UNIVERSITÀ
DEGLI STUDI
FIRENZE

Università degli Studi di Firenze
Dipartimento di Ingegneria dell'Informazione (DINFO)
Corso di Dottorato in Ingegneria dell'Informazione

Curriculum: Automatica, Ottimizzazione e Sistemi Elettrici
SSD MAT/09

On the Interplay between Global Optimization and Machine Learning

Candidate
Francesco Bagattini

Supervisors
Prof. Fabio Schoen

Prof. Marco Sciandrone

PhD Coordinator
Prof. Luigi Chisci

---

ciclo XXX, 2014-2017

Università degli Studi di Firenze, Dipartimento di Ingegneria
dell'Informazione (DINFO).

A Elisa e ai miei genitori

Acknowledgments

I would like to thank my supervisors Prof. Fabio Schoen and Prof. Marco Sciandrone for their patience and precious input, and all of my colleagues at GOL who contributed to create a lovely working environment.

Contents

Chapter 1

Introduction

There are many steps in the design of a Machine Learning (ML) workflow where Global Optimization (GO) can be fruitfully employed. ML tasks can be categorized based on the availability of the class labels in the training set: when all of the samples have a label, we deal with *supervised* learning, while the two scenarios in which samples are partially labeled or not labeled at all are called respectively *semi-supervised* and *unsupervised* learning. While in the latter task, where the class information is completely unknown, what is commonly learnt is the support or the distribution of the input data, robust classification can still be achieved when only a few examples have a label. Semi-supervised learning is a ML concept which has gained increasing attention in the last decade. It aims at exploiting the unlabeled samples alongside with labeled ones, by including them in the learning process. This is helpful when dealing with ML tasks where gathering large amounts of unlabeled patterns is relatively easy, while labeling them is a costly process. In Chapter 2 we go into detail about semi-supervised learning, present a combinatorial algorithm to solve semi-supervised binary classification and prove its effectiveness and efficiency on a selection of small, medium and large scale datasets. Combinatorial approaches to semi-supervised classification are often able to deliver accurate models, successfully involving the unsupervised patterns in the learning process. However, these methods are known to scale badly with the size of the unlabeled dataset, and have been progressively abandoned by the recent literature. Thanks to the GO technique presented in Chapter 2, we are able to get rid of the dependency on the size of the unlabeled dataset, and solve the optimization problem yielded

by semi-supervised classification by means of a very efficient combinatorial procedure. Moreover, we show the optimality of the proposed technique, through an elegant and simple proof.
Not all of the possible traits of an input object are worth to consider – or available – when building a ML model (such as a classifier or a regressor), and a *feature selection* phase is often needed. Even with a relatively small number of features, selecting the most relevant ones is a crucial but complex combinatorial task. Moreover, the literature does not provide a clear definition of feature importance, thus making hard to evaluate and *rank* them before being selected. GO can be very helpful when dealing with feature selection and ranking. The approach presented in Chapter 3 involves GO techniques to build a formal definition of relevance. In particular, input characteristics are ranked and selected by solving a well-defined global problem. This formulation, in addition to clearly state a concept of variable relevance, is able to build a multivariate ranking of the features, that is, simultaneously taking them into account when assessing their importance. Moreover, the proposed ranking framework is well structured and lends itself particularly well to be implemented in a parallel and distributed fashion; to this aim, Chapter 3 also discusses a practical parallel implementation of the proposed feature ranking technique, based on state-of-the-art software tools and paradigms.

From an opposite perspective, ML techniques can enhance a GO scheme, and make it both more effective and efficient in detecting the global optimum. GO methods rarely keep track of past executions, and most of this potentially valuable information is lost. In particular, the only output of a GO algorithm is the prediction of the global optimum. Moreover, the computational effort is often redundant, as many of the solutions visited throughout the global search have been already encountered (and optimized). *Clustering* methods aim at grouping GO solutions around their basins of attraction, and focusing local search on the most promising configurations of each group. However, they lack robustness when the problem scale is large, as the clustering strategy is based (only) on the variables' value. For this reason, clustering methods have been abandoned by the literature after their initial success. In addition, problems with a clear geometrical interpretation have the further issue of having infinite *equivalent* solutions, obtained by trivial space transformations of given ones. In Chapter 4 we show how to improve a GO algorithm with information from its past runs and making a smart use

of the latter. Moreover, the idea of extracting compact geometrical descriptors from GO solutions is used for both stopping redundant lines of search of a perturbation-based GO method and to adapt clustering algorithms to work in a reduced *feature* space, in which configurations are compared and grouped by means of their overall characteristics rather than the value of their variables. This way clustering methods are rediscovered and employed in a modern and innovative fashion. Though we applied it to the scenario of atomic structure prediction (an active field in computational chemistry), our technique is straightforward and high-level, and can be easily used to enhance complex GO schemes and solve problems from different areas.

Chapter 2

Lagrangean-based Combinatorial Optimization for Large Scale Semi-supervised Support Vector Machines

In many ML scenarios, large amounts of samples can be cheaply and easily obtained. What is often a costly and error-prone process is to manually label these instances. The semi-supervised *approach faces this issue by making direct use of the unlabeled patterns when building the decision function. In particular,* Semi-supervised Support Vectors Machines *(S^{3}VMs) extend the well-known SVM classifier to the semi-supervised approach, aiming at maximizing the margin between samples in the presence of unlabeled data. Many optimization techniques have been developed in the last decade to efficiently include the unlabeled patterns in the learning process of SVM. Two broad strategies are followed:* continuous *and* combinatorial.

The approach presented in this chapter belongs to the latter family and is especially suitable when a fair estimation of the proportion of positive and negative samples is available. Our method is very simple and requires a very light parameter selection. Several medium and large scale experiments on both artificial and real-world datasets have been carried out, proving the effectiveness

and the efficiency of the proposed algorithm.[1][2]

2.1 Introduction and related work

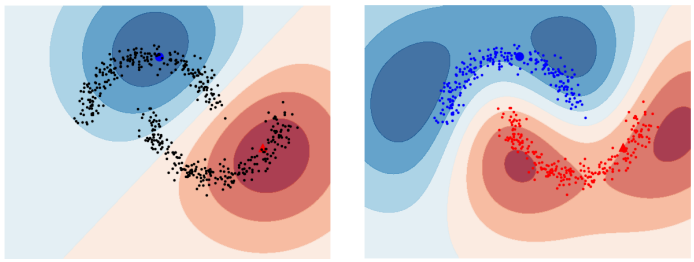
2.1.1 The semi-supervised scenario

Semi-supervised Support Vector Machines, (S^{3}VMs) extend the well-known SVM classifiers ([88]) to the semi-supervised scenario. In addition to using the labeled part of the training set to maximize the margin between classes, these classifiers take advantage of the unlabeled patterns, whose unknown labels are treated as additional optimization variables, by forcing the decision function to traverse through low density areas. This approach implements the so called *cluster assumption*, which states that samples that are close to each other are likely to have the same label. In other words, the decision function learned by a S^{3}VM has the double goal of separating the labeled examples of different classes while lying as far as possible from the unlabeled instances. The motivation for S^{3}VMs is that, in many applications, only a small percentage of the collected samples is provided with a label: learning a separating function with such an insufficient amount of labeled samples is very likely to yield an unconsistent model to be used for classifying future data. Figure 2.1 (a) shows a (rather pathological) example of the weak classification rule learned by making sole use of the labeled dataset (the blue big circle and the red big triangle), while ignoring the unlabeled samples (the black dots). Conversely, in Figure 2.1 (b) the learning process had successfully taken into account the whole dataset, and the resulting decision function is much more accurate and robust.

S^{3}VMs have been first introduced by [88] as *transductive* SVMs. This definition was due to the fact that the unlabeled samples that were used alongside the labeled dataset to draw the separating function were the same the authors would like to label. Conversely, the approach of building a classification rule on the entire input space (which is the classical approach of SVMs) is known in the literature as *inductive*. In [26] the authors elaborate on this distinction and carry out an empirical analysis. As it will be pointed out in Section 2.3,

[1] This work ([8]) has been candidate to the *best paper award* at the *Third International Conference on Machine Learning, Optimization and Big Data* in September 2017. An extended version ([7]) has been accepted by *IEEE Transactions on Neural Networks and Learning Systems* in October 2017.

[2] Part of the figures of this chapter are taken from [26] and [41].



(a) (b)

Figure 2.1: The separation function (a) learnt by making sole use of the labeled patterns (the blue big circle and the red big triangle) and that (b) obtained by including the unlabeled samples (the black dots) in the training process.

our experimental setting refers to the inductive approach.
Let us consider the linear binary classification problem. We are given ℓ labeled samples, $\{\mathbf{x}_i, y_i\}_{i=1}^{\ell}$ and a set of u unlabeled ones, $\{\mathbf{x}_i\}_{i=\ell+1}^{n}$, where each $\mathbf{x}_i$ is a d-dimensional vector, $y_i \in \{-1, 1\}$ and $n = \ell + u$. When dealing with $\mathtt{S}^3\mathtt{VM}$s the objective function one needs to optimize depends on both the parameters $(\mathbf{w}, b)$ of the decision boundary (which is an hyper-plane in the linear case) and the unknown labels $y_{i=\ell+1}^{n}$,

$$P(\mathbf{w}, b, y_{i=\ell+1}^{n}) := \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{\ell} V(y_i, \alpha_i) + C^* \sum_{i=\ell+1}^{n} V(y_i, \alpha_i), \qquad (2.1)$$

where $\alpha_i = \mathbf{w}^T\mathbf{x}_i + b$ are the linear predictions. The *hinge* loss

$$L(z) = \max\{0, 1 - z\} \qquad (2.2)$$

(which is depicted as a black continuous line in Figure 2.4 (a)) is a common choice for V, while the hyper-parameters C and C^* are used to give more importance respectively to the labeled or the unlabeled error term. Note that if we omit the third term from (2.1), that is, we get rid of the unlabeled part of the objective and the dependency on the variables $y_{i=\ell+1}^{n}$, what we obtain boils down to a standard `SVM` formulation. Optimizing (2.1) yields a linear decision function, that is, a separating hyper-plane: nonlinear boundaries

can be built by means of the *kernel trick* ([89]). A *balance constraint* is frequently added to the formulation mainly to avoid trivial solutions (with all the examples being classified as belonging to a single class), in particular when the number of labeled examples is significantly smaller than that of the unlabeled ones. This is achieved by letting the user specify a desired ratio r of unlabeled samples to be classified as positive, which leads to the following constraint:

$$\frac{1}{u}\sum_{i=\ell+1}^{n}\max\{y_i,0\}=r. \tag{2.3}$$

The above equation is equivalent to

$$\frac{1}{u}\sum_{i=\ell+1}^{n}y_i=2r-1, \tag{2.4}$$

as explained in [26].

2.1.2 Annealing

While the underlying optimization problem of SVM is convex, this is not the case for S^{3}VMs, where the objective function to be optimized is non-convex and has a great number of low-quality solutions. Local optimizers are thus no more enough to efficiently locate the global optimum of (2.1). Conversely, GO techniques can be useful to avoid getting trapped in one of the many suboptimal solutions. A common practice is to give an increasing importance to the unlabeled patterns. This GO approach, known as *annealing*, is often used in the literature (see [26] and [41]), and can be easily implemented by updating C^* in an outer loop.
From an optimization point of view, the smaller is the value that C^* is assigned to, the less non-convex is the unlabeled loss function; in particular, when $C^*=0$, (2.1) boils down to a standard SVM optimization. The hyperparameter C^* can therefore be used to control the degree of non-convexity of the entire objective: this can be very helpful to locate the global minimum. In [25] a similar global scheme is used to gradually smooth (2.1): the *continuation* method builds a smoothed objective at each iteration by convolution with a convex function, such as the Gaussian, which in turn is controlled by

a parameter γ. Figure 2.2 depicts this process.

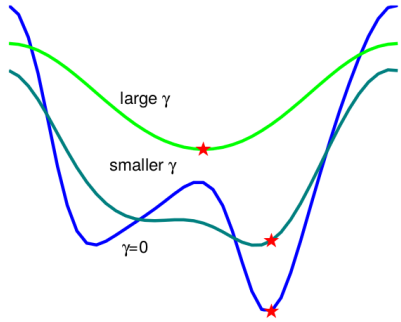


Figure 2.2: The *continuation* method: γ parameterizes a family of Gaussian functions, which are used throughout the learning process to control (by a convolution) the degree of non-convexity of $\mathtt{S^3VM}$'s objective function.

2.1.3 Continuous and combinatorial approach

The optimization techniques that have been recently developed to optimize (2.1) fall into one of two broad categories, *combinatorial* and *continuous*. In this section, the main ideas behind this two approaches are presented. An overview of methods from the literature is also given.

Combinatorial methods

Once that the unknown labels $y^n_{i=\ell+1}$ are fixed, what we get is a standard `SVM` formulation. Defining

$$I(y^n_{i=\ell+1}) := \min_{\mathbf{w},b} P(\mathbf{w}, b, y^n_{i=\ell+1}), \tag{2.5}$$

combinatorial approaches aim at minimizing $I(y^n_{i=\ell+1})$ over the set of binary variables $y^n_{i=\ell+1}$. The same applies to the nonlinear case, where $\mathbf{w} = \sum_{i=1}^n c_i y_i \mathbf{x}_i$ and the real variables $\mathbf{c}$ to be optimized are that of the Lagrangean dual of P. All the known combinatorial methods use a standard `SVM` to learn from the labeled examples, and use the information provided by the supervised subroutine to optimize the unknown labels, while satisfying the balance constraint. The first $\mathtt{S^3VM}$ implementation ($\mathtt{S^3VM^{light}}$, [52])

belongs to this family of strategies. It alternates minimizations in $(\mathbf{w}, b)$ obtained by training a standard SVM on the current labeled set at disposal and a heuristic labeling process, in which the ru unlabeled samples $\mathbf{x}_i$ with highest $\mathbf{w}^T\mathbf{x}_i$ are classified as positive (so to satisfy the balance constraint), while the others are assigned the negative class. An additional label switching phase is carried out to improve the value of the objective function: since only two labels per iteration are switched, the overall training process is very slow. Algorithm 1 sketches the method.

Input: $\{\mathbf{x}_i, y_i\}_{i=1}^n$, r, a SVM and an annealing sequence
$\mathcal{H} \leftarrow$ separating function obtained by SVM on $\{\mathbf{x}_i, y_i\}_{i=1}^{\ell}$
$y_{i=\ell+1}^n \leftarrow$ labeling computed by assigning 1 to the ru samples with highest distance from $\mathcal{H}$, -1 to the others
for C^* drawn from the annealing sequence **do**
  repeat
    $\mathcal{H} \leftarrow$ separating function obtained by SVM on the enhanced labeled set, with C^* weighting the patterns which initially had no label
    $y_{i=\ell+1}^n \leftarrow$ labeling obtained by switching (if possible) y_i and y_j, $i, j \in \ell{+}1, \ldots, n$, such that (2.1) is improved
  until no labels have been swapped
Output: the trained classifier

Algorithm 1: $\mathtt{S^3VM^{light}}$

The combinatorial approach itself is known to be intractable when u is large, due to the huge number of possible labeling combinations of the unlabeled patterns.

Another interesting combinatorial implementation is [25], where the authors develop a *branch-and-bound* (BB) scheme to achieve the optimal labeling for $y_{i=\ell+1}^n$. In particular, each node is associated to a partial labeling (the root corresponds to the set of labeled samples), and a binary tree is built whose branch variables are the unknown labels. At each node, a standard SVM is trained using both the labeled dataset and those unlabeled patterns which have been given a label throughout the construction of the BB tree. The value of the objective function at a node is used as a lower bound to possibly discard the corresponding subset of labeling solutions. Figure 2.3 depicts this process. This approach, as the authors state, is only feasible for small instances. Though, since the returned labeling is optimal, it can be useful for benchmarking $\mathtt{S^3VM}$s implementations.
Among others, we mention [36] and [80] as further methods belonging to the combinatorial family.

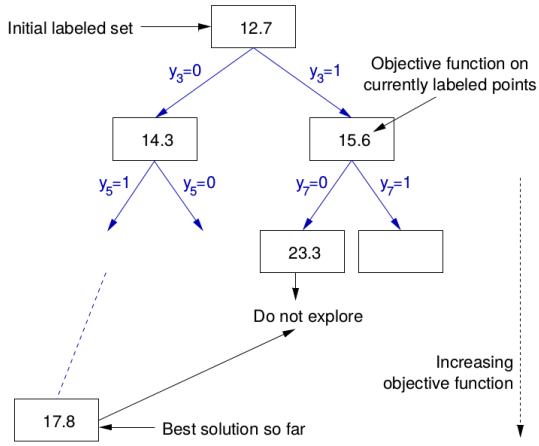


Figure 2.3: A *branch-and-bound* (BB) tree can be used to achieve an optimal labeling for $y_{i=\ell+1}^{n}$.

Continuous methods

This family of methods arises from the idea that, when $(\mathbf{w}, b)$ are fixed,

$$\arg\min_{y_i} V(y_i, \alpha_i) = sgn(\alpha_i) = sgn(\mathbf{w}^T\mathbf{x}_i + b). \tag{2.6}$$

In other words, once the separating hyper-plane has been drawn by maximizing the margin between the labeled examples, the best way of labeling an unknown pattern is to assign its class based on (the sign of) its distance from the decision function. This allows to get rid of the unknown label variables by replacing them with the expression of their prediction. The semi-supervised objective function (2.1) becomes:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{\ell}\max\{0, 1-\alpha_i y_i\} + C^*\sum_{i=\ell+1}^{n}\max\{0, 1-|\alpha|\}. \tag{2.7}$$

As pointed out in [26], the above equation shows how continuous $\mathtt{S^3VMs}$ implement the cluster assumption: the last term of (2.7) drives the separating

function away from the unlabeled patterns. The shape of the *hat* loss,

$$H(z) := \max\{0, 1 - |z|\}, \tag{2.8}$$

is depicted in Figure 2.4 (b) as a black continuos line. The non-convex nature

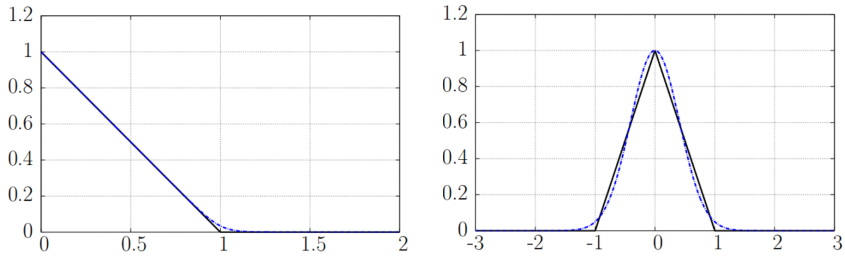


Figure 2.4: The hinge (a) and hat (b) losses (continuous) and their smoothed surrogates (dashed).

of this function violates one of the nicest properties of SVM, namely, the need of solving a convex optimization problem: this is what the continuous approach pays by removing the dependency of P on $y^n_{i=\ell+1}$. In addition, the hat loss is non differentiable in 0: for this reason, the smoothed surrogate

$$\tilde{H}(z) = exp(-tz^2),\ t > 0, \tag{2.9}$$

is often used in practice. The same non differentiability issue holds for the hinge loss, which is usually replaced by the *modified logistic loss* ([99]):

$$\tilde{L}(z) = \frac{1}{\gamma} \log\Big(1 + \exp\big(\gamma(1 - z)\big)\Big). \tag{2.10}$$

The surrogate hat loss (with $t = 3$) and the modified logistic loss are depicted in Figure 2.4 as dashed blue lines.

Another drawback of substituting the unknown labels with their predictions is that of getting a nonlinear balance constraint. A common approach to tackle this issue, introduced in [27], is that of working with a linear relaxation of the constraint; again, this is achieved by replacing the unknown

labels in (2.4) with their linear predictions:

$$\frac{1}{u}\sum_{i=\ell+1}^{n}\mathbf{w}^T\mathbf{x}_i + b = 2r - 1. \tag{2.11}$$

As suggested in [26], an unconstrained optimization problem can then be obtained by translating all the points such that the unlabeled ones have their mean in the origin, and by fixing $b = 2r - 1$. Though, it is worth to notice that replacing (2.4) with a linear constraint may lead to ignore potentially precious information. Directly enforcing the nonlinear balance constraint, especially when a fair estimation of the ratio r of positive samples is available, can be a fruitful design choice, as we will show in Section 2.2 and 2.3.

Due to the loss of convexity, as pointed out in [26], off-the-shelf dual-based `SVM` software cannot be used directly to optimize (2.1): for this reason, nonlinear decision functions are often obtained by implementing the kernel trick on the primal formulation, as in [24] and [41]; [26] shows two ways of obtaining such a primal kernel trick. Exceptions to this common practice are [32] and [60], where the authors develop ad-hoc dual methods.
We now provide a brief description of continuous methods from the literature.

`QN-S`^{3}`VM` ([41]) employs (2.9) and (2.10), differentiable surrogates respectively for the hat and the hinge loss (2.8) and (2.2). Defined $\mathcal{X} \subseteq \mathbb{R}^d$ as the feature space and a *kernel function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, the *representer theorem* ([78]) states that the optimal prediction takes the form:

$$f(\cdot) = \sum_{i=1}^{n} c_i k(\mathbf{x}_i, \cdot). \tag{2.12}$$

Learning a decision function can thus be formulated as a continuous optimization problem involving the variables $\mathbf{c} \in \mathbb{R}^n$. Substituting (2.9), (2.10)

and (2.12) in (2.1) yields the following objective function:

$$F(\mathbf{c}) = C \sum_{i=1}^{\ell} \frac{1}{\gamma} \log \left(1 + \exp\left(\gamma\big(1 - y_i \sum_{j=1}^{n} c_j k(\mathbf{x}_i, \mathbf{x}_j)\big)\right)\right) + C^* \sum_{i=\ell+1}^{n} \exp\Big(-t\big(\sum_{j=1}^{n} c_j k(\mathbf{x}_i, \mathbf{x}_j)\big)^2\Big) + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j). \tag{2.13}$$

Having proved that $F(\mathbf{c})$ and $\nabla F(\mathbf{c})$ can be computed polynomially ($\mathcal{O}(nd)$ in the linear case), the authors of [41] use *l-bfgs* ([19]) to efficiently solve (2.13), and employ an annealing scheme to avoid getting stuck in low-quality local minima. The balance constraint is enforced by making use of the linear approximation (2.11). The authors also claim that, given a indices subset $R \subset \{1, 2, \ldots, n\}$, the optimal prediction (2.12) can be approximated by:

$$f(\cdot) \approx \tilde{f}(\cdot) = \sum_{r \in R} c_r k(\mathbf{x}_r, \cdot). \tag{2.14}$$

The above formulation, known as *subset of regressors methods* ([75]), is used to speed-up predictions when the dataset is large.
`QN-S`^{3}`VM` is an overall efficient and effective method, though it is very sensitive to the choice of the hyper-parameters C and C^*. The latter issue actually affects all of the known `S`^{3}`VM` implementations from the literature, and is very likely to yield weak classification boundaries if the labeled dataset is not large enough to carry out a sound model selection.

`UniverSVM` ([32]) copes with the non-convexity of `S`^{3}`VM` by means of the *convex-concave* procedure (`CCCP`, [96]). `CCCP` decomposes a concave function in two components f_{vex} and f_{cave}, respectively convex and concave, and replace the latter with a (linear) tangential approximation. To get the next point, the sum of f_{vex} and the linear approximation is minimized. In the context of `S`^{3}`VM`, the concave part of the objective is due to the hat loss,

which is decomposed as

$$\max\{0, 1 - |t|\} = \underbrace{\max\{0, 1 - |t|\} + 2|t|}_{convex} \underbrace{-2|t|}_{concave} . \tag{2.15}$$

If, for instance, an unlabeled point is classified as positive, the hat loss takes the (convex) form:

$$\hat{H}(t) = \begin{cases} 0 & \text{if } t \geq 1 \\ (1-t)^2 & \text{if } |t| < 1 \\ -4t & \text{if } t \leq -1. \end{cases} \tag{2.16}$$

`UniverSVM` does not use an outer annealing sequence (although in [26] this design choice is found to be an improvement compared to the original formulation), and is quite slow compared to the other available online solvers.

`Well-SVM` ([61]) is one of the most recent and effective methods from the literature. The relaxed dual problem

$$\min_{\mathbf{y} \in \mathcal{B}} \max_{\boldsymbol{\alpha} \in \mathcal{A}} G(\mathbf{y}, \boldsymbol{\alpha}) = \mathbf{1}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T (\mathbf{K} \odot \mathbf{y}\mathbf{y}^T) \boldsymbol{\alpha} \tag{2.17}$$

is obtained from (2.1), where $\mathcal{B}$ is the set of all possible labeling solutions, $\mathcal{A}$ is the dual variables vector, $\mathbf{K}$ is the kernel matrix and $\odot$ is the element-wise product. The authors use convex programming ([53]) to solve (2.17) and develop a label generation strategy which makes use of the *multiple kernel learning* technique ([6]).

Unlike the other `S`^{3}`VM` solvers, which directly optimize the labeling of the unlabeled examples, **`Mean-S`**^{3}**`VM`** aims at maximizing the margin between the means of the positive and negative class. Once such an estimation of the label means is available, the optimization problem boils down to something very similar to that of `SVM`, with a loss function which is closely related to the hinge loss used by supervised `SVM`s. To solve the resulting problem, the authors employ two strategies: multiple kernel learning (as in `Well-SVM`) and *alternating minimization* ([35]).

`Lap-SVM` ([12]) is a semi-supervised specialization of a broader framework, which is suitable for problems spanning from unsupervised to supervised learning. The authors develop their general purpose scheme (which they call *manifold learning*), combining the theory of *reproducing kernel Hilbert spaces* ([78]) with geometrical knowledge of the data distribution, using the latter as a mean of regularization. Despite being a general purpose framework and an innovative way of working with partially labeled datasets, `Lap-SVM` is clearly the slowest and the least accurate among the S^3VM implementations available online, and has a large number of hyper-parameters which require to be fine-tuned.

2.1.4 Further notes on S^3VMs

An interesting discussion on scenarios where the use of unlabeled samples can be unsafe can be found in [59]. Some real-world applications of semi-supervised learning and particularly of S^3VM are [58] (spoken dialog systems evaluation), [92] (satellite image classification) and [85] (healthcare).

2.2 Lagrangean S^3VM

This section, in which we present our approach, is organized as follows: in Sections 2.2.1-2.2.3 we highlight two common issues affecting state-of-the-art S^3VM implementation and describe the nature of our method. In Section 2.2.4 we present `Lagrangean-`S^3VM in detail, providing the reader with the necessary mathematical tools. In Section 2.2.5 we elaborate on our parameter selection strategy. Finally, a theoretical proof of optimality is given in Section 2.2.6.

2.2.1 Dealing with hyper-parameters

The labeled samples which a classifier is trained on are vital when tuning the hyper-parameters of its learning algorithm. When only a few patterns in a dataset have a label, cross-validation techniques are likely to pick bad hyper-parameter settings, due to the very small size of the validation sets that can be built. Thus, the more hyper-parameters a method needs to be fine-tuned, the less robust the method is. In addition, even with a very efficient learning process, the initial hyper-parameter tuning phase could be

very time-consuming. In Section 2.2.5 we will put emphasis on the way our algorithm tackles these issues.

2.2.2 Balance constraint as a guide

Continuous methods get rid of the dependency on the unknown labels when formulating their semi-supervised objective (2.7), which forces to use a linear approximation of the balance constraint (2.11). Ignoring the nonlinearity of the balance constraint can hide potentially useful insight into the data. Furthermore, in many classification scenarios, a reasonable confidence on the percentage of examples to be classified as positive (r in the literature) is actually available. Let us think about a medical procedure in which we have to distinguish between patients who are likely to contract a particular disease and those who are not: the overall population incidence of a disease is often well-known or can be fairly well estimated from historical data. Moreover, semisupervised approaches are often used when a large amount of unlabeled data is available: this renders our estimation of r more likely to be a fair approximation of the *true* ratio. This insight can be usefully plugged in a S^{3}VM by means of the balance constraint. Our method carefully takes advantage of this information to draw the separating hyper-plane.

2.2.3 Inductive vs transductive S^{3}VMs

S^{3}VMs have been introduced by [88] as transductive classifiers: the unlabeled samples that were used together with the labeled dataset to learn the decision function were the same that needed to be labeled. As we will point out in Section 2.3, our experimental setting is *inductive*, that is, it builds a classification rule on the entire input space.

2.2.4 Method details

We present a combinatorial decomposition algorithm that uses a standard SVM as subroutine. For the sake of simplicity, we are going to use the linear formulation to elaborate on the algorithm's idea. The extension to the non-linear case is straightforward: the presented method only needs to be aware of the distances between the unlabeled samples and the separating function. At first, we initialize the decision function by optimizing (2.1) in $(\mathbf{w}, b)$ using the labeled part of the training set. At each iteration we give a label to the

unlabeled patterns $\{\mathbf{x}_i\}_{i=\ell+1}^{n}$ by taking into account both the current decision function and the balance constraint, that is handled by a Lagrangean technique[3]; this labeling process is extremely fast. The inner `SVM` is trained again on the extended labeled set, comprised of the patterns in $\{\mathbf{x}_i\}_{i=1}^{\ell}$ and those just labeled by the Lagrangean heuristic; the weight C^* is increased at each iteration following an annealing sequence, until it is assigned the same value of C.

Let us elaborate on how the Lagrangean labeling heuristic works. Once $(\mathbf{w}, b)$ have been computed, the variable part of (2.1) remains

$$U(y_{i=\ell+1}^{n}) := \sum_{i=\ell+1}^{n} \max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\}, \tag{2.18}$$

with the balance constraint (2.4). This boils down to a (constrained) optimal labeling problem:

$$\begin{aligned} &\min_{y_{i=\ell+1}^{n}} U(y_{i=\ell+1}^{n}) \\ &\frac{1}{u}\sum_{i=\ell+1}^{n} y_i = 2r - 1. \end{aligned} \tag{2.19}$$

Then, the idea is to relax the constraint by means of a Lagrangean multiplier λ and solve the corresponding dual problem, which takes the form:

$$\begin{aligned} &\max_{\lambda} \min_{y_{i=\ell+1}^{n}} L(\lambda, y_{i=\ell+1}^{n}) := \\ &\max_{\lambda} \min_{y_{i=\ell+1}^{n}} \sum_{i=\ell+1}^{n} \max\{0, 1-\alpha_i y_i\} + \lambda(\sum_{i=\ell+1}^{n} y_i - \beta), \end{aligned} \tag{2.20}$$

where α_i is the prediction $\mathbf{w}^T\mathbf{x}_i + b$ and the constant term β is defined as

[3]The idea of employing Lagrangean techniques to relax the balance constraint is proposed also in [80], but for a totally different formulation.

$(2r-1)u$. We can write, equivalently:

$$L(\lambda, y^n_{i=\ell+1}) = \underbrace{\sum_{i=\ell+1}^{n} \max\{\lambda y_i, 1-\alpha_i y_i \ + \lambda y_i\}}_{:=F(\lambda,\mathbf{y})} - \lambda\beta. \tag{2.21}$$

If we fix the Lagrange multiplier to a starting value λ_0, the above optimization problem becomes separable in the y variables and each component of $\mathbf{y}^* := \arg\min_{\mathbf{y}} F(\lambda_0, \mathbf{y})$ can be independently computed:

$$y_i^* = \arg \min_{y_i \in \{-1,+1\}} \max\{0, 1-\alpha_i y_i\} + \lambda_0 y_i, \ i = l+1 \ldots n. \tag{2.22}$$

We can then plug $\mathbf{y}^*$ in (2.21) and update the multiplier value. To this aim, it can be easily shown that

$$L(\lambda) = \min_{y^n_{i=\ell+1}} L(\lambda, y^n_{i=\ell+1}) \tag{2.23}$$

is a concave function of λ: we can take advantage of this property to obtain an updated value λ^{next} for the subsequent iteration of the Lagrangean heuristic. This iterative approach, called *cutting plane* (see [15]), is a typical non-differentiable optimization method used to solve the Lagrangean dual. Let $(\lambda^a, \mathbf{y}^a)$ and $(\lambda^b, \mathbf{y}^b)$ be a pair of dual solutions, such that

$$\sum_{i=\ell+1}^{n} y_i^a - \beta \ < \ 0 \ \text{ and } \ \sum_{i=\ell+1}^{n} y_i^b - \beta \ > \ 0, \tag{2.24}$$

and λ^{next} the multiplier value for which

$$L(\lambda^{next}, \mathbf{y}^a) = L(\lambda^{next}, \mathbf{y}^b). \tag{2.25}$$

Then, let y^{next} be the labeling obtained by plugging λ^{next} in (2.22): we can now set $y^a = y^{next}$ or $y^b = y^{next}$ according to the sign of the constraint violation $\sum_{i=\ell+1}^{n} y_i^{next} - \beta$, and iterate the process until convergence. The cutting plane method thus uses, at each iteration, an upper approximation of

the concave function of λ that is given by two subgradients, corresponding respectively to a negative and a positive constraint violation. The upper approximation is iteratively refined substituting a subgradient at a time until the optimal value of λ is determined. During the very first iterations of the heuristic, when $(\lambda^a, \mathbf{y}^a)$ and $(\lambda^b, \mathbf{y}^b)$ are not yet available, the multiplier λ is updated (coherently with the constraint violation) as prescribed by the *subgradient method* (see [15]). When the Lagrangean dual is polyhedral (the pointwise minimum of a finite number of affine functions), as it happens in our case, the cutting plane method terminates finitely (see Proposition 6.3.2 of [15]).

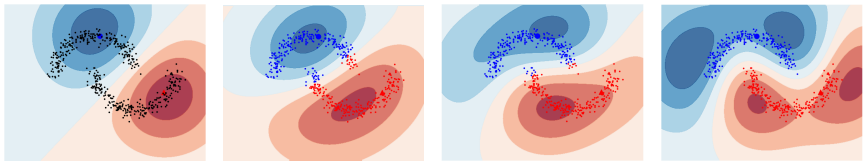
As described above, the Lagrangean heuristic returns with a labeling $y^n_{i=\ell+1}$ when the balance constraint is satisfied within a desired tolerance. During the subsequent annealing iteration of `Lagrangean-S`^{3}`VM`, a new separation function is computed by taking into account the enhanced labeled set. Algorithm 2 reports the pseudocode of our semi-supervised method; a few snapshots of the evolution of the decision function built by the method are depicted in Fig. 2.5.

Input: $\{\mathbf{x}_i, y_i\}_{i=1}^n$, r, a `SVM` and an annealing sequence
1: $C \leftarrow$ cross-validate the `SVM` on the labeled set $\{\mathbf{x}_i, y_i\}_{i=1}^{\ell}$
2: $\mathcal{H} \leftarrow$ separating function obtained by `SVM` on $\{\mathbf{x}_i, y_i\}_{i=1}^{\ell}$
3: **for** C^* drawn from the annealing sequence **do**
4: $\quad \boldsymbol{\alpha} \leftarrow$ distances of $\{\mathbf{x}_i\}_{i=\ell+1}^n$ from $\mathcal{H}$
5: $\quad y^n_{i=\ell+1} \leftarrow$ labeling computed wrt $\boldsymbol{\alpha}$ and r as in (2.18)-(2.25)
6: $\quad \mathcal{H} \leftarrow$ separating function obtained by `SVM` on the enhanced labeled set,
7: $\quad\quad$ with C^* weighting the patterns which initially had no label
Output: the trained classifier

Algorithm 2: `Lagrangean-S`^{3}`VM`

2.2.5 Parameter selection strategy

Almost everywhere in the literature, authors handle C and γ as proper `S`^{3}`VM` parameters, extending this setting with C^*. All the parameters need to be fine tuned by the experimenter (their choice widely influences the accuracy of the `S`^{3}`VM` approach), resulting in a very time-consuming validation phase. Our choice is totally different: our implementation does nothing but validate the internal supervised solver's parameter C on the (usually very limited) labeled set; the other parameter, γ, is kept fixed. This is done to initialize



(a) (b) (c) (d)

Figure 2.5: The snapshots of some selected annealing iterations of Lagrangean-S^{3}VM. Plot (a) shows the contours of the surface that separates the two labeled examples (the blue big circle and the red big triangle). After this initialization the unlabeled patterns are taken into account increasingly, and the decision function evolves until it reaches a balanced solution in (d).

the SVM and is a very quick process. In addition, C^* is handled by a standard annealing sequence and it is limited to assume value from a small finite set. Finally, the choice of r (which is the only actual parameter of our method) comes, as is common in the literature, from the knowledge of the problem domain. Moreover, differently from C, C^* and γ, this parameter's value has a clear and intelligible meaning, and can be easily used by the experimenter to feed our algorithm. In Section 2.3 we will show in detail how C and C^* have been chosen to carry out our experiments.

2.2.6 Optimality of the Lagrangean approach

We outline here a proof of optimality of the Lagrangean approach; in particular, we aim at proving that solving the Lagrangean dual of (2.19) provides an optimal labeling solution, which satisfies the balance constraint.

It can be easily shown that, once $(\mathbf{w}, b)$ are fixed, the labeling problem (2.19) is equivalent to

$$\begin{aligned} K := \max_{\delta} & \sum_{i=l+1}^{n} \alpha_i \delta_i \\ & \sum_{i=l+1}^{n} \delta_i = ru \\ & \delta_i \in \{0,1\} \ \forall i = l+1, \ldots, n, \end{aligned} \tag{2.26}$$

where ru is assumed to be an integer quantity. The above formulation is a *knapsack* problem, where each boolean variable δ_i, defined by means of the relationship $y_i = 2\delta_i - 1$, is equal to 1 if the i-th item is labeled positively and 0 otherwise, while ru is the number of items that have to be given a positive label. The proof of the equivalence between the above formulation and (2.19) is based on the simple observation that, given an optimal solution to (2.19), there cannot exist an item labeled with $+1$ whose coefficient α is smaller than that of an item with an opposite label; otherwise, a simple label swap would improve the objective function without violating the constraint. Due to the particular structure of the knapsack constraint, the linear relaxation of (2.26), obtained by letting the boolean variables δ_i assume value in $[0, 1]$, satisfies the integrality property: consequently, the optimal (integer) solution of the linear relaxation K^R is the same of (2.26). Solving the original labeling problem (2.19) is therefore equivalent to finding the optimal solution of a linear problem. By standard linear duality, we can thus affirm that the optimal solution of the Lagrangean dual of K^R is optimal for (2.19), which proves our claim.

2.3 Experiments

In this section we introduce our experimental setup and show the performance of the proposed algorithm, in terms of classification accuracy and execution time, in comparison with several methods from the literature. We carried out both small, medium and large scale experiments, making use of artificial and real-world datasets.

2.3.1 Algorithms

We have compared our method with a standard `SVM` classifier and several semi-supervised solvers. In particular, we used the `Python sklearn` ([73]) implementation of `SVM` (based on the `LIBSVM` library, [23]) as supervised method. For what concerns the semi-supervised solvers, we selected the most accurate among the S^3VM implementations available online: `QN-`S^3VM ([41]), `Well-SVM` ([61]), `Mean-`S^3VM ([60]), `Lap-SVM` ([12]) and `UniverSVM` ([32]). It is worth to notice that all these implementations belong to the continuous family: in fact, in the last decade, combinatorial methods have been increasingly put aside, due to their poor efficiency and bad scalability. One

of the aims of this work is to show the potential efficiency of combinatorial methods, through a smart use of optimization techniques.

2.3.2 Datasets

We have carried out experiments at different scales. To assess its effectiveness and efficiency on small and medium scale scenarios, we have tested our method with three artificial datasets, `2moons` (Figure 2.5 shows an instance), `2gauss` and `4gauss` (see [41] for construction details) and several real-world datasets, `isolet` ([31]), `mnist`[4], `usps` ([49]) and `coil20` ([70]). For each dataset we rescaled the features so that each value lies in $[0, 1]$; for `coil20`, we also rescaled each picture to 20x20 pixels. It can be noticed that many classification tasks can be derived from each real-world dataset: we denote with `_(i,j)` the binary classification task of distinguish between object i and j (pictures of everyday objects in `coil20`, handwritten digits in `usps` and `mnist`, spoken letters in `isolet`); we have taken into account those pairs of objects that are more difficult to recognize.
Dealing with a large scale scenario, we compared the algorithms on the sparse `rcv1` dataset ([5]), which is an archive of 800K newswire stories that have been manually categorized into one or more of 47K topics; the dataset has been made available by Reuters, Ltd. for research purposes.

2.3.3 Model selection

The setting in which we have compared our algorithm with the selected `S`^{3}`VM` solvers is the following: for the inner `SVM` model we have chosen a gaussian kernel $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2))$, and the supervised solver is internally preset (using the labeled patterns) by cross-validating over a very small set of values for C, while the kernel parameter γ is kept fixed at $1/d$, where d is the number of features. More in detail, with reference to Algorithm 1, C is selected from $\{2^i, i \in [0, 5]\}$ (line 2) and C^* take values $\frac{1}{10}C, \frac{1}{4}C, \frac{1}{2}C, C$, resulting in four annealing iterations (lines 3, 5 and 9). Table 2.1 recaps how the hyper-parameters' values have been choosen for our experiments.

Dealing with the other solvers, we have followed the approach reported in the literature, and we refer to their respective papers. For all of our experiments we used a 3-fold cross-validation approach to validate the algorithms' parameters. The only exception is the `moons` dataset: when $l \leq 10$, we used

[4] `http://yann.lecun.com/exdb/mnist.`

C	picked in $\{2^i, i \in [0,5]\}$ by validating `SVM` on $\{\mathbf{x}_i\}_{i=1}^{\ell}$
C^*	$\{\frac{1}{10}C, \frac{1}{4}C, \frac{1}{2}C, C\}$, resulting in four annealing iterations
γ	fixed at $1/d$
r	ratio of positive samples on $\{\mathbf{x}_i\}_{i=1}^{n}$

Table 2.1: A recap of the hyper-parameter selection.

default values for each solver, and $C = 1$ for `Lagrangean-S`^{3}`VM`. It is worth to notice, as already mentioned above, that our validation process is very light; in fact, we only have to choose a value for C to initialize the inner `SVM` classifier. Differently from the literature, where the hyper-parameters are selected by cross-validating the (parameters of the) whole semi-supervised method, we just validate the inner supervised routine to pick a starting value for C, selecting it from a small set. Nevertheless, in order to keep the comparison as fair as possible, all the execution time analyses we have carried out do not take into account the model selection phase, but only the training step.
For what concerns r, it has been set (for all the compared algorithms) to the ratio of positive samples in the whole dataset. We are thus assuming to have a good confidence on this measure; note that in the training and test sets this percentage can be different. Doing this we add some uncertainty to the value of r we pick and render its selection more fair.

Finally, it is common among the algorithms we have chosen for our comparison to make use of a surrogate function to approximate the hat loss $H(z) = exp(-tz^2)$ (see Section 2.1.3): for all methods using this approximation we set $t = 3$, which is the standard choice in the literature.

2.3.4 Experimental results

In our first experiment, we have compared the algorithms' classification accuracy on artificial and real-world datasets in a small scale setting. Following the experimental setup of [41], two different percentages of labeled examples are used for each classification task. Tables 2.2 and 2.3 report the mean classification error and its standard deviation achieved on ten different splits of each dataset configuration; ℓ, u and t denote respectively the number of labeled, unlabeled and test samples, while the best score for each configuration is marked in bold. Our experimental setting is inductive (see Section 2.1.1): we use two separated unlabeled sets, respectively for training and

testing. In other words, we employ ℓ labeled and u unlabeled samples for training a classification rule, and use the latter to label the t samples in the test set; this holds for all of the experiments in this Section. Looking at Tables 2.2 and 2.3, it is easy to notice that the semi-supervised approach outperforms standard `SVM` classification everywhere. For what concerns the semi-supervised methods, `Lagrangean-S`^{3}`VM` is the most accurate 73.17 percent of the time, which confirms its effectiveness. The overall accuracy of the other solvers are fairly comparable, each one of them doing its best on a subset of the classification tasks; `QN-S`^{3}`VM`, `Mean-S`^{3}`VM` and `Well-SVM`, however, perform slightly better than `Lap-SVM` and `UniverSVM`.

To evaluate the execution time in a small scale scenario, we have varied the training set size $n = \ell + u$ of two different classification tasks of `usps` from 100 up to 1000 samples. The execution time (averaged over ten different dataset splits) is reported in Figure 2.6: in both tasks, `Lagrangean-S`^{3}`VM`, `Mean-S`^{3}`VM`, and `QN-S`^{3}`VM` are the most efficient methods, growing linearly with the size of the training set; conversely, `UniverSVM` turned out to be the slowest among the compared algorithms.

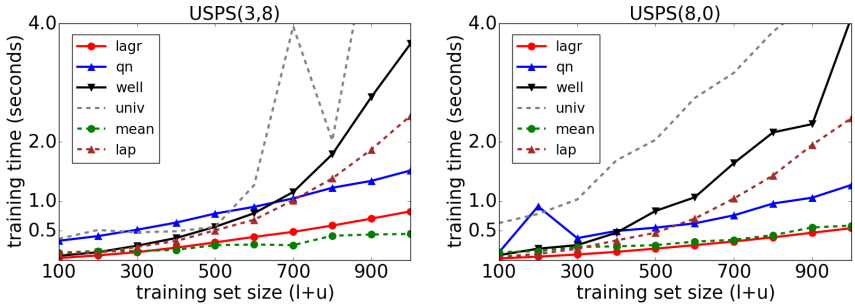


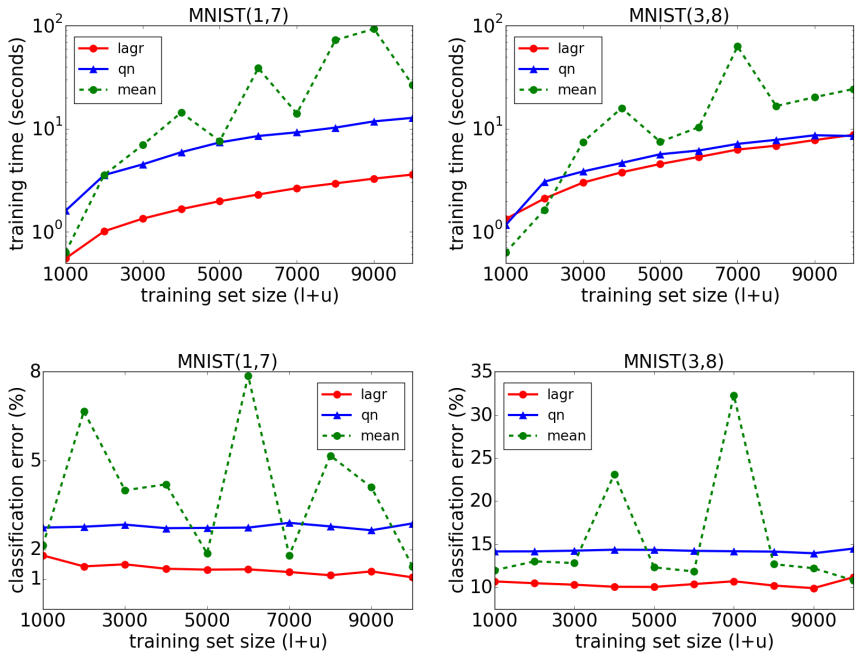
Figure 2.6: Small scale scenario: execution time (in seconds) with growing training set size (labeled and unlabeled samples, with ℓ being fixed at 25) on `usps(3,8)` and `usps(8,0)`. Results are averaged over ten different splits.

Let us switch to a medium scale scenario. `QN-S`^{3}`VM` makes use of an approximation heuristic when working with medium or large instances: in [41] the authors use the subset of regressors method (see Section 2.1.3), in which only a random subset of k support vectors are considered when building the prediction. In these experiments (as the authors did in [41]), we set $k = 2000$. We as well use a (very simple) medium/large scale approximation scheme in

our implementation: at each iteration, when training the inner supervised classifier on the enhanced labeled set, only a random portion of the latter is taken into account; for the medium scale setting, we set such training batch size to 500.
Focusing on the `mnist(1,7)` and `mnist(3,8)` classification tasks, we have gathered the execution time and related accuracy of the competing methods varying the training set size, keeping ℓ fixed at 25 samples. Figure 2.7 reports the outcome of our medium scale experiments, showing algorithms' execution time and classification error; we omit to consider `UniverSVM`, `Lap-SVM` and `Well-SVM`, which turned out to be too slow to be compared with the other methods in the current setting. `Well-SVM` offers, however, an alternative implementation suitable for sparse datasets, which we will use in the large scale scenario. It can be noticed that, in both tasks, `Lagrangean-S`^{3}`VM` and `QN-S`^{3}`VM` are the most efficient and overall accurate methods, showing comparable training time; `Lagrangean-S`^{3}`VM`, however, was able to reach a classification error lower than that of `QN-S`^{3}`VM`. For what concerns `Mean-S`^{3}`VM`, its training time and error is quite unstable on both tasks; in addition, training time peaks correspond often to high classification error: see for example `mnist(3,8)` with $l + u = 4000, 7000$.

Finally, let us consider the large scale sparse `rcv1` dataset. Each document belongs to one or more of 103 topics: we selected three of the most frequent, namely `C15`, `CCAT` and `GCAT`, and built just as many binary classification tasks, giving the positive label to the documents belonging to the target class and the negative one to all the others. Dealing with the compared algorithms, both `QN-S`^{3}`VM` and `Well-SVM` come with an alternative (and faster) ad-hoc implementation for sparse datasets, which we have used in the current setting. For what concerns our method, for the current tests we set the training batch to 2000 samples. Also, we omit to consider `Lap-SVM`, `UniverSVM` and `Mean-S`^{3}`VM`, being too slow to be tested on such a large dataset. For all of the experiments carried out on `rcv1`, a linear kernel is used, as suggested in [61]. Table 2.4 reports the *F-score* of the competing algorithms on the selected `rcv1` topics. We used such a classification measure due to the unbalanced nature of the classification tasks we have built. For instance, only 18% of the documents in the `rcv1` corpus belong to the `C15` topic: reporting the classification error could be misleading, showing an 82% accuracy for a classifier which actually gives the negative class to all of

Figure 2.7: Medium scale: execution time and classification error on `mnist(1,7)` and `mnist(3,8)` (respectively left and right) with increasing training set sizes; l is kept fixed at 25. Results are averaged on ten different dataset splits.



the examples in the test set. The F-score

$$F := 2 \cdot \frac{precision \cdot recall}{precision + recall} \in [0, 1], \tag{2.27}$$

conversely, takes into account both the *recall* (also known as *sensitivity*) – which is the ability of a classifier to identify the positive samples – and the *precision*, which denotes the ratio of true positives to all the instances that have been classified as positive. Also, similarly to what carried out in the small scale setting, we varied the number u of unlabeled samples: in particular, the training set size $n = l + u$ is kept fixed at 100K, while the test set is comprised of 50K instances. Table 2.4 shows that `Lagrangean-S`^{3}`VM` is the most accurate solver, reaching the highest F-score on 8 out of 12

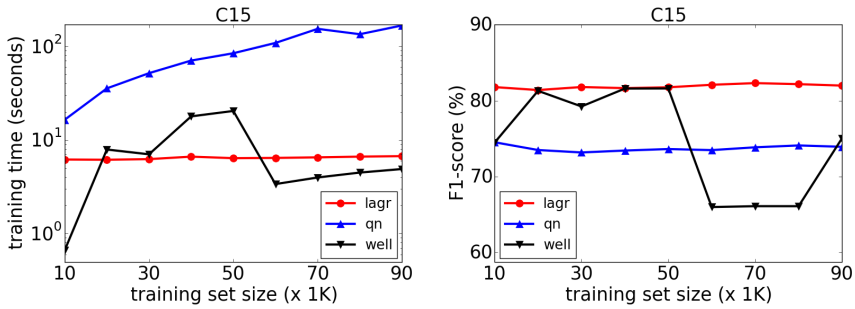


Figure 2.8: Large scale: training time and F-score on the `C15` topic from `rcv1`, with increasing training set size; l is kept fixed at 25, while the test set is comprised of 50K samples.

classification tasks. It can also be noticed that the supervised approach is quite unstable, showing a high performance variance on the 10 different dataset splits. This is due to the small percentage of labeled examples that `SVM` can take advantage of: for some splits, the supervised solver is unable to build a balanced decision function, classifying all the patterns with the negative class (which is most probable); this leads to a 0 recall for such splits.

In a further experiment, we focused on the `C15` topic and gathered the execution time and F-score with increasing training set size n, while l and t have been respectively set to 100 and 50K: Figure 2.8 reports the outcome of the experiment. As in the medium scale setting, `Lagrangean-S`^{3}`VM` and `QN-S`^{3}`VM` are the most stable solvers, both in terms of training time and F-score; actually, `Well-SVM` is quite faster than `QN-S`^{3}`VM` on the selected topic, but its training time does not grow in a linear nor predictable way. The same holds for the F-score; in addition, we can notice that `Lagrangean-S`^{3}`VM` is the only semi-supervised solver for which the F-score is slightly improving as the training set size grows, with its score rising from 81.8 (1000 samples) to 82.0 (10K samples).

In our last experiment, we compare the Lagrangean technique described in Section 2.2.4, Equations (2.18)-(2.25), with a sorting heuristic inspired by that used by `S`^{3}`VM`$^{\texttt{light}}$ in [52] to initialize the unknown label vector $y^n_{i=\ell+1}$. We will refer to the sorting and Lagrangean heuristics respectively with $\mathcal{S}$ and $\mathcal{L}$. Let us recall how $\mathcal{S}$ works: the ru examples with the highest $\mathbf{w}^T\mathbf{x}_i + b$ (namely, the distance between $\mathbf{x}_i$ and the hyper-plane defined by $\mathbf{w}$ and b)

are assigned the positive label, while the others are marked as negatives. Note that this strategy is that needed to solve the continuous relaxation of a generic 0-1 knapsack problem, such as that we used in Section 2.2.6 to prove the optimality of our labeling technique.
It is easy to prove that $\mathcal{S}$ is optimal with respect to (2.18) with the balance constraint, that is, it provides an optimal labeling for the unlabeled patterns, when $\mathbf{w}$ and b are fixed. Indeed, if we swap the labels $y_a = 1$ and $y_b = -1$ of two patterns $\mathbf{x}_a$ and $\mathbf{x}_b$ for which $\mathbf{w}^T\mathbf{x}_a > \mathbf{w}^T\mathbf{x}_b$, the balance constraint is still statisfied and objective function of (2.19) strictly increases; a further theoretical analysis is given in Section 2.2.6.
With the current experiment we therefore aim at showing how $\mathcal{L}$ is able to return a labeling of the same quality of $\mathcal{S}$, while being way faster: in fact, $\mathcal{L}$ is independent of the size u of the unlabeled dataset, while the computational cost of $\mathcal{S}$ is dominated by that of sorting the unlabeled patterns according to their distance from the separating hyper-plane, which costs $\mathcal{O}(u \log u)$. This is clearly shown by Figure 2.9: the two charts refers to `usps(8,0)` and `mnist(3,8)`, respectively a small and a medium scale classification task. In particular, they depict the ratios of the execution times and the resulting classification errors of the two heuristics (with $\mathcal{L}$ as the numerator), varying the size of the unlabeled data (l is kept fixed at 25): when the errors ratio falls below the dashed line, using $\mathcal{L}$ makes `Lagrangean-S`^{3}`VM` achieve a lower classification error than employing $\mathcal{S}$ as labeling heuristic, and vice versa. It is easy to notice how, even at a small/medium scale, the lagrangean heuristic is able to obtain the same quality of the optimal sorting technique, while being asymptotically faster: on the `mnist(3,8)` task, for instance, when $u = 1000$ the execution time of $\mathcal{S}$ is twice as much that needed by $\mathcal{L}$ to converge to a labeling solution, and the ratio decreases to approximately $\frac{1}{10}$ when $u = 11000$.

2.3.5 Statistical analysis

To further prove the superiority of our method, we carried out a statistical analysis, in which we compared the overall performance of `Lagrangean-S`^{3}`VM` with that of the other algorithms. In particular, our aim was to reject the null-hypothesis that `Lagrangean-S`^{3}`VM` performs as well as another method on our selection of datasets: thus, we run 5 different statistical tests, one for every other semi-supervised method involved in this study. As prescribed in [38], only one measure is taken into account when evaluating a method's

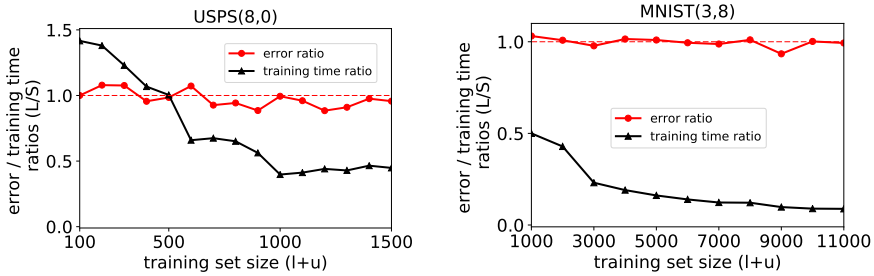


Figure 2.9: Ratios of classification errors and execution times of $\mathcal{L}$ (numerator) and $\mathcal{S}$ labeling heuristics on `usps(8,0)` and `mnist(3,8)`. When the errors ratio falls below the dashed line, the error achieved by `Lagrangean-S`^{3}`VM` employing $\mathcal{L}$ over $\mathcal{S}$ is lower.

performance on a given dataset, that is, all the error measures are averaged dataset-wise. We have considered each different classification task of `usps`, `coil20`, `mnist` and `isolet` as an independent dataset (indeed, there is no reason to suppose any correlation between samples used in two different tasks), totaling 22 datasets for `Lagrangean-S`^{3}`VM`, `QN-S`^{3}`VM` and `Well-SVM` (for which large scale tests have been conducted) and 19 for `UniverSVM`, `Mean-S`^{3}`VM` and `Lap-SVM`. As suggested in [38] to compare many algorithms over multiple datasets, we used the Wilcoxon signed ranks test. The statistical analysis confirmed the effectiveness of our method: in particular, `Lagrangean-S`^{3}`VM` performs better than `UniverSVM`, `Mean-S`^{3}`VM` and `Lap-SVM` with a confidence level of $\alpha = 0.001$, while the null-hypothesis is rejected respectively with $\alpha = 0.005$ and $\alpha = 0.05$ for `Well-SVM` and `QN-S`^{3}`VM`.

2.3.6 Technical details

We have implemented our algorithm[5] in `Python 2.7`, with the `sklearn` implementation of `SVM` as the internal supervised classifier. All execution time analyses have been performed on a desktop computer with an `Intel`® `i7` CPU at 2.93GHz, running `Ubuntu 14.04 LTS`.

[5] Available on `github.com/fbagattini/lagrangean-s3vm.git`.

2.4 Conclusion and remarks

The supervised approach to classification is not reliable when labeled data are scarce. Involving the unlabeled data when training a classifier can help to improve the classification accuracy in such a scenario. However, directly optimizing the unknown labels (combinatorial approach) can be intractable; on the other hand, expressing these variables in terms of their prediction (continuous approach) renders the objective non-convex. Recently, several methods have been proposed to tackle these two main drawbacks of semi-supervised classification. A common weak point of these methods is the large number of hyper-parameters that need to be cross-validated on a usually very small validation set. Our approach faces this issue by implementing an automated and very lightweight validation phase.
An additional drawback of the continuous methods lies in the need of linearly relaxing the balance constraint. Directly involving the balance constraint in the optimization problem has proved to be a good choice to outperform state-of-the-art solvers' accuracy on most datasets. The presented algorithm is also very efficient, thanks to the quick labeling process guided by a Lagrangean combinatorial scheme, which renders our approach suitable for large scale scenarios. Of course, our method is sensitive to the ratio r of unlabeled examples to be classified as positive, and should be used in a scenario in which there is enough confidence on the value of this parameter.

Table 2.2: Experimental results. ℓ, u and t denote respectively the number of labeled, unlabeled and test samples; for each configuration we report the mean and the standard deviation of the classification error on ten different dataset splits. `Lagrangean-S3VM`, `QN-S3VM`, `UniverSVM`, `Well-SVM`, `Mean-S3VM` and `Lap-SVM` are referred to respectively as `lagr`, `qn`, `univ`, `well`, `mean` and `lap`. Best results are in bold.

dataset	ℓ	u	t	svm	lagr	qn	univ	well	mean	lap
moons	2	498	500	14.92 ± 7.21	8.66 ± 9.01	11.22 ± 5.22	9.32 ± 7.72	$\mathbf{5.44 \pm 1.66}$	11.26 ± 7.50	12.06 ± 8.58
moons	3	497	500	29.92 ± 8.31	7.98 ± 9.68	11.24 ± 3.61	13.66 ± 1.02	$\mathbf{5.50 \pm 1.26}$	10.42 ± 7.13	11.14 ± 8.60
moons	5	495	500	20.10 ± 15.52	7.08 ± 8.80	10.60 ± 2.55	11.86 ± 7.99	$\mathbf{5.08 \pm 1.58}$	9.40 ± 4.13	10.74 ± 6.42
moons	10	490	500	11.02 ± 4.74	$\mathbf{1.36 \pm 1.67}$	3.28 ± 3.19	10.82 ± 5.78	4.28 ± 0.93	6.98 ± 1.84	8.60 ± 4.03
moons	20	480	500	6.02 ± 3.13	$\mathbf{1.10 \pm 1.48}$	2.08 ± 3.46	7.40 ± 4.70	1.22 ± 1.87	11.14 ± 13.52	4.02 ± 1.91
2gauss	25	225	250	23.48 ± 14.10	$\mathbf{3.04 \pm 1.12}$	3.68 ± 2.76	8.76 ± 6.42	4.56 ± 1.30	7.60 ± 2.16	15.40 ± 2.86
2gauss	50	200	250	8.68 ± 2.32	$\mathbf{2.40 \pm 0.88}$	2.76 ± 0.83	4.44 ± 1.74	5.12 ± 1.41	6.48 ± 2.68	8.60 ± 1.74
4gauss	25	225	250	17.52 ± 4.68	15.60 ± 18.46	8.60 ± 9.83	9.40 ± 8.82	$\mathbf{8.44 \pm 4.22}$	22.32 ± 13.28	19.32 ± 10.17
4gauss	50	200	250	8.64 ± 3.17	3.84 ± 2.78	$\mathbf{3.12 \pm 1.63}$	4.76 ± 2.71	5.20 ± 1.72	9.24 ± 4.06	11.72 ± 4.77
usps(2,5)	16	806	823	9.74 ± 6.04	$\mathbf{3.49 \pm 0.39}$	4.25 ± 1.32	7.96 ± 4.88	4.69 ± 1.11	4.79 ± 2.28	10.38 ± 2.24
usps(2,5)	32	790	823	5.58 ± 1.08	$\mathbf{3.62 \pm 0.33}$	3.88 ± 1.14	5.67 ± 1.41	4.34 ± 1.02	4.12 ± 2.45	9.64 ± 1.66
usps(2,7)	17	843	861	3.25 ± 0.95	$\mathbf{1.41 \pm 0.29}$	1.79 ± 0.60	6.55 ± 13.94	2.75 ± 0.89	6.35 ± 4.81	7.17 ± 1.82
usps(2,7)	34	826	861	2.46 ± 1.02	$\mathbf{1.38 \pm 0.35}$	1.85 ± 0.48	1.97 ± 0.62	1.94 ± 0.42	5.70 ± 5.60	5.52 ± 1.47
usps(3,8)	15	751	766	9.65 ± 2.70	$\mathbf{6.12 \pm 1.60}$	7.22 ± 2.51	10.14 ± 6.07	8.17 ± 2.33	13.77 ± 11.70	11.80 ± 1.90
usps(3,8)	30	736	766	6.76 ± 1.29	$\mathbf{4.84 \pm 1.39}$	5.16 ± 1.88	7.25 ± 2.46	5.69 ± 1.44	6.10 ± 1.01	8.50 ± 2.21
usps(8,0)	22	1,108	1,131	4.76 ± 2.12	$\mathbf{1.67 \pm 0.65}$	2.40 ± 1.19	6.16 ± 4.59	2.88 ± 1.08	3.02 ± 1.55	20.10 ± 4.19
usps(8,0)	45	1,085	1,131	3.53 ± 1.20	$\mathbf{1.51 \pm 0.52}$	1.95 ± 0.83	3.43 ± 1.34	2.11 ± 0.76	1.72 ± 0.80	19.47 ± 2.17

Table 2.3: Experimental results. ℓ, u and t denote respectively the number of labeled, unlabeled and test samples; for each configuration we report the mean and the standard deviation of the classification error on ten different dataset splits. `Lagrangean-S3VM`, `QN-S3VM`, `UniverSVM`, `Well-SVM`, `Mean-S3VM` and `Lap-SVM` are referred to respectively as `lagr`, `qn`, `univ`, `well`, `mean` and `lap`. Best results are in bold.

dataset	ℓ	u	t	svm	lagr	qn	univ	well	mean	lap
coil(3, 6)	15	100	29	16.90 ± 14.91	22.76 ± 17.46	$\mathbf{15.17 \pm 14.40}$	20.00 ± 21.19	17.59 ± 12.19	16.90 ± 11.69	27.59 ± 17.85
coil(3, 6)	25	90	29	6.21 ± 5.30	3.45 ± 4.88	$\mathbf{2.07 \pm 3.52}$	5.86 ± 6.18	7.59 ± 7.52	9.31 ± 9.76	7.93 ± 11.13
coil(5, 9)	15	100	29	29.31 ± 13.20	21.38 ± 10.09	$\mathbf{15.86 \pm 11.25}$	20.69 ± 16.17	$\mathbf{15.86 \pm 11.56}$	22.07 ± 13.28	18.62 ± 10.82
coil(5, 9)	25	90	29	14.14 ± 6.97	10.00 ± 9.44	$\mathbf{6.90 \pm 9.87}$	13.10 ± 11.42	9.66 ± 7.52	7.93 ± 5.35	7.93 ± 3.79
coil(6, 19)	15	100	29	7.59 ± 10.77	$\mathbf{6.55 \pm 12.09}$	8.62 ± 17.73	10.69 ± 19.90	8.97 ± 17.39	7.93 ± 11.13	13.10 ± 9.97
coil(6, 19)	25	90	29	5.52 ± 9.53	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	2.07 ± 2.76	0.34 ± 1.03	9.31 ± 12.34	4.48 ± 3.79
coil(18, 19)	15	100	29	10.34 ± 15.73	$\mathbf{0.69 \pm 2.07}$	2.07 ± 6.21	9.31 ± 15.27	3.79 ± 4.74	7.93 ± 12.63	5.86 ± 5.35
coil(18, 19)	25	90	29	1.72 ± 2.78	0.69 ± 2.07	$\mathbf{0.00 \pm 0.00}$	2.76 ± 6.32	4.48 ± 4.64	4.14 ± 4.31	4.14 ± 4.02
mnist(1,7)	20	480	500	4.92 ± 1.94	$\mathbf{2.06 \pm 0.69}$	5.30 ± 5.38	3.38 ± 2.32	3.46 ± 1.33	5.10 ± 3.65	4.36 ± 2.22
mnist(1,7)	50	450	500	3.18 ± 1.32	$\mathbf{1.92 \pm 0.67}$	3.60 ± 3.79	2.68 ± 0.92	2.60 ± 0.74	2.06 ± 1.00	3.80 ± 1.77
mnist(2,5)	20	480	500	14.86 ± 12.36	$\mathbf{3.70 \pm 0.72}$	4.56 ± 1.09	9.00 ± 4.39	9.44 ± 7.09	7.24 ± 7.51	7.52 ± 2.30
mnist(2,5)	50	450	500	5.46 ± 1.43	$\mathbf{3.12 \pm 0.72}$	4.76 ± 1.11	5.24 ± 1.33	4.56 ± 1.68	4.30 ± 2.35	5.54 ± 2.63
mnist(2,7)	20	480	500	5.70 ± 2.08	$\mathbf{3.72 \pm 0.78}$	4.18 ± 1.04	4.38 ± 1.24	4.62 ± 1.42	4.68 ± 1.39	6.90 ± 2.69
mnist(2,7)	50	450	500	4.60 ± 1.31	$\mathbf{3.44 \pm 0.96}$	4.42 ± 1.22	5.28 ± 2.33	4.66 ± 1.83	3.58 ± 1.30	4.36 ± 1.25
mnist(3,8)	20	480	500	16.84 ± 5.75	13.56 ± 3.89	12.78 ± 2.95	15.44 ± 4.23	$\mathbf{12.08 \pm 3.54}$	13.78 ± 2.98	15.44 ± 2.81
mnist(3,8)	50	450	500	10.86 ± 2.05	$\mathbf{8.56 \pm 1.32}$	10.32 ± 3.00	11.84 ± 3.70	9.62 ± 1.87	10.60 ± 2.32	12.52 ± 2.67
isolet(m,n)	20	330	249	25.46 ± 9.17	$\mathbf{17.75 \pm 5.53}$	18.39 ± 4.30	22.49 ± 11.09	22.25 ± 10.81	18.51 ± 3.35	22.53 ± 6.82
isolet(m,n)	40	310	249	14.38 ± 3.24	$\mathbf{10.24 \pm 2.07}$	11.57 ± 2.20	14.78 ± 4.08	11.57 ± 2.59	14.54 ± 3.26	14.14 ± 3.83
isolet(t,v)	20	330	250	5.00 ± 3.52	$\mathbf{1.44 \pm 0.60}$	1.56 ± 0.55	5.32 ± 4.62	2.52 ± 1.55	3.40 ± 1.85	4.32 ± 2.10
isolet(t,v)	40	310	250	3.04 ± 2.08	$\mathbf{1.24 \pm 0.49}$	1.64 ± 0.87	3.56 ± 3.03	2.96 ± 3.01	3.88 ± 2.61	3.16 ± 2.64
isolet(b,p)	20	330	250	9.00 ± 8.32	$\mathbf{3.52 \pm 1.17}$	3.88 ± 1.68	5.84 ± 4.14	4.44 ± 1.71	5.84 ± 2.65	8.88 ± 3.95
isolet(b,p)	40	310	250	6.32 ± 1.56	$\mathbf{3.88 \pm 0.88}$	4.16 ± 0.97	5.16 ± 2.50	5.08 ± 1.60	6.60 ± 5.85	5.20 ± 2.41
isolet(t,p)	20	330	250	18.76 ± 10.89	$\mathbf{10.28 \pm 4.01}$	10.48 ± 4.96	13.28 ± 4.87	16.28 ± 7.07	15.20 ± 4.72	15.92 ± 7.72
isolet(t,p)	40	310	250	11.20 ± 3.55	$\mathbf{7.64 \pm 1.95}$	9.24 ± 7.35	11.96 ± 3.93	9.12 ± 2.37	8.92 ± 2.81	9.04 ± 4.04

Table 2.4: Experimental results on a selection of `rcv1` topics. ℓ, u and t denote respectively the number of labeled, unlabeled and test samples; average F-score (on 10 dataset splits) and related standard deviation are reported. For each topic, various amount of labeled examples are considered. `Lagrangean-S`^{3}`VM`, `QN-S`^{3}`VM` and `Well-SVM` are referred to respectively as `lagr`, `qn` and `well`, while `svm` refers to the `Python sklearn` implementation of `SVM`. Best results are marked in bold.

topic	ℓ	u	t	**svm**	**lagr**	**qn**	**well**
C15	50	99,950	50,000	35.23 ± 19.58	$\mathbf{80.48 \pm 2.05}$	71.07 ± 5.00	73.35 ± 4.09
C15	100	99,900	50,000	52.41 ± 10.86	$\mathbf{82.45 \pm 1.57}$	74.10 ± 3.90	75.28 ± 4.00
C15	200	99,800	50,000	67.94 ± 4.23	$\mathbf{83.79 \pm 1.24}$	76.10 ± 3.59	80.82 ± 2.73
C15	500	99,500	50,000	79.21 ± 2.14	$\mathbf{85.40 \pm 0.65}$	79.49 ± 0.65	84.85 ± 0.67
CCAT	50	99,950	50,000	65.07 ± 20.29	$\mathbf{81.46 \pm 2.82}$	80.21 ± 4.39	72.79 ± 4.38
CCAT	100	99,900	50,000	73.43 ± 12.78	$\mathbf{84.34 \pm 2.05}$	81.49 ± 2.34	81.46 ± 2.82
CCAT	200	99,800	50,000	81.96 ± 4.35	$\mathbf{86.67 \pm 0.76}$	85.33 ± 0.98	85.00 ± 1.76
CCAT	500	99,500	50,000	87.17 ± 0.76	88.58 ± 0.67	87.72 ± 0.87	$\mathbf{89.27 \pm 0.43}$
GCAT	50	99,950	50,000	26.84 ± 15.92	87.51 ± 2.34	83.52 ± 3.10	$\mathbf{88.97 \pm 0.38}$
GCAT	100	99,900	50,000	61.61 ± 9.17	88.45 ± 1.06	$\mathbf{89.75 \pm 0.32}$	89.21 ± 0.25
GCAT	200	99,800	50,000	78.34 ± 3.87	$\mathbf{89.15 \pm 0.51}$	84.80 ± 0.62	87.35 ± 4.03
GCAT	500	99,500	50,000	87.11 ± 0.90	89.52 ± 0.26	86.39 ± 0.34	$\mathbf{89.98 \pm 0.22}$

Chapter 3

GO-based Feature Ranking Methods for Nonlinear Regression

Reducing the dimensionality of real-world data by selecting the most informative features is an important task in many ML scenarios. Feature selection *can help building cheaper and more accurate models. On the other hand, it is a very hard combinatorial problem to cope with, and an exhaustive search is unfeasible even with a small variable space. Many optimization models and techniques, including GO, have been proposed in the recent years to provide high(er) classification or regression accuracy while using a limited set of features.*
Today, powerful architectures and programming frameworks allow to implement ML algorithms in a parallel and distributed fashion. Apache Spark*, which provides a convenient in-memory abstraction for parallel operations, is one of the most used.*
In this chapter we present `COBAS`*, an optimization-based technique for feature ranking, and propose a detailed implementation using the Spark framework.*[1]

[1]The content of this chapter is based on [21]. My contribution lies in the parallel implementation and adaptation to large scale scenarios.

3.1 Introduction and related work

In many ML applications, the number of features, that is, of variables which represent a data sample, deserves special attention. We highlight three scenarios.

***Expensive* features** Let us consider a supervised medical ML task, such as, for example, that of predicting the onset of a desease on a patient, based on a number of medical measurements. In this task, a positive label is associated to those patients (the supervised samples) who, in the past, have been diagnosed with the disease; the set of measurements on which the prediction is based is the feature vector. In addition, suppose that one or more of the latter medical variables presuppose invasive or costly medical tests. It is easy to see that, beside the main prediction objective, obtaining some information on the correlation between these expensive features and the disease might be useful; even more so, avoiding the need of using them for the prediction is a further important goal.
Let us denote by E the classification error achieved by the best model M we have at our disposal for the medical task, and by d the size of the feature space. Our prediction problem can be (re)stated as that of *building a model* $\bar{M}$ *using a subset of* $\bar{d} < d$ *non expensive features, which error* $\bar{E}$ *is as good as (or even lower than)* E.

More samples than features In many real-world applications, such as gene expression analysis, combinatorial chemistry, automated text classification, software defects prediction or image retrieval, the number of variables is often one or more orders of magnitude larger than that of the samples at one's disposal. Building a predictive model in such a scenario is intractable, and a feature selection step is needed upstream of the learning process.

Too many (non informative) features Even if there is enough data the learn from, detecting the actually informative variables among the many available is likely to improve the accuracy of the resulting model by avoiding overfitting; in addition, feature selection might allow the user to better interpret the outcomes and reduce both the space and time complexity of the learning process.

The last scenario is the more broad. In general, a feature selection phase can

improve the performance and robustness of a predictive model. A valuable survey on feature selection is [45].

A *ranking* step, where features are sorted by some relevance criterion, is often applied before the actual selection. This additional phase can be useful when building a decision support system; see, for instance, [1], [2] and [28]. Moreover, feature ranking is in itself a valuable tool, and has not to be necessarily used to define a ML model. There are several definitions of variable relevance in the literature (see, for example, [16] and [95]). The peculiarity of the method presented in this chapter is to formally state the concept of relevance in terms of a well-defined optimization problem.
Feature ranking methods can be grouped in three classes. *Wrapper* methods use a ML model as a black box to compute the variable ranking: therefore, these techniques presuppose the availability of a reliable model to be employed. In addition, the time needed for computing the ranking has to include that of training the underlying model. Conversely, *filter* methods make sole use of the training data, and are independent of a given learning algorithm; the most used algorithm of this family is `Relief` ([55]). Finally, *embedded* methods compute the ranking during the training process, by adding a penalty term to the objective function which is related to the number of variables to be minimized. A common point of many of the feature ranking algorithms from the literature is to compute the score of a variable independently, that is, without considering the other features. This can be a drawback: the authors of [45] state that *a variable that is completely useless by itself can provide a significant improvement when taken with others.* The approach presented in this chapter belongs to the wrapper family and is *multivariate*: both when training the underlying model and computing the variable ranking, all the features are considered simultaneously.

Despite being a very powerful tool, feature selection is a costly process; even with a small variable space, an exhaustive search of the best variable configuration is intractable. Selecting a subset of variables can be stated as a combinatorial problem, where the $i-$th entry of the boolean solution vector is 1 if the $i-$th feature has been chosen. A local search in the resulting combinatorial space is very likely to get stuck in one of the many suboptimal solutions. For this reason, many attempts have been done in the literature to use GO to solve feature selection; in the following, some examples are reported. In [20], the accuracy of a cancer diagnosis model was improved employing a variable selection based on concave optimization. The authors

of [91] used *rough sets* ([72]) – a set theory abstraction which is often used in feature selection – alongside with the global scheme provided by *Particle Swarm Optimization* (PSO, [54]). PSO was also used in [63], both to select the most informative features and to refine the hyper-parameters of an SVM; addressing the same goal, the authors proposed in [62] an alternative global approach, based on *Simulated Annealing* (SA, [87]). In [10], two GO schemes, namely SA and *tabu search* ([43]), were used alongside with *k-nearest neighbors* ([34]) to solve the feature selection problem. In [18], GO was used to simultaneously refine the hyper-parameters of a ML model while using a subset of variables. Finally, the authors of [86] selected a subset of features by using a *genetic* algorithm ([83]) and *differential evolution* ([9]), while a local heuristic was employed to produce an initial feasible solution.
In the next section, we will see how the relevance of a feature can be obtained by solving an optimization problem in an efficient way by means of GO.

3.2 Feature ranking by concave optimization

3.2.1 Regression, inversion and a concept of relevance

Unlike classification, where labels are boolean, in a *regression* problem data samples are labeled with continuous values. We are given a training set $\{\mathbf{x}^i, y^i\}_{i=1}^n$, where the samples $\mathbf{x}^i \in \mathbb{R}^d$ belong to a feature space of d variables and $y^i \in \mathbb{R}$, $i = 1, \ldots, n$[2]. Training, or *fitting* a regression model consists into approximating the unknown relation $\phi : \mathbb{R}^d \to \mathbb{R}$, which maps a vector in the feature space to a real number, by using the data sample provided by the training set. With feature ranking, we aim at computing a relevance score for the variables of the underlying mapping.
Regression models assume the unknown relation between data and labels to belong to a given family; for instance, if we seek for a linear approximator, we are carrying out a linear regression. In general, given a parametric family of functions $F(\cdot; \mathbf{w}) : \mathbb{R}^d \to \mathbb{R}$, fitting a regression model boils down to optimize the parameter $\mathbf{w}$, that is, picking the value of $\mathbf{w}$ whose corresponding function from the family fits the data best. A *universal approximator* is a family of functions which is able to build every continuous function on a compact set; *feed-forward neural networks* ([74]) and SVMs with Gaussian kernel

[2]This notation, which is slightly different than that used in Chapter 2, better suits the feature selection scenario.

are known to be universal approximators. From now on, we will assume a regression model to be a universal approximator.

Inverting a regression model consists into determining an input for which model prediction yields a desired output; since it has many solutions, it is regarded as an ill-posed problem. With the additional goal of maximizing a generic criterion f on the input, the inversion problem can be formulated as:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\quad F(\mathbf{x}; \mathbf{w}^*) - \bar{y} = 0 \\ &\quad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}. \end{aligned} \tag{3.1}$$

In the above formulation, $F(\cdot; \mathbf{w}^*)$ is the trained model, $\bar{y}$ is the target output and $\mathbf{l}$ and $\mathbf{u}$ are respectively lower and upper bounds on the variables, resulting in d box constraints. The criterion f defines the kind of inversion we would like to compute. For example, by taking $f(\mathbf{x}) := \|\mathbf{x} - \mathbf{c}\|$, where $\mathbf{c} \in \mathbb{R}^d$ is a *reference* point, we obtain the *inversion nearest to the reference point* problem.
Given a trained model $F(\cdot; \mathbf{w}^*)$, a training sample $\mathbf{x}^p$ and a training label y^q, with $p \neq q$, we introduce the concept of relevance of a variable i, $i = 1, \ldots, d$, with respect to the pair $(\mathbf{x}^p, y^q)$. In particular we call the feature i relevant if, starting from $\mathbf{x}^p$ and modifying the minimum number of components, we determine an input $\mathbf{x}^*$ for which

1. the model yields a prediction $F(\mathbf{x}^*; \mathbf{w}^*) = y^q$ and
2. i is among the modified components, that is, $x_i^* \neq x_i^p$.

We can then define the following inversion problem,

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) := \|\mathbf{x} - \mathbf{x}^p\|_0 \\ &\quad F(\mathbf{x}; \mathbf{w}^*) - y^q = 0 \\ &\quad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \end{aligned} \tag{3.2}$$

where, $\|\cdot\|_0$ is the *zero-norm*, i.e., $\|\mathbf{t}\|_0 := CARD\{i : t_i \neq 0, \quad i = 1, \ldots, d\}$.

3.2.2 A feature score

Let us now define a measure of importance for the features. If we suppose to have determined a finite subset $\mathcal{X}^*$ of solutions of (3.2), we say that a feature $i \in \{1, \ldots, d\}$ is *relevant with respect to the pair* $(\mathbf{x}^p, y^q)$ if there exists a solution $\mathbf{x}^* \in \mathcal{X}^*$ such that $x_i^* \neq x_i^p$. We can then define, for each feature i, the set

$$\mathcal{X}_i^* := \{\mathbf{x} \in \mathcal{X}^* : x_i \neq x_i^p\}. \tag{3.3}$$

We can use (3.12) to compute a score $r_i(\mathbf{x}^p, y^q)$, which represents the importance of the feature i; this could be, for instance, the cardinality of the set $\mathcal{X}_i^*$. By varying the pair $(\mathbf{x}^p, y^q)$ and computing the corresponding scores for each feature, we can build an overall ranking.

3.2.3 Concave optimization for zero-norm minimization

We can highlight two main difficulties of (3.2). First, its formulation contains a highly nonlinear constraint. In case we are not interested in satisfying the latter exactly, we can shift it to the objective function by adding a quadratic penalty to its violation. The resulting objective function would take the form

$$f(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}^p\|_0 + \frac{1}{2} C \left(F(\mathbf{x}; \mathbf{w}^*) - y^q\right)^2, \tag{3.4}$$

with C weighting the importance given by the user to the inversion rule.
Secondly, the objective function in (3.2) is discontinuous and non smooth. To cope with this issue, we transform the objective into an equivalent smooth one, and solve the resulting formulation by means of concave optimization. Let us consider the problem

$$\min_{\mathbf{x} \in S} \ \|\mathbf{x}\|_0 \,, \tag{3.5}$$

where $S \subset \mathbb{R}^d$. We observe that (3.5) is equivalent to

$$\min_{\mathbf{x} \in S} \ \sum_{i=1}^{d} s(|x_i|) \,, \tag{3.6}$$

where the *step* function $s(\cdot)$ is equal to 1 for strictly positive inputs, 0 otherwise. Following [68], we can approximate (3.6) by replacing the discontinuous step function with $1 - e^{-\alpha t},\ \alpha > 0$, which is continuously differentiable and concave, thus obtaining

$$\begin{aligned} \min_{\mathbf{x} \in S,\, \mathbf{z}} & \sum_{i=1}^{d} 1 - e^{-\alpha z_i} \\ & -z_i \leq x_i \leq z_i, \quad i = 1, \ldots, d. \end{aligned} \tag{3.7}$$

The author of [68] proved that problem (3.7) is equivalent to (3.6), under the hypothesis that S is a polyhedral set; more in detail, for α large enough, there exists a solution of (3.7) which is also a solution for (3.6). Though, the feasible set of (3.2) is not polyhedral. Thus, in the following, we will use the above approximation strategy as an heuristic to manage the zero-norm. Finding a vector to a polyhedral set having the minimum number of non-zero components, is a NP-hard problem; the above heuristic helps making it tractable. In [94], a different function is used to smooth the step function. Finally, since discrete or categorical features would influence the structure of the feasible set S, and render the optimization problem a lot harder to solve, we assume to work, from now on, with continuous features.

3.2.4 Details of the ranking method

Following the recipe of the previous section, that is, expressing the inversion rule by a quadratic penalty on the objective function and by using a concave approximation of the zero-norm, we obtain the following formulation:

$$\begin{aligned} \min_{\mathbf{x},\, \mathbf{z}} & \sum_{i=1}^{d} 1 - e^{-\alpha z_i} + \frac{1}{2} C \left(F(\mathbf{x}; \mathbf{w}^*) - y^q \right)^2 \\ & -z_i \leq x_i - x_i^p \leq z_i, \quad i = 1, \ldots, d \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}. \end{aligned} \tag{3.8}$$

The above is a non-convex GO problem. Let $\mathcal{X}^*$ be the finite set of putative global minima found by solving (3.8). In general, as already pointed out, we cannot guarantee that a solution in $\mathcal{X}^*$ is neither feasible nor a solution of

(3.2). We thus define the set $\bar{\mathcal{X}} \subseteq \mathcal{X}^*$ of *quasi-feasible* minima, such that

$$|F(\mathbf{x}^*; \mathbf{w}^*) - y^q| \leq \epsilon \quad \forall \mathbf{x}^* \in \bar{\mathcal{X}}, \tag{3.9}$$

where $\epsilon > 0$ is a user specified tolerance on the inversion rule. Then, we define the set $\hat{\mathcal{X}}$ of *quasi-feasible zero-norm* minima:

$$\hat{\mathcal{X}} := \{\mathbf{x}^* \in \arg\min_{\mathbf{x} \in \bar{\mathcal{X}}} \|\mathbf{x} - \mathbf{x}^p\|_0\}. \tag{3.10}$$

Finally, for $i = 1, \ldots, d$, we define the set $\bar{\mathcal{X}}_i$ of minima "reached" by feature i:

$$\bar{\mathcal{X}}_i := \{\mathbf{x} \in \bar{\mathcal{X}} : x_i \neq x_i^p\}, \tag{3.11}$$

and we compute the score

$$r_i(\mathbf{x}^p, y^q) := |\bar{\mathcal{X}}_i|. \tag{3.12}$$

An overall ranking of the features in built by sampling reference points and target outputs respectively from training patterns and labels. By doing so, if we assume that the model $F(\cdot; \mathbf{w})$ is reliable enough, we can affirm that we are selecting target outputs which are "reachable"; in other words, given a target output y^q, there exist at least one point $\mathbf{x}^p$ such that $|F(\mathbf{x}^p; \mathbf{w}^*) - y^q| \leq \epsilon$. The approach, called `COBAS`, is sketched in Algorithm 3. In the next

Input: M, a training set $\{\mathbf{x}^i, y^i\}_{i=1}^n$ and a trained regression model $F(\cdot; \mathbf{w}^*)$
1: $\mathcal{P} \leftarrow M$ pairs $(\mathbf{x}^p, y^q)$, with $p \neq q$, where $\mathbf{x}^p$ and y^q are respectively a pattern and a label of $\{\mathbf{x}^i, y^i\}_{i=1}^n$
2: **for** each pair $(\mathbf{x}^p, y^q)$ **do**
3: $\quad \mathcal{X}^* \leftarrow$ set of putative optima obtained by solving (3.8)
4: $\quad \bar{\mathcal{X}} \leftarrow$ set of *quasi-feasible optima* as defined in (3.9)
5: $\quad \hat{\mathcal{X}} \leftarrow$ set of *quasi-feasible zero-norm optima* as defined in (3.10)
6: $\quad$ **for** each feature $i = 1, \ldots, d$ **do**
7: $\quad\quad \hat{\mathcal{X}}_i \leftarrow$ set of *minima reached by* i as in (3.11)
8: $\quad\quad r_i(\mathbf{x}^p, y^q) \leftarrow$ ranking of i wrt $(\mathbf{x}^p, y^q)$, computed as in (3.12)
Output: a ranking of the features $i = 1, \ldots, d$ based on scores r_i

Algorithm 3: `COBAS`

section, experimental results are reported for both artificial and real-world datasets. In Section 3.3, a distributed implementation using *Apache Spark* is proposed.

3.2.5 Experimental setting and results

In this section we discuss the numerical results reported in [21]. Experiments are carried out both on artificial and real-world dataset. While the former are performed to provide a proof of concept and to assess the effectiveness of `COBAS` in detecting features actually correlated with the output, real-world experiments allow to qualitatively evaluate the proposed approach.
In [21], a *feed-forward neural network* is employed as regression model; in particular, the authors adopt a *radial basis function* network (`RBFN`) with *inverse multiquadric* as activation function:

$$F(\mathbf{x};\mathbf{w}) := \sum_{i=1}^{h} \lambda_i (\|\mathbf{x}-\mathbf{v}_i\|^2 + \sigma^2)^{-\frac{1}{2}}. \tag{3.13}$$

In the above equation, h is the number of hidden neurons, $\lambda_i \in \mathbb{R}$ and $\mathbf{v}_i \in \mathbb{R}^d$ are hyper-parameters and σ is set to 0.1. The network has been trained by minimizing a least-squares error function, measuring the output error on the set of training pairs; a gradient-based batch strategy with early stopping has been used as optimization strategy, and the number h of hidden neurons is chosen by a cross-validation technique. For what concerns `COBAS`' parameters, the authors set the number M of training pairs to 500, while the quasi-feasible threshold ϵ has been set to 10^{-4}. Finally, the multi-start GO strategy used to solve (3.8) employed `MINOS` ([69]) as local solver.
`COBAS` is a wrapper method, thus needing a regression model to be trained to perform the feature scoring. In addition, the number of training pairs is required to be large enough in order to produce a robust scoring. However, a major part of the above operations do not need to be implemented in a sequential way. For this reason, a distributed implementation is proposed in Section 3.3.

Results on artificial datasets

Consider the following functions:

$$f_1(\mathbf{x}) := (x_1^4 - x_1^2) \cdot (3 + x_2) \tag{3.14a}$$
$$f_2(\mathbf{x}) := 2(x_1^3 - x_1) \cdot (2x_2 - 1) \cdot (x_2 + 1) + (x_2^3 - x_2 + 3) \tag{3.14b}$$
$$f_3(\mathbf{x}) := -2(2x_1^2 - 1) \cdot x_2 \cdot e^{-x_1^2 - x_2^2} \tag{3.14c}$$
$$f_4(\mathbf{x}) := x_1 + (x_2 > 0.5) \cdot (x_3 > 0.5) \tag{3.14d}$$
$$f_5(\mathbf{x}) := 10 \sin(x_1) x_2 + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5. \tag{3.14e}$$

In (3.14d), $(x_i > 0.5)$ is equal to 1 when $x_i > 0.5$ and to 0 otherwise. A synthetic dataset has been obtained by each of the above functions by generating n data points in an interval $[a, b]^d$, and computing the corresponding labels by appling $f_i(\cdot)$ to each point. Such interval is chosen as $[-3, 3]$ for f_1 and f_2, $[-1, 1]$ for f_3 and $[0, 1]$ for f_4. Functions (3.14a)-(3.14c) were originally proposed in [77], while (3.14d) and (3.14e) were introduced in [14]. It can be noticed that these functions depend nonlinearly only on some variables. The authors varied the number d of features and that of data points, n, used to build the `RBFN`. For each test, ten runs were performed by taking into account just as many different randomly generated datasets. `COBAS` is then compared with three filter methods, namely `RRelief` (an adaptation of `Relief` to regression problems, [55]), *Mutual Information Criterion* (`MFI`, [97]) and *Pearson Correlation* (`PC`, [13]), and the embedded *Recursive Feature Elimination* method (`RFE`, [46]).

To assess the effectiveness of the compared methods, a scoring system is built as follows. Let R_t be the number of features actually correlated with the output (for instance, variables 1 and 2 of function f_1), and I_t be the number of those which have no correlation with the function output. A feature is assumed to be marked as relevant by an algorithm if the latter ranks it in the first R_t positions; otherwise, it is marked as not relevant. The number of relevant and irrelevant features selected by a method as relevant are respectively denoted by R_s and I_s. By reference to [17], the following performance index is defined,

$$100\Big(\frac{R_s}{R_t} - \alpha \frac{I_s}{I_t}\Big), \tag{3.15}$$

which rewards the selection of relevant features while penalizing that of irrelevant ones; the tradeoff is controlled by α, which is set as $\min\{1/2, R_t/I_t\}$. If all of the relevant variables are ranked before the irrelevant ones, the performance index reaches its maximum, and is equal to 100. Table 3.1 reports the results on the synthetic datasets.

Table 3.1: Performance of `COBAS` and other selected algorithms on a choice of synthetic datasets.

	COBAS		RReliefF		RFE		MI		PC	
function f_1										
n \ d	6	15	6	15	6	15	6	15	6	15
100	100	94.9	93.7	84.6	93.8	74.4	87.5	64.2	68.8	43.7
200	100	100	93.7	69.3	100	94.9	87.5	69.3	81.3	53.9
500	100	100	100	89.7	100	100	100	84.6	62.5	53.9
1000	100	100	100	100	100	100	100	100	62.5	53.9
function f_2										
n \ d	6	15	6	15	6	15	6	15	6	15
100	87.5	69.3	87.5	89.7	37.5	43.7	81.3	84.6	56.3	59.1
200	100	84.6	93.7	89.7	68.8	53.9	100	84.6	50	59.1
500	100	100	100	100	81.3	18.1	100	100	75	59.1
1000	100	100	100	100	93.8	7.9	100	100	68.8	59.1
function f_3										
n \ d	6	15	6	15	6	15	6	15	6	15
100	100	53.9	100	100	100	100	100	100	50	48.8
200	100	79.5	100	100	100	100	100	100	37.5	48.8
500	100	100	100	100	100	100	100	100	37.5	53
1000	100	100	100	100	100	100	100	100	37.5	48.8
function f_4										
n \ d	6	15	6	15	6	15	6	15	6	15
100	100	100	100	96.5	95	78.8	80	82.3	95	100
200	100	100	100	100	100	71.7	90	64.6	100	100
500	100	100	100	100	65	75.2	100	100	100	100
1000	100	100	100	100	65	78.8	100	100	100	100
function f_5										
n \ d	6	15	6	15	6	15	6	15	6	15
100	100	100	100	87.5	30	55	51	45	58	67.5
200	100	100	100	95	30	70	51	50	58	80
500	100	100	100	100	30	77.5	79	77.5	51	75
1000	100	100	100	100	30	75	100	95	72	77.5

Results in Table 3.1 show that the proposed method is competitive with `RRelief` and outperforms the other algorithms. On the other hand, it is clear how `COBAS` requires a sufficient number of training samples to produce a robust variable ranking; see, for instance, the results reported for functions f_3 and f_4 with $d = 15$ features, where the performance score increases with

the number of data points n.

Results on real-world datasets

Five real-world datasets are considered: `poland` ([56, 57], with $n = 1370$ and $d = 30$), `santafe` ([51, 93], $n = 10081$, $d = 12$), `housing` ($n = 506$, $d = 13$), `abalone` ($n = 4177$, $d = 8$) and `cpusmall` ($n = 8192$, $d = 12$)[3]. As prescribed in [14], raw time series data from `poland` and `santafe` have been transformed into regression data by using the last q values (respectively $q = 30$ and $q = 12$) as features.
In the current setting, `COBAS` has been compared with `RRelief` and the feature selection method `ELM-FS` ([14]); in particular, since the code of the latter algorithm is not publicly available, the results reported in this section are taken from [14], and refer solely to datasets `poland` and `santafe`. In order to compare the methods' performance, a nonlinear regressor (namely, `SVR`, [11, 23]) has been trained, and its prediction accuracy has been evaluated with a growing number of selected features; more in detail, each dataset has been divided into a training (70%) and a test set (30%), and the training process has been repeated ten times, on different data splits. Results depicted in Figure 3.1 (a)-(e) show how `COBAS` is either competitive or outperforms the other methods, and prove the quality of the feature ranking provided on medium scale datasets.

Computational time

The authors of [21] performed a comparison between `COBAS` and `RRelief` in terms of time needed to rank the features. They highlight the inefficiency of the approach: the time needed for training the `RBFN` is in itself one or two orders of magnitude larger than that of `RRelief`, while the scoring time is three orders larger. However, most of the operations required by `COBAS` are suitable to be implemented in a parallel way. The next section provides an implementation sketch, employing a state-of-the-art parallel/distributed platform and programming framework.

[3]Datasets `housing`, `abalone` and `cpusmall` have been obtained from `http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets`.

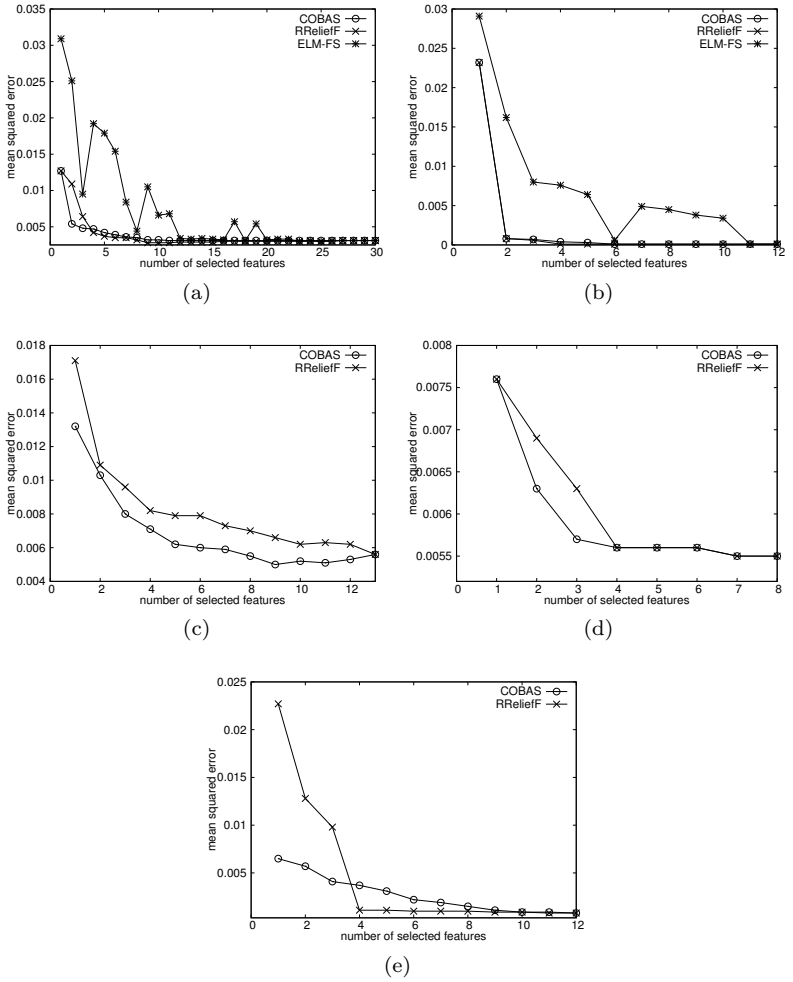


Figure 3.1: A comparison between `COBAS`, `RRelief` and `ELM-FS` on real-world datasets. Results are reported as a function of the mean square error achieved by the methods as the number of selected features increases.

3.3 A distributed implementation of COBAS using Pyomo and Spark

The goal of this section is to propose a parallel and distributed implementation of COBAS. This is achieved using Apache Spark and Pyomo.

3.3.1 Apache Spark

Iterative algorithms reuse intermediate data across multiple computations. This paradigm is common to many ML algorithms. Cluster computing frameworks like MapReduce ([37]) let users writing their code using a set of high-level primitives, providing transparent distribution of operations and fault tolerance. However, these frameworks lack abstractions for exploiting distributed (main) memory. Moreover, the only way they provide for reusing data from past computations is to serialize them onto a reliable storage system, incurring in overheads due to data replication and heavy I/O operations.
Resilient Distributed Datasets (RDD, [98]) is a technique which provides an in-memory abstraction, based on *coarse-grained* transformations, which applies the same operations to distributed data. This kind of operations do not require to share a state or replicate data among different machines: rather than logging the actual transformed data, RDDs provide fault tolerance by building a *lineage* of the transformations applied up to a certain time during the computation. By doing this, if a data partition is lost, it can be easily built from scratch by referring to its lineage.
RDDs can be handled by two types of distributed primitives. *Transformations* (like map, filter and join) convert an RDD into another by applying the same operation to all of the data contained within. *Actions* (like count or sum) are aggregated operations on RDDs which return a value to the main application. Note that, until an action is not applied, RDDs do not need to be materialized (if lost, they can be recomputed by looking at the lineage) and the corresponding transformations do not need to be actually applied: for this reason, transformations are called *lazy* operations.

Spark is the programming platform and framework which implements RDD. It runs across a computers' cluster and provides a distributed memory abstraction. Roughly speaking, all of the RAMs of the cluster members are seen as a single overall memory entity. Consider the following example. A user is interested in counting sql errors from a log file of large size, and has

a cluster of computers at his disposal. The first step is to load such file into the cluster's main memory: Spark provides the .parallelize() primitive to achieve this goal. By doing so, the log file is divided into small portions and shared across the cluster memory. Now, a first transformation is needed to extract those lines which start with the word "ERROR": the .filter() primitive can be used for this purpose. A new RDD is thus obtained (no new data is actually created by the platform, as the transformation just applied is nothing but a record in the RDD lineage): note that the same operation (checking if it starts with a given word) has been applied to each line of the log file. A new application of .filter() is then needed to extract error lines containing the world "SQL"; again, a new "shallow" RDD is created. Finally, the action .count() is applied to return the size of the sql errors RDD to the main application. At this point, all of the lazy transformations are actually performed, and instantly deleted by the system when the requested line count is ready. If the user is interested in saving an RDD for a future use (that is, to checkpoint a RDD in addition to log the corresponding transformation into the lineage), Spark provides a .persist() primitive to do so: by default, RDDs are persisted in the cluster's primary memory; when no more space is available, they are moved onto disk.
Spark is a multi-language platform: among the supported programming languages, it provides a set of primitives for Python.

3.3.2 Pyomo

Pyomo ([47]) is a collection of Python software packages for modeling structured optimization problems. Its high-level object-oriented programming interface provides a syntax similar to the notation commonly used in mathematical optimization. Pyomo can be used to define general *abstract* (symbolic) models and create *concrete* problem instances by providing the data to an abstract model. A model, its objective, constraints and variables are all Python objects, and concrete problems can be initialized using Python data. Pyomo allows the user to solve a problem instance by supporting a large number of commercial and open-source solvers; this interaction is handled with several interfaces, depending on the solver being used.

3.3.3 Implementation details

By reference to Section 3.2.4, it is easy to notice that all of the operations carried out for a given reference/target pair $(\mathbf{x}^p, y^q)$ are independent. For this reason we refer to a pair as the minimal data object referred to by `Spark`: the same bulk operations are applied parallelly to all of the pairs, and the latter are distributed across cluster's memory.
When a `Spark` user wants to transform all of the data contained in an `RDD` into a new one, by applying a 1-to-1 operation in parallel, the `.map()` primitive is the easiest to be used: by applying `.map()` with a specific operation $f(\cdot)$ attached to an `RDD` R, a new `RDD` R' is generated, where each element $r \in R$ will be mapped to a new element $r' := f(r) \in R'$. Another useful feature provided by `Spark` is to allow the user to *broadcast* an entity (such as a `Python` object), so that it can be referenced by all of the cluster's members.

Let us consider Algorithm 4: At line 1, the trained regression model is

Input: M, a training set $\{\mathbf{x}^i, y^i\}_{i=1}^n$ and a trained regression model $F(\cdot; \mathbf{w}^*)$
1: $\mathcal{P} \leftarrow$ build a problem sketch using $F(\cdot; \mathbf{w}^*)$
2: broadcast $\mathcal{P}$
3: $\{(\mathbf{x}^p, y^q)\}_{i=1}^M \leftarrow$ build pairs from training set
4: **for** each pair $(\mathbf{x}^p, y^q)$ **parallelly do**
5: $\quad \mathcal{P}^{(p,q)} \leftarrow$ fetch $\mathcal{P}$ and customize it with $(\mathbf{x}^p, y^q)$
6: $\quad \hat{\mathbf{x}}^{(p,q)} \leftarrow$ solve $\mathcal{P}^{(p,q)}$ by applying a GO algorithm
7: $\quad \mathbf{v}^{(p,q)} \leftarrow$ vector of features changed in $\hat{\mathbf{x}}^{(p,q)}$ wrt $\mathbf{x}^p$
8: $V \leftarrow$ sum all vectors $\mathbf{v}^{(p,q)}$
Output: Reorder features $i = 1, \ldots, n$ according to their scores in V

Algorithm 4: A distributed implementation of `COBAS` using `Pyomo` and `Spark`.

used to build a *sketch* of the GO problem (3.8); the latter is referred to as a sketch because the reference point $\mathbf{x}^p$ and the target output y^q are not yet specified. To this aim, `Pyomo` abstract model is used. At line 2, the problem sketch is broadcast, in order to be used by each machine in the cluster. At line 3, M pairs are generated from the training set.
From line 4 to 7, parallel operations are computed on the pairs. At line 5, the problem sketch is fetched and customized by each pair $(\mathbf{x}^p, y^q)$ – that is, a `Pyomo` concrete problem is built – to obtain problem (3.8), which we refer to as $\mathcal{P}^{(p,q)}$. At line 6, problem $\mathcal{P}^{(p,q)}$ is solved for each pair by a GO algorithm. From `Spark`'s point of view, each pair $(\mathbf{x}^p, y^q)$ is *mapped* (using the `.map()` primitive in a parallel way) into the optimum $\hat{\mathbf{x}}^{(p,q)}$ of its related GO problem. The `.map()` primitive is used again at line 7 to transform every

optimum $\hat{\mathbf{x}}^{(p,q)}$ in a boolean vector $\mathbf{v}^{(p,q)}$ of features changed with respect to the reference point $\mathbf{x}^p$:

$$\mathbf{v}_i^{(p,q)} = \begin{cases} 1 & \text{if } \hat{x}_i^{(p,q)} \neq x_i^p \\ 0 & \text{otherwise.} \end{cases} \quad i = 1, \ldots, d \tag{3.16}$$

Once that all boolean vectors $\mathbf{v}^{(p,q)}$ are computed, the `Spark` action `.sum()` is invoked at line 8 to aggregate them in order to obtain an overall scoring: if, among the M pairs, a feature has changed k times from its corresponding reference point, its score will be equal to k. Finally, features are reordered according to their score, and a ranking is returned.

3.3.4 Numerical experiments

Filter methods such as `RRelief` do not need a ML model to be trained to compute the feature ranking. Therefore, the efficiency of a wrapper method such `COBAS` cannot compete with that of `RRelief`. The aim of the implementation presented in the last section is to alleviate the computational burden of `COBAS`. Ideally, such scheme should be able to obtain a ranking which is of the same quality of that obtained with a serial implementation while spending a time (not taking into account that needed for training the regressor) which is linearly (and inversely) proportional to the degree of parallellism. Roughly speaking, if a parallel/distributed implementation would spend T seconds using K cores, it should ideally take $T/2$ seconds using $2K$ cores: in other words, in a perfect parallel scenario the *overhead* time is negligible.
To assess the efficiency of the proposed implementation, we have used the following setting: an `SVR` (an adaptation of `SVM` to regression, [81]) has been trained and input to the method, while a multi-start approach using `SNOPT` ([42]) as underlying local solver has been employed at line 6 of Algorithm 4 to solve (3.8). Moreover, we have built a cluster of 8 machines (with the same hardware features), each providing 8 independent physical cores. We have considered the five artificial dataset presented in Section 3.2.5, with $M = 1000$ pairs and $n = 15$ features, and compute the average ranking time (on ten different dataset instances) as a function of the cores employed. Consider Figure 3.3.4: the x-axis reports the number n_{cores} of employed cores, which have been added to the cluster in order to incrementally double the degree of parallelism (when $n_{cores} < 8$, only one cluster member is used

and the implementation is parallel without being distributed). The y-axis reports the ratio of times needed respectively by the distributed and serial ($n_{cores} = 1$) implementations of `COBAS`. Figure 3.3.4 depicts a comparison

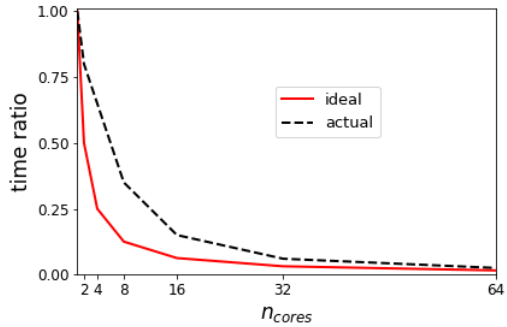


Figure 3.2: Ratio of times needed by the parallel (numerator) and serial implementations of `COBAS`: the red continuous line depicts an ideal trend, while the black dashed one shows the actual trend achieved by the parallel implementation.

between an ideal trend (continuous red line) – where the time needed by `COBAS` halves when n_{cores} is doubled – and the actual trend achieved by our implementation (black dashed line): it can be easily noticed how, asymptotically, the actual trend approaches the ideal one, which proves the efficiency of the proposed parallel/distributed scheme.
For our experiments, we used `hdfs` (the distributed file system provided by `Hadoop`, [37]), and respectively `Spark 2.2.0` and `Pyomo 5.2`.

3.4 Conclusion

Feature ranking is an important brick of a decision support system. Moreover, its outcome can be useful in itself, without necessarily be employed to feed a ML model. The literature does not provide a clear definition of feature relevance. In addition, many feature ranking methods have the common trait of analyzing one feature at a time when building an overall scoring. This can be a drawback, since a feature that seems useless when used alone can be of high importance when employed along with one or more other variables.
The approach presented in this chapter builds the concept of feature relevance by solving a well-defined GO problem, and is multivariate, that is, all of the variables are taken into account simultaneously in the ranking process.

Experiments on both synthetic and real-world dataset have been carried out, and prove how `COBAS` is either competitive or outperforms well-estabilished feature ranking methods.
On the other hand, the computational time of the proposed technique is up to three orders of magnitude larger than that of a filter method like `RRelief`. However, a major part of the operations required by `COBAS` are independent, and can be applied in parallel. For this reason, an implementation scheme is proposed, which uses the state-of-the-art programming paradigm and platform provided by `Apache Spark`. The latter alleviates the computational effort, and renders `COBAS` scalable and applicable to problems of larger scale.

Chapter 4

Learning from Large Scale Global Optimization using Geometrical Features

Seeking for the optimum of a large scale multimodal GO problem can be very time consuming. Geometrical problems of this kind, such as arranging circular-shaped objects in a container of fixed size or predicting the structure of a molecule, have the further issue of having an infinite number of equivalent solutions, yielded by transformations (such as rotations) of given configurations.
In this chapter we elaborate on the idea of representing solutions through compact feature vectors, which allows them to be compared and clustered from a geometrical rather than a pure numerical perspective. We integrate this strategy in a GO algorithm, and exploit it to prune redundant lines of search, while focusing the computational effort into the most promising ones. Moreover, we show how enhancing a GO algorithm with the experience from its past computations and making a smart use of the information provided can lead to significant performance improvements.[1]

[1]Part of the material of this chapter has been presented in 2016 at the *14th EUROPT Workshop on Advances in Continuous Optimization.* For what concerns the part dealing with prediction of atomic clusters structure, a work has been submitted to the *Journal of Chemical Physics*, and it is under review at the time of writing.

4.1 Introduction and related work

4.1.1 Making GO smarter by learning from local search information

Let us consider the problem of efficiently approximating the global optimum of a large scale optimization problem with many local optima. A common heuristic approach to solve this kind of problems is that of repeatedly performing local searches in the feasible set in order to visit good local optima, while looking for the global one. Usually, much of the computational effort is wasted: in fact, only the best point is returned by these techniques, as an estimation of the global optimum. Among the many approaches that have been proposed to tackle large scale GO problems, a rather basic but still successful one is the algorithmic scheme of *Monotonic Basin Hopping* (`MBH`, see [90]), also known as *iterated local search* in GO literature. At each iteration, `MBH` perturbs the current point and attempts to improve the objective value carrying out a local search from the new position, moving to the reached point in case of success. Its parameters are ϕ, the radius of the neighbourhood (e.g., a box) within which points are perturbed, a number of iterations K and a maximum amount $max\_no\_improves$ of consecutive non improving iterations (thus, local searches) that `MBH` is allowed to perform: Algorithm 5 shows the pseudocode of `MBH`.

Input: f, K, $max\_no\_improves$, ϕ, $\mathbf{x}_{start}$
1: $\mathbf{x}_{curr} \leftarrow \mathbf{x}_{start}$
2: $no\_improves \leftarrow 0$
3: **for** $k = 1, 2, \ldots, K$ **do**
4: $\quad \mathbf{x}' \leftarrow$ perturb $\mathbf{x}_{curr}$ in a neighbourhood of radius ϕ
5: $\quad \mathbf{x}'' \leftarrow$ solve locally from $\mathbf{x}'$
6: $\quad$ **if** $f(\mathbf{x}'') > f(\mathbf{x}_{curr})$ **then**
7: $\quad\quad \mathbf{x}_{curr} \leftarrow \mathbf{x}''$
8: $\quad\quad no\_improves \leftarrow 0$
9: $\quad$ **else**
10: $\quad\quad$ **increment** $no\_improves$
11: $\quad\quad$ **if** $no\_improves > max\_no\_improves$ **then** break
Output: $\mathbf{x}_{curr}$

Algorithm 5: `MBH`

It is easy to notice that such a GO scheme has no memory of the past computations, that is, every trial of `MBH` does not employ any information from the

previous ones. This is because only the best known prediction is returned by each run, and no use is made of past executions.

Two main strategies are followed to make a smarter use of the information collected during the search: *population-based* methods (see [44], [4] and [3]), in which several runs of an algorithm are performed simultaneously and communicate with each other aiming to perform a coordinate search, and `LeGO` (*Learning for Global Optimization*, see [22]). Given an optimization problem

$$\min_{\mathbf{x} \in S \subseteq \mathbb{R}^n} f(\mathbf{x}), \tag{4.1}$$

the authors of [22] call $G(\cdot)$ a generic routine to randomly generate starting points, and $R(\cdot)$ a *refinement* procedure, such as an expensive local optimization, which maps an input $\mathbf{x}$ into another point in the feasible space, which objective value has at least the same quality of $\mathbf{x}$; the cost of R is assumed to be dominant on that of G. The `LeGO` framework works in the following way: a gathering phase is carried out, that is, some optimizations are performed to mark starting points $\mathbf{x}_k$ with a label $y_k = +1$ if $f(y_k)$ is below a threshold (which can be obtained from the knowledge of the problem or calibrated by means of a cross-validation step) and $y_k = -1$ otherwise; positively labeled points – the positive class – are therefore "good" starting configurations. This ground truth is then used to train a `SVM`, with the aim of learning the unknown relationship between a starting point and its objective value after an expensive (local) optimization, and used to evaluate and prune starting configurations of future runs.
`LeGO` is one of the first examples of interplay between GO and ML, and highlight how GO algorithms can be enhanced by their local search history. In Section 4.2 we provide the algorithmic scheme of `MBH` with the knowledge from its past trials, and show how this strategy is capable of saving large amounts of local search.

4.1.2 Two geometrical problems

We call a problem *geometrical* when its variables, constraints and objective have a clear physical meaning. Let us consider, for instance, the *2D disk packing problem* ([44], [3], [50], [33]). An integer N is given as well as a "container", which might be the unit square, the unit circle or a rectangle

with unit longest edge and specific shortest one: it is then required to find the position, within the container, of N non overlapping identical disks with maximum radius. Figure 4.1 depicts two solutions of the *Circle-IN-A-Circle* (CINAC) packing problem. Colors are related to the number of contacts; for instance, light brown disks are those having two contacts with other circles and a further one with the container border. Pink disks, the *rattlers*, have at least one degree of freedom. In this scenario, the $2N$ variables are the circles' coordinates; two contraints arise from the request of non overlapping disks lying within the container border, while disks' radius is the objective function to be maximized:

$$\begin{gathered} \max r \\ \sqrt{(x_{i1}-x_{j1})^2+(x_{i2}-x_{j2})^2} \geq 2r \quad i,j \in [1,N],\ i<j \\ \sqrt{(x_{i1}^2+x_{i2}^2)} \leq 1-r \quad i \in [1,N]. \end{gathered} \tag{4.2}$$

The above problem is equivalent to the *maximum dispersion* problem, which is slightly easier to solve:

$$\begin{gathered} \max d \\ \sqrt{(x_{i1}-x_{j1})^2+(x_{i2}-x_{j2})^2} \geq d \quad i,j \in [1,N],\ i<j \\ (x_{i1}^2+x_{i2}^2) \leq 1 \quad i \in [1,N]. \end{gathered} \tag{4.3}$$

A further geometrical large scale application is that of predicting the optimal structure of an atomic cluster, which is an active field in molecular biology and computational chemistry. Let us consider the problem of finding the minimum energy configuration of a cluster of N atoms in $\mathbb{R}^3$, interacting through a pair potential. By this we mean that we would like to solve the unconstrained GO problem

$$\min_{\mathbf{x}_1,\ldots,\mathbf{x}_N \in \mathbb{R}^3} \sum_{i=1}^{N} \sum_{j=1}^{i-1} V(\mathbf{x}_i, \mathbf{x}_j), \tag{4.4}$$

where $V(\cdot,\cdot)$ is a pair-potential function modeling the interaction of two atoms as a function of their distance. The most frequently encountered

models for the pair potential are the Lennard-Jones'

$$V(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^{-12} - 2\|\mathbf{x}_i - \mathbf{x}_j\|^{-6} \tag{4.5}$$

and the Morse potential

$$V(\mathbf{x}_i, \mathbf{x}_j) = D(1 - \mathrm{e}^{\rho(r - \|\mathbf{x}_i - \mathbf{x}_j\|)})^2 - 1. \tag{4.6}$$

In (4.6), r is the optimal (i.e., the equilibrium) distance between $\mathbf{x}_i$ and $\mathbf{x}_j$, while D and ρ control, respectively, the depth and the width of the unique local minimum of $V(\mathbf{x}_i, \mathbf{x}_j)$; more in detail, the larger is ρ, the tighter is the minimum well. We can rescale the model by setting both D and r to be equal to 1, obtaining:

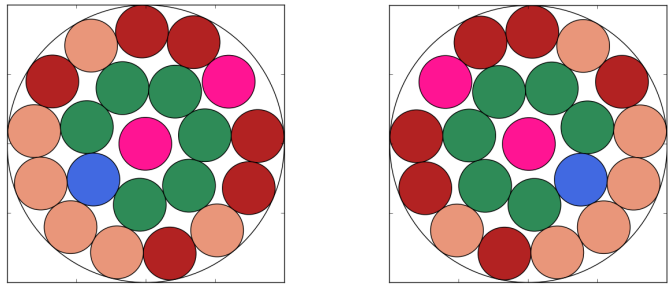
$$V(\mathbf{x}_i, \mathbf{x}_j) = (1 - \mathrm{e}^{\rho(1 - \|\mathbf{x}_i - \mathbf{x}_j\|)})^2 - 1. \tag{4.7}$$

The literature on Lennard-Jones and Morse cluster optimization is huge. Classical references in the chemical-physics literature are [90], [39] and [65]. These problems are very well-known in the GO literature, as finding a low energy configuration is extremely challenging: indeed, it is widely believed that the number of local optima of the potential energy function grows exponentially with the number of atoms N. For what concerns Morse clusters, the difficulty of the underlying GO problems increases as ρ becomes larger, with the extreme case in which $\rho \to \infty$ being combinatorial. The case $\rho = 6$ is considered as pretty equivalent to the Lennard-Jones case, and the putative optima known are quite similar to those obtained in the latter case.

4.1.3 Equivalent solutions and geometrical features

When dealing with geometrical problems, GO algorithms' job is made harder by the existence of multiple (often infinite) *equivalent* solutions. Figure 4.1 depicts two CINAC configurations. We can easily notice that (b) is obtained from (a) by a 90° counterclockwise rotation: (a) and (b) thus overlap perfectly, and share the same objective value. Despite being *equivalent* for what concerns the packing problem, (a) and (b) are clearly very different from the point of view of a GO algorithm: in fact, disks' centers (the variables) are arranged differently. Our aim, then, is to map configurations in a reduced

space induced by an array of geometrical properties, or *features*: we call *fingerprint* such representation. This compact geometrical model, in addition to being invariant to trivial transformations, allows different solutions to be compared and grouped together. For what concerns cluster structure prediction, the idea of classifying a configuration by means of its geometrical properties is not new ([40], [82]). In [29], a compact descriptor is used alongside a *tabu search* ([43]) heuristic for predicting the structure of (actually very small) Lennard-Jones clusters.
In the remainder of this section, we introduce the geometrical descriptors we will use throughout the chapter to describe packings and cluster configurations.



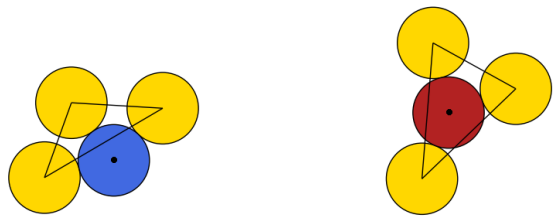
(a) (b)

Figure 4.1: Rotations (but also mirrorings, flips and disk permutations) of a packing configuration (a) yield an equivalent one (b). Colors are related to the number of contacts; for instance, light brown disks are those having two contacts with other circles and a further one with the container border. Pink disks, the *rattlers*, have at least one degree of freedom.

Packings

Let us consider a packing solution. We can easily determine if two circles are *neighbors*, that is, they are in contact with each other, or if a circle is leaning against the container border by computing Euclidean distances. In two dimensions, a circle can have 3 down to 0 neighbors if (also) touching the container border, 6 (the so-called *kissing number*) down to 0 if not.
Pink disks in Figure 4.1 are called *rattlers* (see [67]): they have at least one degree of freedom, thus not being part of the solid structure of the given

configuration. Once that the contact information has been computed, we need to determine whether a circle is a rattler or not: in fact, if it has one or more degrees of freedom, some of its contacts might be spurious. To this end, we introduce a very intuitive check: we call a circle a non-rattler if its center strictly lies inside the convex hull of its neighbors' centers. Figure 4.2, (a) and (b) depict respectively a rattler and a non-rattler.



(a) (b)

Figure 4.2: Three contacts: a rattler (blue) and a non-rattler (red). We can notice that the blue circle's contacts in (a) are spurious. We can claim that a disk is a non-rattler just by looking at its contacts only when it is surrounded by at least 5 neighbors or 3 neighbors plus the container border. Similarly, we are sure we are dealing with a rattler if it has less than 3 neighbors or just one neighbor plus the border. In all other cases, the depicted disambiguation is needed.

Another geometrical invariant we have choosen to describe packings is the *shape factor* (see [39]). This measure gives an idea of how much a packing is circular-shaped, reaching its maximum at 1 when the centers of the most external disks lie on a circumference. More in detail, let $\mathbf{A}$ be the $2 \times N$ centers' matrix. In a 2D scenario, the shape factor is defined as the ratio of the smallest and the largest of the two eigenvalues of $\mathbf{A}^T\mathbf{A}$.

Here below is the complete list of features we have used to describe CINAC configurations:

1 shape factor
2 # of rattlers
3-5 # of disks with 3/4/5 contacts
6-7 # of disks with 2/3 contacts +border.

Atomic clusters

The concept of neighbor introduced for a packing configuration, though slightly different, can be easily extended to the cluster prediction scenario. In particular, the *coordination number* c_i ([40]) of a given atom $\mathbf{x}_i$ is the number of atoms located within a sphere, centered in $\mathbf{x}_i$, of prespecified radius; for standard Morse and Lennard-Jones clusters we use 1.15. As suggested in [29], the average coordination number, as well as its standard deviation and its minimum and maximum values, are good candidates for describing the physical structure of a molecole. The authors of [29] also make use of two descriptors which quantify the departure from a spherical shape, similarly to the shape factor used for the packing structure, towards a more prolate or oblate sctructure. Let I_a, I_b and I_c be the moments of inertia of a given cluster configuration, and assume $I_a \geq I_b \geq I_c$. The descriptors ζ and η are defined as:

$$\begin{aligned} \zeta &= \frac{(I_c - I_b)^2 + (I_b - I_a)^2 + (I_a - I_c)^2}{I_a^2 + I_b^2 + I_c^2} \\ \eta &= \frac{2I_b - I_a - I_c}{I_a}. \end{aligned} \tag{4.8}$$

For what concerns the shape factor, a 3D extension is needed to describe atomic clusters. In particular, let λ_1, λ_2 and λ_3 be the eigenvalues of $A^T A$, where $\lambda_1 \geq \lambda_2 \geq \lambda_3$. The 3D shape factor is defined as

$$S = \Big(\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1}\Big). \tag{4.9}$$

An additional feature we have picked to describe atomic cluster structures in a compact way is the *strain energy* ([71]). Let us refer to (4.7): in an ideal scenario, each pair of neighbors $\mathbf{x}_i$ and $\mathbf{x}_j$ should be distant exactly $r = 1$ from each other, yielding a contribution to the overall potential energy equal to -1; a larger or smaller distance would produce a suboptimal potential $V(\mathbf{x}_i, \mathbf{x}_j) > -1$. The strain energy is defined as

$$E_{\text{strain}} = \sum_{i<j \,:\, \|\mathbf{x}_i - \mathbf{x}_j\| \leq 1+\epsilon} V(\mathbf{x}_i, \mathbf{x}_j) + m, \tag{4.10}$$

where m is the cardinality of the set of neighbor pairs in the cluster. Note that $E_{\text{strain}} = 0$ if atoms in each pair have an optimal distance from each other.

The *Bond-orientational Order Parameters* (BOPs) are used in [82] and [92] to characterize the neighboring environment of an atom. To this aim, the contribution of an atom $\mathbf{x}$ to the molecule's BOPs is modeled by a set of spherical harmonics associated with each atom lying in the surrounding of $\mathbf{x}$. In particular, the authors call *bond* (not related with chemical ones) a straight link between an atom $\mathbf{x}$ and each of its neighbours $\mathbf{x}'$. Let $\mathbf{r}$ be the midpoint of this link in some reference coordinate system $\mathcal{P}$, and define the quantities

$$Q(\mathbf{r}) := Y(\theta_{\mathcal{P}}(\mathbf{r}), \phi_{\mathcal{P}}(\mathbf{r})), \tag{4.11}$$

where $\theta_{\mathcal{P}}$ and $\phi_{\mathcal{P}}$ are the polar angles of the bond measured in $\mathcal{P}$ and $Y(\theta, \phi)$ are spherical harmonics. BOP is then defined averaging Q over the $m_{\mathbf{x}}$ neighbours of $\mathbf{x}$:

$$BOP(\mathbf{x}) := \frac{1}{m_{\mathbf{x}}} \sum_{\text{neighbors of } \mathbf{x}} Q(\mathbf{r}). \tag{4.12}$$

A complete list of the geometrical features we have used is the following:

1-4 coordination number's mean, standard deviation, maximum and minimum values

5-6 prolate and oblate structure indicators

7-8 3D shape factors

9 strain energy

10-13 bond order parameters.

4.2 An *experienced* MBH

In this section, a technique is proposed to enhance the structure of MBH by making a smarter use of the information from past executions. The geometrical descriptor introduced in Section 4.1.3 is used to represent past solutions in a compact way. Section 4.2.1 reports the details of the proposed

scheme. Experiments and numerical results are reported in Section 4.2.2. Finally, in Section 4.2.3 we make a few remarks on the proposed approach and hint at possible improvements.

4.2.1 Methods details

Our aim is to enhance `MBH` with the knowledge of optima that have been reached in the past trials; the idea is very straightforward, and can be looked at as a two steps procedure:

1. gather fingerprints of past `MBH` outputs (i.e., local optima);
2. stop subsequent trials when reaching a configuration with a "known" fingerprint.

The insertions we need to extend `MBH` with, in order to implement the above scheme and make the method *experienced* (we called `ExpMBH` the enhanced version) are highlighted in Algorithm 6.

Input: f, K, $max\_no\_improves$, ϕ, $\mathbf{x}_{start}$, $\mathcal{F}$, $fingerprint\_set$
$\mathbf{x}_{curr} \leftarrow \mathbf{x}_{start}$
$no\_improves \leftarrow 0$
$stopped \leftarrow False$
for $k = 1, 2, \ldots, K$ **do**
  $\mathbf{x}' \leftarrow$ perturb $\mathbf{x}_{curr}$ in a box of side ϕ
  $\mathbf{x}'' \leftarrow$ solve locally from $\mathbf{x}'$
  if $f(\mathbf{x}'') < f(\mathbf{x}_{curr})$ **then**
    $\mathbf{x}_{curr} \leftarrow x''$
    if $\mathcal{F}(\mathbf{x}_{curr}) \in fingerprint\_set$ **then**
      $stopped \leftarrow True$
      break
    $no\_improves \leftarrow 0$
  else
    increment $no\_improves$
    if $no\_improves > max\_no\_improves$ **then** break
if not $stopped$ **then add** $\mathcal{F}(\mathbf{x}_{curr})$ **to** $fingerprint\_set$
Output: $\mathbf{x}_{curr}$, $fingerprint\_set$

Algorithm 6: `ExpMBH`. The insertions with respect to the plain method are highlighted.

`ExpMBH` needs to know how to compute the fingerprint $\mathcal{F}$ of each reached configuration. In addition, a set of fingerprints of known local optima is

given as an extra input parameter. At lines 9-11, ExpMBH computes the fingerprint of the current point $\mathbf{x}_{curr}$ and checks if the resulting geometrical representation is the same of some optimum that has been already returned by a past MBH trial. At line 16, if an unseen solution has been computed, it is included in the experience of ExpMBH in the form of its fingerprint, and the updated *fingerprint_set* is returned as additional output, to be used by subsequent trials. Figure 4.3 depicts an illustrative example of experienced trial, working on the 62 CINAC packing problem.

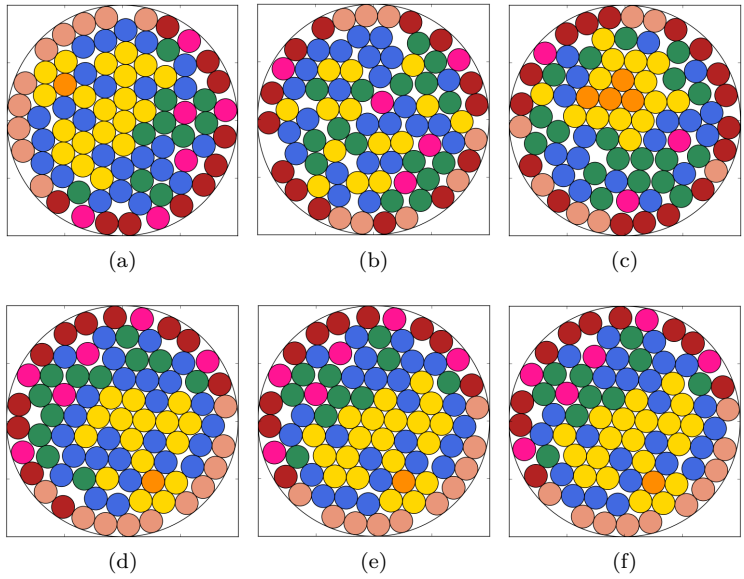


Figure 4.3: An illustrative trial of ExpMBH on the 62 CINAC packing problem.

Figure 4.3 (a) depicts a known optimum $\mathbf{x}^*$, whose fingerprint (following the signature introduced in Section 4.1.3) is $\mathcal{F}(\mathbf{x}^*) = [0.97, 6, 25, 20, 1, 10, 10]$. Let us start a brand new ExpMBH trial, where $\mathcal{F}(\mathbf{x}^*)$ represents the experience of the method; (b)-(d) show the packing configurations obtained at iterations 0, 4 and 10, in which the radius is improved: none of the fingerprints of these configurations is equal to $\mathcal{F}(\mathbf{x}^*)$. The next improvement, (e), takes place at iteration 13, yielding a packing having the same fingerprint of $\mathbf{x}^*$ (it is easy to notice that (e) is just a rotation of $\mathbf{x}^*$): the experienced trial stops here. Finally, (f) shows the configuration that would have been reached by MBH

(that is, without early stopping the trial) after $max\_no\_improves = 100$ local searches: nothing has changed with respect to (f), making the last 100 iterations useless.

4.2.2 Experiments

In this section we describe the experiments we have carried out to assess the efficiency of our strategy. We show how our modified version of `MBH` is capable of saving a large amount of local searches, compared to the plain algorithm. The scenario in which we have tested our method is that of the CINAC packing problem described in Section 4.1.2.
Let us start by introducing our experimental setting. A measure that is widely used in the literature to compare algorithms' efficiency is the number of local searches they need, in average, to *hit* (that is, to reach) a known putative global optimum within a given number of trials:

$$\bar{L} = \frac{\#\text{local searches}}{\#\text{hits}}. \tag{4.13}$$

We define the *gain* G as the percentage of local searches we have saved by employing our method rather than the plain version of `MBH`:

$$G = 100 * (1 - \frac{\bar{L}_{\texttt{ExpMBH}}}{\bar{L}_{\texttt{MBH}}}). \tag{4.14}$$

We have run a number of `MBH` trials: everytime a trial have bumped into a known local optimum (in the form of its fingerprint), thus being supposed to stop according to our method, it has been marked, keeping track of the amount of local searches it needed to reach that configuration, without actually stopping it. We did so in order to gather $\bar{L}$ for both `MBH` and `ExpMBH`. Following the notation of Algorithms 5 and 6, we set the number of iterations $K = 1000$ and $max\_no\_improves = 100$, and carried out 1000 `MBH` trials on instances from 30 up to 100 circles-in-a-circle.

Figure 4.4 depicts the gain achieved on these instances within a different number of trials. In particular, (a), (b), (c) and (d) refer respectively to the gain obtained within 100, 250, 500 and 1000 trials and show how the efficiency of the method increases with the number of past trials.
Note that our strategy is not deterministic: its evolution, in fact, depends

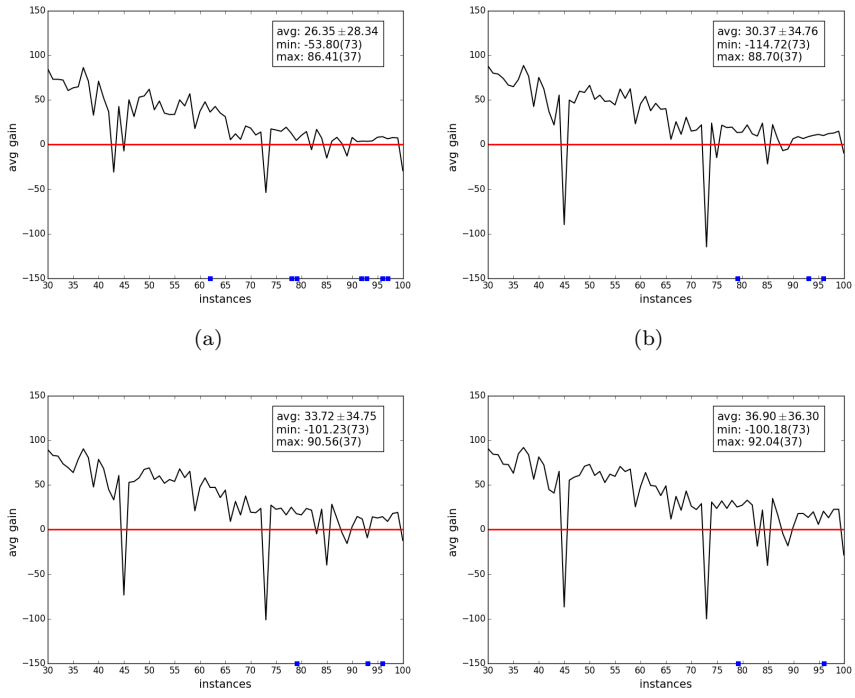


Figure 4.4: Average gain within 100 (a), 250 (b), 500 (c) and 1000 (d) MBH trials on packing instances from 30 up to 100 circles; results are averaged on 10 trials permutations. The legend indicates gain's mean and standard deviation on all the instances alongside the best and worst instance., which is shared between the latter applications Blue squares on the x-axis mark those instances in which MBH has not been able to hit the global optimum: in these cases, the best value obtained by MBH is used to compute $\bar{L}$ for the compared methods.

on the order in which trials are computed; therefore, results in Figure 4.4 are averaged on ten different trials permutations. Is it easy to see how enhancing MBH with the memory of the past local optima lets the algorithm save a large amount of local searches; in particular, within 1000 trials, the average gain is 36.9%. Our method impact is more visible on smaller instances, due to the greater amount of possible local optima in a larger instance; however, the more trials are carried out, the less this gap is evident: we can appreciate

how the whole gain curve rises while more trials are gradually used to feed ExpMBH.

Let us finally consider those instances (in particular 45 and 73 circles) where our method fails, leading to a negative gain: this happens because one or more configurations have been wrongly marked as local optima, making ExpMBH to stop during subsequent trials that would have led to the putative global optimum (that is, to a hit). In general, the more accurately a fingerprint is able to describe a local optimum, the less promising trials are stopped, leading to a higher number of hits; on the other hand, a "relaxed" way of detecting known configurations yields a large saving in terms of local searches: a tradeoff is thus needed.

4.2.3 Remarks on the presented approach

Two downsides of the proposed scheme can be highlighted. The first lies in the nature of MBH. What we are trying to achieve by checking the geometrical structure of the solutions generated throughout the execution of MBH is to possibly early stopping the method and save some computational effort. By doing so we give for granted that a line of search who has reached a known local optimum (that is, of which we have already observed the fingerprint in the past) won't be able to escape from the basin of attraction of that minimum. Actually, one of the strong points of MBH is its capability of "hopping", that is, escaping from a low quality valley down to a better one; this is due to the random perturbations carried out by MBH. This behaviour, while being a good feature when using MBH as a global method, does not accurately fit our scheme. Indeed, marking a line of search as "redundant" and stopping the execution might wrongly prune a promising exploration. In other words, it is not an easy task to predict the outcome of the current MBH search (whether or not it will get stuck in a known solution). As we will see in the next section, using a smart global scheme (which gathers information from the past and employs geometrical decriptors for representing known configurations) with a purely *local* optimizer is a more robust design choice, being able to increase the gain obtained in terms of local search without loosing any information.

Another weak point of the proposed approach lies in the "exact" checking that we carry out on the current solution in order to detect a known local optimum and possibly stop the search. By doing so, we are just grouping equivalent solutions, that is, we are employing our compact representation

solely to (early) detect trivial transformations of a known solution. Actually, our geometrical descriptor can be used in a broader way: in the next section we will introduce a grouping technique, and use fingerprints of known solutions to build *clusters* of "similar" configurations.

4.3 Working with *clustering* methods in a space of geometrical features

4.3.1 Clustering methods

In the early GO literature the idea of *clustering* has been a winning one for several years. Such methods aimed at early abandoning an expensive local search, by estimating that its continuation would have led to an already observed local optima. In recent years these methods have been abandoned, mainly for two reasons. Firstly, in order to cluster points (by means of closeness in the space of variables) in a meaningful way, a sufficiently large sample needs to be observed. The size of such sample grows exponentially as the dimension of the problem to be solved increases, thus making the idea of clustering useless for problems of even a few tens of variables. Another cause of the disappearance of clustering methods was that local optimization, which has evolved in recent years, provides fast and reliable methods, also for large scale problems. Thus, in modern GO methods, although the number of local searches performed is still the preferred complexity measure (as the time devoted to local descent dominates the overall computational cost), most approaches assume that, once a starting point is generated, a local search needs to be performed and run until convergence.

In this section we would like to propose a rediscovery of clustering methods for large scale optimization problems. The main idea is that this technique can be quite easily applied also to large scale problems, once a mapping from the solution space to a low-dimensional *feature* space is defined. This way, clustering can be performed in such space, where the decision whether to start a complete local search is also taken. In order to map solutions to a feature space, context information needs to be included. For this reason, the proposed approaches is a general one, and should be tailored to specific problem families. In the following we will demonstrate its effectiveness and efficiency in solving well-known and hard optimization problems in the field of atomic cluster potential energy minimization.

4.3.2 Multi-level single-linkage

Among GO algorithms, *clustering*[2] methods aim at reducing the number of local searches needed by the naïve *multi-start* (`MS`) approach by generating a non homogeneous sample of observations (*clusters*), with higher density around areas ideally coinciding with local minima's zones of attraction. If this biased generation is achievable, only one local search needs to be started from a representative (*seed*) of each cluster. Consider the GO problem

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\quad \mathbf{x} \in S \subset \mathbb{R}^n, \end{aligned} \tag{4.15}$$

where we assume that the feasible set S is a box $[\ell, u]$ with $\ell_j < u_j, j = 1, \ldots, n$. We might even assume that, after suitable scaling, the problem is defined in the unit hyper-cube:

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\quad \mathbf{x} \in [0,1]^n. \end{aligned} \tag{4.16}$$

One of the most basic GO algorithms is `MS`, which proceeds by randomly choosing a point in the feasible set and starting from there a local descent by means of a standard local optimization algorithm. `MS` is notoriously very inefficient, one of the reasons being that, frequently, the same local minimum is observed several times, starting from different points belonging to the same region of attraction. There exist many alternative GO methods (see, e.g., [66]), most of which try to improve over `MS` by suitably deciding how to choose a "promising" starting point for a computationally expensive local search.

One of the most interesting methods within this class is the *Multi-Level Single-Linkage* (`MLSL`), introduced in [76]. An high level description of this method is given by Algorithm 7. `MLSL` is structured in four steps, which are carried out at each iteration; an overall sample is kept throughout the GO process. At the beginning of the k-th iteration, `MLSL` generates a *batch* of

[2]The term *clustering* here has nothing to do with atomic clusters. Throughout the paper, the term cluster will be used both for atomic clusters as well as for a cluster of points in a generic Euclidean space. It should be clear from the context which of the two interpretations is the correct one.

Input: $\tau(\cdot)$, n_{batch}, n_{steps}, K
1: Overall sample $O \leftarrow \emptyset$
2: **for** $k = 1, 2, \ldots, K$ **do**
3: $X_{batch} \leftarrow$ (uniformly) generate n_{batch} points in $[0,1]^n$ ▷ (sampling)
4: $X_{concentrated} \leftarrow$ perform a few (n_{steps}) steps of a local descent algorithm from each seed in X_{batch} ▷ (concentration)
5: $O \leftarrow O \bigcup X_{concentrated}$
6: $X_{selection} \leftarrow \emptyset$
7: **for** $X \in O$ **do**
8: **if** $\forall Y : f(Y) \leq f(X) \Rightarrow \|X - Y\| \geq \tau(k)$ **then**
9: $X_{selection} \leftarrow X_{selection} \cup X$ ▷ (selection)
10: $X_{fully\_optimized} \leftarrow$ perform a full local optimization from each seed in $X_{selection}$ ▷ (full optimization)
11: $O \leftarrow O \bigcup X_{fully\_optimized}$
Output: best solution in O

Algorithm 7: `MLSL`

n_{batch} seeds (*sampling* step). During a *concentration* step, a truncated local optimization is started from each seed, in order to obtain a non uniform sample which might then be grouped in clusters; once concentrated, the seeds are added to the overall sample. Then, only a selection of seeds in the overall sample is fully optimized by a local optimization algorithm (*selection* and *full optimization* steps).

The main idea behind `MLSL` is that, thanks to the above four phases, it is often possible to avoid exploring each basin of attraction more than once. In this sense, the overall sample's members which are selected for being fully optimized are the cluster best representatives: ideally, denoting by $\mathbf{x}_c^\star$ the local optimum reached by starting a full local search from a point in $X_{selection}$, none of the other cluster members, if optimized, should lead to a local optimum which is better than $\mathbf{x}_c^\star$. In other words, all of a cluster's members should belong to the zone of attraction of $\mathbf{x}_c^\star$. What is meant by "cluster" in this context is the following. We might think of a directed graph in which nodes correspond to points in the concentrated sample and a directed arc connects a point $\mathbf{y}$ to a point $\mathbf{x}$ if

$$\begin{aligned} \|\mathbf{x} - \mathbf{y}\| &\leq \tau \\ f(\mathbf{x}) &\leq f(\mathbf{y}). \end{aligned} \tag{4.17}$$

Directed paths will be observable in such a graph, and those nodes having

no outbound arcs will correspond to the selected points, from which to carry out a complete local search. Borrowing from the chemical-physics literature, these points can be seen as "funnel bottoms", where funnels correspond to points in the concentrated sample from which a directed path leads to a specific point. The above criterion corresponds to the test used in the algorithmic scheme (lines 7-9 of Algorithm 7) in order to select candidate points for further optimization.

The threshold $\tau(k)$ should take into account the probability that two points uniformly sampled in a box are close enough; this probability, of course, depends on the cardinality of the sample. A common choice for τ is the following:

$$\tau(k) := \frac{1}{\sqrt{2\pi}} \left(\frac{\log(bk)}{bk} \sigma \Gamma \left(1 + \frac{n}{2} \right) \right)^{\frac{1}{n}}. \tag{4.18}$$

In the above threshold, $\Gamma(\cdot)$ is the gamma function (a generalization of the factorial), b is the batch size (so that bk is the number of seeds generated within iteration k) and σ is a parameter; Equation (4.18) assumes that all samples belong to $[0, 1]^n$, although the threshold might be easily generalized to different hyper-rectangles. This formula was proposed in [76], were it was also proved that, by suitably choosing the parameter σ, some interesting properties can be obtained. As an example, it was shown that, for $\sigma > 2$, the probability of starting a new local search goes to zero as the iteration number increases; also, with $\sigma > 4$, the expected total number of complete local searches performed remains finite even if the algorithm is run forever.

At each iteration, the whole sample is "clustered" from scratch; note that, in this scheme, since the threshold is decreasing, a solution previously assigned to a cluster might be reassigned to a different one: this way points are not definitely marked as belonging to the same group. The selection step is followed by a full local optimization of the best point of each cluster; of course, points which have already been fully optimized during a previous iteration are not handled again at this step. In Section 4.3.4 we will adapt the above formulation to the goal of predicting atomic cluster structures.

Clustering GO methods, although very successful in the past, have eventually been abandoned. The main reason behind this trend is that those methods displayed excellent performance only for low-dimensional GO problems. In fact, when the number of variables exceeds, say, $n = 10$, the sample

size required in order to be able to build meaningful clusters become prohibitively large. Our aim is to propose a re-discovery of these GO methods as good candidates for solving large scale *structured* GO problems. By "structured" we mean that we are dealing with GO problems coming from some realistic application which gives us the possibility of extracting relevant features from each feasible solution. The idea in this paper is to apply the GO clustering criterion in the space induced by such features, instead of applying it directly to the original problem (and its optimization variables).

4.3.3 Cluster surface smoothing

In [30] a geometrical class of clusters (such as icosahedral or decahedral) is called a *motif*. The authors relate motifs to funnels of the Potential Energy Surface (PES): each funnel might contain a huge amount of basins of attractions and sub-funnels, but all of them belong to the same geometrical class. In other words, the PES is seen as a collection of funnels, where each one of them represents a single cluster motif. Furthermore, while the number of local minima in the PES scales exponentially with the size N of a cluster, the number of different motifs does not increase so much. The authors also highlight the local search weakness of GO methods, such as `MBH`, based on random perturbations of atom positions. The main idea of [30] is to build an ad-hoc local optimizer being able to locate the minimum of a funnel (that is, the best configuration of the related geometry) at the least possible computational cost; doing so allows an outer GO scheme to focus on the whole picture and coordinate the information provided by the underlying local optimizer.

The authors of [30] build an "expert" local optimizer cascading a first step of classical local optimization, based on the `L-BFGS` method ([64]), and a geometrical one, which works on the atoms lying on the *surface* of a cluster. Indeed, after the numerical optimization phase, the internal part (core) of a cluster is assumed to be well optimized, due to the "pressure" of the atoms in the outer shell: the motif of the molecule[3], thus, is determined by that of its core atoms. Conversely, the atoms belonging to the surface are still not well organized and contribute a high value to the potential energy: the *Cluster Surface Smoothing* optimizer (`CSS`) aims at locating the motif's minimum energy by rearranging the surface atoms. Combining `CSS` with a basic global

[3]Even if not strictly correct, here and in the sequel we will sometimes use the term molecule as a synonymous of atomic cluster.

scheme proved to be able to beat random perturbation strategies, being both more effective (that is, able to successfully locate the minimum of a large funnel/motif) and efficient (employing a tiny fraction of local optimizations), especially at large cluster sizes. We refer to [30] for a deeper look into `CSS`, particularly about how the surface of a cluster is rearranged at each step. Similar approaches can be found in [84], [48] and [79]. In the GO literature the perturbations applied by these approaches go under the name of "direct moves".

One trait of `CSS` which is crucial for our purposes is its *local* nature. Indeed, its authors claim that `CSS` *can fast locate the minimum of the current funnel without attempting to jump to other funnels, even when the energy of current funnel is relatively high*, and highlight that the transition between funnels is left to the GO scheme. This is in contrast with random perturbations strategies, such as `MBH`, where transitions between motifs may occur. In the next section we will show how `CSS` perfectly meets our need of providing the global scheme of `MLSL` with an ad-hoc local optimizer.

4.3.4 Method details

Let us refer to Section 4.3.2. In the scenario of Lennard-Jones or Morse clusters prediction, each point handled by `MLSL` is a 3D molecule: thus, throughout the algorithm, atomic clusters are grouped, selected and locally optimized. If a cluster is composed of N atoms, the feasible space will have $n = 3N$ dimension, as many as all of the atoms' coordinates. With `MLSL`, we aim at focusing the local optimization effort on well defined and independent "areas" of the PES, cutting redundant lines of search. Moreover, our goal is to relate the clustering strategy of `MLSL` to the concept of geometrical class – motif – previously introduced. In other words, we would like `MLSL` to group clusters according to their geometrical properties. Note that this is not guaranteed by the standard setting, that is, if `MLSL`'s grouping strategy is based on atoms coordinates: indeed, as it should be clear from Section 4.1.3, the vector of coordinates of atoms in a cluster is far from being a practically useful descriptor of its geometrical nature, at least from the standpoint of a GO algorithm.

One of our contributions is to make `MLSL`'s grouping strategy suitable for working with geometrical descriptors of clusters, namely their fingerprints. This adjustment is straightforward: at the k-th iteration, Equation (4.18) can be directly used to compute the threshold $\tau(k)$ using the size of the

descriptor d in place of the number of variables $n = 3N$. Moreover, in place of the original variables of the generic samples $\mathbf{x}$ and $\mathbf{y}$ in Equation (4.17), we use the corresponding feature vectors (fingerprints). A slight extra tweak is needed to fit our descriptor in `MLSL`'s formulation. As stated in Section 4.3.2, Equation (4.18) assumes the variables to be bounded in $[0, 1]$: to achieve that, we simply scale the features in the descriptor so that they take values in the required interval. Since this is carried out at each iteration, a cluster's descriptor might slightly change in two subsequent iterations, as more cluster structures (and therefore their descriptors) are discovered. By means of the above adjustment, we are able to compare clusters in a reduced space, induced by our choice of geometrical features, that is, to group them based on their geometrical properties: `MLSL`'s selection step will thus yield, for each observed motif, the best element available in the cluster, namely the *funnel leader*.

We are know ready for including the local `CSS` optimizer into our picture. As stated in Section 4.3.3, `CSS` is able to predict the minimum energy configuration of a given motif at the least cost, without attempting any transition between different funnel/motifs. Employing `CSS` as a local optimizer comes with two main advantages: first, funnel leaders are ideal configurations where to start an in-depth local optimization from. Exploiting the ability of `CSS` of not to jump outside a funnel during the descent, we can safely state that starting a local optimization from a funnel leader will yield the local minimum of that funnel/motif, that is, the best configuration of that geometry. Moreover, `CSS` is a step-by-step process – the surface of a cluster is rearranged iteratively until the minimum of a funnel is reached – which perfectly fits `MLSL`'s scheme: indeed, we can consider the first few steps of CSS (which turn a liquid-like, disordered starting configuration into a slightly optimized one) as the concentration phase of `MLSL`.
Summarising, we use `MLSL` as a global scheme, employing an adapted selection strategy which allows to group cluster configurations based on their geometry. An ad-hoc local optimizer, `CSS`, is used at each iteration both to concentrate the cluster sample, gathering the initially liquid-like structures around their motifs, and to fully optimize the best cluster of each funnel, seeking for the local minimum of the corresponding motif. We called `MLSL-CSS` the resulting scheme.
Algorithm 8 sketches the overall process: in the reported pseudocode, $\mathcal{F} : \mathbb{R}^{3N} \to \mathbb{R}^d$ is a mapping from a generic cluster to a vector of d fingerprints,

while $\mathcal{V}(\cdot)$ is the energy of a given cluster; the threshold $\tau_d(\cdot)$ is defined as in (4.18), but with the dimension n replaced by d. We can easily observe the

Input: $\tau_d(\cdot)$, n_{batch}, n_{steps}, N, K, $\mathcal{F}$
1: Overall sample $O \leftarrow \emptyset$
2: **for** $k = 1, 2, \ldots, K$ **do**
3: $X_{batch} \leftarrow$ (uniformly) generate n_{batch} atomic clusters of size N
4: $X_{stable} \leftarrow$ optimize the starting seeds in X_{batch} by local optimization (`L-BFGS`)
5: $X_{concentrated} \leftarrow$ perform n_{steps} `CSS` steps from each seed in X_{stable}
6: $O \leftarrow O \bigcup X_{concentrated}$
7: $X_{selection} \leftarrow \emptyset$
8: **for** $X \in O$ **do**
9: **if** $\forall Y : \mathcal{V}(Y) \leq \mathcal{V}(X) \Rightarrow \|(\mathcal{F}(X) - \mathcal{F}(Y)\| \geq \tau_d(k)$ **then**
10: $X_{selection} \leftarrow X_{selection} \cup X$
11: $X_{fully\_optimized} \leftarrow$ perform a full `CSS` optimization from each seed in $X_{selection}$
12: $O \leftarrow O \bigcup X_{fully\_optimized}$
Output: best solution in O

Algorithm 8: `MLSL-CSS`

similarity of this scheme to the plain `MLSL` one.
Let us give some implementation details. The selection phase at lines (8)-(10) is carried out by first sorting the clusters in the overall sample O based on their energy (from the best down to the worst); the current point X is then added to $X_{selection}$ if none of the clusters previously seen (that is, those with an energy lower than the current) is distant from X less than the current threshold. Furthermore, a selected cluster in $X_{selection}$ is fully optimized only if this has not been done yet.

4.3.5 Numerical experiments

In our first batch of experiments, we have the double goal of calibrating the parameters of `MLSL-CSS` and interpreting them. Furthermore, we aim at showing how our method is able to early group atomic configurations, and exploit this capability by pruning redundant lines of search while exploring promising and *new* ones.

Let us consider Lennard Jones clusters of size N equal to 100 up to 200 atoms. To build our testing ground we first ran, for each instance, 1000 `CSS` trials, that is, we performed a full `CSS` optimization from 1000 different starting configurations. We gathered, for each trial, the optimum configuration obtained by the `CSS` algorithm and the corresponding number

of L-BFGS calls. With the aim of comparing our global scheme with the multi-start one (which from now on we will refer to as multi-CSS), we used, as initial configurations for our MLSL-CSS runs, the same seeds used as starting points for CSS. Due to the excellent quality of the CSS optimizer, multi-CSS is almost always able to provide, within the 1000 trials, the best optimum known in the literature, (the *putative* global optimum); the only instance where multi-CSS fails to detect the putative optimum within 1000 trials is $N = 176$. What we are comparing our method with is then a simple but extremely effective scheme, which succeeds in predicting the putative global optimum of most medium scale instances. We did not chose smaller clusters in this phase, as they are quite trivially solved by multi-CSS, even in the case of *magic numbers*, like $N = 75$, 78 and 98, which were once considered as extremely challenging for an unbiased GO method.

In the current experiments we tried to find a suitable calibration of the parameters of our algorithm in order to achieve two goals:

1. preserving the excellent exploration capability of MS, namely being able to return the best optimum found by multi-CSS;
2. achieve this with significantly lower computational effort.

In other words, we aimed at showing that the proposed method is capable of delivering the same quality of multi-CSS, but at a fraction of the computational cost.

Similarly to what we did dealing with the CINAC problem in Section 4.2.2, we define the *gain* G as

$$G := 1 - \frac{L_{\texttt{MLSL-CSS}}}{L_{\texttt{multi-CSS}}}, \tag{4.19}$$

where $L_{\texttt{MLSL-CSS}}$ and $L_{\texttt{multi-CSS}}$ refer, respectively, to the number of times MLSL-CSS and multi-CSS invoked the L-BFGS optimizer. Metrics that involve the number of calls to a local optimizer are widely used in the literature, and provide a convenient way of evaluating the efficiency of a GO algorithm, being independent from the testing hardware. Moreover, the execution time of L-BFGS (which is employed both by MLSL-CSS and multi-CSS as underlying numerical optimizer) is certainly a bottleneck in the computational effort of the overall process.

We define as a *hit* a MLSL-CSS execution returning the best optimum found by multi-CSS; similarly, we say that our method hits multi-CSS when

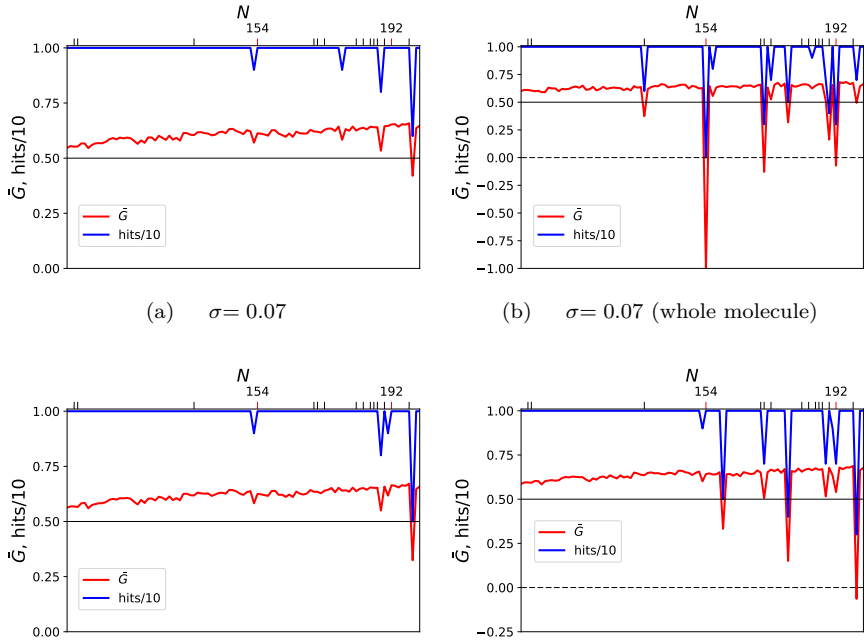
the line of search that would have reached the best configuration found by the multi-start scheme is not wrongly pruned by our clustering approach. On the 100 out of 101 instances where CSS is able to spot the best optimum from the literature, hitting multi-CSS is therefore equivalent to hit the putative global optimum.

In order to compare our method with multi-CSS, having already run 1000 independent CSS local searches, we just checked from which of the elements in the $X_{concentrated}$ set our algorithm decided to perform a full CSS optimization. Similarly to the "experienced" scheme presented in Section 4.2, our method is non deterministic, since its evolution depends on the order in which batches are processed. To take this into account, we ran MLSL-CSS on ten independent seed permutations, that is, we randomly permuted the 1000 CSS seeds before starting MLSL-CSS.

A positive outcome of this experiment should therefore show a high hit rate (ideally 10/10 hits, meaning that our method has been able to detect the best optimum found by multi-CSS regardless of the order in which the 1000 CSS trials were processed) and a large saving in terms of L-BFGS calls. On the other hand, it is important to notice that achieving less than 10/10 hits does not always imply that MLSL-CSS is beaten by MS: indeed, a positive saving can be obtained even with less than 10/10 hits. Let us denote by $x^*_{\texttt{multi-CSS}}$ the best optimum found by multi-CSS and assume that MLSL-CSS have hit multi-CSS, say, nine out of ten times, with a gain in local searches equals to $G = 0.6$; this means that, on average, a single MLSL-CSS run requires 40% of the number of local optimization calls required by multi-CSS. To hit $x^*_{\texttt{multi-CSS}}$, on average, we need $\frac{10}{9}$ MLSL-CSS trials, each requiring 40% of the L-BFGS calls needed by multi-CSS: thus, we are still achieving an *actual* gain of $\bar{G} := 1 - (\frac{10}{9} * 0.4) = 0.56$, meaning that we are saving more than the half of L-BFGS calls. We define the *expected gain* $\bar{G}$ as

$$\bar{G} = 1 - \frac{L_{\texttt{MLSL-CSS}}}{L_{\texttt{multi-CSS}}} \frac{H_{\texttt{multi-CSS}}}{H_{\texttt{MLSL-CSS}}}, \tag{4.20}$$

where $H_{\texttt{multi-CSS}}$ and $H_{\texttt{MLSL-CSS}}$ denote the number of hits achieved respectively by multi-CSS and MLSL-CSS. Note that $\bar{G} = G$ when the hit rate is the same for both algorithms. A negative expected gain indicates that our method has been indeed beaten by the multi-start approach, and quantifies the advantage of using multi-CSS over MLSL; in particular, $\bar{G}$ is set to -1.0 when our method fails to hit $x^*_{\texttt{multi-CSS}}$ on all of the 10 executions.



(a) $\sigma = 0.07$ (b) $\sigma = 0.07$ (whole molecule)

(c) $\sigma = 0.08$ (d) $\sigma = 0.1$

Figure 4.5: Average gain $\bar{G}$ (red) and number of hits (blue) on ten different permutations of the `multi-CSS` trials, on Lennard-Jones cluster instances from $N = 100$ up to 200 atoms. On the x-axis, ticks are reported only for "difficult" instances, namely, where `multi-CSS` finds the putative global optimum less than the 0.5% of the time (that is, less than 5 out of 1000 times); instances where `multi-CSS` is able to spot the putative optimum only once ($N = 154$ and $N = 192$) are also marked in red. A gain value above the $\bar{G} = 0.50$ horizontal line indicates that `MLSL-CSS` allows to save more than the 50% of the `L-BFGS` calls needed by `MS`, while negative gains in (b) and (d), that is, where `MLSL-CSS` is beaten by `multi-CSS`, fall below the additional dashed line. In all experiments a single `CSS` step has been performed as `MLSL` concentration step; the `MLSL`'s threshold σ is risen from 0.07 to 0.1 in charts (a), (c) and (d). In chart (b), the geometrical descriptor is computed, after the concentration step, from the whole molecule, without extracting the core.

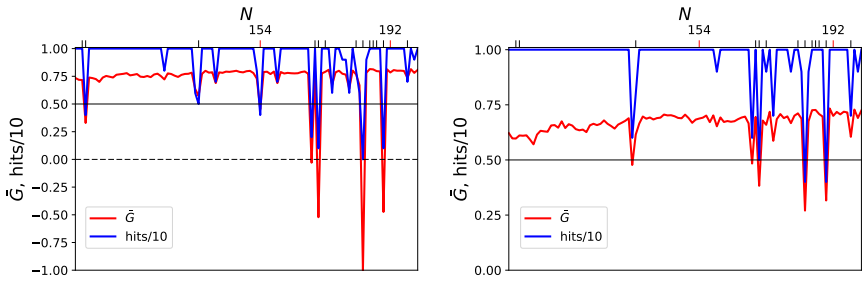
Figure 4.5 reports the average gain $\bar{G}$ (red) and number of hits (blue) on ten different permutations of the `multi-CSS` trials, on cluster instances from $N = 100$ up to 200. Ticks are reported on the x-axis only for "difficult" in-

stances, namely, where `multi-CSS` finds the putative global optimum in less than 5 out of 1000 runs. Those instances where `multi-CSS` is able to obtain the putative optimum only once ($N = 154$ and $N = 192$) are additionally marked in red. For convenience, let us call these two subsets of instances respectively I and J. When the observed gain falls above the $\bar{G} = 0.50$ horizontal line, `MLSL-CSS` allows to save more than one half of the `L-BFGS` calls needed by `multi-CSS`. In Figure 4.5 (a), one `CSS` step is carried out as `MLSL-CSS`'s concentration step ($n_{steps}= 1$) and the threshold parameter σ is set to 0.07. In this setting, `MLSL-CSS` is able to achieve, on average over all the instances in I, a gain of $\bar{G} = 0.62$, reaching 0.63 on J; on all of the 101 instances, the average gain is 0.61. This results clearly show how our scheme is able to successfully predict the putative global optimum while saving more than 60% of the computational effort. This happens also for extremely hard instances, where the putative optimum is seen only once: `MLSL-CSS`'s geometrical clustering is thus able to identify promising motifs, even if the corresponding funnel is very narrow, getting rid of a large amount of redundant configurations.

Let us keep the setting of Figure 4.5 (a) as a benchmark and rise the threshold parameter σ, in charts (c) and (d), respectively to $\sigma = 0.08$ and 0.1. With reference to Equation (4.18), rising σ yields a higher threshold τ, which in turns makes a configuration in the overall sample less likely to be selected and fully optimized. In Figure 4.5 (c), rising the threshold to $\sigma = 0.08$ yields an average gain higher than the benchmark in (a), namely $\bar{G} = 0.64$ on both the I and J subsets, while the overall average gain reaches 0.62. It is worth to notice that, while a hit is lost on both $N = 198$ and 191, rising σ have "recovered" $N = 178$, yielding a full hit rate for that instance. This outcome shows how increasing the threshold τ in Equation (4.18) does not always imply a (possible) reduction in the hit rate: making the selection step more demanding, indeed, can also help getting rid of "noisy" configurations, rendering the method sharper.
Switching to Figure 4.5 (d), we can see how further increasing σ yields a failure on $N = 198$, where the gain is slightly negative ($\bar{G} = -0.06$). Also, other hits are lost for some other cluster sizes, including the red instance $N = 192$, though a positive gain is still obtained. In the setting of Figure 4.5 (d), the average gain for the instances in I remains 0.64, while decreasing on J to 0.59; the average gain on all of the cluster sizes is still 0.62.

Let us now switch to Figure 4.5 (b): in this chart, the parameter setting

is the same as the benchmark in (a), but the fingerprint is computed on the whole molecules, instead of extracting the geometrical descriptor only from the atoms belonging to the clusters' core. It is easy to notice how including the surface atoms in the procedure makes the scheme less accurate, yielding a considerable loss of hits. A negative gain is obtained for instances in J: in particular, `MLSL-CSS` is never able to hit the putative optimum on $N = 192$. This analysis should confirm how extracting a reliable geometrical descriptor is crucial for the selection step of `MLSL-CSS`.



(a) $\sigma = 0.07$ $n_{steps} = 0$ (b) $\sigma = 0.03$ $n_{steps} = 0$

Figure 4.6: Gain profiles for the selected testing ground. In (a), the concentration phase is limited to a pure numerical optimization, without using `CSS` to work on clusters' surface. In (b), the same experiment is carried out, with a lower threshold parameter σ.

In the experiment depicted in Figure 4.6, the starting seeds of `MLSL-CSS` are concentrated only by a `L-BFGS` optimization: therefore, the concentration step does not employ `CSS`'s surface smoothing technique to group the liquid-like configurations around their basins of attraction, and it boils down to a pure numerical optimization. As expected – see Figure 4.6 (a) – many hits are lost due to a shallower concentration, even if the average gain on all of the cluster sizes is the highest seen so far, namely $\bar{G} = 0.71$: in particular, a negative gain is obtained on $N = 169$, 171, 184 and 190, with `MLSL-CSS` achieving 0 out of 10 hits on instance 184.
Let us now try to "recover" these failures, while keeping the gain high by concentrating the starting seeds still only through a numerical optimization, that is, skipping `CSS`'s surface smoothing. To this aim, in Figure 4.6 (b) we

decrease the threshold parameter σ to 0.03. By doing so, a positive gain is obtained on all of the instances, with an average saving of 66% of `L-BFGS` calls. On the instances in J, a maximum hit rate is achieved, with a gain of 0.69; finally, $\bar{G} = 0.61$ on I.

With the above experiments, we have showed how `MLSL-CSS` is able to preserve the quality of an excellent multi-start scheme, while saving a large percentage of local searches. Consider now the Morse potential defined in Section 4.1.2, with $\rho = 14$: the latter parameter controls the width of the unique global optimum, and with our choice we aim at minimizing the potential energy in the hardest scenario. We have carried out experiments on Morse cluster instances from $N = 40$ up to 130 atoms, running 50 "pure" trials of `MLSL-CSS` for each configuration. By pure we mean that we are not simulating our scheme based on existent `CSS` runs (as we did in the last batch of experiments), and that each trial is totally independent from the others. In particular, a trial is stopped whenever the putative global optimum is found. By doing this, we aim at assessing the effectiveness of our method on a set of instances which are among the hardest to be solved; in other words, while the previous experiments have proved the efficiency of `MLSL-CSS` with respect to `multi-CSS`, our goal is now to evaluate the *recall* of our scheme, that is, the percentage of trials, among those carried out, in which our scheme is able to locate the putative optimum. In this experiment, our parameter choice is that of Figure 4.5 (a). Figure 4.7 reports the

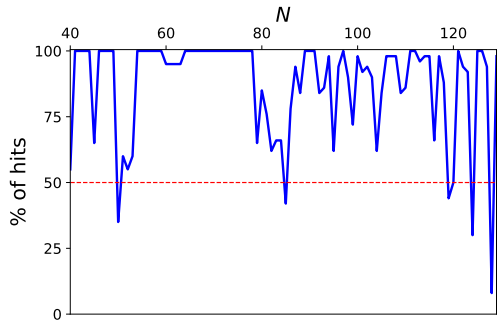
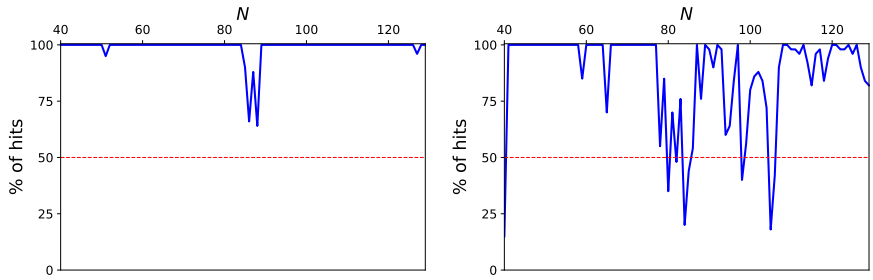


Figure 4.7: Percentage of hits achieved by `MLSL-CSS` on Morse cluster instances ($\rho = 14$) from $N = 40$ to 130 atoms.

percentage of hits on 50 trials of `MLSL-CSS`: on only 4 out of 90 instances the

hit rate is lower than 50%, meaning that our method was able to locate the putative global optimum on less than the half of the trials. In particular, the worst result is $N = 128$, where the putative optimum is hit 4 out of 50 times. However, what is worth to notice is the ability of our scheme to always hit the global optimum at least once on a selection of problems which are known to be very hard: that confirms the effectiveness of `MLSL-CSS`. For the sake



(a) $\rho = 6$ (b) $\rho = 10$

Figure 4.8: Percentage of hits achieved by `MLSL-CSS` on Morse cluster instances from $N = 40$ to 130 atoms with $\rho = 6$ (a) and $\rho = 10$ (b).

of completeness, Figure 4.8 also reports the hit rate in the $\rho = 6$ and $\rho = 10$ scenarios.

4.4 Conclusion

A run of a GO algorithm rarely exploits information from the past, as methods often yield their prediction of the global optimum as only output. Moreover, when dealing with global problems with a clear geometrical component, all those solutions that are obtained from a given one by trivial space transformations, even if different from GO's standpoint, actually refer to the same spatial configuration. In Section 4.2 we have extended the well-known scheme of `MBH` by including a compact description of known solutions as a method input. Such descriptor is extracted by configurations of the CINAC packing problem, and used as a mean to represent the geometry of a solution. The resulting enhanced scheme is thus able to stop the execution of a run when bumping into a solution whose description has already been encountered.

The extension of MBH has been able to save a large amount of computational time, with respect to the plain method.
The above approach, however, has two downsides. First, methods like MBH are based on random perturbations, and this way of generating new solutions makes hard to predict the outcome of the current search. Therefore, a scheme based on the early detection of known structures can fail, especially with large scale problems where the total number of possible structures is much higher. Another weakness of the proposed scheme is the way it checks if a newly reached configuration has already been encountered in the past. By applying a strict equivalence check, we are employing our compact representation only to early detect trivial geometrical transformations of a known solution.

Cluster methods, which aim at focusing local optimization effort on precise areas of the feasible space, have been abandoned by the recent GO literature. This because to group solutions based solely on their variables' value can be a very hard task even at medium scale. In Section 4.3, we used the concept of fingerprinting in a broader way, and adapt the MLSL clustering scheme to work in the compact space induced by our choice of geometrical features. In this setting, large scale molecular structure configurations are grouped based on their geometrical properties, instead that looking at their atoms' position. Numerical experiments have been carried out, showing how the quality of a very efficient multi-start scheme is preserved, while to number of local searches needed to locate the putative global optimum is dramatically decreased.

Chapter 5

Conclusion

Machine learning and global optimization can interact in many ways and help each other. In Chapters 2 and 3 we showed how global techniques can be used to solve optimization problems which arise from machine learning scenarios.
In many classification tasks, manually labeling training instances is a costly process, and supervised methods are likely not to deliver reliable models when labeled data are scarce. Semi-supervised learning, which aims at involving unsupervised patterns in the training process along with the labeled dataset, is often able to improve on the accuracy achieved by the supervised approach. Global optimization can be used to optimize the non convex objective function which arises from semi-supervised classification. While the continuous approach expresses the unknown labels in terms of their predictions, combinatorial methods treat them as optimization variables. These methods usually scale badly with the size of the unsupervised dataset, and the combinatorial approach is known to be intractable even in medium scale applications. In Chapter 2 we presented a combinatorial exact method for $\texttt{S}^3\texttt{VM}$s. Thanks to the use of a Lagrangean-based global optimization scheme, we were able to get rid of the dependency on the number of unsupervised patterns. Moreover, directly taking into account the balance between positive and negative samples turned out to be a successful design choice. Indeed, the resulting method proved to be superior to state-of-the-art $\texttt{S}^3\texttt{VM}$ implementations in terms of both model accuracy and optimization efficiency, as it is clear from small to very large scale experiments. The presented method also comes with a very lightweight parameter selection phase, in contrast

with recent implementations from the literature. Our approach presupposes a fair estimation of the ratio of unlabeled samples to be classified as positive, and should be used when there is enough confidence on the value of this parameter.

Variable selection and ranking are important steps of a machine learning workflow. They also provide valuable information that can be straight used, without necessarily be input to a machine learning model. Feature relevance is not defined accurately in the literature, and a weak point of many variable ranking methods is that of being univariate, that is, taking into account one feature at a time when building the overall ranking. This choice may produce bad quality results as a feature which is useless when taken into account individually can be relevant when employed along with one or more other variables. Global optimization has been used in Chapter 3 to define a concept of feature relevance, and a global problem has been solved to produce a ranking for the variables of a nonlinear regression problem. In addition, such a method is multivariate as all of the variables are considered simultaneously in the ranking process. The resulting ranking strategy turned out to be either competitive or more accurate than state-of-the-art methods. Even if highly competitive in terms of the accuracy achieved by the regression model trained on the variables which are relevant according to the ranking procedure, the proposed technique is clearly less efficient than filter methods. However, its structure is straightforward and can easily fit a parallel workflow. Chapter 3 presents a distributed implementation, which uses the state-of-the-art programming paradigm and platform provided by `Apache Spark`. This design choice alleviates the computational effort, and renders the proposed technique scalable and applicable to problems of a larger scale.

From the perspective of Chapter 4, machine learning can help global optimization to learn from and make a smart use of information from past executions, and machine learning techniques such as clustering and feature engineering can be fruitfully employed to enhance global schemes. Clustering methods provide a way to organize local information and focus the optimization effort on promising areas of the feasible space. Though they were largely used in the past, they proved to be weak when the scale of the problem to be solved is large, as they compare solutions based on the value of their variables. In Chapter 4 we have developed structural descriptors for representing the solutions of two well-known geometrical problems in a compact way, and adapted a clustering method to work in the restricted

space induced by our choice of geometrical features. Large scale solutions are therefore compared and grouped based on their overall characteristics rather than the value of their variables. Thanks to our strategy we were able to obtain the same quality of a very effective multi-start approach, while employing a tiny percentage of local searches to converge to the putative global optimum.

Bibliography

[1] R. Abdel-Aal, "Gmdh-based feature ranking and selection for improved classification of medical data," *Journal of Biomedical Informatics*, vol. 38, no. 6, pp. 456–468, 2005.

[2] T. Abeel, T. Helleputte, Y. Van de Peer, P. Dupont, and Y. Saeys, "Robust biomarker identification for cancer diagnosis with ensemble feature selection methods," *Bioinformatics*, vol. 26, no. 3, pp. 392–398, 2009.

[3] B. Addis, M. Locatelli, and F. Schoen, "Disk packing in a square: A new global optimization approach," *INFORMS Journal on Computing*, vol. 20, no. 4, pp. 516–524, Dec. 2008.

[4] ——, "Efficiently packing unequal disks in a circle," *Operations Research Letters*, vol. 36, no. 1, pp. 37–42, 2008.

[5] M. R. Amini, N. Usunier, and C. Goutte, "Learning from multiple partially observed views - an application to multilingual text categorization," vol. 1, pp. 28–36, 2010.

[6] F. R. Bach, G. R. Lanckriet, and M. I. Jordan, "Multiple kernel learning, conic duality, and the smo algorithm," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, pp. 6–14.

[7] F. Bagattini, P. Cappanera, and F. Schoen, "Lagrangean-based combinatorial optimization for large-scale S3VMs," *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[8] ——, "A simple and effective lagrangian-based combinatorial algorithm for S3VMs," in *International Workshop on Machine Learning, Optimization, and Big Data*. Springer, 2017, pp. 244–254.

[9] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic programming: an introduction*. Morgan Kaufmann San Francisco, 1998, vol. 1.

[10] A. C. Barros and G. D. Cavalcanti, "Combining global optimization algorithms with a simple adaptive distance for feature selection and weighting,"

in *2008 International Joint Conference on Neural Networks (IJCNN 2008)*. IEEE, 2008, pp. 3518–3523.

[11] D. Basak, S. Pal, and D. C. Patranabis, "Support vector regression," *Neural Information Processing-Letters and Reviews*, vol. 11, no. 10, pp. 203–224, 2007.

[12] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, Dec. 2006.

[13] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*. Springer, 2009, pp. 1–4.

[14] F. BenoíT, M. Van Heeswijk, Y. Miche, M. Verleysen, and A. Lendasse, "Feature selection for nonlinear models with extreme learning machines," *Neurocomputing*, vol. 102, pp. 111–124, 2013.

[15] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, 1999.

[16] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1, pp. 245–271, 1997.

[17] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," *Knowledge and Information Systems*, vol. 34, no. 3, pp. 483–519, 2013.

[18] A. Boubezoul and S. Paris, "Application of global optimization methods to model and feature selection," *Pattern Recognition*, vol. 45, no. 10, pp. 3676–3686, 2012.

[19] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[20] P. S. Bradley, O. L. Mangasarian, and W. N. Street, "Feature selection via mathematical programming," *INFORMS Journal on Computing*, vol. 10, no. 2, pp. 209–217, 1998.

[21] L. Bravi, V. Piccialli, and M. Sciandrone, "An optimization-based method for feature ranking in nonlinear regression problems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 4, pp. 1005–1010, April 2017.

[22] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone, "Machine learning for global optimization," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 279–303, Aug. 2012.

[23] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[24] O. Chapelle, M. Chi, and A. Zien, "A continuation method for semi-supervised svms," Max-Planck-Gesellschaft. New York, NY, USA: ACM Press, Jun. 2006, pp. 185–192.

[25] O. Chapelle, V. Sindhwani, and S. S. Keerthi, "Branch and bound for semi-supervised support vector machines," Max Plank Institute, Tech. Rep., 2006.

[26] ——, "Optimization techniques for semi-supervised support vector machines," *Journal of Machine Learning Research*, vol. 9, pp. 203–233, 2008.

[27] O. Chapelle and A. Zien, "Semi-supervised classification by low density separation." in *AISTATS*, 2005, pp. 57–64.

[28] F.-L. Chen and F.-C. Li, "Combination of feature selection approaches with svm in credit scoring," *Expert Systems with Applications*, vol. 37, no. 7, pp. 4902–4909, 2010.

[29] J. Cheng and R. Fournier, "Structural optimization of atomic clusters by tabu search in descriptor space," *Theoretical Chemistry Accounts*, vol. 112, no. 1, pp. 7–15, 2004.

[30] L. Cheng, Y. Feng, J. Yang, and J. Yang, "Funnel hopping: Searching the cluster potential energy surface over the funnels," *The Journal of Chemical Physics*, vol. 130, no. 21, p. 214112, 2009.

[31] R. Cole, Y. Muthusamy, and M. Fanty, "The ISOLET Spoken Letter Database," Oregon Graduate Institute, Tech. Rep., 1990.

[32] R. Collobert, F. Sinz, J. Weston, and L. Bottou, "Large scale transductive svms," *Journal of Machine Learning Research*, vol. 7, pp. 1687–1712, September 2006.

[33] A. Costa, P. Hansen, and L. Liberti, "On the impact of symmetry-breaking constraints on spatial branch-and-bound for circle packing in a square," *Discrete Applied Mathematics*, vol. 161, no. 1â2, pp. 96 – 106, 2013.

[34] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[35] I. Csisz, G. Tusnády *et al.*, "Information geometry and alternating minimization procedures," *Statistics and decisions*, 1984.

[36] T. De Bie and N. Cristianini, *Semi-supervised learning using semi-definite programming*. MIT Press, 2006.

[37] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the Association for Computing Machinery*, vol. 51, no. 1, pp. 107–113, 2008.

[38] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[39] J. P. Doye, R. H. Leary, M. Locatelli, and F. Schoen, "Global optimization of morse clusters by potential energy transformations," *INFORMS Journal on Computing*, vol. 16, no. 4, pp. 371–379, 2004.

[40] R. Fournier, "Theoretical study of the structure of silver clusters," *The Journal of Chemical Physics*, vol. 115, no. 5, pp. 2165–2177, 2001.

[41] F. Gieseke, A. Airola, T. Pahikkala, and O. Kramer, "Fast and simple gradient-based optimization for semi-supervised support vector machines," *Neurocomputing*, vol. 123, pp. 23 – 32, 2014.

[42] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, vol. 12, no. 4, pp. 979–1006, Apr. 2002. [Online]. Available: http://dx.doi.org/10.1137/S1052623499350013

[43] F. Glover, "Tabu search, part i," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[44] A. Grosso, A. Jamali, M. Locatelli, and F. Schoen, "Solving the problem of packing equal and unequal circles in a circular container," *Journal of Global Optimization*, vol. 47, no. 1, pp. 63–81, May 2010.

[45] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[46] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, no. 1, pp. 389–422, 2002.

[47] W. E. Hart, C. Laird, J.-P. Watson, and D. L. Woodruff, *Pyomo - Optimization Modeling in Python*. Springer, 2012, vol. 67.

[48] B. Hartke, "Global cluster geometry optimization by a phenotype algorithm with niches: Location of elusive minima, and low-order scaling with cluster size," *Journal of computational chemistry*, vol. 20, no. 16, pp. 1752–1759, 1999.

[49] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[50] W. Huang and T. Ye, "Global optimization method for finding dense packings of equal circles in a circle," *European Journal of Operational Research*, vol. 210, no. 3, pp. 474 – 481, 2011.

[51] U. Huebner, N. Abraham, and C. Weiss, "Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared nh 3 laser," *Physical Review Letters*, vol. 40, no. 11, p. 6354, 1989.

[52] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 200–209.

[53] J. E. Kelley, Jr, "The cutting-plane method for solving convex programs," *Journal of the society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[54] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*. Springer, 2011, pp. 760–766.

[55] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of the Ninth International Workshop on Machine Learning*, 1992, pp. 249–256.

[56] A. Lendasse, J. Lee, V. Wertz, and M. Verleysen, "Forecasting electricity consumption using nonlinear projection and self-organizing maps," *Neurocomputing*, vol. 48, no. 1, pp. 299–311, 2002.

[57] A. Lendasse, J. A. Lee, V. Wertz, and M. Verleysen, "Time series forecasting using cca and kohonen maps-application to electricity consumption." in *Eighth European Symposium on Artificial Neural Networks (ESANN 00)*, 2000, pp. 329–334.

[58] B. Li, Z. Yang, Y. Zhu, H. Meng, G. Levow, and I. King, "Predicting user evaluations of spoken dialog systems using semi-supervised learning," in *Spoken Language Technology Workshop (SLT), 2010 IEEE*, Dec 2010, pp. 283–288.

[59] Y. F. Li and Z. H. Zhou, "Towards making unlabeled data never hurt," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 1, pp. 175–188, Jan 2015.

[60] Y.-F. Li, J. T. Kwok, and Z.-H. Zhou, "Semi-supervised learning using label mean," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 633–640.

[61] Y.-F. Li, I. W. Tsang, J. T. Kwok, and Z.-H. Zhou, "Convex and scalable weakly labeled svms," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 2151–2188, 2013.

[62] S.-W. Lin, Z.-J. Lee, S.-C. Chen, and T.-Y. Tseng, "Parameter determination of support vector machine and feature selection using simulated annealing approach," *Applied Soft Computing*, vol. 8, no. 4, pp. 1505–1512, 2008.

[63] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines," *Expert Systems with Applications*, vol. 35, no. 4, pp. 1817–1824, 2008.

[64] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[65] M. Locatelli and F. Schoen, "Local search based heuristics for global optimization: atomic clusters and beyond," *European Journal of Operational Research*, vol. 222, no. 1, pp. 1–9, 2012.

[66] ——, *Global Optimization: theory, algorithms, and applications*, ser. MOS-SIAM Series on Optimization. Philadelphia, USA: Society for Industrial and Applied Mathematics and the Mathematical Optimization Society, 2013, vol. MO15.

[67] B. D. Lubachevsky and F. H. Stillinger, "Geometric properties of random disk packings," *Journal of Statistical Physics*, vol. 60, no. 5-6, pp. 561–583, 1990.

[68] O. Mangasarian, "Machine learning via polyhedral concave minimization," in *Applied Mathematics and Parallel Computing*. Springer, 1996, pp. 175–188.

[69] B. A. Murtagh and M. A. Saunders, "Minos 5.0 user's guide." STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB, Tech. Rep., 1983.

[70] S. A. Nene, S. K. Nayar, and H. Murase, "Columbia Object Image Library (COIL-100)," Tech. Rep., Feb 1996.

[71] J. P. K. Doye and D. J. Wales, "Structural consequences of the range of the interatomic potential a menagerie of clusters," *Journal of the Chemical Society Faraday Transactions*, vol. 93, pp. 4233–4243, 1997.

[72] Z. Pawlak, "Rough sets," *International Journal of Parallel Programming*, vol. 11, no. 5, pp. 341–356, 1982.

[73] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[74] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta Numerica*, vol. 8, pp. 143–195, 1999.

[75] R. M. Rifkin, "Everything old is new again: A fresh look at historical approaches in machine learning," Ph.D. dissertation, Cambridge, MA, USA, 2002.

[76] A. H. G. Rinnooy Kan and G. T. Timmer, "Stochastic global optimization methods. Part II: Multi level methods," *Mathematical Programming*, vol. 39, pp. 57–78, 1987.

[77] L. Rosasco, M. Santoro, S. Mosci, A. Verri, and S. Villa, "A regularization approach to nonlinear variable selection," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 653–660.

[78] B. Schölkopf, R. Herbrich, and A. Smola, "A generalized representer theorem," in *Computational learning theory*. Springer, 2001, pp. 416–426.

[79] X. Shao, L. Cheng, and W. Cai, "A dynamic lattice searching method for fast optimization of lennard-jones clusters," *Journal of Computational Chemistry*, vol. 25, no. 14, pp. 1693–1698, 2004.

[80] V. Sindhwani, S. S. Keerthi, and O. Chapelle, "Deterministic annealing for semi-supervised kernel machines," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 841–848.

[81] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[82] P. J. Steinhardt, D. R. Nelson, and M. Ronchetti, "Bond-orientational order in liquids and glasses," *Physical Review B*, vol. 28, no. 2, p. 784, 1983.

[83] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[84] H. Takeuchi, "Clever and efficient method for searching optimal geometries of lennard-jones clusters," *Journal of chemical information and modeling*, vol. 46, no. 5, pp. 2066–2070, 2006.

[85] P. Timsina, J. Liu, O. El-Gayar, and Y. Shang, "Using semi-supervised learning for the creation of medical systematic review: An exploratory analysis," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pp. 1195–1203.

[86] S. Tiwari, B. Singh, and M. Kaur, "An approach for feature selection using local searching and global optimization techniques," *Neural Computing and Applications*, pp. 1–16, 2017.

[87] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[88] V. N. Vapnik and A. Sterin, "On structural risk minimization or overall risk in a problem of pattern recognition," *Automation and Remote Control*, vol. 10, no. 3, pp. 1495–1503, 1977.

[89] V. N. Vapnik and V. Vapnik, *Statistical learning theory*. Wiley New York, 1998, vol. 1.

[90] D. J. Wales and J. P. K. Doye, "Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms," *Journal of Physical Chemistry*, vol. 101, no. 28, pp. 5111–5116, 1997.

[91] X. Wang, J. Yang, X. Teng, W. Xia, and R. Jensen, "Feature selection based on rough sets and particle swarm optimization," *Pattern Recognition Letters*, vol. 28, no. 4, pp. 459–471, 2007.

[92] Y. Wang, S. Teitel, and C. Dellago, "Melting of icosahedral gold nanoclusters from molecular dynamics simulations," *The Journal of Chemical Physics*, vol. 122, no. 21, p. 214722, 2005.

[93] A. S. Weigend and N. A. Gershenfeld, "Results of the time series prediction competition at the santa fe institute," in *IEEE International Conference on Neural Networks*. IEEE, 1993, pp. 1786–1793.

[94] J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping, "Use of the zero-norm with linear models and kernel methods," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1439–1461, 2003.

[95] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of Machine Learning Research*, vol. 5, no. Oct, pp. 1205–1224, 2004.

[96] A. L. Yuille and A. Rangarajan, "The concave-convex procedure," *Neural computation*, vol. 15, no. 4, pp. 915–936, 2003.

[97] M. Zaffalon and M. Hutter, "Robust feature selection by mutual information distributions," in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 577–584.

[98] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–16.

[99] T. Zhang and F. J. Oles, "Text categorization based on regularized linear classification methods," *Information retrieval*, vol. 4, no. 1, pp. 5–31, 2001.