Teaching the Old Dog New Tricks: Supervised Learning with Constraints

Fabrizio Detassis[1,2] and **Michele Lombardi**[1] and **Michela Milano**[1]

Abstract. Methods for taking into account external knowledge in Machine Learning models have the potential to address outstanding issues in data-driven AI methods, such as improving safety and fairness, and can simplify training in the presence of scarce data. We propose a simple, but effective, method for injecting constraints at training time in supervised learning, based on decomposition and bi-level optimization: a master step is in charge of enforcing the constraints, while a learner step takes care of training the model. The process leads to approximate constraint satisfaction. The method is applicable to any ML approach for which the concept of label (or target) is well defined (most regression and classification scenarios), and allows to reuse existing training algorithms with no modifications. We require no assumption on the constraints, although their properties affect the shape and complexity of the master problem. Convergence guarantees are hard to provide, but we found that the approach performs well on ML tasks with fairness constraints and on classical datasets with synthetic constraints.

1 Introduction

Techniques to deal with constraints in Machine Learning (ML) have the potential to address outstanding issues in data-driven AI methods. Constraints representing (e.g.) physical laws can be employed to improve generalization; constraints may encode negative patterns (e.g. excluded classes) and relational information (e.g. involving multiple examples); constraints can ensure the satisfaction of desired properties, such as fairness, safety, or lawfulness; they can even be used to extract symbolic information from data.

Such techniques can highly improve the learning process in terms of versatility and control. As a consequence, our work constitutes a step forward in the direction of a more human-oriented AI, as it is general and allows for the realization of learning models that satisfy specific user requirements. This is a fundamental and essential behaviour in contexts where there exist human-specific constraints, which the trained AI model has to acknowledge. The present paper aims at enlarging the pool of tools that can be used to inject an AI model with external knowledge, a field that is rapidly expanding thanks to its many potential practical applications.

To the best of the authors knowledge, the vast majority of approaches for taking into account external knowledge in ML make assumptions that restrict the type of constraints (e.g. differentiability, no relational information), the type of models (e.g. only Decision Trees, only differentiable approaches), and often require modifications in the employed training algorithms (e.g. specialized loss terms).

We propose a decomposition-based method, referred to as *Moving Targets*, to augment supervised learning with constraints. A master step "teaches" constraint satisfaction to a learning algorithm by *iteratively adjusting the sample labels*. The master and learner have no direct knowledge of each other, meaning that: 1) any ML method can be used for the learner, with no modifications; 2) the master can be defined via techniques such as Mathematical or Constraint Programming, to support discrete values or non-differentiable constraints. Our method is also well suited to deal with relational constraints over large populations (e.g. fairness indicators). Moving Targets subsumes the few existing techniques – such as the one by [13] – capable of offering the same degree of versatility.

When constraints conflict with the data, the approach *prioritizes constraint satisfaction over accuracy*. For this reason, it is not well suited to deal with fuzzy information. Moreover, due to our open setting, it is hard to determine convergence properties. Despite this, we found that the approach performs well (compared to state of the art methods) on classification and regression tasks with fairness constraints, and on classification problems with balance constraints.

Due to its combination of simplicity, generality, and the observed empirical performance, Moving Targets can represent a valuable addition to the arsenal of techniques for dealing with constraints in Machine Learning. The paper is organized as follows: in Section 2 we briefly survey related works on the integration of constraints in ML; in Section 3 we present our method and in Section 4 our empirical evaluation. Concluding remarks are in Section 5.

2 Related Works

Here we provide an overview of representative approaches for integrating constraints in ML, and we discuss their differences with our method.

Most approaches in the literature build over just a few key ideas. One of them is *using the constraints to adjust the output of a trained ML model*. This is done in DeepProbLog [20], where Neural Networks with probabilistic output (mostly classifiers) are treated as predicates. [25] presents a Neural Theorem Prover using differentiable predicates and the Prolog backward chaining algorithm. The original Markov Logic Networks [24] rely instead on Markov Fields defined over First Order Logic formulas. As a drawback, with these approaches the constraints have no effect on the model parameters, which complicates the analysis of feature importances. Moreover, dealing with relational constraints (e.g. fairness) requires access at prediction time either to a

[1] DISI, University of Bologna, Italy, email: {fabrizio.detassis2, michele.lombardi2, michela.milano}@unibo.it
[2] Corresponding author

representative population or to its distribution [12, 11].

A second group of approaches operate by *using constraint-based expressions as regularization terms at training time*. In Semantic Based Regularization [7] constraints are expressed as fuzzy logical formulas over differentiable predicates. Logic Tensor Networks [26] focus on Neural Networks and replace the entire loss function with a fuzzy formula. Differentiable Reasoning [27] uses in a similar fashion relational background knowledge to benefit from unlabeled data. In the context of fairness constraints, this approach has been taken in [1, 10, 28, 5, 15]. Methods in this class account for the constraints by adjusting the model parameters, and can therefore be used to analyze feature importance. They can deal with relational constraints without additional examples at prediction time; however, they ideally require *simultaneous* access at training time to all the examples linked by relational constraints (which can be problematic when using mini-batches). They often require properties on the constraints (e.g. differentiability), which may force approximations; they may also be susceptible to numerical issues.

A third idea consists in *enforcing constraint satisfaction in the data via a pre-processing step*. This is proposed in the context of fairness constraints by [13, 14, 18]. The approach enables the use of standard ML methods with no modification, and can deal with relational constraints on large sets of examples. As a main drawback, bias in the model or the training algorithm may prevent getting close to the pre-processed labels.

Multiple ideas can be combined: domain knowledge has been introduced in differentiable Machine Learning (e.g. Deep Networks) by designing their structure, rather than the loss function: examples include Deep Structured Models in [16] and [19]. These approaches can use constraints to support both training and inference.

Though less related to our approach, constraints can be used to *extract symbolic knowledge from data*, for example by allowing the training algorithm to adjusting the regularizer weights. This approach is considered (e.g.) in [17, 21, 6].

Our approach is closely *related to the idea of enforcing constraints by altering the data*, and shares the same advantages (versatility, support for relational constraints and feature importance analysis, no differentiability assumptions). We counter the main drawbacks mentioned above by using an iterative algorithm rather than a single pre-processing step.

[tb]

Table 1. Notable loss functions (m = # examples, c = #classes)

Loss Function	Expression	Label Space
Mean Squared Error	$\frac{1}{m}\|y - y^*\|_2^2$	$\mathbb{R}^m$
Hamming Distance	$\frac{1}{m}\sum_{i=1}^{m} \mathrm{I}[y_i \neq y_j^*]$	$\{1..c\}^m$
Cross Entropy	$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{c} y_{ij}^* \log y_{ij}$	$[0,1]^m$

3 Moving Targets

In this section we present our method, discuss its properties, draw connections with related algorithms and provide some convergence considerations.

Algorithm 1 MOVING TARGETS

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

The Algorithm Our goal is to adjust the parameters of a ML model so as to minimize a loss function with clearly defined labels, under a set of generic constraints. We acknowledge that any constrained learning problem must trade prediction mistakes for a better level of constraint satisfaction, and we attempt to *control this process by carefully selecting which mistakes should be made*. This is similar to [13, 14, 18], but: 1) we consider generic constraints rather than focusing on fairness; and 2) we rely on an iterative process (which alternates "master" and "learner" steps) to improve the results.

Let $L(y, y^*)$ be the loss function, where y is the prediction vector and y^* is the label vector. We make the (non restrictive) assumption that *the loss is a pre-metric* – i.e. $L(y, y^*) \geq 0$ and $L(y, y^*) = 0$ iff $y = y^*$. Examples of how to treat common loss functions can be found in Table 1.

We then want to solve, in an exact or approximate fashion, the following constrained optimization problem:

$$\arg\min_{\theta}\{L(y, y^*) \mid y = f(X;\theta), y \in C\} \tag{1}$$

where f is the ML model and θ its parameter vector. With some abuse of notation we refer as $f(X;\theta)$ to the vector of predictions for the examples in the training set X. Since the model input at training time is known, constraints can be represented as a feasible set C for the sole predictions y.

The problem can be rewritten without loss of generality by introducing a second set B corresponding to the ML model bias. This leads to a formulation *in pure label space*:

$$\arg\min_{y}\{L(y, y^*) \mid y \in B \cap C\} \tag{2}$$

where $B = \{y \mid \exists\theta, y = f(X;\theta)\}$.

The Moving Targets method is described in Algorithm 1, and starts with a learner step w.r.t. the original label vector y^* (pretraining). Each learner step, given a label vector as input, solves approximately or exactly the problem:

$$l(z) = \arg\min_{y}\{L(y, z) \mid y \in B\} \tag{3}$$

Note that this is *a traditional unconstrained learning problem*, since B is just the model/algorithm bias. The result of the first learner step gives an initial vector of predictions y^1.

Next comes a master step to adjust the label vector: this can take two forms, depending on the current predictions. *In case of an infeasibility*, i.e. $y^k \notin C$, we solve:

$$m_\alpha(y) = \arg\min_{z}\left\{L(z, y^*) + \frac{1}{\alpha}L(z, y) \mid z \in C\right\} \tag{4}$$

Intuitively, we try to find a feasible label vector z that is close (in terms of loss function value) to both the original labels y^* and the

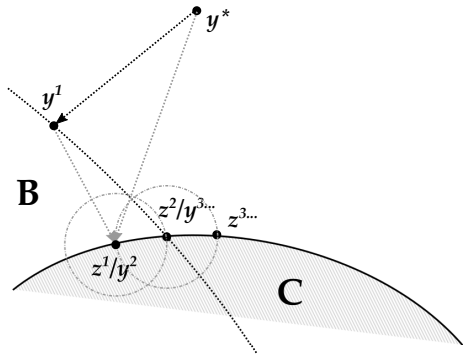

Figure 1. A sample run of our algorithm

current prediction y. A parameter $\alpha \in (0, \infty)$ controls which of the two should be preferred. *If the input vector is feasible* we instead solve:

$$m_\beta(y) = \arg\min_z \{L(z, y^*) \mid L(z, y) \leq \beta, z \in C\} \qquad (5)$$

i.e. we look for a feasible label vector z that is 1) not too far from the current predictions (in the ball defined by $L(z, y) \leq \beta$) and 2) closer (in terms of loss) to the true labels y^*. Here, we are seeking an accuracy improvement.

We then make a learner step trying to reach the adjusted labels; the new predictions will be adjusted at the next iteration and so on. In case of convergence, the predictions y^k and the adjusted labels z^k become stationary (but not necessarily identical). An example run, for a Mean Squared Error loss and convex constraints and bias, is in Figure 1.

Properties The learner is not directly affected by the constraints, thus enabling the use of arbitrary ML approaches. The master problems do not depend on the ML model, often leading to clean structures that are easier to solve. Since we make no explicit use of mini-batches, we can deal well with relational constraints on large groups (e.g. fairness). The master step can be addressed via any suitable solver, so that discrete variables and non-differentiable constraints can be tackled via (e.g.) Mathematical Programming, Constraint Programming, or SAT Modulo Theories. Depending on the constraints, the loss functions, and the label space (e.g. numeric vs discrete) the master problems may be NP-hard. Even in this case, their clean structure may allow for exact solutions for datasets of practical size. Moreover, for separable loss functions (e.g. all those from Table 1), the master problems can be defined over only the constrained examples, with a possibly significant size reduction. If scalability is still a concern, the master step can be solved in an approximate fashion: this may lead to a lower accuracy and a higher level of constraint violation; however, such issues are partly inevitable, due to algorithm/model bias, and since constraint satisfaction on the training data does not necessarily transfer to unseen examples.

Analysis and Convergence Due to its open nature and minimal assumptions, establishing the convergence of our method is hard. Here we provide some considerations and connect the approach to existing algorithms. We will make the simplifying assumption that the learner problem from Equation (3) can be solved exactly. This holds for a few cases (e.g. convex ML models trained to close optimality), but in general the assumption will not be strictly satisfied.

We start by observing that Equation (2) is simply the Best Approximation Problem (BAP), which involves finding a point in the intersection of two sets (say B and C) that is as close as possible to a reference point (say y^*). For convex sets in Hilbert spaces, the BAP can be solved optimally via Douglas-Rachford splits or the method from [2], both relying on projection operators. Since our learner is essentially a projection operator on B, it would seem sensible to apply these methods in our setting. Unfortunately: 1) *we cannot reliably assume convexity*; and 2) in a discrete space, the Douglas-Rachford splits *may lead to "label" vectors that are meaningless for the learner*.

We therefore chose a design that is less elegant, but also less sensitive to which assumptions are valid. In particular: 1) in the m_α steps, we use a modification of a suboptimal BAP algorithm to find a *feasible* prediction vector; then 2) in m_β we apply a modified proximal operator to improve its distance (in terms of loss) w.r.t. the original labels y^*. The basis for our m_α step is the Alternating Projections (AP) method, discussed e.g. in [4]. The AP simply alternates projection operators on the two sets, which never generates vectors outside of the label space.

Indeed, for $\alpha \to 0$ our m_α step becomes a projection of the predictions y^k on C. *With this setup we recover the AP behavior, and its convergence to a feasible point for convex sets*. For $\alpha \to \infty$ we obtain, essentially, the pre-processing method from [13]: m_α becomes a projection of the true labels y^* on C, and subsequent iterations have no further effect; convergence to a feasible point is achieved only if the pre-processed labels are in B, which may not be the case (e.g. a quadratic distribution for a linear model). For reasonable α values, our m_α step balances the distance (loss) from both the predictions y and the targets y^*. Convergence in this case is an open question, but especially in a non-convex or discrete setting (where multiple projections may exist) this modification helps escaping local minima and accelerate progress.

When a feasible prediction vector is obtained, our method switches to the m_β step; we then search for a point in C that is closer to the true labels, but also not too far from the predictions. This is related to the Proximal Gradient method, discussed e.g. in [22], but we limit the distance via a constraint rather than a squared norm, and we search in a ball rather than on a line. As in the proximal gradient, a too large search radius prevents convergence: for $\beta \to \infty$ the m_β step always returns the same adjusted labels, corresponding to the projection of y^* on C set.

4 Empirical Evaluation

Our experimentation is designed around a few main questions: 1) How does the Moving Targets method work on a variety of constraints, tasks, and datasets? 2) What is the effect of the α, β parameters? 3) How does the approach scale? 4) How different is the behavior with different ML models? 5) How does the method compare with alternative approaches? We proceed to describe our setup and the experiments we performed to address such questions.

Tasks and Constraints We experiment on three case studies, covering two types of ML tasks and two types of constraints. First, we consider *a classification problem augmented with a balance constraint*, which forces the distribution over the classes to be approximately uniform. The loss function is given by the Hamming distance and the label space is $\{1..c\}^m$. The $m_\alpha(y)$ problem is defined as a Mixed Integer Linear Program with binary variables z_{ij} such that

$z_{ij} = 1$ iff the adjusted class for the i-th example is j. Formally:

$$\min \frac{1}{m}\sum_{i=1}^{m}(1 - z_{i,y_i^*}) + \frac{1}{\alpha m}\sum_{i=1}^{m}(1 - z_{i,y_i}) \tag{6}$$

$$\text{s.t.} \sum_{j=1}^{c} z_{ij} = 1 \qquad \forall i = 1..m \tag{7}$$

$$\sum_{i=1}^{m} y_{ij} \leq \left\lceil \frac{(1+\xi)m}{c} \right\rceil \qquad \forall j = 1..c \tag{8}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i = 1..m, j = 1..c \tag{9}$$

The summations in Equation (6) encode the Hamming distance w.r.t. the true labels y^* and the predictions y. Equation (7) prevents assigning two classes to the same example. Equation (8) requires an equal count for each class, with tolerance defined by ξ ($\xi = 0.05$ in all our experiments); the balance constraint is stated in exact form, thanks to the discrete labels. The m_α formulation generalizes the knapsack problem and is hence NP-hard; since all examples appear in Equation (8), no problem size reduction is possible. The m_β problem can be derived from m_α by changing the objective function and by adding the ball constraint as in Equation (5).

Our second use case is *a classification problem with realistic fairness constraints*, based on the DIDI indicator from [1]:

$$DIDI^c(X, y) = \sum_{k \in K}\sum_{v \in D_k}\sum_{j=1}^{c} d_{kvj} \tag{10}$$

$$d_{k,v,j} = \left| \frac{1}{m}\sum_{i=1}^{m} \mathrm{I}[y_i = j] - \frac{1}{|X_{k,v}|}\sum_{i \in X_{k,v}} \mathrm{I}[y_i = j] \right|$$

where K contains the indices of "protected features" (e.g. ethnicity, gender, etc.). D_k is the set of possible values for the k-th feature, and $X_{k,v}$ is the set of examples having value v for the k-th feature. The $m_\alpha(y)$ problem can be defined via the following Mathematical Program:

$$\min \frac{1}{m}\sum_{i=1}^{m}(1 - z_{i,y_i^*}) + \frac{1}{\alpha m}\sum_{i=1}^{m}(1 - z_{i,y_i}) \tag{11}$$

s.t. Equation (7)

$$\sum_{k \in K}\sum_{v \in D_k}\sum_{j=1}^{c} d_{kvj} \leq \epsilon \qquad \forall j = 1..c \tag{12}$$

$$d_{kvj} \geq \sum_{i=1}^{m}\frac{y_{ij}}{m} - \sum_{i \in X_{k,v}}\frac{y_{ij}}{|X_{k,v}|} \tag{13}$$

$$d_{kvj} \geq -\sum_{i=1}^{m}\frac{y_{ij}}{m} + \sum_{i \in X_{k,v}}\frac{y_{ij}}{|X_{k,v}|} \tag{14}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i = 1..m, j = 1..c \tag{15}$$

where Equation (12) is the constraint on the DIDI value and Equation (13)-(14) linearize the absolute values in its definition. The DIDI scales with the number of examples and has an intrinsic value due to the discrimination in the data. Therefore, we compute $DIDI_{tr}$ for the training set, then in our experiments we have $\epsilon = 0.2DIDI_{tr}$. This is again an NP-hard problem defined over all training examples. The m_β formulation can be derived as in the previous case.

Our third case study is *a regression problem with fairness constraints*, based on a specialized DIDI version from [1]:

$$DIDI^r(X, y) = \sum_{k \in K}\sum_{v \in D_k} d_{kv} \tag{16}$$

$$d_{k,v,j} = \left| \frac{1}{m}\sum_{i=1}^{m} y_i - \frac{1}{|X_{k,v}|}\sum_{i \in X_{k,v}} y \right| \tag{17}$$

In this case, we use the Mean Squared Error (MSE) as a loss function, and the label space is $\mathbb{R}^m$. The m_α problem can be defined via the following Mathematical Program:

$$\min \frac{1}{m}\sum_{i=1}^{m}(y_i^* - z_i)^2 + \frac{1}{\alpha m}\sum_{i=1}^{m}(z_i - y_i)^2 \tag{18}$$

$$\text{s.t.} \sum_{k \in K}\sum_{v \in D_k} d_{kv} \leq \epsilon \qquad \forall j = 1..c \tag{19}$$

$$d_{kv} \geq \sum_{i=1}^{m}\frac{y_i}{m} - \sum_{i \in X_{k,v}}\frac{y_i}{|X_{k,v}|} \tag{20}$$

$$d_{kv} \geq -\sum_{i=1}^{m}\frac{y_i}{m} + \sum_{i \in X_{k,v}}\frac{y_i}{|X_{k,v}|} \tag{21}$$

$$z_i \in \mathbb{R} \qquad \forall i = 1..m \tag{22}$$

This is a linearly constrained, convex, Quadratic Programming problem that can be solved (unlike our classification examples) in polynomial time. The m_β problem can be derived as in the previous cases: while still convex, m_β is in this case a quadratically constrained problem.

Datasets, Preparation, and General Setup We test our method on seven datasets from the UCI Machine Learning repository [9], namely *iris* (150 examples), *redwine* (1,599), *crime* (2,215), *whitewine* (4,898), *adult* (32,561), *shuttle* (43,500), and *dota2* (92,650). We use *adult* for the classification/fairness case study, *crime* for regression/fairness, and the remaining datasets for the classification/balance case study.

For each experiment, we perform a 5-fold cross validation (with a fixed seed for random reshuffling) to account for noise due to sampling and in the training process. Hence, the training set for each fold will include 80% of the data. All our experiments are run on an Intel Core i7 laptop with 16GB RAM, and we use Cplex 12.8 to solve the master problems. Our code and datasets are publicly available[3].

All the datasets for the classification/balance case study are prepared by standardizing all input features (on the training folds) to have zero mean and unit variance. The *iris* and *dota2* datasets are very balanced (the constraint is easily satisfied), while the remaining datasets are quite unbalanced. In the *adult* (also known as "Census Income") dataset the target is "income" and the protected attribute is "race". We remove the features "education" (duplicated) and "native country" and use a one-hot encoding on all categorical features. Features are normalized between 0 and 1. Our *crime* dataset is the "Communities and Crime Unnormalized" table. The target is "violentPerPop" and the protected feature is "race". We remove features that are empty almost everywhere and features trivially related to the target ("murders", "robberies", etc.). Features are normalized between 0 and 1 and we select the top 15 ones according to the `SelectKBest` method of scikit-learn (excluding "race"). The protected feature is then reintroduced.

[3] git repository publicly available

Table 2. Effect of parameters α and β on different datasets.

NN (α, β)		Ptr	$\alpha = 1$ $\beta = .01$	$\alpha = 1$ $\beta = .05$	$\alpha = 1$ $\beta = .1$	$\alpha = .1$ $\beta = .01$	$\alpha = 0^+$ $\beta = 0.1$	Ideal case
Iris	S	$.970 \pm .002$	$.99 \pm .01$	$\mathbf{.997} \pm .004$	$\mathbf{.997} \pm .004$	$.99 \pm .02$	0.995 ± 0.008	$.9968 \pm .0004$
	C	$.016 \pm .001$	$.014^* \pm .004$	$.013^* \pm .004$	$.013^* \pm .004$	$.015^* \pm .005$	$.013^* \pm .004$	$.013 \pm .004$
Redwine	S	$.709 \pm .005$	$.508 \pm .006$	$\mathbf{.511} \pm .009$	$.506 \pm .006$	$.484 \pm .007$	$.50 \pm .01$	$.525 \pm .002$
	C	$.200 \pm .001$	$.019^* \pm .001$	$.0186^* \pm .0005$	$.0184^* \pm .0008$	$.0188^* \pm .0004$	$.019^* \pm .001$	$.019 \pm 0$
Whitewine	S	$.644 \pm .002$	$\mathbf{.446} \pm .006$	$.437 \pm .009$	$.439 \pm .009$	$.40 \pm .02$	$.401 \pm .009$	$.524 \pm .002$
	C	$.189 \pm .003$	$.015^* \pm .002$	$.013^* \pm .004$	$.015^* \pm .003$	$.014^* \pm .004$	$.014^* \pm .004$	$.015 \pm .002$
Shuttle	S	$.999 \pm 0$	$\mathbf{.39} \pm .04$	$.37 \pm .01$	$.375 \pm .007$	$.37 \pm .03$	$.37 \pm .03$	$.3608 \pm .0008$
	C	$.267 \pm 0$	$.045^* \pm .03$	$.029 \pm .003$	$.028 \pm .006$	$0.05 \pm .03$	$.04 \pm .04$	$.017 \pm 0$
Dota2	S	$.686 \pm .002$	$\mathbf{.666} \pm .007$	$.661 \pm .002$	$.66 \pm .01$	$.672 \pm .004$	$.656 \pm .006$	$.9984 \pm .0009$
	C	$.070 \pm .008$	$.04^* \pm .03$	$.04^* \pm .03$	$.09 \pm .06$	$.023 \pm .006$	$.07 \pm .03$	$.025 \pm 0$
Adult	S	$.867 \pm 0.001$	$.818 \pm .005$	$\mathbf{.86} \pm .02$	$.841 \pm .006$	$.852 \pm .004$	$.84 \pm .02$	$0.992 \pm .0005$
	C	$1 \pm .03$	$.18^* \pm .04$	$.16^* \pm .02$	$.22^* \pm .07$	$.22^* \pm .03$	$.22^* \pm .04$	$0.200 \pm .0005$
Crime	S	$.56 \pm .02$	$\mathbf{.49} \pm .01$	$.46 \pm .04$	$.48 \pm .03$	$.45 \pm .05$	$.46 \pm .06$	$.910 \pm .007$
	C	$.97 \pm .03$	$.22^* \pm .07$	$.16^* \pm .08$	$.2^* \pm .1$	$.19^* \pm .02$	$.21^* \pm .04$	$.2 \pm 0$

Parameter tuning We know that extreme choices for α and β can dramatically alter the method behavior, but not what effect can be expected for more reasonable values. With this aim, we perform an investigation by running the algorithm for 15 iterations (used in all experiments), with different α and β values. As a ML model, we use a fully-connected, feed-forward Neural Network (NN) with two hidden layers with 32-Rectifier Linear Units. The last layer has either a SoftMax activation (for classification) or Linear (for regression). The loss function is respectively the categorical cross-entropy or the MSE. The network is trained with 100 epochs of RMSProp in Keras/Tensorflow 2.0 (default parameters and batch size 64).

The results are in Table 2. We report a score (row *S*, higher is better) and a level of constraint violation (row *C*, lower is better). The *S* score is the accuracy for classification and the R2 coefficient for regression. For the balance constraint, the *C* score is the standard deviation of the class frequencies; in the fairness case studies, we use the ratio between the DIDI of the predictions and that of the training data. Each cell reports mean and standard deviation for the 5 runs. Near feasible values are marked with a $*$; accuracy comparisons are fair only for similar constraint violation scores. We consider a solution as near feasible when the associated constraint score lies within one standard deviation from the imposed score.

All columns labeled with α and β values refer to our method with the corresponding parameters. We explore different values of β (for a fixed $\alpha = 1$), corresponding to different ball radii in the m_β step; we also explore different values of α, corresponding to a behavior of the m_α step progressively closer to that of the Alternating Projections method. The *ideal case* column refers to a simple projection of the true labels y^* on the feasible space C: this corresponds (on the training data) to the best possible accuracy that can be achieved if the constraints are fully satisfied. The *ptr* column reports the results of the pretraining step, as defined in algorithm 1.

Results are to be read as follows: the *ideal case* constitutes the upper bound to the performance of a constrained learner; it exactly matches the constraint threshold while minimizing the loss function. On the other hand, the *ptr* represents the constraint-agnostic behavior: it retains good performances while not necessarily satisfying the constraints. Our method lies in between the two extreme cases.

The Moving Targets algorithm can *significantly improve the satisfaction of non-trivial constraints*: this is evident for the (very) unbalanced datasets *redwine*, *whitewine*, and *shuttle* and all fairness use cases, for which feasible (or close) results are almost always obtained. Satisfying very tight constraints (e.g. in the unbalanced dataset) generally comes at a significant cost in terms of accuracy. *When the constraints are less demanding, however, we often observe accuracy improvements* w.r.t. the pretraining results (even substantial ones for *adult* and *crime*): this is not simply a side effect of the larger number of training epochs, since we reset the NN weights at each iteration. Rather, this seems a positive side-effect of using an iterative method to guide training (which may simplify escaping from local optima): further investigation is needed to characterize this phenomenon. Finally, *reasonable parameter choices have only a mild effect on the algorithm behavior*, thus simplifying its configuration. Empirically, $\alpha = 1, \beta = 0.1$ seems to works well and is used for all subsequent experiments.

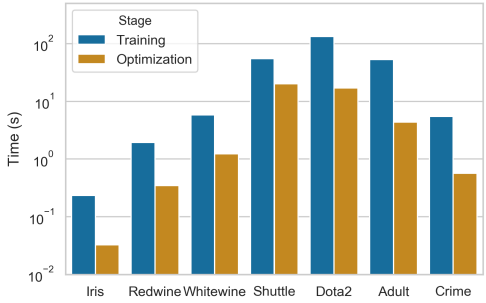

Figure 2. Average master step time, compared to NN training

Scalability We next turn to investigating the method scalability: from this perspective our examples are worst cases, since they must be defined on all the training data, and in some case involve NP-hard problems. We report the average time for a master step in Figure 2. The average time for a learner step (100 epochs of our NN) is shown as a reference. At least in our experimentation, *the time for a master step is always very reasonable*, even for the *dota2* dataset for which we solve NP-hard problems on 74,120 examples. This is mostly due to the clean structure of the m_α and m_β problems. Of course, for sufficiently large training sets, exact solutions will become impractical.

Setup of Alternative Approaches Here we describe how to setup alternative approaches that will be used for comparison. Namely, we consider the regularized linear approach from [3], referred to as RLR, and a Neural Network with Semantic Based Regularization [8], referred to as SBR. Both approaches are based on the idea of introducing constraints as regularizers at training time. Hence, their loss function is in the form:

$$L(f(X;\theta), y^*) + \mu g(f(X;\theta)) \tag{23}$$

The regularization term must be differentiable. We use SBR only for the case studies with the balance constraint, which we are forced to approximate to obtain differentiability:

$$g(f(X;\theta)) = \max_{j=1..c} \sum_{i=1}^{m} f(X;\theta) \tag{24}$$

i.e. we use the sums of the NN output neurons to approximate the class counts and the maximum as a penalty; this proved superior to other attempts in preliminary tests. The L term is the categorical cross-entropy.

[tb]

Table 3. Effect of parameter μ in regularization.

	μ	0.01	0.1	1
Iris	S	**0.984**	0.97	0.4
	C	0.006	0.03	0.13
Redwine	S	0.15	0.15	**0.17**
	C	0.12	0.12	0.04
Whitewine	S	0.17	**0.15**	0.14
	C	0.08	0.02	0.03
Shuttle	S	0.7	**0.31**	0.14
	C	0.14	0.03	0.03
Dota2	S	**0.61**	0.48	0.49
	C	0.2	0.5	0.5
Adult	S	**.83**	.75	.75
	C	1.6	3.1	3
Crime	S	.39	**0.30**	**0.30**
	C	.4	0	0

Our SBR approach relies on the NN model from the previous paragraphs. Since access to the network structure is needed to differentiate the regularizer, SBR works best when all the examples linked by relational constraints can be included in the same batch. When this is not viable the regularizer can be treated stochastically (via subsets of examples), at the cost of one additional approximation. We use a batch size of 2,048 as a compromise between memory usage and noise. The SBR method is trained for 1,600 epochs.

The RLR approach relies on linear models (Logistic or Linear Regression), which are simple enough to consider large group of examples simultaneously. We use this approach for the fairness use cases. In the *crime* (regression) dataset L is the MSE and the regularizer is simply Equation (17). In the *adult* (classification) dataset L is the cross-entropy; the regularizer is Equation (10), with the following substitution:

$$d_{k,v,j} = \left| \frac{1}{m} \sum_{i=1}^{m} \theta^\top x_i - \frac{1}{|X_{k,v}|} \sum_{i \in X_{k,v}} \theta^\top x_i \right|$$

This is an approximation obtained according to [3] by disregarding the sigmoid in the Logistic Regressor to preserve convexity. We train this approach to convergence using the CVXPY 1.1 library (with default configuration). In RLR and SBR classification, the introduced approximations *permit to satisfy the constraints by having equal output for all classes*, i.e. completely random predictions. This undesirable behavior is countered by the L term.

There is no simple recipe for choosing the value of μ in Equation (23); therefore, we performed experiments with different μ values to characterize its effect. The results are reported in Table 3. In most cases, larger μ values tend as expected to result in better constraint satisfaction, with a few notable exceptions for classification tasks (*iris*, *dota*, and *adult*). The issue is *likely due to the approximations introduced in the regularizers*, since it arises even on small datasets that fit in a single mini-batch (*iris*). Further analysis will be needed to confirm this intuitions. The accuracy decreases for a larger μ, as expected, but at a rather rapid pace. In the subsequent experiments, we will use for each dataset the RLR and SBR that offer the best accuracy while being as close to feasible as possible (the "ideal case" column from Table 2): these are the cells in bold font in Table 3.

Alternative Approaches and ML Models We can now compare the performance of Moving Targets using different ML models with that of the alternative approaches presented above (with $\mu = 0.1$), plus a pre-processing approach adapted from [13], referred to as $\mathrm{NN_{pp}}$ and obtained by setting $\alpha, \beta \to \infty$ in our method.

For our method, we consider the following ML models: 1) the NN from the previous section with $\alpha = 1, \beta = 0.1$; 2a) a Random Forest Classifier with 50 estimators and maximum depth of 5 (used for all classification case studies); 2b) a Gradient Boosted Trees model, with 50 estimators, maximum depth 4, and a minimum threshold of samples per leaf of 5 (for the regression case study); 4a) a Logistic Regression model (for classification); 4b) a Linear Regression model (for regression). All models except the NN are implemented using scikit-learn [23]. In the table, the tree ensemble method are reported on a single column, while another column (LR) groups Logistic and Linear regression.

Our algorithm seems to work well with all the considered ML models: tree ensembles and the NN have generally better constraint satisfaction (and higher accuracy for constraint satisfaction) than linear models, thanks to their larger variance. The preprocessing approach is effective when constraints are easy to satisfy (*iris* and *dota2*) and on all the fairness case studies, though less so on the remaining datasets. All Moving Targets approaches tend to perform better and more reliably than RLR and SBR. The case of RLR and LR is especially interesting, as they differ only for the mechanism used to enforce constraint satisfaction. In principle we can expect the two models to have zero gap, for the regularization term has the same formulation and the problem is convex. The gap is then due to an incomplete exploration of the space of the multiplier μ, since for the same multipliers value the gap is null under optimal training (which is the standard for linear regression). The example emphasizes a practical problem that often arises when dealing with regularized loss function: the value of the multiplier has to be thoroughly calibrated by hand, while Moving Targets allows to directly define the desired constraint threshold and obtain the best solution associated to it.

Generalization Since our main contribution is an optimization algorithm, we have focused so far on evaluating its performance on the training data, as it simplifies its analysis. Now that the property of our methods are clearer, we can assess the performance of the models we trained on the test data. The results of this evaluation are reported in Table 5, in the form of average ration between the scores and the level of constraint satisfaction in the test and the train data. With a few exceptions (e.g. satisfiability in *iris*), the models (especially the tree ensembles and LR) generalize well in terms of both accuracy and

Table 4. Benchmarks with different ML models and alternative approaches

		Regularized methods	NN	LR	Ensemble trees	NN_{pp}
Iris	S	$.984 \pm .006$	$\mathbf{.997} \pm .004$	$.96 \pm .02$	$.995 \pm .004$	$.96 \pm .01$
	C	$.006^* \pm .003$	$.013^* \pm .004$	$.014^* \pm .005$	$.012^* \pm .003$	$.014^* \pm .005$
Redwine	S	$.17 \pm .05$	$\mathbf{.506} \pm .006$	$.32 \pm .01$	$.40 \pm .02$	$.480 \pm .001$
	C	$.05 \pm .01$	$.018^* \pm .001$	$.031 \pm .005$	$.08 \pm .01$	$.073 \pm .006$
Whitewine	S	$.15 \pm .03$	$\mathbf{.439} \pm .009$	$.025 \pm .009$	$.37 \pm .04$	$.47 \pm .02$
	C	$.02^* \pm .004$	$.015^* \pm .003$	$.027 \pm .003$	$.10 \pm .01$	$.084 \pm .009$
Shuttle	S	$.31 \pm .04$	$.375 \pm .007$	$\mathbf{.332} \pm .007$	$.51 \pm .05$	$.5 \pm .1$
	C	$.03^* \pm .02$	$.028 \pm .005$	$.023^* \pm .007$	$.11 \pm .01$	$.2 \pm .02$
Dota2	S	$.61 \pm .02$	$.66 \pm .01$	$.592 \pm .005$	$.53 \pm .01$	$\mathbf{.689} \pm .003$
	C	$.2^* \pm .1$	$.09 \pm .06$	$.038 \pm 0$	$.16 \pm .04$	$.02^* \pm .02$
Adult	S	$.834 \pm .001$	$.841 \pm .006$	$.805 \pm .006$	$.76 \pm .01$	$\mathbf{.865} \pm .003$
	C	$1.7 \pm .05$	$.22^* \pm .07$	$.20^* \pm .03$	$.01^* \pm .03$	$.081^* \pm .09$
Crime	S	$.30 \pm .01$	$.48 \pm .03$	$.369 \pm .008$	$.49 \pm .01$	$\mathbf{.484} \pm .008$
	C	0 ± 0	$.2^* \pm .1$	$.02^* \pm 0$	$.24 \pm .01$	$.19^* \pm .02$

constraint satisfaction. Given the tightness of some of the original constraint and the degree to which the labels were altered, this is a remarkable result.

[tb]

Table 5. Generalization of various models in the test scenario

		NN	Ensemble Trees	LR
Iris	S_{ts}/S_{tr}	0.96	0.96	0.99
	C_{ts}/C_{tr}	5.68	5.17	4.31
Redwine	S_{ts}/S_{tr}	0.62	0.92	0.94
	C_{ts}/C_{tr}	1.22	1.04	1.35
Whitewine	S_{ts}/S_{tr}	0.70	0.96	1.00
	C_{ts}/C_{tr}	1.11	1.00	0.99
Shuttle	S_{ts}/S_{tr}	0.99	1.00	0.99
	C_{ts}/C_{tr}	0.97	1.00	1.01
Dota2	S_{ts}/S_{tr}	0.83	1.00	0.99
	C_{ts}/C_{tr}	1.10	1.00	1.03
Adult	S_{ts}/S_{tr}	0.99	1.00	1.00
	C_{ts}/C_{tr}	1.55	1.92	0.98
Crime	S_{ts}/S_{tr}	0.75	0.73	0.93
	C_{ts}/C_{tr}	0.74	1.05	1.03

5 Conclusion

In this paper we have introduced Moving Targets, a decomposition approach to augment a generic supervised learning algorithm with constraints, by iteratively adjusting the example labels. The method is designed to prioritize constraint satisfaction over accuracy, and proved to behave very well on a selection of tasks, constraints, and datasets. Its relative simplicity, reasonable scalability, and the ability to handle any classical ML model make it well suited for use in real world settings.

Many open questions remain: we highlighted limitations of regularization based techniques that deserve a much deeper analysis. The convergence properties of our method still need to be formally characterized (even in simpler, convex, scenarios). The method scalability should be tested on larger datasets, for which using approximate master steps will be necessary. Given the good performance of the pre-processing approach in Table 4, it may be interesting to skip the pretraining step in our method. Improvements may be possible by using specialized algorithms in specific settings: Douglas-Rachford splits could be applied in a numeric setting, or probabilistic predictions could be employed (when available) to refine the projection operators.

Acknowledgement

This research has been partially funded by the H2020 Project AI4EU, grant agreement 825619.

REFERENCES

[1] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos, 'Learning optimal and fair decision trees for non-discriminative decision-making', in *Proceedings of AAAI, IAAI, EAAAI*, pp. 1418–1426, (2019).

[2] Francisco J Aragón Artacho and Rubén Campoy, 'A new projection method for finding the closest point in the intersection of convex sets', *Computational optimization and applications*, **69**(1), 99–132, (2018).

[3] Richard Berk, Hoda Heidari, Shahin Jabbari, Matthew Joseph, Michael J. Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth, 'A convex framework for fair regression', *CoRR*, **abs/1706.02409**, (2017).

[4] Stephen Boyd, Jon Dattorro, et al., 'Alternating projections', *EE392o, Stanford University*, (2003).

[5] Toon Calders and Sicco Verwer, 'Three naive bayes approaches for discrimination-free classification', *Data Mining and Knowledge Discovery*, **21**(2), 277–292, (2010).

[6] Alessandro Daniele and Luciano Serafini, 'Knowledge enhanced neural networks', in *Proceedings of PRICAI 2019*, pp. 542–554, (2019).

[7] Michelangelo Diligenti, Marco Gori, and Claudio Sacca, 'Semantic-based regularization for learning and inference', *Artificial Intelligence*, **244**, 143–165, (2017).

[8] Michelangelo Diligenti, Marco Gori, and Claudio Saccà, 'Semantic-based regularization for learning and inference', *Artificial Intelligence*, **244**, 143 – 165, (2017). Combining Constraint Solving with Mining and Learning.

[9] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[10] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel, 'Fairness through awareness', in *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 214–226, (2012).

[11] 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*, pp. 144–152. SIAM, (2016).

[12] Moritz Hardt, Eric Price, and Nati Srebro, 'Equality of opportunity in supervised learning', in *Advances in neural information processing systems*, pp. 3315–3323, (2016).

[13] Faisal Kamiran and Toon Calders, 'Classifying without discriminating', in *2009 2nd International Conference on Computer, Control and Communication*, pp. 1–6. IEEE, (2009).

[14] Faisal Kamiran and Toon Calders, 'Data preprocessing techniques for classification without discrimination', *Knowl. Inf. Syst.*, **33**(1), 1–33, (2011).

[15] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy, 'Discrimination aware decision tree learning', in *2010 IEEE International Conference on Data Mining*, pp. 869–874. IEEE, (2010).

[16] Guosheng Lin, Chunhua Shen, Anton Van Den Hengel, and Ian Reid, 'Efficient piecewise training of deep structured models for semantic segmentation', in *Proc. of the IEEE CVPR*, pp. 3194–3203, (2016).

[17] Marco Lippi and Paolo Frasconi, 'Prediction of protein β-residue contacts by markov logic networks with grounding–specific weights', *Bioinformatics*, **25**(18), 2326–2333, (2009).

[18] 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*, pp. 502–510, (2011).

[19] Xuezhe Ma and Eduard Hovy, 'End-to-end sequence labeling via bi-directional lstm-cnns-crf', in *Proc. of ACL*, pp. 1064–1074. Association for Computational Linguistics, (2016).

[20] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt, 'Deepproblog: Neural probabilistic logic programming', *arXiv preprint arXiv:1805.10872*, (2018).

[21] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori, 'Integrating learning and reasoning with deep logic models', in *Proc. of ECML*, (2019).

[22] Neal Parikh, Stephen Boyd, et al., 'Proximal algorithms', *Foundations and Trends® in Optimization*, **1**(3), 127–239, (2014).

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

[24] Matthew Richardson and Pedro Domingos, 'Markov logic networks', *Machine learning*, **62**(1-2), 107–136, (2006).

[25] Tim Rocktäschel and Sebastian Riedel, 'End-to-end differentiable proving', in *Advances in Neural Information Processing Systems*, pp. 3788–3800, (2017).

[26] Luciano Serafini and Artur d'Avila Garcez, 'Logic tensor networks: Deep learning and logical reasoning from data and knowledge', *arXiv preprint arXiv:1606.04422*, (2016).

[27] Emile Van Krieken, Erman Acar, and Frank Van Harmelen, 'Semi-supervised learning using differentiable reasoning', *Journal of Applied Logic*, (2019). to Appear.

[28] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork, 'Learning fair representations', in *International Conference on Machine Learning*, pp. 325–333, (2013).