A COMPUTATIONALLY EFFICIENT HEURISTIC ALGORITHM FOR PIECEWISE LINEAR REGRESSION

PARÇALI DOĞRUSAL REGRESYON İÇİN HESAPSAL VERİMLİLİĞİ YÜKSEK BİR SEZGİSEL ALGORİTMA

KÜBRA DOĞAN

ASSOC. PROF. DR. BURKAY GENÇ
Supervisor

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Engineering

January 2023

ABSTRACT

A COMPUTATIONALLY EFFICIENT HEURISTIC ALGORITHM FOR PIECEWISE LINEAR REGRESSION

Kübra Doğan
Master of Science, Computer Engineering
Supervisor: Assoc. Prof. Dr. Burkay Genç
January 2023, 86 pages

Regression analysis is a method used to model the relationship between the dependent variables and the independent variable in the data, and to predict the dependent variable from the independent variables in the future data. Piecewise linear regression (PLR) is a powerful approach used in regression analysis. PLR models the data with multiple linear regression functions. Thus, it can model non-linear data as well as retain the interpretability of linear regression. Interpretability has recently become a hot topic for machine learning. In application areas such as finance and health, it is important for a model not only provide a good prediction, but also be interpretable and/or verifiable by domain experts. In this respect, we think PLR is a promising approach.

In this study, we define PLR problem as in which data is partitioned into intervals on a predetermined dimension and each interval is represented by a unique multivariate linear regression. We offer a method by adopting heuristic approaches and aim for a solution with practically acceptable computational efficiency even in large data sets. The proposed method is compared with Decision Tree, Random Forest, XGBoost, XGBoost with Random Forest learners, Support Vector Machines, K-Nearest Neighbors, Artifical Neural Network and Multivariate Adaptive Regression Splines algorithms. Several synthetic and real-world datasets containing moderate and large number of observations are used in the experiments. Synthetic data are particularly targeted by the proposed approach because they are generated in a way to include structural shifts. The results reveal that our method remains competitive with the well-known machine learning algorithms and outperforms especially in the synthetic dataset instances. Our method is also compared with a mathematical programming-based heuristic method and it is clearly observed that the proposed method provides better scores. When we examine at the computational efficiency results of the proposed method, we observe that even the largest datasets (containing 100000 observations) have computation times expressed in milliseconds. Overall, the results show that PLR can be an effective method, especially when considered the interpretability property it holds.

Keywords: Piecewise linear regression, Heuristic algorithm, Interpretability, Machine learning, Non-linear data

ÖZET

PARÇALI DOĞRUSAL REGRESYON İÇİN HESAPSAL VERİMLİLİĞİ YÜKSEK BİR SEZGİSEL ALGORİTMA

Kübra Doğan
Yüksek Lisans, Bilgisayar Mühendisliği
Danışman: Assoc. Prof. Dr. Burkay Genç
Eylül 2021, 86 sayfa

Regresyon analizi verideki bağımlı değişkenler ile bağımsız değişken arasındaki ilişkiyi modellemek ve gelecek veride bağımsız değişkenlerden bağımlı değişkeni tahminlemek için kullanılan bir yöntemdir. Parçalı doğrusal regresyon (PDR) ise regresyon analizinde kullanılan güçlü bir yaklaşımdır. PDR veriyi birden fazla doğrusal regresyon fonksiyonu ile modeller. Böylece, doğrusal regresyonun yorumlanabilirlik özelliğini barındırmakla birlikte doğrusal olmayan verileri de modelleyebilir. Yorumlanabilirlik özelliği son zamanlarda makine öğrenmesi için revaçta olan bir konu haline gelmiştir. Finans ve sağlık gibi uygulama alanlarında bir modelin sadece iyi bir tahminde bulunması değil, aynı zamanda yorumlanabilir ve/veya konu uzmanları tarafından doğrulanabilir olması önem taşımaktadır. Bu açıdan PDR'nin umut verici bir yaklaşım olduğunu düşünüyoruz.

Bu çalışmada, PDR'yi, verinin önceden belirlenmiş bir boyutta aralıklara bölündüğü ve her aralığın benzersiz bir çok değişkenli doğrusal regresyon ile ifade edildiği bir problem olarak tanımlıyoruz. Büyük veri setlerinde dahi pratik olarak kabul edilebilir seviyede hesapsal verimliliğe sahip bir çözüm hedefleyerek sezgisel bir yaklaşım kullanan bir çözüm yöntemi sunuyoruz. Önerilen yöntem Decision Tree, Random Forest, XGBoost, Random

Forest öğrenicileri kullanan XGBoost, Support Vector Machines, K-Nearest Neighbors, Artifical Neural Network ve Multivariate Adaptive Regression Splines algoritmalarıyla karşılaştırılmıştır. Deneylerde orta ve büyük ölçekte gözleme sahip birçok sentetik ve gerçek veri seti kullanılmıştır. Sentetik veriler, yapısal kaymaları içerecek şekilde üretildikleri için önerilen yaklaşım tarafından özellikle hedeflenmektedir. Sonuçlar yöntemimizin iyi bilinen makine öğrenmesi algoritmalarıyla rekabetçi kaldığını ve özellikle de sentetik verilerde daha iyi performans gösterdiğini ortaya koymaktadır. Yöntemimiz ayrıca matematiksel programlama tabanlı sezgisel bir yöntem ile de karşılaştırılmış ve daha iyi sonuç gösterdiği gözlemlenmiştir. Önerilen yöntemin hesapsal verimlilik sonuçlarına baktığımızda ise en büyük (100000 gözlem içeren) veri setlerinde dahi milisaniyelerle ifade edilen hesaplama sürelerine sahip olduğunu görüyoruz. Genel olarak sonuçlar özellikle yorumlanabilirlik özelliği dikkate alındığında PDR'nin etkili bir yöntem olabileceğini göstermektedir.

Anahtar Kelimeler: Parçalı doğrusal regresyon, Sezgisel algoritma, Yorumlanabilirlik, Makine öğrenmesi, Doğrusal olmayan veri

ACKNOWLEDGEMENTS

I would like to express my most sincere gratitude to my advisor Assoc. Prof. Dr. Burkay Genç for his guidance, understanding, and support. Thank you for your invaluable comments and recommendations. I feel lucky for having the opportunity to work with you and learn from you.

I would also like to thank Assoc. Prof. Dr. Hüseyin Tunç for his generously provided knowledge and expertise. I am very grateful for your precious support. The conversations we had with you and Dr. Burkay Genç taught me to think and look from an academic point of view.

I thank the chair of my thesis committee, Asst. Prof. Dr. Cemil Zalluhoğlu, for his valuable suggestions during my thesis defense presentation.

I thank all my colleagues at TÜBİTAK SAGE who contributed to my work with their valuable suggestions.

I would like to thank my lovely family: my mother, my father, and my twin sister. They always believed in me more than myself. I could not move forward without their unconditional and endless love. I am grateful to my dear twin sister for her understanding and great support throughout my thesis and master's programs. I am deeply indebted to her. I am grateful to my beloved mother who always listens to me patiently and motivates me when I am in low spirits. I am grateful to my dear father who always believes in me and supports me.

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

PLR	:	**P**iecewise **L**inear **R**egression
MaSH	:	**M**erge **a**nd **S**plit **H**euristic
MILP	:	**M**ixed **I**nteger **L**inear **P**rogramming
CG	:	**C**olumn **G**eneration
ANN	:	**A**rtifical **N**eural **N**etwok
KNN	:	**K**-**N**earest **N**eighbors
MARS	:	**M**ultivariate **A**daptive **R**egression **S**plines
XGB	:	**XGB**oost
XGBRF	:	**XGB**oost with **R**andom **F**orest learners
DT	:	**D**ecision **T**ree
RF	:	**R**andom **F**orest
SVM	:	**S**upport **V**ector **M**achine
SSE	:	**S**um of **S**quared **E**rror
MSE	:	**M**ean **S**quared **E**rror
RMSE	:	**R**oot **M**ean **S**quared **E**rror
MAE	:	**M**ean **A**bsolute **E**rror
MAPE	:	**M**ean **A**bsolute **P**ercentage **E**rror
TAE	:	**T**otal **A**bsolute **E**rror
MLE	:	**M**aximum **L**ikelihood **E**stimation
DP	:	**D**ynamic **P**rogramming

1. INTRODUCTION

With the widespread use of software applications, data is generated from various application areas. Inferring from data is valuable in today's data driven world, and regression analysis comes as a powerful instrument. Regression analysis is used to understand the relationship between a set of independent variables and a dependent variable. It provides an understanding about how input variables affect the output variable and represent this relationship through a mathematical function. The mathematical function is computed from data by minimizing a predefined error metric and used to predict the output of out-of-sample data.

There are many methodologies to conduct a regression analysis. Linear regression is arguably the simplest method among common regression methods. It fits a linear function to data and gives interpretable outputs about changes in the data. However, it implies a linearity assumption, i.e. it assumes that changes in the data can be modeled by a single linear function. For most of the real-world cases, this assumption does not hold and linear regression fails to adequately represent such data. Piecewise linear regression (PLR) comes to the play for this reason. It takes the power of interpretability of linear regression, while allowing non-linearities in data.

In a piecewise linear regression model, data is grouped into different segments and each segment is fit with a unique linear function. The main idea of PLR is that each partition consists of a good subsample of the data such that it can be modeled accurately with a single linear function.

In literature, PLR is a well studied area. Though, the studies do not agree on a common definition for piecewise linear regression. In the literature, it can be referred to as piecewise, segmented, broken-line [1] or broken-stick regressions [2]. Broken-line or broken-stick regressions imply continuity requirement between regions, whereas segmented regression evokes mutually exclusive nature of data in each region. Indeed, some PLR studies in the literature imply a continuity requirement of their linear regression functions, i.e. start point of a line should be intersected by the end point of the preceding line. Other studies do not impose a continuity requirement; it is sufficient to fit a linear function to data in the region that best describes it.

PLR is simple in idea and explanation, but there are a number of things to determine for an accurate PLR model: which feature or features should be used for partitioning a multivariate dataset, how many pieces should we split, what are the breakpoints that define a partition and what are the optimal parameters for linear functions in each partition. Finding these parameters is not a trivial task. Studies in literature make restrictions on their problem definition, or benefit from some heuristic approaches to speed up the process.

In addition to restricting the PLR problem definition, studies in the literature can make assumptions or predefinitions about the parameters of the PLR problem. As far as we observe, the number of intervals in the resulting PLR model is a common predefined parameter in the studies, only the breakpoint locations need to be solved. This requires an external sight of the data. Therefore, an automatic process for finding the number of intervals is necessary.

1.1. Scope Of The Thesis

This thesis mainly focuses on solving PLR problem for medium to large-sized datasets by aiming for a computationally efficient solution. In this study, we consider a PLR problem where multivariate data is partitioned into an unknown number of regions using a predefined partition feature. A heuristic algorithm is developed to solve PLR problem in a computationally efficient way for larger datasets while maintaining the predictive accuracy within acceptable limits. We compare our algorithm to some popular machine learning algorithms including decision tree, random forest, XGBoost, XGBoost with random forest learners, support vector machine, k-nearest neighbors, artificial neural network, and multivariate adaptive regression splines. We also benchmark our algorithm against a PLR problem solution presented in [3]. We experiment using real-world and synthetic datasets and provide computational results of each algorithm. Compared to the machine learning algorithms, our algorithm remains competitive in terms of prediction performance and outperforms most of them in terms of time efficiency. Compared to the column generation algorithm proposed in [3], our algorithm outperforms it both in modeling accuracy and time efficiency.

1.2. Contributions

In this research, we study PLR problem by aiming to enable the utilization of PLR extensively in regression analysis tasks. The main contributions of this study can be enumerated as follows:

- We propose a heuristic solution procedure that achieves good prediction scores in short execution times.
- Unlike similar studies in the literature, the number of linear functions is not a predefined parameter in our work and is found in the developed heuristic.
- We experiment on medium to large-sized datasets and compare the effectiveness of the proposed algorithm.
- The proposed algorithm provides a white-box solution, unlike most machine learning algorithms.
- Our heuristic solution can be used as an acceleration tool for providing an initial PLR solution to be used in other algorithms.

1.3. Organization

The organization of the thesis is as follows:

- Chapter 1 presents our motivation, contributions, and the scope of the thesis.
- Chapter 2 provides background information aligned with the scope of the thesis.

- Chapter 3 summarizes related work from the literature.
- Chapter 4 presents the proposed MaSH method and gives the details.
- Chapter 5 demonstrates the numerical work conducted to assess the performance of the proposed method and analyzes the results.
- Chapter 6 states the summary of the thesis and provides possible research directions.

2. BACKGROUND OVERVIEW

2.1. Piecewise Linear Regression

Linear regression models the relationship between independent variables and a dependent continuous variable by a linear function. Let n be the number of observations and j be the number of independent variables. x_i^j shows the j^{th} feature of the i^{th} observation and y_i is the dependent variable of the i^{th} observation. The mathematical representation of linear regression is as follows:

$$y_i = \beta_1 x_i^1 + \beta_2 x_i^2 + \ldots + \beta_j x_i^j + \alpha,\ i = \{1, ..., n\} \tag{1}$$

where α is the intercept and β values are the regression coefficients.

Linear regression is powerful in terms of producing explainable outputs about the changes of the predicted value. However, its main disadvantage lies in its linearity assumption on data. Many real-world data are not linear in nature and linear regression cannot adequately represent such data. In this case, piecewise linear regression (PLR) becomes a great option to represent non-linear data while preserving the explainability feature of the single linear regression.

PLR is a well-studied area in literature, although it is not a commonly used algorithm in practice. The main idea of PLR is that the data is partitioned into different regions and each region is modeled with a unique linear function. The underlying logic is that the subsample of data in each region can be accurately modeled with a single linear function. PLR has such a simple logic, but finding an optimal PLR model is a difficult task. Firstly, it is required to determine the feature or features to partition the data. In this study, we use a single predefined partition feature.

$$y_i = \begin{cases} \beta_1^{(1)} x_i^1 + \beta_2^{(1)} x_i^2 + \ldots + \beta_j^{(1)} x_i^j + \alpha^{(1)}, & b_1 \leq x_i^z \leq b_2 \\ \beta_1^{(2)} x_i^1 + \beta_2^{(2)} x_i^2 + \ldots + \beta_j^{(2)} x_i^j + \alpha^{(2)}, & b_2 < x_i^z \leq b_3 \\ \vdots & \vdots \\ \beta_1^{(K)} x_i^1 + \beta_2^{(K)} x_i^2 + \ldots + \beta_j^{(K)} x_i^j + \alpha^{(K)}, & b_K < x_i^z \leq b_n \end{cases} \tag{2}$$

Equation 2 shows the mathematical representation of a PLR model where the data is partitioned on z^{th} feature. $\alpha^{(k)}$ are intercepts and $\beta_z^{(k)}$ values are regression coefficients of the corresponding linear function in the k^{th} region, where $k = \{1, 2, \ldots, K\}$. b_k represents the x^z location of breakpoint k and $b_1 < b_2 < \cdots < b_K < b_n$. The first breakpoint b_1 is the first value and the last breakpoint b_n is the last value along the z^{th} axis.

In Equation 2, K represents the number of regions in the corresponding PLR model with $(K - 1)$ breakpoints. Deriving the number of breakpoints/regions is a difficult problem. In the literature, most of the studies use a predefined K value in their PLR problem definition (e.g. [3–6]). In this thesis, we don't define the number of regions a priori. Instead, we define an error metric to determine whether a change in the existing intervals will provide a gain in terms of overall error of the PLR model. This error metric will be introduced in Section 4.

Another parameter that construct a PLR model is the location of the breakpoints. Breakpoints define the region boundaries. Different approaches are developed in the literature to discover the optimal breakpoint locations including dynamic programming and mathematical programming models. These methodologies attempt to find an optimal PLR model, yet they lack time efficiency. Hence, it is impractical to use them for larger datasets. In the literature, heuristic algorithms are developed to reach a sufficiently good solution faster (e.g. [7, 8]). Additionally, studies can benefit from some heuristic approaches accompanying with their algorithm to accelerate reaching a solution (see [3]).

In the literature, there are two different variations of the PLR problem in terms of continuity requirement: continuous and discontinuous. In a continuous PLR model, the start point and endpoint of the adjacent regions must be intersected. In a mathematical representation,

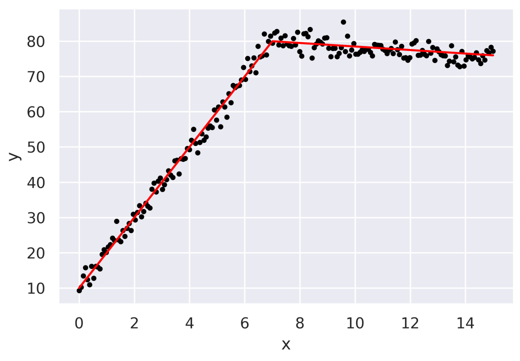


Figure 2.1 Continuous PLR model

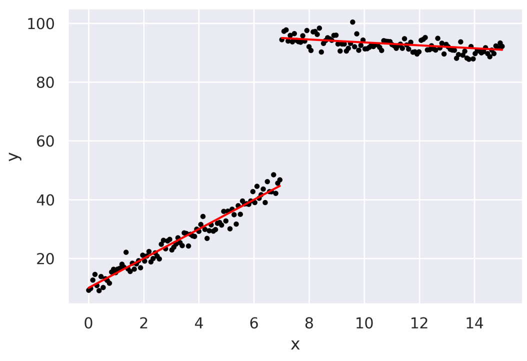


Figure 2.2 Discontinuous PLR model

Figure 2.3 Continuous and discontinuous PLR models

$$\beta_1^{(k)} x_i^1 + \ldots + \beta_j^{(k)} x_i^j + \alpha^{(k)} = \beta_1^{(k+1)} x_i^1 + \ldots + \beta_j^{(k+1)} x_i^j + \alpha^{(k+1)} \tag{3}$$

if Equation 3 holds, where x_i is a breakpoint, we refer it to as continuous PLR model. Conversely, we refer as discontinuous PLR model where Equation 3 does not hold. Figure 2.3 shows the difference between continuous and discontinuous PLR models.

In this thesis, we do not imply a continuity requirement and consider a discontinuous PLR problem.

2.2. Heuristic Algorithm

Heuristics are shortcuts that humans naturally use to make decisions in daily life. Examples include trial and error, educated guess, and common sense. In computer science, heuristics refers to a solution approach that is designed to produce fast solutions to a specific problem. A heuristic algorithm is developed when the classical methods are too slow in solving the problem in practical times, or an exact solution does not exist. Heuristic algorithms produce near-optimal solutions that are good enough for the problem in question, and not guarantee the optimal or perfect solution. The purpose of developing a heuristic algorithm is often to solve the problem at hand in reasonable times.

2.3. Interpretability

Recently, interpretability has emerged as an important concept in machine learning. Many machine learning algorithms that are popular today are black-box models. For example, we can mention deep neural networks, artificial neural networks, and ensemble algorithms. These algorithms can learn complex relationships in data and can make good predictions. However, they cannot answer the question of "why" they made this prediction. In some problems, it is not enough to make correct predictions, further, it is important to find an answer to the question of "why" Because as humans, we want to know the reason that made a decision, and when we don't know the reason, it becomes difficult to trust. Application areas such as healthcare and self-driving cars are examples of sectors where trust is important [9]. Besides, we evaluate the prediction performance of machine learning algorithms with some metrics. This is an incomplete assessment when we consider the real world [10].

Additionally, Kim et. al. [11] mention learning to criticize in their work. We feed machine learning algorithms with data. Therefore, these algorithms generate a model from the data we provide and use this model to predict. We usually pre-process the data before feeding it to the algorithm and provide clean data with the aim of increasing the generalization ability. Thus, we draw conclusions from prototype data, but there is a possibility of eliminating meaningful data. Kim et. al. [11] tell us that in order to retain interpretability, we should use also such data along with prototypes.

Interpretable machine learning concept is important for various application areas, such as finance [12–16], scientific discovery [17], intelligent decision systems [18], and healthcare [19–23].

The piecewise linear regression model presented in this study is also an interpretable model. In addition, linear regression and decision tree can be given as examples for interpretable models. Linear regression and decision tree are superior to artificial neural networks, deep neural networks, and ensemble algorithms in terms of interpretability, yet they fall behind these algorithms in terms of prediction performance. For this reason, the PLR model we propose in this study is important in terms of being an interpretable model and providing good prediction performance.

3. RELATED WORK

In the literature, there exist numerous studies on the PLR problem. These studies address the PLR problem by defining it in different ways. As mentioned before, while some studies impose a continuity constraint between intervals, some define the PLR problem without this constraint. Some studies deal with functions with convexity or concavity constraint. Some studies consider the PLR problem on discrete data, while others try to fit PLR on a polynomial function. This is also referred to as “curve fitting”. Also, studies consider multivariate or univariate datasets. In studies dealing with multivariate data, the partitioning can be realized on a single dimension or on more than one dimension. All of these are gathered under the umbrella of piecewise linear regression (PLR). Therefore, we can claim that there is no common definition of PLR in the literature.

Studies in the literature solve the PLR problem by adopting different approaches. In this thesis, we present the literature by classifying according to the solution methodology. These are dynamic programming, mathematical programming models, and heuristic methods.

3.1. Dynamic Programming (DP)

[24] is one of the early studies considering PLR problem. Bellman and Roth consider curve fitting problem. Therefore, they inherently impose the continuity requirement for piecewise linear functions. The number of breakpoints k, or “joint” points as referred to in [24], is predefined. All cases from one breakpoint to k breakpoints are enumerated and the corresponding errors are computed. Then, by dynamic programming, starting from the joint point of the rightmost interval, all joint points are found in a way resulting in the minimum error.

Guthery [25] presents another early study that solves the PLR problem with dynamic programming. In this study, sum of squared errors is used as the error metric. This approach assumes that data is ordered in the predictor (independent) variable and experiments on two time-series data.

Hawkins [26] addresses the discontinuous PLR problem. He presents two approaches to solving the PLR problem, one of which is a dynamic programming algorithm with maximum likelihood estimation (MLE). The other method is called the hierarchical method, in which hierarchical split operations are performed. First, data is split into two segments at the ”best” point, which results in the lowest error. A search process is conducted to find the best point. After splitting the data into two, a search is performed to find the best points of each interval that split the interval into two. Among the best points found in each interval, the best one is selected as the second breakpoint and the other is eliminated. Thus, we obtain two breakpoints that split the data into three intervals. Subsequently, the algorithm keeps running until a predefined k number of breakpoints are found.

Bai and Perron [4] propose a DP algorithm for finding a predefined number of breakpoints. They present a procedure to obtain the possible breakpoint locations. Firstly, they impose a minimum distance between each breakpoint, resulting in some reduction in the number of possible breakpoints. Furthermore, they offer more reductions by making practical judgments relating to possible breakpoint locations. After obtaining the regression errors resulting from each breakpoint combination, they proceed with the DP procedure to find the solution that provides the minimum error. To arrive at a solution, they start with a one-break solution up to the k-break solution, where k is a predefined value.

The study [27] by Rote deals with isotonic regression problems. He proposes a DP algorithm which uses a priority queue structure in the solution. The data discussed in this work show a monotonically increasing characteristic. This study presents a mathematical perspective to solve the PLR problem rather than giving experimental results using a dataset.

Wu et. al. [6] study the PLR problem by considering the application on financial time-series data. They propose a dynamic programming algorithm with a two-level design and provide their time efficiency as $O(kn^2)$, where k is the number of intervals and n is the number of observations. The data they experiment with consists of synthetic and real-world datasets. The largest dataset they experiment with has 1,560 points in total.

3.2. Linear Programming

Bertsimas and Shioda [28] introduce integer optimization models for classification and regression problems. They propose an algorithm which they named CRIO (Classification and Regression via Integer Optimization). For regression, they compare CRIO with decision tree and multivariate adaptive regression splines (MARS) algorithms. They report mean absolute error (MAE) and mean squared error (MSE) as the error metrics and show that CRIO performs better in terms of both metrics. The largest dataset experimented with has 8 independent variables and 4,177 observations.

[29] and [30] use mixed-integer linear programming (MILP) models to solve PLR problems in optimality. Toriello and Vielma [30] deal with piecewise linear convex functions. They report that scalability is an issue for their algorithm because the computational efficiency is poor.

In the study of Yang et. al. [5], a definition is given for the considered PLR problem. In this problem, data is partitioned into mutually exclusive segments on a single dimension and each segment is represented with a distinct multivariate linear regression. The dimension that partitions the data and the number of breakpoints are defined a priori. They propose a MILP model for estimating the breakpoint locations and regression coefficients of each linear regression. Experimental results are provided where the proposed method is compared with several methods including single linear regression, support vector machine, artificial neural networks, k-nearest neighbors, and MARS. They show that the prediction performance of their algorithm is better than all mentioned algorithms on average. However, this method is limited to small and medium datasets, as the computational cost is high due to the mathematical programming methodology they use.

Gkioulekas et. al. [31] extend the study of Yang et. al. ([5]) and deal with selecting the optimal number of breakpoints. They use Akaike and Bayesian information criteria to detect the convergence of the algorithm. In their algorithm, model complexity is penalized in order to prevent the overfitting problem.

Gopalswamy et. al. [32] study PLR problem on multivariate data with a concavity constraint and adopt mathematical programming (MILP) methodology. They propose valid inequalities for the MILP model. Valid inequalities restrict the search space of the solution, therefore MILP model with valid inequalities provides better computational efficiency than a standard MILP model.

In the work of Rebennack et. al. [33], PLR problem with convexity constraint is studied. They consider the PLR problem with the continuity requirement. They experiment with discrete data and also univariate continuous functions (curve-fitting). The largest dataset they experiment with contains 1,216 data points.

Warwicker and Rebennack [34] study the PLR problem by imposing a continuity constraint and adopting the MILP methodology. Particularly, they study outlier detection and propose an algorithm to detect the outliers in the optimization process. To enable faster implementation of MILP, they benefit from combinatorial Benders decomposition.

3.3. Heuristic Methods

Terzi and Tsaparas [7] propose a heuristic algorithm based on dynamic programming with a predefined value of k intervals. Initially, they partition the data into m partitions. After, they employ the DP algorithm for each m partition, where the number of intervals inside each m partition is k. After the completion of this step, they have the $k \times m$ breakpoints. These points are representatives of the interval they belong to, and each is weighted by the interval length. DP algorithm is performed on $k \times m$ representatives, and the result is reported as the output. This work presents the computational efficiency of their heuristic algorithm as well as experimental evaluation.

Magnani and Boyd [35] consider a PLR problem with a convexity constraint. They propose a heuristic method based on a variation of the k-means algorithm in a multivariate setting. They start with an initial solution. The initial solution is generated randomly by following a procedure in which data points are clustered into groups according to their match to some distribution. Then, the initial partitions are updated to get a better solution by iterating until the result converges to the optimum or the maximum iteration limit given is reached.

Acharya et. al. [36] propose a heuristic procedure where the number of breakpoints is predefined. They provide the computational boundary of the proposed heuristic and demonstrate its performance with numerical experiments. The algorithm they provide is based on a greedy merging of the intervals. Initially, each data point defines an interval on its own. After, each interval is assumed to be merged with its neighbor. A certain number of intervals providing a lower error are merged while the rest remain as before. This merging process is performed until the number of the resulting intervals reaches a certain number. They compare their algorithm to a DP solution. The largest data they experiment with contains 10,000 observations.

Diakonikolas et. al. [8] study the PLR problem for multivariate data. They propose a heuristic algorithm where the partition is conducted on more than one feature. In particular, if data has d features, they partition the data on d' features, where $d' < d$, with a presumption that some features have more effect on dramatic changes in the data, such as time and location. The proposed heuristic is based on a greedy merging procedure. Experiments are conducted with synthetic and real-world datasets. The largest data they experiment with is a synthetic dataset with $n = 8000$ observations and $d = 10$ features, where the partitioning occurs on $d' = 2$ features.

The study by Tunç and Genç [3] addresses a similar problem definition as [5]. Data is partitioned on a single feature and subsets of data in each interval are represented by a unique multivariate linear regression. Their work is based on mixed-integer linear programming, yet they propose a heuristic procedure to enable faster computation. They start with an initial solution where data is separated into equal-length observations on the partition feature. After, they perform a column generation algorithm based on the initial solution. The experiments conducted include several synthetic and real-world datasets. They compare their algorithm to the study of [5] and show that their algorithm can handle larger datasets better.

4. PROPOSED METHOD

In this thesis, a heuristic approach is adopted to enable a computationally efficient solution that can be used even for larger datasets. First, we shall remark on the main idea of PLR. Piecewise linear regression partitions the data into subsamples and fits a unique linear regression to each partition such that each can be adequately modeled by a single linear regression. Using this fact, we first split the data into partitions, each with an equal number of observations. Those partitions serve as an initial solution. Then, we iteratively merge or split the intervals until there is no benefit in doing another merge or split according to the criteria to be introduced.

As previously stated, we consider a discontinuous piecewise linear regression problem in the current study where data is multivariate. We assume that the dataset is in increasing order on the predefined partition feature. The problem is to find the number of breakpoints, breakpoint locations, and optimal linear regression parameters of each interval. As the error metric of linear regression, sum of squared errors (SSE) is used. The overall error of the resulting PLR model is given in Equation 4.

$$\sum_{k=1}^{K}\sum_{i\in I_k}(y_i-\hat{y}_i)^2 \tag{4}$$

where I_k is the set of observations in the interval k, y_i is the value of the target feature of observation i and $\hat{y}_i$ is the corresponding prediction.

We start by generating an initial solution by simply partitioning the data into equal-length intervals across the range of the partition feature. We refer to this initial solution as *base intervals*. The length (or the number of observations) of each interval is determined by a user-defined parameter w, *base interval size*. For a dataset with n observations, the number of intervals of the initial solution will be $K = \lceil n/w \rceil$. Let I be the base interval set. Thus, $|I| = K$ and $I = \{I_{1w}, ..., I_{((K-1)w+1)n}\}$, where $I_{1w} = \{(x_1, y_1), ..., (x_w, y_w)\}, ..., I_{((K-1)w+1)n} = (x_{((K-1)w+1)}, y_{((K-1)w+1)},), ..., (x_n, y_n)\}$. Note that the number of observations in the last interval may be less than w because the data size is not always a multiple of w. The algorithm for generating the initial solution is given in Algorithm 1.

```
Algorithm 1 GenerateBaseIntervals($X, w$)
Input:
    $X$: a multivariate dataset
    $w$: base interval size
Output:
    $I$: initial intervals

1: $I \leftarrow []$
2: $n \leftarrow$ the number of observations in $X$
3: $i \leftarrow 0$
4: Split the dataset into w equal length intervals (the last interval size might be less)
5: for $j = \{1, 2, ..., \lceil n/w \rceil\}$ do
6:     Calculate linear regression of $I_j$, where $I_j = \{(x_{i+1}, y_{i+1}), \ldots, (x_{j*w}, y_{j*w})\}$
7:     $I \leftarrow I \cup I_j$
8:     $i \leftarrow j * w$
9: end for
```

We benefit from memoization and store the sum of squared error of each interval when generating the base intervals. Having obtained the initial solution, we start with the merge process. We iterate over the base intervals and check whether two intervals are eligible to be merged. If they are, we merge the corresponding intervals. We keep iterating over all the intervals at hand to check for a possible merge. Then, we start iterating over the intervals to check for a possible split. The merge and split processes will be explained in more detail. But at this point, it is necessary to explain the criteria to perform a merge or split.

When two intervals are merged, it is obvious that the SSE after the merge will not be smaller than the total error resulting from the two intervals. Before merging, intervals were represented by two different linear regressions. After, only one linear regression represents all data inside the two intervals. If the regression error due to the merged interval were smaller, the intervals before the merge would be represented by the same linear regression of the interval after the merge. Because linear regression tries to fit a linear function with the smallest error. In this case, the error of the merged interval will be at least equal to the total error of the two intervals before the merge, but not less. Thus, regression error after a merge never decreases. Conversely, a split of an interval will decrease the SSE since each interval after the split will be represented by a unique linear regression that fits at least as well as the linear regression of the interval before the split. This fact leads to establishing the following metric provided in Equation 5 to be used when checking for a merge or split operation.

$$c = \frac{E_{pr} - (E_{pi} + E_{(i+1)r})}{\sum_{k=1}^{K}(E_{I_{((k-1)w+1)(kw)}})} \tag{5}$$

where E_{pi} and $E_{(i+1)r}$ are errors of the intervals I_{pi} and $I_{(i+1)r}$ respectively.

This equation serves in two ways. First, if it is being used in a merge operation, it shows the cost of the merge. Intervals I_{pi} and $I_{(i+1)r}$ show the intervals before merging and I_{pr} shows the interval established via merging. Thus, this equation basically describes whether the increase in the regression error due to merging is significant relative to the total error from all intervals.

The second use of Equation 5 is in a split operation. This time I_{pr} shows the interval before the split while I_{pi} and $I_{(i+1)r}$ show the intervals established by the split from the breakpoint i. Equation 5 describes whether the decrease due to split provides a significant gain over the total error of all intervals.

We check the cost of a merge or gain of a split provided by Equation 5 against a user-defined threshold value: T_m for the merge and T_s for the split. If the cost of a merge is less than T_m, we perform merging. Similarly, if the gain of a split is greater than T_s, we perform a split. At this point, we would like to emphasize the reason why two different threshold values are used for the merge and split, although the cost and gain equations are the same. We check if the merge cost is less than T_m and if not we do not merge. This means that we do not allow the regression error to increase if the cost is significantly higher. Conversely, in a split, we check if the split gain is higher than T_s and if not we do not perform the split. Meaning, we do not allow the split to be realized if the split does not provide a significant gain in terms of the total error. Therefore, we do not perform split operations for minor shifts in the data, and we do not perform merge operations that prevent capturing significant structural shifts in the data.

Algorithm 2 Merge(X, I, T_m, M)

```
Input:
    $X$: a multivariate dataset
    $I$: current intervals
    $T_m$: merge threshold
    $M$: previously merged intervals
    $S$: previously splitted intervals
Output:
    $I, M$

 1: $p \leftarrow length(I)$
 2: while $i < (p-1)$ do
 3:     $merge\_flag \leftarrow false$
 4:     $I_{left}, I_{right}, I_{merge} \leftarrow I_i, I_{(i+1)}, I_i \cup I_{(i+1)}$
 5:     if $I_{merge}$ not in $M$ and $I_{merge}$ not in $S$ then
 6:         $M \leftarrow M \cup I_{merge}$
 7:         $merge\_flag \leftarrow true$
 8:     end if
 9:     if $merge\_flag$ then
10:         Fit a linear regression for $I_{merge}$
11:         Calculate the cost using Equation 5 and store it in $c$
12:         if $|c| < T_m$ then
13:             $I \leftarrow I \setminus \{I_{left}, I_{right}\} \cup I_{merge}$
14:             $p \leftarrow p - 1$
15:         end if
16:     end if
17:     $i \leftarrow i + 1$
18: end while
```

After obtaining K number of base intervals, we start to iterate over them to perform merge operations. Let $I_{pi} = \{(x_p, y_p), ..., (x_i, y_i)\}$ and $I_{(i+1)r} = \{(x_{i+1}, y_{i+1}), ..., (x_r, y_r)\}$ be the intervals at some iteration where $x_p \leq x_i \leq x_r$ and I_{pr} be the interval established by merging. First, we check if I_{pi} and $I_{(i+1)r}$ have been checked for merge or already split. If these intervals have been established by split, we do not merge them again. If they have not been split but have been checked for merging and were not eligible, we do not attempt to merge them again. After, if they survive to be merged, the linear regression is computed for the new interval I_{pr} and its error E_{pr} is calculated. Errors of the I_{pi} and $I_{(i+1)r}$ are not calculated again, but are instead obtained via memoization. Having obtained the regression errors of the respective intervals, the merge cost is calculated by substituting the corresponding variables in Equation 5. If the cost is less than T_m, the merge is performed. The pseudo-code of the merge process is provided in Algorithm 2.

After we have iterated over all the intervals for the merge, we start the split process. In the split process, we iterate over all the intervals and split the eligible ones. At each iteration, we first check whether the interval is established by merging. If it is, we do not attempt to split the interval. If not, then we first need to decide at which point the interval will be split. For this, we establish a certain number of candidate split points to be split at and ultimately choose only one for the split. *Split number (r)* parameter decides which and how many points are the candidate split points. The interval is split into r split points with an equal number of observations and these points constitute the candidate split points. However, *minimum interval size (u)*, which is another user-defined parameter, can cause some candidate split points to be eliminated. The minimum interval size parameter determines the minimum number of observations that each interval after the split should have. Hence, r and u together determine the candidate split points. The following example illustrates the establishment of candidate split points.

Let $I_{p(p+299)} = \{(x_p, y_p), \ldots, (x_{p+299}, y_{p+299})\}$ be an interval with $m = 300$ observations, and let $r = 5$ and $u = 100$. This means that the interval $I_{p(p+299)}$ can be split at a point chosen from 5 possible points. There exists $\lfloor m/(r + 1) \rfloor$ observations between each consecutive candidate point. Each candidate point creates two intervals that differ in the number of observations. The candidate points are shown in Figure 4.1.

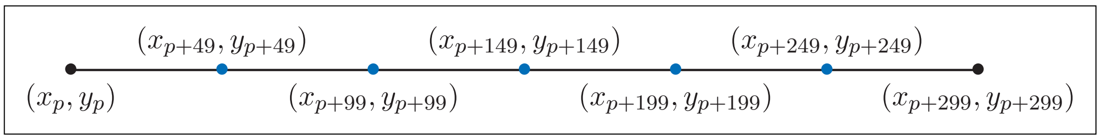


Figure 4.1 Candidate split points (shown in blue)

Each candidate split point is checked to see if it meets the minimum interval size requirement. When the interval is split from (x_{p+49}, y_{p+49}), the number of observations to the left of it will be 50, which is less than the minimum interval size of 100. Therefore, this point is eliminated and not checked to be split. Similarly, when the interval is split at (x_{p+249}, y_{p+249}), the number of observations to the right will be 50. Even though the left interval meets the minimum interval size requirement of 50, this point is still eliminated because the right interval does not meet the requirement. Therefore, only the remaining 3 points will be checked to be split at. Figure 4.2 demonstrates the candidate split points elimination process. In the figure, the points shown in red are eliminated while the green ones are eligible to be a split point.

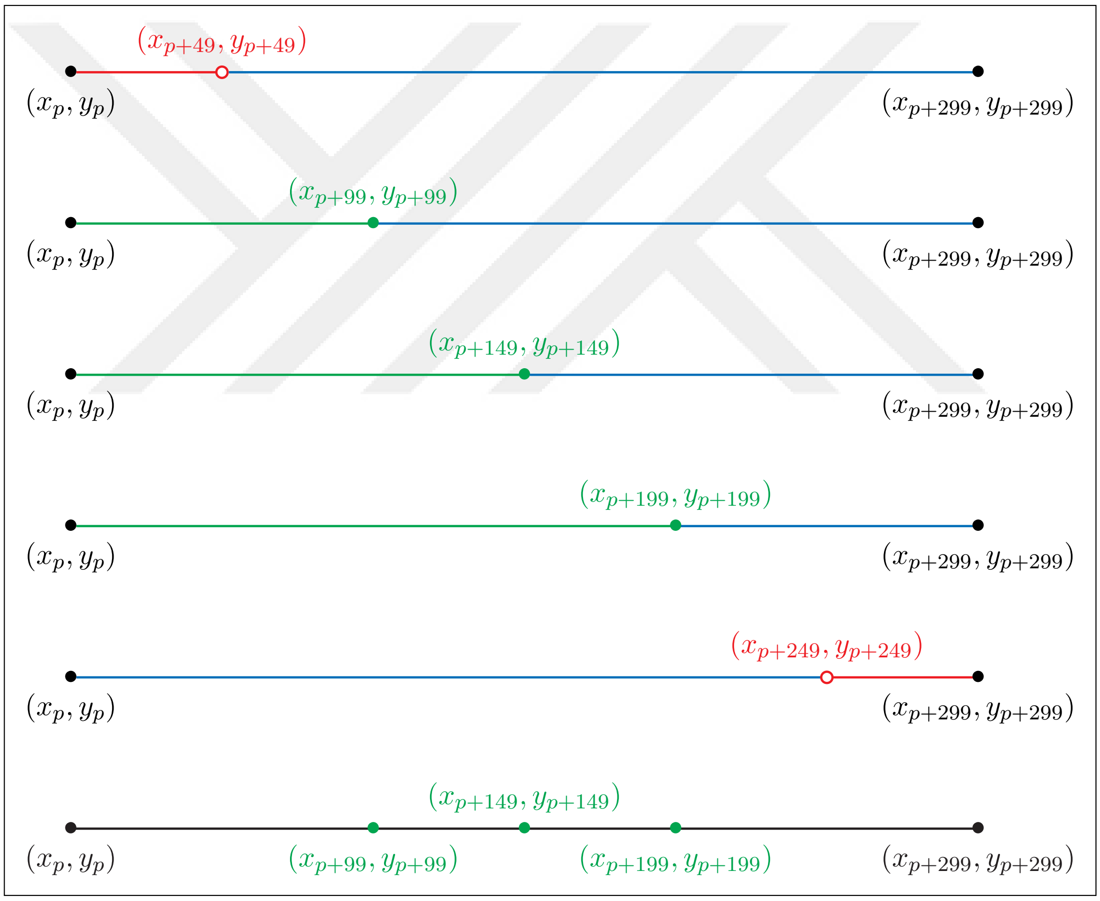


Figure 4.2 Split point elimination process (shown in red are discarded)

After deciding on the candidate split points, we compute the gain resulting from each point by Equation 5 and select the point which provides the highest gain among them. If the interval is not split at the selected point before, we check whether the corresponding gain is greater than T_s, and if it is, we realize the split. The algorithm for the split process is given in Algorithm 3.

Algorithm 3 Split(X, I, T, S, r, u)

```
Input:
    $X$: a multivariate dataset
    $I$: current intervals
    $T_s$: split threshold
    $S$: previously splitted intervals
    $M$: previously merged intervals
    $r$: split number
    $u$: minimum interval size
Output:
    $I$, $S$

 1: $p \leftarrow length(I)$
 2: while $i < p$ do
 3:     if $I_i$ in $M$ then
 4:         Skip to the next iteration
 5:     end if
 6:     $best\_cost, best\_point \leftarrow -\infty, -\infty$
 7:     Let $s$ be the start point of $I_i$ and $e$ be the end point of $I_i$
 8:     for $j$ in $\{1, 2, ..., r\}$ do
 9:         Let $split\_point$ be the $j^{th}$ split point in interval $I_i$
10:         Let $I_{s(split\_point)}$ be the interval containing observations from $s$ to $split\_point$
11:         Let $I_{(split\_point+1)e}$ be the interval containing observations from $(split\_point + 1)$ to $e$
12:         if $||I_{s(split\_point)}|| < u$ or $||I_{(split\_point+1)e}|| < u$ then
13:             Skip to the next iteration
14:         end if
15:         Fit linear regressions for $I_{s(split\_point)}$ and $I_{(split\_point+1)e}$
16:         Calculate the cost using Equation 5 and update $best\_point$ and $best\_cost$ if $|c| > best\_cost$
17:     end for
18:     if $|best\_cost| > T_s$ and $I_i$ not in $S$ where the $best\_point$ is the corresponding breakpoint then
19:         $S \leftarrow S \cup I_i$ /* add $I_i$ to split intervals along with the $best\_point$ */
20:         $I \leftarrow I \setminus I_i \cup \{I_{left}, I_{right}\}$
21:         $p \leftarrow p + 1$
22:     end if
23:     $i \leftarrow i + 1$
24: end while
```

Merge and split processes are performed consecutively on the available intervals until no interval is found eligible for either a merge or split. A merge or split process is conducted by iterating over all intervals, and when the iteration is complete, the next process is started. At each phase, we store the information of the intervals processed earlier, so, we do not consider the same intervals over and over again. The general framework of the proposed merge-and-split heuristic (MaSH) is provided in Algorithm 4.

Algorithm 4 MergeAndSplitHeuristic(X, w, T_m, T_s, r, u)

```
Input:
    $X$: a multivariate dataset
    $w$: base interval size
    $T_m$: merge threshold
    $T_s$: split threshold
    $r$: split number
    $u$: minimum interval size
Output:
    $I$: resulting intervals

1: $M, S \leftarrow [], []$
2: $I \leftarrow GenerateBaseIntervals$(X, w)
3: repeat
4:     $I, M \leftarrow$ Merge($X, I, T_m, M, S$)
5:     $I, S \leftarrow$ Split($X, I, T_s, S, M, r, u$)
6: until no change occurs in the intervals $I$
```

Figures 4.3 to 4.9 show the execution of the proposed heuristic for sample data. The vertical lines in black show all breakpoints. Orange and green vertical lines indicate the breakpoints that are currently in a merge or split operation. Orange lines indicate that the current interval is not eligible to be merged or split, while green lines indicate that the merge/split operation has been performed. Red lines exist only in a split operation and show the candidate split points of the corresponding interval. Figure 4.10 shows the resulting intervals after the execution of the merge-and-split heuristic is completed. The total number of iterations is 3, this means that the heuristic finds the solution after 3 iterations of merge and split processes. Note that this does not mean that all existing intervals are iterated 3 times. Iterations are reduced for each merge and split process due to the condition of not splitting a previously merged interval and not merging a previously split interval.

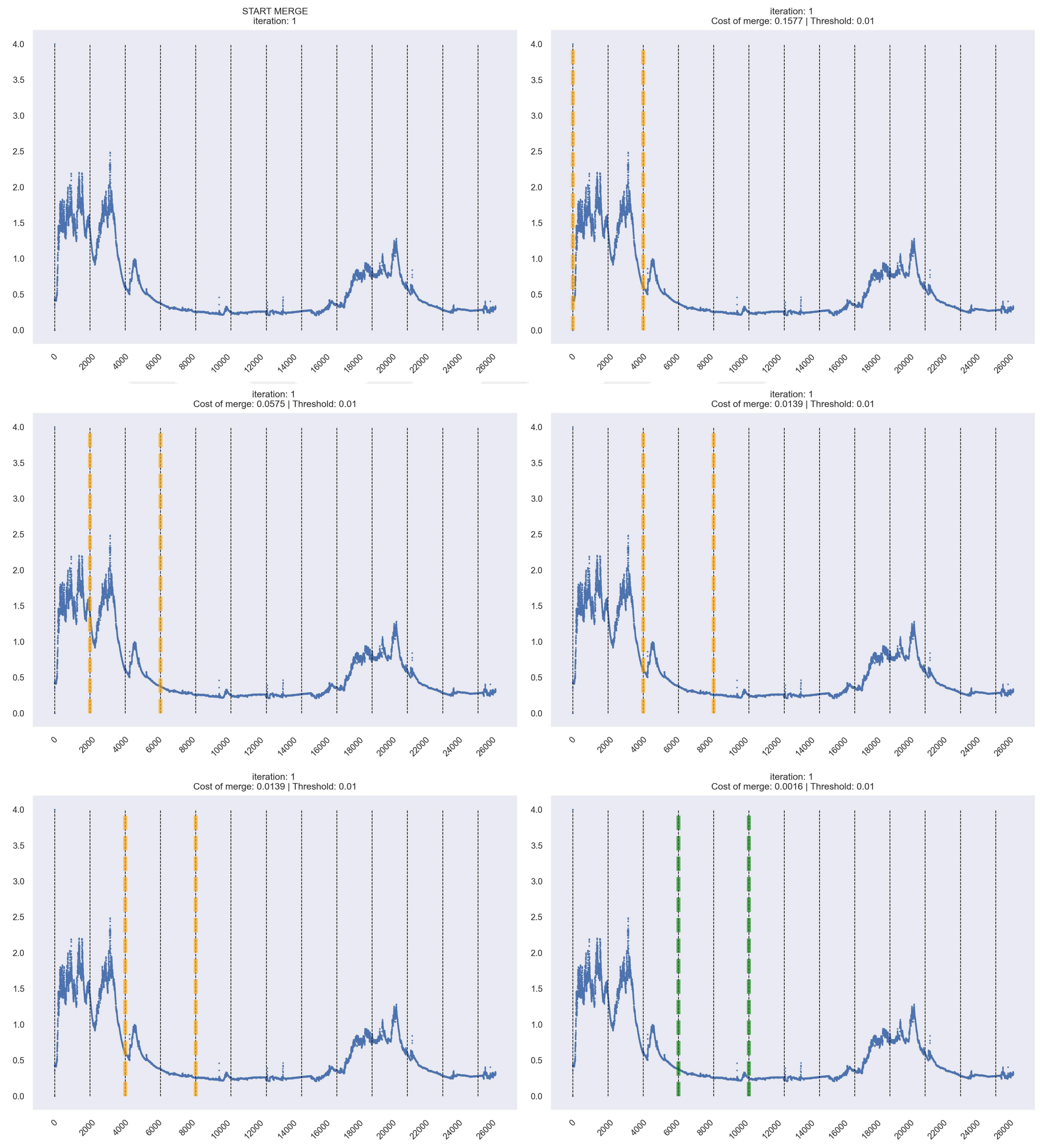


Figure 4.3 Execution of MaSH for sample data (1)

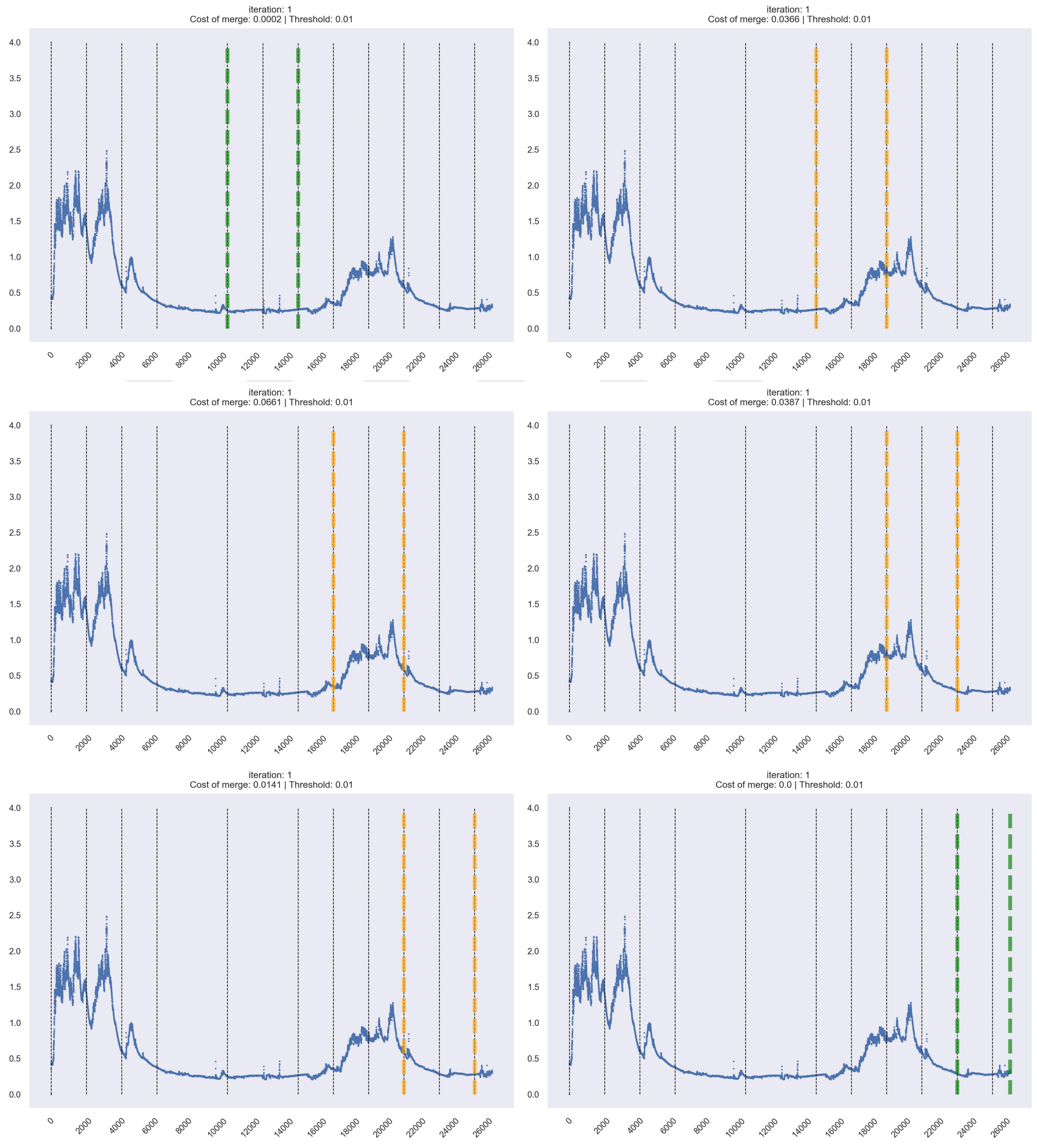


Figure 4.4 Execution of MaSH for sample data (2)

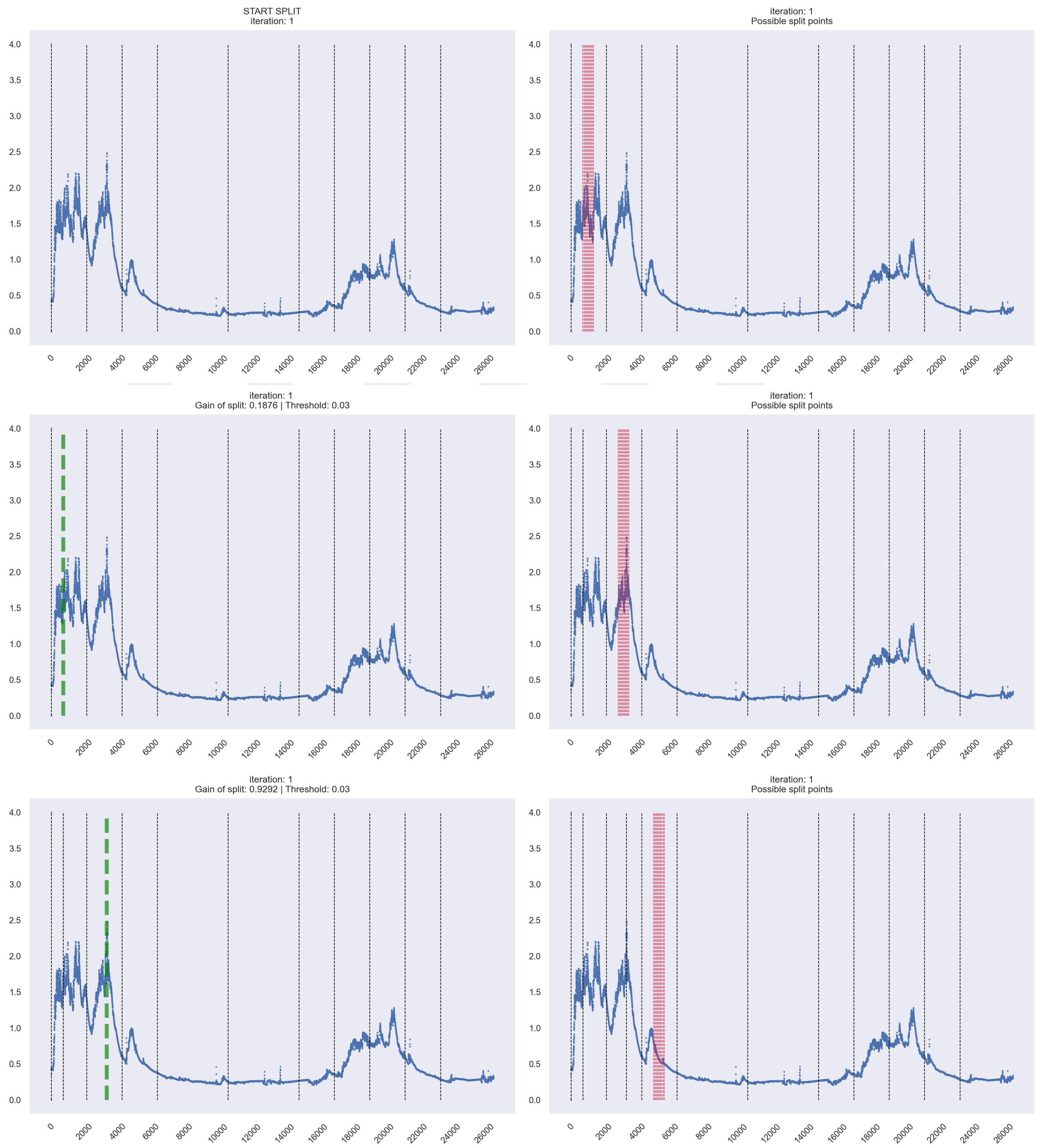


Figure 4.5 Execution of MaSH for sample data (3)

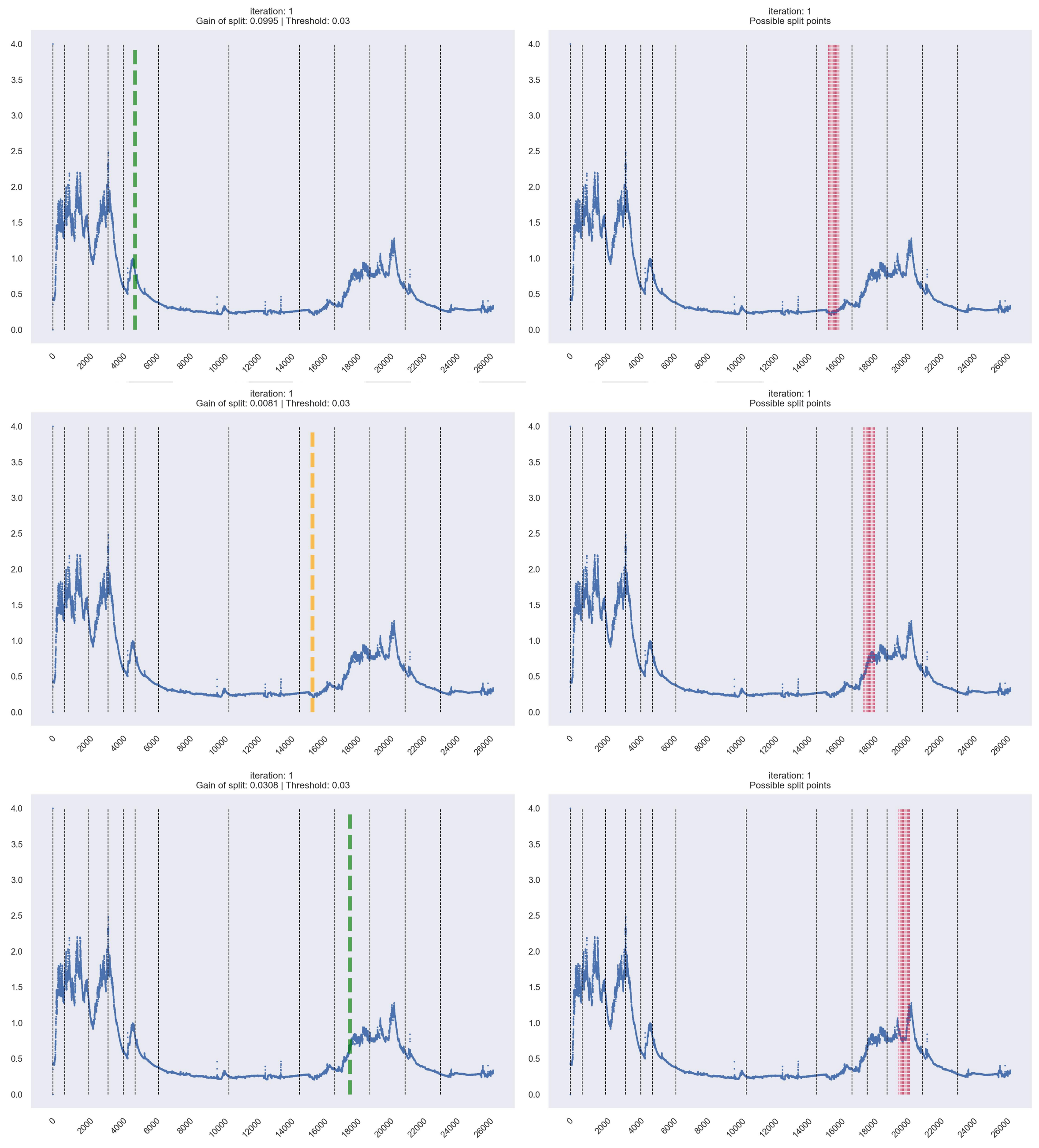


Figure 4.6 Execution of MaSH for sample data (4)

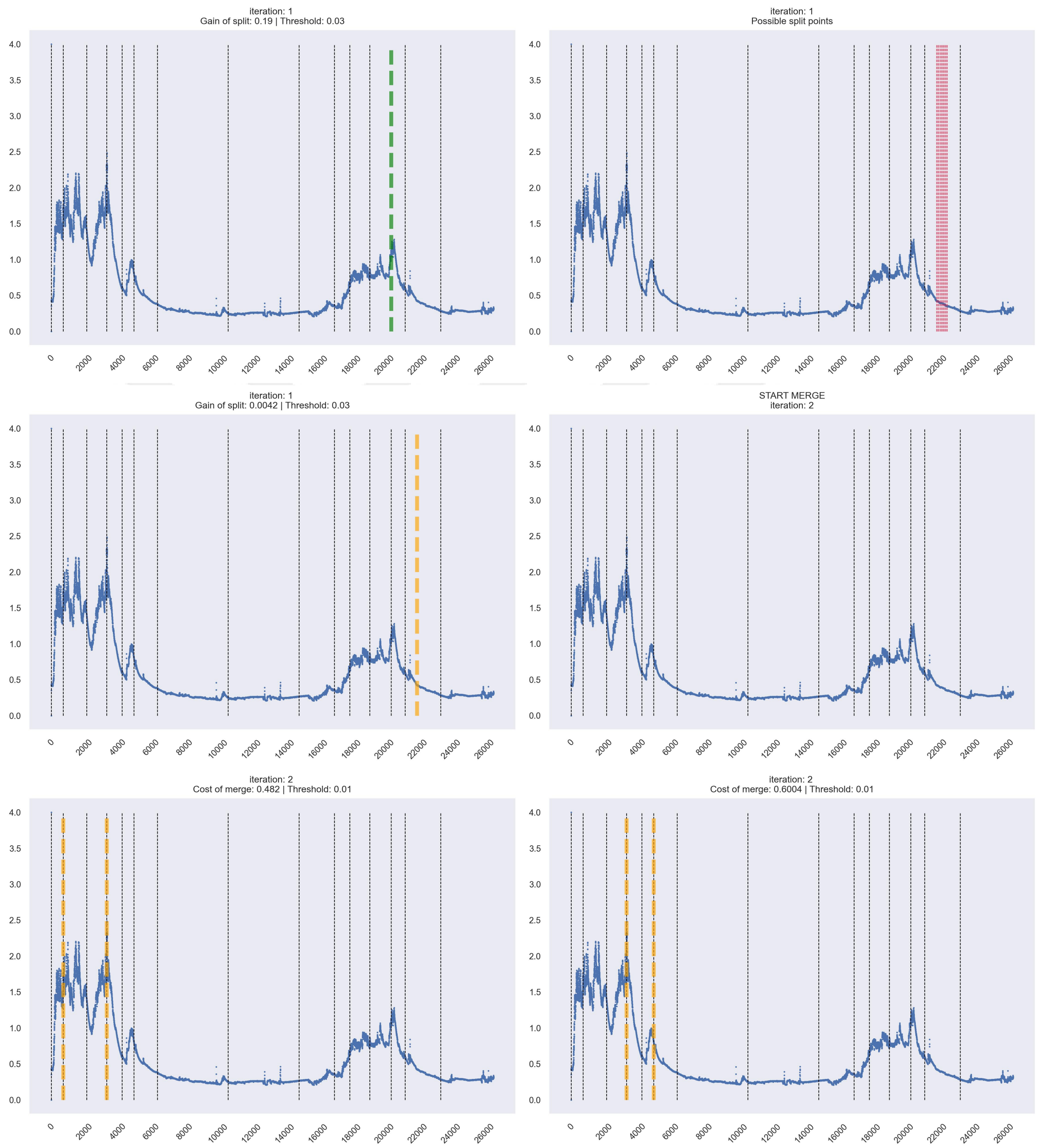


Figure 4.7 Execution of MaSH for sample data (5)

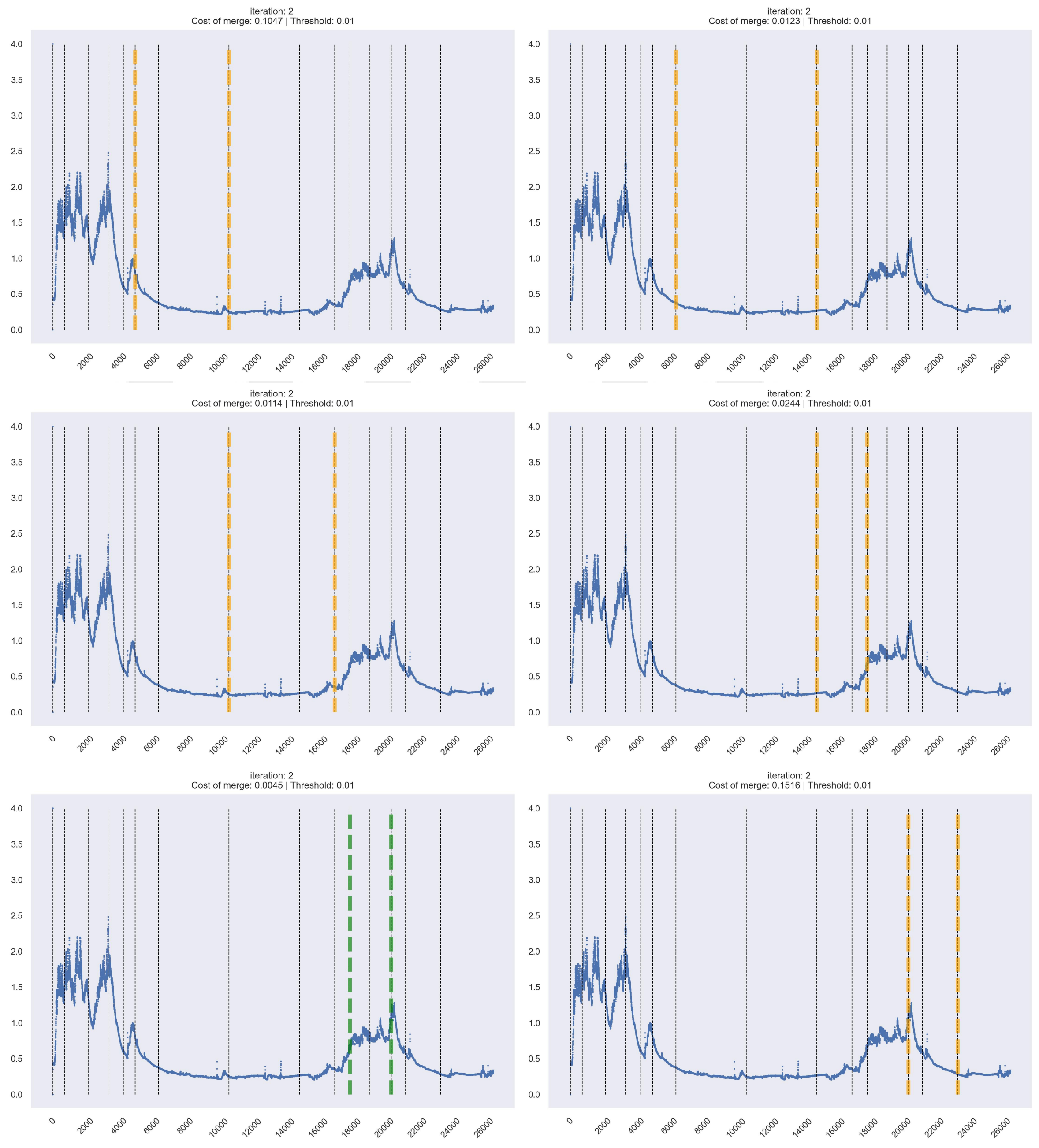


Figure 4.8 Execution of MaSH for sample data (6)

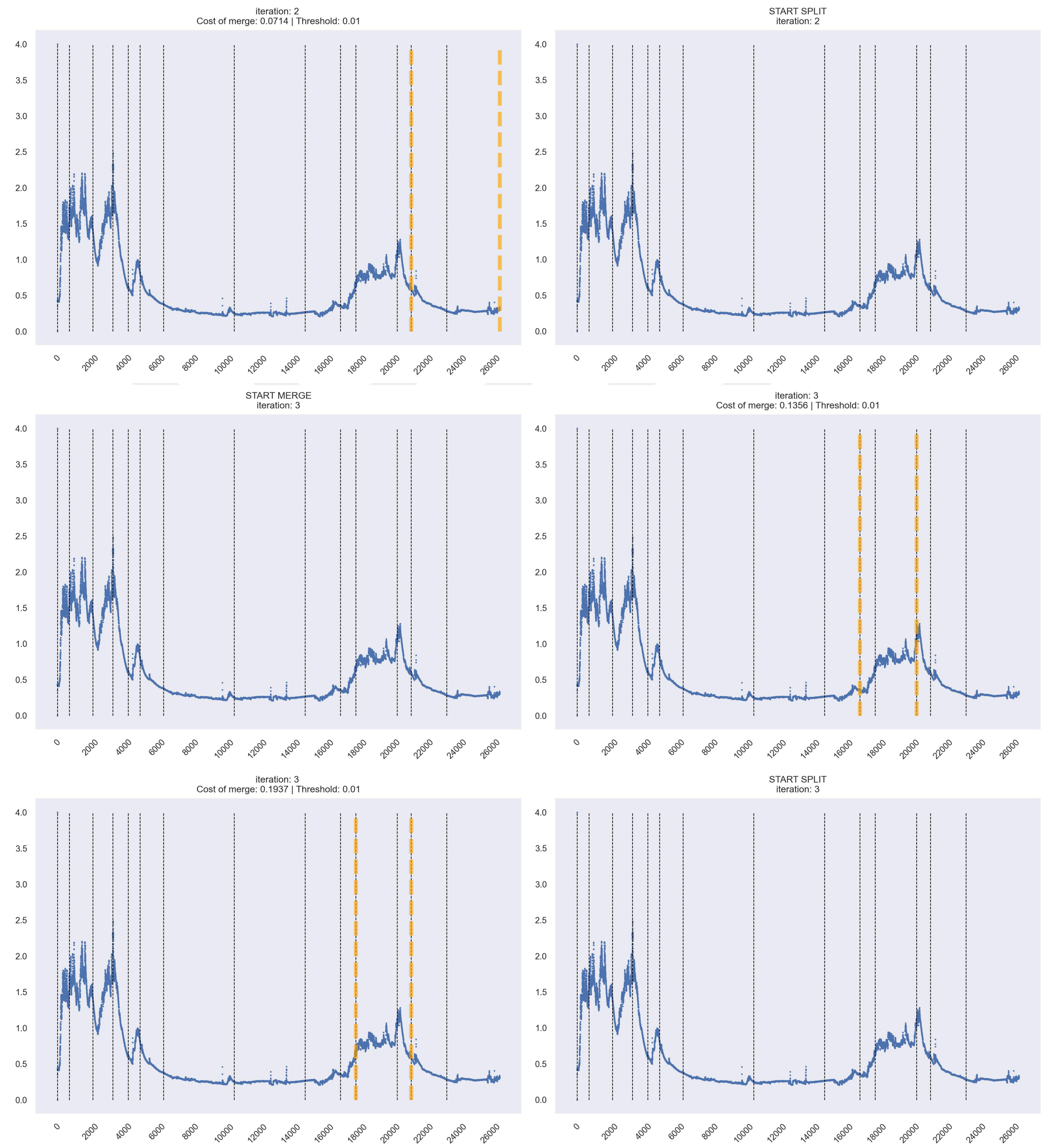


Figure 4.9 Execution of MaSH for sample data (7)

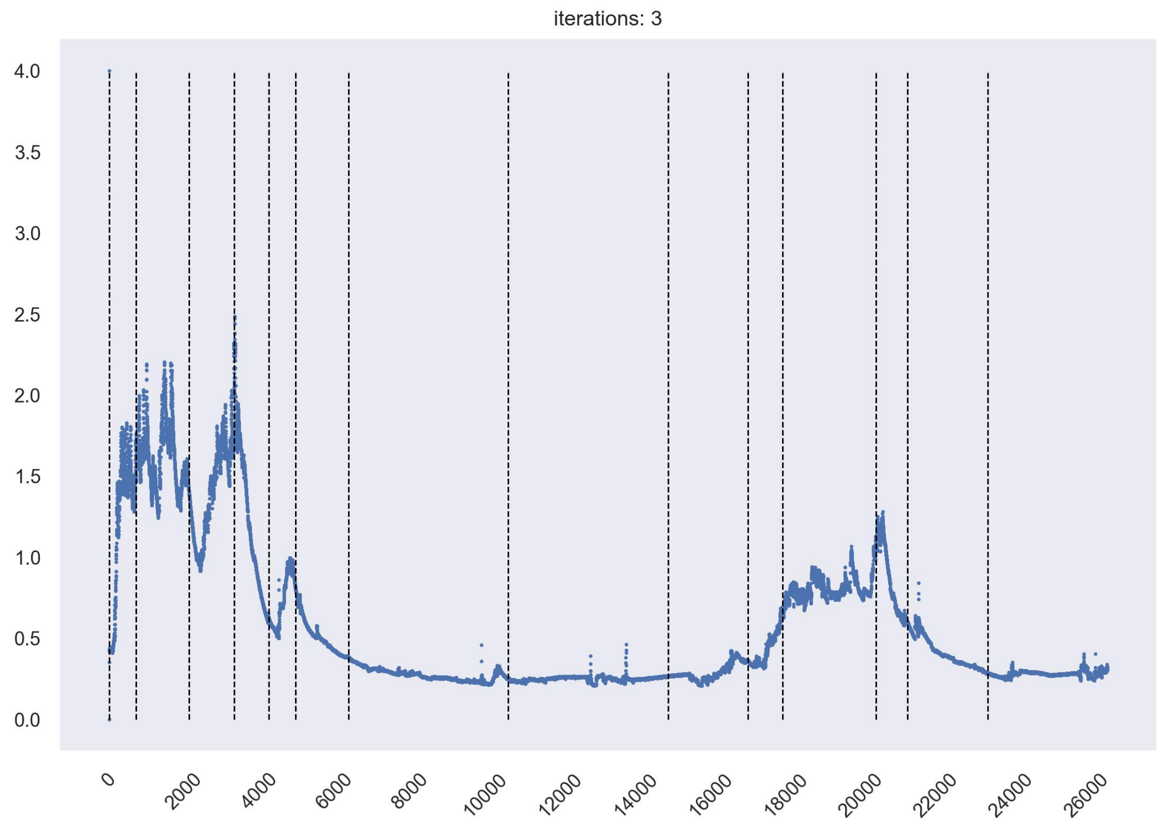


Figure 4.10 Output of MaSH for sample data

5. EXPERIMENTAL RESULTS

In this section, the computational work to assess the performance of the proposed MaSH algorithm is presented.

The evaluation consists of two parts. First, the prediction performance and computational efficiency of MaSH are evaluated against the commonly used machine learning algorithms. In the second part, the fitting performance along with the computational efficiency of MaSH are evaluated against a mixed-integer programming based PLR heuristic.

Evaluation is carried out with several real-world and synthetic datasets. In the following, the datasets used in the experiments are introduced. Then, the evaluation metrics used in this study are briefly summarized. Lastly, the details of the experiments are provided.

5.1. Datasets

5.1.1. Real-world Datasets

Nine real-world datasets obtained from online sources are used in the experiments. Eight of them are obtained from UCI Machine Learning Repository and one from Kaggle. Table 5.1 shows the summary of the real-world datasets. In the table, the number of observations and the number of features for each dataset are provided. Partition feature denotes the independent feature of the data that MaSH splits from. In this study, the same partition features as in study [3] are used for common datasets.

Figure 5.1 shows the scatter plot of each dataset with respect to the partition feature and target variables. As can be seen from the figure, the datasets are highly non-linear and each displays a different non-linear character.

Concrete Compressive Strength [37] dataset includes 8 independent variables such as cement and age, and establishes a relation between them and the compressive strength of concrete. This dataset consists of 1030 samples.

Data	Number of observations	Number of features	Partition feature
Concrete	1030	9	Age
Airfoil	1503	6	Frequency
Red Wine	1599	12	Alcohol
White Wine	4898	12	Volatile Acidity
WESAD S17-IBI	2138	2	1502435962
Power Plant	9565	5	Exhaust Vacuum
WESAD S8-EDA	26551	2	x
Temperature	45253	2	Datetime
Tetuan city power consumption	52416	9	Timestamp

Table 5.1 Real-world datasets used in this study

Airfoil Self-Noise [38] consists of 5 independent variables for estimating the noise level of airfoils.

Red Wine [39] and White Wine [39] datasets predict the wine quality using physicochemical features. Both datasets consist of 11 independent features.

Power Plant [40] contains 9565 observations collected between 2006 and 2011. Thıs dataset consists of 4 input variables to predict the plant's energy output.

WESAD stands for Wearable Stress and Affect Detection [41] and includes observations from 15 subjects obtained from wrist-worn and chest-worn devices in a laboratory environment. This dataset includes physiological and motion data such as electrocardiogram, electrodermal activity, blood volume pulse, respiration, and body temperature, etc. Each data is kept in separate files with the corresponding timestamps of the record. Two dataset instances are selected from the WESAD dataset, where data follows a non-linear trend. One of them is the interbeat interval (IBI) dataset from subject 17. The second one is the electrodermal activity (EDA) dataset of subject 8.

Tetuan city power consumption dataset [42] contains the electricity power consumption of Tetuan city of Morocco. The data were collected every ten minutes for a one-year period. Predictor variables include date time with ten-minute intervals, temperature, and humidity. It was shown that the hour in which electricity is used is the most important predictor [42]. Therefore, timestamp values are extracted from date time values.

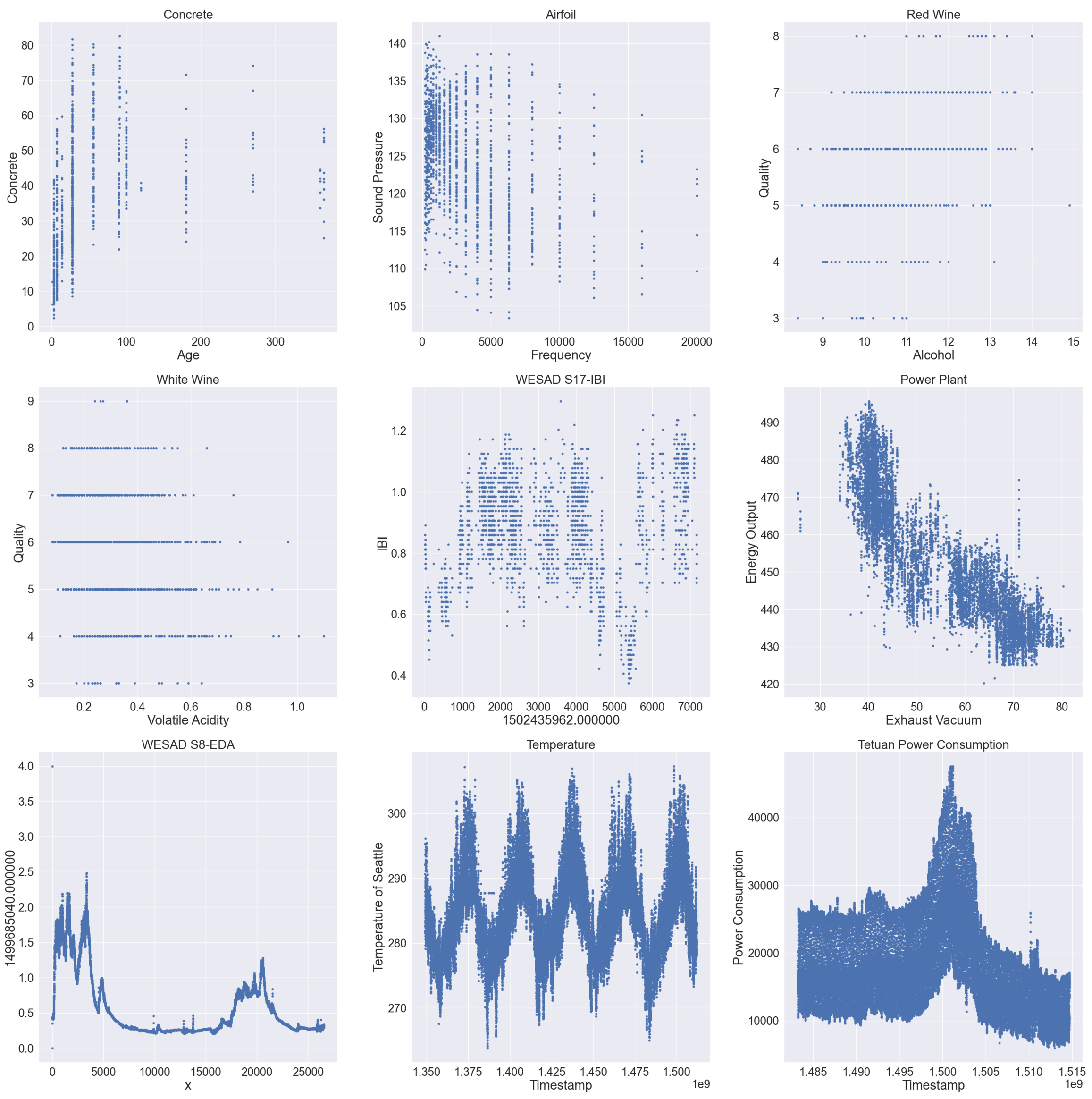


Figure 5.1 Scatter plot of the real-world datasets with respect to partition feature and target variables

Hourly Temperature dataset [43] is obtained from Kaggle. It contains hourly temperature data of 36 cities. Among them, a city's data in which non-linearity is present is selected.

Hourly Temperature and WESAD are univariate datasets, while the rest are multivariate.

5.1.2. Synthetic Datasets

PLR is most effective when structural shifts are present in the data. Therefore, synthetic datasets containing structural shifts are created to better demonstrate the performance of the PLR model. We use the data generation algorithm presented in [3]. In this algorithm, there are three parameters for generating the data: data size (n), the number of independent variables (m), and the number of intervals (K). Data size is the number of observations in the dataset. It has a direct effect on the computational efficiency of the PLR model. The more observations exist, the more computations are required. The number of independent variables represents the dimensions. It affects the regression coefficients. The number of intervals is the number of linear functions to be estimated by the PLR model. It can show the number of structural shifts in data. In addition to those parameters, noise is added systematically to data. Noise can obscure the structure of the data. Therefore, it is added to demonstrate the performance of the PLR model better. A fourth parameter in the algorithm is the seed value. Seed value provides controlled randomness in data. Thanks to the seed value, different datasets with exactly the same data size and the number of features are created.

In this thesis, two sets of synthetic datasets are created. The first set consists of three different data size levels (1000, 3000, and 9000). The second set consists of larger datasets with data sizes of (30000, 50000, and 100000). Both sets are created using three different levels of the number of independent features (4, 8, and 16), and three different levels of the number of intervals (3, 5, and 7). This leads to 54 different test cases generated. 3 different seed values are used for each test case, thus producing a total of 162 datasets.

Both sets of synthetic datasets are used in testing the prediction performance of MaSH against the machine learning algorithms. To test the fitting performance of MaSH, only the first set consisting of the number of observations (1000, 3000, and 9000) is used. The details will be provided in the corresponding sections.

5.2. Evaluation Metrics

After obtaining a regression model from data, it is essential to evaluate the model to see its performance and to observe how the model is good at predicting data that it has never encountered. Regression metrics are used to evaluate the model and compare it with other regression approaches.

5.2.1. Total Absolute Error (TAE)

Total absolute error (TAE) shows the sum of all absolute errors resulting from the model. The equation of TAE is as follows:

$$\text{TAE} = \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{6}$$

The absolute value of the error is taken to prevent negative and positive errors from canceling out each other. TAE shows the sum of all errors and does not scale to data size.

5.2.2. Mean Absolute Error (MAE)

Mean absolute error (MAE) is calculated by taking the average of the TAE:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{7}$$

Thus, MAE is scaled to the data size. Datasets having different number of observations and giving errors on the same scale can be compared with MAE. For example, if a model gives errors on a scale of 0 to 10 for both datasets, a comparison can be made on which data the model performs better, regardless of the number of observations the data contains. MAE gets a value between zero and infinity. When the MAE approaches zero, it means that the performance of the model is good.

5.2.3. Mean Squared Error (MSE)

Mean squared error (MSE) is calculated by averaging the squared differences between observed (actual) and predicted values. It is the average of the sum of squared errors (SSE). Squaring differences ensures that no negative values exist and larger errors are weighted more. Therefore, larger errors have more effect on the increase of the error while small errors become smaller. This makes MSE more sensitive to outliers while MAE is considered more robust in this regard. For this reason, MSE can be used in situations where larger errors are desired to be eliminated. The formula of MSE is given below.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{8}$$

5.2.4. Root Mean Squared Error (RMSE)

Root mean squared error (RMSE) is basically the square root of MSE and is calculated as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{9}$$

MSE measures the error in the square unit of the target variable. RMSE takes the square root and reduces the error back to the scale of the target variable, so the error becomes the same unit as the target variable.

5.2.5. Mean Absolute Percentage Error (MAPE)

Mean absolute percentage error (MAPE) measures the regression error as a percentage. This makes it easy to compare different datasets and different models in terms of predictive accuracy. The formula of MAPE is provided below:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{10}$$

Due to division by target value, this metric does not give accurate measurements for datasets with target values close to zero and cannot be used with target values that are zero. When this case occurs, MAPE gives a very high value rather than a percentage.

5.2.6. Normalized Mean Absolute Percentage Error (n-MAPE)

The target values of the datasets we use in the experiments are zero or very close to zero. Thus, to tackle the division by zero error, we introduce the metric normalized mean absolute percentage error (n-MAPE) to be used in the comparison of different datasets. This metric is similar to MAPE, but instead of taking the target value as the divisor, we take the range of the target variable. The formula is given as follows:

$$\text{n-MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{\max(y) - \min(y)} \right| \tag{11}$$

5.3. Experiments

In testing the performance of MaSH, we have two aims to pursue. The first one is testing the prediction and/or fitting performance and the second one is testing the computational efficiency of MaSH. To fulfill these aims, we design the experiments in two parts. In the first part, MaSH is compared with some commonly used machine learning algorithms and its prediction performance along with the computational efficiency is tested. In the second part, we compare the fitting performance and computational efficiency of MaSH with the column generation (CG) heuristic presented in [3], which is a mixed-integer programming based PLR solution. Figure 5.2 shows the general picture of the experimental design.

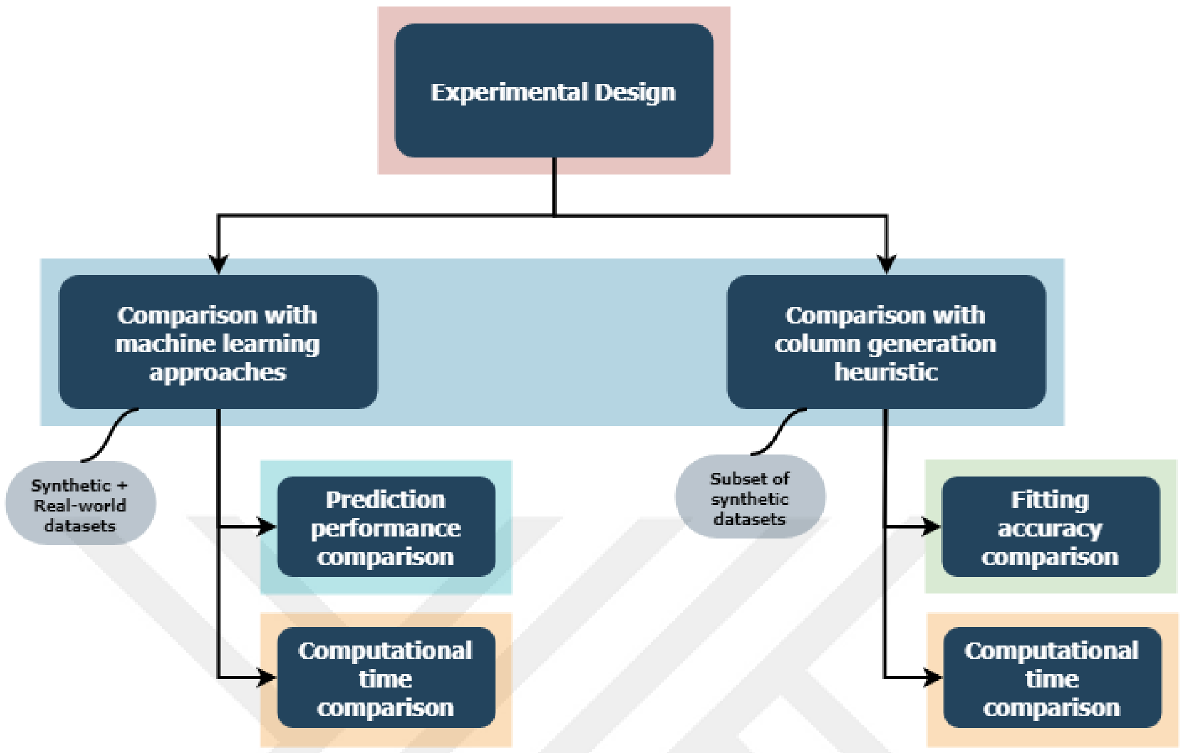

Experimental Design
Comparison with machine learning approaches
Comparison with column generation heuristic
Synthetic + Real-world datasets
Subset of synthetic datasets
Prediction performance comparison
Computational time comparison
Fitting accuracy comparison
Computational time comparison

Figure 5.2 Experimental design

5.3.1. Comparison with the Machine Learning Algorithms

Machine learning algorithms experimented with in this study are decision tree (DT), random forest (RF), XGBoost (XGB), XGBoost with random forest learners (XGBRF), support vector machine (SVM), k-nearest neighbors (KNN), artificial neural network (ANN), and multivariate adaptive regression splines (MARS). These are commonly used machine learning algorithms in practice and are also frequently studied in the literature. All experiments are conducted with Python 3.9. For all machine learning approaches, scikit-learn [44] implementations are used. MaSH is also implemented as compatible with scikit-learn.

Both real-world and synthetic datasets are used to assess the prediction and time performance of each algorithm stated above along with MaSH. Each data is split into train and test sets using 80 percent and 20 percent ratios, respectively. Then, a hyperparameter optimization process is conducted on the train set for model selection. Cross-validation is used in the optimization to obtain prediction accuracy on out-of-sample observations (validation set). For real-world datasets, 5-fold cross-validation is performed by repeating 10 times. At each repetition, data is permuted randomly and the average performance is reported at the end. Thus, the optimization process is conducted using various train and validation sets. Grid-search methodology is used in searching the optimal hyperparameters. Hyperparameter grids are formed for each method and dataset and candidate parameters are chosen empirically based on the prediction performance on the validation set. For computational feasibility, hyperparameter optimization process for the synthetic datasets is performed via 3-fold cross-validation. Because, the synthetic datasets are more in number, and the second group of the synthetic datasets are greater in terms of the number of observations.

Hyperparameter grids are selected empirically to include the parameters that affect the prediction performance most. Hyperparameter values are chosen separately for real-world and synthetic datasets based on the prediction accuracy of the cross-validation. Table 5.2 and Table 5.3 show the hyperparameter values of each method for the real-world and synthetic datasets, respectively.

Method	Hyperparameters
MaSH	• Base interval size: 100, 150, 200, 300, 350, 500, 1000, 2000, 3000 • Merge threshold: 0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2 • Split threshold: 0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2 • Split count: 10, 15, 20, 30 • Minimum interval size: 50, 100, 200, 500
Decision Tree	• Minimum samples split: 2, 5, 15, 35, 50 • Maximum depth: None, 5, 6, 7, 8, 10, 15

Table 5.2 Hyperparameter grid values used for the real-world datasets (cont.).

Continue on the next page

Method	Hyperparameters
Random Forest	• Number of estimators: 5, 6, 7, 8, 9, 10, 11, 12, 13, 15 • Maximum features: auto, log2 • Maximum depth: 5, 10, None
XGBoost	• Learning rate: 0.05, 0.1, 0.2, 0.3 • Maximum depth: 2, 3, 4, 5, 6 • Number of estimators: 100, 150, 200 • Subsample: 0.8, 1.0
XGBRF	• Learning rate: 0.01, 1.0, 1.05, 2.0 • Maximum depth: 1, 2, 3, 4, 5, 6, 7, 8 • Number of estimators: 100, 150, 200, 300, 500 • Subsample: 0.8, 1.0
SVM	• Kernel: Radial basis function (RBF) • Gamma: Scale • Strictness (C): 1, 2, 3, 4, 5 • Epsilon: 0.1, 0.2, 0.3, 1.0
KNN	• Number of neighbors: 3, 5, 7, 9, 13, 19 • Weights: uniform, distance • Power parameter for the Minkowski metric: 1, 2

Table 5.2 Hyperparameter grid values used for the real-world datasets (cont.).

Continue on the next page

Method	Hyperparameters
ANN	• Hidden layer sizes: 150, 200 • Activation function: tanh, relu • Solver: Stochastic Gradient Descent (SGD), Adam • Strength of the L2 regularization term ([44]): 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001 • Learning rate: 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001
MARS	• Maximum degree: 1, 2, 3 • Penalty: 1.0, 3.0

Table 5.2 Hyperparameter grid values used for the real-world datasets

Method	Hyperparameters
MaSH	• Base interval size: $n/20$, $n/10$, $n/5$ • Merge threshold: 0.01, 0.05, 0.1, 0.2, 0.3 • Split threshold: 0.01, 0.02, 0.03, 0.05, 0.1 • Split count: 10, 20, 30, 50 • Minimum interval size: $n/30$, $n/20$, $n/10$ *(n: number of observations)*
Decision Tree	• Minimum samples split: 2, 5, 15, 35, 50 • Maximum depth: None, 5, 6, 7, 8, 10, 15

Table 5.3 Hyperparameter grid values used for the synthetic datasets (cont.).

Continue on the next page

Method	Hyperparameters
Random Forest	• Number of estimators: 10, 57, 105, 152, 200 • Maximum features: auto, log2 • Maximum depth: 5, 10, None
XGBoost	• Learning rate: 0.05, 0.1, 0.2, 0.3 • Maximum depth: 2, 3, 4, 5, 6 • Number of estimators: 50, 100, 150, 300 • Subsample: 0.6, 0.8, 1.0
XGBRF	• Learning rate: 0.1, 1.0, 1.05, 2.0 • Maximum depth: 1, 3, 5, 6, 7, 8 • Number of estimators: 100, 199, 230, 251, 459, 472 • Subsample: 0.8, 1.0
SVM	• Kernel: Radial basis function (RBF) • Gamma: Scale • Strictness (C): 1, 3, 5 • Epsilon: 0.1, 0.2, 0.3, 1.0 • Maximum number of iterations: None, 200
KNN	• Number of neighbors: 7, 13, 15, 17 • Weights: uniform, distance • Power parameter for the Minkowski metric: 1, 2

Table 5.3 Hyperparameter grid values used for the synthetic datasets (cont.).

Continue on the next page

Method	Hyperparameters
ANN	• Hidden layer sizes: 50, 100, 200 • Activation function: relu • Solver: Adam • Strength of the L2 regularization term ([44]): 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001 • Learning rate: 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001
MARS	• Maximum degree: 1, 2, 3 • Penalty: 3.0

Table 5.3 Hyperparameter grid values used for the synthetic datasets

After obtaining the best hyperparameter combinations with the optimization process, each algorithm is trained on the entire train set using the best parameters of the respective algorithm. Afterward, each model is used for the prediction of the unseen test set and its prediction performance is obtained.

In the following, we present the experimental results. We prefer to give the results on a logarithmic scale rather than a linear scale. In a logarithmic scale, the logarithm of a number is used instead of the number itself. On a linear scale, equal distances between each point show equal differences. On the other hand, on a logarithmic scale, the difference between two values is not equal to the difference between the actual values. For example, the logarithm base 10 of 100 is 2, and the logarithm base 10 of 10000 is 4. The difference between the logarithmic values is 2, but the difference between the actual values is 9900. Because of this, the logarithmic scale allows to display a large range of values more clearly. The results we obtain can contain a large range of values, and to be able to observe the difference, we present them on a logarithmic scale when needed. The captions below each figure indicate whether the scale is logarithmic.

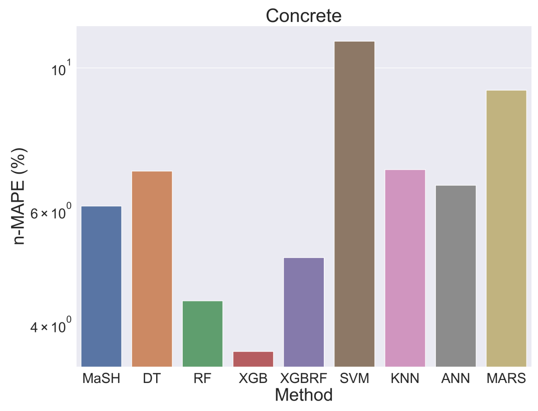


Figure 5.3 Concrete - n-MAPE scores (logarithmic scale)

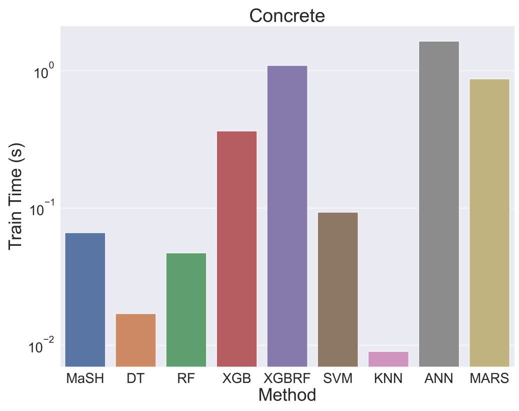


Figure 5.4 Concrete - Train times (logarithmic scale)

Figure 5.3 and Figure 5.4 show the n-MAPE scores and train times of each approach for the Concrete dataset. Both n-MAPE scores and time values are in the logarithmic scale. When n-MAPE values are examined, it can be seen that MaSH shows a competitive performance. Some approaches including XGB, RF, and XGBRF produces better n-MAPE scores but MaSH outperforms decision tree, SVM, KNN, ANN, and MARS. When the time values are examined, KNN, RF, and decision tree seem to be more computationally efficient than MaSH. However, MaSH outperforms the rest with its computation time. Especially, XGB is the top performant algorithm with its n-MAPE score, but the time efficiency falls behind the aforementioned approaches.

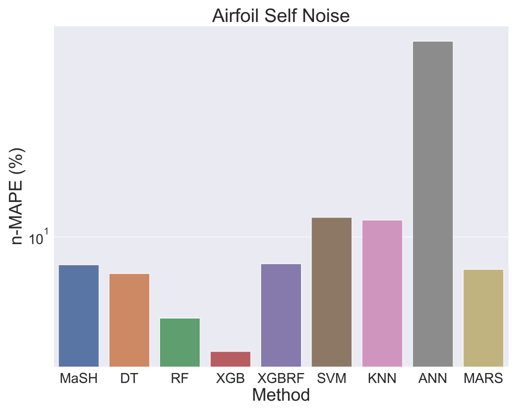


Figure 5.5 Airfoil Self Noise - n-MAPE scores (logarithmic scale)

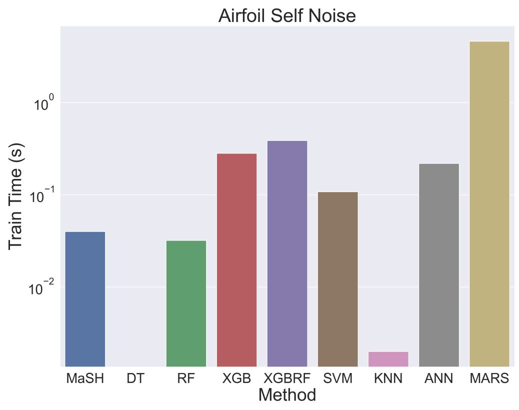


Figure 5.6 Airfoil Self Noise - Train times (logarithmic scale)

Figures 5.5 and 5.6 show the predictive accuracy and time efficiency for the dataset Airfoil Self Noise. As can be observed from Figure 5.5, again, XGBoost appears to be the top performant algorithm, despite its time efficiency is not that good. Decision tree and KNN have the lowest computation times, whereas random forest and MaSH come after them. MARS appears to have the worse computational efficiency. We would like to emphasize that the predictive accuracy of MARS and MaSH seems very similar. However, MaSH is much better in terms of time efficiency.

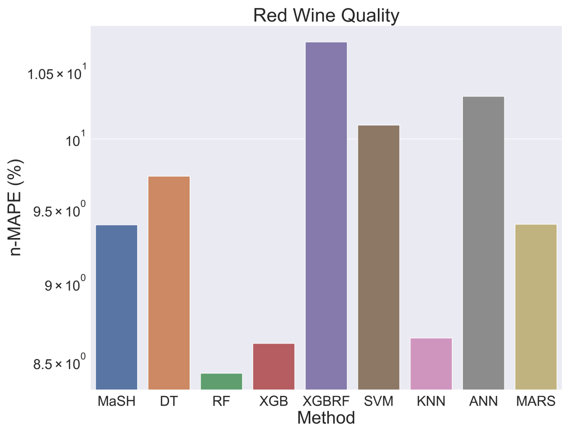


Figure 5.7 Red Wine Quality - n-MAPE scores (logarithmic scale)

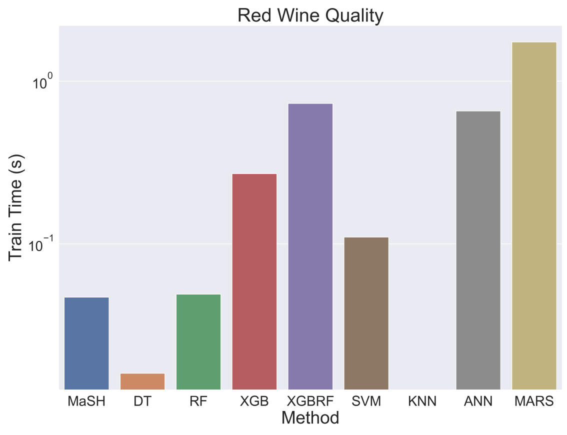
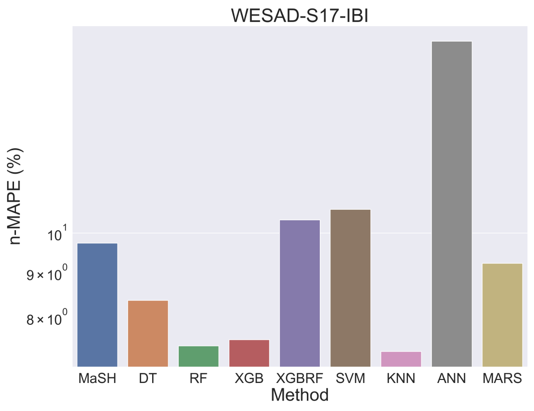


Figure 5.8 Red Wine Quality - Train times (logarithmic scale)

In Figure 5.7, we can see the n-MAPE scores of each method for the Red Wine Quality dataset. Random forest, XGBoost, and KNN are the top approaches with their predictive accuracy where random forest is the best. MaSH and MARS follow the first three. Recall that the values are in the logarithmic scale. For this reason, for instance, the difference between RF and MaSH seems large but RF has an n-MAPE score of approximately 8.5% whereas MaSH has approximately 9.5%. Overall, the range of n-MAPE scores lies between 8% to 11% and all algorithms show a good performance on this dataset. As for time efficiency, we can refer decision tree and KNN as the best algorithms. MaSH comes after them and is better than the rest in terms of time efficiency. Again, we can see that the n-MAPE scores of MaSH and MARS are similar, but the computation time of MaSH is shorter.

We see the WESAD S17-IBI dataset scores in Figures 5.9 and 5.10. For this dataset, KNN is the best algorithm in terms of predictive accuracy and ranks second with its time efficiency,



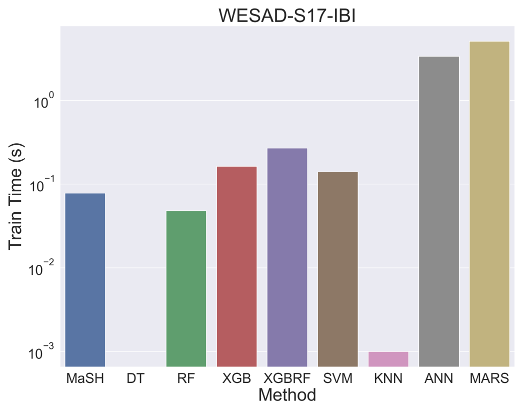
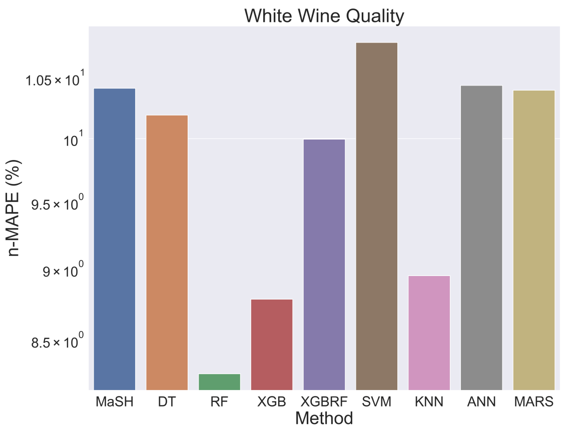


Figure 5.9 WESAD S17-IBI - n-MAPE scores (logarithmic scale)

Figure 5.10 WESAD S17-IBI - Train times (logarithmic scale)

while decision tree is the first. In terms of n-MAPE scores, RF and XGB follow KNN, and decision tree ranks fourth. MARS and MaSH produce close n-MAPE scores, while ANN appears to be the worst. MARS and ANN have the longest execution times.



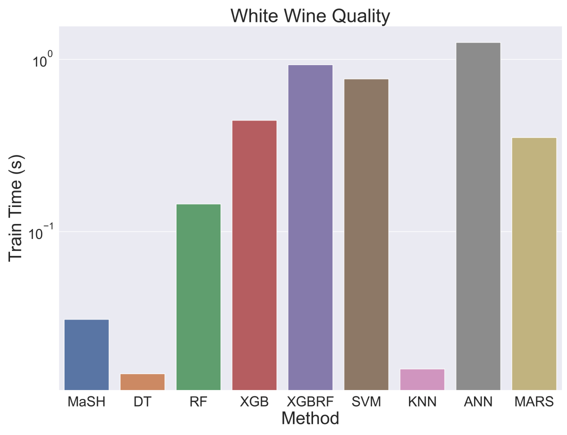


Figure 5.11 White Wine Quality n-MAPE scores (logarithmic scale)

Figure 5.12 White Wine Quality Train times (logarithmic scale)

Figure 5.11 shows the n-MAPE scores for the White Wine Quality dataset. Similar to the Red Wine Quality, all approaches produce close scores. However, for the spirit of comparison, we present the graph on a logarithmic scale to be able to see the differences between them. For this dataset, RF seems as the top-performant algorithm. XGB and KNN rank second and third. Decision tree, MARS, MaSH, and ANN produce close n-MAPE values whereas SVM ranks last. When the computation times are examined from Figure 5.12, we observe that decision tree and KNN are better than the rest and MaSH comes after them. We see that the computational efficiency of MaSH is much better than MARS despite their n-MAPE scores being similar.

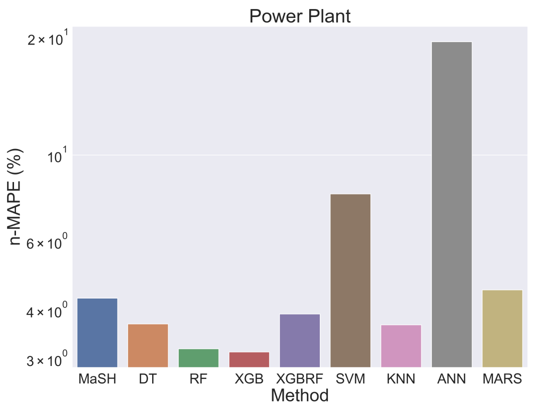


Figure 5.13 Power Plant - n-MAPE scores (logarithmic scale)

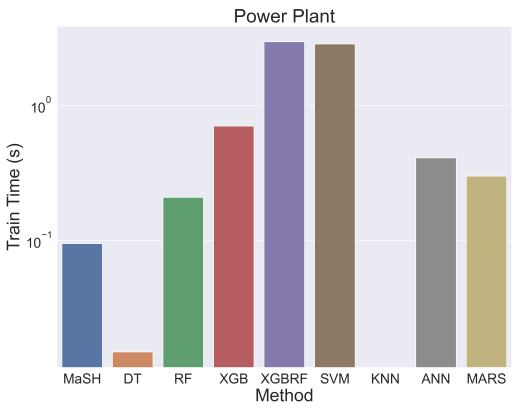


Figure 5.14 Power Plant - Train times (logarithmic scale)

Figure 5.13 and 5.14 show the n-MAPE scores and train times for the Power Plant dataset. We see that XGB and RF outperform others with their n-MAPE values. MaSH shows comparable predictive accuracy and produces scores similar to MARS while outperforming SVM and ANN. The time efficiency of MaSH is better than most approaches, only decision tree and KNN are better.

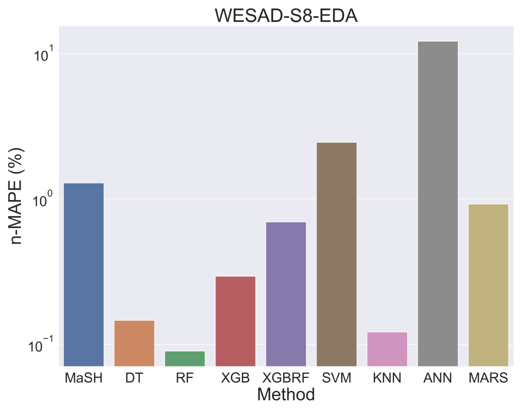


Figure 5.15 WESAD S8-EDA - n-MAPE scores (logarithmic scale)

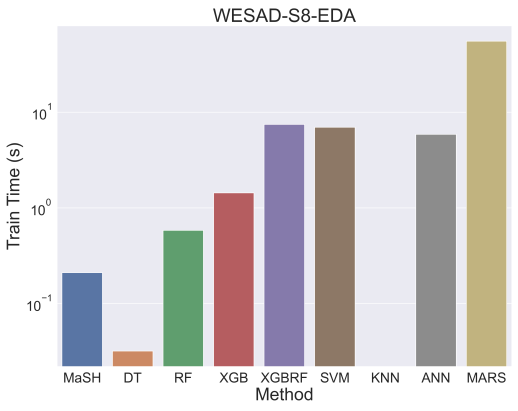


Figure 5.16 WESAD S8-EDA - Train times (logarithmic scale)

n-MAPE and time scores for WESAD S8-EDA dataset are provided in Figures 5.15 and 5.16. Random forest, KNN, and decision tree produce better n-MAPE scores than the rest. XGB and XGBRF come after them. MaSH and MARS produce close n-MAPE scores, but similar to the previous instances, the time efficiency of MaSH is much better than MARS. MaSH is better than most of the instances in terms of time efficiency, while KNN and decision tree are the best.

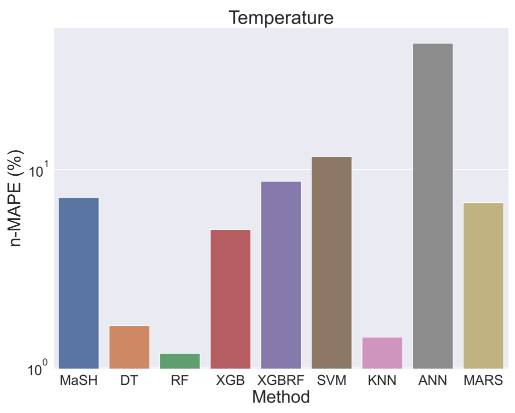


Figure 5.17 Temperature - n-MAPE scores (logarithmic scale)

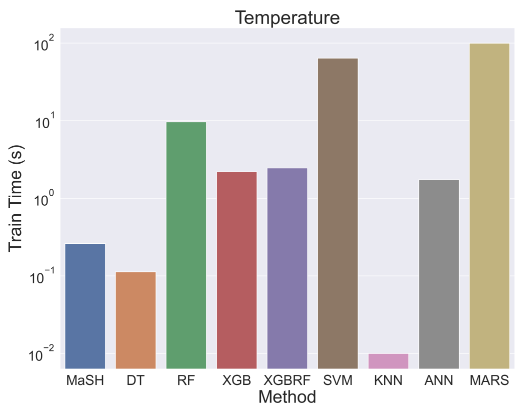


Figure 5.18 Temperature - Train times (logarithmic scale)

We see the n-MAPE scores and time efficiencies for the Temperature dataset in Figure 5.17 and 5.18. We see from Figure 5.17 that MaSH shows a comparable predictive accuracy. Random forest outperforms all approaches, followed by KNN and decision tree. These three approaches show noticeably better predictive performance compared to the others. XGBoost and MARS come after them. MaSH follows XGBoost and MARS, and outperforms XGBRF, SVM, and ANN. We see the time efficiency of each method in Figure 5.18. While KNN and decision tree have better time efficiency than the others, MaSH follows them in the third rank, outperforming the rest.

Figure 5.19 and 5.20 show the scores for the Tetuan Power Consumption dataset. In terms of n-MAPE scores, KNN, random forest, and decision tree are the top three performant algorithms, followed by XGBoost. MaSH, XGBRF, and MARS produce similar n-MAPE scores and outperform SVM and ANN. We see the time efficiency of each algorithm in Figure 5.20. MaSH and KNN outperform the others and decision tree comes after the two.

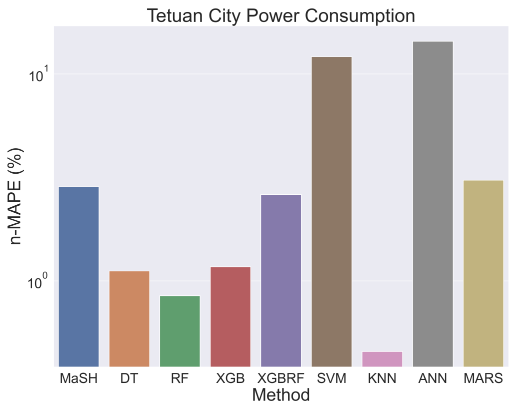


Figure 5.19 Tetuan City Power Consumption - n-MAPE scores (logarithmic scale)

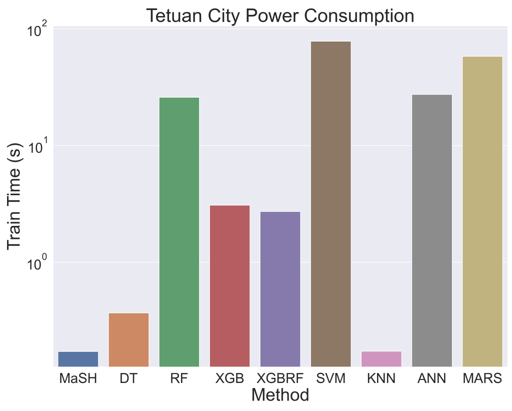


Figure 5.20 Tetuan City Power Consumption - Train times (logarithmic scale)

As in other instances, we still observe a similar view for MaSH and MARS in terms of prediction and computational performance. The n-MAPE scores of the two approaches are very similar, but MaSH is much better than MARS in terms of time efficiency.

We provide the scores for each real-world dataset separately. It can be clearly seen that there is no rule-of-thumb algorithm that works best for all datasets. An approach can be the best in one dataset and worse in another one. Therefore, we provide box and whisker plots (box plots in short) for each algorithm in the following. Box plots show the general performance of an algorithm across all datasets.

The “box” in a box plot shows the interquartile range of the numerical data it displays and represents half of the scores, which are between 25% and 75% of the numerical data. The vertical lines outside of a box are “whiskers”. The lower whisker shows the first 25% and the upper whisker shows the last 25%, which is between 75% and 100% of the numerical data. The end of a lower whisker is the minimum value and the end of an upper whisker is the maximum value. The individual points show the outliers, so these points are significantly different than the other values. The line inside a box show the median of the numerical data. One box plot being shorter than the other indicates that the numerical data it represents contains values close to each other. In terms of the current context, if an algorithm has a shorter box plot, this indicates that it produces consistent results across all datasets.

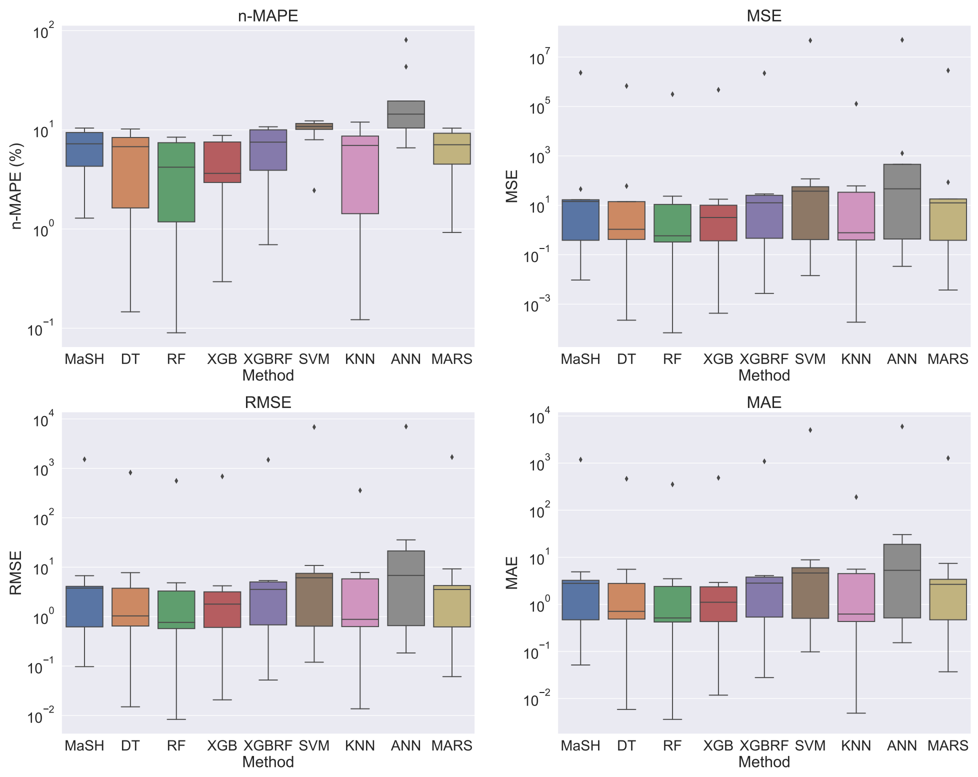


Figure 5.21 Error scores of each method for all real-world datasets (logarithmic scale)
A shorter box plot is better and shows the consistency of an approach.
e.g. for the n-MAPE case, SVM produces consistent scores despite falling behind most approaches.

Figure 5.21 shows the error scores of each method across all the real-world datasets. All error metrics indicate that MaSH remains competitive with the popular machine learning algorithms. It displays an average performance overall and even outperforms SVM and ANN in all metrics. It shows very similar scores to MARS and XGBRF, but MaSH's time efficiency is superior to both. From the figure, we see that MSE, RMSE, and MAE metrics provide a similar picture, whereas n-MAPE values show a distinction between them. Box plots of decision tree, random forest, and KNN become taller. This indicates that these algorithms produce scores with a larger variability across all datasets. Conversely, box plots of SVM and ANN become shorter. Particularly, the length of SVM's box plot shrinks significantly. Hence, we can claim that SVM produces consistent scores despite falling behind most algorithms. Also, MaSH and MARS seem to be more consistent in terms of n-MAPE scores.

To compare the time efficiency of algorithms, train and test processes are repeated five times for each dataset and method, and the average is taken as the result. Time is taken as the sum of the training and prediction times. Computation times for the real-world datasets are demonstrated in Figure 5.24.

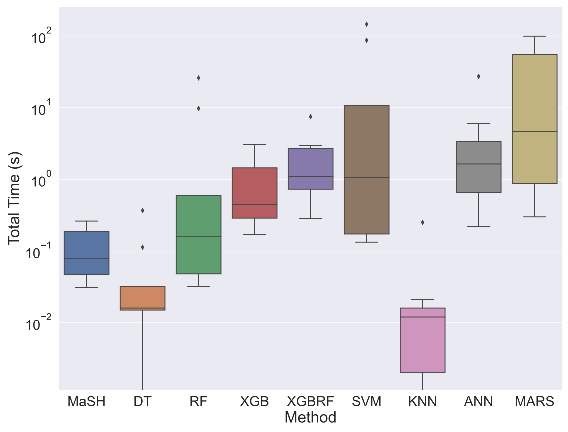


Figure 5.22 Time comparison of the algorithms (logarithmic scale)
A shorter box plot is better.

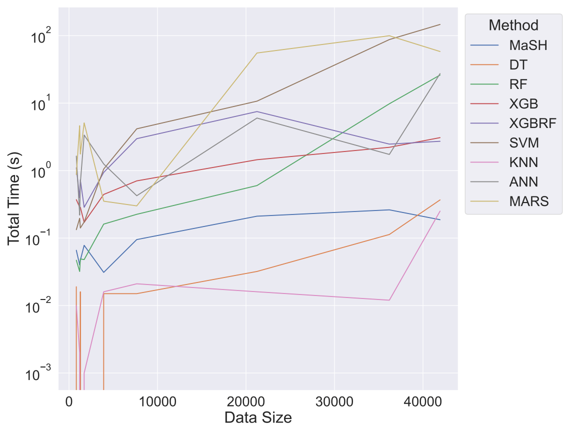


Figure 5.23 Time comparison of the algorithms with respect to data size (logarithmic scale on y-axis)

Figure 5.24 Time comparison for real-world data

Figure 5.22 shows the total time of each algorithm across all datasets. Decision tree and KNN stand as the fastest methods whereas MARS seems to be the slowest. Median values of SVM and ANN are close, but the time efficiency of SVM has more variety across datasets. XGBRF and XGB follow each other whereas the latter has a better time score. Random forest is better on average than XGB, but XGB provides more consistent time scores across datasets. MaSH ranks third and is consistent in producing results within computational efficiency. Figure 5.23 shows the time efficiency of each method with respect to data size. We also see here that MaSH ranks third and its computational efficiency does not increase much as data grows.

Figure 5.25 shows the error scores of each method for all synthetic datasets. All error measures reveal that MaSH surpasses MARS, SVM, KNN, XGBRF and decision tree in

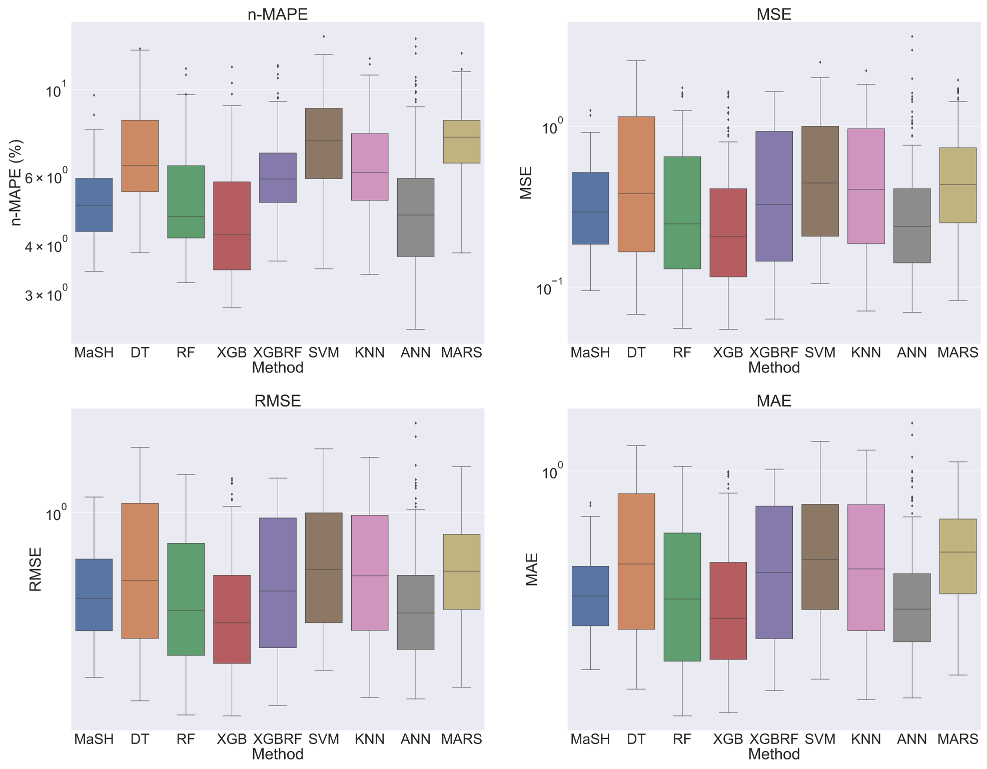


Figure 5.25 Error scores of each method for all synthetic datasets (logarithmic scale)

terms of prediction performance. The median values of random forest and ANN are less than MaSH, but the greatest error that MaSH gives is less than both. XGBoost appears to be producing the best n-MAPE scores on average, whereas ANN have the smallest n-MAPE score. Overall, MaSH produces more consistent scores across datasets.

Figure 5.26 shows the n-MAPE scores of each method according to different data size levels. When we take a look at the n-MAPE scores of MaSH for each data size level, we can observe that its prediction performance remains consistent. Data size does not have a considerable effect on MaSH. When the number of observations is 1000 and 3000, MaSH has the best scores. When data size is 3000, it outperforms decision tree, XGBRF, SVM, KNN, and MARS, and is slightly behind the remaining algorithms. Median n-MAPE scores of MaSH is around 6%. As data grows, the prediction performance of MaSH increases a bit and its

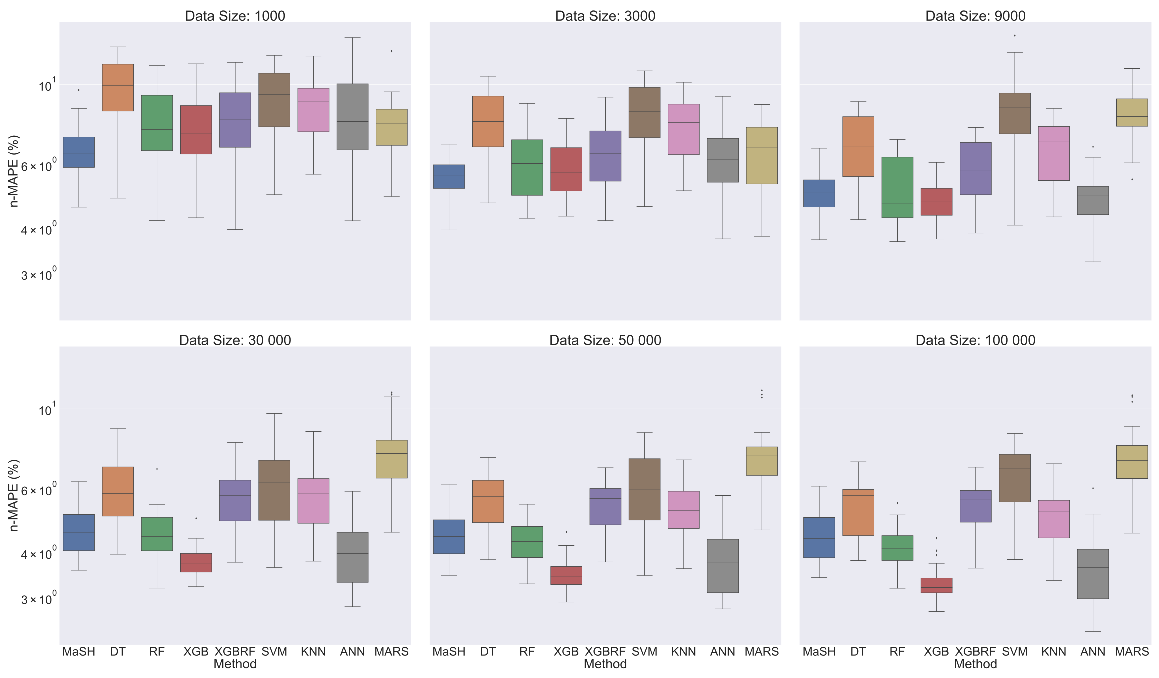


Figure 5.26 n-MAPE scores of each method for synthetic datasets with respect to data size levels (logarithmic scale)

median becomes around 4%. It keeps outperforming MARS, KNN, XGBRF, SVM, and decision tree. We see that XGBoost gets better when the number of observations increase, and ANN follows it closely on average. Random forest and MaSH produce close scores in the largest three data size levels. Overall, we find that MaSH remains competitive, and outperforms all algorithms on the two smallest data sizes 1000 and 3000.

Computational time comparison for synthetic datasets is also performed by repeating training and test processes five times. In this part of the experiment, we impose a 300-second time limit for each algorithm. All algorithms but SVM and MARS provide a result within this time limit. For a total of 162 instances, SVM cannot yield results for 16 instances whereas MARS cannot for 11 instances. Time performance of each algorithm for each data size level is provided in Figure 5.27.

Looking at the time performance of MaSH, we can state that data size has little effect on its time efficiency. For the smallest data size levels, its execution times remain under a second. For the remaining data sizes 30000, 50000, and 100000, the median values across all

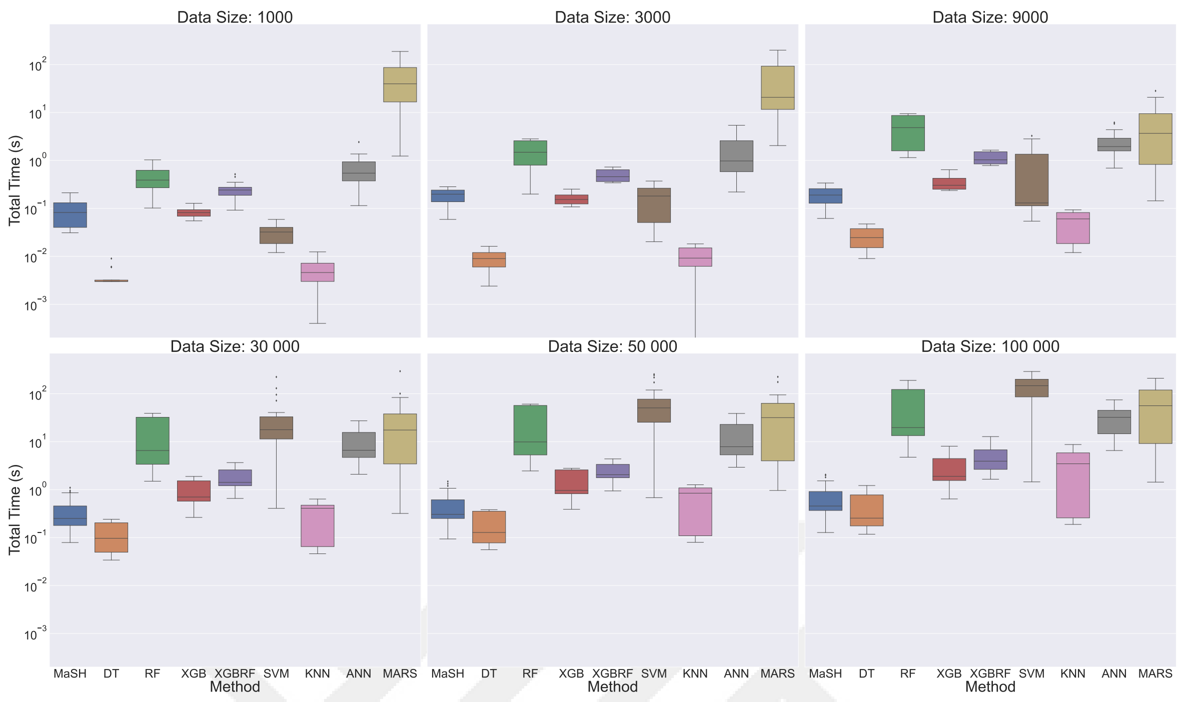


Figure 5.27 Total time scores of each method for synthetic datasets with respect to data size levels (logarithmic scale)

datasets keep remaining under a second while the maximum time value is about 2 seconds. KNN is the fastest method when the number of observations is 1000. However, as data grows, its time efficiency decreases. Random forest also fails to scale with data size. It is moderately good at the smallest data size compared to the others but becomes one of the worst in terms of time efficiency as data grows. SVM displays a similar trend, but data size has more of an impact on SVM. It ranks third when the data size is 1000, but as the number of observations increases, it loses time efficiency and turns out to be the last at the largest data size level. ANN appears to have a low time efficiency even with the smallest data sizes while MARS is the slowest among all. As data grows, ANN catches up with MARS in this regard. Nevertheless, the growth rate of its computation time is less than SVM and random forest. We see a tremendous increase in the computation times of SVM and random forest as data gets larger. XGB and XGBRF follow each other closely for all data size levels, whereas XGB is better in all cases. We see that both keep showing moderately good time performance as data size increases. On average, XGBoost produces time scores very close to MaSH when the number of observations is 1000, but MaSH surpasses it as the data grows larger. Regarding time efficiency, decision tree appears to be superior to all algorithms for most data size levels. In particular, it ranks first at data size levels of 9000, 30000, 50000, and 100000. Nevertheless, we find that the rise in its computation time is greater than MaSH as data grows. Therefore, we can claim that MaSH is the most robust method in terms of time efficiency.

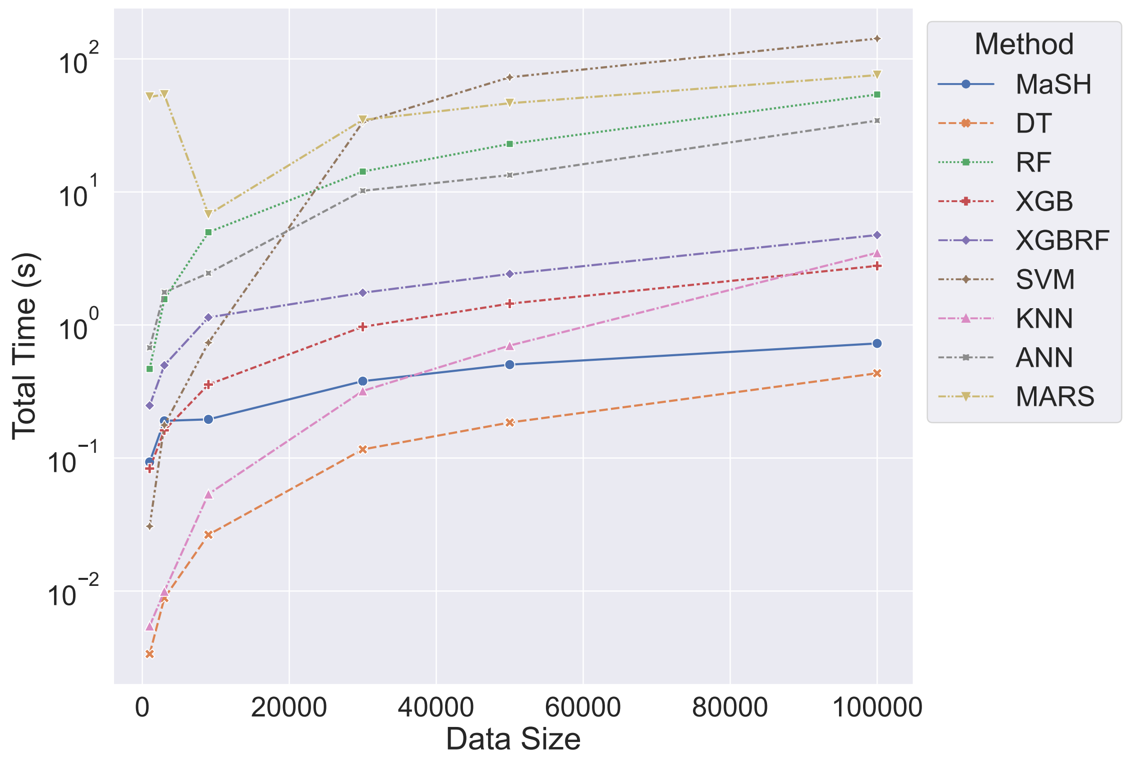


Figure 5.28 Computation time trends of each method at different data size levels (logarithmic scale on y-axis)

Computation time trends for different data size levels are provided in Figure 5.28 as a line plot. The time values in this plot are taken on a logarithmic scale. We can also observe from this figure that MaSH scales well with data size in terms of time efficiency. It is average when the number of observations is smaller and ranks as second at the largest data size. Particularly, we see that decision tree and MaSH converge as data gets larger.

Figure 5.29 shows the log-log plot of computation times for different data size levels, i.e. both time and data size values are in the logarithmic scale. Log-log plots allow comparing

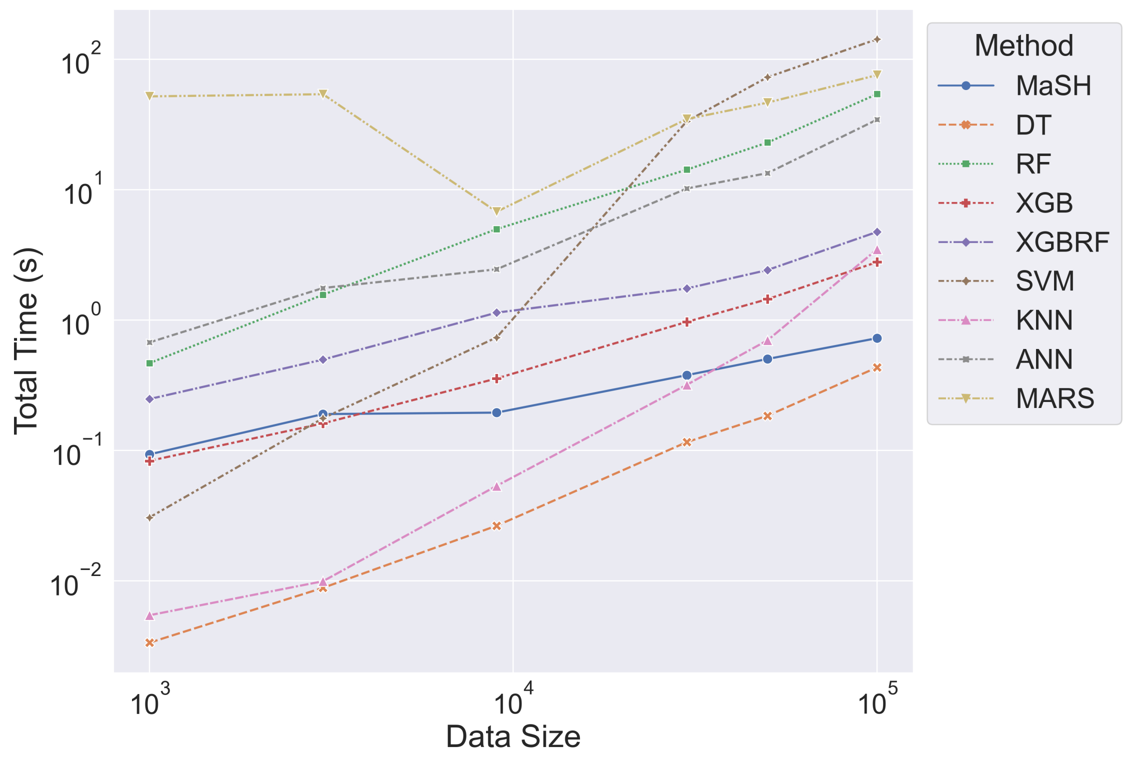


Figure 5.29 Computation time trends of each method at different data size levels (logarithmic scale on x and y axes)

the relative changes between each point on the graph. In Figure 5.29, each point represents the data size levels 1000, 3000, 9000, 30000, 50000, and 100000 respectively. Looking at the graph, we see that the computation times of all approaches increase as data grows, except that the computation times of MARS decrease as the data grows from 3000 to 9000. We see that SVM has the steepest line and it ends up having the worst computation time at the largest data size of 100000. Random forest also has a steep line. Computational efficiency of KNN is at a good level compared to the other approaches but we see that the increase rate of its execution time is high. The increase rate of XGBoost's computation time is almost constant. We see a slightly different slope between data sizes of 1000 and 3000, but we observe a straight line as the data grows larger. XGBoost and XGBRF are similar in terms of the increase rate in computation time when the data sizes are 1000, 3000, and 9000. However, the increase rate of XGBRF is less when data grows from 9000 to 30000. ANN does not show a constant increase rate. Its computation time increases from 9000 to 30000 more than from 30000 to 50000. Decision tree is the top-ranked algorithm for all data size levels. However, its increase rate is higher than that of MaSH. MaSH appears to have the smallest computation time increase rate. The line between data size levels of 3000 to 9000 is almost a horizontal line and even has a negative slope, since the computation time decreases slightly. When data grows from 9000 to 100000, MaSH has a straight line and the slope is small. Therefore, we can restate our claim that MaSH appears to be the most robust approach in terms of computational efficiency.

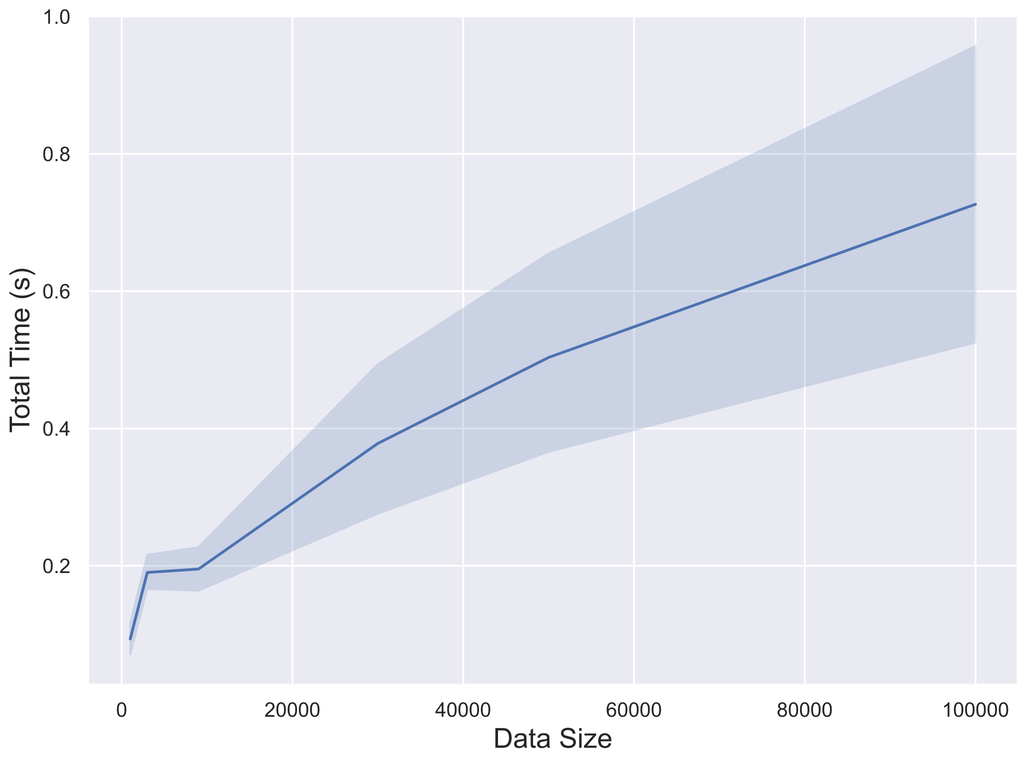


Figure 5.30 Computation times of MaSH for different data size levels

We provide the computation times of MaSH in Figure 5.30. It can be clearly seen that the computation times of MaSH show a smooth increase as data size grows and remains under a second on average.

5.3.2. Comparison with the Column Generation Heuristic

In this section, MaSH is compared with a mixed-integer programming based PLR solution, namely column generation (CG) heuristic presented in [3]. In study [3], prediction scores are not reported, instead, the entire dataset is used in fitting. Therefore, we perform experiments accordingly and use the entire data to compare the fitting performance of MaSH with the CG heuristic. We use synthetic datasets with the number of observations of 1000, 3000, and 9000 in the experiments. We do not use larger datasets in this part of the experiments, because the computation times of the CG heuristic are impractical for such data. The error metric reported in [3] for fitting performance is total absolute error (TAE), so we use the same metric to provide a comparison accordingly. We also compare the computational efficiency of each algorithm. In the study [3], a time limit of 200 seconds is imposed for the experiments. We use a 300-second time limit in the experiments, but we never hit even 200 seconds limit.

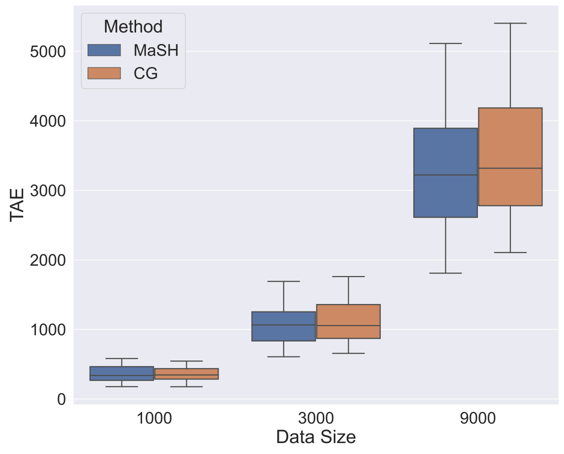


Figure 5.31 Total absolute error scores (TAE) of MaSH and CG for different data size levels

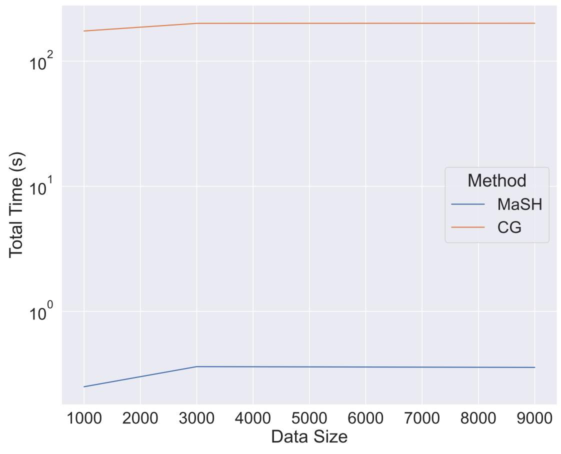


Figure 5.32 Computation times of MaSH and CG for different data size levels (logarithmic scale on y-axis)

Figure 5.31 shows the TAE scores of MaSH and CG for each data size level. When the number of observations is 1000, CG is barely better than MaSH. For the remaining two data size levels, we can clearly observe that the proposed MaSH algorithm outperforms the CG heuristic. Column generation is a mathematical programming based heuristic. Therefore, it is expected to yield an optimal solution, and it does when the data size is 1000. However, when the number of observations increases, it cannot further converge the optimal value within the given time limit. The surprising result we found is the out-performance of MaSH in the other two cases. Because we initially aim to solve PLR problem in a computationally efficient manner by compromising some accuracy. Nevertheless, the results indicate that MaSH outperforms CG in terms of fitting accuracy with efficient computation times. From Figure 5.32, we clearly observe that MaSH outperforms CG in a much more efficient way. Note that CG hits the 200 seconds boundary after the number of observations gets greater than 3000. Therefore, the actual computation times would be even more.

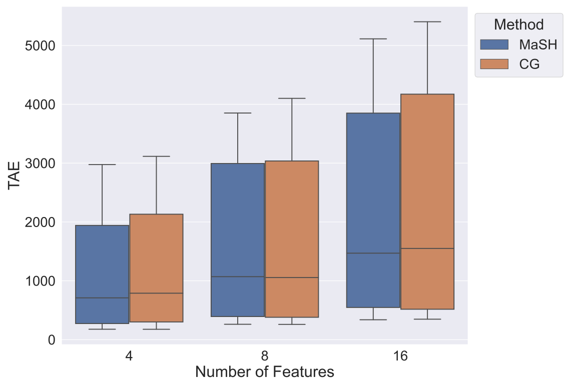


Figure 5.33 TAE vs. number of features

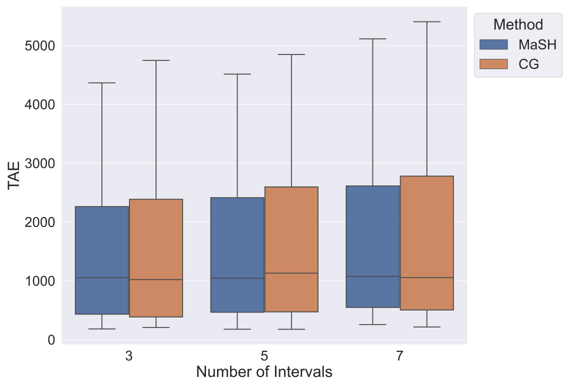


Figure 5.34 TAE vs. number of intervals

We provide the comparison of MaSH and CG with respect to different number of feature and number of interval levels in Figure 5.33 and Figure 5.34. Figures demonstrate that MaSH and CG follow a similar trend in all instances. Their median values are very close to each other and there are slight differences between them without any particular order. We can claim that MaSH is better than CG heuristic in that its worst scores are less than CG.

6. CONCLUSION

In this thesis, piecewise linear regression problem is addressed. Particularly, the PLR problem defined in [5] and further studied in [3] is adopted. In this PLR problem definition, multivariate datasets are considered where data is partitioned into intervals using a known feature and each interval is represented by a distinct multivariate linear regression. In the literature, there exist several studies that address the PLR problem and solve it with different approaches. We present the studies that exist in the literature by classifying them into three groups: solutions adopting dynamic programming, solutions adopting mathematical programming methodologies, and solutions adopting heuristic approaches. Some studies may present a solution by combining these approaches. In this case, we classify them according to the primary methodology presented in the paper. The work in this thesis falls in category three. A heuristic algorithm, namely merge-and-split heuristic (MaSH), is proposed in this study, where the main aim is to be able to handle even larger datasets without sacrificing computational efficiency within acceptable prediction/fitting scores. Unlike most of the studies present in the literature, MaSH does not require a predefined number of intervals.

We compare the computational efficiency and prediction performance of MaSH with well-known machine learning algorithms including decision tree, random forest, XGBoost, XGBoost with random forest learners, SVM, KNN, ANN, and MARS. In the experiments, we use real-world and synthetic datasets and show that MaSH remains competitive for real-world datasets, and even outperforms SVM and ANN. Also, it provides very similar scores to MARS, yet MaSH has much better computational efficiency. For synthetic datasets, we show that MaSH outperforms most of the experimented machine learning algorithms, and competes with the rest. We also provide a comparison with respect to different data size levels. We find that MaSH remains robust in terms of prediction performance across different data sizes. As for the comparison of computational efficiency, we show that MaSH scales well even for larger datasets. It shows a smooth increase as data grows and its execution time is in milliseconds even for the largest data sizes.

We also present a comparison of MaSH with the column generation (CG) ([3]) heuristic which is a mixed-integer linear programming (MILP) based PLR solution. We compare the fitting performance of MaSH against CG and show that MaSH is better in most cases. This is to our surprise because initially we aim for a computationally efficient solution, and presume this will compromise some accuracy. However, the results show that MaSH performs well in terms of fitting efficiency as well as time efficiency. We claim this by examining the smallest data size. CG cannot converge well for larger datasets, but it converges well when data is smaller. Therefore, we find that the fitting performance of MaSH is quite well since it outputs nearly the same scores as CG in the smallest data sizes. Besides, it finds the solution in very short execution times. We do not provide a time efficiency comparison with CG because it is obvious that MaSH is much faster than CG which uses MILP.

Overall, when the results are examined, we find the proposed heuristic very promising. First, its computational efficiency is really at a good level. Therefore, we achieve our goal of solving PLR for much larger datasets in computationally practical times. Second, it demonstrates good prediction and fitting accuracy. We provide scores in different error metrics and compare it to a number of algorithms including various machine learning approaches and a mathematical programming based PLR heuristic. Third, we would like to draw attention that interpretability is a trending concept in machine learning. Among the aforementioned machine learning algorithms, only decision tree and MARS are considered as interpretable algorithms. We show that MaSH is far better than both when the prediction and computation efficiency results are considered together. Consequently, MaSH is a promising approach with the interpretability it provides.

For future research, MaSH can be used as an acceleration tool and used with sophisticated methods to provide an initial solution to reach the optimal solution faster. Also, the single partition feature can be selected in the algorithm itself. Moreover, more than one partition feature can be used to separate data points in multidimensional space. In addition to all of that, hyperparameters of the proposed solution can be selected dynamically rather than one-time fixed parameters. This dynamic approach would be according to an iteration count or would be selected in a data-specific manner.

REFERENCES

[1] Vito Muggeo. Estimating regression models with unknown break-points. *Statistics in medicine*, 22:3055–71, **2003**. doi:10.1002/sim.1545.

[2] Judith Toms and Mary Lesperance. Piecewise regression: A tool for identifying ecological thresholds. *Ecology*, 84:2034–2041, **2003**. doi:10.1890/02-0472.

[3] Huseyin Tunc and Burkay Genç. A column generation based heuristic algorithm for piecewise linear regression. *Expert Systems with Applications*, 171:114539, **2021**. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2020.114539.

[4] Jushan Bai and Pierre Perron. Computation and analysis of multiple structural change models. *Journal of Applied Econometrics*, 18(1):1–22, **2003**. doi:10.1002/jae.659.

[5] Lingjian Yang, Songsong Liu, Sophia Tsoka, and Lazaros G. Papageorgiou. Mathematical programming for piecewise linear regression analysis. *Expert Systems with Applications*, 44:156–167, **2016**. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2015.08.034.

[6] Chi-Jen Wu, Wei-Sheng Zeng, and Jan-Ming Ho. Optimal segmented linear regression for financial time series segmentation. pages 623–630. **2021**. doi:10.1109/ICDMW53433.2021.00082.

[7] Evimaria Terzi and Panayiotis Tsaparas. Efficient algorithms for sequence segmentation. volume 2006. **2006**. doi:10.1137/1.9781611972764.28.

[8] Ilias Diakonikolas, Jerry Zheng Li, and Anastasia Voloshinov. Efficient algorithms for multidimensional segmented regression. *ArXiv*, abs/2003.11086, **2020**.

[9] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23(1), **2021**. ISSN 1099-4300. doi:10.3390/e23010018.

[10] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, **2017**. doi:10.48550/ARXIV.1702.08608.

[11] Been Kim, Rajiv Khanna, and Oluwasanmi Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2288–2296. Curran Associates Inc., Red Hook, NY, USA, **2016**. ISBN 9781510838819.

[12] Rong Liu, Feng Mai, Zhe Shan, and Ying Wu. Predicting shareholder litigation on insider trading from financial text: An interpretable deep learning approach. *Information & Management*, 57(8):103387, **2020**. ISSN 0378-7206. doi:https://doi.org/10.1016/j.im.2020.103387.

[13] Yimou Li, Zachary Simon, and David Turkington. Investable and interpretable machine learning for equities. *The Journal of Financial Data Science*, 4(1):54–74, **2022**. ISSN 2640-3943. doi:10.3905/jfds.2021.1.084.

[14] Yu-En Chang and Hsun-Ping Hsieh. An interpretable deep learning framework for assessing financial potential of urban spaces. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '21, page 678–679. Association for Computing Machinery, New York, NY, USA, **2021**. ISBN 9781450386647. doi:10.1145/3474717.3486810.

[15] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. A holistic approach to interpretability in financial lending: Models, visualizations, and summary-explanations. *Decision Support Systems*, 152:113647, **2022**. ISSN 0167-9236. doi:https://doi.org/10.1016/j.dss.2021.113647.

[16] Markus Jaeger, Stephan Krügel, Dimitri Marinelli, Jochen Papenbrock, and Peter Schwendner. Interpretable machine learning for diversified portfolio

construction. *The Journal of Financial Data Science*, 3(3):31–51, **2021**. ISSN 2640-3943. doi:10.3905/jfds.2021.1.066.

[17] Ribana Roscher, Bastian Bohn, Marco F. Duarte, and Jochen Garcke. Explainable machine learning for scientific insights and discoveries. *IEEE Access*, 8:42200–42216, **2020**. doi:10.1109/ACCESS.2020.2976199.

[18] Yawen Li, Liu Yang, Bohan Yang, Ning Wang, and Tian Wu. Application of interpretable machine learning models for the intelligent decision. *Neurocomputing*, 333:273–283, **2019**. ISSN 0925-2312. doi:https://doi.org/10.1016/j.neucom.2018.12.012.

[19] Alfredo Vellido. The importance of interpretability and visualization in machine learning for applications in medicine and health care. *Neural Computing and Applications*, 32:1–15, **2020**. doi:10.1007/s00521-019-04051-w.

[20] Gregor Stiglic, Primoz Kocbek, Nino Fijacko, Marinka Zitnik, Katrien Verbert, and Leona Cilar. Interpretability of machine learning-based prediction models in healthcare. *WIREs Data Mining and Knowledge Discovery*, 10(5), **2020**. doi:10.1002/widm.1379.

[21] Khansa Rasheed, Adnan Qayyum, Mohammed Ghaly, Ala Al-Fuqaha, Adeel Razi, and Junaid Qadir. Explainable, trustworthy, and ethical machine learning for healthcare: A survey. *Computers in Biology and Medicine*, 149:106043, **2022**. ISSN 0010-4825. doi:https://doi.org/10.1016/j.compbiomed.2022.106043.

[22] Muhammad Aurangzeb Ahmad, Carly Eckert, and Ankur Teredesai. Interpretable machine learning in healthcare. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '18, page 559–560. Association for Computing Machinery, New York, NY, USA, **2018**. ISBN 9781450357944. doi:10.1145/3233547.3233667.

[23] Inna Kolyshkina and Simeon Simoff. Interpretability of machine learning solutions in public healthcare: The crisp-ml approach. *Frontiers in Big Data*, 4, **2021**. ISSN 2624-909X. doi:10.3389/fdata.2021.660206.

[24] Richard Bellman and Robert Roth. Curve fitting by segmented straight lines. *Journal of the American Statistical Association*, 64(327):1079–1084, **1969**. ISSN 01621459.

[25] Scott B. Guthery. Partition regression. *Journal of the American Statistical Association*, 69(348):945–947, **1974**. doi:10.1080/01621459.1974.10480233.

[26] Douglas M. Hawkins. Point estimation of the parameters of piecewise regression models. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 25(1):51–57, **1976**. ISSN 00359254, 14679876.

[27] Günter Rote. Isotonic Regression by Dynamic Programming. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, **2018**. ISBN 978-3-95977-099-6. ISSN 2190-6807. doi:10.4230/OASIcs.SOSA.2019.1.

[28] Dimitris Bertsimas and Romy Shioda. Classification and regression via integer optimization. *Operations Research*, 55:252–271, **2007**. doi:10.1287/opre.1060.0360.

[29] Claudia D'Ambrosio, Andrea Lodi, and Silvano Martello. Piecewise linear approximation of functions of two variables in milp models. *Operations Research Letters*, 38(1):39–46, **2010**. ISSN 0167-6377. doi:https://doi.org/10.1016/j.orl.2009.09.005.

[30] Alejandro Toriello and Juan Pablo Vielma. Fitting piecewise linear continuous functions. *European Journal of Operational Research*, 219(1):86–95, **2012**. ISSN 0377-2217. doi:https://doi.org/10.1016/j.ejor.2011.12.030.

[31] Ioannis Gkioulekas and Lazaros G. Papageorgiou. Piecewise regression analysis through information criteria using mathematical programming. *Expert Systems with Applications*, 121:362–372, **2019**. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2018.12.013.

[32] Karthick Gopalswamy, Yahya Fathi, and Reha Uzsoy. Valid inequalities for concave piecewise linear regression. *Operations Research Letters*, 47(1):52–58, **2019**. ISSN 0167-6377. doi:https://doi.org/10.1016/j.orl.2018.12.004.

[33] Steffen Rebennack and Vitaliy Krasko. Piecewise linear function fitting via mixed-integer linear programming. *INFORMS Journal on Computing*, 32(2):507–530, **2020**. doi:10.1287/ijoc.2019.0890.

[34] John Alasdair Warwicker and Steffen Rebennack. Generating optimal robust continuous piecewise linear regression with outliers through combinatorial benders decomposition. *IISE Transactions*, 0(0):1–13, **2022**. doi:10.1080/24725854.2022.2107249.

[35] Alessandro Magnani and Stephen Boyd. Convex piecewise-linear fitting. *Optimization and Engineering*, 10:1–17, **2009**. doi:10.1007/s11081-008-9045-3.

[36] Jayadev Acharya, Ilias Diakonikolas, Jerry Li, and Ludwig Schmidt. Fast algorithms for segmented regression. *CoRR*, abs/1607.03990, **2016**.

[37] I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, **1998**. ISSN 0008-8846. doi:https://doi.org/10.1016/S0008-8846(98)00165-3.

[38] Thomas Brooks, D.S. Pope, and Michael Marcolini. Airfoil Self-Noise. UCI Machine Learning Repository, **2014**. Accessed: 2022-10-30.

[39] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, **2009**. ISSN 0167-9236. doi:https:

//doi.org/10.1016/j.dss.2009.05.016. Smart Business Networks: Concepts and Empirical Evidence.

[40] Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, **2014**. ISSN 0142-0615. doi:https://doi.org/10.1016/j.ijepes.2014.02.027.

[41] Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger, and Kristof Van Laerhoven. Introducing wesad, a multimodal dataset for wearable stress and affect detection. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction*, ICMI '18, page 400–408. Association for Computing Machinery, New York, NY, USA, **2018**. ISBN 9781450356923. doi:10.1145/3242969.3242985.

[42] Abdulwahed Salam and Abdelaaziz El Hibaoui. Comparison of machine learning algorithms for the power consumption prediction : - case study of tetouan city –. In *2018 6th International Renewable and Sustainable Energy Conference (IRSEC)*, pages 1–5. **2018**. doi:10.1109/IRSEC.2018.8703007.

[43] Historical Hourly Weather Data 2012-2017. `https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data`. Accessed: 2022-10-30.

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