO. In the particular case of the large dataset *acoustic* the use of a cache has no impact on running times, however MSMO does. For the rest of datasets caching helps in improving running times, as expected. This fact, though, is still remarkable by the sole reason that other SMO–like implementations of LS–SVM solvers do not use make use of caching. This adds further value to the LS–SVM implementation presented here, as it turns out to be an appealing practical way of training LS–SVM models.

3.2.3.5 Influence of SVM parameters

Caching and shrinking are not the only factors that alter the functioning of MSMO. As it turns out, the parameters selected for the SVM model largely influence MSMO's capability to reduce the number of iterations until convergence, and are in fact the cause of the variability observed when testing the method on different datasets. To assess this effect running times were measured for "plain" SMO and MSMO solving SVM classification for the datasets *phw*, *letter* and *ijcnn1* for a grid of γ and C parameters. The results obtained are shown in Figure 3.9 (other datasets present similar behavior). A general pattern can be observed: MSMO is able to obtain better savings in the

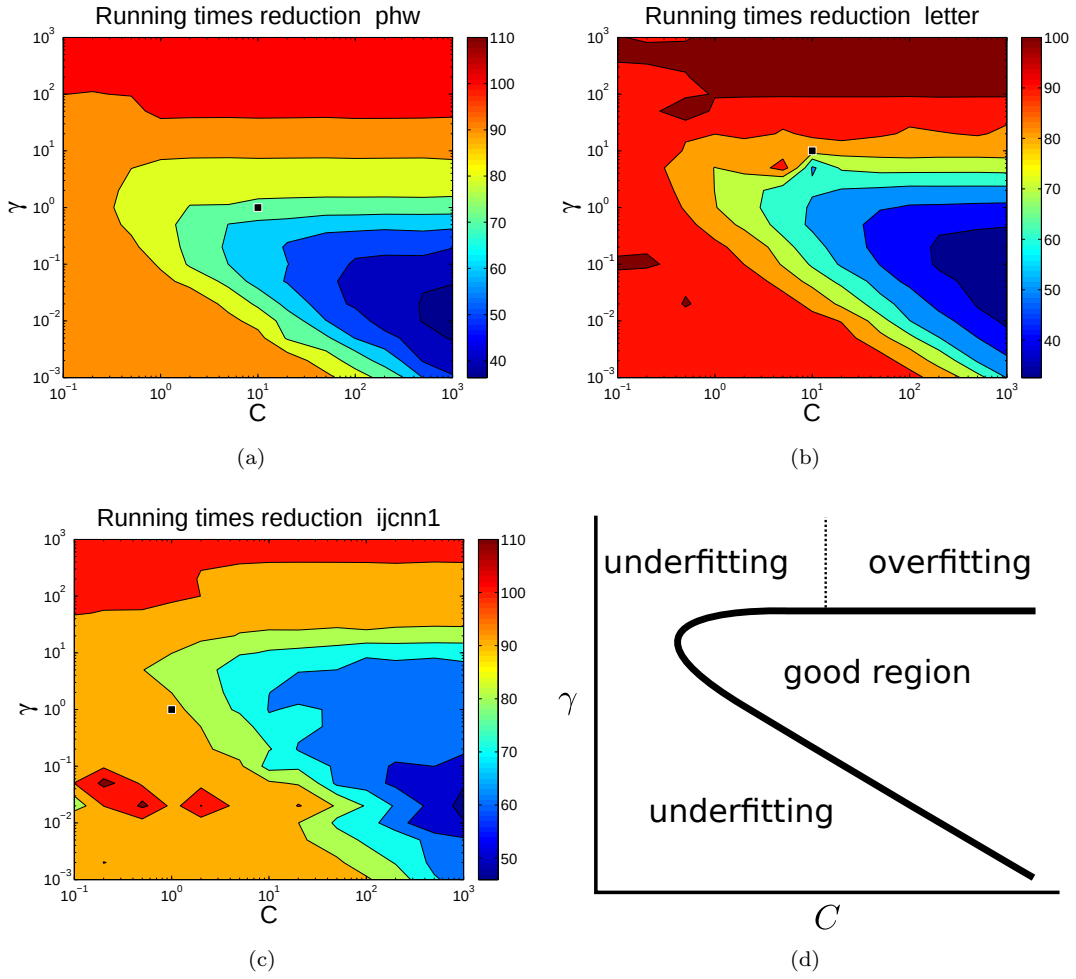


FIGURE 3.9: a-c) Percentage of running times required by MSMO w.r.t SMO for different settings of C and γ parameters. The squared dots represent the corresponding values used in the rest of experiments (Table 3.4). d) Performance regions of the SVM parameters (figure adapted from [84]).

parameter region where C is large and γ has an intermediate value. This behavior can be explained as follows.

First, the influence of the γ parameter can be explained in terms of the structure of the matrix Q of the dual problem. Whatever the particular SVM model being used, Q always depends on K, $K_{i,j} = k(x_i, x_j) = e^{-\gamma||x_i - x_j||_2^2}$. Consequently, whenever $\gamma \to \infty$, Q tends to the identity matrix I, and when $\gamma \to 0$, Q tends to the all ones matrix $\to \mathbf{1}$. In the identity case the problem becomes separable – problem variables are independent – but for the $\alpha \cdot y = \Delta$ constraint, and so it can basically be solved by optimizing for each α_i separately. It is obvious then that a momentum term has no use in this setting, as previous updates have practically no influence in the current update. Similarly, when $Q \to \mathbf{1}$ each α_i has exactly the same influence as all the others over the problem (but for the linear term $\alpha \cdot p$), and so again a strategy trying to combine previous updates to

generate better directions will not be effective.

Second, the influence of parameter C can be addressed as follows. As already stated before, by Theorem 6 in [64] it is known that after a certain number of iterations, a subset of the α_i at a bound $\{L, C\}$ remain there until convergence. This result is the keystone of the shrinking strategy, but also shows that once a coefficient hits a boundary, it is likely that no further improvement in the objective function can be obtained by modifying it, i.e. the problem is already solved for that coefficient. This situation is not likely to arise in unconstrained optimization problems when a first–order optimization method is used, due to the aforementioned zigzagging towards the optimum, and SMO falls into this category. The larger the C, the more the problem resembles an unconstrained problem, and so the performance of SMO will be poorer. Conversely, since MSMO avoids this zigzagging, large values of C have much lesser impact in running times than in SMO, and so the performance differences between both algorithms become larger.

It is also worth mentioning that in [84] a study of the asymptotic behavior of the gaussian kernel parameters is presented, concluding that there exists a "good" region in the $(\log C, \log \gamma)$ space of parameters where the best leave-one-out error is likely to be found. This region is depicted in Figure 3.9(d), and can roughly be defined as a cone originating at a certain (C, γ) pair, one of its borders extending towards $C \to \infty$ with no change in γ and the other one moving diagonally in (C, γ) towards $(\infty, 0)$. This pattern appears also in the results here, as seen in Figure 3.9, as the region where MSMO is able to achieve larger speed-ups. This hints that MSMO is generally able to produce accelerations for those SVM parameters in which the model performs correctly, i.e. those bound to be used in practice.

3.3 Discussion

Research in the field of Support Vector Machine training algorithms has been thorough during the last years, producing a series of efficient solvers and accelerating strategies, such as caching and shrinking, that further improve the performance of such solvers. Still, the results obtained in this chapter show how improvements are still possible, or more specifically, that the state–of–the–art algorithm Sequential Minimal Optimization (SMO) can still be improved for higher efficiency.

Between the two proposed approaches to improve SMO, the Momentum Sequential Minimal Optimization (MSMO) algorithm stands as a clear winner, and when comparing it against standard SMO, significant reductions in number of iterations for convergence

are often achieved. When no other accelerating strategies are used, these savings in iterations almost directly translate into savings in running times. If caching or shrinking strategies are also used, savings in iterations still appear, though the time savings are attenuated. Nevertheless and in almost all of the experiments run, MSMO produces equal or better results than SMO, and therefore its application could be recommended in general.

It is also remarkable that the proposed methods solve a generalized SVM optimization problem, and thus the models of SVM classification, Support Vector Regression and One-class SVM directly benefit from it. Additionally and thanks to such generalized formulation used here, these algorithms apply as well to Least–Squares SVM. Though SMO–like algorithms for LS–SVM had already been developed in the past, the inclusion into this general framework and the application of the presented accelerating techniques is new.

In conclusion, the algorithms presented in this chapter, while based on simple ideas, manage to improve over the state–of–the–art for several SVM models. Detailed analysis of the structure of the problem, review of existing algorithms and careful design of the proposed ones were decisive factors to obtain these results.

Chapter 4

Newton optimization approaches for TV–regularization

"For the person for whom small things do not exist, the great is not great"
José Ortega y Gasset

This chapter constitutes the second major contribution of this thesis, consisting in a series of methods for rapidly solving problems associated with the Total–Variation (TV) regularizer. First, it is shown how such problems can be tackled elegantly through the use of proximal methods . An essential prerequisite to follow this approach is to have an algorithm able to solve the so–called proximity operator induced by the non–smooth part of the model, in this case the TV regularizer. In this line, reliable and efficient solvers to perform such task are developed here, based on Newton optimization ideas. Experimental results show how these methods are able to perform better than the state–of–the–art. Next, it is shown how the Fused–Lasso model, which employs TV regularization, can be solved efficiently making use of a general–purpose proximity algorithm and the implemented proximity operator solvers. Applications of the model for microarray classification are presented.

Generalizing over this, 2–dimensional and multi–dimensional versions of the TV regularization are presented, showing how their respective proximity operators can be solved by using the 1–dimensional (standard) TV regularization proximity operator solvers as building blocks. This and the application of suitable proximal methods, allows to address more complex models making use of such regularizers, with applications to image denoising and deblurring for the 2–dimensional TV case, and video deblurring as an instance of the multi–dimensional case.

4.1 Total–Variation proximity

The Total–Variation regularizer follows the already presented expression (Section 1.3)

$$\mathrm{TV}_p^{1\mathrm{D}}(x) = \left(\sum_{i=1}^{n-1} |x_{i+1} - x_i|^p \right)^{1/p},$$

where it is stressed that the standard (1–dimensional) case is being considered, as an opposite to the more general cases to follow in subsequent sections. Solving optimization problems involving this regularizer,

$$\min_x f(x) + \mathrm{TV}_p^{1\mathrm{D}}(x),$$

is hard because of its non-differentiability, even for well-behaved choices of norm p. For instance, when $p = 2$ the inner absolute value of $\mathrm{TV}_2^{1\mathrm{D}}(x)$ can be removed, but nevertheless its outer exponent $1/2$ produces non–differentiable points. A good approach to do so is by making use of methods employing **proximity operators** [80], which stand as a kind of basic operation in non-smooth optimization, much in the way a gradient step is a basic operation in smooth optimization. Proximity operators are defined as

$$\mathrm{prox}_R(y) = \underset{x}{\mathrm{argmin}}\ \tfrac{1}{2}\|x - y\|_2^2 + R(x),$$

where R is a given function. Informally, the proximity operator seeks to minimize the function R in a neighborhood of the given point y, i.e. x choices far away from y produce large values of the term $\frac{1}{2}\|x - y\|_2^2$, thus making them unacceptable. In the limit case where R is an indicator function $\iota_{\mathcal{C}}$ of a given set $\mathcal{C}$ [1], the proximity operator turns out to be the Euclidean projection operator: find the point $x \in \mathcal{C}$ nearest to y in Euclidean distance. Because of this, proximity operators are sometimes regarded as a generalization of the projection operator.

The usefulness of proximity operators comes from their use as a key step in a large number of **proximal methods** for non-smooth optimization [78, 79, 85–89]. An illustrative example of this is given by the **Forward-backward splitting** (FBS) algorithm [90], also known as FOBOS, which allows to solve the problem

$$\min_x \quad f_1(x) + f_2(x),$$

[1] Indicator functions were defined in Table 1.2: $\iota_{\mathcal{C}}(x) = 0$ if $x \in \mathcal{C}$, $+\infty$ otherwise.

Algorithm 7 Forward-Backward Splitting

Inputs: f_1, f_2, β.
Initialize $\epsilon \in \left(0, \min\left\{1, \frac{1}{\beta}\right\}\right), x_0 \in \mathbb{R}^N$, $n = 0$.
while stopping criterion not met **do**
  Choose gradient stepsize $\gamma_n \in \left[\epsilon, \frac{2}{\beta} - \epsilon\right]$.
  Gradient step: $y_n = x_n - \gamma_n \nabla f_2(x_n)$.
  Choose proximity stepsize: $\lambda_n \in [\epsilon, 1]$.
  Proximity step: $x_{n+1} = x_n + \lambda_n(\text{prox}_{\gamma_n f_1}(y_n) - x_n)$.
end while
Return x^*.

with $f_1(x)$ a lower semicontinuous[2] convex function and $f_2(x)$ a convex differentiable function with a β-Lipschitz continuous gradient, that is, $\forall\ (x, y) \in \mathbb{R}^N \times \mathbb{R}^N, \|\nabla f_2(x) - \nabla f_2(y)\| \leq \beta\|x - y\|$ for some norm $\|\cdot\|$. Therefore, this kind of problem accepts a class of non-differentiable functions as $f_1(x)$. The pseudocode of FBS is shown in Algorithm 7. Basically, it consists in alternating the minimization of the smooth function f_2 by performing gradient descent, and the minimization of the non-smooth function f_1 through proximity steps. The sequence $x_1, x_2, \ldots$ generated by the algorithm has been shown to converge to a solution of the problem [91].

Proximity operators, thus, provide an effective way of addressing non-differentiability in optimization problems; models using TV regularization are nothing but an instance of those, and so can be approached following these ideas. Unfortunately, there is a catch: a way to solve the subproblem posed by the proximity operator must be provided. Even more, solving the proximity operator turns out to be the most expensive step of the FBS algorithm, as well as of quite a few of other algorithms based on proximity. Therefore, in most of the cases the proximity operator constitutes the bottleneck in performance of the overall solver, and so using or designing efficient proximity solvers is crucial.

In what follows, a way of constructing a solver for the proximity operator applied to 1-dimensional TV regularization is detailed, making use of the particular structure of this regularizer to do so efficiently. This constitutes the basis on which a proximal method can be used to optimize models making use of this regularizer.

4.1.1 Structure of the TV regularizer

The problem posed by the proximity operator of 1-dimensional TV takes the form

[2] Lower semicontinuous functions where already introduced in Section 2.1.1

$$\begin{aligned}\operatorname{prox}_{\mathrm{TV}_p^{1\mathrm{D}}}(y) &= \operatorname*{argmin}_x \tfrac{1}{2}\|x-y\|_2^2 + \lambda \mathrm{TV}_p^{1\mathrm{D}}(x), \\ &= \operatorname*{argmin}_x \tfrac{1}{2}\|x-y\|_2^2 + \lambda \left(\sum_{i=1}^{n-1} |x_{i+1}-x_i|^p\right)^{1/p},\end{aligned}$$

where the regularization parameter λ has also been included as part of the regularizer. The hardness of this problem comes not only from its non-differentiability, but also because of its **non-separability**, i.e. the problem cannot be decomposed into N sub-problems, each involving a single x_i variable. This is relevant because, for instance, the proximity operator of the L_1 norm

$$\operatorname{prox}_{L_1}(y) = \operatorname*{argmin}_x \tfrac{1}{2}\|x-y\|_2^2 + \lambda\|x\|_1$$

is also non-smooth, but its separability makes it solvable in closed form through the **soft-thresholding operator** [92]

$$\operatorname{prox}_{L_1}(y) = \operatorname{sgn}(y) \odot \max\left\{|y| - \lambda, 0\right\},$$

where $\odot$ denotes componentwise product. Non-separability of the TV-norm prevents obtaining such a direct solution for its proximity operator. Nevertheless, the fact is that the TV-norm only presents "weak" non-separability, in the sense that each x_i variable appears coupled with just its preceding x_{i-1} and following x_{i+1} variables. This is especially obvious upon rewriting the proximity problem making use of a differencing matrix

$$D = \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & & \ddots & \\ & & & & -1 \quad 1 \end{pmatrix} \in \mathbb{R}^{(N-1)\times N},$$

that is, $D_{i,j} = 0$, except for $D_{i,i} = -1$ and $D_{i,i+1} = 1$. Using this, the proximity problem results in

$$\operatorname{prox}_{\mathrm{TV}_p^{1\mathrm{D}}}(y) = \operatorname*{argmin}_x \tfrac{1}{2}\|x-y\|_2^2 + \lambda\|Dx\|_p.$$

The matrix D sets the separability of the problem. On the one hand for a diagonal D the problem is separable; on the other hand a full D matrix results in a problem where every variable is linked to each other. Clearly, the differencing matrix D for TV stands at a point quite close to the separable case, and thus some advantage should come from this "weak" non-separability.

This hypothetical advantage becomes real when considering the dual formulation of the TV proximity problem (Section 2.1.1 already provided an introduction to duality). Such dual formulation has already been used in the past (for instance, in [93]), and actually the dual problem can be obtained directly by applying Theorem 31.2 in [22]. Nevertheless, in what follows a new, clearer approach to arrive at the dual is introduced. To to do so, first a more general primal problem is considered, namely

$$\inf_x \quad f(x) + \lambda r(Bx),$$

for f, r convex (not necessarily smooth) functions [3]. This can be casted into a constrained problem by introducing an auxiliary variable $z = Bx$ so that

$$\begin{aligned} \inf_{x,z} \quad & f(x) + \lambda r(z), \\ \text{s.t.} \quad & z = Bx. \end{aligned}$$

The Lagrangian of this problem turns out to be

$$L(x, z, u) = f(x) + \lambda r(z) + u^T(Bx - z),$$

which produces the saddle-point problem

$$\inf_{x,z} \quad \sup_u \quad L(x, z, u).$$

Through convexity arguments the infimum and supremum operators can be swapped, and so the dual problem takes the form

[3] The discussion is made using the infimum instead of the minimum operator, as done in Section 2.1.1. This is more general than the minimum operator, in the sense that functions defined in an open set could be considered as well. Although no such functions are used in what follows, this approach is favored here, since the Fenchel–conjugate works by definition with the infimum / supremum operators.

$$\begin{array}{rll} & \sup_u & \inf_{x,z} L(x,z,u) \\ = & \sup_u & \inf_{x,z} f(x) + \lambda r(z) + u^T(Bx - z) \\ = & \sup_u & \inf_x \left\{ f(x) + u^T Bx \right\} + \inf_z \left\{ \lambda r(z) - u \cdot z \right\} \\ = & \sup_u & -\sup_x \left\{ -f(x) - u^T Bx \right\} - \sup_z \left\{ u \cdot z - \lambda r(z) \right\}. \end{array}$$

The inner maximization problems are instances of the **Fenchel-conjugate** [22]. Given a convex function $f(x)$, its Fenchel-conjugate is defined as

$$f^*(u) = \sup_x \left\{ u \cdot x - f(x) \right\}.$$

Using this definition the dual problem becomes

$$\sup_u \quad -f^*(-B^T u) - \lambda r^*(\lambda^{-1} u).$$

Returning to the $\mathrm{TV}_p^{1\mathrm{D}}$ proximity problem, $\mathrm{prox}_{\mathrm{TV}_p^{1\mathrm{D}}}(y) = \mathrm{argmin}_x \frac{1}{2}\|x-y\|_2^2 + \lambda \|Dx\|_p$, the functions f and r can be identified as $f(x) = \frac{1}{2}\|x-y\|_2^2$, $r(x) = \|x\|_p$, and $B = D$. The Fenchel-conjugate $f^*(z)$ is easily seen to be $f^*(z) = \frac{1}{2} z \cdot z + z \cdot y$, while the conjugate of $r(x)$, that is, of the Lp norm function, is known [22] to be $r(z)^* = \iota_{\|z\|_q \leq 1}$, that is, the indicator function of the dual Lq norm with $\frac{1}{p} + \frac{1}{q} = 1$. This results in the dual problem

$$\max_u \quad -\tfrac{1}{2}\|D^T u\|_2^2 + u^T D y - \lambda \iota_{\|u\|_q \leq \lambda},$$

in which the indicator function can rewritten as a constraint as

$$\begin{array}{ll} \min\limits_u & \frac{1}{2}\|D^T u\|_2^2 - u^T D y, \\ \text{s.t.} & \|u\|_q \leq \lambda. \end{array}$$

Completing the square in the objective function produces

$$\begin{array}{ll} \min\limits_u & \frac{1}{2}\|D^T u - y\|_2^2, \\ \text{s.t.} & \|u\|_q \leq \lambda. \end{array}$$

It is remarkable how such a simple problem comes out from the seemingly hard primal $\mathrm{TV}_p^{1\mathrm{D}}$ proximity problem. In the subsequent sections methods to solve this problem for primal norms $p = 1, 2$ are introduced, which make explicit use of the particular structure of the differencing matrix D to produce efficient solvers. Before that, though, it is relevant to notice that the primal solution x^* can be recovered from the dual u^* by making use of the Karush–Kuhn-Tucker optimality conditions (Theorem 2.4). More specifically, the stationarity condition requires that $\nabla_x L(x, z, u) = 0$, i.e.

$$\nabla_x \left(\tfrac{1}{2}\|x - y\|_2^2 + \lambda \|z\|_p + u^T (Dx - z) \right) = 0$$

which results in the relationship

$$x^* = y - D^T u^*.$$

Furthermore and making use of this relationship, the dual gap can be obtained as

$$\begin{aligned} \mathrm{gap}(x, u) &= \tfrac{1}{2}\|x - y\|_2^2 + \lambda \|Dx\|_p - (-\tfrac{1}{2}\|D^T u\|_2^2 + u^T D y), \\ &= \lambda \|Dx\|_p - u^T D x, \end{aligned} \tag{4.1}$$

which is used as a stopping criterion in the algorithms to follow.

4.1.2 TV-$\mathbf{L}_1$ proximity

Focusing now on the TV proximity operator with $p = 1$ norm, the corresponding dual norm turns out to be $q = \infty$, and so the dual problem becomes

$$\begin{aligned} \min_u \quad & \tfrac{1}{2}\|D^T u - y\|_2^2, \\ \text{s.t.} \quad & -\lambda \leq u \leq \lambda, \end{aligned}$$

as the infinity-norm imposes nothing but box-constraints on the problem. Thanks to this simplicity in the constraints and since the objective function is quadratic convex, the problem can be solved by standard methods such as **TRON** [94], **L-BFGS-B** [95], or **Projected-Newton** (PN) [96]. However, as good solvers as they might be, these methods do not make explicit use of the structure of the problem at hand when invoked

out-of-the-box, and so careful adaption is needed to obtain a truly efficient solver. The approach proposed here is to design an adapted PN method taking into account these facts.

The generic PN procedure runs iteratively: it first identifies the subset of **active variables**[4] and uses these to compute a reduced Hessian. Then, this Hessian is used to correct the gradient and move in the direction opposite to it, scaling by a stepsize, if needed. Finally, the next iterate is obtained by projecting onto the constraints, and the cycle repeats. At the beginning of each iteration, the subset of active variables I is identified as follows:

$$I = \{i|\ (u_i = -\lambda \text{ and } [\nabla f(u)]_i > \epsilon)\ \text{ or } (u_i = \lambda \text{ and } [\nabla f(u)]_i < -\epsilon)\},$$

which are those variables that are at a bound and the gradient indicates that improvement can be obtained by moving further beyond the bound. To avoid numerical issues, the gradient condition is checked using a small threshold ϵ (for instance, 10^{-10}). These variables should not be used in the current update, as they will produce infeasible updating directions. Consequently, the complementary set $\bar{I} = \{1 \dots N\} \setminus I$ constitutes the set of variables to update in the current iteration. From the full Hessian $H = \nabla^2 f(u)$ the **reduced Hessian** $H_{\bar{I}}$ is extracted by selecting rows and columns indexed by $\bar{I}$; this defines the "reduced" update

$$u_{\bar{I}} \leftarrow P\left[u_{\bar{I}} - \alpha H_{\bar{I}}^{-1}[\nabla f(u)]_{\bar{I}}\right],$$

where α is a stepsize, and $P[z]$ denotes componentwise projection of z onto the constraints. So, in essence, the PN algorithm performs Newton steps taking into account the box-constraints of the problem. Being a second–order method, strong theoretical guarantees exist on its convergence and performance [96]. Nevertheless and as stated before, careful design is needed to adequately exploit the problem's structure in order to run the above steps efficiently. In what follows it is shown how to do so for the problem at hand.

First, it should be noticed that the Hessian $H = DD^T$ is symmetric and tridiagonal, with 2s on the main diagonal and -1s on the sub- and superdiagonals. Next, no matter what the active set I contains, the corresponding reduced Hessian $H_{\bar{I}}$ remains symmetric tridiagonal. This is crucial because then the updating direction $d_{\bar{I}} = H_{\bar{I}}^{-1}[\nabla f(u^t)]_{\bar{I}}$ is computed by solving the linear system

[4] The name active variables comes from the fact that they correspond to active constraints, i.e. those met with equality.

$$H_{\bar{I}} d_{\bar{I}} = [\nabla f(u^t)]_{\bar{I}}.$$

This can be done very efficiently by computing the Cholesky decomposition $H_{\bar{I}} = R^T R$ [97], and then solving instead the systems

$$R^T v = [\nabla f(u^t)]_{\bar{I}},$$

and then

$$R d_{\bar{I}} = v.$$

Since H is tridiagonal, R turns out to be a bidiagonal matrix. This has two major advantages: first, the Cholesky decomposition can be performed in linear ($O(N)$) time, and second, the linear systems involving R or R^T can also be solved in linear time through forward / back-substitution. Additionally, high-quality routines to perform such tasks are available in the LAPACK [98] linear algebra libraries, which guarantee top-notch performance. Thus, obtaining the updating direction can be done in $O(N)$ time.

The next crucial ingredient is stepsize selection. The original Projected Newton method [96] recommends Armijo-search along the projection arc. However, for this problem Armijo-search is expensive, since a large number of function evaluations might be required. Consequently, a backtracking strategy using quadratic interpolation [99] is used instead. This strategy is as follows: if the current stepsize α_k does not provide enough decrease in f, a quadratic model is built using $f(u)$, $f(u - \alpha_k d)$, and $\partial_\alpha f(u - \alpha_k d)$. Then, the stepsize α_{k+1} is set to the value that minimizes this model. If the new α_{k+1} is larger than or too similar to α_k, its value is halved. Attention must be paid to the fact that the gradient $\nabla f(u)$ might be misleading if u has components at the boundary and d points outside this boundary (because then, due to the subsequent projection no real improvement would be obtained by stepping outside the feasible region). To address this concern, the computation of the gradient $\nabla f(u)$ is modified, replacing by zeros the entries that relate to direction components pointing outside the feasible region. This procedure, though involved, can also be run in linear time.

All the above ideas are encapsulated as Algorithm 8. Special consideration is given to the particular case in which λ is so large that the unconstrained optimum coincides with the constrained one. In this case, the optimum is obtained by just solving the system $\hat{u}$ via $DD^T\hat{u} = Dy$. This case is checked at the very beginning of the algorithm, and only

Algorithm 8 PN algorithm for TV-L_1-proximity

Inputs: y, λ.
Solve $DD^T\hat{u} = Dy$.
if $\|\hat{u}\|_\infty \leq \lambda$ **return** $x^* = y - D^T\hat{u}$; **end if**
$u^0 = P[\hat{u}]$, $t = 0$.
while dual gap (Equation 4.1) > tolerance **do**
  Identify set of active variables I; let $\bar{I} = \{1 \dots N\} \setminus I$.
  Construct reduced Hessian $H_{\bar{I}}$.
  Solve $H_{\bar{I}} d_{\bar{I}} = [\nabla f(u^t)]_{\bar{I}}$.
  Compute stepsize α using backtracking + interpolation.
  Update $u^{t+1}_{\bar{I}} = P[u^t_{\bar{I}} - \alpha d_{\bar{I}}]$.
  $t \leftarrow t + 1$.
end while
Recover primal solution $x^* = y - D^T u^t$.
return x^*.

if the unconstrained optimum is out of bounds, i.e. $\|\hat{u}\|_\infty > \lambda$, the rest of the algorithm is run. By the arguments above, every step can be carried out in linear time, and so the method has computational cost of $O(N)$ per iteration. Even more, experimentally a sufficiently good solution is usually found in less than 10 iterations, and so in practice the algorithm scales linearly with the size of the inputs (see Section 4.1.4).

Regarding the novelty of the proposed approach, it must be noted that the idea of applying a Newton–like method to TV regularization is not completely new. In the work by Vogel and Oman [100, 101] an approximate version of the $\mathrm{TV}_1^{1\mathrm{D}}$ regularizer is proposed, satisfying smoothness properties. This allows to easily apply Newton approaches to the primal problem, also making use of problem structure to produce an $O(N)$ cost per iteration solver. However, by removing the points of non-differentiability from the problem, much of the structure–inducing properties of the TV regularizer are lost. In particular, sparsity in the gradient is no longer present. Because of this, solving the exact TV regularizer instead of such approximation is more interesting; however, the non-smoothness of the primal problem makes the application of a Newton–like method infeasible. In spite of this, it has been shown here that through dualization and the use of a Projected Newton method this is, in fact, possible.

4.1.3 TV-L_2 proximity

For the primal norm $p = 2$, the dual norm is also $q = 2$, and so the dual problem becomes

$$\begin{aligned} \min_u \quad & \tfrac{1}{2}\|D^T u - y\|_2^2, \\ \text{s.t.} \quad & \|u\|_2 \leq \lambda. \end{aligned}$$

This problem results to be an instance of the well-known trust-region subproblem, for which a variety of numerical methods are available [102]. Below an algorithm based on the **Moré-Sorensen Newton** (MSN) iteration [13] is derived, which in general would be very expensive but in this case proves to be efficient owing again to the tridiagonal structure of the Hessian. In addition to this, an alternative, much simpler method based on **Gradient Projection** ideas is proposed as well.

First, the KKT conditions for the dual problem must be considered. They can be derived from the Lagrangian of such dual problem, which takes the form

$$L(u, \alpha) = \tfrac{1}{2}\|D^T u - y\|_2^2 + \alpha(\|u\|_2^2 - \lambda^2),$$

for $\alpha \geq 0$ a Lagrange multiplier. Since both sides of the constraint are always positive, they can be squared without modifying the problem. This helps to ease some calculations in what follows. Following a similar derivation to the one shown previously in Section 2.1.1, the KKT conditions result to be

Stationarity	**Dual feasibility**
• $(DD^T + \alpha I)u = Dy$.	• $\|u\|_2 \leq \lambda$.
Primal feasibility	**Complementary slackness**
• $\alpha \geq 0$,	• $\alpha(\|u\|_2 - \lambda) = 0$.

There are two cases: $\|u\|_2 < \lambda$ or $\|u\|_2 = \lambda$. If $\|u\|_2 < \lambda$, then $\alpha = 0$ and u is obtained by solving $DD^T u = Dy$, which is the unconstrained optimum of the problem. The inverse is also true: if the solution to $DD^T u = Dy$ lies in the interior of the L_2 ball, then it solves the problem. This can be checked at the beginning of the algorithm as it is done in the method proposed for TV-L_1 proximity, and so only the case $\|u\|_2 = \lambda$ must be considered from then on.

For a given α, u is determined as a function of α, as $u(\alpha) = (DD^T + \alpha I)^{-1} Dy$, and so the optimal value of α must be found to obtain the solution u^*. This can be done by solving $\|u(\alpha)\|_2^2 = \lambda^2$, or alternatively solving the MSN equation

$$h(\alpha) = \lambda^{-1} - \|u(\alpha)\|_2^{-1} = 0,$$

which is written in this way because it results to be almost linear in the search interval, producing fast convergence [13]. Solving this non-linear equation system for α can be done by applying the classic Newton's method for root-finding, whose iterations take the form

$$\alpha \leftarrow \alpha - \frac{h(\alpha)}{h'(\alpha)},$$

and some calculation shows that

$$\frac{1}{h'(\alpha)} = -\frac{\|u(\alpha)\|_2^3}{u(\alpha)^T(DD^T + \alpha I)^{-1}u(\alpha)}.$$

The key idea in MSN is to eliminate the matrix inverse here by making use again of the Cholesky decomposition $DD^T + \alpha I = R^T R$ and defining a vector $q = (R^T)^{-1}u$, so that $\|q\|_2^2 = u(\alpha)^T(DD^T + \alpha I)^{-1}u(\alpha)$. As a result, the Newton iteration becomes

$$\begin{aligned}
\alpha - \frac{h(\alpha)}{h'(\alpha)} &= \alpha - (\|u(\alpha)\|_2^{-1} - \lambda^{-1}) \cdot \frac{\|u(\alpha)\|_2^3}{u(\alpha)^T(DD^T + \alpha I)^{-1}u(\alpha)}, \\
&= \alpha - \frac{\|u(\alpha)\|_2^2 - \lambda^{-1}\|u(\alpha)\|_2^3}{\|q\|_2^2}, \\
&= \alpha - \frac{\|u(\alpha)\|_2^2}{\|q\|_2^2}\left(1 - \frac{\|u(\alpha)\|_2}{\lambda}\right),
\end{aligned}$$

and therefore

$$\alpha \leftarrow \alpha - \frac{\|u(\alpha)\|_2^2}{\|q\|_2^2}\left(1 - \frac{\|u(\alpha)\|_2}{\lambda}\right).$$

As in TV-L_1, the tridiagonal structure of $(DD^T + \alpha I)$ allows to compute both R and q in linear time, so the overall iteration runs in $O(N)$ time.

The above ideas are presented as pseudocode in Algorithm 9. As a stopping criterion two conditions are checked: whether the dual gap is small enough, and whether u is close enough to the boundary. This latter check is useful because intermediate solutions could be dual-infeasible, thus making the dual gap an inadequate optimality measure on its own.

Even though the MSN method can be run at linear time per iteration, it is fairly sophisticated, and in fact a much simpler method can be devised. This is illustrated here by a **Gradient Projection** method with a fixed stepsize α_0, whose updating rule is simply defined as

Algorithm 9 MSN algorithm for TV-L_2 proximity

Inputs: y, λ.
Solve $DD^T\hat{u} = Dy$.
if $\|\hat{u}\|_2 \leq \lambda$ **return** $x^* = y - D^T\hat{u}$; **end if**
Initialize $\alpha^0 = 0$, $t = 0$.
while ($\neg$ converged) **do**
  Compute Cholesky decomp. $DD^T + \alpha^t I = R^T R$.
  Obtain u by solving $R^T R u = Dy$.
  Obtain q by solving $R^T q = u$.
  Update α: $\alpha^t = \alpha^{t-1} - \frac{\|u\|_2^2}{\|q\|_2^2}\left(1 - \frac{\|u\|_2}{\lambda}\right)$.
  $t \leftarrow t + 1$.
end while
Recover primal solution $x^* = y - D^T u^t$.
return x^*.

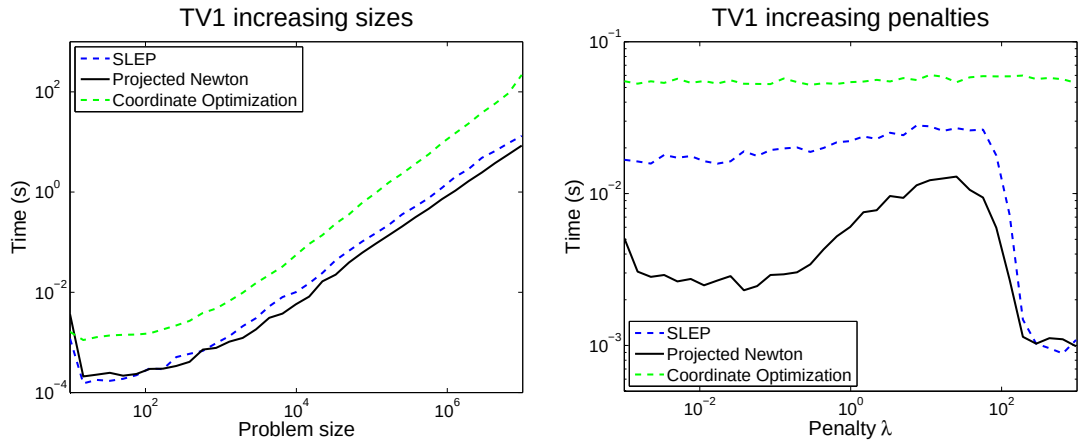
Algorithm 10 GP algorithm for TV-L_2 proximity

Inputs: y, λ.
Initialize $u^0 \in \mathbb{R}^N$, $t = 0$.
while ($\neg$ converged) **do**
  Gradient update: $v^t = u^t - \frac{1}{4}\nabla f(u^t)$.
  Projection: $u^{t+1} = \max(1 - \lambda/\|v^t\|_2, 0) \cdot v^t$.
  $t \leftarrow t + 1$.
end while
Recover primal solution $x^* = y - D^T u^t$.
return x^*.

$$u^{t+1} = P_{\|\cdot\|_2 \leq \lambda}\left[u^t - \alpha_0 \nabla f(u^t)\right],$$

for a given function f to optimize. A good choice for the stepsize α_0 is given by the inverse of the Lipschitz constant L of the gradient $\nabla f(u)$, as already shown in other methods [78, 90, 103]. Since in the case at hand $f(u)$ is a quadratic function, L can be shown to be the largest eigenvalue of the Hessian DD^T. Thanks to the structure of D, the eigenvalues have a closed-form expression, namely $\lambda_i = 2 - 2\cos\left(\frac{i\pi}{N}\right)$ (for $0 \leq i \leq N$) [104]. The largest is $\lambda_{N-1} = 2 - 2\cos\left(\frac{(N-1)\pi}{N}\right)$, which tends to 4 as $N \to \infty$; so $\alpha_0 = 1/4$ is a good approximation and is the stepsize used here in practice.

A pseudocode for this alternative GP approach is presented as Algorithm 10. It is clearly simpler than the MSN algorithm, and thus requires even less cost per iteration. Still, GP is just a first–order method as opposed to MSN's second–order iterates. This implies that MSN has stronger guarantees of convergence, though depending on the situation GP can outperform it. In practice a hybrid approach joining both of them is recommended. Details on this are given later in Section 4.1.4.



(a) (b)

FIGURE 4.1: Running times (in secs) for PN, SLEP and Coordinate Optimization solvers for $\text{TV}_1^{1\text{D}}$-proximity with increasing a) input sizes, b) penalties. Both axes are plotted in logarithmic scale.

4.1.4 Experiments

Experiments on the performance of the introduced proximity solvers for the $\text{TV}_1^{1\text{D}}$ and $\text{TV}_2^{1\text{D}}$ regularizers are presented here. Comparisons with state–of–the–art algorithms in this field are made, showing the benefits of the proposed methods.

The proximity solvers were implemented in the C programming language, with calls to the LAPACK library [98] (made in Fortran) for the required matrix decomposition and linear system solving steps of the algorithms. A MATLAB toolbox implementing some of the methods here is available at `http://arantxa.ii.uam.es/~gaa/software.html`.

The solvers were tested under two scenarios

I) Increasing input size N ranging from 10^1 to 10^7. A random penalty $\lambda \in [0, 50]$ is selected for each run, and the data vector y is also generated randomly (following a uniform distribution) by picking $y_i \in [-2\lambda, 2\lambda]$ (proportionally scaled to λ) $\forall\ i$.

II) Varying penalty parameter λ ranging from 10^{-3} (negligible regularization) to 10^3 (the TV term dominates). N is set to a medium value (1000) and y_i is randomly generated in the range $[-2, 2]$ (again, uniformly).

Starting with the Projected Newton (PN) proximity solver of $\text{TV}_1^{1\text{D}}$ (Section 4.1.3), the method is compared against the **FLSA** function (C implementation) of the SLEP library by Liu et al. [105], which seems to be the state–of–the–art method for $\text{TV}_1^{1\text{D}}$-proximity [103]; and the **Pathwise Coordinate Optimization** method (R + Fortran implementation) by Friedman et al. [106]. For PN and SLEP, a duality gap of 10^{-5} is used as

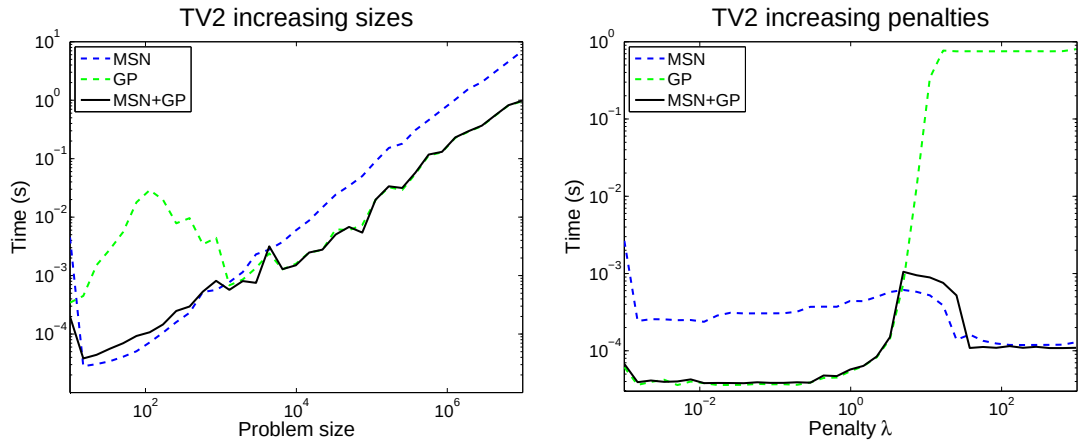
$\log_{10} N$	Times SLEP	Times PN	Times Coord.
1.53	**0.17**	0.2476	1.37
2.06	0.30	**0.29**	1.52
2.58	0.59	**0.41**	2.69
3.11	1.33	**1.04**	6.74
3.64	5.25	**3.10**	22.20
4.17	15.10	**8.22**	92.41
4.70	67.60	**39.35**	359.50
5.23	221.58	**137.81**	1550.27
5.75	759.62	**464.32**	5678.25
6.28	2874.83	**1655.25**	23285.00
6.81	9457.11	**5659.42**	93366.00

Table 4.1: Running times (in milliseconds) for PN, SLEP and Coordinate Optimization solvers for $\mathrm{TV}_1^{1\mathrm{D}}$ problems with increasing input sizes (in log-scale); N denotes problem size.

the stopping criterion. Duality gap is not supported by Coordinate Optimization, and so its default stopping criteria is employed. Timing results are presented in Figure 4.1 for both experimental scenarios. From the plots it is clear that both SLEP and PN are much faster than Coordinate Optimization. It must be mentioned, though, that the latter returns the full regularization path (solutions for every choice of λ), while SLEP and PN compute the solution for only one λ. But this is no limitation; SLEP and PN run much faster and with warm-starts solutions for several λ values could be computed rapidly, if needed.

With increasing input sizes PN finds a solution faster than SLEP, taking roughly at most 60% of the time: explicit numerical values are reported Table 4.1 for easy reference. Figure 4.1(b) indicates that larger speedups are observed for small λ, while for large λ, both SLEP and PN perform similarly. The rationale behind this behavior is simple: for smaller λ the active set I (variables at bound) is prone to become larger, and PN explicitly takes advantage of this set by updating only the variables not indexed by I. On the other hand, for large λ, PN's strategy becomes similar to that of SLEP, hence the similar performance. Finally, as Coordinate Optimization computes the full regularization path, its runtime is invariant with λ.

Moving now to the $\mathrm{TV}_2^{1\mathrm{D}}$ proximity problem, the proposed More–Sorensen Newton (MSN) and Gradient Projection (GP) methods are analyzed. The stopping criterion is a duality gap of 10^{-5}, with the addition of an extra boundary proximity criterion for MSN with tolerance 10^{-6} to avoid early stopping due to non-feasible solutions generation. Figure 4.2 shows results for the two experimental scenarios under test.



(a) (b)

FIGURE 4.2: Running times (in secs) for MSN, GP and a hybrid MSN+GP approach for TV_2^{1D}-proximity with increasing a) input sizes, b) penalties. Both axes are plotted in logarithmic scale.

Looking at the results it can be seen that the performance of MSN and GP differs noticeably in the two experimental scenarios. While the results for the first scenario (Figure 4.2(a)) might indicate that GP converges faster than MSN for large inputs, it actually does so depending on the size of λ relative to $\|y\|_2$. Indeed, the second scenario (Figure 4.2(b)) shows that although for small values of λ, GP runs faster than MSN, as λ increases, GP's performance worsens dramatically, so much that for moderately large λ it is unable to find an acceptable solution even after 10000 iterations (an upper limit imposed in our implementation). Conversely, MSN finds a solution satisfying the stopping criterion under every situation, thus showing a more robust behavior.

Therefore, it seems sensible to propose a hybrid approach combining the strengths of MSN and GP. This hybrid method is guided using the following (empirically determined) rule of thumb: if $\lambda < \|y\|_2$ use GP, otherwise use MSN. Further, as a safeguard, if GP is invoked but fails to find a solution within 50 iterations, the hybrid should switch to MSN. This combination guarantees rapid convergence in practice. Results for this hybrid approach are also included in the plots in Figure 4.2, and show how it successfully mimics the behavior of the better algorithm amongst MSN and GP.

4.2 Fused–Lasso

After developing efficient solvers for the TV proximity operator, this section shows how they can be used to solve the optimization problem posed by the **Fused-Lasso** (FL) model, introduced by Tibshirani et al. [17] . This model takes the form

$$\min_x \tfrac{1}{2}\|Ax - y\|_2^2 + \lambda_1\|x\|_1 + \lambda_2 \mathrm{TV}_1^{1\mathrm{D}}(x),$$

that is, a Least–Squares loss $L(x) = \frac{1}{2}\|Ax - y\|_2^2$ plus two regularizers: an L_1 and a $\mathrm{TV}_1^{1\mathrm{D}}$ norm ($r(x) = \lambda_1\|x\|_1 + \lambda_2\mathrm{TV}_1^{1\mathrm{D}}(x)$). The combination of these two regularizers induces a sparse structure where most of the x_i tend to be 0, while those $x_i \neq 0$ tend to appear in blocks of equal values $x_{i-1} = x_i = x_{i+1} = \ldots$. This model has been successfully applied in several bioinformatics applications [106–108], as it encodes prior knowledge about consecutive elements in microarrays becoming active at once.

While this problem could be solved using the FBS algorithm reviewed in Section 4.1, instead the general **Trust-Region Proximal** (Trip) framework of Kim et al. [89] is applied. Similarly to the FBS method, Trip works by repeated calls to the proximity operator of the non-smooth function; the basic step in Trip is

$$x^{k+1} \leftarrow \mathrm{prox}_{r(\cdot)/\alpha^k}\left(x^k - \frac{1}{\alpha^k}\nabla L(x^k)\right).$$

The key aspect that distinguishes this step from other proximal methods is the stepsize α^k, which is selected using the famous spectral formulæ of Barzilai–Borwein [109], and has been shown to have tremendous impact on empirical performance [89, 109]. A known problem of this stepsize, though, is its non-monotonicity: it does not ensure that improvements in the objective function are obtained at every iteration, which in turn might produce non-convergent behaviors in some settings. To circumvent this potential problem, if after a fixed number of iterations Trip fails to produce an improvement, it enforces a monotonic step that guarantees such improvement, though at a higher cost than a standard step. Still, in practice the monotonic step is seldom required, and thus Trip features a low amortized cost per iteration.

In order to apply Trip to the Fused–Lasso problem a way to compute the proximity operator for the scaled regularizer $r(x) = \lambda_1\|x\|_1 + \lambda_2\mathrm{TV}_1^{1\mathrm{D}}(x)$ is needed. While proximity operators do not decompose in general, for this case it can be shown [17, 103, 110] that $\mathrm{prox}_{r(\cdot)}(y) = \mathrm{prox}_{\lambda_1\|\cdot\|_1}\left(\mathrm{prox}_{\lambda_2\mathrm{TV}_1^{1\mathrm{D}}(\cdot)}(y)\right)$. The inner proximity operator can be computed by using the proximity solver presented in Section 4.1.2, while the outer corresponds to an L_1 norm and thus can be calculated in closed form by soft-thresholding (Section 4.1.1). Therefore, Fused–Lasso is solvable straightforwardly by making use of the proximity operator introduced above and a proximal method like Trip.

Additionally and taking advantage of the modularity of the Trip + proximity solver framework, variants of Fused Lasso with different choices of the loss or regularization functions can be addressed as well. Those are

- **Standard Fused–Lasso (FL):** Least–Squares loss + L_1 norm + TV_1^{1D} norm, as just presented.
- **ℓ_2-variable fusion (VF):** Least–Squares loss + L_1 norm + TV_2^{1D} norm. Though Variable fusion was already introduced in the past by Land and Friedman [111], their approach proposes an ℓ_p^p-like regularizer in the sense that $r(x) = \sum_{i=1}^{n-1} |x_{i+1} - x_i|^p$ instead of the TV regularizer $TV_p^{1D}(x) = \left(\sum_{i=1}^{n-1} |x_{i+1} - x_i|^p\right)^{1/p}$, which would correspond to a ℓ_p regularization in these terms. The ℓ_2 approach used here leads to a more conservative penalty that does not oversmooth the estimates. This FL variant seems to be new.
- **Logistic-fused lasso (LFL):** Logistic loss + L_1 norm + TV_1^{1D} norm. The logistic loss takes the form $L_{\log}(x, c) = \sum_i \log\left(1 + e^{-y_i(a_i^T x + c)}\right)$, and can be introduced in the FL formulation to obtain a more appropriate model for classification on a dataset $\{(a_i, y_i)\}$ [112].
- **Logistic + ℓ_2-fusion (LVF):** Logistic loss + L_1 norm + TV_2^{1D} norm.

All these models can be solved within the Trip framework if means to compute the gradient of the loss and the proximity operator of the regularizer are provided. The Least–Squares loss and the $L_1 + TV_1^{1D}$ regularizer were already addressed above for the standard Fused–Lasso. Regarding the Logistic loss, obtaining its gradient is straightforward. The only hard point is dealing with a proximity operator for the $L_1 + TV_2^{1D}$ regularizer, as in this case decomposition into the proximity operators for L_1 and TV_2^{1D} is not possible. Still, a solution can be found by making use of the **Proximal Dykstra** (PD) method [80], yet another algorithm making use of proximity operators. This one is designed to solve the problem

$$\min_x \quad f(x) + g(x) + \tfrac{1}{2}\|x - z\|_2^2,$$

where $f(x), g(x)$ are both lower-semicontinuous functions. The structure of this problem makes it ideally suited to obtain the proximity operator of a combination of regularizers, since the proximity operator of the combined regularizer $r(x) = \lambda_1 r_1(x) + \lambda_2 r_2(x)$ (like the ones above) has the form

$$\text{prox}_{r(\cdot)}(y) = \min_x \tfrac{1}{2}\|x - y\|_2^2 + \lambda_1 r_1(x) + \lambda_2 r_2(x),$$

which coincides with the problem solved by the PD method as long as $r_1(x), r_2(x)$ are lower-semicontinuous (all the regularizers above meet this condition).

Algorithm 11 Proximal Dykstra

Inputs: f, g, z.
Initialize $x_0 = z$, $p_0 = q_0 = 0$, $t = 0$.
while stopping criterion not met **do**
  g proximity operator: $y_t = \text{prox}_{g(\cdot)}(x_t + p_t)$.
  g step: $p_{t+1} = x_t + p_t - y_t$.
  f proximity operator: $x_{t+1} = \text{prox}_{f(\cdot)}(y_t + q_t)$.
  f step: $q_{t+1} = y_t + q_t - x_{t+1}$.
  $t \leftarrow t + 1$.
end while
Return x_t.

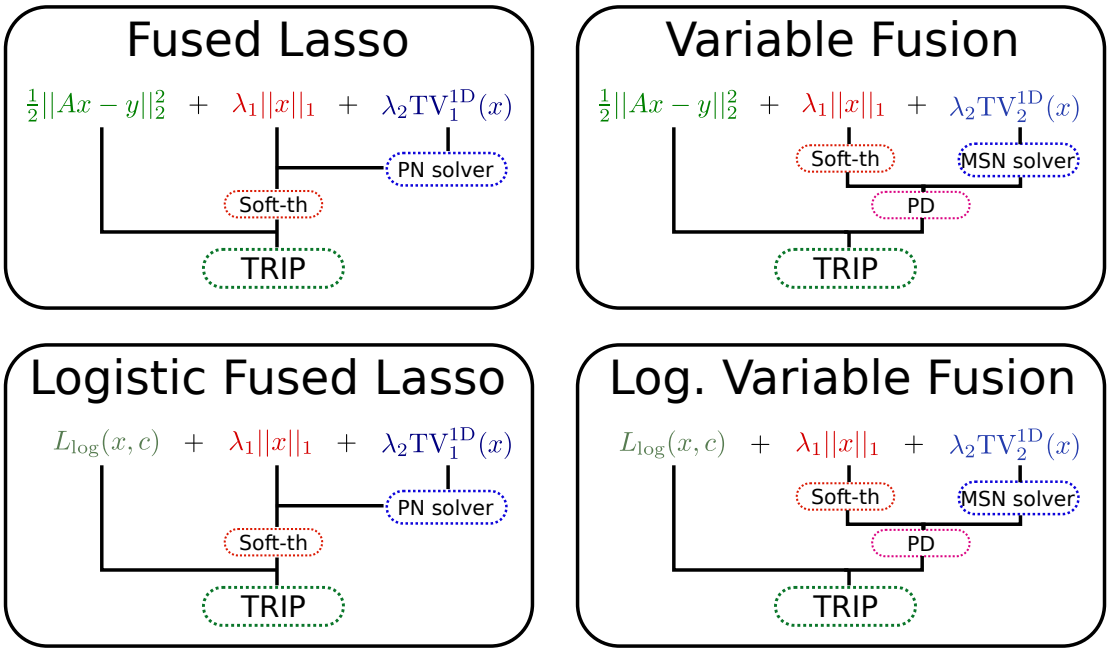


FIGURE 4.3: Fuses–Lasso models addressed and hierarchy of algorithms used to solve them.

The pseudocode of the Proximal Dykstra method is shown in Algorithm 11. The algorithm has been shown to produce a sequence of x_t converging to the optimum of the problem [113].

Using PD the proximity operator of a pair of regularizers can be obtained as long as the proximity operator for each of them can be computed individually. As the regularizers above reduce to the L_1, TV_1^{1D} and TV_2^{1D} norms, proximity solvers for all of them are also available. For clarity, Figure 4.3 shows the methods and proximity solvers employed for each of the Fused–Lasso models presented above. Numerical experiments studying the efficiency of these solvers for the cited models are presented in what follows.

Model	SLEP			TRIP		
N	10^3	10^4	10^5	10^3	10^4	10^5
FL	0.089	1.43	41.80	**0.02**	**0.10**	**0.86**
VF	0.16	1.26	35.77	**0.02**	**0.10**	**0.90**
LFL	**0.21**	15.01	144.81	0.78	**5.35**	**53.88**
LVF	0.86	**0.02**	132.13	**0.81**	0.15	**11.24**

TABLE 4.2: Running times (secs) for SLEP and TRIP for optimizing different versions of fused-lasso with increasing input sizes. Both methods were run to satisfy the same convergence criterion.

4.2.1 Experiments on synthetic data

Since the standard Fused–Lasso model has been throughly studied in the literature, a number of practical algorithms addressing it have been already introduced in the past. One outstanding example of such is the recent Fused–Lasso algorithm in the **SLEP** library by Liu et al. [103], which results to be an adapted FISTA method [78] based on an efficient proximity step (FLSA). The previous experiments for TV_1^{1D}-proximity (Section 4.1.4) have already shown the superiority of the proximity solver proposed here over FLSA, and so what remains is to check whether the TRIP strategy followed here is also superior to the FISTA strategy in SLEP. To do so, first a test with synthetic data is performed.

Random matrices $A \in \mathbb{R}^{N\times M}$ were generated, whose entries were selected to follow a zero mean, unit variance normal distribution. The number of variables was fixed to $M = 100$, and the penalties were set to $\lambda_1 = \lambda_2 = 0.01$. Then, matrices were sampled with the number of columns (patterns) N varying as 10^3, 10^4, and 10^5. To select the vector of responses y, the formula $y = \text{sgn}(Ax_t + v)$ was followed, where x_t, and v are random vectors whose entries have variances 1 and 0.01, respectively. The numerical results are summarized in Table 4.2, where SLEP (version 4.0) [105] is compared against the TRIP-based approach. Since SLEP only supports the FL and LFL models, the TV_2^{1D}-proximity operator developed here is also coded within SLEP to allow the comparison for the VF and LVF models. While for smaller matrices with $N = 10^3$ both methods run similarly fast, as the size of the input matrices increases, the TRIP-based fused-lasso solvers run much faster than SLEP.

4.2.2 Microarray classification

Addressing now an applied bioinformatics problem, the four FL models were tested on real binary classification tasks for the following microarray datasets: ArrayCGH [114],

Dataset	FL	VF	LFL	LVF
ArrayCGH	73.6%	**78.9%**	73.6%	73.6%
Leukemias	92.0%	92.0%	**96.0%**	**96.0%**
Colon	**77.2%**	**77.2%**	**77.2%**	**77.2%**
Ovarian	**88.8%**	83.3%	77.7%	77.7%
Rat	67.2%	65.5%	**70.4%**	**70.4%**

TABLE 4.3: Classification accuracies for the presented Fused–Lasso models on microarray data.

Leukemias [115], Colon [116], Ovarian [117] and Rat [118]. Each dataset was split into three equal parts (ensuring both classes where present in every split) for training, validation and test. The penalty parameters were found by exhaustive grid search in the range $\lambda_1, \lambda_2 \in [10^{-3}, 10^{1}]$ to maximize classification accuracy on the validation splits.

Table 4.3 shows test accuracies. In general, logistic-loss based FL models yield better classification accuracies than those based on least-squares. This result is natural: logistic-loss is more suited for classification tasks, since it turns out to be an smooth approximation to the hinge loss (already introduced in Section 1.4). Regarding the TV–regularizer, three out of five datasets seem to be insensitive to this choice, though the $\mathrm{TV}_1^{1\mathrm{D}}$-penalty performs better for Ovarian, while $\mathrm{TV}_2^{1\mathrm{D}}$ works best for ArrayCGH. It can be concluded, thus, that new proposed variants of Fused–Lasso using $\mathrm{TV}_2^{1\mathrm{D}}$ regularization can be of use in some situations.

4.3 2-dimensional TV-regularization

The previous section has shown how $\mathrm{TV}_p^{1\mathrm{D}}$-proximity can be solved efficiently, and building on top of this, how practical models using this regularization can be solved. Now a 2-dimensional version of this regularizer is considered, taking the form

$$\mathrm{TV}_{p,q}^{2\mathrm{D}}(x) = \sum_{i=1}^{N} \left(\sum_{j=1}^{M-1} |x_{i,j+1} - x_{i,j}|^p \right)^{1/p} + \sum_{j=1}^{M} \left(\sum_{i=1}^{N-1} |x_{i+1,j} - x_{i,j}|^q \right)^{1/q},$$

for a 2-dimensional input $x \in \mathbb{R}^{N \times M}$. The effect of this regularizer is applying a $\mathrm{TV}_p^{1\mathrm{D}}$ regularization over each row of x, and a $\mathrm{TV}_q^{1\mathrm{D}}$ regularization over each column. When $p = q$ the same kind of regularization is applied over the rows and the columns. By defining D_N and D_M differencing matrices for the row and column dimensions the regularizer can be rewritten as

$$\mathrm{TV}^{2\mathrm{D}}_{p,q}(x) = \sum_i \|D_N x_{i,:}\|_p + \sum_j \|D_M x_{:,j}\|_q,$$

where $x_{i,:}$ denotes the i-th row of x and $x_{:,j}$ its j-th column. That is, the $\mathrm{TV}^{2\mathrm{D}}_{p,q}$ regularizer decomposes into N $\mathrm{TV}^{1\mathrm{D}}_{p}$ and M $\mathrm{TV}^{1\mathrm{D}}_{q}$ regularizers. This observation is relevant because, similarly to the Fused-Lasso models of Section 4.2, proximity for $\mathrm{TV}^{2\mathrm{D}}_{p,q}$ is solvable by making use of the basic $\mathrm{TV}^{1\mathrm{D}}_{p}$ proximity solvers. And, analogously to 1–dimensional TV, being able to compute proximity for 2–dimensional TV allows to solve problems of interest through the use of proximal solvers, as is shown later on.

The corresponding $\mathrm{TV}^{2\mathrm{D}}_{p,q}$-proximity problem is

$$\min_x \quad \tfrac{1}{2}\|x-y\|^2_{\mathrm{F}} + \lambda\, \mathrm{TV}^{2\mathrm{D}}_{p,q}(x),$$

where the Frobenius norm $\|x\|_{\mathrm{F}}$ is defined as $\|x\|_{\mathrm{F}} = \sqrt{\mathrm{Tr}(x^T x)}$, or equivalently, $\|x\|_{\mathrm{F}} = \sqrt{\sum_i \sum_j x^2_{i,j}} = \|\mathrm{vec}(x)\|_2$ where $\mathrm{vec}(x)$ is the vectorization of x. By using the decomposition before

$$\min_x \quad \tfrac{1}{2}\|x-y\|^2_{\mathrm{F}} + \lambda \left(\sum_i \|D_N x_{i,:}\|_p \right) + \lambda \left(\sum_j \|D_M x_{:,j}\|_q \right),$$

where the parenthesis make explicit that the regularizer is in fact combination of two regularizers: one acting over the rows and the other over the columns. The resulting problem clearly fits into the model solved by the Proximal Dykstra algorithm already introduced (Section 4.2). Therefore, $\mathrm{TV}^{2\mathrm{D}}_{p,q}$-proximity can be solved via this method. There is a difference, though, which is that each of the two regularizers involved is in turn composed by the sum of a number (N or M) of 1–dimensional TV regularizers. However, each of those operates over a different row or columns of x, and thus they can be solved independently.

To clarify things, Algorithm 12 shows how $\mathrm{TV}^{2\mathrm{D}}_{p,q}$-proximity can be solved through the Proximal Dykstra algorithm and $\mathrm{TV}^{1\mathrm{D}}_{p}$ and $\mathrm{TV}^{1\mathrm{D}}_{q}$-proximity solvers. A useful remark is that in the proximity steps, all the proximity operators can be computed simultaneously, since they operate in different rows / columns. This naturally leads to a parallel implementation of such steps; details on this are given in the experimental reports (Section 4.3.4).

Algorithm 12 Proximal Dykstra adapted for $\mathrm{TV}^{2\mathrm{D}}_{p,q}$-proximity

Inputs: x, λ, y, p, q.
Initialize $x^0 = y$, $u^0 = w^0 = 0$, $t = 0$.
while stopping criterion not met **do**
  Proximity over the columns: $v^t_{:,j} = \mathrm{prox}_{\mathrm{TV}^{1\mathrm{D}}_q}(x^t_{:,j} + u^t_{:,j})\ \forall\ j = 1, \ldots, M$.
  Columns step: $u^{t+1} = x^t + u^t - v^t$.
  Proximity over the rows: $x^{t+1}_{i,:} = \mathrm{prox}_{\mathrm{TV}^{1\mathrm{D}}_p}(v^t_{i,:} + w^t_{i,:})\ \forall\ i = 1, \ldots, N$.
  Rows step: $w^{t+1} = v^t + w^t - x^{t+1}$.
  $t \leftarrow t + 1$.
end while
Return x^t.

4.3.1 Anisotropic filtering

A first straightforward application of the $\mathrm{TV}^{2\mathrm{D}}_{p,q}$ regularizer in the field of image processing is **anisotropic filtering**, which is an approach to **image denoising**. Given an image $u \in \mathbb{R}^{N \times M}$ contaminated by additive noise n in the form

$$u_0 = u + n,$$

where only u_0 is observed, the denoising problem involves obtaining an estimate of the original image u from its noisy version u_0. Such task is by no means approachable unless additional assumptions on the noise properties are made. It is therefore common to assume that the noise follows a Gaussian distribution, or some other zero-mean distribution. Under these conditions, a classic method to perform such denoising task is given by the **Rudin–Osher–Fatemi** (ROF) model [119], which finds an approximation x to the original image as

$$\min_x \quad \|x - u_0\|^2_{\mathrm{F}} + \lambda \sum_{i=2}^{N} \sum_{j=2}^{M} \|\partial x_{i,j}\|_2,$$

where $\partial x_{i,j}$ is defined as a discrete gradient at the (i, j)-th point of the image, which is computed as

$$\partial x_{i,j} = \begin{bmatrix} x_{i,j} - x_{i-1,j} \\ x_{i,j} - x_{i,j-1} \end{bmatrix},$$

that is, it is the vector of differences of $x_{i,j}$ and its neighbors in both axes. The objective of the first term in the ROF model is to penalize any deviation of x from the observed image u_0, while the second term can be readily recognized as a mixed $(2, 1)$ norm over

the gradient of x. This particular choice of regularization attends to prior knowledge about the problem: in natural images sharp discontinuities in intensity between neighboring points only appear in borders of objects, while the rest of the pixels usually show smooth variations in intensity. It makes sense, therefore, to regularize large values of the gradient, as sharp changes have a higher probability of having being produced by noise. Conversely, as the mean of the noise is zero, it is also sensible to maintain the denoised image x close to the observed u_0. Joining these two goals produces the ROF model as presented.

A closer look at the regularizer employed in the ROF model reveals that it follows the spirit of the 2–dimensional Total-Variation regularizer as presented here, in the sense of penalizing variations between neighboring pixels. In fact, this kind of regularizer is also generally known as Total-Variation within the image processing community. It is clear, though, that this regularizer does not coincide with the $\mathrm{TV}^{\mathrm{2D}}_{p,q}$ regularizer presented so far. Because of this, some authors [79] differentiate these two regularizers by naming the ROF approach as **isotropic TV** and the $\mathrm{TV}^{\mathrm{2D}}_{p,q}$ approach as **anisotropic TV**. This naming comes from the fact that isotropic TV penalizes each component of the discrete gradient $\partial x_{i,j}$ following an L_2 norm, i.e. a ball-shaped norm, putting the same weight over every direction. Conversely, the anisotropic $\mathrm{TV}^{\mathrm{2D}}_{p,q}$ and in particular $\mathrm{TV}^{\mathrm{2D}}_{1,1}$ penalizes rows and columns independently.

While image filtering using isotropic TV is generally preferred for natural images denoising [120], in some settings anisotropic filtering can produce better results, and in fact has been favored by some authors in the past [121, 122]. This is specially true on those images that present a "blocky" structure, and thus are better suited to the structure imposed by the $\mathrm{TV}^{\mathrm{2D}}_{p,q}$ regularizer. An example of this can be seen in Figures 4.4 and 4.5. In the first one the famous image *Lena* is corrupted with gaussian noise, and then filtered with both an isotropic and an anisotropic model. Visually, isotropic filtering produces a slightly better reconstruction. Conversely, in the second Figure a QR barcode is filtered, and this time the anisotropic model produces better looking results (quantitative results for both classes of filters are given later in Section 4.3.4). Therefore, efficient methods to perform anisotropic filtering are also of interest.

The denoising problem using the anisotropic $\mathrm{TV}^{\mathrm{2D}}_{p,q}$ regularizer has the expected form

$$\min_{x} \quad \|x - u_0\|_{\mathrm{F}}^2 + \lambda \mathrm{TV}^{\mathrm{2D}}_{p,q}.$$

This is exactly the $\mathrm{TV}^{\mathrm{2D}}_{p,q}$-proximity problem, and hence can be solved directly by the method presented above. In practice, filtering is made by choosing $p = q = 1$, thus simplifying things.

(a) (b)

(c) (d)

FIGURE 4.4: *Lena image* (a) Original image (b) Noisy image (c) Isotropic denoising (d) Anisotropic denoising.

4.3.2 Image deconvolution

Taking a step forward from image filtering the problem of **image deconvolution** (or image deblurring) is confronted. In this more complex setting the image recovery task is made harder by the presence of a **convolution kernel** K in the form of a linear operator, which introduces further distortion in the image as

$$u_0 = K * u + n,$$

for n a noise vector, and $*$ the convolution operation. Thus, to recover the original image u from the observed u_0 the following deconvolution problem needs to be solved

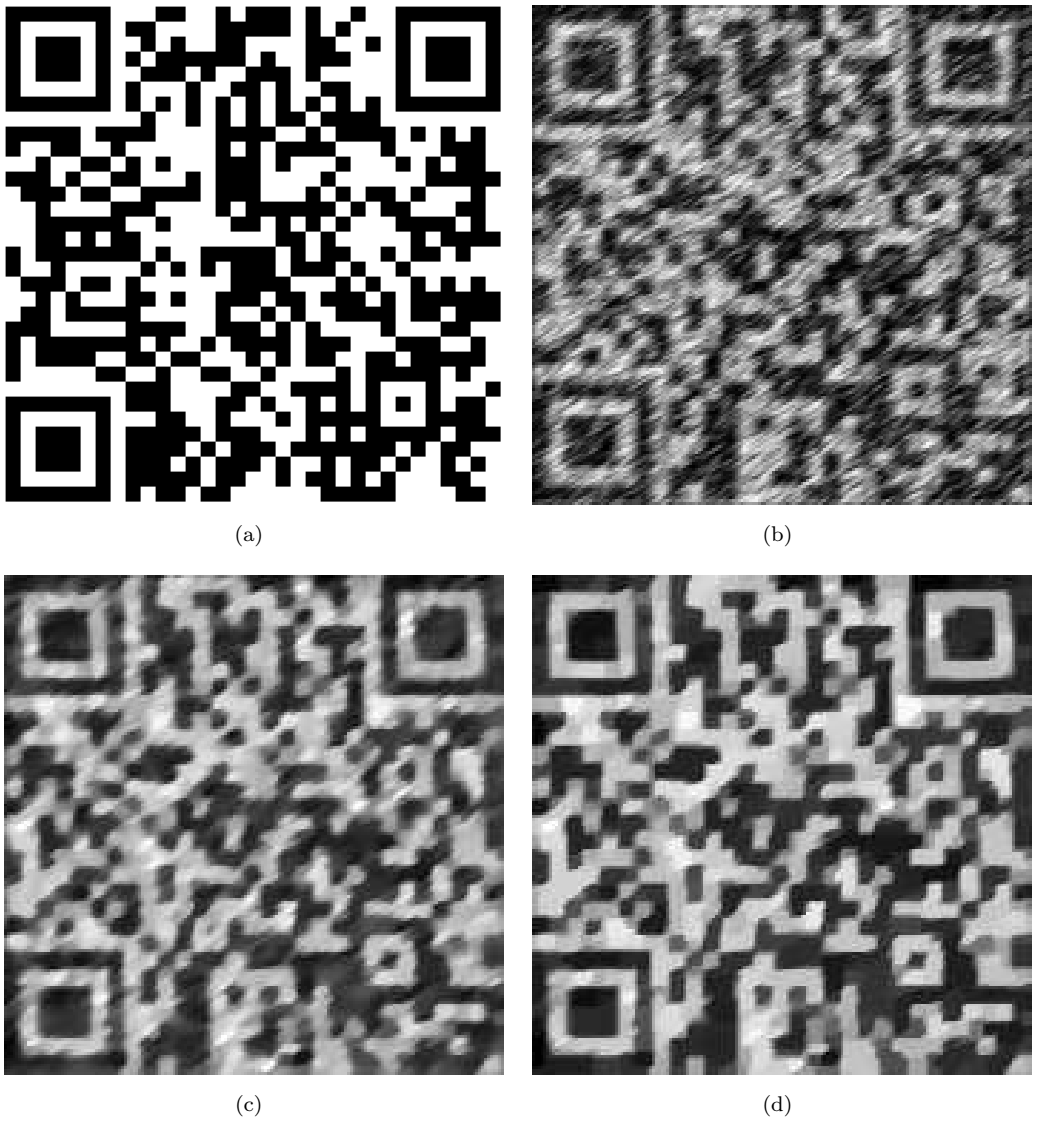
(a) (b)

(c) (d)

FIGURE 4.5: QR code image (a) Original image (b) Noisy image (c) Isotropic denoising (d) Anisotropic denoising.

$$\min_{x} \quad \tfrac{1}{2}\|K * x - u\|_{\mathrm{F}}^2 + \lambda R(x).$$

Again, the regularization term $R(x)$ can be an isotropic or an anisotropic regularizer (among others), each of them being better suited for different settings. An example of deconvolution where an anisotropic filter performs better is shown in Figure 4.6. This problem is harder than the denoising one due to the inclusion of K, but nonetheless can be addressed by making use of a proximal algorithm as in the case of the Fused–Lasso problem. Several solvers of this kind explicitly designed for dealing with the convolution operator are available in the literature [78, 79, 87], which only require to provide a function to solve the denoising problem, i.e. the proximity operator. A



(a) (b)

(c) (d)

FIGURE 4.6: QR code image (a) Original image (b) Image convoluted with a motion blur kernel (c) Isotropic deconvolution (d) Anisotropic deconvolution.

notable solver specific for the isotropic case is given by the work of Krishnan and Fergus [123], providing fast solutions even for non–convex versions of the isotropic TV norm ($0 < p < 1$). Unfortunately, such approach can not be extended to the anisotropic case, and so here the focus is placed on more general proximal solvers addressing any kind of regularizer $R(x)$. To this end, experiments showing how the $\mathrm{TV}^{2\mathrm{D}}_{p,q}$-proximity solver can be plugged into the recent proximal method **SALSA** [87] are presented in Section 4.3.4.

4.3.3 2-dimensional Fused–Lasso Signal Approximator

The **Fused–Lasso Signal Approximator** (FLSA) [106] can be regarded as a particular case of Fused–Lasso where the input matrix A is the identity matrix I, i.e.

$$\begin{aligned} & \min_x \quad \tfrac{1}{2}\|Ix - y\|_2^2 + \lambda_1\|x\|_1 + \lambda_2 \mathrm{TV}_1^{1\mathrm{D}}(x), \\ = & \min_x \quad \tfrac{1}{2}\|x - y\|_2^2 + \lambda_1\|x\|_1 + \lambda_2 \mathrm{TV}_1^{1\mathrm{D}}(x). \end{aligned}$$

In the context of this thesis, FLSA appears as the proximity operator of the $\mathrm{L}_1 + \mathrm{TV}_1^{1\mathrm{D}}$ regularizer, and as such can be solved straightforwardly following the methods presented in Section 4.2. A slightly more interesting problem is the one posed by the 2-dimensional variant of FLSA, taking the form

$$\min_x \quad \|x - y\|_{\mathrm{F}}^2 + \lambda_1\|\mathrm{vec}(x)\|_1 + \lambda_2 \mathrm{TV}_{1,1}^{2\mathrm{D}}.$$

This model has been used in [106] for the denoising of signals where a large number of pixels are known to be completely black (intensity 0), thus adequately exploiting the structure imposed by the L_1 regularizer.

Similarly to the 1-dimensional case, 2-dimensional FLSA can be solved by realizing again that the problem is just a proximity operator of the $\mathrm{L}_1 + \mathrm{TV}_{1,1}^{2\mathrm{D}}$ regularizer. Fortunately, in this case this proximity operator can again be directly decomposed into the two constituting proximity operators [106], as

$$\mathrm{prox}_{\lambda_1\|\cdot\|_1 + \lambda_2 \mathrm{TV}_{1,1}^{2\mathrm{D}}(\cdot)}(y) = \mathrm{prox}_{\lambda_1\|\cdot\|_1}\left(\mathrm{prox}_{\lambda_2 \mathrm{TV}_{1,1}^{2\mathrm{D}}(\cdot)}(y)\right).$$

Therefore, solving 2-dimensional FLSA amounts to calling the developed $\mathrm{TV}_{p,q}^{2\mathrm{D}}$ proximity solver and then applying soft-thresholding to the results. Since the soft-thresholding step is done in closed form, the performance of an FLSA solver depends only on its ability to compute $\mathrm{TV}_{p,q}^{2\mathrm{D}}$-proximity efficiently. Experimental results (Section 4.3.4) show that the proximity solver proposed here amply fulfills this requirement.

4.3.4 Experiments

Since the developed $\mathrm{TV}_{p,q}^{2\mathrm{D}}$-proximity method is based on its corresponding 1–dimensional solvers, an implementation of it was also made in the C programming language. The Matlab toolbox publicly available at `http://arantxa.ii.uam.es/~gaa/software.html` contains the implementation of this method that was used to run the following experiments.

4.3.4.1 Image denoising

The most immediate application of the developed $\mathrm{TV}^{2\mathrm{D}}_{p,q}$-proximity is to image denoising, as already presented, since it constitutes a problem directly solvable by this method. As stated before, an image filter making use of a $\mathrm{TV}^{2\mathrm{D}}_{p,q}$ regularizer corresponds to an anisotropic filtering, which produces superior quality reconstructions when the images to denoise present a blocky structure. A good example of such are 2-dimensional QR barcodes, where anisotropic filters have already been applied [121]. Images where the gradient is generally constant are also a good field of application of this filtering, for instance diagrams or images with plain colors (i.e. not presenting color gradients).

In what follows the presented proximity operator based in Proximal Dykstra (PD) is put into practice for those settings. Norms $p = 1$ and $q = 1$ are used for the filtering, since this is the standard and actually produces better results than, for instance, $p = q = 2$. For this $\mathrm{TV}^{2\mathrm{D}}_{1,1}$ choice, the Proximal Dykstra method just requires the use of the $\mathrm{TV}^{1\mathrm{D}}_{1}$ Projected Newton solver.

In order to offer a contrast with other existing methods, experiments were run also for the following algorithms:

- The state–of–the–art **Primal Dual Hybrid Gradient** (PDHG) method by Zhu et al. [93] for isotropic denoising.
- The **Pathwise Coordinate Optimization** method by Friedman et al. [106], designed to solve the FLSA problem (Section 4.3.3), and hence performing an anisotropic denoising if a weight $\lambda_1 = 0$ is given to the L_1 norm regularizer.
- An adapted PDHG method solving the anisotropic case, which was easily obtained by modifying the original formulation. This is to check whether such method is able to perform well also for this case. It must be mentioned that the parameter selection rules recommended for the original PDHG failed to produce fast running times when applied in this context. Thus, to make PDHG competitive, optimal parameters were obtained by exhaustive grid search aiming to minimize training times over the *randomQR-0* image.
- A median filter, which is standard for filtering of images presenting a clear spatial structure [124].

The images used in the experiments are displayed in Figure 4.7 (at the end of the chapter). QR barcode images were generated by encoding random text using Google

Image	Gaussian	Speckle	Poisson	Salt & Pepper
randomQR	0.2	0.3	∅	∅
shape	0.05	∅	∅	∅
trollface	∅	1	∅	∅
diagram	∅	∅	✓	∅
text	∅	∅	∅	0.1
comic	0.05	∅	✓	∅
contour	∅	∅	✓	0.4
phantom	∅	2	✓	∅

TABLE 4.4: Kinds of noise and parameters for each test image. A ∅ indicates that such noise was not applied for the image. *Gaussian* and *Speckle* correspond to gaussian additive and multiplicative (respectively) noises with zero mean and the indicated variance. *Salt & Pepper* noise turns into black or white the indicated fraction of image pixels. *Poisson* regenerates each pixel by drawing a random value from a Poisson distribution with mean equal to the original pixel value, thus producing a more realistic noise.

chart API [125]. Images *shape* and *phantom* [5] are publicly available and frequently used in image processing. *trollface* and *comic* [6] are also publicly available. The rest of the images were originally created by the author.

To test the filters under a variety of scenarios, different kinds of noise were introduced for each image. Table 4.4 gives details on this, while the noisy images are shown in Figure 4.8. All QR barcode images used the same kind and parameters of noise, for reasons to be discussed next. Noise was introduced using Matlab's *imnoise* function.

Values for the regularization parameter λ in the isotropic and anisotropic models were found by maximizing the quality of the reconstruction, measured in **Improved Signal-to-Noise Ratio** (ISNR) [87], with the exception of the QR barcode images. In those, λ is optimized only for the *randomQR-2* image, and then that value is used for the rest of QR images as well. This is to see how a fixed choice of λ behaves when applied to different though similar images. ISNR is defined as

$$\mathrm{ISNR}(x, u, u_0) = 10 \log_{10} \frac{\|u_0 - x\|_{\mathrm{F}}^2}{\|x - u\|_{\mathrm{F}}^2},$$

where u is the original image, u_0 is its observed noisy variant, and x is the reconstruction obtained.

Table 4.5 presents the runtimes and ISNR values that were obtained in the denoising experiments. Pathwise Coordinate Optimization and anisotropic PDHG were not applied

[5] Extracted from `http://en.wikipedia.org/wiki/File:Shepp_logan.png`
[6] Author: Francisco Molina. `http://www.afrikislife.net/english/`

	Anisotropic				Isotropic		Median	
Image	ISNR	PD	Coord.	PDHG	ISNR	PDHG	ISNR	Time
randomQR-0	2.39	0.11	2.85	0.64	2.04	0.03	1.24	0.00
randomQR-1	4.14	0.27	15.99	8.71	3.38	0.11	1.74	0.02
randomQR-2	5.48	0.88	140.78	128.72	4.38	0.37	2.35	0.03
randomQR-3	6.04	1.39	167.68	93.87	4.39	0.76	2.42	0.07
randomQR-4	4.42	2.59	228.55	203.19	3.58	1.30	2.18	0.09
shape	6.02	0.09	45.51	7.75	4.50	0.03	-0.60	0.01
trollface	8.00	1.68	1899.17	316.80	7.69	0.65	2.89	0.05
diagram	6.73	0.32	1937.05	53.12	4.84	0.13	-6.24	0.02
text	5.02	4.00	296.35	1287.11	3.72	1.99	4.49	0.07
comic	7.04	1.09	491.91	209.47	6.13	0.51	2.18	0.05
contour	12.49	11.19	∅	∅	10.71	5.96	9.12	0.33
phantom	6.09	49.78	∅	∅	6.00	16.59	2.78	0.60

TABLE 4.5: Denoising results obtained via the proposed Proximal Dykstra (PD) method, Coordinate Optimization and adapted PDHG for the anisotropic model, genuine PDHG for the isotropic model, and a median filter. ISNR (dB) values (higher is better) and running times for each method in seconds are shown. All methods solving the anisotropic model produce roughly the same ISNR value. Cases showing a ∅ were not tested because of excessive requirements in running times.

to the largest images, as they required excessive running times. The actual denoised images are shown in Figures 4.9, 4.10 and 4.11. To compensate for the loss of contrast produced by filtering, intensity values were rescaled to the range of the original image.

In general, the median filter performs worse than the isotropic filter, which in turn shows lower quality reconstructions than the anisotropic filter, for all the tested types of noise (in terms of ISNR). In the *QR* images it is confirmed visually that the median filter produces distorted reconstructions where noise still abounds. Even though the median filter works well in images with clear spatial structure like these, the speckle (multiplicative) noise makes this filter fail, as computing medians no longer gives a good reconstruction. Running again these experiments with just gaussian additive noise (not shown here), the median filter does produce good results. Still, the anisotropic and isotropic filters perform well in both settings. It is further visually checked that the anisotropic version does a better job.

Regarding the rest of images, the advantage of the anisotropic filter can be generally confirmed by looking at the image regions presenting constant intensity: artificial changes in intensity appear when applying the isotropic filter. The median filter generally does a bad job in removing the noise, and thus attains smaller ISNR values. It is remarkable, though, that in spite of this lower ISNR, better-looking results are obtained in *text* and

contour, as the isotropic and anisotropic filters produce an undesired blurring of the image. This fact suggests that in some situations the use of a metric different from ISNR might better assess the visually-perceived quality of the reconstructed image. Nevertheless ISNR (or SNR) is still a widely-used quality measure for image reconstruction.

In any case, it must be stressed that the main point of these experiments is showing how the proposed $\mathrm{TV}^{2\mathrm{D}}_{p,q}$-proximity method efficiently solves the anisotropic filtering problem, and how it can produce better reconstructions than isotropic filtering in some settings. In spite of this, a vast number of filtering methods have been developed in the field of image processing, many of them producing higher-quality reconstructions than these approaches, such as for instance the collaborative filtering approach of Dabov et al. [126]. Nevertheless, efficient methods for isotropic filtering is still an active area of research, as it can be extended to deal with harder problems such as deblurring, inpainting or super-resolution [127, 128], and so the anisotropic approach followed here is also of interest. Therefore, the relevant discussion comes in terms of running times, which comes next.

Looking again at the results, the proposed Proximal Dykstra solver vastly outperforms Coordinate Optimization and anisotropic PDHG in terms of efficiency. The isotropic version of the problem is simpler than the anisotropic one, so it is no surprise that the carefully tuned PDHG approach by Zhu et al. requires less time than Proximal Dykstra. Indeed, the performance of the PDHG method seems to depend heavily on the choice of its parameters. While the authors provide some recommended values that ensure convergence, in practice a set of involved parameter-setting rules are required to obtain good performance. These rules, when applied to the anisotropic case, failed to produce good running times. Even a careful selection of parameter values like the one done here seems to be unable to make the method run fast. In utter contrast to this, the method proposed here requires no parameter tuning at all.

On a side note, it is also worth mentioning that in [121] an L_1 loss is proposed instead of the $\|x-y\|^2_{\mathrm{F}}$ presented here, and denoising is then cast as a Linear Program, to which a generic solver is applied; this approach requires runtimes of over 10^3 seconds for the *randomQR-4* image, and so it is of no use in practice.

It must also be noted that the results of Table 4.5 additionally prove that the proposed Proximal Dykstra method is able to solve the 2–dimensional Fused–Signal–Approximator–Problem (Section 4.3.3) faster than the Pathwise Coordinate Optimization method proposed by its authors [106], since, as stated before, the FLSA problem is solved by applying soft-thresholding to the solution of $\mathrm{TV}^{2\mathrm{D}}_{1,1}$-proximity, which in turn is the anisotropic filtering problem tested here. It should be remarked, though, that the Coordinate Optimization method computes the full regularization path for λ instead of just for a single value of it. Still, and similarly to the 1–dimensional Fused–Lasso problem, the speed of

Image	1 processor	2 processors	Reduction
shape	0.09	0.10	111.1 %
trollface	1.68	1.43	85.1 %
diagram	0.32	0.40	125 %
text	4.00	3.14	78.5 %
comic	1.09	1.01	92.6 %
contour	11.19	8.04	71.8 %
phantom	49.78	28.65	57.5 %

TABLE 4.6: Running times (in seconds) for the PD algorithm in anisotropic filtering when 1 or 2 parallel processors are used in the computation, together with the percentage of time required by 2 processors w.r.t. 1 processor.

Proximal Dykstra allows to rapidly compute solutions for several values of λ if required, and so this is not a problem in practice.

Finally and to illustrate the parallelization potential of the developed anisotropic filtering method, Table 4.6 shows running times for the PD algorithm when 1 or 2 parallel processors are used in the computation of some of the experiments in Table 4.5. While for the small images using 2 processors does not seem to be advantage, clear reductions in times are obtained for the larger *text*, *contour*, and *phantom* images. This is not surprising: any parallel process incurs into a costly overhead due to thread management and synchronization operations. This overhead, however, scales only with the number of parallel threads, not with the size of the data, i.e. the work to be performed by each thread. Therefore speedups coming from parallelization generally show up only for large quantities of data, where the parallelization overhead becomes negligible when compared to the computations performed by each thread. Precisely this effect is observed here, and as the input image becomes larger, the running times when using 2 processors becomes closer to the theoretically optimal 50 % reduction factor.

4.3.4.2 Image deconvolution

As stated before, the problem of image deconvolution can be addressed as an extension to image denoising, and in fact deconvolution can be solved by building over a denoising method. Here the deconvolution method **SALSA** [87] is used to run the experiments below, which only requires to supply a denoising function. Thanks to this modularity, anisotropic deconvolution can be performed by just plugging in the developed $\mathrm{TV}^{\mathrm{2D}}_{1,1}$ solver, as well as isotropic deconvolution by using SALSA's internal isotropic denoising filter, which is based on Chambolle's method [129]. In what follows these two approaches

Image	Convolution	Parameters
randomQR	Motion	Length 5, Angle 35^o
shape	Average	Size 3×3
trollface	Disk	Radius 5
diagram	Motion	Length 5, Angle 0^o
text	Average	Size 1×10
comic	Gaussian	Size 15, Deviation 2
contour	Disk	Radius 5
phantom	Motion	Length 100, Angle 240^o

TABLE 4.7: Convolution kernels used for each test image. *Average* substitutes each pixel with the average of its surrounding $n \times m$ neighbors. *Disk* performs the same operation within a disk-shaped neighborhood of the shown radius. *Gaussian* uses a $n \times n$ neighborhood and assigns different weights to each neighbor following the value of a gaussian distribution of the indicated deviation centered at the current pixel. *Motion* emulates the distortions produced when taking a picture in motion, defining a neighborhood following a vector of the indicated length and angle.

to deconvolution are compared, together with the standard **Richardson–Lucy** (RL) [130] method as implemented in Matlab.

Test images were the same as for the filtering experiments (Figure 4.7), using identical noise patterns (Table 4.4) for the QR images and just Gaussian noise with variance 0.05 for the rest. Additionally each image is convolved with a different type of kernel to observe the behavior of these methods for a variety of convolutions; Table 4.7 shows the types of kernels applied on each case. All the kernels were constructed using Matlab's *fspecial* function. The resulting images after convolution are shown in Figure 4.12.

Similarly to what was done for image filtering, values for λ in the isotropic and anisotropic models were found by maximizing the quality of the reconstruction, measured in ISNR, but for the QR barcode images, where λ is optimized only for the *randomQR-2* image and applied for all of them. Since deconvolution is a much more costly problem to solve, instead of performing and exhaustive search for the best λ choice, a Focused Grid Search strategy [131, 132] was used to find the best performing values. The Richardson-Lucy method does not require any parameter setting.

Table 4.8 shows the results obtained in terms of ISNR and running times. The actual deconvoluted images are shown in Figures 4.13, 4.14 and 4.15. As in the image denoising experiments before, the particular structure of the images used allows the anisotropic method to obtain superior quality results in terms of ISNR when compared to the isotropic alternative. Again, most of this gain in ISNR comes from a better reconstruction of large constant-color areas, where noise and blurring distortions are almost completely removed. Regarding the Richardson–Lucy method, its application

	Anisotropic		Isotropic		RL	
Size	ISNR	Time	ISNR	Time	ISNR	Time
randomQR-0	1.55	1.19	1.10	0.12	0.73	0.04
randomQR-1	2.79	0.81	2.15	0.55	0.79	0.18
randomQR-2	4.07	3.34	3.07	2.40	1.07	0.46
randomQR-3	4.05	5.41	2.92	3.71	1.13	0.61
randomQR-4	3.21	8.98	2.37	5.71	1.04	1.26
shape	2.13	0.83	1.01	0.15	0.13	0.07
trollface	0.99	8.08	0.40	2.08	0.23	1.14
diagram	1.61	2.30	0.33	9.24	-0.11	0.62
text	0.73	26.41	-0.09	147.01	0.35	2.76
comic	0.61	12.08	0.50	4.62	1.08	1.29
contour	0.82	86.60	0.64	18.63	-0.07	4.73
phantom	3.26	1326.29	0.73	155.47	2.19	32.91

TABLE 4.8: Deconvolution results for anisotropic and isotropic models using the SALSA solver, and Matlab's Richardson-Lucy (RL) method. ISNR (dB) values and runtimes (in secs) are shown.

generally better reverts the convolution; however, it leaves the noise mostly unchanged and produces some artifacts in the form of "shadow copies" of the image. Also, attention should be drawn again to the fact that higher ISNR does not necessarily translate into better-looking reconstructions: a good example of this are the *diagram*, *text* and *comic* images, where both the isotropic and anisotropic filters cannot remove all of the blurring, and so leave hard to read text pieces. Conversely, the Richardson–Lucy filter better maintains the sharpness of the text, though at the price of an inferior removal of noise.

Still, again the objective of these experiments is to show how the developed $\mathrm{TV}^{2\mathrm{D}}_{1,1}$ proximity solver can also be applied in the context of image deconvolution, producing an effective method to solve the problem. The image filtering experiments above already showed that this method stands out as clearly faster when compared with other anisotropic denoising methods, and so the same results are to be expected in the deconvolution setting. While other deconvolution approaches in the image processing literature might produce better quality results, the fact that an efficient anisotropic method is provided here is already of interest.

4.4 N-dimensional TV-regularization

Moving even beyond $\mathrm{TV}^{2\mathrm{D}}_{p,q}$, a **generalized multidimensional** version of TV-proximity can be considered. Suppose X to be an order-M tensor in $\mathbb{R}^{N_1\times N_2\times\cdots\times N_M}$, that is, an M

dimensional array indexed as $\mathsf{X}_{i_1,i_2,\ldots,i_M}$, where the indexes range as $i_1 \in \{1,\ldots,N_1\}$, $i_2 \in \{1,\ldots,N_2\}$, $\ldots$, $i_M \in \{1,\ldots,N_M\}$. Then the multidimensional Total-Variation could be defined as

$$\mathrm{TV}_p^M = \sum_{k=1}^{M} \sum_{\{i_1,\ldots,i_M\}\setminus i_k} \left(\sum_{j=1}^{N_k-1} |\mathsf{X}_{i_1,\ldots,i_{k-1},j+1,i_{k+1},\ldots,i_M} - \mathsf{X}_{i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_M}|^{p_k} \right)^{1/p_k},$$

for a vector of norms $p = [p_1,\ldots,p_M]$. This corresponds, for each dimension, to applying a 1-dimensional TV to each of the 1-dimensional rows of X along that dimension. For instance, if $\mathsf{X} \in \mathbb{R}^{N_1\times N_2}$, that is, a matrix, then 1-dimensional TV is applied over every row and column, as expected.

Introducing the **multi-index** $i(k) = (i_1,\ldots,i_{k-1},i_{k+1},\ldots,i_M)$, which iterates over every 1-dimensional row of X along the k-th dimension, the regularizer can be rewritten as a sum of M terms of the form

$$\mathrm{TV}_p^M = \sum_{k=1}^{M} \sum_{i(k)} \|D_{N_k} x_{i(k)}\|_{p_k},$$

where $x_{i(k)}$ denotes a row of X along the k-th dimension, and D_{N_k} is a differencing matrix of adequate size for the 1-dimensional rows along that dimension (of size N_k).

The corresponding M-dimensional-TV proximity problem is

$$\min_{\mathsf{X}} \ \tfrac{1}{2}\|\mathsf{X}-\mathsf{Y}\|_{\mathrm{F}}^2 + \lambda\,\mathrm{TV}_p^M(\mathsf{X}),$$

where $\lambda > 0$ is a penalty parameter, and the Frobenius norm for a tensor just denotes the ordinary sum-of-squares norm over the vectorization of such tensor. This can be further generalized by applying a different penalty for each dimension, as

$$\min_{\mathsf{X}} \ \tfrac{1}{2}\|\mathsf{X}-\mathsf{Y}\|_{\mathrm{F}}^2 + \sum_{k=1}^{M} \lambda_k \sum_{i(k)} \|D_{N_k} x_{i(k)}\|_{p_k},$$

This proximity operator looks very challenging. In spite of this, it is clearly decomposable, and as done already for the 2-dimensional TV case and the Fused Lasso models, it can be solved through the solutions of its constitutive proximity operators. More explicitly, the problem can be written as a sum of $\mathrm{TV}^{1\mathrm{D}}$ terms as

Algorithm 13 Parallel-Proximal Dykstra for N-dimensional TV

Inputs: $\mathsf{Y} \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_M}, \lambda, p$.
Initialize $\mathsf{X}_0 = \mathsf{Y}$, $\mathsf{Z}_0^k = 0$, for $1 \leq k \leq M$; $t = 0$
while ($\neg$ converged) **do**
  for $k = 1$ to M in *parallel* **do**
    $\mathsf{P}_t^k = \min_{\mathsf{X}} \frac{1}{2}\|\mathsf{X} - \mathsf{Z}_t^k\|_{\mathrm{F}}^2 + M\lambda_k \sum_{i(k)} \mathrm{TV}_{p_k}^{1\mathrm{D}}\left(x_{i(k)}\right)$
  end for
  $\mathsf{X}_{t+1} = \frac{1}{M}\sum_k \mathsf{P}_t^k$
  for $k = 1$ to M in *parallel* **do**
    $\mathsf{Z}_{t+1}^k = \mathsf{X}_{t+1} + \mathsf{Z}_t^k - \mathsf{P}_t^k$
  end for
  $t \leftarrow t + 1$
end while
Return X_t

$$\min_{\mathsf{X}} \ \frac{1}{2}\|\mathsf{X} - \mathsf{Y}\|_{\mathrm{F}}^2 + \sum_{k=1}^{M} \lambda_k \sum_{i(k)} \mathrm{TV}_{p_k}^{1\mathrm{D}}\left(x_{i(k)}\right).$$

This can be regarded as the sum of M proximity terms, each of them further decomposing into a number of inner $\mathrm{TV}^{1\mathrm{D}}$ terms. This latter inner decomposition is trivial to address since, as in the 2-dimensional TV, each of the $\mathrm{TV}^{1\mathrm{D}}$ terms operates on different entries of X. The problem comes from the M outer terms, which operate over the same X entries. Using Proximal Dykstra (Section 4.2) a problem of this kind can be solved for $M = 2$, though not for a general M. Fortunately, a generalization of this algorithm allows to effectively deal with M proximity terms: the **Parallel-Proximal Dykstra algorithm** [80, 133].

A pseudocode of the algorithm adapted to the problem at hand is shown in Algorithm 13. Convergence of this method towards the optimum has been proved [133]. Also, all the loops shown can be run in parallel, and in turn each of their iterates can be further parallelized to solve the $\mathrm{TV}_{p_k}^{1\mathrm{D}}$ proximity problems, resulting in a method with a vast potential for multi-thread computing.

4.4.1 Experiments

To the best of the knowledge of the author of this thesis, Total Variation regularization has not been applied before for N–dimensional data with $N > 2$. Still, as a sample of an application of the developed N–dimensional TV method, an anisotropic filtering for **video denoising** can be performed with it. By directly extending from the already presented 2–dimensional anisotropic filter (Section 4.3.1), a 3–dimensional filter for video can be devised as well. A video containing F frames of size $N \times M$ pixels

naturally fits into a 3–dimensional tensor $\mathsf{X} \in \mathbb{R}^{N\times M\times F}$, and so the 3–dimensional filter is straightforwardly defined as

$$\min_{\mathsf{X}} \quad \|\mathsf{X}-\mathsf{U}_0\|_{\mathrm{F}}^2 + \lambda \mathrm{TV}^{\mathrm{3D}}_{p_1,p_2,p_3}(\mathsf{X}),$$

with U_0 the observed noisy video, $\mathrm{TV}^{\mathrm{3D}}_{p_1,p_2,p_3} = \mathrm{TV}^3_p$, $p = [p_1, p_2, p_3]$. Like in the 2–dimensional case, application of the filter just requires solving the corresponding proximity operator, which can be done using the Parallel-Proximal Dykstra algorithm presented above.

As an example of application of this filtering method, the *salesman* video sequence was used, which is available at [134]. The video consists of 50 frames with a resolution of 288 $\times$ 352 grayscale pixels per frame, Figure 4.16 shows some of the frames of the sequence. Additive gaussian noise with zero mean and variance 10 was introduced independently into each frame of the sequence (Figure 4.17). Then this noisy sequence was filtered following a two-step procedure:

1. Using only the first 25 % frames of the video, look for the optimal value of the regularization parameter λ in terms of ISNR value of the reconstruction. This was done testing for a range of values of λ, and selecting the best performing one.
2. Filter the whole sequence using the λ value found.

The filtered sequence is shown in Figure 4.18. The anisotropic filter successfully removes the noise, obtaining an ISNR value of 5.6, although the expected slight blurring of the image appears as a result, as in the image experiments above. Once a good value for λ has been chosen, filtering the video takes just 33 seconds. Given that the sequence contains in total $50 \times 288 \times 352 = 5068800$ pixels, this could be roughly compared against the filtering of a 2–dimensional image of size $\simeq 2251 \times 2251$ pixels. Looking at the previous results for image denoising (Table 4.5), it seems like the Parallel-Proximal Dykstra algorithm performs well when compared against the filtering of the *countour* and *phantom* images. When the same filtering is performed in parallel using 2 processors, running time drops down to 19 seconds, confirming the suitability of the algorithm for multi-thread computation.

4.5 Discussion

This chapter has introduced a new series of methods based on Newton optimization for addressing the problem of Total Variation proximity. Not only do these methods perform

better than state–of–the–art algorithms, but also it has been shown how they can be used as building blocks to deal with more complex problems. Building on this and using recent proximal methods from the literature, efficient solvers for 2–dimensional and N–dimensional TV proximity, fused–lasso, and anisotropic image denoising and deconvolution have been constructed, showing good experimental performance. New approaches to the Fused–Lasso model and an anisotropic video filter have been proposed as well.

Two key ingredients have been fundamental for this success: a careful implementation of the basic proximal solvers and the use of appropriate proximal methods for each problem. While recent literature on proximal methods has provided ample algorithms and approaches for excellent solvers, at their core an efficient proximity algorithm is critical to ensure their performance. Therefore, efforts on this seems to be a more demanding priority.

Similarly to the SVM case in Chapter 3, this chapter has shown that by analyzing the problem at hand and taking full advantage of its structure applying tailored optimization methods, very efficient solvers can be developed. The modularity of these solvers when integrated into larger optimization frameworks not only guarantees their applicability in a broad range of problems but also, as seen here, their ability to naturally fit into a parallel framework, further guaranteeing scalability.

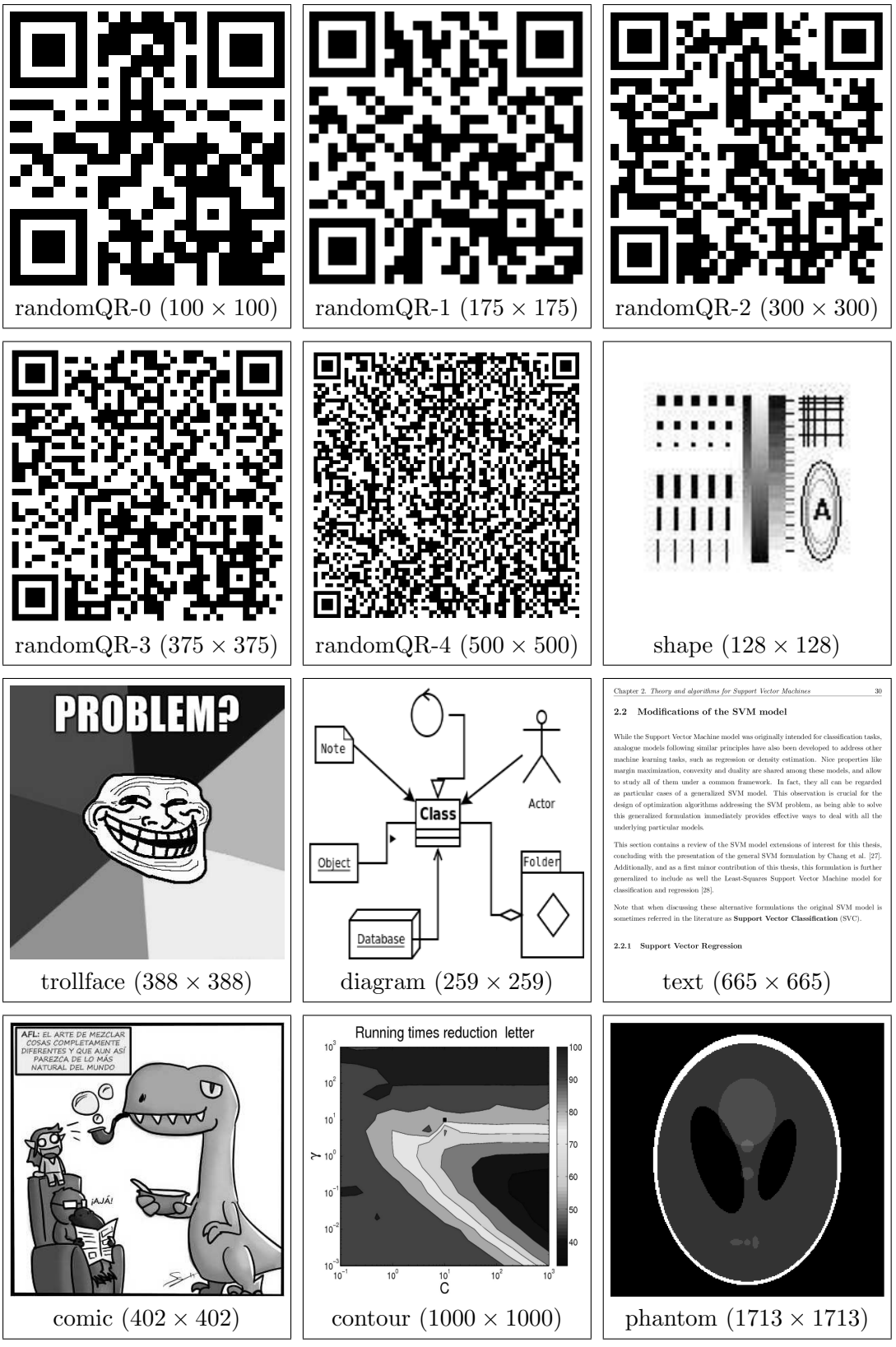
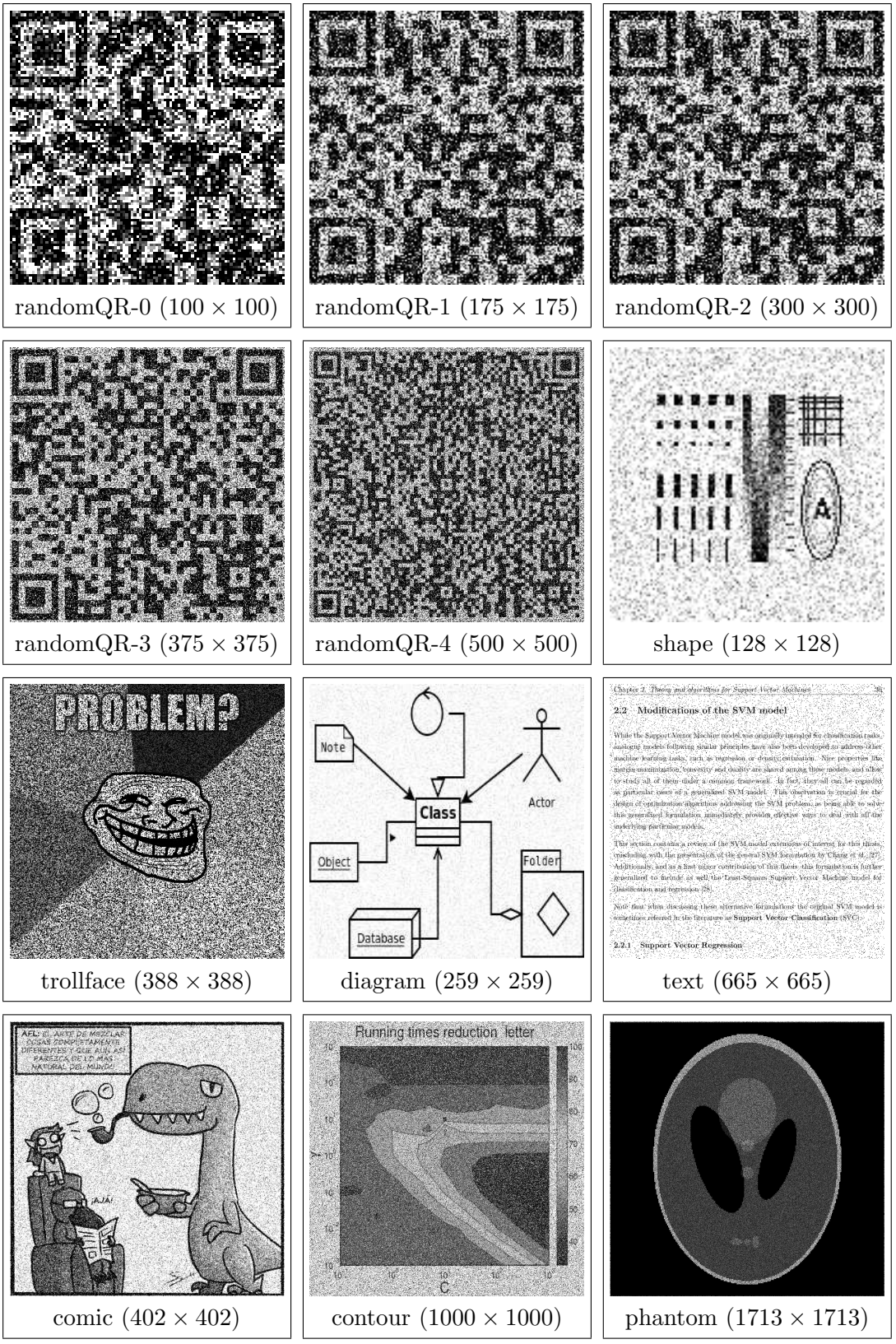


FIGURE 4.7: Test images used in the experiments together with their sizes in pixels. Images displayed have been scaled down to fit in page.



randomQR-0 (100 × 100) | randomQR-1 (175 × 175) | randomQR-2 (300 × 300)

randomQR-3 (375 × 375) | randomQR-4 (500 × 500) | shape (128 × 128)

trollface (388 × 388) | diagram (259 × 259) | text (665 × 665)

comic (402 × 402) | contour (1000 × 1000) | phantom (1713 × 1713)

FIGURE 4.8: Noisy versions of images used in the experiments.

FIGURE 4.9: Filtering results for the test images (1 out of 3).

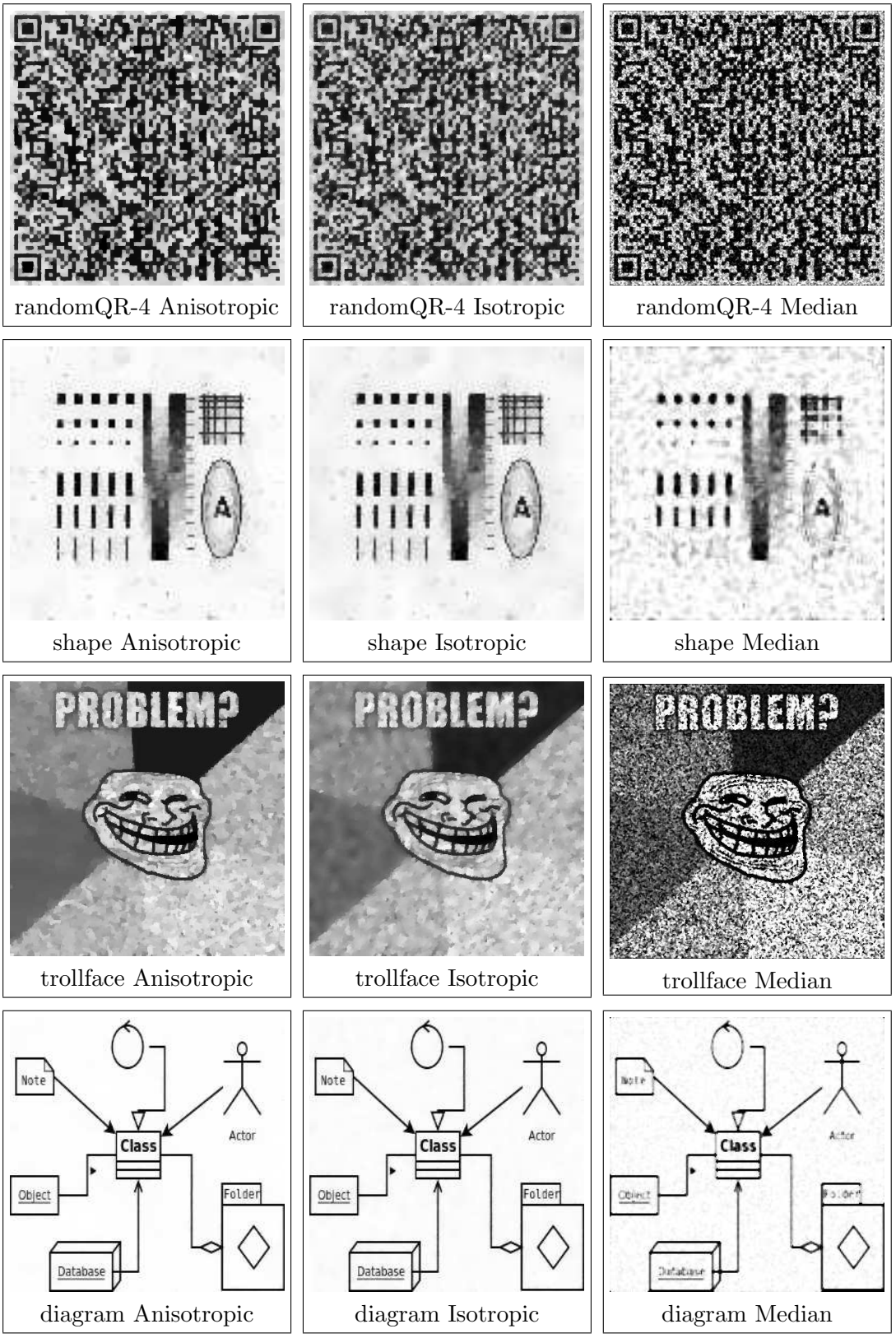


FIGURE 4.10: Filtering results for the test images (2 out of 3).

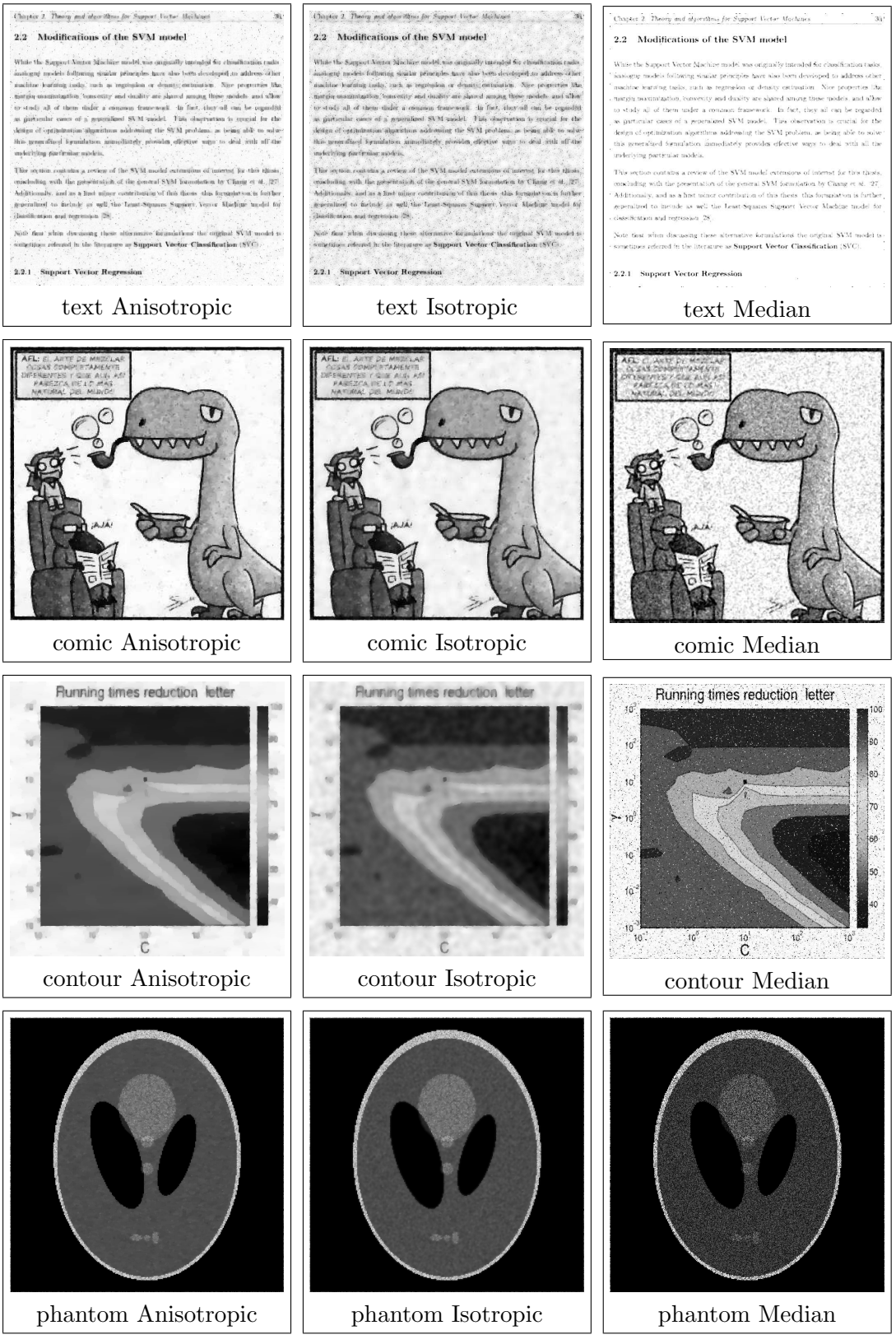
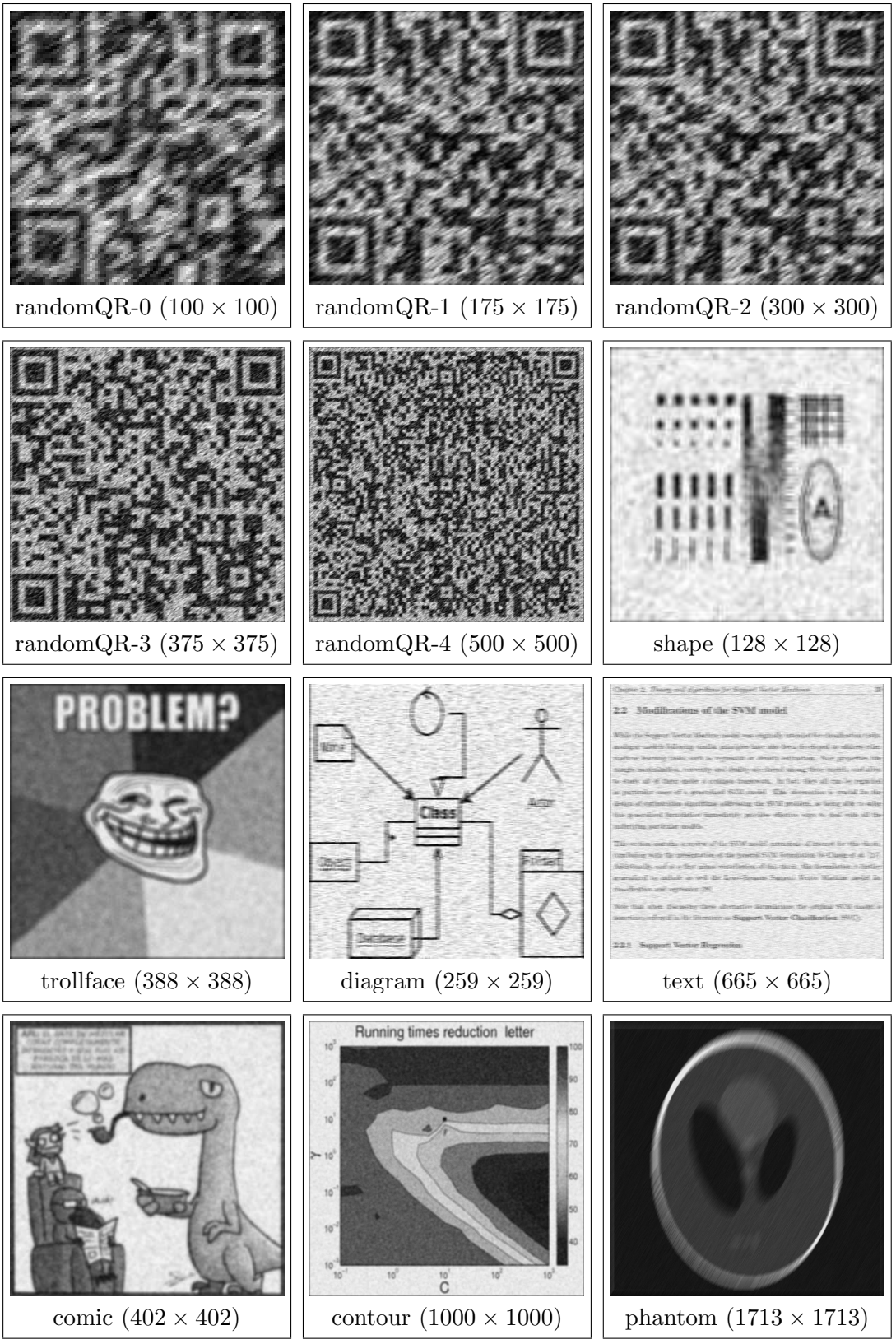


FIGURE 4.11: Filtering results for the test images (3 out of 3).



randomQR-0 (100 × 100) randomQR-1 (175 × 175) randomQR-2 (300 × 300)

randomQR-3 (375 × 375) randomQR-4 (500 × 500) shape (128 × 128)

trollface (388 × 388) diagram (259 × 259) text (665 × 665)

comic (402 × 402) contour (1000 × 1000) phantom (1713 × 1713)

FIGURE 4.12: Noisy and convoluted versions of images used in the experiments.

FIGURE 4.13: Deconvolution results for the test images (1 out of 3).

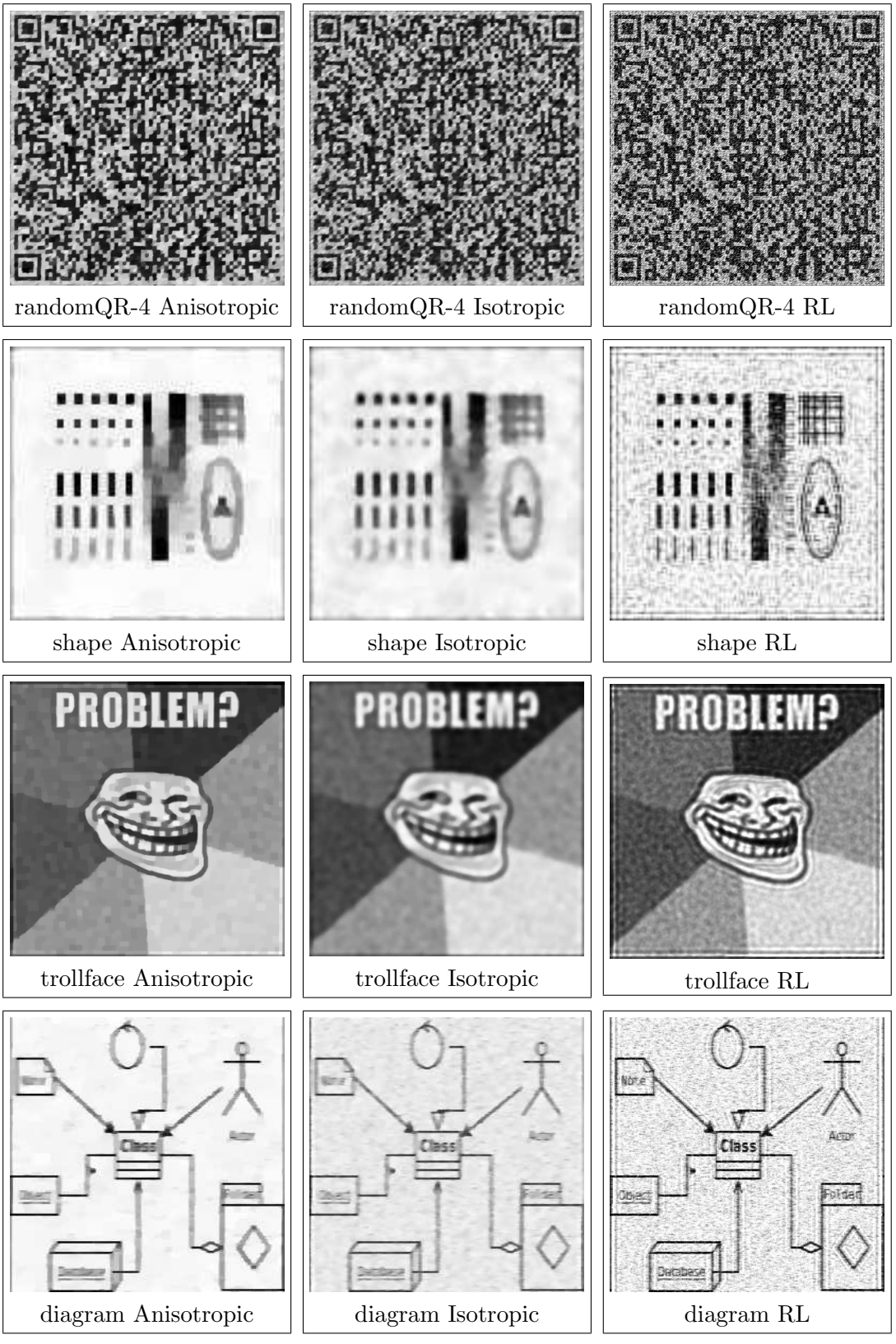


FIGURE 4.14: Deconvolution results for the test images (2 out of 3).

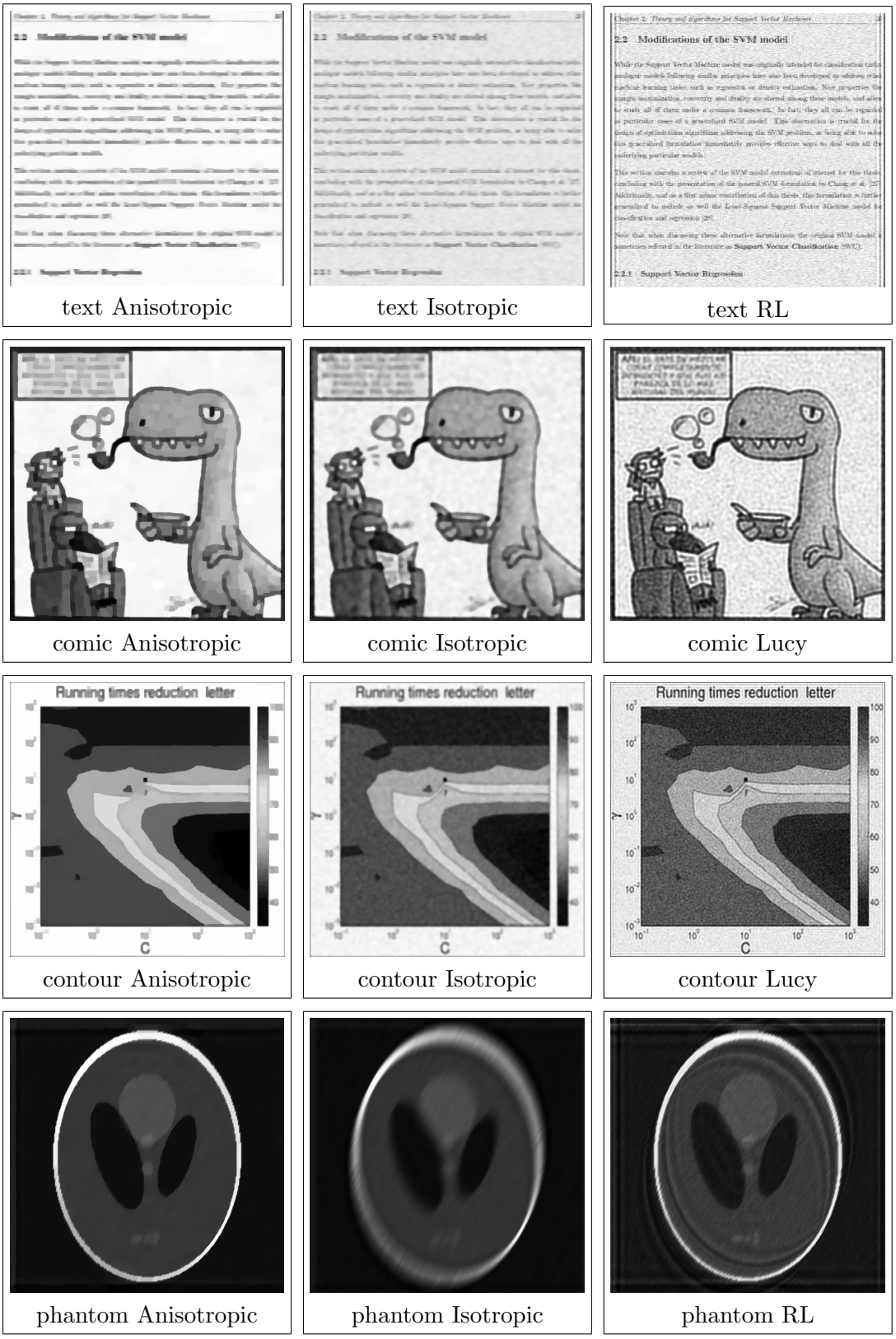


FIGURE 4.15: Deconvolution results for the test images (3 out of 3).

FIGURE 4.16: A selection of frames from the *salesman* video sequence.

FIGURE 4.17: Noisy frames from the *salesman* video sequence.

FIGURE 4.18: Denoised frames from the *salesman* video sequence using an anisotropic filter.

Chapter 5

Conclusions

"Walking with the wind blowing on your face.
Walking forwards the future.
Busy days with no compass to guide you."
Type–Moon

5.1 Conclusions and discussion

Regularized learning presents itself as a flexible framework to design models for machine learning tasks, featuring good properties such as guarantees on generalization error through complexity control and the ability to induce a variety of useful structures into the model parameters. Still, once a model within this framework has been chosen to address a particular task, an optimization algorithm is required to train such model on the available data. While general-purpose solvers are able to tackle a broad range of such problems, specifically designed solvers are bound to provide better results in terms of efficiency of the training procedure.

In this thesis an emphasis has been placed on the fact that a careful study of the optimization problem at hand and its structure, followed by the design or adaption of appropriate optimization methods, is essential to produce high-quality solvers. This is specially relevant in large-scale data settings where inefficient solvers are impractical, or in those application contexts where time is a valuable resource. As particular instances of these general ideas, the following results have been obtained

- The introduction of the Cycle–Breaking (CB) and Momentum Sequential Minimal Optimization (MSMO) methods for non–linear SVM training, which improve over

the state–of–the–art SMO method by making use of more informative updating directions, thus reducing the number of iterations required for convergence without significantly increasing the computational cost per iteration.

- Experimental results showing how MSMO is a more efficient method than CB, and when the reductions in iterations attained by MSMO result in reduced training times for a variety of settings and combinations with other accelerating strategies.
- A general dual SVM formulation extending the one presented by Chang et al. [30] to include the Least Squares Support Vector Machine model as a particular case, and thus allowing the application of the presented methods. Experimental results on this have been presented as well.
- Efficient algorithms to solve the Total–Variation proximity operator in its norms $p = 1$ and $p = 2$ variants, performing better than state–of–the–art solvers.
- Modular and parallelizable algorithms to solve the 2–dimensional and a general N–dimensional Total–Variation proximity operators building over the introduced 1–dimensional (standard) Total–Variation proximity solvers.
- Efficient methods to solve Fused–Lasso and anisotropic image denoising and deconvolution problems by employing proximal methods also making use of the developed proximity solvers.
- Experimental results showing the benefits of all the developed methods for Total–Variation.
- Public software implementations of the presented MSMO algorithm and the 1–dimensional, 2–dimensional and N–dimensional Total–Variation proximity solvers..

Of course, this idea of precise, almost crafted solver implementations is more a principle for good practice than an actual, definite methodology. Nevertheless, some general recommendations can be inferred from the particular results obtained here. A thorough study of the properties and structure of the problem as well as of already developed algorithms for the same or similar problems is mandatory. The combination of classic, well–established ideas with recently developed or even new approaches seems also to be always advisable, especially when building on already existent methods. When possible, hybrid strategies pooling the advantages of several different approaches while avoiding their drawbacks are also a desirable objective. And finally, exhaustive experimental testing of the new proposed methods is a must, not only to assess their performance in comparison to other approaches, but also to detect and understand their limitations and, hopefully, to be able to mend them. Reproducibility in the form of available software

implementing the new methods should also be implicit in those experimental tests, for future ease of comparison and, what is more, for more immediate impact and practical utility.

5.2 Further work

This thesis, as every work in science, is just a mere step in the research areas addressed by it. While new ideas have been proposed and advances have been made, also new questions and hypotheses have arisen. Therefore, as a closing of this thesis, some possible ways of further extending this work are proposed:

- The effectiveness of the Cycle-Breaking and, above that, of the Momentum Sequential Minimal Optimization algorithms indicates that the basic updating directions issued by Sequential Minimal Optimization can indeed be improved by the use of more informative directions. This hints that even better ways of constructing updating the directions might be possible, for instance by introducing even more second–order information with Conjugate-Gradient-like techniques. Even recent methods using just first–order information have been shown to be able to produce much better results than classic gradient optimization approaches [85]. Still, it is not straightforward to adapt such methods to the dual SVM problem, where sparsity in the updating directions is a must. Nevertheless, exploring these ideas could turn out to be very profitable.

- In parallel to this, it seems that efforts to reduce the number of iterations for convergence in SMO-like settings may not have a great impact on performance when combined with caching and shrinking strategies, as these reduce the cost per iteration precisely on those iterates more likely to be saved. It seems therefore also reasonable to analyze approaches more directly focused on saving kernel function evaluations, which are not only the most costly operations in non-linear SVM algorithms, but also the source of this variability in iteration costs.

- Extending the presented general SVM framework to include other models such as Eν-SVM, or to address explicitly the Kernel Fisher Discriminant Analysis seems also to be an interesting approach to provide reliable and efficient solvers for these models. While this should be easy for KFD, it is unknown whether it is even possible for Eν-SVM.

- Regarding the Total–Variation regularizer, more applications of the presented solvers should be studied. The particular structure imposed by this regularizer

might be of use in other areas apart from the ones addressed here, and its application to N–dimensional contexts might be of special relevance.

- It would also be of interest to design methods for other Total–Variation norms different from $p = 1, 2$, such as $p = \infty$, or even $p = \frac{1}{2}$ to induce further sparsity. While the first case might be more approachable, the second results into a challenging non-convex model.

- Going further, the general model $f(x) + g(Bx)$ could be considered. This results in the Total–Variation regularization when B is the differencing matrix D (Section 4.1.1) and g is an L_p norm, but other choices of B and g can lead to different models such as Group Lasso, Group Lasso with overlapping groups, or even multi–task learning models [135]. The framework presented here could be applied in those settings as long as a proximity operator is developed for the $g(Bx)$ part. It will be hard, though, to design efficient solvers for complex g or non–sparse matrices B, since in the TV–proximity solvers here it has been made extensive use of the nice structural properties of $g = \mathrm{L}_p$ and $B = D$.

- The proposed 2–dimensional and, in general, N–dimensional Total–Variation proximity solvers, are especially suitable to high parallelization when working with large data volumes. As such, a GPU implementation of these methods presents itself as an interesting way to take advantage of this potentially massive multi-thread computational power. Whether this is feasible in practice still needs to be studied.

- Conversely, these solvers might not be the best option in a single-processor system, as a hypothetical method addressing directly the 2–dimensional or N–dimensional problem instead of making use of multiple 1–dimensional solvers might turn out to be more efficient. However, the structure of these problems is more complex than the one presented by their 1–dimensional counterpart, and as such adapting the presented 1–dimensional algorithms for them is a daunting challenge.

5.3 Conclusiones

El aprendizaje regularizado se presenta como una metodología flexible para diseñar modelos para tareas de aprendizaje automático, teniendo propiedades deseables como garantías sobre el error de generalización a través de un control de la complejidad y la capacidad de inducir propiedades estructurales útiles en los parámetros del modelo. No obstante, una vez que se ha seleccionado un modelo, se requiere el uso de un método de optimización para realizar su entrenamiento con los datos disponibles. Aunque existen métodos de optimización genéricos que pueden resolver esta tarea, el diseño de métodos

específicamente adaptados para el problema proporciona mejores resultados en términos de eficiencia del proceso de entrenamiento.

En esta tesis se ha hecho énfasis en el hecho de que un estudio cuidadoso del problema de optimización ligado al modelo y de su estructura, junto con el diseño o adaptación de métodos de optimización apropiados, es esencial para producir métodos de buena calidad. Esto es especialmente relevante en aquellos contextos donde se manejan grandes volúmenes de datos, puesto que todo método ineficiente no resulta de utilidad práctica, o en aquellas aplicaciones donde el tiempo disponible para el proceso de entrenamiento sea limitado. Como casos particulares de estas ideas generales, se han obtenido aquí los siguientes resultados

- La creación de los métodos Cycle–Breaking (CB) y Momentum Sequential Minimal Optimization (MSMO) para el entrenamiento de Máquinas de Vectores de Soporte (SVM) no lineales, los cuales mejoran sobre el algoritmo SMO (estado del arte) mediante el uso de direcciones de avance más informativas, obteniendo así reducciones en el número de iteraciones necesarias para la convergencia sin incrementar significativamente el coste por iteración.
- Resultados experimentales demostrando cómo MSMO es un método superior a CB, y cuándo las reducciones en iteraciones obtenidas por MSMO se traducen en tiempos de entrenamiento reducidos dependiendo de la situación y de la combinación de este método con otras estrategias de aceleración.
- Una formulación general de la SVM dual que extiende la presentada por Chang et al. [30], y que incluye el modelo Least Squares Support Vector Machine como un caso particular, permitiendo así el uso de los métodos desarrollados. También se presentaron resultados experimentales en esta línea.
- Algoritmos eficientes para resolver el operador de proximidad de Variación Total en sus versiones con normas $p = 1$ y $p = 2$, mejorando sobre métodos del estado del arte.
- Algoritmos modulares y paralelizables para resolver las versiones bidimensionales y multidimensionales del operador de proximidad de Variación Total, y que se basan en los métodos desarrollados para el caso unidimensional (estándar).
- Métodos eficientes para resolver el modelo del Lasso Fundido, así como problemas de filtrado y deconvolución anisotrópica de imágenes, mediante el uso de algoritmos proximales que también se basan en los operadores de proximidad desarrollados.
- Resultados experimentales demostrando los beneficios de todos los métodos para Variación Total presentados.

- Implementaciones software públicas del algoritmo MSMO y de los operadores de proximidad para Variación Total desarrollados.

Por supuesto, esta idea de implementaciones precisas y casi artesanales de algoritmos de optimización para el modelo en cuestión es más una filosofía a seguir que una metodología concreta. No obstante, pueden extraerse algunas recomendaciones generales de los resultados obtenidos aquí. Un estudio detallado de las propiedades y estructura del problema a resolver, así como de métodos ya existentes que resuelvan el mismo problema o uno similar, resulta ser un paso ineludible en esta tarea. El combinar métodos e ideas clásicas y establecidas con nuevas aproximaciones también parece ser siempre algo recomendable. Cuando así sea posible, el emplear estrategias híbridas aunando las ventajas de distintos métodos y evitando sus desventajas también es un objetivo deseable. Y finalmente, probar extensivamente los nuevos métodos desarrollados mediante experimentos es completamente necesario, no sólo para comparar su rendimiento con el de otros métodos ya disponibles, sino también para detectar y entender sus limitaciones para que éstas puedan ser enmendadas. La reproducibilidad de tales experimentos, por ejemplo suministrando implementaciones públicas de los algoritmos desarrollados, también debe ser un componente esencial, para facilitar futuras comparaciones con otras aproximaciones así como para un mayor impacto y utilidad práctica de los resultados obtenidos.

Appendix A

Publications

The work realized during the development of this thesis gave rise to a number of publications in scientific journals and conference contributions. While some of them have been cited in the main text, here they are presented again together with a short summary of their contents. For completeness, and to offer a complete picture of all the lines of research explored during the PhD period, other publications obtained but not strictly related with the topics addressed in this thesis are presented as well.

Geometric SVMs

- Using the relationships between SVMs, their geometric counterparts and Support Vector Regression, in [136] a framework to cast a regression problem into a geometric SVM one is presented. While these connections were already known in the literature, an alternative proof is given for them, and a relationship is established between the weights of the support vectors of the model and their position with respect to the ϵ-tube.

- In geometric SVM methods usually only the weights representative of one of the two classes are updated per iteration, since the constraint $y_u = y_l$ enforces choosing both components of the working set in the same class. In [137] a method for a MDM–like update of a working set of 4 elements – 2 from each class – is proposed, showing some improvements over the standard MDM algorithm.

- The important relationship between the MDM and the SMO algorithms was established in [70], showing how MDM coincides with SMO when the additional constraint $y_u = y_l$ is introduced. Furthermore, a new and efficient "clipped" MDM algorithm solving the Reduced Convex Hulls problem is proposed.

- This last algorithm is adapted as well to the GSK algorithm in [71], and compared with other geometric approaches to geometric SVMs in the literature. An adaption to the problem of Scaled Convex Hulls was developed as well [49].
- Finally, in [138] a review of methods for the Reduced Convex Hulls problem is presented, placing in context the contributions above, and showing how the clipped MDM algorithm performs significantly better than other methods of choice for such problem.

Cycle–Breaking and Momentum SMO

- The Cycle–Breaking algorithm was first proposed in [70] for the geometric MDM algorithm, showing how cycles appear during the MDM procedure, and how Cycle–Breaking takes advantage of this.
- After that, Cycle–Breaking was applied to the SMO algorithm, first in a WSS1 setting and later on in WSS2 [74, 75]. Some first intuitions are given about the variability in the behavior of Cycle–Breaking for different datasets and parameters.
- In [139] an alternative point of view to Support Vector Regression is provided, which is used later on in [76] to apply the Cycle–Breaking technique to SVR as well.
- The Momentum Sequential Minimal Optimization algorithm was proposed in [140], and applied over the SMO algorithm as implemented in the LIBSVM library. Some of the experimental results presented in this thesis for MSMO were already included in this work.
- A preliminary application of this method to the LS–SVM model was also presented in another contribution in [141].

Total–Variation

- A preliminary work on this topic was presented in the form of a Max Planck Institute technical report [142], already including the proposals for 1–dimensional Total–Variation as presented here, which are put in comparison with other approaches to the problem.
- Building over that report, a contribution with 2–dimensional and multi–dimensional Total–Variation methods, together with Fused–Lasso and image denoising and deconvolution applications was presented at ICML 2011 [143].

Other lines of research

- Finding optimal parameter choices for a Support Vector Machine or a Multilayer Perceptron Model is a mandatory step in almost any practical application of these models, where fine–tuning of these parameters is required to guarantee good performance. In this line, a series of methods to optimize these parameters were developed, based on a refined grid search procedure [131, 132].

- The application of Support Vector Machines and other models for the practical problem of large–scale wind power forecasting was tested in [144, 145], showing better results than the standard Multilayer Perceptron approaches generally used in this setting.

- As an outcome of collaborations with "Machine Learning in Neuroscience" group of the Max Planck Institute for Biological Cybernetics, an analysis of the application of implicit Wiener series methods for epileptic seizures recordings was performed [146], together with an study of the influence of a biased visual feedback in motor–imagery Brain–Computer interfaces [147].

- In a completely different line and a result from a side project in multi–agent computing simulation, a system was developed which emulated the behavior of a group of students interacting in collaborative web application [148].

Bibliography

[1] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[2] A. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950. URL `http://links.jstor.org/sici?sici=0026-4423%28195010%292%3A59%3A236%3C433%3ACMAI%3E2.0.CO%3B2-5`.

[3] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, 2000.

[4] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[5] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. Adaptive Computation and Machine Learning. The MIT Press, 2010.

[6] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, Seattle, WA, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014067. URL `http://portal.acm.org/citation.cfm?id=1014067&dl=GUIDE&coll=GUIDE&CFID=69344422&CFTOKEN=94303924`.

[7] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979).

[8] Vladimir N. Vapnik. Estimation of dependences based on empirical data (in russian). 1979.

[9] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[10] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. Machine Learning. MIT Press, 2002.

[11] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2, 2002.

[12] Robert Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Statist. Soc.*, 58(1):267–288, 1996.

[13] Jorge J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM J. Sci. Stat. Comput.*, 4(3), September 1983.

[14] Jason Weston, André Elisseeff, and Bernhard Schölkopf. Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3:1439–1461, 2003.

[15] Lukas Meier, Sara van de Geer, and Peter Bühlmann. The group lasso for logistic regression. *J. R. Statist. Soc.*, 70:53–71, 2008.

[16] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *J. R. Statist. Soc. B*, 67(2):301–320, 2005.

[17] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108(18), 2005.

[18] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Statistics. Springer, 2001.

[19] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958.

[20] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. M.I.T. Press, 1969.

[21] Alexander J. Smola. *Advances in Large Margin Classifiers*. MIT Press, 2000.

[22] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1997.

[23] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[24] Maurice Sion. On general minimization theorems. *Pacific Journal of Mathematics*, 8(1):171–176, 1958.

[25] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, September 1999.

[26] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A(209):415–446, 1909.

[27] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37(5), January 2008.

[28] Kai Yu, Liang Yi, and Xuegong Zhang. Kernel nearest-neighbor algorithm. *Neural Processing Letters*, 15:147–156, 2002.

[29] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *ICANN 97 proceedings*, 1997.

[30] C. C. Chang and C. J. Lin. *LIBSVM: a Library for Support Vector Machines*, 2001. URL `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[31] J. A. K. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999. ISSN 1370-4621. doi: http://dx.doi.org/10.1023/A:1018628609742.

[32] S. K. Shevade, S. S. Keerthi, and K. R. K. Murthy. Improvements to the smo algorithm for svm regression. *IEEE transactions on neural networks*, 11(5):1188–1193, September 2000.

[33] Alex J. Smola and Bernhard Schölkopf. A Tutorial on Support Vector Regression. *Statistics and Computing*, 48:199–222, 2003.

[34] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. Technical report, Microsoft Research, November 1999.

[35] G. Rätsch. *Benchmark Repository*, 2000. Datasets available at `http://ftp.tuebingen.mpg.de/pub/fml/raetsch-lab/benchmarks/`.

[36] K. De Brabanter, P. Karsmakers, F. Ojeda, C. Alzate, J. De Brabanter, K. Pelckmans, B. De Moor, J. Vandewalle, and J.A.K. Suykens. LS-SVMlab Toolbox User's Guide version 1.7. Technical Report 10-146, Katholieke Universiteit Leuven, 2010. Software available at `http://www.esat.kuleuven.be/sista/lssvmlab`.

[37] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO Algorithm for SVM Classifier Design. *Neural Computation*, 13(3):637–649, 2001. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/089976601300014493.

[38] J. López and J.A.K. Suykens. First and Second Order SMO Algorithms for LS-SVM classifiers. *Neural Processing Letters*, 33(1):33–44, 2011.

[39] Tony Van Gestel, Johan A. K. Suykens, Bart Baesens, Stijn Viaene, Jan Vanthienen, Guido Dedene, Bart De Moor, and Joos Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learning*, 54 (1):5–32, 2004. URL `http://dblp.uni-trier.de/db/journals/ml/ml54.html#GestelSBVVDMV04`.

[40] S. S. Keerthi and S. K. Shevade. SMO Algorithm for Least-Squares SVM Formulations. *Neural Computation*, 15(2):487–507, 2003. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/089976603762553013.

[41] Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller. Fisher discriminant analysis with kernels. *Neural Networks*, 2002.

[42] Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

[43] D.J. Crisp and C.J.C. Burges. A geometric interpretation of ν-svm classifiers. *Adv. Neural Inform. Process. Syst. (NIPS)*, 12:244–250, 1999.

[44] Fernando Perez-Cruz, Jason Weston, Daniel Herrmann, and Bernhard Schölkopf. *Extension of the ν-SVM Range for Classification*, pages 179–196. Advances in Learning Theory: Methods, Models and Applications Nato Science. IOS Press, 2003.

[45] Akiko Takeda and Masashi Sugiyama. ν-Support Vector Machine as Conditional Value-at-Risk Minimization. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[46] K.P. Bennett and E.J. Bredensteiner. Duality and geometry in svm classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 57–64, 2000.

[47] J. Bi and K.P. Bennett. A Geometric Approach to Support Vector Regression. *Neurocomputing*, 55:79–108, 2003.

[48] Z. Liu, J.G. Liu, C. Pan, and G. Wang. A Novel Geometric Approach to Binary Classification Based on Scaled Convex Hulls. *IEEE Transactions on Neural Networks*, 20(7):1215–1220, July 2009.

[49] J. López, A. Barbero, and J.R. Dorronsoro. An mdm solver for the nearest point problem in scaled convex hulls. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8, 2010. doi: 10.1109/IJCNN.2010.5596984.

[50] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com/cvx`, February 2011.

[51] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. 2007. URL `http://ttic.uchicago.edu/~shai/papers/ShalevSiSr07.pdf`. A fast online algorithm for solving the linear svm in primal using sub-gradients.

[52] Vojtech Franc and Sören Sonnenburg. Optimized cutting plane algorithm for support vector machines. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 320–327. ACM, 2008. ISBN 978-1-60558-205-4. URL `http://dblp.uni-trier.de/db/conf/icml/icml2008.html#FrancS08`.

[53] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the Conference on Knowledge Discovery and Data Mining*, 2006.

[54] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 408–415. ACM, 2008. ISBN 978-1-60558-205-4. URL `http://dblp.uni-trier.de/db/conf/icml/icml2008.html#HsiehCLKS08`.

[55] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://dblp.uni-trier.de/db/journals/jmlr/jmlr9.html#FanCHWL08`.

[56] Stephen J. Wright. *Primal-Dual Interior Point Methods*. SIAM, 1997.

[57] E. Osuna, R. Freund, and F. Girosi. An Improved Training Algorithm for Support Vector Machines. *Proc. 1997 IEEE Workshop*, pages 276–285, 1997.

[58] Thorsten Joachims. Making Large-Scale Support Vector Machine Learning Practical. In *Advances in Kernel Methods: Support Vector Learning*, pages 169–184, Cambridge, MA, USA, 1999. MIT Press. ISBN 0-262-19416-3.

[59] G. Zoutendijk. *Methods of Feasible Directions: a Study in Linear and Non-linear Programming*. Elsevier, 1970.

[60] J. C. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods: Support Vector Learning*, pages 185–208, Cambridge, MA, USA, 1999. MIT Press. ISBN 0-262-19416-3.

[61] C.-J. Lin. Asymptotic Convergence of an SMO Algorithm without any Assumptions. *IEEE Transactions on Neural Networks*, 13(1):248–250, January 2002.

[62] C.-J. Lin. On the Convergence of the Decomposition Method for Support Vector Machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, November 2001.

[63] R. E. Fan, P. H. Chen, and C. J. Lin. Working Set Selection using Second Order Information for Training Support Vector Machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005. ISSN 1533-7928.

[64] P-H. Chen, R-E. Fan, and C-J. Lin. A Study on SMO-type Decomposition Methods for Support Vector Machines. *IEEE Transactions on Neural Networks*, 17: 893–908, 2006.

[65] V. Franc and V. Hlavăc. An Iterative Algorithm Learning the Maximal Margin Classifier. *Pattern Recognition*, 36:1985–1996, 2003.

[66] E.G. Gilbert. Minimizing the quadratic form on a convex set. *SIAM J. Contr.*, 4: 61–79, 1966.

[67] M.E. Mavroforakis and S. Theodoridis. A Geometric Approach to Support Vector Machine (SVM) Classification. *IEEE Transactions on Neural Networks*, 17(3): 671–682, May 2006.

[68] B.F. Mitchell, V.F. Dem'yanov, and V.N. Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM J. Contr.*, 12:19–26, 1974.

[69] Q. Tao, G. W. Wu, and J. Wang. A General Soft Method for Learning SVM Classifiers with L1-Norm Penalty. *Pattern Recognition*, 41:939–948, 2008.

[70] J. López, Á. Barbero, and J. R. Dorronsoro. On the Equivalence of the SMO and MDM Algorithms for SVM Training. In *Lecture Notes in Artificial Intelligence: Machine Learning and Knowledge Discovery in Databases - ECML 2008 Proceedings, Part II*. Springer, 2008.

[71] J. López, Á. Barbero, and J. R. Dorronsoro. Simple Clipping Algorithms for Reduced Convex Hull SVM Training. In *Lecture Notes in Computer Science: Hybrid Artificial Intelligent Systems - HAIS 2008*, pages 369–377, 2008.

[72] Magnus R. Hestenes and Eduard Stiefel. Method of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6): 409–436, December 1952.

[73] Á. Barbero, J. López, and J. R. Dorronsoro. An Accelerated MDM Algorithm for SVM Training. In *Proceedings of the 16th European Symposium on Artificial Neural Networks*, pages 421–426, 2008.

[74] Á. Barbero, J. López, and J. R. Dorronsoro. Cycle-breaking Acceleration of SVM Training. *Neurocomputing*, 72(7-9):1398–1406, 2009. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2008.12.014.

[75] Á. Barbero and J. R. Dorronsoro. Faster directions for second order smo. In *Lecture Notes in Computer Science*, volume 6353, pages 30–39, Berlin, Heidelberg, 2010. Springer-Verlag.

[76] Álvaro Barbero and José R. Dorronsoro. Cycle-breaking acceleration for support vector regression. To appear in Neurocomputing, 2011.

[77] M. Bazaraa, D. Sherali, and C. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, 1992.

[78] A. Beck and M. Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM J. Imgaging Sciences*, 2(1):183–202, 2009.

[79] José M. Bioucas-Dias and Mário A. T. Figueiredo. A new twist: Two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Transactions on Image Processing*, 16(12):2992–3004, December 2007.

[80] Patrick L. Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. *arXiv*, 2009.

[81] Amit Bhaya and Eugenius Kaszkurewicz. Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method. *Neural Networks*, 17(1), July 2003.

[82] F. Chang, C.-Y. Guo, X.-R. Lin, and C.-J. Lu. Tree decomposition for large-scale svm problems. *Journal of Machine Learning Research*, 11:2935–2972, Oct. 2010. URL `http://ocrwks11.iis.sinica.edu.tw/~dar/Download/DataSets/DTSVM/datasets.htm`.

[83] Chih-Jen Lin. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 13(5):1045–1052, September 2002.

[84] S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation*, 15(7):1667–1689, March 2006.

[85] Y. Nesterov. Gradient methods for minimizing composite objective function. Technical Report 76, Catholic University of Louvain, CORE, 2007.

[86] J. Mairal, R. Jenatton, G. Obozinski, and F. Bach. Network Flow Algorithms for Structured Sparsity. In *NIPS*, 2010.

[87] Manya V. Afonso, José M. Bioucas-Dias, and Mário A. T. Figueiredo. Fast image recovery using variable splitting and constrained optimization. *IEEE Transactions on Image Processing*, 19(9), September 2010.

[88] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Trans. Sig. Proc.*, 57(7):2479–2493, 2009.

[89] Dongmin Kim, Suvrit Sra, and Inderjit Dhillon. A scalable trust-region algorithm with application to mixed-norm regression. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[90] J. Duchi and Y. Singer. Online and Batch Learning using Forward-Backward Splitting. *JMLR*, Sep. 2009.

[91] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling and Simulation*, 4(4):1168–1200, 2005.

[92] D. Donoho. Denoising by soft-thresholding. *IEEE Tran. Inf. Theory*, 41(3):613–627, 2002.

[93] Mingqiang Zhu and Tony Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. Technical report, UCLA CAM, 2008.

[94] Chih-Jen Lin and Jorge J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM J. Optim.*, 9(4):1100–1127, 1999.

[95] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. Technical report, Northwestern University, 1994.

[96] Dimitri P. Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM J. Control and Optimization*, 20(2), March 1982.

[97] Gene H. Golub and Gerard Meurant. *Matrices, Moments and Quadrature with Applications*. Princeton University Press, 2010.

[98] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

[99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Verlag, 2000.

[100] C. R. Vogel and M. E. Oman. Iterative methods for total variation denoising. *SIAM Journal on Scientific Computing*, 1996.

[101] Curtis R. Vogel and Mary E. Oman. Fast, robust total variation-based reconstruction of noisy, blurred images. *IEEE Transactions on Image Processing*, 7(6), 1998.

[102] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. SIAM, 2000.

[103] J. Liu, L. Yuan, and J. Ye. An efficient algorithm for a class of fused lasso problems. In *SIGKDD*, 2010.

[104] Dianne P. O'Leary. *Scientific computing with case studies*. SIAM, 2009.

[105] J. Liu, S. Ji, and J. Ye. *SLEP: Sparse Learning with Efficient Projections*. Arizona State University, 2009. *http://www.public.asu.edu/ jye02/Software/SLEP*.

[106] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, August 2007.

[107] R. Tibshirani and P. Wang. Spatial smoothing and hot spot detection for CGH data using the fused lasso. *Biostatistics*, 9(1):18–29, 2008.

[108] F. Rapaport and E. Barillot J.-P. Vert. Classification of arrayCGH data using fused SVM. *Bioinformatics*, 24(13):i375–i382, 2008.

[109] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8:141–148, 1988.

[110] A. Rinaldo. Properties and refinements of the fused lasso. *Annals of Statistics*, 37 (5B):2922–2952, 2009.

[111] S. R. Land and J. H. Friedman. Variable fusion: A new adaptive signal regression method. Technical Report 656, Department of Statistics, Carnegie Mellon University Pittsburgh, 1997.

[112] M. Kolar, L. Song, A. Ahmed, and E. Xing. Estimaging time-varying networks. *The Annals of Applied Statistics*, 4(1):94–123, 2010.

[113] H. H. Bauschke and P. L. Combettes. A dykstra-like algorithm for two monotone operators. *Pacific. J. Optim.*, 4:381–391, 2008.

[114] Nicolas Stransky et al. Regional copy number-independent deregulation of transcription in cancer. *Nature Genetics*, 38(12):1386–1396, December 2006.

[115] T. R. Golub et al. Molecular classification of cancer. *Science*, 286(5439):531–537, October 1999.

[116] U. Alon et al. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. Natl. Acad. Sci. USA*, 96:6745–6750, June 1999.

[117] Simon Rogers, Mark Girolami, Colin Campbell, and Rainer Breitling. The latent process decomposition of cdna microarray data sets. *IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS*, 2(2), April-June 2005.

[118] Jianping Hua, Waibhav D. Tembe, and Edward R. Dougherty. Performance of feature-selection methods in the classification of high-dimension data. *Pattern Recognition*, 42:409–424, 2009.

[119] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[120] José M. Bioucas-Dias, Mário A. T. Figueiredo, and Joao P. Oliveira. Total variation-based image deconvolution: A majorization-minimization approach. In *ICASSP Proceedings*, 2006.

[121] Rustum Choksi, Yves van Gennip, and Adam Oberman. Anisotropic total variation regularized l1-approximation and denoising/deblurring of 2d bar codes. Technical report, Department of Mathematics and Statistics, McGill University, July 2010.

[122] Yuying Li and Fadil Santosa. A computational algorithm for minimizing total variation in image restoration. *IEEE Transactions on Image Processing*, 5(6):987–995, 1996. URL `http://dblp.uni-trier.de/db/journals/tip/tip5.html#LiS96`.

[123] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-laplacian priors. In *Advances in Neural Information Processing Systems*, 2009.

[124] Gonzalo R. Arce. *Nonlinear Signal Processing: A Statistical Approach.* Wiley, 2004.

[125] Google chart api. URL `http://code.google.com/intl/en-EN/apis/chart/`.

[126] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3d transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8), August 2007.

[127] Joachim Dahl, Per Christian Hansen, Søren Holdt Jensen, and Tobias Lindstrøm Jensen. Algorithms and software for total variation image reconstruction via first-order methods. *Numer Algor*, (53):67–92, 2010.

[128] Michael K. Ng., Huanfeng Shen, Edmund Y. Lam, and Liangpei Zhang. A total variation regularization based super-resolution reconstruction algorithm for digital video. *EURASIP Journal on Advances in Signal Processing*, 2007.

[129] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20:89–97, 2004.

[130] David S. C. Biggs and Mark Andrews. Acceleration of iterative image restoration algorithms. *Applied Optics*, 36(8), March 1997.

[131] Á. Barbero, J. López, and J. R. Dorronsoro. Finding Optimal Model Parameters by Discrete Grid Search. In *Advances in Soft Computing: Innovations in Hybrid Intelligent Systems 44*, pages 120–127. Springer, 2008.

[132] Á. Barbero, J. López, and J. R. Dorronsoro. Finding Optimal Model Parameters by Deterministic and Annealed Focused Grid Search. *Neurocomputing*, 72(13-15): 2824–2832, 2009. ISSN 0925-2312. doi: DOI:10.1016/j.neucom.2008.09.024.

[133] Patrick L. Combettes. Iterative construction of the resolvent of a sum of maximal monotone operators. *Journal of Convex Analysis*, 16:727–748, 2009.

[134] Bm3d software and test sequences. URL `http://www.cs.tut.fi/~foi/GCF-BM3D/`.

[135] Andreas Argyriou, Charles A. Micchelli, Massimiliano Pontil, Lixin Shen, and Yuesheng Xu. Efficient first order methods for linear composite regularizers. *CoRR*, abs/1104.1436, 2011. URL `http://dblp.uni-trier.de/db/journals/corr/corr1104.html#abs-1104-1436`. informal publication.

[136] Á. Barbero, J. López, and J. R. Dorronsoro. Square Penalty Support Vector Regression. In *Lecture Notes in Computer Science: Intelligent Data Engineering and Automated Learning - IDEAL 2007*, pages 537–546. Springer, 2007.

[137] Á. Barbero, J. López, and J. R. Dorronsoro. A 4–Vector MDM Algorithm for Support Vector Training. In *Lecture Notes in Computer Science: Artificial Neural Networks*, volume 5165, pages 315–324, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87535-2.

[138] Jorge López, Álvaro Barbero, and José R. Dorronsoro. Clipping algorithms for solving the nearest point problem over reduced convex hulls. *Pattern Recognition*, 44:607–614, 2011.

[139] Á. Barbero and J. R. Dorronsoro. A simple maximum gain algorithm for support vector regression. In *Lecture Notes in Computer Science, Bio-Inspired systems: Computational and Ambient Intelligence*, volume 5517, pages 73–80, 2009.

[140] Álvaro Barbero and José R. Dorronsoro. Momentum sequential minimal optimization: an accelerated method for support vector machine training. In *IJCNN proceedings (to appear)*, 2011.

[141] Jorge López, Álvaro Barbero, and José R. Dorronsoro. Momentum acceleration of least-squares support vector machines. In *ICANN proceedings*, 2011. (To appear).

[142] A Barbero Jimenez and S Sra. Fast algorithms for total-variation based optimization. Technical Report 194, Max Planck Institute for Biological Cybernetics, Tübingen, Germany, 8 2010. URL `http://www.kyb.tuebingen.mpg.de/fileadmin/user_upload/files/publications/MPIK-TR-194_[0].pdf`.

[143] Álvaro Barbero and Suvrit Sra. Title undisclosed to respect double-blind reviewing process. In *Submitted to ICML*, 2011.

[144] Á. Barbero, J. López, and J. R. Dorronsoro. Kernel Methods for Wide Area Wind Generation Forecasting. In *Proceedings of the 2008 European Wind Energy Conference (EWEC)*, 2008.

[145] Carlos Alaíz, Álvaro Barbero, Ángela Fernández, and José R. Dorronsoro. High wind and energy specific models for global production forecast. In *Proceedings of the 2009 European Wind Energy Conference (EWEC)*, 2009.

[146] Álvaro Barbero, Matthias Franz, Win van Drongelen, José R. Dorronsoro, Bernhard Schölkopf, and Moritz Grosse-Wentrup. Implicit wiener series analysis of epileptic seizure recordings. In *Proceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2009.

[147] Alvaro Barbero and Moritz Grosse-Wentrup. Biased feedback in brain-computer interfaces. *Journal of NeuroEngineering and Rehabilitation*, 7(1):34, 2010. ISSN 1743-0003. doi: 10.1186/1743-0003-7-34. URL `http://www.jneuroengrehab.com/content/7/1/34`.

[148] Álvaro Barbero, Mario Salvador González-Rodríguez, Juan de Lara, and Manuel Alfonseca. Multi-agent simulation of an educational collaborative web system. In *European Simulation and Modelling Conference 2007 proceedings*, 2007.