# Unlabeled multi-target regression with genetic programming

**Uriel López**

Tecnológico Nacional de México, Tijuana, BC, Mexico

**Leonardo Trujillo**

Tecnológico Nacional de México, Tijuana, BC, Mexico

**Sara Silva**

Universidade de Lisboa, Lisboa, Portugal

**Leonardo Vanneschi**

Universidade Nova de Lisboa, Lisboa, Portugal & Universidade de Lisboa, Lisboa, Portugal

**Pierrick Legrand**

University of Bordeaux, France

**This is the author accepted manuscript version of the paper published by ACM as:**

Lopez, U., Trujillo, L., Silva, S., Vanneschi, L., & Legrand, P. (2020). Unlabeled multi-target regression with genetic programming. In GECCO 2020: Proceedings of the 2020 Genetic and Evolutionary Computation Conference (pp. 976-984). (GECCO 2020 - Proceedings of the 2020 Genetic and Evolutionary Computation Conference). Association for Computing Machinery. https://doi.org/10.1145/3377930.3389846

# Unlabeled Multi-Target Regression with Genetic Programming

Uriel López
Tecnológico Nacional de México
IT de Tijuana
Tijuana, BC, Mexico
uriel.lopez@tectijuana.edu.mx

Leonardo Trujillo
Tecnológico Nacional de México
IT de Tijuana
Tijuana, BC, Mexico
leonardo.trujillo@tectijuana.edu.mx

Sara Silva
LASIGE, Departamento de Informática,
Faculdade de Ciências,
Universidade de Lisboa
1749-016 Lisboa, Portugal
sara@fc.ul.pt

Leonardo Vanneschi
NOVA Information Management School (NOVA IMS),
Universidade Nova de Lisboa,
Campus de Campolide
1070-312 Lisboa, Portugal
LASIGE, Departamento de Informática,
Faculdade de Ciências,
Universidade de Lisboa
1749-016 Lisboa, Portugal
lvanneschi@novaims.unl.pt

Pierrick Legrand
University of Bordeaux
France
IMB, Institute of Mathematics of Bordeaux, UMR CNRS 5251
France
Inria Bordeaux Sud-Ouest
France
pierrick.legrand@u-bordeaux.fr

## ABSTRACT

Machine Learning (ML) has now become an important and ubiquitous tool in science and engineering, with successful applications in many real-world domains. However, there are still areas in need of improvement, and problems that are still considered difficult with off-the-shelf methods. One such problem is Multi Target Regression (MTR), where the target variable is a multidimensional tuple instead of a scalar value. In this work, we propose a more difficult variant of this problem which we call Unlabeled MTR (uMTR), where the structure of the target space is not given as part of the training data. This version of the problem lies at the intersection of MTR and clustering, an unexplored problem type. Moreover, this work proposes a solution method for uMTR, a hybrid algorithm based on Genetic Programming and RANdom SAmple Consensus (RANSAC). Using a set of benchmark problems, we are able to show that this approach can effectively solve the uMTR problem.

## CCS CONCEPTS

• **Computing methodologies** → *Modeling methodologies*; *Heuristic function construction*; **Genetic programming**; • **Theory of computation** → **Genetic programming**;

## KEYWORDS

Multi-Target Regression, Unlabeled Multi-Target Regression, RANSAC, Genetic Programming, Clustering

-
, ,

...$15.00

Uriel López, Leonardo Trujillo, Sara Silva, Leonardo Vanneschi, and Pierrick Legrand. 2022. Unlabeled Multi-Target Regression with Genetic Programming. In . , 9 pages.

## 1 INTRODUCTION

Machine Learning (ML) provides a wide variety of techniques for solving a large range of real world problems, and the field has grown immensely over recent years. ML techniques are attractive because of their competitive results, friendliness to layman, and because they can be tuned to solve different types of tasks, be it regression, classification, data mining or clustering. However, as the reach of technology has grown, so has the complexity of real world problems, that continues to push researchers towards the development of more specialized ML techniques. For example, when a model is required, for making accurate predictions or just for increasing our understanding of a system based on past historical data, a common approach is to pose a regression task to find a function or model that maps inputs to outputs, as shown in Figure 1. While regression is a common ML task, it is by no means trivial, and can increase in difficulty very suddenly due to several possible factors, which obscure the relationship between inputs and outputs. This scenario occurs due, for instance, to the presence of outliers or gross measurement errors, missing data, or simply due to the nature of high-dimensional data. In such cases, hybrid techniques are very useful. In [9], genetic programming (GP) was applied to solve regression tasks where the datasets were highly contaminated, by up to 90% of outliers in the target variable, a difficult scenario where most techniques fail [10, 14]. In [18] the authors present an analysis of the ten most challenging open problems for the ML community, which range from Developing a Unifying Theory of Data Mining to Security, Privacy and Data Integrity. One of these open problems is Multi-Target Regression (MTR) [1, 2, 15].

Traditional regression can be complicated enough, but MTR extends the complexity of developing a predictive model by considering an $n$ dimensional output space; i.e. by having multiple target variables to predict concurrently. MTR is also known as multi-variate or multi-output regression, which in most cases aims to predict multiple continuous variables using a set of common input variables [1, 15]. MTR falls under the scope of structured output prediction which concerns predicting values of structured data types. Figure 2 shows how in MTR an input matrix is mapped to an output matrix, where each $X_i \in \mathbb{R}^m$ describes $y_i \in \mathbb{R}^n$. Notice that the dimension of the output matrix, number of targets, is known. Examples of MTR can be found in multiple domains. In [6], the goal is to predict quality and conditions of vegetation, while in [8] the goal is to predict the complete audio spectrum of wind noise (vector of six sound pressure values) for a given vehicle component, stock selection in [5], and the simultaneous estimation of different biophysical parameters from remote sensing images in [17], among others.

This paper presents a special case of the MTR task, where the mapping of inputs to outputs is not possible by current MTR solution methods, because these methods assume that the dimensions of the output matrix, number of targets, is given in the problem formulation. When this information is not provided, we call this special case Unlabelled Multi-Target Regression (uMTR), which is presented for the first time in this work. This paper also presents a first approach to solve this problem, an algorithm we call Multi-Target Random Sample Consensus (MTRANSAC), which is an extension of the RANdom SAmple Consensus (RANSAC) algorithm [4] and RANSAC-GP [9] algorithms, which were originally developed for modeling under the presence of large numbers of outliers. The proposal studies problems with targets, each one containing outliers. Using random samples of training instances, it was possible to find multiple target functions for the uMTR. Interestingly, this was possible also in scenarios with large overlap between the different targets.

The paper is organized as follows. Section 2 describes MTR. Section 3 formalizes the definition of uMTR problem and in Section 4 presents the proposed algorithm to solve this task. Section 5 describes the experimental setup and Section 6 presents the results. Finally, this work concludes in Section 7 with closing remarks and future work.

## 2 MULTI-TARGET REGRESSION

MTR is defined by the following elements [2]: (1) An input space $X$, with tuples of dimension $d$, containing primitive data types i.e., $\forall x_i \in X$, $x_i = (x_{i1}, x_{i2}, ..., x_{id})$. (2) An output (target) space $Y$ **with tuples of dimension $n$**, containing real values, i.e., $\forall y_i \in Y, y_i = (y_{i1}, y_{i2}, .., y_{in})$ where $y_{ik} \in \mathbb{R}$ and $1 \leq k \leq n$. (3) A set of training instances $\mathbb{T}$, where each training instance is a pair of tuples from the input and the output space, i.e., $\mathbb{T} = \{(x_i, y_i) | x_i \in X, y_i \in Y, 1 \leq i \leq N\}$ and $N$ is the number of examples in $\mathbb{T}$. (4) A quality criterion $c$, which rewards models with high predictive accuracy and low complexity. (5) The goal is to find a function $f : X \longrightarrow Y$ such that $f$ maximizes $c$. The task of solving MTR problems can be categorized in two general approaches: global methods and local methods [1, 2, 13].

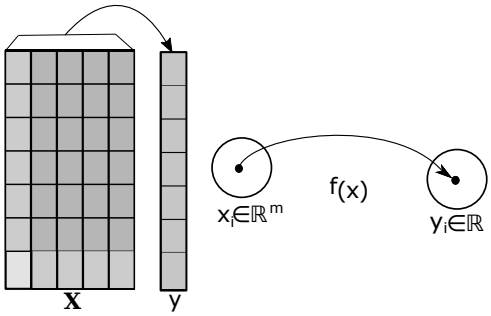



**Figure 1: Traditional symbolic regression problem, where the goal is to find a single function that maps the input space to a single-target.**

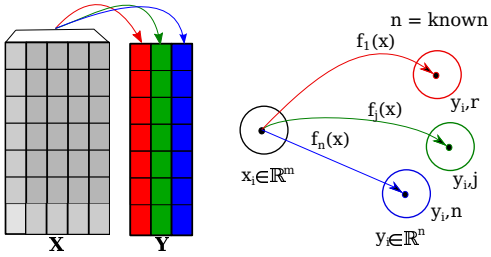



**Figure 2: The MTR problem, where the goal is to find $n$ functions that map the input space to $n$ targets.**

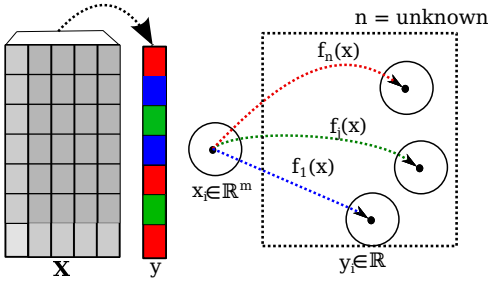



**Figure 3: The uMTR problem, where the goal is to find $n$ functions that map the input space to $n$ targets, with $n$ unknown. The labels (colors) in the output array are not provided in the problem definition, they are shown only for visual reference to highlight the link with the underlying, but hidden, structure in the data.**

*Local methods or problem transformation methods.* These methods decompose the multi-target task into n single-target problems by creating a model for each target and when the task is over, the method returns a set of solutions, or models, one for each target. These methods include single-target methods, multi-target regressor stacking and regressor chains [1].

*Global methods or algorithm adaptation methods.* These methods treat the MTR task as a whole, instead of decomposing it. The rationale beyond global methods is that there may be important loss of information in the local methods. The global methods try to detect

relationships in the structural components of the data, to predict all targets at once. They rely on statistical methods (correlation between target variables), kernel methods or rule methods [1].

While MTR provides the basis for the present work, a wider survey of this problem domain is beyond the scope of the paper, and the interested reader is referred to [1, 2, 15]. To summarize, the main feature of the MTR problem is that it always involves *predicting multiple output targets from a common set of inputs features*, which is illustrated in Figure 2. From this formulation, we now define the uMTR problem.

## 3 UNLABELED MULTI-TARGET REGRESSION

Based on the previous definition of MTR, the uMTR problem can be defined as follows: (1) An input space $X$, with tuples of dimension $d$, containing primitive data types i.e., $\forall x_i \in X$, $x_i = (x_{i1}, x_{i2}, ..., x_{id})$. (2) An output (target) space $Y$ with an unknown number of tuples for each target $n$, containing real values, i.e., $\forall y_i \in Y, y_i = (y_{i1}, y_{i2}, .., y_{in})$ where $y_{ik} \in \mathbb{R}$ and $1 \leq k \leq n$, with $n$ unknown. (3) A set of training instances $\mathbb{T}$, where each training instance is a pair of tuples from the input and the output space, i.e., $\mathbb{T} = \{(x_i, y_i) | x_i \in \mathrm{X}, y_i \in Y, 1 \leq i \leq N\}$ and $N$ is the number of examples in $\mathbb{T}$. (4) Some of the elements $\mathbf{y_{ij}}$ may be missing the output tuples, expressed as $\mathbf{y_{ij}} = \emptyset$. (5) At least one element $\mathbf{y_{ij}} \neq \emptyset$ in each $\mathbf{i-th}$ training instance, but $\mathbf{j}$ is unknown for each training instance. (6) A quality criterion $c$, which rewards models with high predictive accuracy. (7) The goal is to find a set of functions $f_i : X \longrightarrow Y$, such that $f$ maximizes $c$. The key differences between MTR and uMTR are highlighted in bold text in the above definitions. Figure 3 illustrates these differences. The first main difference is that in the uMTR problem the number of targets $n$ is not given as part of the problem definition. This means that the solution method does not know how many target functions need to be derived from the training data. Moreover, the color coding scheme in Figure 3 is merely to illustrate that each value in the output vector is associated to one of the possible targets, but this information is not part of the problem definition. In other words, the elements of the output vector are unlabeled. In this sense, the actual data is presented as in a standard regression task (Figure 1), with the underlying structure of the MTR problem not provided (i.e. the number of targets and the target label for each training instance). Therefore, a solution method for uMTR has to basically solve a MTR problem without access to important information regarding the structure of the problem. An example of a real-world uMTR is the following (a version of which will be studied in future research). Imagine a mobile system or device with a set of environmental sensors, for instance temperature or light sensors, but position information or movement information is not available or provided (due to security constraints). Now imagine that the system is free to move around, maybe passing through $n$ distinct environments. For instance, from an office space, to an outdoor space to a special storage area. Lets also assume that the sensors on the device are monitoring the variables of interest at a fixed sampling rate, and that at any given time window it is impossible to know exactly how many areas the device visits or in what order. Given only the measurements of the sensors, lets also suppose that we want to derive a model of, for instance, the temperature variations captured by the sensor for each of the environments visited by the device. In this case, it would be an error to attempt to derive a single model from all the measurements, since in fact there are $n$ different environments, but it is impossible to know, given our assumptions, $n$ and the correct label (environment) for each sensor reading. This scenario poses a uMTR problem formulation for a learning algorithm.

Another way to look at uMTR is to view it as a clustering problem. However, while clustering is thought of as a form of classification with missing labels, in this case uMTR defines a regression task, also with missing labels. To the authors knowledge, such a regression task is completely unique in literature, and as such requires new techniques to solve. In this sense, it is reasonable to state that the goal of a solution method for a uMTR problem is not necessarily the models, but the labels assigned to each of the training instances that allow us to separate them, and possibly pose new standard regression tasks using the training instances from each detected target. This is an important aspect to consider, since generalization is not necessarily an important initial goal when solving uMTR. Some of the characteristics of uMTR resemble those of the Blind Source Separation (BSS) problem. BSS consists on retrieving a set of unobservable signals (sources) from an observed set of mixed ones [3, 12]. The cocktail party is the best example for explaining BSS. In this scenario there are $N$ sources of audio at the same time: voices, background music, ambient sound and only one source is of interest. The BSS problem involves $N$ number of sensors (microphones) in different parts of the room to record all the sources, to afterwards analyze all of the recordings as a matrix of vectors that contain the superposition of all sources, and try to recover the source signal of interest. BSS can be found in biomedical signals, feature extraction, voice controlled devices and the methods to solved it revolve around principal component analysis or independent component analysis [7]. BSS may resemble uMTR, but BSS assumes that the sources contain at least two sensors [16] while the uMTR problem contains only one reading of the sources. Moreover, in BSS the sources are superimposed while in uMTR the data from different outputs is intact but unlabeled.

## 4 MULTI-TARGET RANSAC

To solve uMTR problems, we propose a computational method called Multi-Target Random Sample Consensus(MTRANSAC). This method is based on the RANSAC [4] algorithm and the recently proposed RANSAC-GP hybrid [9].

*RANSAC and RANSAC-GP.* RANSAC is a method to perform regression in datasets with a high number of outlier training instances. When the number of outliers is above 50% then robust regression methods fail, and a sampling technique such as RANSAC is required. In an iterative manner, RANSAC randomly samples the training set $\mathbb{T}$, and uses each sample to build a candidate solution or model. This sample is called the Minimal Sample Set ($MSS_i \subset \mathbb{T}$), and is of size $s$. RANSAC then builds a model using the training instances in $MSS_i$, call this $K_i$. Then all of the data in $\mathbb{T}$, including the $MSS_i$, is evaluated relative to $K_i$. All of the training instances that are consistent with the model, considering the residuals and a threshold $Th$, define what is known as the Consensus Set $CS_i$ of model $K_i$. This process is repeated until a model is found for which the size of $CS_i$ is above a certain value $v$, or when a maximum number of iterations

**Algorithm 1:** MTRANSAC: Multi-Target RANSAC.

Input: Mixed data set with $N$ training instances $\mathbb{T} = \{(x_i, y_i)\}$ with $i = 1, ..., N$.
Input: Size of the Minimal Sample Set $MSS$, specified by $s$.
Input: Size of the Consensus Set ($CS$), specified by $v$.
Input: Threshold $Th$ used determine if a training instance is part of $CS$.
Input: Maximum number of iteration $j_{max}$.
Out: Set of solutions (models) $\mathbb{M}$.

(1) Initiate solution set $\mathbb{M} = \{\}$ and a counter $j = 1$
(2) Generate a copy of the dataset, $\mathbb{O} = \mathbb{T}$
(3) Take a random sample $MSS_j$, with replacement, of size $s$ from $\mathbb{O}$
(4) Generate a model $K_j$ with the training instances in $MSS_j$ with GP
(5) Compute the residuals $r_i$ for all the training instances in $\mathbb{O}$
(6) Construct a consensus set $CS_j$ with all the training instances in $\mathbb{O}$ where the residual satisfies $r_i \leq Th$
(7) If $|CS_j| \geq v$ then set $\mathbb{M} = \mathbb{M} \cup K_j$, $\mathbb{O} = \mathbb{O} \setminus CS_j$; OTHERWISE do nothing.
(8) Set $j = j + 1$ and repeat steps 2 to 7 UNTIL:
  (a) IF size of $\mathbb{O} = \{\}$ or $j = j_{max}$, THEN return $\mathbb{M}$.
  (b) IF the size of $|\mathbb{O}| < s$ THEN REBOOT the algorithm by setting $\mathbb{M} = \{\}$, and go to Step 2.

is reached, and in some cases $CS_i$ may contain the majority of the $MSS_i$. RANSAC-GP is basically a RANSAC algorithm that uses GP as the modeling process by which the $K_i$ models are constructed within RANSAC. Moreover, in RANSAC-GP a robust fitness measure is used, namely the Least Median Squares (LMS) error measure. This makes RANSAC-GP robust in the presence of outliers, solving problems with as much as 90% of outlier contamination in the training dataset.

*MTRANSAC.* The general outline of MTRANSAC is shown in Algorithm 1. In synthesis, it assumes that for any given possible target in the training data, all other training instances from the other targets are outliers. The process described in Algorithm 1, is similar to RANSAC, but with some key differences. The training dataset $\mathbb{T}$, as said before, is assumed to contain training instances $(x_i, y_i)$ from possibly several targets, and the target (label) of each training instance is unknown. The parameters that need to be specified are the size of the $MSS$ $s$, the size of the $CS$ $v$, the threshold $Th$ used to determine inclusion into the $CS$ for each training instance given a particular model, and the maximum number of iterations $j_{max}$. All of the models produced by MTRANSAC, with a sufficiently large $CS$, are returned in set $\mathbb{M}$. This is the first main difference with respect to RANSAC, MTRANSAC is meant to return $n$ models, one for each target, and RANSAC is meant to return a single one (since most of the training instances are considered to be outliers in that case). Step (1) is an initialization, while steps (2) to (6) are all the same as in RANSAC-GP. The role of GP in this process is in building a representative model for the $MSS$ in step (4). An important element is that, as is done in RANSAC-GP, the GP algorithm used with MTRANSAC uses as a fitness function the LMS error measure. As stated before, MTRANSAC models the training instances of a particular target as outliers of the other targets. Therefore, the LMS measure allows the GP search to focus on the majority of the training instances in the $MSS$, and when it is the case that the majority comes from a single target, then selection pressure will allow GP to converge to a representative model for those instances. In step (7), while RANSAC would stop and return the found model $K_j$, MTRANSAC simply stores the model, removes $CS_j$ from the copy of the dataset, and continues on with the random sampling process, repeating steps (2) to (7) iteratively (step 8) until reaching one of the following conditions. First, the stop condition is reached when the copy of the dataset $\mathbb{O}$ is empty or a maximum number of iterations is reached; i.e. when all of the training instances have been included in the $CS$ of one of the found models or the computation budget has run out. The second condition is called the REBOOT, which means that the process is restarted from step (2). This condition is reached when $|\mathbb{O}| < s$, which indicates that the number of remaining training instances in $\mathbb{O}$ is less than the size of the $MSS$.

The entire algorithm is illustrated in Figure 4, for a cases where $\mathbb{T}$ contains training instances from two sources, shown in Figure 4(a). Then, Figure 4(b) shows the $MSS$ (red filled circles to represent the chosen training instances) and the model generated by GP, respectively steps (3) and (4) in Algorithm 1. Figure 4(c) shows the $CS$, illustrated as yellow circles for each of the training instances in the set, which were determined based on $Th$ that is illustrated by the dotted curves surrounding the model; this corresponds to step (5) in Algorithm 1. Notice that the training instances in $MSS$ are not necessarily in the $CS$, since the model could fail to represent any of the targets, as is the case in Figure 4(c). In this case the $CS$ does not satisfy the condition in step (7), so the process restarts at the beginning. Figure 4(d) shows a new $MSS$, which in this case leads to an accurate model for one of the targets, as can be seen in the $CS$ of Figure 4(e). Finally, Figure 4(f) shows the last full iteration of the algorithm and the second model extracted from this dataset.

It is also imperative to differentiate between solving a traditional problem with GP and using MTRANSAC to solve the uMTR problem. We are not making a distinction between training and testing in the experimental work, the goal is to separate the complete dataset, making the comparison with a clustering process very useful. Nonetheless, the dataset is partitioned at each iteration, where each $MSS_i$ is used to train a GP population, and after each evolutionary process a form of testing is carried out. Here the MTRANSAC

**Table 1: Benchmark problems used in this work, where $U[a, b, c]$ denotes $c$ uniform random samples drawn from the range $[a, b]$, that specifies how the training instances from each target are generated.**

| Problem | Symbolic Expression | Training set |
|---|---|---|
| $f_1$ | $x^3 + x^2 + x$ | U[-1, 1, 100] |
| $f_2$ | $x^5 - 2x^3 + x$ | U[-1, 1, 100] |
| $f_3$ | $x^4 + x^3 + x^2 + x$ | U[-1, 1, 100] |

does not check the quality of the $K_i$ solution, MTRANSAC determines which training instances are congruent with $K_i$ and builds the $CS_i$ for each model.

## 5 EXPERIMENTAL SETTINGS

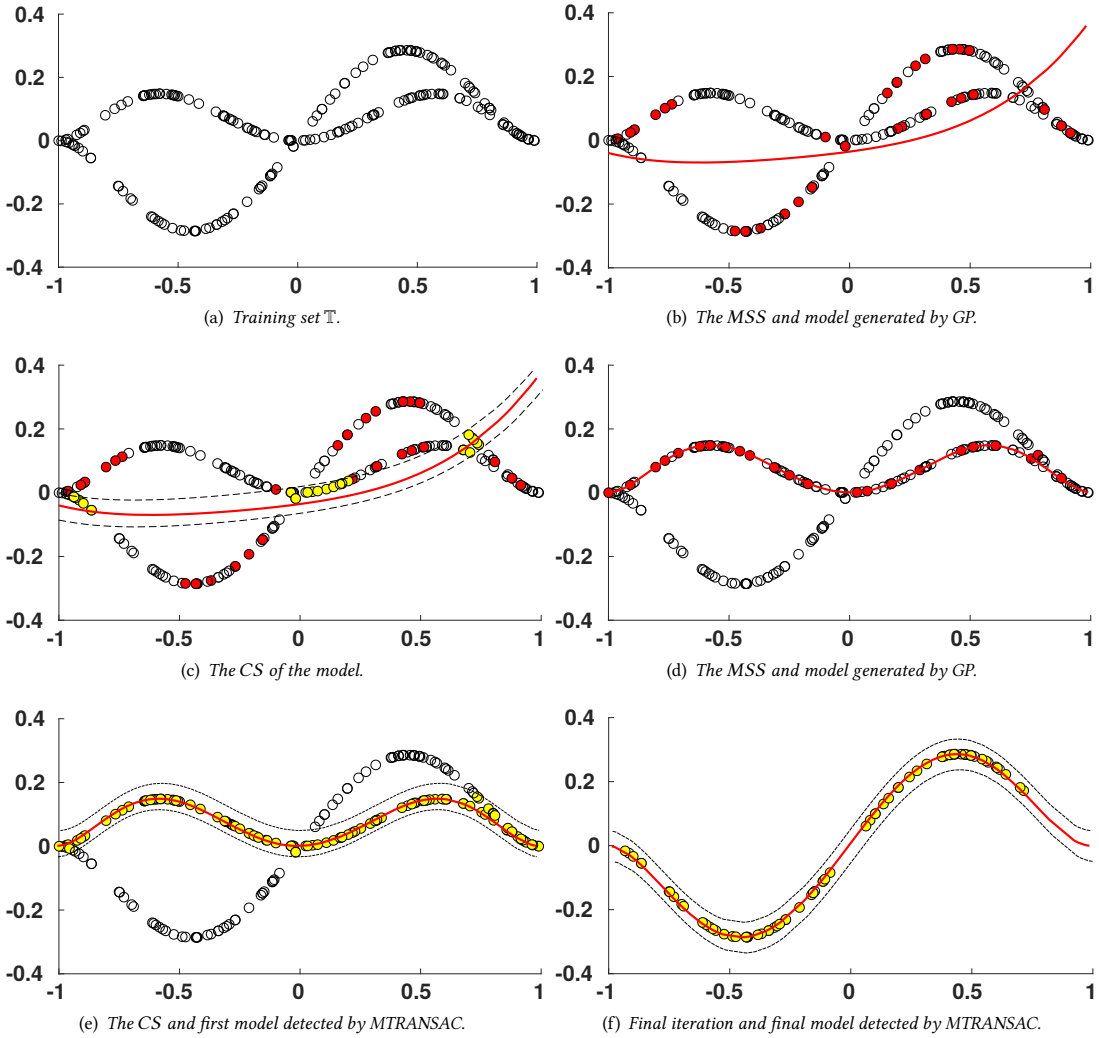
To evaluate if MTRANSAC is capable of solving the uMTR problem, the design of experiments needs to be different from a usual regression task. In order to properly evaluate the algorithm, it is important to select benchmark problems that GP can solve with a high degree of accuracy under normal conditions. It is important to consider that the uMTR problem already describes an extremely challenging task, being defined basically as an MTR problem without some crucial information provided. The goal is to evaluate MTRANSAC as a whole, not specifically the ability of GP to solve each of the individual target tasks. In other words, if benchmarks that are too "difficult" were chosen, they might obfuscate the performance of MTRANSAC. Therefore, three benchmarks are used from [11], that are known to be relatively simple for standard GP to solve in a normal symbolic regression setting [9]. The problems are used, shown in Table 1 that specifies the number of training instances and how they were sampled in the function domain. From each benchmark problem, 100 training instances were randomly generated using a uniform distribution. Another important aspect to consider is the amount of overlap between the two targets, considering both the domain and the codomain of each target (benchmark function). First, considering the domain it is obvious that for the uMTR problem and the MTR problem, an important part of their complexity derives from the fact that the domains of the targets overlap, for if they did not overlap it would be possible to solve the problem using standard regression techniques in a piece-wise manner. For this reason, for the benchmarks used we use the same domain in all three cases. Second, in the case of the codomain, one of the experimental goals will be to evaluate the performance of MTRANSAC as the amount of overlap between the target increases. Intuitively, it is reasonable to suggest that if two targets do not overlap, it should be easier to find each of them, since even a visual inspection might allow to do this visually. On the other hand, if the two targets overlap substantially separating the training instances from each target becomes more difficult. If we think of uMTR as a clustering problem, as we suggested before, it is easier to visualize how the amount of overlap increases problem difficulty. Therefore, in our experimental scenarios we control the amount of overlap between the functions by adding in some cases a bias term to one of the functions, and gradually reducing it until the amount of overlap between the targets is maximized. In total, we define four test cases; these are: **Case 1:** the training set is composed by the training instances from function $f_1$ and $f_2$, adding a bias of 1 to $f_1$. **Case 2:** the training set is composed by the training instances from function $f_1$ and $f_2$, adding a bias of 0.5 to $f_1$. **Case 3:** the training set is composed by the training instances from function $f_1$ and $f_2$, without adding a bias to either function. **Case 4:** the training set is composed by the training instances from function $f_1$, $f_2$ and $f_3$, without adding a bias to any of the functions. The parameters used in our experiments are presented in Table 2 for MTRANSAC, and in Table 3 for the standard tree-based GP used to find the candidate models. In the case of the GP parameters, no parameter tuning was carried out, using an "out of the box" implementation. However, these parameter values were tested on each benchmark function individually, as a standard symbolic regression task, and we found that in all cases training RMSE was always less than 0.01. Training performance is the most important aspect in this case, since the goal is to separate the targets, not necessarily to build models that generalize well. Nonetheless, for these problems the generalization error of GP is very close to the training error.

For MTRANSAC, some of the parameters are left constant in all our experiments, while others are modified and evaluated based on the difficulty of each experimental case, which is controlled by the amount of overlap in the codomain. In particular the size of $CS$ and $Th$ are modified based on the performance observed in each case, while the size of the $MSS$ and the maximum number of iterations $j_{max}$ were left constant. The size of the $MSS$ was based on the parametrization used by RANSAC-GP [9]. In all experiments, a total of 30 runs were executed for each reported configuration, each time using a different sampling of the target functions. A given parameter configuration is said to have failed when at least one of the runs did not correctly differentiate between the target outputs, while the opposite is considered to be a successful configuration. The maximum number of iterations is set to $j_{max} = 50$. In the most difficult Case 4, with three targets that overlap substantially, the maximum number of iterations in all successful configurations was 44 and the average was 11. The REBOOT condition was relatively rare, occurring mostly in Case 4 with a total of 15 REBOOTS over all 30 runs. A final experimental aspect to consider before presenting the main results is the fact that in these experiments the targets are completely balanced; i.e., the number of training instances from each target is the same. Future work will consider the effect of target imbalance, but once again we must emphasize the overall difficulty of the uMTR problem, even if the training data is balanced. For instance, it is worth noting that most work on MTR considers balanced problems [1, 2, 15].

## 6 EXPERIMENTAL RESULTS

The results are organized and presented based on each of the experimental cases defined in the previous section.

*Results for Case 1.* The training instances are shown in Figure 5(a) and a successful configuration with results are shown in Figure 5(b). Figure 5(a) uses two different symbols, one for each target, to illustrate the correct label of each training instance, but this information is not provided to MTRANSAC. Figure 5(b) shows the parameter values used and the final models found in the successful configuration. All subsequent experimental cases show similar plots. This version of the problem was the simplest to solve, since the



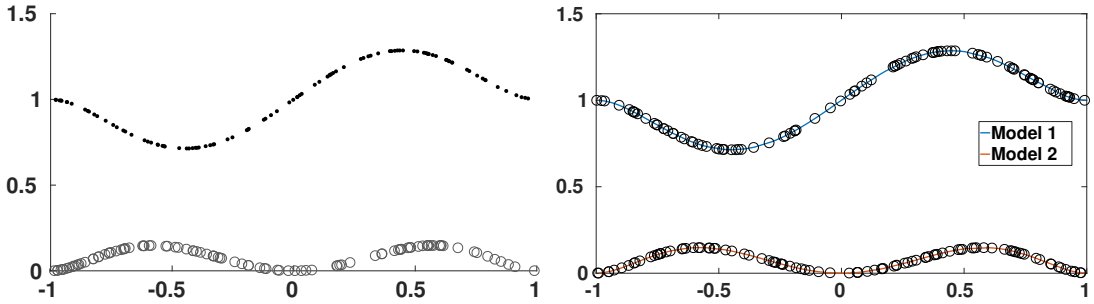

(a) *Training set* $\mathbb{T}$.
(b) *The MSS and model generated by GP.*
(c) *The CS of the model.*
(d) *The MSS and model generated by GP.*
(e) *The CS and first model detected by MTRANSAC.*
(f) *Final iteration and final model detected by MTRANSAC.*

**Figure 4: Illustrative example of MTRANSAC. Training instances are represented as hollow circles, training instances in a *MSS* are shown as red circles, and training instances in a *CS* are shown as yellow circles. The models found in each iteration are shown as red curves, and the threshold parameter $Th$ is shown as dashed lines on either side of the model curves. This example presents a case where two targets are included in the training set, and the algorithm performs three full iterations without reaching the REBOOT condition**

**Table 2: Parameters for the MTRANSAC algorithm .**

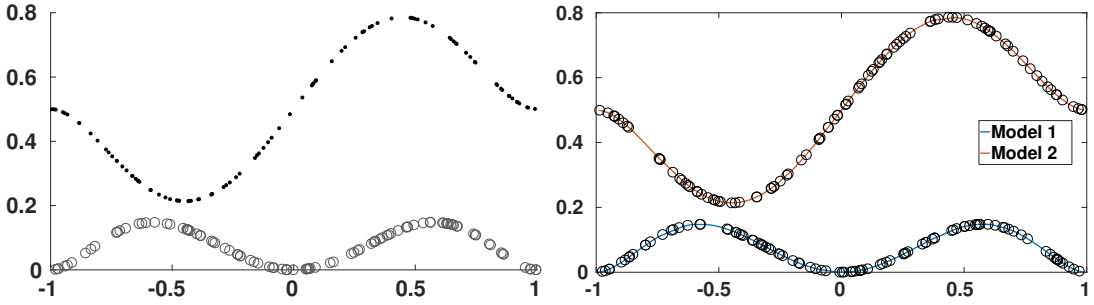
| Parameter | Description | Value |
|---|---|---|
| $v$ | Size of the Consensus Set $CS$ | 100 or 75 training instances<br>Depending on the overlap between targets |
| $Th$ | Threshold to consider an $r_i$ a part of $CS$ | $Th$ = 1.0, 0.5, 0.01, 0.05<br>Depending on the overlap between targets |
| $s$ | Size of the Minimal Sample Set | $s$ = 20 training instances |
| $jmax$ | Maximum number of iterations | $jmax$ = 50 training instances |

targets do not overlap. In these experiments, $Th$ is set to 0.1 and the size of $CS$ is $v = 100$. For the latter, this was set with the knowledge



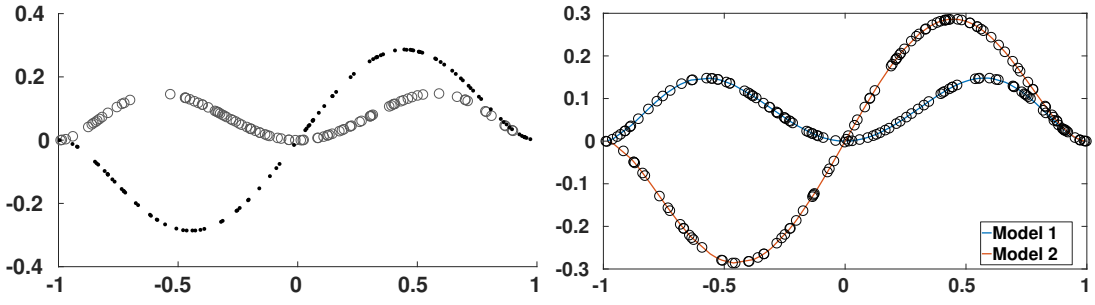

(a) *Training set* $\mathbb{T}$. (b) *Successful configuration with* $Th = 0.1$, $v = 100$.

**Figure 5: Training instances (a) and successful configuration (b) for Case 1. In (a) two different symbols are used, one for each target. In both plots the horizontal axis represents the input variable and the vertical axis is the output.**




(a) *Training set* $\mathbb{T}$. (b) *Successful configuration with* $Th = 0.05$, $v = 100$.

**Figure 6: Training instances (a) and successful configuration (b) for Case 2. In (a) two different symbols are used, one for each target. In both plots the horizontal axis represents the input variable and the vertical axis is the output.**

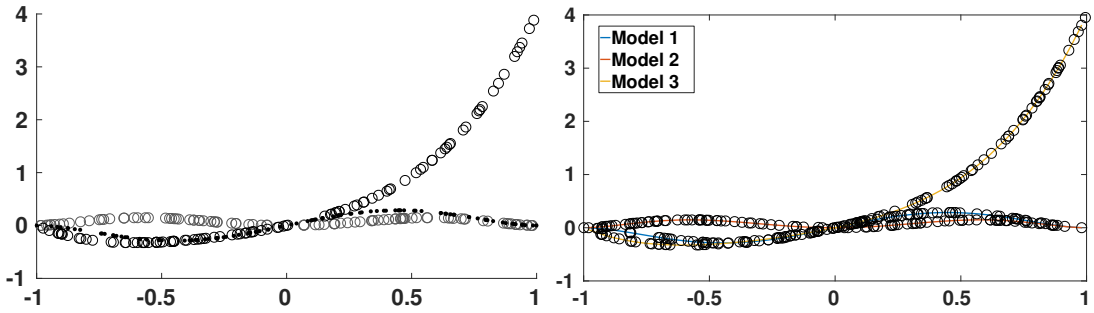



(a) *Training set* $\mathbb{T}$. (b) *Successful configuration with* $Th = 0.05$, $v = 75$.

**Figure 7: Training instances (a) and successful configuration (b) for Case 3. In (a) two different symbols are used, one for each target. In both plots the horizontal axis represents the input variable and the vertical axis is the output.**

that the dataset is balanced. However, slight modifications of this value (as shown in the following experiments) did not degrade performance.

*Results for Case 2.* The results for this experiment are summarized in Figure 6. In this case, given that the two target functions are closer together, the parameters from the previous case did not prove to be successful. The issue is parameter $Th$, the value of 0.1 produced several false positive in the $CS$ of the first model found, which did not allow MTRANSAC to reach the stopping condition. In other words, the $CS$ of the first model detected by MTRANSAC was not composed of training instances from a single target, and was most times larger than $v$, never allowing the second model to meet the required $CS$ size ($v = 100$). The false positives usually appeared near $-0.5$ in the horizontal axis, where both targets almost overlap.




(a) *Training set* $\mathbb{T}$. (b) *Successful configuration with* $Th = 0.01$, $v = 75$.

**Figure 8: Training instances (a) and successful configuration (b) for Case 4. In (a) two different symbols are used, one for each target. In both plots the horizontal axis represents the input variable and the vertical axis is the output.**

**Table 3: GP parameters used in MTRANSAC.**

| Parameter | Description |
|---|---|
| Population size | 100 Individuals |
| Generations | 300 Generations |
| Initialization | *Ramped Half-and-Half*, with maximum depth level 6 |
| Operator probabilities | Crossover $p_c = 0.9$, Mutation $p_\mu = 0.1$ |
| Function set | $(+, -, \times, \div)$ |
| Terminal set | $x, rand(-1, 1)$ |
| Maximum tree depth | 17 levels |
| Selection | Tournament size 3 |
| Elitism | Best individual always survives |

Therefore, the threshold value was reduced to $Th = 0.05$, producing a successful configuration for Case 2 with this modification.

*Results for Case 3.* The results for this experiment are summarized in Figure 7. This scenario removes the bias, leading to a clear overlap in the codomain of both targets. Once again, the previous configuration was not successful in this case. The issue was not $Th$, since it is almost impossible in this case to avoid false positives in the *CS*, since the targets overlap in several regions of the domain. Even a perfect model will include training instances from the other target, at least from the overlap regions. The overlap is not an issue for the modeling process because GP is using a robust fitness function, LMS, so it can derive good models even when the *MSS* is not exclusively composed of training instances from a single target. Therefore, the problem lies with the $v$ parameter, it should allow for an imperfect compositions of the *CS* and not eliminate the possibility of finding the second model when the first model found contains many instances from the second target in it's corresponding *CS*. The solution is to lower the value of $v$, one successful configuration was to set $v = 75$.

*Results for Case 4.* The final experimental case involves adding the third target function, $f_3$, without a bias to any of the functions, producing a notable overlap between all three targets. The results for this experiment are summarized in Figure 8. Figure 8(a) shows that even human visual inspection is very difficult in this case, without the help of the different markers used for each target, it would be very difficult to separate the training instances appropriately. The configuration from Case 3 performed well, but was not successful. Further tuning was required, lowering $Th$ to 0.01 produced a successful configuration. This case illustrates the ability of uMTR to solve a very difficult multi target problem, producing a very effective and efficient detection of the three targets.

## 7 CONCLUSIONS

The contribution of this work to the fields of symbolic regression and machine learning is twofold: first, it presents a new problem formulation, a new and more difficult version of the relatively recent MTR problem. The uMTR problem increases the complexity of the MTR by removing useful information for the solution method, namely the number of targets and the labels of each training instance. One promising perspective of this problem is to visualize it as a clustering task for regression data. Second, it proposes a solution method for uMTR, which is based on the RANSAC algorithm and the hybrid RANSAC-GP. MTRANSAC is shown to be able to solve four experimental cases of increasing complexity. Given that this work represented a first study of uMTR problems, and given that the uMTR formulation poses challenges that might not be solvable with standard techniques, we decide to limit our study to relatively simple benchmark functions. Future work will extend the experimental evaluation of the proposal, using datasets with more target functions and using real-world multivariate data.

## ACKNOWLEDGMENTS

This research was funded by TecNM 2020 project entitled *Resolución de múltiples problemas de aprendizaje supervisado de manera simultánea con programación genética*, and first author is supported by CONACYT graduate scholarship No. 573397.

This work was partially supported by FCT, Portugal, through funding of LASIGE Research Unit (UIDB/00408/2020) and projects DSAIPA/DS/0113/2019, DSAIPA/DS/0022/2018, PTDC/CCI-INF/29168/2017, PTDC/CCI-CIF/29877/2017, PTDC/ASP-PLA/28726/2017 and PTDC/CTA-AMB/30056/2017.

## REFERENCES

[1] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larrañaga. 2015. A Survey on Multi-Output Regression. *Wiley Int. Rev. Data Min. and Knowl. Disc.* 5, 5 (Sept. 2015), 216–233.

[2] Martin Breskvar, Dragi Kocev, and Sašo Džeroski. 2018. Ensembles for Multi-Target Regression with Random Output Selections. *Mach. Learn.* 107, 11 (Nov. 2018), 1673–1709.

[3] G.D Clifford. 2007. Course materials for HST.582J / 6.555J / 16.456J, Biomedical Signal and Image Processing. MIT OpenCourseWare(http://ocw.mit.edu), Massachusetts Institute of Technology. (2007).

[4] Martin A Fischler and Robert C Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.

[5] Joumana Ghosn and Yoshua Bengio. 1997. Multi-Task Learning for Stock Selection. In *Advances in Neural Information Processing Systems 9*, M. C. Mozer, M. I. Jordan, and T. Petsche (Eds.). MIT Press, 946–952.

[6] Dragi Kocev, Sašo Džeroski, Matt White, Graeme Newell, and Peter Griffioen. 2009. Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition. *Ecological Modelling - ECOL MODEL* 220 (04 2009), 1159–1168.

[7] Eleftherios Kofidis. 2016. Blind Source Separation: Fundamentals and Recent Advances (A Tutorial Overview Presented at SBrT-2001). *arXiv preprint arXiv:1603.03089* (2016).

[8] Damjan Kužnar, Martin Možina, and Ivan Bratko. 2009. *Curve prediction with kernel regression.* 61–68. http://lpis.csd.auth.gr/workshops/mld09/mld09.pdf

[9] Uriel López, Leonardo Trujillo, Yuliana Martinez, Pierrick Legrand, Enrique Naredo, and Sara Silva. 2017. *RANSAC-GP: Dealing with Outliers in Symbolic Regression with Genetic Programming.* Springer International Publishing, Cham, 114–130.

[10] Yuliana Martínez, Enrique Naredo, Leonardo Trujillo, Pierrick Legrand, and Uriel Lopez. 2017. A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence* 29 (05 2017).

[11] James Mcdermott, David White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaśkowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, and Una-May O'Reilly. 2012. Genetic Programming Needs Better Benchmarks. *GECCO'12 - Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*, 791–798.

[12] Lucas Parra. 2002. Tutorial on blind source separation and independent component analysis. *Adaptive Image & Signal Processing Group, Sarnoff Corporation* (2002).

[13] Oscar Reyes and Sebastian Ventura. 2019. Performing Multi-Target Regression via a Parameter Sharing-Based Deep Network. *International Journal of Neural Systems* (03 2019).

[14] Lee Spector. 2012. Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion (GECCO Companion '12)*. ACM, 401–408.

[15] Eleftherios Spyromitros-Xioufis, Grigorios Tsoumakas, William Groves, and Ioannis Vlahavas. 2016. Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning* 104, 1 (Feb 2016), 55–98.

[16] Sam T. Roweis. 2001. One Microphone Source Separation. *Adv Neural Inform Process Syst* 13 (02 2001).

[17] Devis Tuia, Jochem Verrelst, Luis Alonso, Fernando Perez-Cruz, and Gustau Camps-Valls. 2011. Multioutput Support Vector Regression for Remote Sensing Biophysical Parameter Estimation. *Geoscience and Remote Sensing Letters, IEEE* 8 (08 2011), 804 – 808.

[18] Qiang Yang and Xindong Wu. 2006. 10 CHALLENGING PROBLEMS IN DATA MINING RESEARCH. *International Journal of Information Technology & Decision Making (IJITDM)* 05, 04 (2006), 597–604.