Multi-Target Support Vector Regression Via Correlation Regressor Chains

Gabriella Melki[a], Alberto Cano[a], Vojislav Kecman[a], Sebastián Ventura[b,c]

[a]*Department of Computer Science, Virginia Commonwealth University, USA*
[b]*Department of Computer Science and Numerical Analysis, University of Cordoba, Spain*
[c]*Department of Computer Science, King Abdulaziz University, Saudi Arabia Kingdom*

Abstract

Multi-target regression is a challenging task that consists of creating predictive models for problems with multiple continuous target outputs. Despite the increasing attention on multi-label classification, there are fewer studies concerning multi-target (MT) regression. The current leading MT models are based on ensembles of regressor chains, where random, differently ordered chains of the target variables are created and used to build separate regression models, using the previous target predictions in the chain. The challenges of building MT models stem from trying to capture and exploit possible correlations among the target variables during training, at the expense of increasing the computational complexity of model training. This paper presents three multi-target support vector regression models. The first involves building independent, single-target Support Vector Regression (SVR) models for each output variable. The second builds an ensemble of random chains using the first method as a base model. The third calculates the targets' correlations and forms a maximum correlation chain, which is used to build a single chained support vector regression model. The experimental study evaluates and compares the performance of the three approaches with ~~six~~ **seven** other state-of-the-art multi-target regressors. The experimental results are then analyzed using non-parametric statistical tests. The results show that the maximum correlation SVR approach improves the performance of using ensembles of random chains.

Keywords: Multi-target regression, multi-output regression, regressor chains, support vector regressor.

1. Introduction

In supervised learning, *single-target* (ST) models are trained to predict the value of a single, categorical or numeric, target attribute of a given example. In some cases, more than one target, or output, can be associated with a single sample input. These situations are handled by a generalization of ST learning, which involves predicting these multiple outputs concurrently, and is known as multi-target (MT) learning [1, 5]. Specifically, MT learning includes *multi-target regression* (MTR), which addresses the prediction of continuous targets, *multi-label classification* [52] which focuses on binary targets, and *multi-dimensional classification* which describes the prediction of discrete targets [5, 40].

Multi-target prediction has the capacity to generate models representing a wide variety of real-world applications, ranging from natural language processing [26] to bioinformatics [37]. Other application areas include ecology [1], gene function prediction [30], predicting the quality of vegetation [22, 31], stock price index forecasting [49], and operations research [5, 23]. Constructing models for these types of real-world problems presents many challenges, such as missing data, (due to targets not being observed or recorded), and noisy data (due to instrument, experimental or human error)**, and the curse of high dimensionality**[1]. **Along with these challenges, the most difficult is identifying patterns between the input data and its corresponding output value. In the context of multi-target modeling, an additional complexity is inherintly created, where multiple outputs must now be trained against. The targets may or may not be correlated, and the corresponding model must accomodate for both scenarios**[2]**.** However, a characteristic of the MT datasets used in these applications and elsewhere, is that they are generated by a single system, **most likely** indicating that the nature of the outputs captured has some structure [23]. Even though modeling the multi-variate nature and possible complex relationships between the target variables is challenging [5], they are more accurately represented by an MT model.

Several base-line methods have been proposed for solving such multi-target tasks such as **Multi-Objective Descision Trees and Random Forests, Boosted Neural Networks, Ensembles of Trees, and many others.** [31], [22], [33]. **Support Vector Machines are a popular set of linear and non-linear supervised**

[1]I added this challengs because it pertains to one of the advantages of SVMs. It will be elaborated on later in the text.

[2]I added this because I wanted to highlight the importance of SVR as a single target model (for non-correlated outputs), as well as max correlation chain in the case of correlation.

machine learning algorithms with a strong theoretical basis on Vapnik-Chervonenkis theory [12]. **It has previously been shown that they outperform most modeling methods in terms of model performance, scalability, and the ability to efficiently deal with outliers** [29, 15]. **Input space dimensionality does not have an adverse affect on the model training time, and furthermore, the final model produced is sparse, allowing for quick predictions.**[3]

There are two main approaches for using such base-line methods in the context of MT learning. ~~**These base-line methods and can be categorized into two groups.**~~ The first being *problem transformation* methods, or *local* methods, in which the multi-target problem is transformed into multiple single-target problems, each solved separately using **classical methods, as described above.** ~~**base, or standard, classification and regression algorithms**~~. The second being *algorithm adaptation* methods, or *global*, or *big-bang* methods, that adapt existing single-target methods to predict all the target variables simultaneously [5, 30]. Using *problem transformation* algorithms for a domain of t target variables, t predictive models must be constructed, each predicting a single-target variable [30]. Prediction for an unseen sample would be obtained by running each of the t single-target models and concatenating their results. Conversely, when using *algorithm adaptation* algorithms for the same domain of t target variables, only one model would need to be constructed which would output all t predictions.

~~**It has previously been shown**~~ **Literature shows** that *algorithm adaptation* methods perform better than *problem transformation* methods [30, 41]. The most valuable advantage of using multi-target techniques is that, not only are the relationships between the sample variables and the targets exploited, but the relationships between the targets amongst themselves are as well [3, 9]. Single-target techniques, on the other hand, eliminate any possibility of learning from the possible relationships between the target variables because a single, independent model is trained for each target separately [4]. Another advantage of MT techniques is model interpretability [1, 49]. A single multi-target model is highly more interpretable than a series of single-target models **because it not only exploits the relationship between the data and targets, but also the targets amongst themselves.** Not only is a single MT model more interpretable, but it could also be considerably more computationally efficient to train, rather than training multiple single-target models individually [2].

This paper presents three novel approaches to solving multi-target regression

[3] I included this because the reviewer wanted a justification for using SVR and how it is unique.

problems. The objective of this research topic is to investigate ~~whether~~ **the performance changes when** building a regression model using **two distinct *algorithm adaptation* chaining methods** ~~chaining is better than~~ **versus** building independent single-target models for each target variable **using a novel framework.** ~~, and whether the maximization of the correlation in a chain performs better than an ensemble of random chains, as conducted by current state-of-the-art algorithms.~~ **This paper's contributions includes the following.**

- Using a Support Vector Regressor (SVR) as a *problem transformation* method to determine whether it outperforms current state-of-the-art ST algorithms, while analyzing its performance as a base-line model for MT chaining methods.

- Building an MT ensemble, using SVR, of randomly chained models (SVRRC), inspired by the classification method, Ensemble of Random Chains Corrected (ERCC) [49], to investigate the effect of exploiting the possible correlations among target variables.

- Eliminating the need to test possible correlations of the target attributes using random chains by creating a single chain, built in the direction of maximum correlation among the targets.

~~The first approach consists of a *problem transformation* method, in which a single-target Support Vector Regression (SVR) model is built for each target variable. This would be the base case for our model comparisons, where each of the models would be independently predicting a single-target variable. The second approach is an *algorithm adaptation* method called Support Vector Regression with Random Chains (SVRRC), inspired by the classification MT method called Ensemble of Random Chains Corrected (ERCC), where random chaining is used to build an ensemble of support vector regressors. In this case, a number of random chains is generated to represent how each of the targets will be chained to train the model. This exploits possible dependencies between the target variables and ensures that they get used during model training. The third *algorithm adaptation* approach eliminates the need to generate a number of random chains, and is called Support Vector Regression with Correlation Chaining (SVRCC). It involves a single maximum correlation chain, in which the targets are arranged in an order that represents their correlation to one another. This ensures that the targets that will be used as input are correlated with the following target prediction. Rather than~~

~~randomly creating chains and having the final model be based on a combination of the generated models, the third approach builds a single model based on the target's maximum correlation.~~ [49]

The experimental study evaluates and compares the performance of the three approaches, together with ~~six~~ **seven** other state-of-the-art multi-target regressors, on a set of 24 datasets with varied input size and output targets. The results of the experiments are analyzed using non-parametric statistical analysis, namely the Bonferroni-Dunn and Wilcoxon tests [18]. These post-hoc tests involve multiple comparisons among the algorithms, where they ~~try and~~ show significant differences in their performances across all datasets. The statistical analysis of the experiments presented in this paper shows the increase in performance of the support vector regressors, specifically, the maximum correlation chain, SVRCC.

The paper is structured as follows. Section 2 reviews related works on multi-target regression. Section 3 presents the three multi-target support vector regression approaches. Section 4 presents the experimental study. Section 5 discusses the results and the statistical analysis. Finally, Section 6 shows the main conclusions of this work.

2. Background

This section defines first the notation that will be used in the paper~~, then it formally~~ **and formally** describes the multi-target regression problem along with the relevant state-of-the-art algorithms.

2.1. Notation

Let $\mathcal{D}$ be a training dataset of $\mathcal{N}$ instances **and their continuous outputs**[4]. Let $\boldsymbol{X} \in \mathcal{D}$ be a matrix consisting of d input variables and $\mathcal{N}$ samples, sometimes called input space, having a domain of $\boldsymbol{X} \in \mathbb{R}^{\mathcal{N}\times d}$. Let $\boldsymbol{Y} \in \mathcal{D}$ be a matrix consisting of m target variables for the $\mathcal{N}$ input samples, sometimes called output space, having a domain of $\boldsymbol{Y} \in \mathbb{R}^{\mathcal{N}\times m}$. For each sample $(\boldsymbol{x}^{(l)}, \boldsymbol{y}^{(l)}) \in \mathcal{D}$, $\boldsymbol{x}^{(l)} = (x_1^{(l)}, \ldots, x_d^{(l)})$ and $\boldsymbol{y}^{(l)} = (y_1^{(l)}, \ldots, y_m^{(l)})$ are the input and output vectors respectively, where $l \in \{1, \ldots, \mathcal{N}\}$. Using the training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(\mathcal{N})}, \boldsymbol{y}^{(\mathcal{N})})\}$, the goal is to learn a multi-target regression model $h : \boldsymbol{X} \times \boldsymbol{Y}$, that assigns a vector $\boldsymbol{y}$ with m target values, for each input instance $\boldsymbol{x}$. The model will then be used to predict the values of $\{\boldsymbol{y}^{(\mathcal{N}+1)}, \ldots, \boldsymbol{y}^{(\mathcal{N}')}\}$ for new

[4]I added this because it didn't include the targets as part of the full dataset

unlabeled input vectors $\{\boldsymbol{x}^{(\mathcal{N}+1)}, \dots, \boldsymbol{x}^{(\mathcal{N}')}\}$. Table 1 summarizes the notation used in this paper.

2.2. Multi-Target Regression Methods

In the context of *problem transformation* for multi-target models, m single-target models will be trained on the dataset $\mathcal{D}_j = \{(x_1^{(1)}, y_j^{(1)}), \dots, (x_d^{(N)}, y_j^{(\mathcal{N})})\}$, where $j \in \{1, \dots, m\}$. This way there are m independent, single-target models, one model for each target variable. This is described as the baseline Single-Target (ST) model in [41]. A generalized visualization of the *problem transformation* method is shown in Table 2. Many *problem transformation* methods have been proposed to solve multi-target problems, which are detailed next.

~~Multiple~~ **Many**[5] authors have proposed Multiple-Target Support Vector Regression methods [5, 49, 50]. Specifically, *Xiong et. al.* presented a support vector regression method with a firefly heuristic in [49], in the context of interval forecasting of a stock price index. Originally proposed in [38], the algorithm was modified in [49] with a firefly heuristic, named FA-MSVR, in order to intelligently

[5]Changed this at the request of Reviewer # 3.

Table 1: Notation

Definition	Notation
Number of Samples	$\mathcal{N}$
Number of Input Attributes	d
Input Space	$\boldsymbol{X} = \{\boldsymbol{X}_1, \dots, \boldsymbol{X}_i, \dots, \boldsymbol{X}_d\} \in \mathbb{R}^{\mathcal{N} \times d}$, $1 \leq i \leq d$
Input Instance	$\boldsymbol{x}^{(l)} = (x_1^{(l)}, \dots, x_d^{(l)}) \in \boldsymbol{X}$, $1 \leq l \leq \mathcal{N}$
Number of Dataset Targets/Outputs	m
Target Space	$\boldsymbol{Y} = \{\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_j, \dots, \boldsymbol{Y}_m\} \in \mathbb{R}^{\mathcal{N} \times m}$, $1 \leq j \leq m$
Predicted Target Space	$\hat{\boldsymbol{Y}} = \{\hat{\boldsymbol{Y}}_{\boldsymbol{1}}, \dots, \hat{\boldsymbol{Y}}_{\boldsymbol{j}}, \dots, \hat{\boldsymbol{Y}}_{\boldsymbol{m}}\} \in \mathbb{R}^{\mathcal{N} \times m}$, $1 \leq j \leq m$
Target Instance	$\boldsymbol{y}^{(l)} = (y_1^{(l)}, \dots, y_m^{(l)}) \in \boldsymbol{Y}$, $1 \leq l \leq \mathcal{N}$
Full Multi-Target (MT) Training Dataset	$\mathcal{D} = \{(x_1^{(1)}, y_1^{(1)}), \dots, (x_d^{(\mathcal{N})}, y_m^{(\mathcal{N})})\}$
Single-Target (ST) Dataset with j^{th} Target	$\mathcal{D}_j = \{(x_1^{(1)}, y_j^{(1)}), \dots, (x_d^{(\mathcal{N})}, y_j^{(\mathcal{N})})\} \in \mathcal{D}$, $1 \leq j \leq m$
Number of Cross-Validation (CV) Sets	k
ST Test Dataset with j^{th} Target, i^{th} CV Fold	$\mathcal{D}_j^{(i)} = \{(x_1^{(i)}, y_j^{(i)}), \dots, (x_d^{(i)}, y_j^{(i)}) \in \mathcal{D}_j$, $i \in \{1, \dots, \mathcal{N}\}$
ST Training Dataset with j^{th} Target, Excluding the i^{th} CV Fold	$\mathcal{D}_j^{(k-i)} = \mathcal{D}_j \setminus \mathcal{D}_j^{(i)}$
MT Regression Model	$h : \boldsymbol{X} \times \boldsymbol{Y}$
ST Regression Model	$h_j : \boldsymbol{X} \times \boldsymbol{Y}_j$, $1 \leq j \leq m$
Unknown Sample	$\boldsymbol{x}^{(\mathcal{N}')} = \{\boldsymbol{x}^{(\mathcal{N}+1)}, \dots, \boldsymbol{x}^{(N')}\}$
Predicted Values for Unknown Sample	$\boldsymbol{y}^{(\mathcal{N}')} = \{\boldsymbol{y}^{(\mathcal{N}+1)}, \dots, \boldsymbol{y}^{(\mathcal{N}')}\}$

Table 2: Transformation to m Single-Target Datasets

Dataset	Values	Output
$\mathcal{D}_1$	$\{(x_1^{(1)}, y_1^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_1^{(\mathcal{N})})\}$	$h_1 : \mathcal{D}_1 \rightarrow \mathbb{R}$
$\mathcal{D}_2$	$\{(x_1^{(1)}, y_2^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_2^{(\mathcal{N})})\}$	$h_2 : \mathcal{D}_2 \rightarrow \mathbb{R}$
$\vdots$	$\vdots$	$\vdots$
$\mathcal{D}_j$	$\{(x_1^{(1)}, y_j^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_j^{(\mathcal{N})})\}$	$h_j : \mathcal{D}_j \rightarrow \mathbb{R}$
$\vdots$	$\vdots$	$\vdots$
$\mathcal{D}_m$	$\{(x_1^{(1)}, y_m^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_m^{(\mathcal{N})})\}$	$h_m : \mathcal{D}_m \rightarrow \mathbb{R}$

identify the appropriate SVR hyper-parameters. Their method for optimizing the FA-MSVR was using an iterative reweighted least squares (IRWLS) approach based on a quasi-Newton strategy. To produce attractive results using support vector machines, appropriate hyper-parameters must be selected. This is a crucial and computationally expensive step to ensure the SVR model is performing to the best of its capabilities [53], which is why a firefly heuristic was used in [49]. The results obtained by *Xiong et. al.* indicate that FA-MSVR proved to be a promising alternate method for time series forecasting, thus highlighting the importance of setting the SVR hyper-parameters.

Moreover, other approaches based on Linear Target Combinations for MT Regression [45], and Multi-Objective ~~Decision Trees~~ **Random Forests**[6] (MORF) [32] have been proposed. Most commonly investigated issues for MT learning problems include dimensionality reduction for high-dimensional multi-labeled data. The curse of dimensionality is problematic due to the possible correlations between the data and the targets [11, 14, 25]. This multi-collinearity may cause the learning task to be more difficult and complex [39]. To reduce the dimensionality of the data, without losing the possible relationships to the targets, careful feature selection could be performed, which is a difficult task as well [34, 35]. Another issue would be processing large datasets quickly and feasibly (because of memory constraints), while providing insightful information [5, 8].

The RC, MTS, MTSC, ERC, and ERCC methods are introduced by *Spy-*

[6]This was a typo.

romitros et. al. in [41]. The idea behind these algorithms was to investigate whether advances in multi-label learning can be successfully used and implemented in a multi-target regression setting, as well as shedding light on modeling target dependencies. They first describe the RC, MTS, and ERC models, which are ~~both~~ inspired by multi-label classification algorithms, and then introduce their corrected versions. These ~~two~~ methods involve two stages of learning, the first being building ST models and the second uses the knowledge gained by the first step to predict the target variables while using possible relationships the target variables might have with one another.

The two stages of training in MTS involve firstly, training m independent single-target models, like in ST. In the second step, a second set of m meta models are learned for each target variable, $\boldsymbol{Y}_j$, $1 \leq j \leq m$. These meta models are learned on a transformed dataset, where the input attributes space is expanded by adding the approximated target variables obtained in the first stage, excluding the j^{th} target being predicted. Each m meta model, $h_j^* : \boldsymbol{X} \times \mathbb{R}^{m-1} \rightarrow \mathbb{R}$, is learned by the modified dataset $\mathcal{D}_j^* = \{(x_1^{*(1)}, y_j^{*(1)}), \ldots, (x_d^{*(\mathcal{N})}, y_j^{*(\mathcal{N})})\}$, where $\boldsymbol{x}_j^{*(l)} = \{x_1^{(l)}, \ldots, x_d^{(l)}, \hat{y}_1^{(l)}, \ldots, \hat{y}_{j-1}^{(l)}, \hat{y}_{j+1}^{(l)}, \ldots, \hat{y}_m^{(l)}\}$, $l \in \{1, \ldots, \mathcal{N}\}$ are the input vectors along with the $m-1$ predicted target variables, represented by $\hat{\boldsymbol{y}}$, obtained by the first step. To predict the output for a new input vector $\boldsymbol{x}^{(q)}$, the models trained in the first stage are applied and an output vector, $\hat{\boldsymbol{y}}^{(q)} = \{\hat{\boldsymbol{y}}_1^{(q)}, \ldots, \hat{\boldsymbol{y}}_m^{(q)}\} = \{h_1(\boldsymbol{x}^{(q)}), \ldots, h_m(\boldsymbol{x}^{(q)})\}$, is obtained. The second stage models are then applied on the transformed input vector $\boldsymbol{x}_j^{*(q)}$, as shown above, to produce a final output vector.

The ERC method is somewhat similar to the MTS method. In the training of a Regression Chain (RC) model, a random chain, or sequence, of the set of target variables is selected and for each target in the chain, models are built sequentially by using the output of the previous model as input for the next [42]. If the default, ordered chain is $C = \{\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_m\}$, the first model $h_1 : \boldsymbol{X} \rightarrow \mathbb{R}$ is trained for $\boldsymbol{Y}_1$, as in ST. For the subsequent models $h_{j,j>1}$, the dataset is transformed into $\mathcal{D}_j^* = \{(\boldsymbol{x}_j^{*(1)}, y_j^{(1)}), \ldots, (\boldsymbol{x}_j^{*(\mathcal{N})}, y_j^{(\mathcal{N})})\}$, where $\boldsymbol{x}_j^{*(l)} = \{x_1^{(l)}, \ldots, x_d^{(l)}, y_1^{(l)}, \ldots, y_{j-1}^{(l)}\}$ are the input vectors transformed by sequentially appending the true values of each of the previous targets in the chain. For a new input vector $\boldsymbol{x}^{(q)}$, the target values are unknown. So once the models are trained, the unseen input vector $\boldsymbol{x}^{(q)}$ will be appended with the approximated target values, making the models dependent on the approximated values obtained in each step. One of the issues associated with this method is that, if a single random chain is used, the possible relationships between the targets at the head of the chain and the end of the chain are not exploited due to the algorithm's sequential nature. Also, prediction error in the earlier stages of the models will be propagated as the rest of the models are trained, which is why the Ensemble of Regressor Chains was proposed in [41]. Instead of a single chain, k chains are created at random, and the final prediction values are obtained by taking the mean values of the k predicted values for each target.

In the methods described above, the estimated target variables (meta-variables) are used as input in the second stage of training. In both methods, the models are trained using these meta-variables that become noisy at prediction time, and thus the relationship between the meta-variables and target variable is muddied. Dividing the training set into sets, one for each stage, would not help this situation because both methods would be trained on training sets of decreasing size. Due to these issues, *Spyromitros et. al.* proposed modifications, in [41], to both methods that resembles k-fold cross-validation (CV) to be able to obtain unbiased estimates of the meta-variables. These methods are called Regression Chains Corrected (RCC) and Multi-Target Stacking Corrected (MTSC).

The ERCC and MTSC procedures involve repeating the RCC and MTS procedures k times, respectively, with k randomly ordered chains for ERCC, and k different modified training sets for MTSC. The algorithms were tested and compared using Bagging of 100 regression trees as their base regression algorithm with ERC and ERCC ensemble size of 10, and 10-fold cross-validation. The corrected methods exhibited better performance than their original variants, as well as ST models. The ERCC algorithm had the best overall performance, as well as being statistically significantly more accurate of all the methods tested. These methods can be found and used through the open-source Java library, Mulan [44]; to replicate the results found in [41].

The following section presents our approaches to solving the multi-target regression problem inspired by the techniques presented in [41]. It will show that exploiting and making use of the possible correlations in the target variables produces better results than not.

3. Multi-target SVR proposal

Three novel models have been implemented for the purposes of multi-target regression. The ~~base model~~ **base-line model is the *problem transformation* method, named** ~~is the~~ SVR, where m single-target soft-margin non-linear support vector regressors (NL-SVR) are built for each target variable $\boldsymbol{Y}_j$. For NL-SVR, the regularized soft-margin loss function given in equation (1) is minimized [15, 29],

$$\underset{\boldsymbol{w},\xi,\xi^*}{\text{minimize}} \quad \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{l=1}^{\mathcal{N}}\left(\xi^{(l)} + \xi^{*(l)}\right) \tag{1}$$

$$\text{subject to} \begin{cases} y^{(l)} - \langle \boldsymbol{w}, \phi\left(\boldsymbol{x}^{(l)}\right)\rangle \leq \epsilon + \xi^{(l)} & \text{(1a)} \\ \langle \boldsymbol{w}, \phi\left(\boldsymbol{x}^{(l)}\right)\rangle - y^{(l)} \leq \epsilon + \xi^{*(l)} & \text{(1b)} \\ \xi^{(l)}, \xi^{*(l)} \geq 0 & \text{(1c)} \end{cases}$$

where $\boldsymbol{w}$ represents the SVR weight vector, ϵ represents how precise the approximations are, $C > 0$ determines how to penalize deviations from ϵ, $\phi(\cdot)$ represents a feature mapping function, $\xi^{(l)}$ and $\xi^{*(l)}$ are slack variables, and $y^{(l)}$ is the label corresponding to the input vector $\boldsymbol{x}^{(l)}$. For simplicity, the bias SVR term has been excluded. In our algorithm implementation, the dual of this formulation [12, 15] given by (2) is solved,

$$\begin{aligned} \underset{\boldsymbol{\alpha},\boldsymbol{\alpha}^*}{\text{maximize}} \quad & -\frac{1}{2}\sum_{l,k=1}^{\mathcal{N}}\left(\boldsymbol{\alpha}^{(l)} - \boldsymbol{\alpha}^{*(l)}\right)\left(\boldsymbol{\alpha}^{(k)} - \boldsymbol{\alpha}^{*(k)}\right)\mathcal{K}\left(\boldsymbol{x}^{(l)}, \boldsymbol{x}^{(k)}\right) \\ & -\epsilon\sum_{l=1}^{\mathcal{N}}\left(\boldsymbol{\alpha}^{(l)} + \boldsymbol{\alpha}^{*(l)}\right) + \sum_{l=1}^{\mathcal{N}} y^{(l)}\left(\boldsymbol{\alpha}^{(l)} - \boldsymbol{\alpha}^{*(l)}\right) \end{aligned} \tag{2}$$

$$\text{subject to} \quad \textstyle\sum_{l=1}^{\mathcal{N}}\left(\boldsymbol{\alpha}^{(l)} - \boldsymbol{\alpha}^{*(l)}\right) = 0, \quad \boldsymbol{\alpha}^{(l)}, \boldsymbol{\alpha}^{*(l)} \in [0, C] \tag{2a}$$

where the $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ vectors correspond to the SVR dual variables and $\mathcal{K}\left(\boldsymbol{x}^{(l)}, \boldsymbol{x}^{(k)}\right) = \phi(\boldsymbol{x}^{(l)})' * \phi(\boldsymbol{x}^{(k)})$ is an $(\mathcal{N} \times \mathcal{N})$ Gaussian kernel matrix which is dependent on the γ parameter for its broadness. Using the SVR optimization problem described, the multi-target problem is solved by transforming it into m single-target problems, as shown in Algorithm 1 and Figure 1. This algorithm will output m single-target models, h_j, $\forall j = 1, \ldots, m$, for a given dataset $\mathcal{D}$. It first splits the dataset into m separate ones, each with a single-target variable $\boldsymbol{Y}_j$, and then builds a distinct SVR model for each of the datasets. **For predicting the output values for a new and unseen instance, each of the m models would have to be run on the same sample.**

There are many advantages to using an SVM as a base-line model. Firstly, the optimization problem defining it contains a regularization parameter,

Figure 1: SVR Flow Diagram

$$\begin{tikzcd}
 & \mathcal{D}_1 : [\boldsymbol{X}][\boldsymbol{Y}_1] \arrow[r] & h_1 : \mathcal{D}_1 \to \hat{\boldsymbol{Y}}_{\boldsymbol{1}} \\
 & \mathcal{D}_2 : [\boldsymbol{X}][\boldsymbol{Y}_2] \arrow[r] & h_2 : \mathcal{D}_2 \to \hat{\boldsymbol{Y}}_{\boldsymbol{2}} \\
\mathcal{D} : [\boldsymbol{X}][\boldsymbol{Y}] \arrow[uur, bend left] \arrow[ur, bend left] \arrow[dr, bend right] & \vdots & \\
 & \mathcal{D}_m : [\boldsymbol{X}][\boldsymbol{Y}_m] \arrow[r] & h_m : \mathcal{D}_m \to \hat{\boldsymbol{Y}}_{\boldsymbol{m}}
\end{tikzcd}$$

The first step of the SVR model involves separating the MT dataset into m ST datasets, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m$. It then individually trains an independent model, $h_1, h_2, \ldots, h_m$, for each ST dataset.

Algorithm 1 MT Support Vector Regression (SVR)

```
Input: Training dataset $\mathcal{D}$, number of cross-validation folds $k$
Output: ST models $h_j, j = 1, \ldots, m$
    Build $m$ ST SVR Models
 1: for $j = 1$ to $m$ do
 2:     $\mathcal{D}_j = \{(x_1^{(1)}, y_j^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_j^{(\mathcal{N})})\}$
        Create the CV training $\mathcal{D}^{(k-i)}$ and test $\mathcal{D}^{(i)}$ sets
 3:     for $i = 1$ to $k$ do
 4:         $\mathcal{D}_j^{(k-i)} = \{(x_1^{(k-i)}, y_j^{(k-i)}), \ldots, (x_d^{(k-i)}, y_j^{(k-i)})\}$
 5:         $\mathcal{D}_j^{(i)} = \{(x_1^{(i)}, y_j^{(i)}), \ldots, (x_d^{(i)}, y_j^{(i)})\}$
            Train the $j^{th}$ model on the training set $\mathcal{D}^{(k-i)}$
 6:         $h_j^{(k-i)} : \mathcal{D}_j^{(k-i)} \to \mathbb{R}$
            Test the $j^{th}$ model on the test set $\mathcal{D}^{(i)}$
 7:         $\hat{\mathbf{Y}}_j^{(i)} = h_j^{(k-i)}(\boldsymbol{X}^{(i)})$
 8:     end for
        Calculate and store aRRMSE error for the $j^{th}$ model
 9: end for
10: return $h_j, j = 1, \ldots, m$
```

which prevents model overfitting to the training dataset. Secondly, the optimization problem is convex, indicating a unique global minimum, which can

be found efficiently. Furthermore, it also allows for use of the kernel trick which allows for more flexibility and knowledge in model training as well as a final sparse model, based only on the number of support vectors [6] [29].

Building m ST models was a good base-line model, but as mentioned previously, it does not use any of the possible correlations between the target attributes during training. If these correlations are not exploited, it could retract from the model's potential performance. Therefore, we also proposed to construct a series of random chains and create an ensemble model, as done in [41], but using our base-line SVR method, named SVR Random Chains (SVRRC). For SVRRC, ensembles of at most 10 random chains are built, with length m, of different and distinct permutations of the target variable indices. For each chain, we train m chained models using the targets true values. **An illustration of this process is shown in Figure** 2. Due to the computational complexity of building $m!$ distinct chains and training $(m!) \times m$ models, the number of ensembles and chains are limited to a maximum of 10.~~, as proposed by *Spyromitros et. al.* in~~ [41][7]. However, if the number of target variables is less than 3, i.e. $m! \leq 10$, we construct all $m!$ random chains.

For each of the random chains, the model is trained by predicting the first target variable in the chain. Next, the first target's true value, $\boldsymbol{Y}_j$, is appended to the end of the training set as such, $\mathcal{D}^*_{j+1} = \{\boldsymbol{X}^*_{j+1}, \boldsymbol{Y}_j\}$, where $\boldsymbol{X}^*_{j+1}$ is the appended training set, $\boldsymbol{X}^*_{j+1} = \{x_1^{(l)}, \ldots, x_d^{(l)}, y_1^{(l)}, \ldots, y_j^{(l)}\}, l = \{1, \ldots, \mathcal{N}\}$.

The chaining process is repeated for all the target indices in the chains. When chaining target values, there are two main options: using the predicted value as input for the following target, or using the true value of the target variable as input of the subsequent targets. The main problem with the former approach is that errors are propagated **throughout the training process. As each estimated, predicted value gets appended to the training set, additional noise now exists in the input space when training for the next target. However, the latter approach,** minimizes error propagation, ~~and results in better accuracy results~~ **resulting in a more accurate model**. Our approach ~~and the other methods compared~~ employs chaining of the true values**, rather than appending the targets estimation produced while training**. Given the ensemble of SVRs, the predicted values for a given instance are calculated by taking the mean of the multiple models generated using different random chains. For predicting unseen inputs that have

[7]I removed this because we already mentioned this in the same paragraph. I felt it was a little redundant.

Figure 2: SVRRC Flow Diagram

$$\begin{tikzcd}
 & \mathcal{D} : [\boldsymbol{X}][\boldsymbol{Y}_1\boldsymbol{Y}_2\boldsymbol{Y}_3] \arrow[dl] \arrow[dr] & \\
{[1,2,3]} \arrow[d] & \dots & {[3,2,1]} \arrow[d] \\
h_1 : [\boldsymbol{X}] \to \hat{\boldsymbol{Y}}_{\mathbf{1}} \arrow[d] & \dots & h_1 : [\boldsymbol{X}] \to \hat{\boldsymbol{Y}}_{\mathbf{3}} \arrow[d] \\
h_2 : [\boldsymbol{X}\boldsymbol{Y}_{\mathbf{1}}] \to \hat{\boldsymbol{Y}}_{\mathbf{2}} \arrow[d] & \dots & h_2 : [\boldsymbol{X}\boldsymbol{Y}_{\mathbf{3}}] \to \hat{\boldsymbol{Y}}_{\mathbf{2}} \arrow[d] \\
h_3 : [\boldsymbol{X}\boldsymbol{Y}_{\mathbf{1}}\boldsymbol{Y}_{\mathbf{2}}] \to \hat{\boldsymbol{Y}}_{\mathbf{3}} & \dots & h_3 : [\boldsymbol{X}\boldsymbol{Y}_{\mathbf{3}}\boldsymbol{Y}_{\mathbf{2}}] \to \hat{\boldsymbol{Y}}_{\mathbf{1}}
\end{tikzcd}$$

This figure illustrates how the SVRRC algorithm trains a dataset with 3 target variables. It first builds the 6 random chains of the target's indices (the first and last chains are shown above). It then constructs a chained model by proceeding recursively over the chain, building a model, and appending the current target to the input space to predict the next target in the chain.

Figure 3: SVRCC Flow Diagram

$$\begin{tikzcd}
\mathcal{D} : [\boldsymbol{X}][\boldsymbol{Y}_1\boldsymbol{Y}_2\boldsymbol{Y}_3] \arrow[rr, "\textit{generate maximum correlation chain}"] & & {[1,2,3]} \arrow[dll, rounded corners] \\
h_1 : [\boldsymbol{X}] \to \hat{\boldsymbol{Y}}_{\mathbf{1}} \arrow[r] & h_2 : [\boldsymbol{X}\boldsymbol{Y}_{\mathbf{1}}] \to \hat{\boldsymbol{Y}}_{\mathbf{2}} \arrow[r] & h_3 : [\boldsymbol{X}\boldsymbol{Y}_{\mathbf{1}}\boldsymbol{Y}_{\mathbf{2}}] \to \hat{\boldsymbol{Y}}_{\mathbf{3}}
\end{tikzcd}$$

This figure shows how the SVRCC algorithm trains a sample dataset with 3 taget variables. It first finds the direction of maximum correlation among the targets and uses that order as the only chain. It then constructs the chained model as done in SVRRC.

no target values, the predicted value at each step of the chain $\hat{y}_j^{(l)}$ is appended to the input as shown in Algorithm 3. For SVRRC described in Algorithm 2, at most 10 models will be built (one for each chain RC, $RC_c = \{RC_c^{(1)}, \dots, RC_c^{(m)}\}$, $c \leq 10$), each iterating and training m chained models. For each of the models in the chain, we use 10-fold cross-validation to ensure the hyper-parameters chosen best describe the data.

When the number of output variables increases, the number of possible chains increases factorially. Therefore, there is no guarantee that the random chains generated will truly reflect the relationships among the target variables. Moreover, building an ensemble of regressors is computationally expensive. Finding a heuristic that allows the identification of a single, most appropriate chain,

Algorithm 2 MT SVR with Random-Chains (SVRRC)

```
Input: Training dataset $\mathcal{D}$, number of cross-validation folds $k$
Output: $c$ chained models $h_j, j = \{1, \ldots, m\}, c \leq 10$
        Create at most 10 distinct random chains of size $m$
 1: $RC_c^{(m)}$
        Create ST transformation of $\mathcal{D}$ with the $1^{st}$ index in chain $c$
 2: $\mathcal{D}_1^* = \{(x_1^{(1)}, y_1^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_1^{(\mathcal{N})})\}$
        Build $m$ chained models for each chain $c \in \{1, \ldots, 10\}$
 3: for $j = 1$ to $m$ do
 4:     $h_j : \mathcal{D}_j^* \to \mathbb{R}$
        Initialize appended dataset, $\mathcal{D}_{j+1}^*$
 5:     if $j < m$ then
 6:         $\mathcal{D}_{j+1}^* \leftarrow \emptyset$
            Split $\mathcal{D}_j^*$ into $k$ disjoint parts $\mathcal{D}_j^{*(i)}$, $i = 1, \ldots, k$ for training and testing
 7:         for $i = 1$ to $k$ do
 8:             $h_j^{(i)} : \mathcal{D}_j^{(k-i)} \to \mathbb{R}$
                Test the model on the test set $D_j^{*(i)}$
 9:             for $\boldsymbol{x}^{*(i)} \in \mathcal{D}_j^{*(i)}$ do
10:                 $\hat{y}_j^{(i)} = h_j^{(i)}(\boldsymbol{x}^{*(i)})$
                    Append $\boldsymbol{x}^{*(i)}$ with the true value $y_j^{(i)}$, and add it to set $\mathcal{D}_{j+1}^*$
11:                 $\mathcal{D}_{j+1}^* = \mathcal{D}_{j+1}^* \cup [\boldsymbol{x}^{*(i)}, y_j^{(i)}]$
12:             end for
13:         end for
14:     end if
15: end for
16: return $h_j, j = 1, \ldots, m$
```

which fully reflects the output variable interrelations would improve the computational complexity of training the ensemble. Our third proposal builds a single chain based on the maximization of the correlations among the target variables. By calculating the correlation of the target variables and imposing that on the order of the chain, we ensure that each appended target provides some additional knowledge on the training of the next. With SVRRC, there is no proof or reasoning behind the generation of these chains, and since the number of random chains generated is limited to 10, there is no way of ensuring that the 10

Algorithm 3 SVR Chained Prediction

```
Input: Unknown sample $\boldsymbol{x}^{(q)}$, chained model $h_j, j = \{1...m\}$
Output: Output vector $\hat{\boldsymbol{y}}^{(q)}$ of size $m$
     Initialize the output vector
  1: $\hat{\boldsymbol{y}}^{(q)} = \mathbf{0}$
     Initialize the input space for the first target
  2: $\mathcal{D}_1^* = \boldsymbol{x}^{(q)}$
  3: for $j = 1$ to $m$ do
  4:     $\boldsymbol{x}^* \in \mathcal{D}_j^*$
  5:     $\hat{y}_j^{(q)} = h_j(\boldsymbol{x}^*)$
  6:     $\mathcal{D}_{j+1}^* = \mathcal{D}_j^* \cup [\boldsymbol{x}^*, \hat{y}_j^{(q)}]$
  7: end for
     Create ST transformation of $\mathcal{D}$ with $1^{st}$ index in chain $c$
  8: return $\hat{\boldsymbol{y}}^{(q)}$
```

chains fully represent the targets' dependencies. Therefore, calculating and using the correlations of the targets would break this uncertainty, as done in Algorithm 4 and Figure 3, the SVR Correlation Chain (SVRCC) method. The computational complexity and hardware constraints (memory size) are negligible during the construction of the targets' correlation matrix, since the correlation matrix would be an $(m \times m)$ matrix, and the likelihood that the number of targets is large enough to cause a memory issue is minimal. To calculate the correlation coefficients of the targets, we first construct the targets' co-variance matrix, $\sum_{i,j} = cov(Y_i, Y_j) = \mathbf{E}\left[(Y_i - \mu_i)(Y_j - \mu_j)\right]$, where $\mu_i = \mathbf{E}(Y_i)$, and $\mathbf{E}(Y_i)$ is the expected value of Y_i, $\forall i, j \in \{1, \ldots, m\}$. This matrix will show how the targets change together. We then calculate the correlation coefficients matrix $\rho_Y = corrcoef(Y) = \frac{cov(Y_i,Y_j)}{\sqrt{cov(Y_i,Y_i)cov(Y_j,Y_j)}}$, $\forall i, j \in \{1, \ldots, m\}$. The correlation coefficients matrix will describe the linear relationship among the target variables and we can then sort them in decreasing order, creating our maximum correlation chain. Having a single chain reduces the computational complexity, while providing a competitive model. For an unknown sample, the prediction method ~~is the same~~ **follows the same procedure** as Algorithm 3**, where the predicted value $\hat{y_j}^{(l)}$ is appended to the input**.

Algorithm 4 MT SVR with Max-Correlation Chain (SVRCC)

```
Input: Training dataset $\mathcal{D}$, number of cross-validation folds $k$
Output: Chained model $h_j, j = \{1, \ldots, m\}$
    Find maximum correlation chain for the target variables
 1: $correlation_{(m \times m)} = corrcoef(\boldsymbol{Y})$
 2: $sums_m = sum\,(correlation\,(l, j))\,, l = 1, \ldots, m$
    Create the maximum correlation chain $c$ of size $m$
 3: $c_m = sort\,(sums_m, \textbf{decreasing})$
 4: $\mathcal{D}^*_{c_1} = \{(x_1^{(1)}, y_{c_1}^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_{c_1}^{(\mathcal{N})})\}$
    Build a chained model, for max-chain $c_m$
 5: for $j = 1$ to $m$ do
 6:     $h_{c_j} : \mathcal{D}^*_{c_j} \to \mathbb{R}$
        Initialize appended dataset, $\mathcal{D}^*_{c_{(j+1)}}$
 7:     if $j < m$ then
 8:         $\mathcal{D}^*_{c_{j+1}} \leftarrow \emptyset$
            Split $\mathcal{D}^*_{c_j}$ into $k$ disjoint parts $\mathcal{D}^{*(i)}_{c_j}$, $i = 1, \ldots, k$ for training and testing
 9:         for $i = 1$ to $k$ do
10:             $h^{(i)}_{c_j} : \mathcal{D}^{*(k-i)}_{c_j} \to \mathbb{R}$
                Test the model on the test set $\mathcal{D}^{*(i)}_{c_j}$
11:             for $\boldsymbol{x}^{*(i)} \in \mathcal{D}^{*(i)}_{c_j}$ do
12:                 $\hat{y}^i_{c_j} = h^{(i)}_{c_j}(\boldsymbol{x}^{*(i)})$
                    Append $\boldsymbol{x}^{*(i)}$ with the true value $y^{(i)}_{c_j}$, and add it to set $\mathcal{D}^*_{c_{(j+1)}}$
13:                 $\mathcal{D}^*_{c_{(j+1)}} = \mathcal{D}^*_{c_{(j+1)}} \cup [\boldsymbol{x}^{*(i)}, y^{(i)}_{c_j}]$
14:             end for
15:         end for
16:     end if
17: end for
18: return $h_j, j = \{1, \ldots, m\}$
```

4. Experiments

This section presents the experimental comparison of the three models proposed, along with ~~six~~ **seven** others from state-of-the-art. It introduces the datasets, algorithms, and performance measures.

4.1. Datasets

This section presents a description of the datasets used in the experiments. Although there are many interesting applications of multi-target regression, there are not many publicly available datasets to use. The datasets used in the experimental study were collected from the Mulan website [44], as well as the UCI Machine Learning Repository [36]. Information on the 24 datasets used is summarized in Table 3, where the number of samples, attributes (dimensionality), and targets are shown. Note that the datasets used in this experiment are the same datasets used in [41] to properly perform our model comparisons, as well as ~~some~~ additional datasets **to ensure throrough investigation**.

- **Electrical Discharge Machining** The EDM dataset's purpose is to optimize the machining time by approximating the behavior of a human operator who controls the values of two variables, gap control and flow control.

- **Enb** The Energy and Buildings dataset is used to analyze the effect of eight input variables on two output variables, heating and cooling load, of residential buildings [43].

- **Jura** The Jura dataset is used to analyze [21].

- **Osales** The Online Product Sales dataset is used to identify patterns in the online sales of a consumer product based on a data set of product features [27].

- **Scpf** The See Click Predict Fix dataset is used to predict which *311* issues are most important to citizens [28].

- **Slump** The Slump dataset is used to predict 3 characteristics of the slump flow of high performance concrete [51].

- **Solar Flare 1 & Solar Flare 2** The Solar Flare dataset [44] is used for predicting how often 3 types of solar flare are observed in 24 hours. The three types of solar flare - common, moderate, and severe.

- **Water Quality** The Water Quality dataset [44] is used for approximating 14 target attributes that represent chemicals found in Slovenian rivers. 16 input attributes describing the biology of the river, the chemistry and physical water quality, are used to predict the target variables.

- **OES97 & OES10** The Occupational Employment Survey datasets were compiled during the years 1997 (OES97) and 2010 (OES10) [44]. There are 16 targets that are randomly selected from a set of employment categories.

- **ATP1d & ATP7d** The Airline Ticket Price datasets [44] are used to predict airline ticket prices. The 411 input variables include details about the different flights and the 6 targets are the minimum prices observed over the next 7 days for 6 airline preferences.

- **Andro** The Andromeda dataset contains data produced from the Andromeda Analyzer; a device that gathers water quality data in the Thermaikos Gulf of Thessaloniki, Greece [24].

- ~~**SCM1d** The Supply Chain Management dataset is used to predict how likely a product is to succeed. The input variables are the observed prices for a specific day, and 4 projected prices. Each of the 16 target variables correspond to the next day mean price for each of the products.~~ [44]

- **Wisconsin Cancer** Each sample represents follow-up data for one breast cancer patient, who is exhibiting invasive breast cancer, and no evidence of metastases at the time of diagnosis. This dataset can be used to predict whether the cancer is recurrent, and the time in which it might recur.

- **Stock** This dataset includes daily stock prices from January 1988 through October 1991 for 10 aerospace companies.

- **California Housing** This dataset includes information on block groups of individuals in California, from the 1990 Census. It contains $20,640$ samples with 7 continuous attributes, which are used to predict the 2 target variables, median house value and the median income.

- **Puma8NH & Puma32H** The Puma8H and Puma32H datasets are synthetically generated from a simulation of the dynamics of a Unimation Puma 560 robot arm [36], used to predict the angular acceleration of one robot arm link. The inputs, 8 for Puma8H and 32 for Puma32H, include angular positions, velocities, and torques of the robot arm.

- **Friedman** This is an artificial dataset where 25 attributes of each of the samples are generated independently from a uniform distribution over $[0, 1]$.

To measure the effects of non-related attributes, additional attributes are added to the datasets, which are independent from the output.

- **Polymer** The Polymer Test Plant dataset contains 10 input variables representing measurements of controlled variables, such as temperatures, feed rates, etc, in a polymer processing plant. The 4 target variables are the measurements of the output of that plant.

- **M5SPEC & MP5SPEC & MP6SPEC** These datasets consist of 80 samples of corn measured on 3 different NIR spectrometers, M5, MP5, and MP6.

4.2. Algorithms

This section presents the algorithms that we compare our proposals' performances to; namely *RC, ST, MTS, MTSC, ERC, ERCC,* and *MORF* which has been used in the experimental study conducted in [41]. Our contributions are compared to these algorithms because they have shown considerable performance in training multi-target models. The have also made their framework readily available for reproducing their results, allowing proper comparisons to be made with their methods. Moreover, note that all three SVR algorithms are implemented within the general framework of Mulan's MTRegressor[8] [44], which was built on top of Weka[9] [47], and we also used LIBSVM's Epsilon-SVR [10] implementation of Support Vector Regressors as the base SVR model.

In order to train a model that accurately describes a dataset and is not over-fit, one must experiment with different model hyper-parameters. In the case of SVMs and the SVM regression task, these parameters are the penalty parameter C, the Gaussian kernel parameter γ, and the error or tube parameter ϵ. We experimented with a range of parameters given in equations (3a) to (3c), referred to as (3), to ensure the final model is best representative of the dataset the model is being trained on.

$$C \in \{1, 10, 100\} \tag{3a}$$
$$\gamma \in \{1^{-9}, 1^{-7}, 1^{-5}, 1^{-3}, 1^{-1}, 1, 5, 10\} \tag{3b}$$
$$\epsilon \in \{0.01, 0.1, 0.2\} \tag{3c}$$

[8]http://mulan.sourceforge.net
[9]http://www.cs.waikato.ac.nz/ml/weka

Table 3: Multi-Target Regression datasets

Dataset	Samples ($\mathcal{N}$)	Attributes (d)	Targets (m)
EDM	145	16	2
Enb	768	8	2
Jura	359	15	3
Osales	639	413	12
Scpf	1137	23	3
Slump	103	7	3
Solar Flare 1	323	10	3
Solar Flare 2	1,066	10	3
Water Quality	1,060	16	14
OES97	323	263	16
OES10	403	298	16
ATP1d	201	411	6
ATP7d	188	411	6
Andro	49	30	6
Wisconsin Cancer	198	34	2
Stock	950	10	3
California Housing	20,640	7	2
Puma8NH	8,192	8	3
Puma32H	8,192	32	6
Friedman	500	25	6
Polymer	41	10	4
M5SPEC	80	700	3
MP5SPEC	80	700	3
MP6SPEC	80	700	3

We have compared the models trained on the different hyper-parameters using the cross-validation procedure which ensures that the models' performances are accurately assessed and the model built is not biased towards the full dataset.

To ensure having a controlled environment when conducting our performance comparisons, the experimental environment for running the competing algorithms was the same as what was done in [41]. This includes the following. The ST baseline model used was Bagging [7] of 100 regression trees [48]. The MTSC and ERCC methods are run using 10-fold cross-validation, and the ensemble size for the ERC and ERCC methods was set to 10. The ensemble size of 100 trees was used for MORF, and the rest of its parameters were set as recommended by [33].

The main preprocessing step used for the experiments was normalization. The input variables were scaled to have a mean value of 0 and standard deviation of 1. To ensure that each sample is definitely used when building the final model, we used sampling without replacement when creating our training and testing sets. The preprocessed and final datasets used in the experiments, along with the code for the algorithms presented in this paper, can be publicly found here:
`http://people.vcu.edu/~acano/MTR-SVRCC`

4.3. Performance Evaluation

The performance metric**s** used to analyze our contributions' performances ~~is~~ **are** shown in Equation**s** 4 to 7. For unseen or test datasets of size $\mathcal{N}_{test}$, the performances are evaluated by taking the ~~average, relative root mean square error (aRRMSE)~~ **the run time each algorithm takes to build a classifier, as well as the following metrics,** [5, 17] ~~given by,~~

- The average correlation coefficient (aCC):

$$\max \ aCC = \frac{1}{m}\sum_{j=1}^{m}\frac{\sum_{l=1}^{\mathcal{N}_{test}}(y_j^{(l)} - \bar{y}_j)(\hat{y}_j^{(l)} - \bar{\hat{y}}_j)}{\sqrt{\sum_{l=1}^{\mathcal{N}_{test}}(y_j^{(l)} - \bar{y}_j)^2 \sum_{l=1}^{\mathcal{N}_{test}}(\hat{y}_j^{(l)} - \bar{\hat{y}}_j)^2}} \tag{4}$$

- The mean squared error (MSE):

$$\min \ MSE = \frac{1}{m}\sum_{j=1}^{m}\frac{1}{\mathcal{N}_{test}}\sum_{l=1}^{\mathcal{N}_{test}}(y_j^{(l)} - \hat{y}_j^{(l)})^2 \tag{5}$$

- The average root mean squared error (aRMSE):

$$\min \ aRMSE = \frac{1}{m}\sum_{j=1}^{m}\sqrt{\frac{\sum_{l=1}^{\mathcal{N}_{test}}(y_j^{(l)} - \hat{y}_j^{(l)})^2}{\mathcal{N}_{test}}} \tag{6}$$

- The average relative root mean squared error (aRRMSE):

$$\min \ aRRMSE = \frac{1}{m}\sum_{j=1}^{m}\sqrt{\frac{\sum_{l=1}^{\mathcal{N}_{test}}(y_j^{(l)} - \hat{y}_j^{(l)})^2}{\sum_{l=1}^{\mathcal{N}_{test}}(y_j^{(l)} - \bar{y}_j)^2}} \tag{7}$$

where $\hat{\mathbf{y}}$ is the predicted output and $\bar{\mathbf{y}}$ is the average of the true output target variable. The test dataset is the hold-out set during cross validation. This ensures our model is evaluated on data that it has not been trained on, and thus unbiased towards the training datasets. **It also contributes to the generalizability and robustness of the model.**

5. Results

This section presents the results from the experimental study. **Tables** 4, 5, 6, 7, and 8 show~~s~~ the ~~aRRMSE~~ results **of the metrics, described in section** 4.3, achieved by our algorithm implementations, along with the ~~aRRMSE~~ results obtained by *RC, MORF, ST, MTS, MTSC, ERC,* and *ERCC*. **The optimal value for each metric is typeset in bold. The last two rows of each table correspond to the average metric value across all datasets, and the average rank of each**

algorithm calculated over the dataset metric averages, according to Friedman [18]. ~~The value with the lowest aRRMSE error is typeset in bold, and the last two rows of the table correspond to the average aRRMSE achieved across all datasets, as well as the average rank of each algorithm calculated over the dataset aRRMSE averages, according to Friedman~~ [18]. ~~The Friedman statistic, with alpha $= 0.05$, (distributed according to chi-square with 8 degrees of freedom) is 31.2560, with a p-value of $1.2600E^{-4}$. The results are sorted in order of each algorithm's decreasing average aRRMSE error.~~

- **Average Correlation Coefficient** - The Friedman statistic for the aCC results in Table 4, with alpha $= 0.05$, (distributed according to chi-square with 9 degrees of freedom) is 48.8432, with a p-value of $1.7764E^{-7}$. **add decription**

- **Mean Square Error** -The Friedman statistic for the MSE results in Table 5, with alpha $= 0.05$, (distributed according to chi-square with 9 degrees of freedom) is 48.0000, with a p-value of $2.5545E^{-7}$. **add decription**

- **Average Root Mean Square Error** - The Friedman statistic for the aRMSE results in Table 6, with alpha $= 0.05$, (distributed according to chi-square with 9 degrees of freedom) is 53.6796, with a p-value of $2.1753E^{-8}$. **add decription**

- **Average Relative Root Mean Square Error** - The Friedman statistic for the aRRMSE results in Table 7, with alpha $= 0.05$, (distributed according to chi-square with 9 degrees of freedom) is : 58.4796, with a p-value of $2.6598E^{-9}$. **add decription**

- **Time** - The Friedman statistic for the Time results in Table 8, with alpha $= 0.05$, (distributed according to chi-square with 9 degrees of freedom) is 159.1636, with a p-value of $7.6444E^{-11}$. **add decription** Table 8 contains two subtables, one that compares the time taken to build a model using *problem transformation* methods, and another that compares the results for *algorithm adaptation* methods. The results in these

As Table **??** shows, the SVR method's results are better than the competing algorithms. Even though a single-target model is being built for each of the target variables, it still out-ranked the competing *problem transformation* algorithms, such as the ERCC and MTSC algorithms. Having the single-target method, SVR, perform this well indicates that it would perform even better in an ensemble environment, where the correlations of the target variables are taken into account. This is shown by the increase in average performance and decrease in rank of SVRRC. The indicator that target variable correlation does play a crucial role in multi-target problems is made apparent when SVRRC performed better than building m single-target SVR models. Creating an ensemble model from the random chains of the target variables proved to capture additional invaluable information about the dataset that single-target models could not. For datasets Puma8NH and Mp5spec, the single-target SVR models performed better than SVRRC, which indicates that the targets possibly have minimal correlation with one another. Table **??** also shows that building an ensemble with random chains using the base SVR model performed better than the ERCC model, which uses regression trees as it's base model.

The overall improvement in performance of SVRRC over SVR shows that chaining the target variables to the input positively contributes to the training of the model. This implies that capturing the correlation between the target variables is valuable during the model's training process. Therefore, having a single chain representing the direction of maximum correlation between the targets should also boost the algorithm's performance. This is shown by the results in Table **??**, where the SVRCC algorithm outperforms the ensemble SVRRC, the single-target SVR, as well as all the competing methods.

Some of the datasets used in our experiments are considered to be sparse, such as the M5SPEC, MP5SPEC, MP6SPEC datasets, which only have 80 samples with 700 attributes, or the OES datasets where the number of samples and attributes is almost equal. These types of datasets are particularly difficult to learn from due to the lack of samples, as opposed to attributes describing them, as well as the risk of training a model that is over-fit. However, this type of machine learning problem does not affect the performance of support vector machines because the problem is scaled to the training set size rather than the attribute space dimension. Also, overfitting can be controlled with the SVM penalty parameter, C, which handles the trade-off of having a maximal margin and minimal error. This trait is shown by the results in Table **??**, where our SVR algorithms perform the best when learning from the sparse datasets. For both OES datasets, the SVRCC algorithm performs the best. For the M5SPEC, MP5SPEC, and MP6SPEC datasets, SVR and SVRRC performed the best, with SVRCC's results being second best. Also note, for these datasets, the performance differ-

Table 4: aCC Results & Algorithm Rank

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	0.6965	0.7062	0.7163	0.6977	0.6956	0.6977	0.7023	0.7245	0.7339	**0.7457**
Polymer	0.7305	0.7336	0.7371	0.7228	0.7015	0.7029	0.7222	0.7634	0.7857	**0.7905**
Andro	**0.7349**	0.6454	0.6793	0.6581	0.6915	0.6806	0.6653	0.6880	0.6951	0.7056
EDM	**0.6722**	0.6352	0.6412	0.6354	0.6355	0.6379	0.6354	0.6484	0.6565	0.6567
Solar Flare 1	0.1083	0.1258	0.1034	0.1193	**0.1492**	0.1387	0.1292	0.1066	0.0857	0.1152
Jura	0.7854	0.7907	0.7880	0.7882	0.7877	0.7884	0.7897	0.7789	0.7921	**0.7983**
Enb	0.9828	0.9832	0.9822	0.9829	0.9813	0.9823	0.9837	0.9858	0.9867	**0.9868**
Solar Flare 2	0.2357	0.2295	0.2375	0.2343	0.2302	0.2351	**0.2432**	0.1470	0.1648	0.1656
Wisconsin Cancer	0.3362	0.3587	**0.3652**	0.3588	0.3628	0.3609	0.3590	0.3187	0.3208	0.3373
California Housing	0.7705	0.7720	0.7149	0.7451	0.7007	0.7844	**0.8065**	0.7847	0.7949	0.8007
Stock	0.9785	0.9747	0.9755	0.9752	0.9753	0.9757	0.9763	0.9825	**0.9829**	0.9822
SCPF	0.5827	0.5508	0.5503	0.5477	0.5569	0.5656	0.5515	0.5891	**0.5975**	0.5946
Puma8NH	0.5424	0.4828	0.4942	0.4205	0.4677	0.4656	0.4650	**0.6041**	0.5975	0.6038
Friedman	0.1507	0.1609	0.1548	0.1667	0.1558	0.1608	0.1632	0.1710	0.1748	**0.1752**
Puma32H	0.3085	0.2934	0.2890	0.2504	0.2754	0.2870	0.2797	0.3358	0.3351	**0.3385**
Water Quality	**0.4303**	0.4063	0.4019	0.4051	0.3992	0.4052	0.4147	0.3545	0.3828	0.3857
M5SPEC	0.8161	0.8346	0.8134	0.8228	0.8333	0.8340	0.8308	0.9451	0.9452	**0.9472**
MP5SPEC	0.8315	0.8536	0.8244	0.8535	0.8524	0.8526	0.8542	0.9560	0.9602	**0.9633**
MP6SPEC	0.8317	0.8531	0.8231	0.8531	0.8507	0.8515	0.8541	0.9444	0.9500	**0.9528**
ATP7d	0.8260	0.8408	0.8422	**0.8474**	0.8273	0.8351	0.8464	0.8305	0.8407	0.8400
OES97	0.7829	0.7995	0.7990	0.8001	0.7986	0.7990	0.7999	0.8116	0.8134	**0.8137**
Osales	0.7186	0.6912	0.7104	0.7076	0.6357	0.7136	**0.7193**	0.6511	0.6433	0.6677
ATP1d	0.8961	0.9066	0.9051	0.9075	0.9048	0.9081	0.9071	0.9092	**0.9130**	0.9100
OES10	0.8708	0.8808	0.8805	0.8806	0.8804	0.8804	0.8809	0.8911	0.8924	**0.8963**
Average	0.6508	0.6462	0.6429	0.6409	0.6396	0.6476	0.6492	0.6634	0.6685	**0.6739**
Ranks	6.4167	5.8958	6.6042	6.4792	7.5208	5.8958	4.8542	4.7917	3.7083	**2.8333**

ence between our proposed algorithms against the others. The best performances achieved by the competing algorithms are at about 0.51, while our approaches reduce the error significantly. In regards to the sparse ATP1d and ATP7d datasets which also contain categorical attributes; the performance of algorithms that cannot process categorical data, depends highly on the preprocessing techniques used to transform these categorical variables into numeric ones. Regression trees are able to use categorical data, while support vector machines cannot. This is the most likely reason why our contributions did not perform as well as the competing algorithms on the Airline Ticket Price datasets. However, even with this disadvantage, our proposed methods surpass the others performances on 12 of the 24 datasets, and specifically, SVRCC performs the first and second best on 8

Table 5: MSE Results & Algorithm Rank

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	1.4388	1.4161	1.3667	1.4414	1.4602	1.4727	1.4183	1.2991	1.1726	**1.1614**
Polymer	1.6718	1.8120	1.5446	1.6726	1.8259	1.9999	1.6873	1.1874	1.1068	**1.0796**
Andro	1.4930	2.1467	1.4714	1.7525	2.2603	2.0812	1.8707	1.5406	1.2847	**1.2187**
EDM	**0.8342**	0.9373	0.9352	0.9418	0.9389	0.9326	0.9393	0.9092	0.8650	0.8817
Solar Flare 1	3.3458	3.1196	3.1193	3.0524	3.0357	3.0381	3.0594	**2.9912**	3.0176	3.0129
Jura	1.0973	1.0595	1.0732	1.0695	1.0744	1.0694	1.0632	1.1167	1.0435	**1.0315**
Enb	0.0381	0.0361	0.0407	0.0377	0.0452	0.0403	0.0343	0.0255	0.0216	**0.0214**
Solar Flare 2	2.9619	2.8532	**2.7732**	2.8282	2.8510	2.8273	2.8110	2.9518	2.9204	2.8713
Wisconsin Cancer	1.7666	1.7155	1.7156	1.7256	**1.7119**	1.7146	1.7195	1.8171	1.7915	1.7692
California Housing	0.8665	0.8221	0.9642	0.8673	1.0125	0.8952	0.7513	0.7477	0.6987	**0.6726**
Stock	0.0841	0.1039	0.0990	0.1008	0.0998	0.0987	0.0949	0.0578	0.0596	**0.0554**
SCPF	**2.2244**	2.3173	2.3661	2.3517	2.3923	2.3025	2.3295	2.2960	2.2510	2.3179
Puma8NH	1.9678	2.1133	2.0989	2.2024	2.1413	2.1473	2.1467	**1.8242**	1.8728	1.8299
Friedman	5.4573	5.3357	5.3478	5.3260	5.3482	5.3253	5.3210	5.3038	5.2942	**5.2812**
Puma32H	5.3419	**4.9499**	4.9627	5.0405	4.9905	4.9662	4.9805	5.2711	5.2749	5.1306
Water Quality	**11.3143**	11.5621	11.6276	11.5931	11.6495	11.6022	11.5004	12.2974	12.2042	12.0593
M5SPEC	1.0081	0.8754	1.0336	0.9421	0.8847	0.8824	0.8903	0.2578	0.2597	**0.2575**
MP5SPEC	1.1483	0.9817	1.1953	0.9970	0.9886	0.9880	0.9882	0.2261	**0.1979**	0.2136
MP6SPEC	1.1626	0.9928	1.1906	0.9992	1.0115	1.0045	0.9905	0.2926	**0.2903**	0.2954
ATP7d	1.7859	1.7348	**1.6435**	1.6460	1.8521	1.7888	1.6739	1.7820	1.7433	1.7098
OES97	4.6331	4.8340	4.8379	4.8082	4.8573	4.8591	4.8187	3.1440	3.0633	**3.0499**
Osales	7.3631	6.6850	**5.8848**	6.0850	7.8575	6.4746	5.9155	7.0727	7.3153	7.1374
ATP1d	1.0589	0.9056	0.9053	0.8982	0.9125	**0.8783**	0.9004	0.9091	0.8837	0.8922
OES10	3.6471	3.8931	3.8952	3.8909	3.9031	3.9063	3.8869	2.2623	2.1608	**2.1320**
Average	2.6546	2.6334	2.5872	2.5946	2.7127	2.6373	2.5747	2.3993	2.3664	**2.3368**
Ranks	6.5833	5.6667	6.0833	6.2500	7.8333	6.1250	5.1250	4.6667	3.6250	**3.0417**

datasets.

The results presented in this section show that the SVRCC method outperforms the rest of the models. It has a consistent lower aRRMSE average value compared to the competing methods. This is a strong indicator that if the targets are correlated, using a single maximum correlation chain to build a single chained model will benefit prediction accuracy. Another benefit of using the SVRCC method over the ensemble of random chains, is the time taken to train the model. Training is single chained model, with a known and calculated maximum correlation chain will outperform building an ensemble of random chains in the context of speed and accuracy. Moreover, the time taken to predict the output of an unknown instance would be greatly improved due to the fact that one SVR model is trained and tested, rather than an ensemble of 10 random chains.

Table 6: aRMSE Results & Algorithm Rank

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	0.6711	0.6652	0.6456	0.6699	0.6787	0.6793	0.6649	0.5561	0.5345	**0.5337**
Polymer	0.5277	0.5409	0.5042	0.5336	0.5536	0.5803	0.5319	0.4403	0.4062	**0.4060**
Andro	0.4649	0.5420	0.4414	0.4871	0.5390	0.5317	0.5039	0.4326	0.4061	**0.3989**
EDM	0.6372	0.6715	0.6705	0.6729	0.6722	0.6704	0.6721	0.6449	0.6411	**0.6366**
Solar Flare 1	0.9777	0.9274	0.9271	0.9089	0.8921	0.9016	0.9121	0.8856	0.8844	**0.8801**
Jura	0.5800	0.5686	0.5720	0.5706	0.5726	0.5712	0.5693	0.5794	0.5687	**0.5622**
Enb	0.1212	0.1166	0.1237	0.1214	0.1272	0.1253	0.1140	0.0981	0.0914	**0.0903**
Solar Flare 2	0.8725	0.8420	**0.8127**	0.8305	0.8313	0.8300	0.8304	0.8418	0.8349	0.8345
Wisconsin Cancer	0.9290	0.9163	0.9158	0.9187	**0.9153**	0.9160	0.9173	0.9422	0.9362	0.9306
California Housing	0.6541	0.6366	0.6889	0.6530	0.7053	0.6632	0.6079	0.6038	0.5859	**0.5755**
Stock	0.1643	0.1830	0.1774	0.1790	0.1790	0.1777	0.1739	0.1357	0.1329	**0.1308**
SCPF	0.7113	0.7235	0.7342	0.7255	0.7285	0.7143	0.7227	0.7155	0.7081	**0.7048**
Puma8NH	0.7855	0.8139	0.8114	0.8307	0.8196	0.8202	0.8203	**0.7650**	0.7740	0.7671
Friedman	0.9382	0.9203	0.9219	0.9199	0.9219	0.9197	0.9193	0.9203	0.9195	**0.9183**
Puma32H	0.9395	**0.8700**	0.8713	0.8778	0.8739	0.8716	0.8727	0.9353	0.9356	0.9331
Water Quality	**0.8921**	0.9015	0.9041	0.9025	0.9051	0.9030	0.8990	0.9284	0.9293	0.9271
M5SPEC	0.5707	0.5324	0.5761	0.5515	0.5347	0.5339	0.5376	0.2745	0.2744	**0.2740**
MP5SPEC	0.5315	0.4914	0.5426	0.4947	0.4930	0.4928	0.4928	0.2337	**0.2176**	0.2177
MP6SPEC	0.5344	0.4939	0.5416	0.4943	0.4982	0.4967	0.4927	0.2627	**0.2460**	0.2497
ATP7d	0.5216	0.4956	**0.4752**	0.4765	0.5194	0.5024	0.4824	0.5141	0.5066	0.5018
OES97	0.4652	0.4634	0.4635	0.4622	0.4643	0.4644	0.4627	0.3794	0.3768	**0.3749**
Osales	0.7190	0.6912	**0.6496**	0.6615	0.7591	0.6772	0.6515	0.7212	0.7343	0.7121
ATP1d	0.4053	0.3608	0.3587	0.3591	0.3653	0.3562	0.3596	0.3693	0.3638	**0.3507**
OES10	0.3954	0.3896	0.3897	0.3892	0.3901	0.3903	0.3889	0.3085	0.3039	**0.3038**
Average	0.6254	0.6149	0.6133	0.6121	0.6225	0.6162	0.6083	0.5620	0.5547	**0.5506**
Ranks	7.3333	5.7708	5.8125	6.0625	7.6250	6.0208	4.8542	5.0625	3.9167	**2.5417**

5.1. Statistical analysis

In order to compare the performances of the multiple models, non-parametric statistical tests are used to validate the experiments results obtained [13, 18].

To determine whether significant differences exist among the performance and results of the algorithms, the Iman-Davenport non-parametric test is run [19]. It is applied to rank the algorithms over the datasets used, according to the chi-squared distribution. The average ranks obtained by each method in the Friedman test are shown in Table **??**. The Iman-Davenport test, which follows the F-distribution, is conducted to find significant differences among algorithms. The Iman-Davenport statistic, distributed according to the F-distribution with 8 and 144 degrees of freedom is 4.66, with a p-value of $4.3730E^{-5}$. The p-value obtained by the Iman-

Table 7: aRRMSE Results & Algorithm Rank

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	0.6939	0.6886	0.6690	0.6938	0.7019	0.7022	0.6886	0.5765	**0.5545**	0.5560
Polymer	0.6159	0.5971	0.5778	0.6493	0.6270	0.6544	0.6131	0.5573	0.5253	**0.5116**
Andro	0.5097	0.5979	0.5155	0.5633	0.5924	0.5885	0.5666	0.4856	0.4651	**0.4455**
EDM	0.7337	0.7442	0.7413	0.7446	0.7449	0.7452	0.7443	0.7058	0.7070	**0.6978**
Solar Flare 1	1.3046	1.1357	1.1168	1.0758	0.9951	1.0457	1.0887	0.9917	0.9455	**0.9320**
Jura	0.5969	0.5874	0.5906	0.5892	0.5910	0.5896	0.5880	0.5952	**0.5764**	0.5885
Enb	0.1210	0.1165	0.1231	0.1211	0.1268	0.1250	0.1139	0.0977	0.0910	**0.0899**
Solar Flare 2	1.4167	1.1503	**0.9483**	1.0840	1.0092	1.0522	1.0928	1.0385	1.0253	1.0298
Wisconsin Cancer	0.9413	0.9314	0.9308	0.9336	**0.9305**	0.9313	0.9323	0.9555	0.9483	0.9427
California Housing	0.6611	0.6447	0.6974	0.6630	0.7131	0.6690	0.6146	0.6130	0.5945	**0.5852**
Stock	0.1653	0.1844	0.1787	0.1803	0.1802	0.1789	0.1752	0.1364	**0.1337**	0.1388
SCPF	0.8273	0.8348	0.8436	0.8308	0.8263	0.8105	0.8290	0.8164	0.8037	**0.8013**
Puma8NH	0.7858	0.8142	0.8118	0.8311	0.8199	0.8205	0.8207	**0.7655**	0.7744	0.7676
Friedman	0.9394	0.9214	0.9231	0.9210	0.9231	0.9209	0.9204	0.9218	0.9208	**0.9196**
Puma32H	0.9406	**0.8713**	0.8727	0.8791	0.8752	0.8729	0.8740	0.9364	0.9367	0.9319
Water Quality	**0.8994**	0.9085	0.9109	0.9093	0.9121	0.9097	0.9057	0.9343	0.9310	0.9045
M5SPEC	0.5910	0.5523	0.5974	0.5671	0.5552	0.5542	0.5558	0.2951	0.2935	**0.2925**
MP5SPEC	0.5522	0.5120	0.5683	0.5133	0.5145	0.5143	0.5119	0.2484	**0.2323**	0.2358
MP6SPEC	0.5553	0.5152	0.5686	0.5119	0.5198	0.5187	0.5109	0.2850	0.2669	**0.2623**
ATP7d	0.5563	0.5308	**0.5141**	0.5142	0.5558	0.5397	0.5182	0.5455	0.5371	0.5342
OES97	0.5490	0.5230	0.5229	0.5217	0.5239	0.5237	0.5222	0.4641	**0.4618**	0.4635
Osales	0.7596	0.7471	**0.7086**	0.7268	0.8318	0.7258	0.7101	0.7924	0.7924	0.7811
ATP1d	0.4173	0.3732	0.3733	0.3712	0.3790	**0.3696**	0.3721	0.3773	0.3707	0.3775
OES10	0.4518	0.4174	0.4176	0.4171	0.4178	0.4180	0.4166	0.3570	0.3555	**0.3538**
Average	0.6910	0.6625	0.6551	0.6589	0.6611	0.6575	0.6536	0.6039	0.5935	**0.5893**
Ranks	7.5000	5.7708	5.9375	6.1667	7.4375	6.3750	4.9792	4.7708	3.2708	**2.7917**

Davenport test is significantly less than 0.01, implying statistical confidence larger than 99%. Therefore, the test rejected the null-hypothesis and it can be said that there exist statistically significant differences between the aRRMSE results of the algorithms.

The Iman-Davenport test indicates statistically significant differences, then the Bonferroni-Dunn post-hoc test [16] is used to find these differences that occur between the algorithms. The test assumes that the two classifiers performances are significantly different if their average ranks differ by at least some critical value [20]. Table 9 shows the results of the Bonferroni-Dunn test on the aRRMSE rate with $\alpha = 0.05$, with the control algorithm being SVRCC. Figure 4 represents the values that are proportional to the mean rank obtained by each algorithm. The

Table 8: Time Results & Algorithm Rank

Table 9: Problem Transformation Methods

Datasets	MORF	ST	RC	SVR
Slump	38.1000	2.6000	1.8000	**0.6000**
Polymer	7.6000	2.7000	1.9000	**0.5000**
Andro	25.7000	4.4000	3.4000	**1.1000**
EDM	24.8000	2.8000	2.1000	**0.9000**
Solar Flare 1	34.1000	3.5000	2.7000	**2.3000**
Jura	64.3000	7.9000	6.4000	**4.7000**
Enb	71.4000	6.6000	**5.4000**	11.3000
Solar Flare 2	55.4000	7.4000	**6.3000**	9.4000
Wisconsin Cancer	51.4000	6.1000	4.9000	**2.0000**
California Housing	93.0000	9.7000	**8.2000**	15.8000
Stock	93.7000	11.7000	**11.0000**	18.5000
SCPF	66.3000	19.3000	**15.0000**	32.8000
Puma8NH	130.4000	29.7000	**27.9000**	94.1000
Friedman	79.5000	27.0000	25.0000	**12.3000**
Puma32H	93.9000	68.1000	87.7000	**32.2000**
Water Quality	108.4000	**93.1000**	127.2000	110.2000
M5SPEC	89.8000	68.9000	73.7000	**39.2000**
MP5SPEC	84.5000	94.6000	91.5000	**49.3000**
MP6SPEC	90.3000	93.4000	89.1000	**47.2000**
ATP7d	**70.5000**	262.6000	242.1000	80.0000
OES97	**83.4000**	485.3000	499.8000	148.2000
Osales	**92.0000**	1094.8000	986.5000	437.0000
ATP1d	**70.7000**	272.9000	261.9000	95.0000
OES10	**90.0000**	738.9000	688.5000	229.1000
Average	71.2000	142.2000	136.2000	**61.4000**
Ranks	5.5000	3.7100	3.0000	**1.8800**

Table 10: Algorithm Adaptation Methods

Datasets	MTS	MTSC	ERC	ERCC	SVRRC	SVRCC
Slump	9.9000	15.9000	11.1000	50.5000	1.9000	**0.7000**
Polymer	9.1000	15.5000	14.9000	80.5000	2.6000	**0.5000**
Andro	15.0000	34.2000	33.2000	197.9000	6.2000	**1.1000**
EDM	9.4000	18.1000	5.8000	19.0000	1.0000	**0.9000**
Solar Flare 1	13.6000	26.7000	17.7000	86.9000	9.3000	**2.6000**
Jura	31.8000	74.3000	43.5000	254.2000	18.7000	**5.3000**
Enb	26.1000	63.6000	**15.6000**	69.6000	17.7000	15.9000
Solar Flare 2	30.7000	68.0000	42.9000	241.5000	53.5000	**15.6000**
Wisconsin Cancer	21.9000	53.7000	14.8000	61.6000	2.4000	**2.0000**
California Housing	34.8000	75.9000	**21.3000**	102.0000	25.2000	23.6000
Stock	46.8000	96.7000	75.4000	427.3000	90.5000	**26.3000**
SCPF	65.9000	176.3000	104.2000	734.2000	162.8000	**48.8000**
Puma8NH	**106.7000**	288.6000	201.6000	1227.7000	516.6000	177.1000
Friedman	81.2000	258.3000	273.7000	2871.6000	322.3000	**18.8000**
Puma32H	181.0000	635.0000	667.9000	6087.0000	1018.7000	**53.1000**
Water Quality	262.1000	912.3000	925.4000	10993.3000	2567.9000	**189.5000**
M5SPEC	166.3000	604.6000	262.3000	3132.1000	546.7000	**45.1000**
MP5SPEC	221.2000	888.3000	557.0000	6864.1000	1132.1000	**58.4000**
MP6SPEC	212.6000	871.0000	557.6000	6761.3000	1227.1000	**58.5000**
ATP7d	452.1000	2319.8000	1779.2000	24373.8000	1897.4000	**136.5000**
OES97	1146.6000	4928.9000	5315.0000	58072.1000	3759.1000	**342.6000**
Osales	2340.7000	8322.2000	11361.2000	122265.3000	4830.1000	**843.6000**
ATP1d	476.5000	2568.9000	2138.9000	26768.9000	2127.8000	**174.4000**
OES10	1633.6000	6682.9000	7150.8000	83533.1000	5419.4000	**577.1000**
Average	316.5000	1250.0000	1316.3000	14803.2000	1073.2000	**117.4000**
Ranks	6.0000	8.2900	7.0800	9.9200	6.7100	**2.9200**

critical difference value for our control, 2.43, is represented in Figure 4 as the gray rectangle. The algorithms that are to the right of the critical difference rectangle and our control algorithm SVRCC, are the ones with significantly different results. Therefore, the algorithms beyond the critical difference are significantly worse. Table 9 shows the p-values obtained by applying the Bonferroni-Dunn procedure over the results obtained by the Friedman procedure, and it rejects those hypotheses that have an unadjusted p-value $\leq 6.2500E^{-3}$. From Table 9 and from Figure 4, it can be observed that algorithms MORF, MTS, MTSC, ERC, and ST perform significantly worse. The insignificance of the results for algorithms SVR

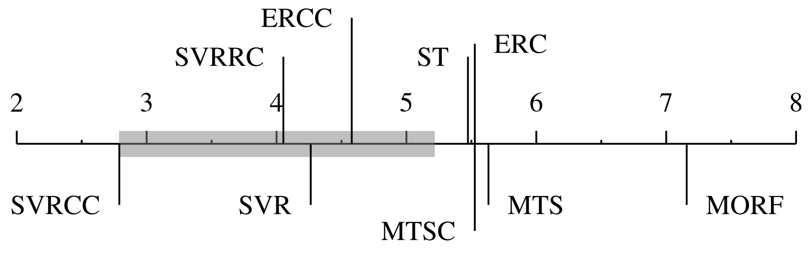

Figure 4: Bonferroni-Dunn test for aRRMSE

Table 11: Bonferroni-Dunn for aRRMSE

	$z = (R_0 - R_i)/SE$	$p - value$
MORF	4.9165	$1.0000E^{-6}$
MTS	3.1987	$1.3810E^{-3}$
MTSC	3.0802	$2.0690E^{-3}$
ERC	3.0802	$2.0690E^{-3}$
ST	3.0210	$2.5200E^{-3}$
ERCC	2.0140	$4.4011E^{-2}$
SVR	1.6586	$9.7201E^{-2}$
SVRRC	1.4216	$1.5513E^{-1}$

and SVRRC is due to the fact that they are algorithms with a similar base, which uses support vector regression. The same can be said for ERCC, where correlation chains have also been used, except that in SVRCC, a maximum chain is used. The results in Table 9, obtained by the post-hoc Bonferroni-Dunn test, show that our control algorithm, SVRCC, surpasses 5 out of 8 algorithms.

Table 10 shows the results of the Wilcoxon rank-sum test [46] for aRRMSE to compute multiple pairwise comparisons among the proposed algorithms and the other competing methods. MTS performs the best among the competing methods, but SVRCC outperforms it with a confidence of approximately 92.7160%. Also note that the results of the Wilcoxon test show that there are significant differences between SVRCC and ERCC's performance, where SVRCC succeeds a larger number of times than ERCC.

This section showed the results of the experimental study with respect to the performance of our SVRCC algorithm's performance against competing multi-target methods, using the aRRMSE measure. The proposed SVRCC is generally better than the other methods, where it has the lowest average aRRMSE rate, which is statistically significant against the competing methods. We could not say that the ERCC performed worse on all counts, but looking at the aRRMSE results, we can see that it never performed the best on any of the datasets, and as the Wilcoxon test showed, it performed significantly worse. Also, considering the randomness of the chains created, along with the limited number that can be tested on, there is no guarantee that it will perform consistently with different numbers of multi-target output variables.

Table 12: Wilcoxon Test for aRRMSE

SVRCC vs.	R^+	R^-	p-value
SVR	175	15	$5.2260E^{-4}$
SVRRC	154	36	$1.5972E^{-2}$
MORF	179	11	$2.0980E^{-4}$
ST	161	29	$6.1800E^{-3}$
MTS	140	50	$7.2840E^{-2}$
MTSC	161	29	$6.1800E^{-3}$
ERC	162	28	$5.3300E^{-3}$
ERCC	155	35	$1.4070E^{-2}$

6. Conclusion

The experiments performed in this study investigate the multi-target regression problem, where a model is trained using a multi-dimensional input space in order to predict a multi-dimensional output space. Despite the potential use of this type of learning, multi-target regression has been investigated **significantly** less than its classification counterpart, multi-label classification.

It has been shown that *algorithm adaptation* methods perform better than *problem transformation* methods due to their ability to capture relationships between target variables, which contributes positively to model training. ~~An example of an *algorithm adaptation* method would be building a regression model using ensembles of random chains of the target variables. This method however, is computationally complex, and might not capture the true correlation of the target variables.~~ **Both approaches tend to be comutationally complex, not only due to the multi-dimensional input space, but also the multi-dimensional output space.**

To address ~~this~~ **the MTR** problem, this investigation introduces three novel approaches, which use ~~a base~~ **the base-line** model of Support Vector Regression, and explore ~~various~~ **two** techniques ~~for solving multi-target regression problems, ensuring~~ **that ensure** the correlation of the target variables is captured. Their performances are then compared to 6 competitive *algorithm adaptation* and *problem transformation* methods.

The first method takes a *problem transformation* approach, which generates m single-target models, each trained independently of each other. This **base-line** approach works well, but does not take the possible correlations between the target variables into account ~~while training the m models~~ **during training**. ~~This lead to our second contribution~~ **To capture these correlations, our second contribution**, SVRRC, ~~which~~ creates **an** ensemble chained model based on 10 random generated orders of the target variables. ~~By using these differently ordered chains, a single ensemble model can be built based on the possible correlations of the target variables.~~ Due to this methods computational complexity, along with the fact that the~~, at most 10,~~ randomly generated chains may not represent the targets' correlations, we proposed our final method, SVRCC. This method eliminates the computational complexity ~~for~~ **of** creating **multiple** chained models, and it ~~also~~ ensures that the relationship between the target variables is fully exploited. This relationship is captured by creating a *maximum correlation chain*, where the targets are ordered in the direction of decreasing correlation. By doing this, each chained target attribute will positively contribute to the next ones prediction.

The experiments in this paper were all conducted using the Mulan framework, MTRegressor, as well as LIBSVM's Epsilon-SVR implementation of Support Vector Regressors. Firstly, they show the superior performance of using the SVR method as a base-line model, rather than regression trees **as used in MORF** ~~, for Multi-Target regression~~. ~~They also~~ **The results for SVRRC** show an increase in performance when **random** chaining is used to develop an ensemble model~~, SVRRC, using random chains~~. This ~~portrays~~ **indicates** the importance of the relationship among the target variables~~, when~~ **during** training ~~a regression model~~. Finally, the results show the superiority of using the SVRCC method, which has the lowest ~~average~~ aRRMSE value for performing 10-fold cross-validation on 19 datasets. ~~It~~ **SVRCC** performed better than the single-target SVR model and the randomly chained ensemble model SVRRC, ~~indicating~~ **showing** that the targets' maximum correlation **does positively contribute toward model training.** ~~contributed to SVRCC's performance.~~ In some cases, SVR performed better, which indicates that the targets have minimal correlation to one another. The ~~statistical~~ analysis of these results show the statistical significance of the error rates obtained by our experiments. They showed that statistically significant differences existed between 5 out of 9 algorithms, where the other 4 were our proposed methods along with the ERCC proposed by *Spyromitros et. al.*. The differences were minimal for these 4 due to their similar nature; our proposed methods all used support vector regression and the ERCC method used and ensemble of random chains. This, however, does not dismiss the findings for SVRCC. It's competitive accuracy, as well as speed (due to using a single chained model), show that it is a powerful learning algorithm for multi-target problems.

Acknowledgment

This research was supported by the Spanish Ministry of Economy and Competitiveness, project TIN2014-55252-P, and by FEDER funds.

References

[1] T. Aho, B. Zenko, S. Dzeroski, T. Elomaa, Multi-Target Regression with Rule Ensembles, Journal of Machine Learning Research 13 (2012) 2267–2407.

[2] A. Appice, S. Dzeroski, Stepwise induction of multi-target model trees, European Conference of Machine Learning Lecture Notes on Artificial Intelligence 4701 (2007) 502–509.

[3] J. Baxter, A bayesian/information theoretic model of learning to learn via multiple task sampling, Machine Learning 28 (1997) 7–39.

[4] S. Ben-David, R. Schuller, Exploiting task relatedness for multiple task learning, In: Proceedings of the Sixteenth Annual Conference on Learning Theory (2003) 567–580.

[5] H. Borchani, G. Varando, C. Bielza, P. Larrañaga, A survey on multi-output regression, WIREs Data Mining Knowledge Discovery 5 (2015) 216–233.

[6] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.

[7] L. Breiman, Bagging predictors, Machine Learning 24 (1996) 123–140.

[8] A. Cano, J. Luna, E. Gibaja, S. Ventura, LAIM discretization for multi-label data, Information Sciences 330 (2016) 370–384.

[9] R. Caruana, Multitask learning, Machine Learning 28 (1997) 41–75.

[10] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[11] F. Charte, A. Rivera, M. Del Jesus, F. Herrera, LI-MLC: A label inference methodology for addressing high dimensionality in the label space for multilabel classification, IEEE Transactions on Neural Networks and Learning Systems 25 (2014) 1842–1854.

[12] C. Cortes, V. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.

[13] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (2011) 3–18.

[14] Y. Ding, L. Cheng, W. Pedrycz, K. Hao, Global nonlinear kernel prediction for large data set with a particle swarm-optimized interval support vector regression, IEEE Transactions on Neural Networks and Learning Systems 26 (2015) 2521–2534.

[15] H. Drucker, C. Burges, L. Kaufman, A. Smola, V. Vapnik, Support vector regression machines, In: Proceedings of the Advances in Neural Information Processing Systems (1997) 155–161.

[16] O. Dunn, Multiple comparisons among means, J. Amer. Stat. Assoc. 56 (Mar. 1961) 52–64.

[17] J. Friedman, T. Hastie, R. Tibshirani, Regularization paths for generalized linear models via coordinate descent, Journal of Statistical Software 33 (2010) 1–22.

[18] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Information Sciences 180 (2010) 2044–2064.

[19] S. García, F. Herrera, An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons, Journal of Machine Learning Research 9 (2008) 2677–2694.

[20] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of nonparametric tests for analyzing the evolutionary algorithms' behaviour - a

case study on the CEC2005 Special Session on Real Parameter Optimization, Heuristics 15 (2008) 617–644.

[21] P. Goovaerts, Geostatistics for natural resources evaluation, Oxford University Press, 1997.

[22] E. Hadavandi, J. Shahrabi, S. Shamishirband, A novel Boosted-neural network ensemble for modeling multi-target regression problems, Engineering Applications of Artificial Intelligence 45 (2015) 204–219.

[23] E. Hadavandi, K. Shahrabi, Y. Hayashi, SPMoE: a novel subspace-projected mixture of experts model for multi-target regression problems, Soft Computing 20 (2016) 2047–2065.

[24] E. Hatzikos, G. Tsoumakas, G. Tzanis, N. Bassiliades, I. Vlahavas, An empirical study on sea water quality prediction, Knowledge-Based Systems 21 (2008) 471–478.

[25] J. He, H. Gu, Z. Wang, Multi-instance multi-label learning based on gaussian process with application to visual mobile robot navigation, Information Sciences 190 (2011) 162–177.

[26] M. Jeong, G. Lee, Multi-domain spoken language understanding with transfer learning, Speech Communication 51 (2009) 412–424.

[27] Kaggle, Online product sales, 2012. URL: `https://www.kaggle.com/c/online-sales`.

[28] Kaggle, See click predict fix, 2013. URL: `https://www.kaggle.com/c/see-click-predict-fix`.

[29] V. Kecman, Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models, MIT Press, 2001.

[30] D. Kocev, M. Ceci, Ensembles of Extremely Randomized Trees for Multi-target Regression, Lecture Notes in Computer Science 9356 (2015) 86–100.

[31] D. Kocev, S. Dzeroski, M. White, G. Newell, P. Griffioen, Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition, Ecological Modelling 20 (2009) 1159–1168.

[32] D. Kocev, C. Vens, J. Struyf, S. Dzeroski, Ensembles of Multi-Objective Decision Trees, Springer, Heidelberg (2007) 86–100.

[33] D. Kocev, C. Vens, J. Struyf, S. Dzeroski, Tree ensembles for predicting structured outputs, Pattern Recognition 43 (2013) 817–833.

[34] J. Lee, D.-W. Kim, Memetic feature selection algorithm for multi-label classification, Information Sciences 293 (2015) 80–96.

[35] H. Li, D. Li, Y. Zhai, S. Wang, J. Zhang, A novel attribute reduction approach for multi-label data based on rough set theory, Information Sciences 367-368 (2016) 827–847.

[36] M. Lichman, UCI machine learning repository, 2013. URL: `http://archive.ics.uci.edu/ml`.

[37] Q. Liu, Q. Xu, V. Zheng, H. Xue, Z. Cao, Q. Yang, Multi-task learning for cross-platform sirna efficacy prediction: an in-silico study, BMC Bioinformatics 11 (2010) 181–196.

[38] F. Pérez, G. Camps, E. Soria, J. Pérez, A. Figueiras, A. Artés, Multidimensional function approximation and regression estimation, Artificial Neural Networks (2002) 757–762.

[39] B. Qian, X. Wang, J. Ye, I. Davidson, A reconstruction error based framework for multi-label and multi-view learning, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 594–607.

[40] J. Read, C. Bielza, P. Larranaga, Multi-dimensional classification with super-classes, IEEE Transactions on Knowledge and Data Engineering 26 (2014) 1720–1733.

[41] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, I. Vlahavas, Multi-Label Classification Methods for Multi-Target Regression, Cornell University Library (2014).

[42] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, I. Vlahavas, Multi-target regression via input space expansion: Treating targets as inputs, Machine Learning 104 (2016) 55–98.

[43] A. Tsanas, A. Xifara, Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools, Energy and Buildings 49 (2012) 560–567.

[44] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, I. Vlahavas, Mulan: A java library for multi-label learning, Journal of Machine Learning Research 12 (2011) 2411–2414.

[45] G. Tsoumakas, E. Spyromitros-Xioufis, A. Vrekou, I. Vlahavas, Multi-target regression via random linear target combinations, Machine Learning and Knowledge Discovery in Databases 8726 (2014) 225–240.

[46] F. Wilcoxon, Individual comparisons by ranking methods, Biometr. Bull. 1 (Dec. 1945) 80–83.

[47] I. Witten, E. Frank, M. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd edition ed., Morgan Kaufmann, 2011.

[48] Q. Wu, Y. Ye, H. Zhang, T. Chow, S.-S. Ho, ML-TREE: A tree-structure-based approach to multilabel learning, IEEE Transactions on Neural Networks and Learning Systems 26 (2015) 430–443.

[49] T. Xiong, Y. Bao, Z. Hu, Multiple-output support vector regression with a firefly algorithm for interval-valued stock price index forecasting, Knowledge-Based Systems 55 (2014) 87–100.

[50] S. Xu, X. An, X. Qiao, L. Zhu, L. Li, Multi-output least-squares support vector regression machines, Pattern Recognition 34 (2013) 1078–1084.

[51] I. Yeh, Modeling slump flow of concrete using second-order regressions and artificial neural networks, Cement and Concrete Composites 29 (2007) 474–480.

[52] M.-L. Zhang, Z.-H. Zhou, A review on multi-label learning algorithms, IEEE Transactions on Knowledge and Data Engineering 26 (2014) 1819–1837.

[53] Y.-P. Zhao, K.-K. Wang, F. Li, A pruning method of refining recursive reduced least squares support vector regression, Information Sciences 296 (2015) 160–174.