ALMA MATER STUDIORUM

UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE

AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Artificial Intelligence in Industry

EXTENDING THE MOVING TARGETS METHOD FOR INJECTING CONSTRAINTS IN MACHINE LEARNING

CANDIDATE

Luca Giuliani

SUPERVISOR

Prof. Michele Lombdardi

CO-SUPERVISOR

Eng. Fabrizio Detassis

Academic year 2020/21

Session 1st

Abstract

Informed Machine Learning is an umbrella term that comprises a set of methodologies in which domain knowledge is injected into a data-driven system in order to improve its level of accuracy, satisfy some external constraint, and in general serve the purposes of explainability and reliability. The said topid has been widely explored in the literature by means of many different techniques. Moving Targets is one such a technique particularly focused on constraint satisfaction: it is based on decomposition and bi-level optimization and proceeds by iteratively refining the target labels through a master step which is in charge of enforcing the constraints, while the training phase is delegated to a learner. In this work, we extend the algorithm in order to deal with semi-supervised learning and soft constraints. In particular, we focus our empirical evaluation on both regression and classification tasks involving monotonicity shape constraints. We demonstrate that our method is robust with respect to its hyperparameters, as well as being able to generalize very well while reducing the number of violations on the enforced constraints. Additionally, the method can even outperform, both in terms of accuracy and constraint satisfaction, other state-of-the-art techniques such as Lattice Models and Semantic-based Regularization with a Lagrangian Dual approach for automatic hyperparameter tuning.

Contents

Introduction

Progresses made over the last few years in the field of Machine Learning made it possible to expand the application of such a techniques to a wider set of both industrial and research areas. Notably, a renovated interest in the development of hybrid symbolic/subsymbolic systems is evident from the plethora of scientific publications on the theme, which constantly show how the combination of these two methods can undoubtedly help to overcome the limits of each of them, singularly. In fact, while symbolic models have been traditionally designed by domain experts and mainly employed for constraint reasoning, with evident drawbacks when it comes to scalability and robustness, subsymbolic ones are able to provide resilient outcomes even when fed with noisy data from stochastic processes, but they are not conceived to satisfy explicit rules, nor to be interpretable. The main idea, therefore, is to let the two modules interact during either the learning or the optimization phase, depending on the purposes and the methodologies adopted in the application.

The benefits are several: from a pure improvement of the effectiveness and efficiency of the system, to the satisfaction of external requirements regarding reliability, explainability, robustness when dealing with uncertainty and noise, and trustworthiness in general. Among all, this second aspect is gradually obtaining a larger share of the debate on technological evolution, given the expected impact that Artificial Intelligence systems will bring to all the aspects of our society in a short time frame [1].

The work exposed in this dissertation is located within the research area of Informed Machine Learning, and consists in the extension of Moving Targets, a recently proposed state-of-the-art method for constraint injection in data-driven applications [2]. After some preliminary investigations about specific properties of the algorithm that were left unexplored, we focused our efforts on the development of ad-hoc techniques in order to deal with semi-supervised learning scenarios rather then supervised ones only. More specifically, we examined the effect of both existing and newly-introduced hyperparameters on the handling of monotonicity shape constraints both in regression and classification tasks.

Soon after an introductory chapter consisting in an overview of the research area and the current state of the art, a comprehensive summary of our contribution can be found in chapter 2. Afterwards, chapter 3 will provide an outline of the empirical results obtained both from the analysis of the hyperparameters, measured by means of their effect on the model accuracy and the constraint satisfaction, and the benchmarking against other traditional methods from the literature. Eventually, our final conclusions about the work done and the future directions to follow in order to improve the method will be presented in chapter 4.

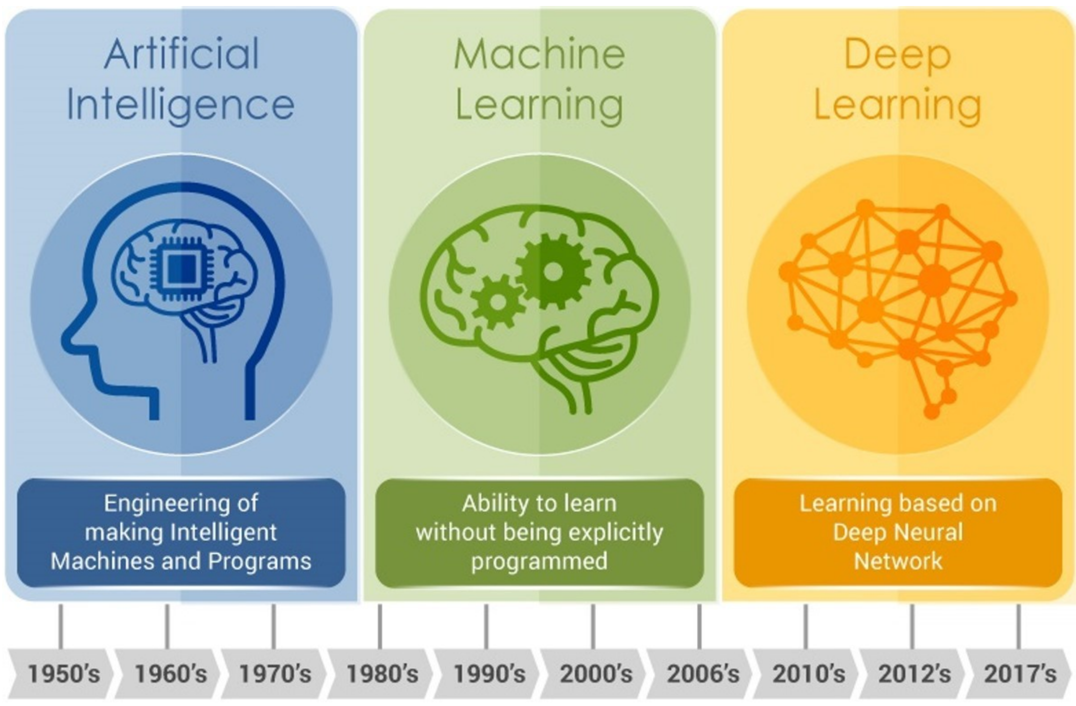
Chapter 1

Background

For the purpose of guaranteeing the essential background knowledge about the context related to our work, this chapter will provide a quick overview of the main concepts that we will refer to throughout the course of this dissertation. Precisely, in section 1.1 we will introduce the field of Machine Learning, with major emphasis to semi-supervised learning tasks and data augmentation techniques. Section 1.2 will be devoted to the description of constraint reasoning in machine learning scenarios, with the presentation of a wide range of state-of-the-art methodologies. Finally, section 1.3 will present a specific class of constraints, that is shape constraints, with a detailed description of both the theoretical and practical aspects of two of the predominant approaches employed to inject them into data-driven applications.

1.1 Machine Learning

With Machine Learning (ML) we define the area of Artificial Intelligence (AI) which involves all those algorithms that are not explicitly programmed to solve a task, but rather they can automatically improve their behaviour by learning from a set of structured data samples. Indeed, as opposed to symbolic techniques – e.g., expert systems –, which allow the machine to engage in a cognitive task via the declaration of high-level procedural rules often expressed in a logical framework, machine learning algorithms are called subsymbolic since they are statistical methods that maintain and iteratively update an internal state in order to best adapt to the given set of data. Finally, the recent explosion of the sub-field of Deep Learning (DL) in the community has brought a new series of challenges and opportunities, with successful applications in a variety of domains ranging from the medical to the epidemiological ones, as well


Artificial Intelligence
Machine Learning
Deep Learning
Engineering of making Intelligent Machines and Programs
Ability to learn without being explicitly programmed
Learning based on Deep Neural Network
1950's
1960's
1970's
1980's
1990's
2000's
2006's
2010's
2012's
2017's

(source: `www.linkedin.com/pulse/automated-bias-via-ai-try-human-algorithm-andy-mcleod`, last accessed on 28/06/2021.)

Figure 1.1: Historical perspective on Artificial Intelligence.

as the logistics and scheduling sectors. Along with this, various model structures and learning paradigms have been proposed and investigated.

From a statistical perspective, machine learning algorithms are able to learn the conditional distribution, with respect to a series of input features, of the event that generated the data points. This process is commonly formulated as a likelihood maximization, which can be mathematically implemented through an iterative procedure aimed at optimizing a predefined metric – either the maximization of a cost, or the minimization of a loss function. Additionally, depending on the structure of the input data, the learning paradigm may be diverse. The most common paradigms are:

Supervised Learning, in which the output labels are provided along with the data samples and explicitly employed in the learning algorithm. Accordingly, the loss function assumes the form $\mathcal{L}(y, \mathcal{M}(x, \theta))$, where y is the vector of ground truths, x the input samples, $\mathcal{M}$ the machine learning model, and θ the vector of its learnable parameters. This is the most commonly adopted approach for data-driven applications, and it is generally addressed via a learning procedure that exploits either likelihood maximization techniques, such as in Naive Bayes Classifiers or Decision Trees, or an iterative gradient-based technique, such as in Linear and Logistic Regression Models, and in Neural Networks.

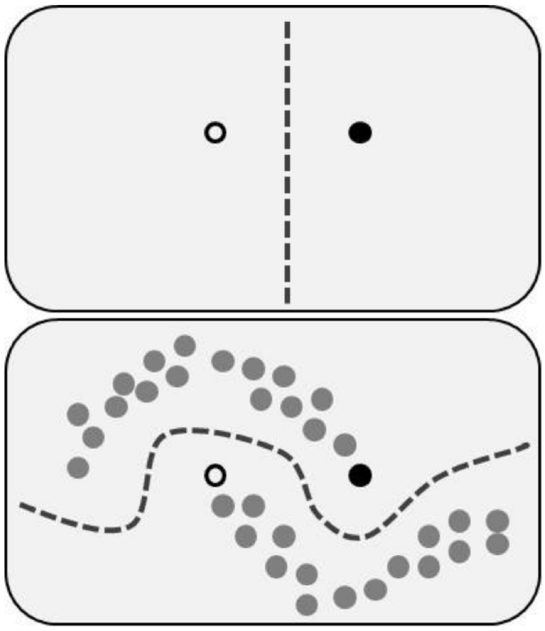
Unsupervised Learning, in which the output labels are not provided, thus the model is not required to return a prediction but rather to arrange the data points into coherent groups according to a specific similarity metric. The most common unsupervised tasks are clustering and anomaly detection. The former is aimed at partitioning the input data into (unlabelled) classes of objects that are near one another within the input space, and it is usually addressed via algorithms such as K-means and DBSCAN. The latter, instead, has the sole purpose of performing a binary partition of the dataset in order to separate outliers from the rest of the input data, and it is usually addressed via Density Estimation techniques, Neural Autoencoders, or Isolation Forests.

Reinforcement Learning, in which there are not output labels at all, but rather reward mechanisms provided by the external environment. In fact, this paradigm is primarily used to develop artificial agents which are placed inside this (usually simulated) environment and trained in a learn-by-doing fashion.

1.1.1 Semi-supervised Learning

Labelled datasets are notoriously difficult to obtain and maintain. There even exists cases in which the output of an event cannot be measured at all, in particular when we are dealing with high-cognitive tasks such as natural language processing or image recognition. In these scenarios, datasets may be manually annotated by human operators, a process which is not only time-consuming, but also inclined to the introduction of noise, errors and, depending on the context, even undesirable social biases [3]. Still, supervised learning techniques are preferable to unsupervised ones since they are more accurate and more precise – e.g., they can deal with regression tasks too, differently from unsupervised techniques that are intrinsically able to perform classifications only. For this reason, whenever known relationships in the underlying input distribution exist and can be exploited, semi-supervised learning might be adopted in order to train the model on a small set of supervised samples while regularizing the predictions thanks to a larger set of unsupervised ones.

Successful applications of semi-supervised learning have been proposed in various domainds. Among all, computer vision is one of them that most benefits from this paradigm, since manual labelling of images is particularly costly and error-prone. Besides, different techniques could be employed to take advantage of the patters in unlabelled data. For instance, Peikari et al. [4] propose a cluster-then-label approach for image classification in the medical



(source: `https://commons.wikimedia.org/wiki/File:Semi-supervised_learning.png`, last accessed on 28/06/2021.)

Figure 1.2: An example of the influence of unsupervised data (grey circles) in a classification scenarios with two classes (white and black circles). Considering the unsupervised points, previously gathered via a clustering algorithm, helps the model to better define the decision boundary.

domain, i.e., they run a clustering algorithm in order to identify groups of similar images, then use a Support Vector Machine to obtain a finer decision boundary for separating the classes. Alternatively, Rosenberg et al. [5] opt for a self-training approach, which consists in training a classifier on the small set of labelled data, then using it to predict the class of unlabelled data points, and eventually include the most confident unlabelled examples in the training set before iteratively retrain the model on a larger collection of data samples.

1.1.2 Data Augmentation

It is not always the case that unlabelled data is provided. Nevertheless, as it may still bring conspicuous benefits to the training process, it might be desirable to generate it from scratch. This process, known as data augmentation, consists of increasing the size of the training set by adding slightly modified copies of already existing data points or newly created synthetic ones obtained by randomly sampling the input space. Even though in some cases the augmented data can be instantly labelled under reasonable assumption – e.g., a common practice to avoid overfitting in computer vision tasks is to rotate, crop, or slightly warp an input image and insert it in the training set along with the label of the original one –, in the majority of applications this strategy must be paired with the above-mentioned semi-supervised techniques in order to regularize and, potentially, constraint the model predictions within a feasible region by exploiting the interrelation between samples.

1.2 Constrained Machine Learning

As opposed to subsymbolic techniques, there are symbolic ones. The term comprises many different algorithms which, nonetheless, share a common feature: they lack of a hidden state, but rather they leverage explicit procedures to find the solutions of some instances of a given problem. Some of these methods – e.g., constraint programming, linear and non-linear programming, mixed-integer programming, etc. – are strictly related to the operational research area. Indeed, they are aimed at solving Combinatorial Optimization tasks, namely those ones involving a finite set $S = \{x_1, ..., x_n\}$ in which each object $x \in S$ is linked to a cost – or a score – computed thanks to an objective function $f(x)$. To optimize $f(x)$ means to find the object $x^* \in S$ that returns the best value $f^*(x)$, which can be either its minimum or its maximum depending on the application.

A commonly acknowledge advantage of symbolic approaches is that constraints can be natively embedded into the definition of the set S, which represents in fact a feasible region within a larger, unbounded space – usually, $S \subseteq \mathbb{R}^k$ or $S \subseteq \mathbb{Z}^k$. On the contrary, machine learning systems are totally inappropriate for this kind of tasks since, by relying on a subsymbolic representation, there is no way to know which of their internal parameter to constraint in order to get a specific outcome and within which bounds. However, having guarantees on the outputs yielded by a data-driven model is a desirable feature in many contexts. For instance, depending on the domain area, we may want to enforce:

Fairness Constraints, i.e., those ones aimed at settling guarantees on the correct behaviour of the system with respect to minorities and other less-favoured social groups so to avoid unwanted discrimination.

Physical Constraints, i.e., those ones engaged to comply with laws from natural sciences, such as physics, chemistry, biology, etc.

Logical Constraints, i.e., those ones employed to respect logical inferences.

As formerly said, hybridization between symbolic and subsymbolic artificial intelligence techniques started to be explored and widely used in the last few years to allow for constraint satisfaction in machine learning environments. Besides, the primary aim of these approaches is not limited to the constraint enforcement, but rather, depending on the application, may serve the purpose of increasing both the model interpretability and its accuracy, particularly when constraints are employed as a form of knowledge injection – a meticulous taxonomy of the various techniques and sources of knowledge is presented in [6]. Among the variety of methodologies that have been proposed, we will now list some of those that share similarities with our presented approach.

Most of these constraint satisfaction methods rely on some kind of logical framework, within which the data-driven module is integrated in order to obtain an explicit formulation of the model that might be easily constrained without any further effort. For instance, Richardson and Domingos propose Markov Logic Networks [7], a framework to combine first-order logic and Markov networks which has been also extended in [8] in order to allow the handling of continuous variables as well. In Deep-ProbLog [9], Neural Networks with probabilistic outputs are embedded into a first-order logic program and treated as predicates, i.e., the data-driven module is used to extract high-level features – the classes – which are then used as inputs for symbolic reasoning. Finally, Rocktäschel and Riedel present a ”neural network for end-to-end differentiable proving of queries to knowledge bases by operating on dense vector representations of symbols” [10]. The main pitfall of these approaches is that, by directly constraining the output predictions via explicit rules, there is no effect on the learning procedure and, subsequently, on the inner state of the machine learning module. This might be undesirable both in terms of explainability and feature importance analysis. Additionally, relational constraints, i.e., those imposed on a group of predictions, become difficult to handle since they require access to the entire distribution rather than to each single output [11, 12].

Another group of approaches focuses solely on the pretraining step. This have been proposed and widely investigated particularly in the context of fairness constraints by Luong et al. [13] and Kamiran et al. [3]. Indeed, these methodologies are well-disposed towards the enforcement of relational constraints on the dataset in order to mitigate the effects of social biases. Then, once the data has been freed of all biases, it can be used to train a data-driven model with a standard procedure that requires no further intervention to ensure the satisfaction of the constraints. Still, a common drawback of these approaches is that it might happen that, somehow, it is either the model itself or the training algorithm that introduces the discrimination – e.g., due to the underrepresentation of certain groups –, thus preventing from getting close to the desired labels.

Furthermore, a third class of approaches is based on the regularization of the model at training time via some constraint-based expression. Diligenti et al. propose Semantic-based Regularization [14], a framework where constraints, which translate into fuzzy logical formulae, can express general prior knowledge about the environment and can be used to adjust the model's predictions. Similarly, Logic Tensor Networks [15] tries to implement *Real Logic* within a neural-network framework. In [16], relational background knowledge is used to enforce constraints on a set of unsupervised data, while the – smaller – set of supervised samples is kept to increase the system accuracy. The exploitation of unsupervised data points has been investigated as well in [17], where the theory behind Lagrangian duality is applied to the enforcement of complex constraints. Finally, domain-specific procedures related to the context of fairness have been proposed in [18–22]. Shortcomings about this group of approaches, however, include issues with relational constraints – that may eventually translate into serious problems when it comes to mini-batches –, and prior requirements on the properties of the constraints, such as differentiability. Likewise, numerical issues may worsen the training procedure up to the point that the approach becomes almost useless.

Finally, some ultimate methods may derive from the combination of previously mentioned techniques – e.g., Deep Structured Models [23, 24] –, or they may arise from novel integration paradigms aimed at solving specific issues – e.g., differentiability, as in [25].

1.3 Shape Constraints

Shape constraints are a specific kind of constraint that act on the silhouette of a function. They often come from an economic, physical, or biological domain, as well as from both stochastic and deterministic processes. The most common shape constraint enforced in machine learning is monotonicity [26], i.e., the output of the model is supposed to be monotonically increasing – or decreasing – with respect to one or more input features, all else being equal [27] – *ceteris paribus* condition. Nonetheless, many other constraints involving the shape of a function have been formulated and investigated. Cotter et al. [28] propose the concepts of Edgeworth Shape Constraints and Trapezoid Shape Constraints. In [29], Gupta et al. investigate the effect of diminishing returns – i.e., monotonicity constraints paired with concavity/convexity ones – on model regularization and improvement, while in [30] and [31], partial monotonic constraints and joint monotonic constraints on multidimensional functions are examined, respectively.

In this dissertation, we will focus solely on monotonicity constraints. Still, we need to distinguish between the concept of monotonicity in regression and classification tasks. In fact, when it comes to regression, increasing monotonicity[1] can be formalized as a relationship between pairs (x_1, x_2) of data samples in this way:

$$is\_monotonic(x_1, x_2) \implies f(x_1) \geq f(x_2), \text{with}$$

$$is\_monotonic(x_1, x_2) \iff x_1 = (x_1^M \mid x_1^{\neg M}) \wedge x_2 = (x_2^M \mid x_2^{\neg M}) \wedge x_1^{\neg M} = x_2^{\neg M} \wedge x_1^M \geq x_2^M$$

where x_i^M and $x_i^{\neg M}$ represent the attribute(s) involved in the expected monotonicity and the remaining attribute(s), respectively. Additionally, regarding the symbol "$\geq$" in the formula $x_1^M \geq x_2^M$, this has to be intended with a wider semantic meaning, not necessarily limited to its numerical definition.

On the contrary, when we think about classification tasks, we expect the model f to return a categorical value. Yet, the formulation of monotonicity constraints in classification tasks does not regard the output classes, but rather the output probabilities, in the sense that x_1 is expected to have a higher probability to be classified in a certain group with respect to x_2 but, depending on the thresholds, it may happen that they will eventually fall into the same class. Nonetheless, referring to binary classification tasks only – the only ones that we will address in the rest of our work –, the formulation of the constraint is the same one of the regression scenario as long as we consider f to yield the (binary) probability rather than the actual class.

Now that we have presented a formal definition of what a shape constraint is and which kinds do exist, we will proceed by providing two exemplifying techniques which are commonly adopted to handle them in data-driven applications, namely Semantic-based Regularization and Lattice Models.

1.3.1 Semantic-based Regularization

Semantic-based regularization techniques rely on the introduction of a penalty $\mathcal{P}$ in the loss function which accounts for the constraint violation. In our particular scenario, the penalty

[1]Monotonically decreasing constraints have an identical formulation, up to the switch of the inequality.

concerns pairs of data samples, thus it can be formulated as:

$$\mathcal{P}(x_i, x_j, \hat{y}_i, \hat{y}_j) = \max(0, (\hat{y}_j - \hat{y}_i) \cdot \mathit{monotonicity}(x_i, x_j))$$

where x_i and x_j are the input samples, and $\hat{y}_i$ and $\hat{y}_j$ their respective predictions yielded by the model. Regarding the function $\mathit{monotonicity}(x_i, x_j)$, it represents the sign of the expected monotonicity on the outputs, i.e., it returns a value $m \in \{-1, 0, 1\}$ which stands for decreasing, null, or increasing, respectively. Finally, the $\max$ function serves the purposes of ignoring already satisfied constraints; indeed, it acts as a barrier to filter out all the pairs for which $\hat{y}_j - \hat{y}_i$ is already respecting the expected constraint.

The total loss $\mathcal{L}$ of the data-driven model can be eventually formulated as a weighted sum of the standard supervised loss $\mathcal{S}$ – e.g., mean squared error, mean absolute error, negative log-likelihood, etc. – and the penalty $\mathcal{P}$ over all the samples and all the pairs:

$$\mathcal{L}(x, y, \hat{y}, \lambda) = \mathcal{S}(y, \hat{y}) + \lambda \cdot \sum_{(x_i, \hat{y}_i)} \mathcal{P}(x, x_i, \hat{y}, \hat{y}_i)$$

which degenerates into the standard loss function in case $\lambda = 0$.

This approach comes with three main burdens:

1. λ is a hyperparameter. It may be hard to tune it, especially in scenarios where the constraint is not injected as a form of knowledge but rather as a system requirement which may be in conflict with the final accuracy – e.g., fairness constraints.

2. When dealing with mini-batches, the number of exploitable relational constraints decrease dramatically. As well, it is necessary to take care of the batch generation process so that the set of data points in a single batch has a similar distribution to that of the entire set, an assumption which might be broken in various scenarios.

3. The penalty term may need to meet additional requirements. For instance, since the majority of data-driven models rely on gradient-based techniques, the loss function $\mathcal{L}$ must provide a gradient, a condition that can be satisfied only if all its parts are differentiable.

While there is no acknowledged solution to handle the latter, the former can be bypassed with a lagrangian dual approach, as proposed in [17]. Indeed, the loss function $\mathcal{L}$ can be seen as a relaxation of the constrained optimization problem:

$$\min_{\hat{y}}\{\mathcal{S}(y,\hat{y}) \mid \sum_{(x_i,\hat{y}_i)} \mathcal{P}(x,x_i,\hat{y},\hat{y}_i) = 0\}$$

It follows that, whenever the constraint is satisfied, then $\mathcal{L}(x,y,\hat{y},\lambda) = \mathcal{S}(y,\hat{y})$ holds independently from the value of λ. On the contrary, if the constraint is not satisfied, then the global minimum of the relaxed formulation will be certainly smaller – or equal – than the constrained formulation, otherwise there would have existed an even smaller value of $\mathcal{L}$ obtained by enforcing the constraint up to the feasibility. This means that, regardless of the value of λ:

$$\min_{\hat{y}}\{\mathcal{L}(x,y,\hat{y},\lambda)\} \leq \min_{\hat{y}}\{\mathcal{S}(y,\hat{y}) \mid \sum_{(x_i,\hat{y}_i)} \mathcal{P}(x,x_i,\hat{y},\hat{y}_i) = 0\}$$

namely, $\mathcal{L}(x,y,\hat{y},\lambda)$ represents a lower bound on the optimum of the constrained problem. Hence, the solution that mostly approaches the original constrained formulation is the one that assigns the higher weight to the penalty term, i.e., the one with maximal λ. Formally, this is equivalent to solving the problem:

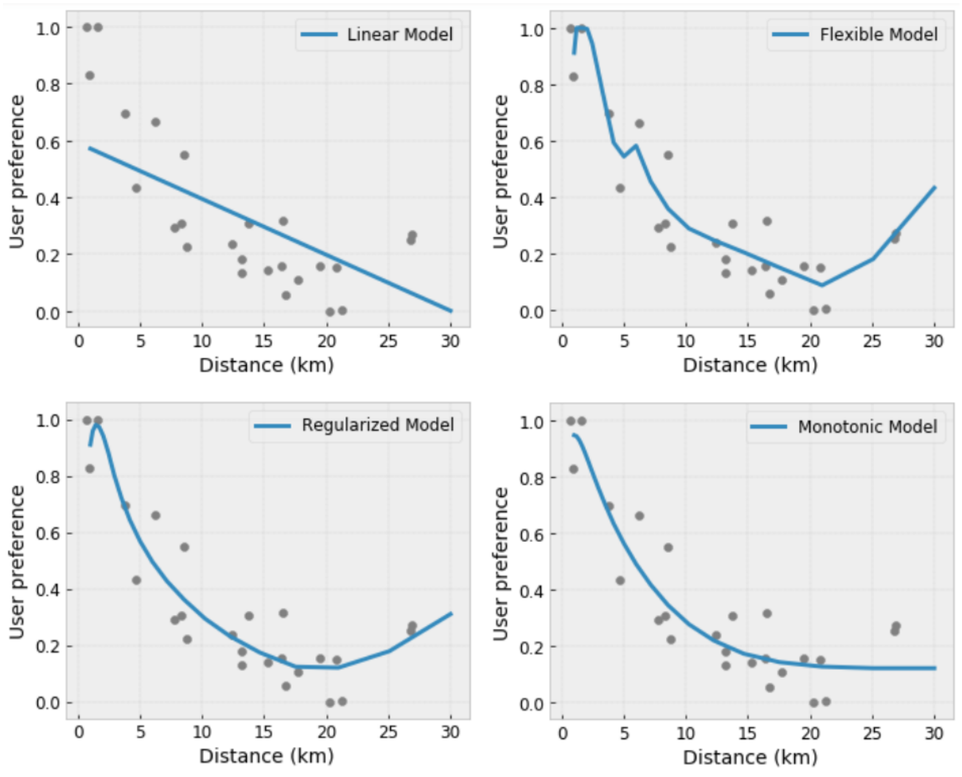
$$\max_{\lambda}\{\min_{\hat{y}}\{\mathcal{L}(x,y,\hat{y},\lambda)\}\}$$

which can be achieved via a two-step optimization process that exploits the gradient descent to minimize the loss function with respect to the predictions at first, and then maximizing the loss function with respect to λ.

1.3.2 Lattice Models

Lattices are data-driven models able to approximate arbitrary input-output relationships in a given set of data [26, 32]. They work as interpolated look-up tables, i.e., at training time, they learn the best values $\hat{y}_i$ associated to a set of knots $K = \{k_1, ..., k_m\}$ in the input space. Eventually, during prediction time, given an input vector x they predict its output $\hat{y}$ via linear interpolation from the y-values of the knots surrounding x. The most popular library to develop lattice models is `Tensorflow Lattice`[2], which provides plenty of layers and predefined estimators that can be easily integrated with the collection of `Keras` layers as well.

[2]`www.tensorflow.org/lattice`, last accessed on 28/06/2021.

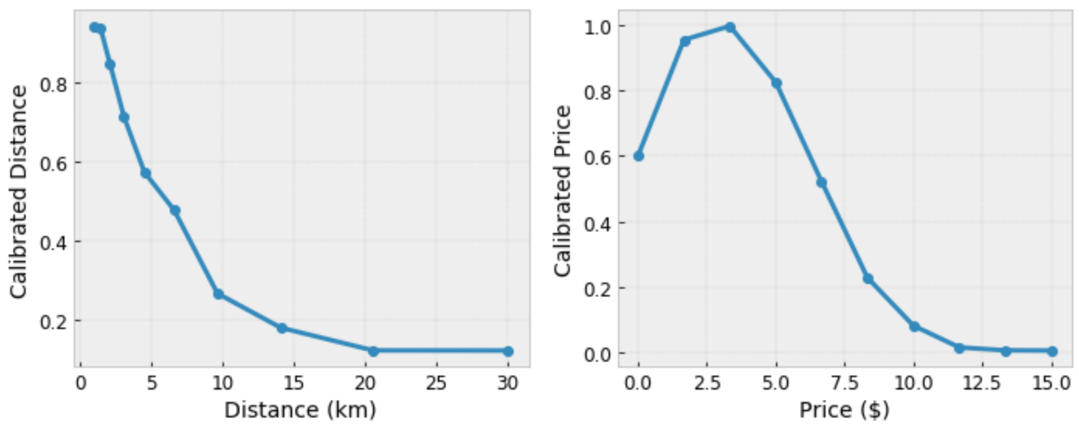


(source: `www.tensorflow.org/lattice/overview`, last accessed on 28/06/2021.)

Figure 1.3: Output of four lattice models having different number of parameters, different amount of regularization, and different constraint enforcement.

One of the main advantages of lattices is that they natively come with a support for regularization and shape constraints enforcement. More specifically, the injection of these constraints can be performed both on a single feature lever, and on the final output of the model – as shown in figure 1.3 –, allowing to deal with any kind of shape constraint previously exposed, from partial to joint ones, in a simpler and effortless way. The other advantage is that, since they are piecewise-linear, these models are easier to interpret. Moreover, as calibration layers can be inserted before the final lattice layer in order both to reduce the number of model parameters and to normalize the input features, it is possible to inspect the response curve of each calibrated feature before it enters the lattice layer, as shown in figure 1.4.

Still, there are some shortcomings about lattice models, mainly concerning the impossibility to rely on flexible knots – indeed, their values must be manually chosen, at least up to now – and, more importantly, scalability issues which prevent from the application of the technique on datasets with a high number of features. Nonetheless, their employment has been proven successful with respect to a wide range of regression and classification problems. In [30], Deep Lattice Networks are combined with partial monotonicities in order to achieve state-of-the-art performances while keeping shape guarantees. In [29], expert knowledge about diminishing



(source: `www.tensorflow.org/lattice/overview`, last accessed on 28/06/2021.)

Figure 1.4: Response curves after the calibration of two numeric features.

returns is applied on multiple regression tasks regarding the economic and social domains in order to increase the performances of the model and, simultaneously, obtain more regularized and interpretable results. Finally, in [33], monotonicity shape constraints are exploited to satisfy deontological requirements in classification tasks, as well as limiting the amount of unfair penalization within a fairness insurance framework.

Chapter 2

Moving Targets

Moving Targets is a state-of-the-art algorithm ”for injecting constraints at training time in supervised learning, based on decomposition and bi-level optimization” [2]. It is an iterative technique that relies on two alternated steps: a learner step, which trains the model on the given vector of labels, and a master step, which is in charge of enforcing the constraints by adjusting the labels. A more detailed review regarding the functioning of the method will be presented in section 2.1. Instead, section 2.2 will be focused on our contribution to extend the original formulation in order to handle semi-supervised learning and monotonicity shape constraints.

2.1 Original Formulation

Given a machine learning model f, a vector of supervised labels y^*, and a loss function $\mathcal{L}$, Moving Targets is aimed at solving the following constrained optimization problem:

$$\arg\min_{\theta}\{\mathcal{L}(y, y^*) \mid y = f(X;\theta), y \in C\}$$

where θ is the vector of learnable parameters of the model, and C is the feasible area, i.e., the region of the output space in which the constraints are satisfied. This problem can be reformulated *in pure label space* as:

$$\arg\min_{y}\{\mathcal{L}(y, y^*) \mid y \in C \cap B\}$$

where $B = \{y \mid \exists\theta, y = f(X;\theta)\}$ is the region of the output space that can be covered by the machine learning model. Intuitively, this means that we are no more interested in optimizing

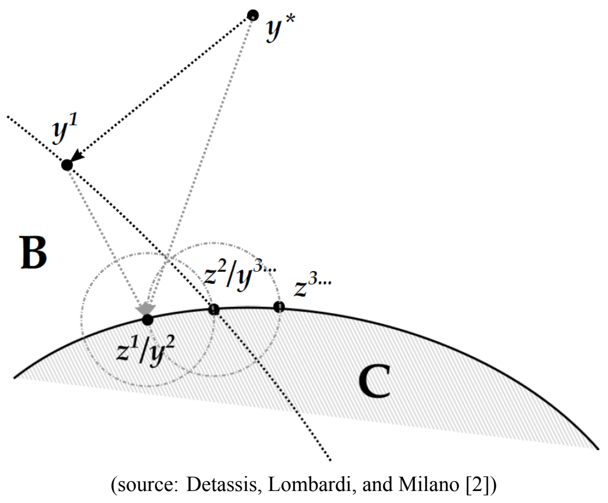


Figure 2.1: A sample run of Moving Targets involving Mean Squared Error loss and convex constraints and bias.

the model parameter, but rather we aim at optimizing the vector of predictions, which must be constrained into both the model's output space B – which is trivial since the predictions are generated precisely by that model – and the constrained output space C. Hence, the algorithm proceeds as described in algorithm 1: at the beginning, the vector of original labels y^* is used to *pretrain* the machine learning model; then, the obtained predictions are used to refine the original labels, which are in turn adopted to retrain the learner and obtain a new set of predictions, and so on.

The learning task is formulated straightforwardly, namely:

$$l(z) = \arg\min_y \{\mathcal{L}(y, z) \mid y \in B\}$$

where $l(\cdot)$ represents the data-driven module and z is the vector of supervised labels[1]. Regarding the master step, instead, the formulation varies depending on the feasibility of the i^{th} vector of predictions y^i returned by the data-driven model. Indeed, in case of infeasibility – i.e., $y^i \notin C$ –, the optimization problem is formulated as:

$$m_\alpha(y) = \arg\min_z \{\mathcal{L}(z, y^*) + \frac{1}{\alpha}\mathcal{L}(z, y) \mid z \in C\}$$

where $\alpha \in (0, +\infty)$ is a hyperparameter of the algorithm – allegedly, the problem is supposed to return a vector of labels z belonging to the feasible region C and close both to the original

[1]This will coincide with y^* during the pretraining step only, while in subsequent iterations it will be replaced by the i^{th} output of the master step, i.e., z^i.

Algorithm 1: Moving Targets

```
input : label vector $y^*$, scalar parameters $\alpha$; $\beta$; $n$
$y^1 = l(y^*)$              # pretraining
for $i$=1..$n$ do
    if $y^i \notin C$ then
        $z^i = m_\alpha(y^i)$    # infeasible master step
    else
        $z^i = m_\beta(y^i)$     # feasible master step
    end
    $y^{i+1} = l(z^i)$         # learner step
end
```

labels y^* and to the model's predictions y, up to that balancing factor. On the contrary, in case the outputs y^i are already feasible, the problem formulation is:

$$m_\beta(y) = \arg\min_z \{\mathcal{L}(z, y^*) \mid \mathcal{L}(z, y) \leq \beta, z \in C\}$$

where $\beta \in (0, +\infty)$ is another hyperparameter – in this case, we are not balancing between the two losses, but rather minimizing one of them while imposing the other one not to exceed a given amount β.

2.1.1 Algorithm Properties

As stated in [2], "due to the its open nature and minimal assumptions, establishing the convergence of our method is hard". Nevertheless, when the hyperparameters approach one of their bounds, the algorithm degenerates in a scenario of immediate convergence that is worth further analysis. In fact, for $\alpha \to 0$ or $\beta \to 0$, the master step becomes:

$$m_\alpha(y) = m_\beta(y) = \arg\min_z \{\mathcal{L}(z, y) \mid z \in C\}$$

i.e., since the original labels are discarded, there is no margin for accuracy improvement once the constraints get satisfied. Likewise, for $\alpha \to +\infty$ or $\beta \to +\infty$, it turns into:

$$m_\alpha(y) = m_\beta(y) = \arg\min_z \{\mathcal{L}(z, y^*) \mid z \in C\}$$

i.e., the model predictions are discarded this time, leading to a master step that will always return the same vector z^* which is the projection of the original labels y^* on the feasible region

C, preventing any for of iterative improvement. In particular, this latter case is essentially a preprocessing technique similar in fashion to the one proposed in [3].

Despite the complications to assess its convergence, the general nature of Moving Targets is still a strong advantage under many points of view. Indeed, the method poses no limits on the implementation of both the learner and the master. On the one hand, the former may be either a Linear or Logistic Regression Model, a Naive Bayes Classifier, a Random Forest, a Support Vector Machine, or even a Deep Neural Network; the sole requirement is the ability of the model to learn from supervised samples. On the other hand, the master step can be tackled via any kind of technique which is able to deal with numerical variables and constraints, such as Mixed-Interger Programming, Constraint Programming, or SAT Modulo Theories.

Finally, empirical investigations on the algorithm functioning has proven its efficiency and reliability on small and medium-sized datasets involving both regression and classification problems, with constraints ranging from class balance to fairness enforcement. Still, some issues remain. Firstly, scalability on very large datasets has not been deeply explored, but it is expected to be a tough challenge. Secondly, the role of the hyperparameters is still under examination, and it would be interesting to devise an adaptive procedure able to automatically tune α and β, as well as to develop two additional features already mentioned in the conclusive chapter of [2], namely the possibility to start the algorithm by directly projecting the targets into the feasible space rather than pretraining the learner, and the adoption of crossentropy loss on output probabilities rather than hamming distance on output classes in the master step. Thirdly, since Moving Targets has been advised as a mechanism to handle supervised tasks only, the original formulation provides no support to deal with unsupervised data. As far as this dissertation is concerned, we will focus solely on the last two issues; in particular, we will implement and evaluate the effects of those supplementary features, and eventually extend the use cases of Moving Targets by developing methodologies to manage semi-supervised learning and shape constraints, while retaining the same algorithmic core.

2.2 Extending the Algorithm

Our contribution on the extension of Moving Targets is two-fold. On the one hand, we intend to prove that the said method can be applied to semi-supervised learning tasks without any modification to the core of the algorithm, which maintains the same iterative structure with two alternated steps. In order to do so, we will need to introduce new hyperparamters that serve the purpose of balancing the influence of supervised and unsupervised data points during both the learner and the master step, as well as some other expedients to speed up the computation and get a higher accuracy. On the other hand, given that we focus on monotonicity shape constraints – which, as stated in section 1.3, need to satisfy the *ceteris paribus* condition in order to be enforced –, we propose as well a data augmentation procedure to generate a set of useful unsupervised samples.

2.2.1 Newly-introduced Hyperparameters

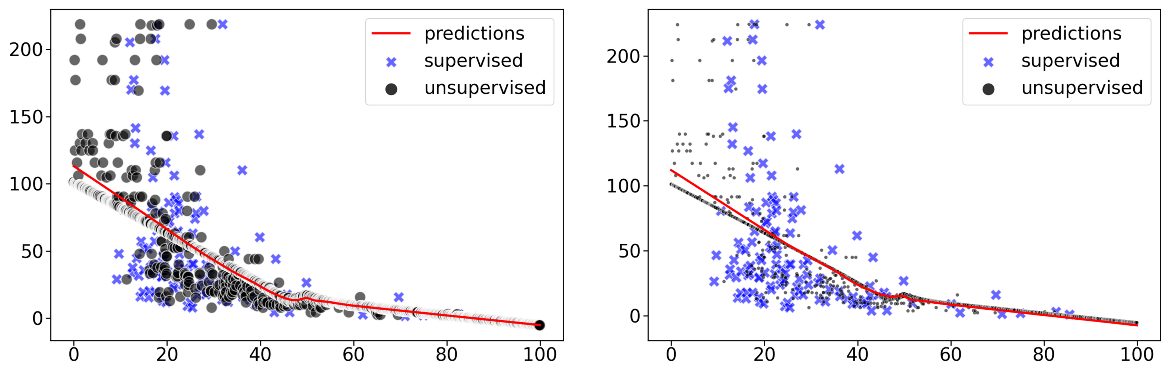
To the extent of our work, we will consider datasets having a form $D = D^S \cup D^U$, where $D^S = \{(x^s_1, y^s_1), ..., (x^s_n, y^s_n)\}$ is the subset of supervised examples, and $D^U = \{x^u_1, ..., x^u_m\}$ the subset of unsupervised ones. Even though we will later consider unsupervised samples generated via data augmentation techniques only, up to this point of the discussion we pose no limits neither on the provenance of unlabelled data nor on the kind of constraints involved.

Since Moving Targets proceeds by relabelling all the data points – with the sole exception of the pretraining phase in which the machine learning module is trained without any prior intervention of the master step –, from the first iteration on, each unlabelled sample will be provided a synthetic label y^u_i obtained by the enforcement of the constraints in the optimization problem. Nonetheless, the introduction of these new supervised data points poses some problems on how to consider them with respect to the original ones. Indeed, due to the absence of authentic ground truths to balance the effects of the model's predictions and the constraints, the relabelling process performed by the master may have pushed away the new labels from a reasonable region of the output space. Therefore, in order to maintain some degrees of freedom on how to weight the unsupervised data points, we propose three new hyperparameters:

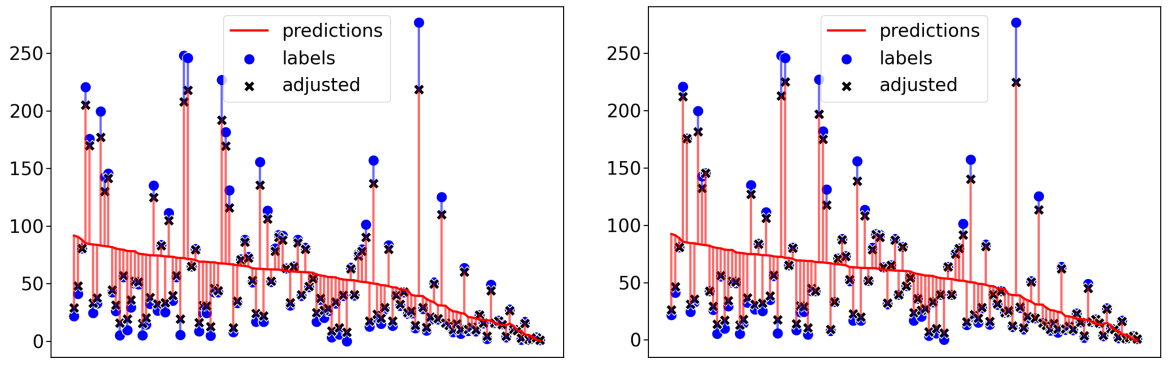
Learner Omega, a positive real number ω_l that balances the weight of supervised samples with respect to unsupervised ones during the learning phase. Practically, given the loss function $\mathcal{L}$ of the learner l, instead of averaging the losses of each sample independently from its kind, the final value will be computed as:

$$\mathcal{L}(D_i) = \sum_{(x^s, y^s) \in D^S_i} \frac{\mathcal{L}(l(x^s), y^s)}{\mid D^S_i \mid} + \frac{1}{\omega_l} \cdot \sum_{(x^u, y^u) \in D^U_i} \frac{\mathcal{L}(l(x^u), y^u)}{\mid D^U_i \mid}$$

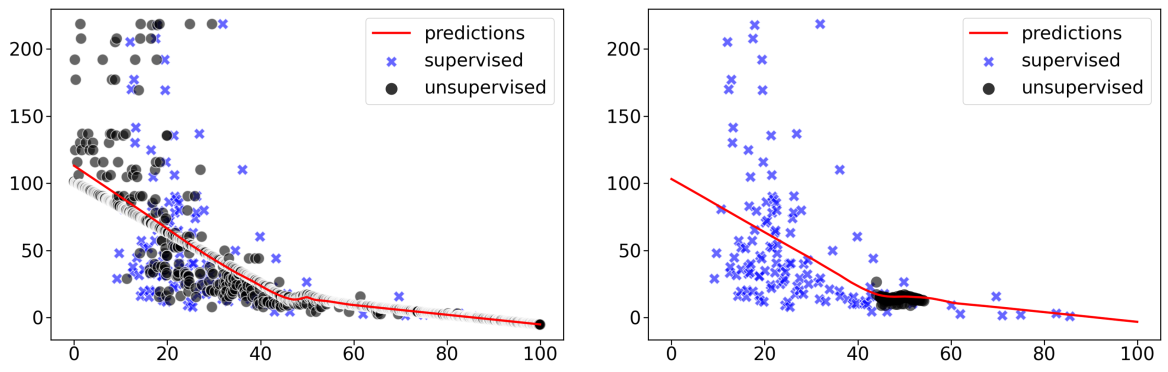
where D^S_i and D^U_i are the two datasets containing the supervised and the (originally) un-



(a) Adjusted labels adopting the standard configuration (left) and the same one with `learner_omega = 10` (right). Since in the second case unsupervised samples (black circles) are given less weight than supervised ones (blue crosses), the resulting predictions from the learner (red line) stays lower.



(b) Distance between the original labels (blue circles), the adjusted labels (black crosses), and the predictions (red line) using the standard configuration (left) and the same one with `master_omega = 100`. In the second scenario, unsupervised samples are given more weight than supervised ones, thus the adjusted labels for supervised samples are nearer to the original targets rather than to the learner's prediction.



(c) Adjusted labels for the standard configuration (left) and the same configuration with `learner_weights = 'infeasible'` (right). As in the second scenario only infeasible unsupervised samples (black circles) are considered, their presence is concentrated in those areas.

Figure 2.2: Effect of the hyperparameters after one iteration in a univariate, monotonically-decreasing dataset. The standard configuration (left) has values `learner_omega = 1`, `master_omega = 1`, and `learner_weights = 'all'`.

supervised samples, respectively, at the i^{th} iteration of Moving Targets, thus containing in both cases the adjusted labels. The parameter ω_l can be used to express a degree of confidence in the relabelling process carried out by the master. Indeed, since it represents the ratio between the weights of supervised and unsupervised samples, choosing a lower value indicates higher confidence, while choosing a higher value means to trust unsupervised labels less, up to the limit value of $+\infty$ where the unsupervised samples are completely ignored, leading to a degeneration into the original Moving Target's formulation.

Master Omega, namely the analogue of ω_l for the master step. It is a positive real number too (ω_m), and it balances the weight of supervised and unsupervised samples in the master's *p_loss*, i.e., the loss $\mathcal{L}(z, y)$ computed between the adjusted targets z and the learner's prediction y. In this case, ω_m represents an inverse ratio, thus the loss value will be computed as:

$$\mathcal{L}(D_i) = \sum_{(x^s,y^s)\in D_i^S} \frac{\mathcal{L}(m(x^s), y^s)}{\mid D_i^S \mid} + \omega_m \cdot \sum_{(x^u,y^u)\in D_i^U} \frac{\mathcal{L}(m(x^u), y^u)}{\mid D_i^U \mid}$$

where $m(x)$ represents the adjusted label z returned by the master step. This parameter has an effect which approximates that of α any time that $\mid D^U \mid \gg \mid D^S \mid$ holds. In fact, with a much greater number of unsupervised samples with respect to supervised ones, the *p_loss* is almost completely dominated by the unsupervised; on the contrary, given that the *y_loss* is computed between the original targets and the adjusted ones, the unsupervised samples cannot influence it since they lack of a label. Still, whenever this is not the case, ω_m allows to better control the influence of the unsupervised samples during the relabelling process.

Learner Weights, a parameter that influences which and how many unsupervised samples should be considered during the learning process. In particular, one option is to use *all* the unsupervised samples, along with their new labels, while the other one is to use only those samples which have somehow resulted *infeasible* at least once during the entire progression of the algorithm. The latter option allows to reduce the number of data points used in the subsequent training phases in order both to speed up the computation and to limit the influence of unsupervised samples only around those regions in which some regularization is needed. Internally, the model has a data structure that keeps track

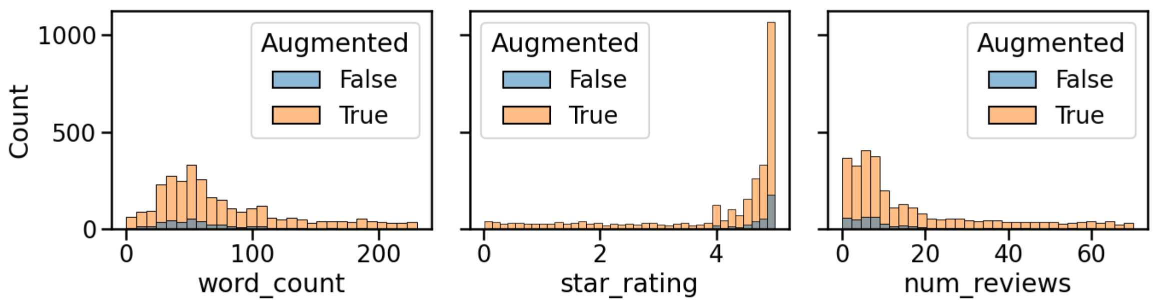


Figure 2.3: Distribution of original (blue) and random/augmented samples (orange) in a dataset with three monotonic features. The introduction of random data points allows to cover even the regions of the input space where no supervised data was present.

of the unsupervised samples which have contributed to increase the level of violations in at least one iteration: those samples only will be adopted in the training phase, with respective weight $\frac{1}{\omega_l}$. Trivially, it follows that the number of adopted unsupervised points cannot decrease during the execution of the algorithm.

Finally, one last investigated hyperparameter – **Warm Start** – introduces the possibility to maintain the internal status of the data-driven model throughout the iterations of Moving Targets. Ideally, since the learner is originally trained on the supervised samples during the pretraining step, embracing a warm start technique may help to regularize the infeasible regions only, without altering too much the already feasible ones.

2.2.2 Data Augmentation & Monotonicities

Since our purpose is to concentrate solely on the extension of the algorithm, we opted for keeping the data augmentation procedure detached from it. Namely, the process is run *before* the execution of Moving Targets rather then during its iterations, and the purpose of the data augmentation is merely to generate a set D^A of – unsupervised – augmented samples from the original dataset D^S as if it was already available.

In order to do so, we proceed by creating n_a copies of all the input samples x for each input feature $f \in F_C$ on which a constraint can be imposed, for a total of $n_a \cdot \mid F_C \mid \cdot \mid D^S \mid$ augmented data points. Moreover, some additional unlabelled data point might be generated as well prior to the core of the augmentation procedure in order to better fill the input space and allow for the enforcement of constraints even in uncovered areas, as in the case exposed in figure 2.3. Then, as clarified in algorithm 2, each of the newly generated record is modi-

Algorithm 2: Data Augmentation

```
input: dataset $D^S$, features $F$; # random $n_r$, # augmented $n_a$; sampling function $s$
$D^R = \emptyset$        # dataset of random samples
$D^A = \emptyset$        # dataset of augmented samples
# create $n_r$ new unlabelled data points by
# sampling a random value for each input
# feature $f$, then add it to $D^R$
for $f \in F$ do
    $y^r = \emptyset$
    $x^r = \emptyset$
    for $i$=1..$n_r$ do
        $\pi_f(x^r) = s(f)$
    end
    $D^R = D^R \cup \{(x^r, y^r)\}$
end
$D^S = D^S \cup D^R$  # merge the two datasets
# create a pool of $n_a$ new unlabelled samples by copying an
# original/random data point $x$ and sampling a different
# value only for each feature $f \in F_C$ for which a
# constraint can be imposed, then add it to $D^A$
for $(x, y) \in D^S$ do
    for $f \in F_C$ do
        for $i$=1..$n_a$ do
            $y^a = \emptyset$
            $x^a = \pi_{\neg a}(x) \bowtie s(a)$
            $D^A = D^A \cup \{(x^a, y^a)\}$
        end
    end
end
return $D^S \cup D^A$
```

fied only in the field regarding the specific feature f via the sampling function s, so that the *ceteris paribus* condition is satisfied. On a side note about s, even though the literature provides plenty of different sampling methods, we opted for leveraging either a standard *sampling with replacement* technique in case of non-constrained features or a *random uniform sampling* technique in case of constrained ones – in order to generate samples within upper and lower bounds selected accordingly to the input range –, so that the process would have been kept as simple as possible.

Furthermore, respectively to our specific task, we needed to address the computation of expected monotonicities in order to inject them during the master step. As the dataset D is not modified throughout the iterative process – except for the labels which, however, are not

Loss Function	Expression	Label Space
Mean Absolute Error	$\frac{1}{n} \mid\mid z - y^* \mid\mid_1$	$\mathbb{R}^n$
Mean Squared Error	$\frac{1}{n} \mid\mid z - y^* \mid\mid_2^2$	$\mathbb{R}^n$

Table 2.1: Loss functions for regression tasks (n = # examples)

involved in the definition of the constraints –, expected monotonicities can be computed just once, at the beginning of the process. In particular, regarding the computation per se, we tested three different methodologies, namely:

Ground, where a constraint is imposed solely between each augmented sample $(x^a, \emptyset) \in D^A$ and its respective original sample $(x, y) \in D^S$ from which it was generated[2]. A monotonicity between the two data points is always expected to exists, since the augmented point differs from the original one for exactly one feature f. It follows that the total number of constraints will be the same as the size $\mid D^A \mid = n_a \cdot \mid F_C \mid \cdot \mid D^S \mid$ of the augmented dataset.

Group, where a constraint is imposed between each pair of data points that belong to the group of samples augmented from the same original one. Since, for any constrainable feature $f \in F_C$, the process generates n_a augmented samples – plus the original one itself – that satisfy the *ceteris paribus* condition with respect to f, the total number of constraints will be $\frac{(n_a+1)\cdot n_a}{2} \cdot \mid F_C \mid \cdot \mid D^S \mid$, which is $\frac{n_a+1}{2}$ times greater than the previous one.

All, where a constraint is imposed between all the pairs in $D = D^S \cup D^A$. In this case, it is difficult to identify in advance the total number of constraints due to the presence of many different data points which, usually, will hardly satisfy the *ceteris paribus* condition with respect to a single feature f. Especially, in case of multiple real-valued features, we can expect this number to be $\frac{(n_a+1)\cdot n_a}{2} \cdot \mid F_C \mid \cdot \mid D^S \mid + \varepsilon$, where $\varepsilon \in \mathbb{N}$ is a small integer.

2.2.3 Regression vs. Classification

If we focus on the specific constrained problems that we have chosen, i.e., those ones regarding monotonicity shape constraints, we need to make a distinction between regression and (binary) classification tasks. In particular, once the formulation is perfected with the addition

[2]With D^S we indicate the dataset containing both the original supervised samples and the potential randomly-generated samples which might be introduced in the set at the beginning of the data augmentation procedure.

Loss Function	Expression	Label Space
Hamming Distance	$\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}[z_i \neq y_i^*]$	$\{0,1\}^n$
Crossentropy	$-\frac{1}{n}\sum_{i=1}^{n}[z_i \cdot \log y_i^* + (1-z_i)\cdot\log(1-y_i^*)]$	$\{0,1\}^n$
Reversed Crossentropy	$-\frac{1}{n}\sum_{i=1}^{n}[y_i^* \cdot \log z_i + (1-y_i^*)\cdot\log(1-z_i)]$	$[0,1]^n$
Symmetric Crossentropy	$-\frac{1}{n}\sum_{i=1}^{n}[z_i \cdot \log y_i^* + (1-z_i)\cdot\log(1-y_i^*) + y_i^* \cdot \log z_i + (1-y_i^*)\cdot\log(1-z_i)]$	$[0,1]^n$

Table 2.2: Loss functions for binary classification tasks (n = # examples)

of the constraints originated from the expected monotonicities, in the regression scenario we obtain a quite straight-forward master problem which can be either a Linear or a Quadratic Program depending on the chosen loss function, as shown in table 2.1.

On the contrary, as previously exposed in section 1.3, the formulation is rather more complicated in the classification scenario, since the semantics of the constraints concerns the output probabilities instead of the output classes. This means that we aim at obtaining continuous targets from the master step, even though our original labels were categorical[3]. It should follow that, of all the loss functions shown in table 2.2, we may want to rely either on Reversed or on Symmetric Crossentropy, which are the only ones allowing for continuous targets[4]. Still, these two loss functions come with a considerable burden, namely the variables that must be optimized are under a logarithm operator, which causes a way more difficult management of the optimization problem, since logarithms are usually handled by solvers via piecewise linearization, leading to a larger instance and a less precise solution. A possible workaround to bypass this issue is to consider the classification task on par with a regression one, being the formulation exactly the same. Specifically, one solution may be to use Mean Squared (or Absolute) Error in the master step, and Binary Crossentropy in the learner step. Otherwise, since some investigations about exploiting the MAE loss on classification tasks in order to better handle noisy data have been carried out with positive results [34, 35], another option might be to adopt a regression loss in both the steps. Nevertheless, a more detailed discussion on the effects of these choices will be presented in the next chapter, where the results of our empirical evaluation are presented.

[3]This is not such a troublesome issue in the majority of the cases, since the most adopted loss function for binary classification problems in data-driven systems is the crossetropy loss, which can handle continuous targets as well. Still, there may be cases in which this behaviour is not particularly suitable for the application.

[4]Take in mind that, in this scenario, the variables that we want to optimize are the adjusted targets z.

Chapter 3

Empirical Evaluation

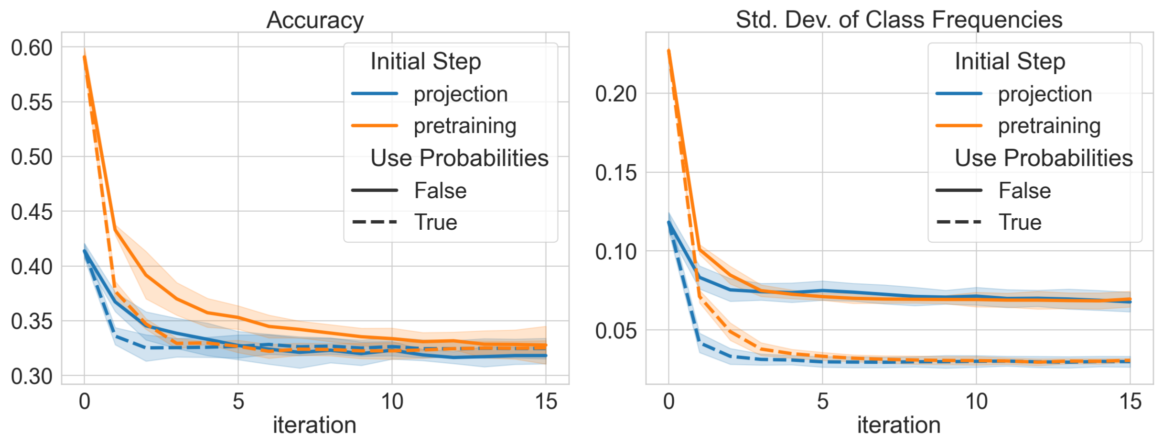
Now that we have introduced the main concepts related to our contribution, we can proceed with some empirical tests to evaluate it. In particular, section 3.1 will contain some preliminary analysis of the two additional features added to the original formulation of Moving Targets. Then, in section 3.2 we will focus on the core of our contribution by assessing the role of the hyperparameters that we introduced to deal with semi-supervised learning. Finally, section 3.3 will provide the outcomes of a benchmarking process in which we compared our method to other common data-driven approaches.

Our setup consists in an Intel Core i7-8565U laptop with 8 GB RAM. For the experiments in section 3.1 we use `CPLEX 12.8` to solve the master problems and a Linear or Logistic Regression model from `Scikit-Learn 0.24` as learner. Instead, experiments in sections 3.2 and 3.3 rely on `Gurobi 9.1` and on a Multi-layer Perceptron with two hidden layer of 128 units implemented using `Tensorflow 2.4`, respectively.

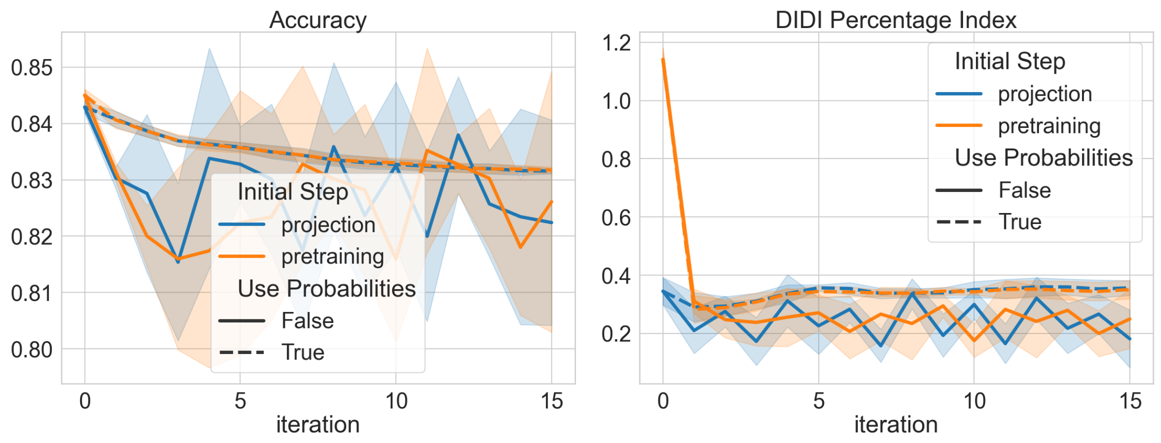
The full results related to each group of tests are available at `www.wandb.ai/giuluck/mt_preliminary`, `www.wandb.ai/giuluck/mt_tuning`, and `www.wandb.ai/giuluck/mt_benchmarking`, respectively.

3.1 Preliminary Studies

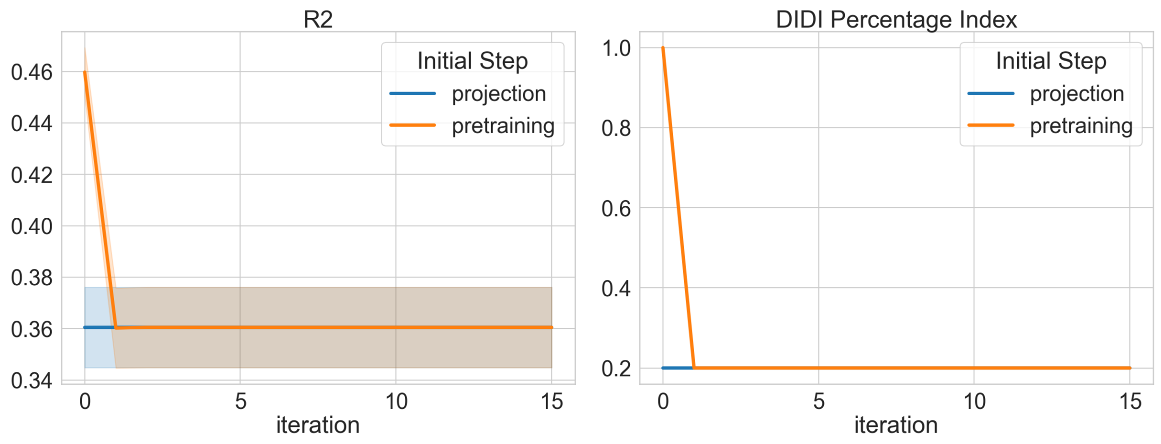
Before diving into the semi-supervised extension, we implemented and investigated the role of the two features already mentioned in subsection 2.1.1, namely the possibility to skip the `pretraining` step, thus directly start the algorithm with a `projection`, and the adoption of output `probabilities` via crossentropy loss rather than output `classes` via hamming dis-



(a) Training history for the `redwine` dataset. While the accuracy converges to the same value, independently from the configuration, adopting the probabilities allows to reduce the level of violation, measured as the standard deviation of the frequencies of the output classes.



(b) Training history fro the `adult` dataset. Even though probabilities lead to worse results, it is preferable to adopt them since the trends are less subjected to fluctuations and stochasticity.



(c) Training history for the `crime` dataset – since this is a regression task, no probability is involved. This test confirms that there is no difference among the initial steps, both in terms of accuracy and violations, measured by the Disparate Impact Discrimination Index (DIDI) [18].

Figure 3.1: Accuracy and constraint satisfaction on three datasets considered in [2].

tance. For this purpose, we compared the training history of four different configurations – namely all the combinations involving the two initial steps and the probabilities/classes choice – on the seven datasets considered in [2][1].

Figure 3.1 shows the outcomes of our experiments. We trained a Moving Target instance for 15 iterations, then we aggregated the results of each iteration on the average value of 5 *cross-validated* folds. Results demonstrate how the initial step is not such a decisive hyperparameter, since both the `pretraining` and the `projection` scenarios converge to the same solution within a reasonable amount of iterations. On the contrary, adopting the output probabilities rather than the output classes leads to a stabler process, which is not only less influenced by fluctuations, but in certain cases may even guarantee a better satisfaction of the constraints.

3.2 Hyperparameters Investigation

In this section, we will investigate the role of the algorithm's hyperparameters respectively to semi-supervised tasks involving monotonicity shape constraints. In order to assess the quality of the solutions, we rely on two synthetic datasets:

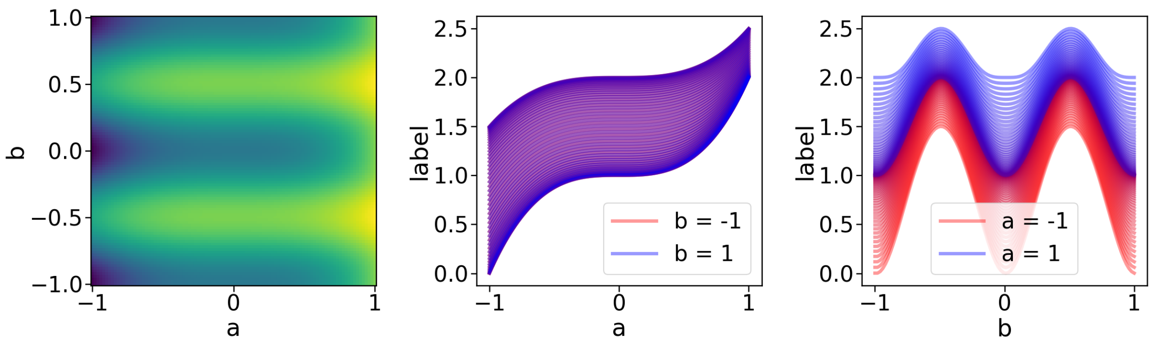
Synthetic, a bivariate dataset with a numerical output in which data points are sampled from the function $f(a,b) = \frac{a^3}{1+\sin^2(\pi b - 0.01\pi))} + \sin^2(\pi b - 0.01\pi) + 1$, with $a, b \in [-1, 1]$. The expected (increasing) monotonicity regards the attribute a only, as evident from figure 3.2a. Moreover, the train and test sets have different distributions; indeed, while a is sampled from a uniform distribution $\mathcal{U}(-1, 1)$ in the latter, in the former those values are obtained from a normal distribution $\mathcal{N}(0, 0.3)$, thus leading to a concentration of the data points in the central region[2].

Restaurants, a classification dataset aimed at computing the click-through rate of a restaurant on dedicated websites for comparisons and suggestions – e.g., Tripadvisor. The dataset has been proposed in `www.tensorflow.org/lattice/tutorials/shape_constraints`[3], and it involves three partial monotonicities, one regarding a categorical feature and two regarding numerical ones. As before, train and test sets have different distributions to simulate the fact that training data is usually available for restaurants with a high click-through rate only.

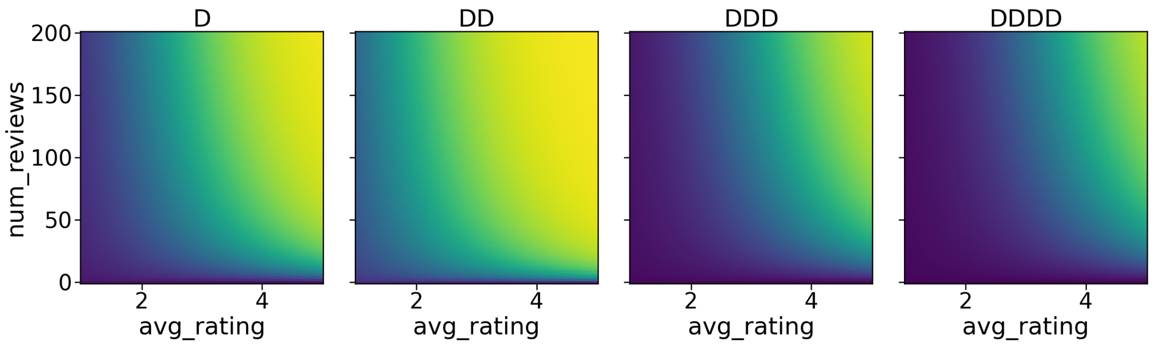
[1]The other hyperparameters, α and β, were kept to the default value of 1.

[2]The attribute b is uniformly sampled in both cases.

[3]Last accessed on 03/07/2021.



(a) Function output (left) and marginalized responses for each individual input feature.



(b) Theoretical click-through rates with respect to the four categories of restaurants.

Figure 3.2: Ground truths for the two synthetic datasets.

In order to assess the importance of each parameter, we proceed by running three consecutive experiments involving different subsets of parameters, since a unique factorial design would be prohibitive in terms of computational effort. For each tested configuration, we use the following experimental procedure:

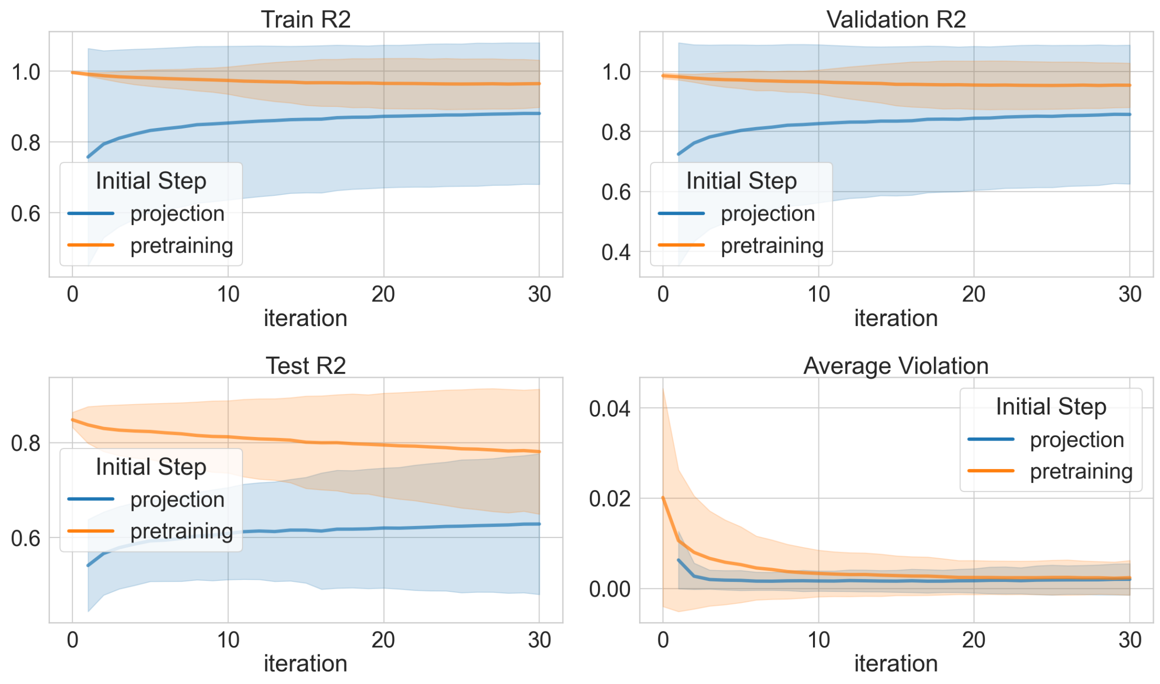
1. we partition the dataset into a training and a test set;
2. we augment the test set in order to dispose of data subsets satisfying the *ceteris paribus* condition with respect to a feature, which will be eventually used to compute the level of constraint violation;
3. we run a *5-fold cross-validation* procedure on the training set;
4. at the end of each iteration, we measure both the loss and the accuracy for the train, validation, and test sets, as well as the average level of violation and number of violations with respect to the augmented test set;
5. we collect the results of each fold and aggregate on the mean.

3.2.1 Original Hyperparameters

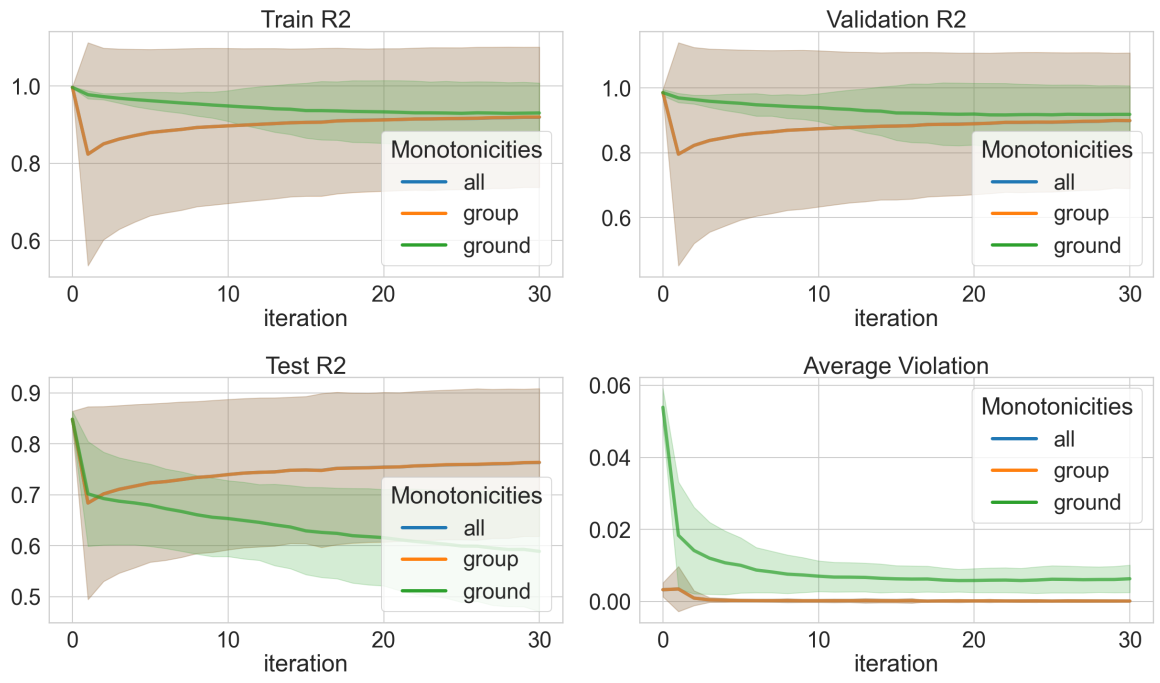
Firstly, we investigate the effects of the hyperparameters that were already present in the original formulation of the algorithm, or that are not strictly inherent to it, namely: α, β, the `initial step`, the `master loss`, and the `kind` of monotonicities computation procedure – i.e., `ground`, `group`, and `all`. The total number of configurations obtained by this factorial design is 144, which brings to $144 \cdot 5 = 720$ runs since we perform *5-fold cross-validation*; still, for the sake of the experiment, we remove two runs in which the learner yields completely wrong predictions at a certain iteration, leading to biased average results that are abnormally out of scale.

The outcomes show a slight, though expected, variation of both the solution quality and the level of constraint violation depending on α. Similarly, even though it seems not to have an impact on the number of violations, the choice of the `master loss` slightly alters the metric score, thus further investigation on the parameter will be carried out. On the contrary, there is no significant difference in the results respectively to the value of β, nor if we do not adopt the beta step at all: for this reason, we will not use the beta step anymore from now on in order to limit the degrees of freedom for the configuration.

Nonetheless, some differences rise up when examining the behaviour of the other two parameters. As demonstrated in figure 3.3a, starting with a projection step is not as effective as it was in the previous tests. In fact, even though the trends suggest that in both cases the algorithm will eventually converge to the same values if enough time is provided, in the `pretraining` scenario the level of violations is almost the same but there metric scores are higher since the first iterations – by any means, this is in line with our expectations, as in this case the unlabelled may bring the algorithm towards unreasonable regions if no pretraining step balances the injected constraints. Furthermore, figure 3.3b shows how significantly higher violations arise when monotonicities are imposed with respect to the ground data point only. On the contrary, the value is practically the same in the `group` and `all` configurations, thus it is preferable to use the former since it requires $\mathcal{O}(k^2 \cdot N)$ instead of $\mathcal{O}(k^2 \cdot N^2)$ time to compute the expected monotonicities, where $N =\mid D^S \mid$ is the cardinality of the original dataset and $k = n_a \cdot \mid F_C \mid$ is the number of augmented samples respectively to each original data point.



(a) Aggregated results with respect to the `initial step`.



(b) Aggregated results with respect to the `monotonicities kind`.

Figure 3.3: Outcomes of the investigation on the original hyperparameters.

3.2.2 Newly-introduced Hyperparameters

A further investigation is carried out on the same dataset, but it is focused on the hyperparameters introduced specifically to deal with semi-supervised tasks – i.e., `learner omega`, `master omega`, `learner weights`, and `warm start` –, plus α and `master loss`, whose role could not be unequivocally established from the previous experiment.

Figure 3.4a shows how, contrarily to our expectations, giving more weights to unsupervised data points allows not only to reduce the number of violations, but also to increase the accuracy scores for all the three splits. Similarly, in figure 3.4b we can notice how `mse` seems to be a more effective and more robust loss function for the master step, even though the number – and the distribution – of violations is similar to the `mae` cases.

Regarding the other parameters, no significant difference was found between the runs. We can interpret this outcome as a confirmation that our algorithm is rather general and robust, since it is not particularly influenced by the chosen configuration. Still, as well as before, the unique exception regards α which, being the most delicate parameter, is able to considerably influence the obtained results.

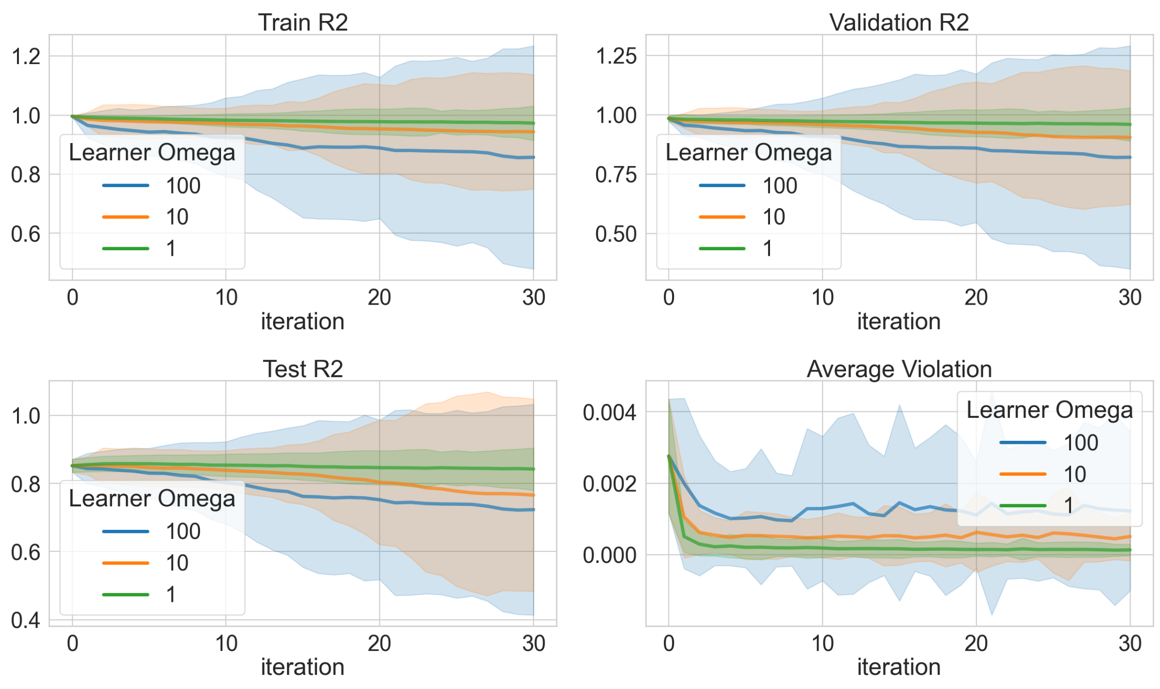
3.2.3 Classification Losses

In addition to the tests on regression tasks, we run a last factorial design experiment to understand the effect of the losses presented in table 2.1 and 2.2 on classification tasks. Thus, while keeping the three values of α as usual (0.01, 0.1, and 1), we analyze the four classification losses for the master step, combined with `binary crossentropy` for the learner, plus the two regression losses for the master step combined with both `binary crossentropy` and `mean squared error` for the learner[4].

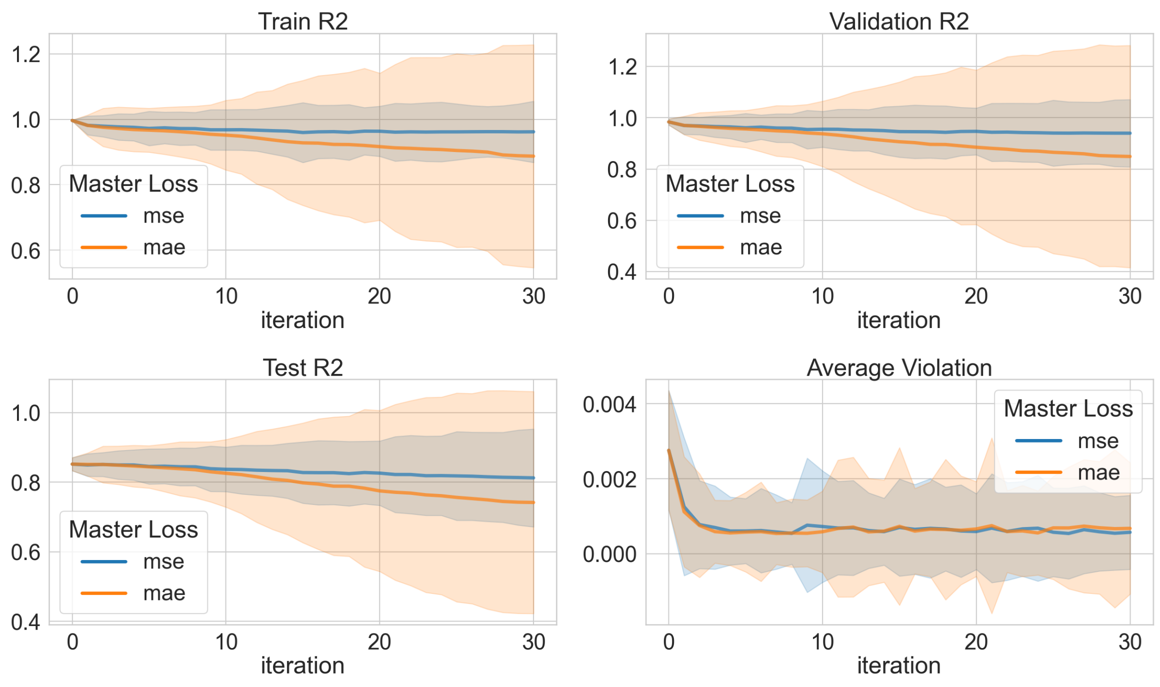
Figure 3.5 shows the outcomes of this experiment. Even though the best approach seems to be the adoption of the `reversed binary crossentropy` in the master step, the improvement with respect to `mean squared error` is so minimal that does not justify the longer training times, especially if considering that the former scales much worse than the latter. Moreover, `mse` behaves better than `mae` even in classification scenarios.

A conclusive notable result, which did not follow our expectations, is the fact that using different losses in the two steps – e.g., `bce` for the learner, and `mse` for the master – does not

[4]In any case, the learner always has a *sigmoid* activation on the output neuron.



(a) Aggregated results with respect to `learner omega`.



(b) Aggregated results with respect to the `master loss`.

Figure 3.4: Outcomes of the investigation on the newly-introduced hyperparameters.

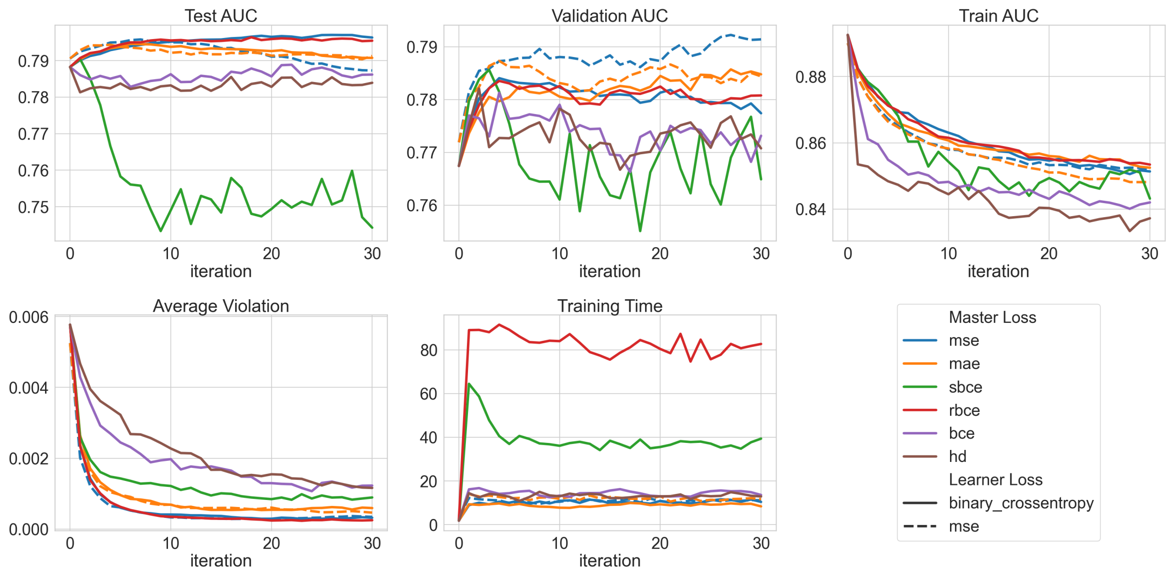


Figure 3.5: Outcomes of the investigation on the classification losses.

undermines the convergence of the method. On the contrary, adopting the crossentropy for the learner seems to be a more reasonable choice even when the master problem is formulated as a regression task.

3.3 Benchmarking

As a final assessment, we demonstrate the applicability of our proposed method both on the two previously-introduced synthetic datasets, and on two additional regression and binary classification real-world datasets taken from [29] and [33], respectively. Additionally, for these real-world problems, we test two scenarios: an easier one (***Partial Features***), in which a small subset of the input features is used, and a regular one (***Full Features***), where all the features at disposal are adopted.

For each of these tasks, we compare the results of three instances of Moving Targets (**MT**), using the default configuration and three different α values (0.01, 0.1, and 1), with just as many other common data-driven techniques, namely:

MLP an unconstrained multi-layer perceptron, with two hidden layers having 128 units each – this is used, too, as *learner* in the Moving Target's process.

SBR a semantich-based regularized version of that same MLP, with automatic tuning of the parameter λ as discussed in subsetion 1.3.1.

	MLP	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test R^2	0.81465	**0.89365**	0.82484	0.84176	0.84722	0.85654
Test MSE	0.03636	**0.02086**	0.03436	0.03104	0.02997	0.02815
Val R^2	0.97941	0.98304	0.96280	0.97619	**0.98338**	0.97992
Val MSE	0.00251	**0.00200**	0.00467	0.00292	0.00205	0.00248
Train R^2	0.99349	**0.99508**	0.96771	0.98609	0.99174	0.99158
Train MSE	0.00082	**0.00062**	0.00408	0.00175	0.00104	0.00106
Avg. Violation	0.00886	**0.00005**	< 1e-5	0.00039	0.00025	0.00022
Pct. Violations	0.16140	**0.01166**	< 1e-5	0.05841	0.05166	0.03723
Training Time	10.4124	15.6683	10.9543	202.905	129.783	107.416

Table 3.1: Results for the `synthetic` dataset.

TFL a lattice model implemented via `Tensorflow Lattice`'s Canned Estimator.

The experimental procedure is the same as before, with the training sets obtained by randomly sampling – with stratification, in case of classification tasks – 20% of the data points for the non-synthetic datasets[5]. Still, since this time we are not interested in the whole training history but rather on the goodness of the final solution only, we collect the results of each fold and compute the average value for all the tested models. The best scores, respectively to each metric, are highlighted in the conclusive tables[6].

3.3.1 Regression

Regression datasets share the same loss function and metric, i.e., Mean Squared Error (*MSE*) and R^2 Score (R^2), respectively. All the neural networks – i.e., the MLP model, the SBR model, and the Moving Target's learners – are trained with *MSE* loss function and no output activation. Additionally, each Moving Target instance adopts the same loss during the master step as well, and it is run for a total of 30 iterations.

SYNTHETIC DATASET

A description of the task has been already provided in section 3.2. Among the most notable features of this dataset, one is that it is not built upon a stochastic process, therefore the enforced

[5]The only exception regards the `puzzle` dataset, which already comprises an explicit test set.

[6]When considering violations, lattice models are ignored as they will certainly bring the best results since they are conceived to respect the semantics of the shape constraint.

Partial Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test R^2	0.54249	0.54036	0.47919	0.50471	0.52880	**0.54593**
Test MSE	6855.03	6886.92	7803.53	7421.12	7060.11	**6803.44**
Val R^2	0.41195	0.44419	0.48128	0.44589	**0.49590**	0.45784
Val MSE	3947.17	3715.05	3682.72	3776.49	**3583.07**	3667.49
Train R^2	**0.66980**	0.65957	0.57456	0.58746	0.61441	0.63318
Train MSE	**2473.87**	2561.81	3233.21	3140.36	2923.50	2790.91
Avg. Violation	9.55039	1.13464	$<$ 1e-5	**0.10926**	0.28870	0.34289
Pct. Violations	0.19564	0.07088	$<$ 1e-5	**0.03970**	0.04810	0.04617
Training Time	3.75024	15.3248	22.2058	303.348	323.664	400.130

Full Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test R^2	0.52864	0.59019	0.52015	0.59761	**0.63541**	0.58458
Test MSE	7062.50	6140.40	7189.84	6029.15	**5462.87**	6224.40
Val R^2	0.55850	0.53161	0.52243	0.54714	**0.61920**	0.59452
Val MSE	3068.34	3264.89	3106.25	3135.00	**2441.43**	2600.37
Train R^2	0.82339	0.76653	0.74308	0.80898	0.95622	**0.96101**
Train MSE	1358.02	1725.53	1925.14	1432.32	338.121	**302.327**
Avg. Violation	23.8483	0.41846	$<$ 1e-5	**0.33665**	0.56701	0.70334
Pct. Violations	0.43341	**0.03304**	$<$ 1e-5	0.10420	0.11593	0.10807
Training Time	2.43442	15.3112	36.2656	610.614	483.499	528.429

Table 3.2: Results for the `puzzles` dataset.

constraints directly result from each pair of samples sample having certain input values rather than on the aggregation of multiple samples. Table 3.1 shows the outcomes of the tests. SBR clearly outperforms all the other methods in this case; nonetheless, Moving Targets is the one that mostly approaches its efficiency both in terms of loss, accuracy, and constraint satisfaction, beating all the other rival techniques.

PUZZLES DATASET

This task consists in predicting the sales of jigsaw puzzles using 8 input features obtained from an aggregation of Amazon reviews. For the *Partial Features* variant, monotonic attributes only are retained, i.e., *word count* (decreasing), *star rating* (increasing), and *num reviews* (increasing). Still, in both scenarios we use the entire dataset, which is composed of 522 samples.

As exposed in table 3.2, differently from the previous task, this time Moving Targets reveals itself as the most effective technique in terms of accuracy and constraint satisfaction, both in the full and in the partial features scenario.

CARS DATASET

Similarly to the previous task, this one is aimed at predicting the (monthly) sales of an item, i.e., cars. The dataset is really small, with 157 samples only. In the *Full Features* variant, 11 input attributes – both numeric and categorical – are included; conversely, in the *Partial Features* variant, we keep a unique attribute, namely the price of the car. Since the latter case is a univariate task, the *ceteris paribus* condition is always satisfied, and the monotonicities can be imposed on *all* the pairs of samples without any need for data augmentation. Thus, along with the usual $3 + 3$ methods, four other ones are tested, namely **SBR (NA)**, **MT 0.01 (NA)**, **MT 0.1 (NA)**, and **MT 1.0 (NA)**, where *(NA)* stands for ”Not Augmented”. As well as in the puzzles dataset, Moving Targets is able to outperform all the other techniques as displayed in table 3.3. Also, regarding the univariate case, it is interesting to notice how the effect of augmented data is still beneficial when it comes to Moving Targets, while Semantic-based Regularization works better by enforcing the constraints on original data only.

3.3.2 Classification

While sharing the same loss function, i.e., Binary Crossentropy (*BCE*), classification datasets differ in the chosen metric – Area under the Receiver Operating Characteristic curve (AUC) for the restaurant dataset, and Accuracy score (*Acc.*) for the other two. Both MLP and SBR models, as well as the Moving Target’s learners, are trained with `binary crossentropy` loss and sigmoid output activation, while the master step of Moving Targets adopts `mean squared error`, as suggested in subsection 3.2.3. Also, in order to lower the training times, Moving Target instances are run for 10 iterations only.

RESTAURANTS DATASET

As previously asserted in section 3.2, the purpose of this dataset is to predict the click-through rate of a restaurant given one categorical and two numerical features. The outcomes of the benckmarking procedure are shown in table 3.4, from which it results that Moving Targets is able to slightly improve the test accuracy with respect to MLP and SBR – even though TFL

Partial Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test R^2	0.03285	0.02410	0.02997	**0.04158**	0.00613	0.01498
Test MSE	9520.57	9606.73	9548.95	**9434.62**	9783.65	9696.49
Val R^2	0.05160	0.02468	0.07440	0.09132	-0.00092	-0.02714
Val MSE	3103.01	3178.13	3030.94	3002.53	3337.05	3304.13
Train R^2	0.15178	0.13715	**0.17216**	0.11478	0.03281	0.09421
Train MSE	2837.88	2885.56	**2769.66**	2958.61	3230.31	3030.77
Avg. Violation	0.04981	0.04479	$< 1e-5$	**0.00022**	0.18159	0.01446
Pct. Violations	0.01002	0.00861	$< 1e-5$	**0.00072**	0.16548	0.07819
Training Time	3.20636	4.96835	8.80360	80.9019	153.855	159.115
		SBR (NA)		**MT 0.01 (NA)**	**MT 0.1 (NA)**	**MT 1.0 (NA)**
Test R^2		0.03503		0.02325	0.03864	0.04004
Test MSE		9499.16		9615.12	9463.56	9449.82
Val R^2		0.05850		-0.00023	**0.09186**	0.06020
Val MSE		3079.79		3290.48	**2974.85**	3073.19
Train R^2		0.15040		0.06873	0.12554	0.13779
Train MSE		2842.36		3117.54	2922.56	2884.13
Avg. Violation		0.02409		0.00038	0.04581	0.00085
Pct. Violations		0.00632		0.00136	0.09347	0.00118
Training Time		5.18381		94.5574	91.3459	92.7209
Full Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test R^2	**0.17219**	-0.03171	-1.13504	0.12015	0.05831	0.11057
Test MSE	**3373.35**	4204.25	8700.33	3585.42	3837.40	3624.44
Val R^2	-0.60541	-0.72179	-1.20390	-0.60064	**-0.43101**	-0.48723
Val MSE	5374.08	5946.53	7160.70	5438.21	**5066.78**	5207.88
Train R^2	0.59631	0.53362	0.75481	0.61858	0.92857	**0.96307**
Train MSE	1962.86	2014.71	1169.80	1867.51	349.681	**180.919**
Avg. Violation	8.17774	5.30826	$< 1e-5$	0.56351	0.99641	**0.56137**
Pct. Violations	0.38304	0.19488	$< 1e-5$	0.08586	0.13480	**0.07428**
Training Time	2.10312	7.99095	18.5090	77.0019	85.7401	109.782

Table 3.3: Results for the `cars` dataset.

	MLP	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test AUC	0.78379	0.78650	**0.80254**	0.79063	0.78795	0.78878
Test BCE	0.67221	0.56502	**0.52752**	0.83721	0.92973	0.96632
Val AUC	0.78543	0.79579	**0.80031**	0.78617	0.78258	0.78515
Val BCE	0.54956	0.51769	**0.51007**	0.62986	0.66554	0.67145
Train AUC	0.86740	0.82948	0.83646	**0.86937**	0.86048	0.86094
Train BCE	0.42060	0.47132	0.45786	**0.41238**	0.43159	0.42846
Avg. Violation	0.00035	**0.00001**	< 1e-5	0.00027	0.00024	0.00077
Pct. Violations	0.00998	**0.00125**	< 1e-5	0.01671	0.01362	0.01286
Training Time	4.35492	8.32683	26.5865	91.8858	114.754	142.365

Table 3.4: Results for the `restaurants` dataset.

scores better –, and even to achieve the best scores on the training sets. The major shortcoming regards the level of constraint satisfaction: indeed, Moving Targets could not reduce so much the violations if compared to the unconstrained model, even though the original degree of violation was already small from the beginning.

LAW DATASET

The task is based upon the *Law School Admission Dataset* [36] and it is aimed at predicting whether or not a student will pass the bar exam based on a series of demographic and academic features. In particular, as described in [33], the adoption of monotonicity constraints is justified by the necessity to avoid unfair penalization towards good attributes, e.g., the Law School Admission Test (LSAT) score and the Undergraduate Grade Point Average (UGPA), in this case. The dataset has a rather considerable size, at least with respect to previous ones, since it is made of 27478 records and 9 input features. Due to its high cardinality, no results could be obtained from the lattice model within a reasonable time frame for the *Full Features* variant; instead, in the *Partial Features* one, where the two monotonic attributes only are employed, the TFL model was correctly trained up to convergence. Results shown in table 3.5 exhibit a slight, though significant, improvement regarding all the accuracy scores for the Moving Targets method in the partial features scenario, with a similar trend for the level of constraint violations, which is however beaten by the semantic-based regularization approach. On the contrary, when it comes to the full features alternative, SBR definitely outperforms Moving Targets, in particular if we consider the computational overhead that is required by our method.

Partial Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test Acc.	0.94977	0.94964	0.94951	0.94990	**0.95016**	0.94981
Test BCE	**0.17008**	0.17053	0.17081	0.17197	0.17352	0.17642
Val Acc.	0.94864	0.94886	0.94881	0.94897	**0.94903**	0.94705
Val BCE	0.17274	**0.17244**	0.17265	0.17338	0.17459	0.17791
Train Acc.	0.94904	0.94922	0.94897	**0.94938**	0.94931	0.94731
Train BCE	0.17149	**0.17095**	0.17205	0.17252	0.17357	0.17638
Avg. Violation	0.00034	**< 1e-5**	< 1e-5	0.00002	0.00004	0.00004
Pct. Violations	0.04433	**0.00073**	< 1e-5	0.01565	0.03270	0.01791
Training Time	25.7467	113.713	707.052	1130.07	1084.39	1323.96
Full Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test Acc.	0.94886	0.94915	/	**0.94963**	0.94546	0.93301
Test BCE	0.15292	**0.15223**	/	0.20907	0.20822	0.25441
Val Acc.	0.95086	**0.95152**	/	0.95056	0.94733	0.93458
Val BCE	0.14533	**0.14450**	/	0.20195	0.18203	0.22184
Train Acc.	**0.95333**	0.95317	/	0.95013	0.94865	0.94833
Train BCE	0.13676	**0.13550**	/	0.18582	0.16069	0.14242
Avg. Violation	0.00024	**< 1e-5**	/	0.00004	0.00017	0.00146
Pct. Violations	0.02499	**0.00108**	/	0.05254	0.03300	0.05152
Training Time	30.1492	79.2779	/	6884.64	8841.56	6882.53

Table 3.5: Results for the `law` dataset.

Nevertheless, it should be noticed that the scores are similar when considering the two alternative settings; thus, if the additional features are not required due to the presence of further constraints, it may be reasonable to discard them.

DEFAULT DATASET

This final task is similar to the previous one. It leverages the *Default of Credit Card Clients* benchmark dataset from the UCI repository [37, 38], which contains samples collected from 30000 Taiwanese credit card users and a binary label representing whether or not that user defaulted on a payment, plus 23 additional input features regarding marital status, gender, education, and how long a user is behind on payment of their existing bills [33] – monotonicity constraints are imposed in these last 6 numerical features in order not to penalize users who pay

Partial Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test Acc.	**0.81705**	**0.81705**	**0.81705**	**0.81705**	0.81357	**0.81705**
Test BCE	**0.45322**	0.45477	0.45633	0.45584	0.46468	0.47561
Val Acc.	**0.82091**	**0.82091**	**0.82091**	**0.82091**	0.81754	**0.82091**
Val BCE	**0.45155**	0.45290	0.45476	0.45407	0.46455	0.47607
Train Acc.	**0.82091**	**0.82091**	**0.82091**	**0.82091**	0.81716	**0.82091**
Train BCE	**0.45126**	0.45241	0.45464	0.45424	0.46454	0.47589
Avg. Violation	0.00791	**< 1e-5**	< 1e-5	0.00008	0.00002	0.00024
Pct. Violations	0.22277	**0.00448**	< 1e-5	0.08123	0.01391	0.06800
Training Time	49.7223	183.306	1459.72	937.307	1120.18	1303.76
Full Features	**MLP**	**SBR**	**TFL**	**MT 0.01**	**MT 0.1**	**MT 1.0**
Test Acc.	0.81880	**0.81930**	/	0.81043	0.80773	0.77203
Test BCE	**0.43888**	0.43890	/	0.48947	0.49504	0.63390
Val Acc.	0.81762	**0.81808**	/	0.81067	0.80783	0.77542
Val BCE	**0.43553**	0.43563	/	0.48246	0.48997	0.62929
Train Acc.	0.82342	0.82172	/	0.83378	0.85911	**0.91175**
Train BCE	0.42256	0.42585	/	0.40701	0.32839	**0.21999**
Avg. Violation	0.00378	**0.00001**	/	0.00132	0.00121	0.00274
Pct. Violations	0.12268	**0.00537**	/	0.05940	0.04097	0.05635
Training Time	33.2046	199.056	/	10544.2	13033.4	17722.2

Table 3.6: Results for the `default` dataset.

more in advance, all else being equal. For the *Partial Features* scenario, marital status and payment related to the last month only are considered. Results (table 3.6) show how the accuracies are practically the same independently from the model; still, the constraints are better enforced with SBR. Instead, regarding the *Full Features* scenario, Moving Target is hardly applicable at all not only due to size of the master problem, but most importantly because of the slowness of retraining the learner 10 times on a dataset with such a high dimensionality and even higher cardinality. In fact, the number of samples goes up to $(n_a \cdot \mid F_C \mid +1) \cdot \mid D^S \mid = (3 \cdot 6 + 1) \cdot \mid D^S \mid$, i.e., 19 times the cardinality of the dataset used to train the unconstrained neural network.

Chapter 4

Conclusions

In this work, we extended Moving Targets, a state-of-the-art technique for injecting constraints in supervised learning [2]. We focused our contribution on three main steps. Firstly, we implemented and, eventually, empirically evaluated the role of two additional features that were proposed in the conclusive chapter of the original paper presenting the method. We demonstrated that starting the process by projecting the original targets into the feasible region leads to the same results as starting with a pretraining step, at least if enough iterations are run. Secondly, we developed new methodologies to deal with semi-supervised learning scenarios as well as supervised ones, while maintaining the core of the algorithm intact. We finally assessed the quality of our contribution via hyperparameter investigation and benchmarking against other techniques from the literature in the third step. We were able to prove that Moving Targets is rather a robust method since it is not particularly influenced by its hyperparameters, with the sole exception of α. Moreover, when employed to solve semi-supervised tasks involving monotonicity shape constraints, the algorithm could outperform other common techniques such as semantic-based regularization and lattice models, particularly when dealing with regression problems.

4.1 Future Works

While being proved effective in certain scenarios, Moving Targets exhibited some defects in other ones, especially regarding classification tasks. This is most probably due to the drawbacks natively brought by monotonicity shape constraints, as exposed in section 1.3, which somehow forces the learner and the master step to rely on two different formulations of the problem. Nonetheless, even though the algorithm could not outperform the opposing approaches, it should be taken in mind that Moving Targets comes with other advantages, i.e., an inner support for relational and non-differentiable constraints, as well as for multiple constraints imposed on the same set of data points. Indeed, it would be interesting to extend the evaluation of the `law` and `default` dataset by including fairness constraints on demographic attributes along with the monotonic ones, and to study the algorithm behaviour respectively to other methods.

Another crucial issue of this approach, which was already mentioned in the original paper, is that of scalability. When the dimensionality and the cardinality of the dataset increase, both the master and the learner steps become computationally unsustainable. For the latter, one solution may be to adopt GPUs instead of CPUs, even though it is widely known in the Deep Learning research field that the benefits of general-purpose GPU computing can be appreciated on large neural models only. The other solution, at least when it comes to semi-supervised tasks, might be to rely on more intelligent sampling techniques [39, 40]. This, combined with an iterative augmentation procedure that recomputes the unlabelled data points at each iteration in order to cover decisive areas only, should allow to substantially limit the number of samples and, accordingly, to reduce both the training and the optimization times.

Bibliography

[1] High-Level Expert Group on AI. *Ethics guidelines for trustworthy AI.* eng. Report. Brussels: European Commission, Apr. 2019. URL: `https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai`.

[2] Fabrizio Detassis, Michele Lombardi, and Michela Milano. "Teaching the Old Dog New Tricks: Supervised Learning with Constraints". In: *arXiv preprint arXiv:2002.10766* (2020).

[3] Faisal Kamiran and Toon Calders. "Data preprocessing techniques for classification without discrimination". In: *Knowledge and Information Systems* 33.1 (Dec. 2011), pp. 1–33. DOI: `10.1007/s10115-011-0463-8`. URL: `https://doi.org/10.1007/s10115-011-0463-8`.

[4] Mohammad Peikari et al. "A Cluster-then-label Semi-supervised Learning Approach for Pathology Image Classification". In: *Scientific Reports* 8.1 (May 2018). DOI: `10.1038/s41598-018-24876-0`. URL: `https://doi.org/10.1038/s41598-018-24876-0`.

[5] C. Rosenberg, M. Hebert, and H. Schneiderman. "Semi-Supervised Self-Training of Object Detection Models". In: *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05) - Volume 1*. IEEE, Jan. 2005. DOI: `10.1109/acvmot.2005.107`. URL: `https://doi.org/10.1109/acvmot.2005.107`.

[6] Laura von Rueden et al. "Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems". In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. DOI: `10.1109/tkde.2021.3079836`. URL: `https://doi.org/10.1109/tkde.2021.3079836`.

[7] Matthew Richardson and Pedro Domingos. "Markov logic networks". In: *Machine learning* 62.1-2 (2006), pp. 107–136.

[8] Jue Wang and Pedro M Domingos. “Hybrid Markov Logic Networks.” In: *AAAI*. Vol. 8. 2008, pp. 1106–1111.

[9] Robin Manhaeve et al. “Deepproblog: Neural probabilistic logic programming”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 3749–3759.

[10] Tim Rocktäschel and Sebastian Riedel. “End-to-end differentiable proving”. In: *arXiv preprint arXiv:1705.11040* (2017).

[11] Moritz Hardt, Eric Price, and Nathan Srebro. “Equality of opportunity in supervised learning”. In: *arXiv preprint arXiv:1610.02413* (2016).

[12] Benjamin Fish, Jeremy Kun, and Ádám D. Lelkes. “A Confidence-Based Approach for Balancing Fairness and Accuracy”. In: *Proceedings of the 2016 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, June 2016. DOI: 10.1137/1.9781611974348.17. URL: https://doi.org/10.1137/1.9781611974348.17.

[13] Binh Thanh Luong, Salvatore Ruggieri, and Franco Turini. “k-NN as an implementation of situation testing for discrimination discovery and prevention”. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’11*. ACM Press, 2011. DOI: 10.1145/2020408.2020488. URL: https://doi.org/10.1145/2020408.2020488.

[14] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. “Semantic-based regularization for learning and inference”. In: *Artificial Intelligence* 244 (Mar. 2017), pp. 143–165. DOI: 10.1016/j.artint.2015.08.011. URL: https://doi.org/10.1016/j.artint.2015.08.011.

[15] Luciano Serafini and Artur d’Avila Garcez. “Logic tensor networks: Deep learning and logical reasoning from data and knowledge”. In: *arXiv preprint arXiv:1606.04422* (2016).

[16] Emile Van Krieken, Erman Acar, and Frank Van Harmelen. “Semi-supervised learning using differentiable reasoning”. In: *arXiv preprint arXiv:1908.04700* (2019).

[17] Ferdinando Fioretto et al. “Lagrangian Duality for Constrained Deep Learning”. In: *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track*. Springer International Publishing, 2021, pp. 118–135. DOI: 10.1007/978-3-030-67670-4_8. URL: https://doi.org/10.1007/978-3-030-67670-4_8.

[18] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. "Learning Optimal and Fair Decision Trees for Non-Discriminative Decision-Making". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 1418–1426. DOI: `10.1609/aaai.v33i01.33011418`. URL: `https://doi.org/10.1609/aaai.v33i01.33011418`.

[19] Cynthia Dwork et al. "Fairness through awareness". In: *Proceedings of the 3rd innovations in theoretical computer science conference*. 2012, pp. 214–226.

[20] Rich Zemel et al. "Learning fair representations". In: *International conference on machine learning*. PMLR. 2013, pp. 325–333.

[21] Toon Calders and Sicco Verwer. "Three naive Bayes approaches for discrimination-free classification". In: *Data Mining and Knowledge Discovery* 21.2 (July 2010), pp. 277–292. DOI: `10.1007/s10618-010-0190-x`. URL: `https://doi.org/10.1007/s10618-010-0190-x`.

[22] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. "Discrimination Aware Decision Tree Learning". In: *2010 IEEE International Conference on Data Mining*. IEEE, Dec. 2010. DOI: `10.1109/icdm.2010.50`. URL: `https://doi.org/10.1109/icdm.2010.50`.

[23] Guosheng Lin et al. "Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016. DOI: `10.1109/cvpr.2016.348`. URL: `https://doi.org/10.1109/cvpr.2016.348`.

[24] Xuezhe Ma and Eduard Hovy. "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2016. DOI: `10.18653/v1/p16-1101`. URL: `https://doi.org/10.18653/v1/p16-1101`.

[25] Andrew Cotter et al. "Optimization with Non-Differentiable Constraints with Applications to Fairness, Recall, Churn, and Other Goals." In: *Journal of Machine Learning Research* 20.172 (2019), pp. 1–59.

[26] Maya Gupta et al. "Monotonic calibrated interpolated look-up tables". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 3790–3836.

[27] Eric E Altendorf, Angelo C Restificar, and Thomas G Dietterich. “Learning from sparse data by exploiting monotonicity constraints”. In: *arXiv preprint arXiv:1207.1364* (2012).

[28] Andrew Cotter et al. “Shape constraints for set functions”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1388–1396.

[29] Maya R Gupta et al. “Diminishing returns shape constraints for interpretability and regularization”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 6835–6845.

[30] Seungil You et al. “Deep lattice networks and partial monotonic functions”. In: *arXiv preprint arXiv:1709.06680* (2017).

[31] Maya Gupta et al. “Multidimensional shape constraints”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3918–3928.

[32] Eric Garcia and Maya Gupta. “Lattice regression”. In: *Advances in Neural Information Processing Systems* 22 (2009), pp. 594–602.

[33] Serena Wang and Maya Gupta. “Deontological ethics by monotonicity shape constraints”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2043–2054.

[34] Yisen Wang et al. “Symmetric Cross Entropy for Robust Learning With Noisy Labels”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019. DOI: `10.1109/iccv.2019.00041`. URL: `https://doi.org/10.1109/iccv.2019.00041`.

[35] Zhilu Zhang and Mert R Sabuncu. “Generalized cross entropy loss for training deep neural networks with noisy labels”. In: *arXiv preprint arXiv:1805.07836* (2018).

[36] Linda F Wightman. *LSAC national longitudinal bar passage study*. Law School Admission Council, 1998.

[37] Moshe Lichman et al. *UCI machine learning repository*. 2013.

[38] I-Cheng Yeh and Che-hui Lien. “The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients”. In: *Expert Systems with Applications* 36.2 (Mar. 2009), pp. 2473–2480. DOI: `10.1016/j.eswa.2007.12.020`. URL: `https://doi.org/10.1016/j.eswa.2007.12.020`.

[39] Carolyn Kim, Ashish Sabharwal, and Stefano Ermon. “Exact sampling with integer linear programs and random perturbations”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[40] Carla P Gomes, Ashish Sabharwal, and Bart Selman. “Near-uniform sampling of combinatorial spaces using XOR constraints”. In: *NIPS*. 2006, pp. 481–488.