
Faculty of
Mathematics
and Informatics

VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
MODELLING AND DATA ANALYSIS
MASTER'S STUDY PROGRAMME

# Prediction using functional regression tree

**Master's thesis**

Author: Miglė Kavaliauskaitė
VU email address: migle.kavaliauskaite@mif.stud.vu.lt
Supervisor: Assoc. Prof., Dr. Jurgita Markevičiūtė

Vilnius

2021

# Abstract

Nowadays, the main goal of many models for data analysis is to predict the results as accurately as possible. Machine learning methods help to achieve better results. However, only a few methods are adapted for using functional data. One of them is the Regression Energy Tree. This regression tree using framework of decision tree where the testing procedure is implemented using energy statistics such as distance correlation. This paper deals with scalar-on-function regression where the regressors are curves and the response variable is scalar. In this work, the Regression Energy Tree is complemented by different resampling techniques such as bagging, repeated training, and k-fold cross validation. Tests have shown that using any of the resampling techniques presented in the paper, the model predicts more accurately than without using resampling techniques.



**Keywords:** Regression Energy Tree, functional data, resampling techniques, bagging, repeated training, k-fold cross validation.



# Santrauka

Šiais laikais daugelio modelių, skirtų duomenų analizei, pagrindinis tikslas yra kuo tiksliau numatyti rezultatus. Mašininio mokymosi metodai yra būdai, kuriuos naudojant galima pasiekti geresnių rezultatų. Tačiau tik keletas metodų pritaikyti naudojant funkcinius duomenis. Vienas iš jų yra regresinis energijos medis. Šis regresinis energijos medis remiasi sprendimų medžio sistema, kuriame testavimo procedūra įgyvendinama naudojant energijos statistiką, pvz., atstumo koreliaciją.Šiame darbe nagrinėjama skaliarinė regresija, kai regresoriai yra kreivės, o atsako kintamasis yra skaliarinis. Šiame darbe regresijos energijos medis yra sujungiamas su įvairiais pakartotinio atrinkimo metodais, tokiais kaip maišymas, pakartotinis mokymas ir k-karto kryžminis patvirtinimas. Eksperimentai parodė, kad naudojant bet kurį iš darbe pateiktų pakartotinio atrinkimo metodų, modelis prognozuoja tiksliau nei nenaudojant šių metodų.



**Keywords:** Regresinins energijos medis, funkciniai duomenys, perrinkimo technikos, maišymas, pakartotinis mokymas, k-karto kryžminis patvirtinimas.

# Contents

# 1 Introduction

These days not only simple models but also more complex ones are used to analyze the data. A lot of model has the main goal to predict the results as accurately as possible. Machine learning methods are used to give the highest possible accuracy. These algorithms create a model that is based on sample data or otherwise training data to predict solutions without being specifically programmed. Machine learning algorithms are used to address a wide range of issues: from medicine to language recognition. Moreover, in modern statistics , the models are being built for more complicated data, such as functional data.

Functional data analysis deals with the the data that are in the form of functions, images or shapes. The atom of functional data is a function, where for each subject in a random sample functions are recorded. The data composed of curves that vary over a continuum, such as time. The curves of the functional data are usually interrelated, which means that the measurement at a point, such as $t_{i+1}$, depends on the measurements at some other points, such as $t_1, ..., t_i; i \in N$. The functional data typically consists of a random sample of independent functions, $X_1(t), ..., X_n(t)$, on a compact interval $I = [0, T]$ on the real line. These functions can be viewed as the realizations of a one-dimensional stochastic process, often assumed to be in a Hilbert space, such as $L^2(I)$. Here a stochastic process $X(t)$ is said to be an $L^2$ process if and only if it satisfies $E(\int_I X^2(t)dt) < \infty$. [15]

However, there are very few machine learning methods adapted to functional data. Several regression methods have been developed for simple functional data. It could be scalar responses and functional predictors, functional responses and scalar predictors and functional responses and functional predictors. This paper deals with the first case where the response variable is scalar and the covariates are functions.

The aim of this work is to extend the regression energy tree for functional data using different resampling techniques.

In the present paper, we deal with the existing method - Regression Energy tree. This method is prepared by Marco Brandi.[1] This regression tree using framework of decision tree where the testing procedure is implemented using energy statistics such as distance correlation. The first step in this method is transform and smooth data. Then we take a global test if there is association between response variable Y and the functional transformed and smooth covariates $\hat{f}^J(t)$. After that, we choose the functional covariates associated the minimum adjusted P-value using Bonferroni correction conditionally that is less than $\alpha$. Next step is partition data for the observed coefficient value that minimazes the variances of Y in the subsets and repeat all this procedure on the two obtained subsets. We extend this method using resampling techniques. First technique which we use is bagging. Bagging is an ensemble method that improves the accuracy of unstable models by averaging a set of the same model fit to bootstrapped samples of a feature space. In bagging, a random sample of data in a training set is selected with

replacement—meaning that the individual data points can be chosen more than once. A bagged estimator is created by aggregating m regression energy trees. Another resampling technique is considered in the work - K-Fold Cross-Validation. For K-Fold cross-validation, data are subdivided into k subsets. The maintenance method is now repeated k times so that each time one of the k subsets is used as a test kit and the other k-1 subsets are combined to form a training kit. The third technique is called repeated training. We take a training set from our observations m times and train the model.

The present work is organized as follows. In first part of this work the is introduction to functional data and how to manage it. The second part in more detail the regression energy tree and resampling techniques used in this work: bagging, k-fold cross validation and repeated training. The third part presents the results of the practical part using the generated and real data.

# 2 Functional data

## 2.1 Introduction to functional data

Functional Data Analysis (FDA) has expanded rapidly in recent years and has been applied in a variety of fields. This diversity makes it difficult to define exactly what the FDA is. However, it can be assumed that the FDA may already be present when one of the variables of interest in the dataset can be considered as a smooth curve or function.

Functional data was already discussed in the works of Grenander [4] and Karhunen [5] in the twentieth century. In their works, they spoke of the division of the square-integrated continuous time stochastic process into its own components, now known as the Karhunen-Loève decomposition. One of the main and largest contributions to the analysis of functional data was made by J.O. Ramsey and B.W. Silwerman, who developed the concept of functional data. [13] In their book "Functional data analysis", they introduce several functional features of the data, such as data representation, descriptive statistics, smoothing methods and dimensional reduction. Also, they provide that "the objectives of functional data analysis:

- to represent the data in ways that aid further analysis;
- to display the data so as to highlight various characteristics;
- to study important sources of pattern and variation among the data;
- to explain variation in an outcome or dependent variable by using input or independent variable information;
- to compare two or more sets of data with respect to certain types of variation, where two sets of data can contain different functions for a common set of replicates."[13]

In another paper, J.O. Ramsay along with C.J. Dalzell [12] singled out the reasons why it is useful to examine the data from a functional point of view. "There are four of them:

- Functional observations are increasingly presented in applicable contexts as automated data collection tools become available to more researchers. In addition, smoothing and interpolation procedures can provide functional representations with unlimited sets of observations.
- Some modeling problems are more natural to consider functional conditions, although only a finite number of observations are possible.
- The purposes of the analysis may be functional values, as would be the case if the completed data were used to estimate the values of the whole function, its derivatives or other functional functions.

- Consideration of such an analysis may have significant implications for their analysis." [12]

Functional data usually consists of a sample of n functions $X_1(t), ..., X_n(t)$ that are independent real valued and $I = [0, T]$. In most cases, these functions correspond to the smooth realization of the basic stochastic process. Because of this aspect, that functions are seen as realizations of a one-dimensional stochastic process, it is supposed that it is in Hilbert space $L^2(I)$. This means that the stochastic process $X(t)$ is an $L^2$ process if and only if it satisfies the condition $E(\int_I X^2(t)dt) < \infty$. [15]

In real data applications, the underlying process often cannot be monitored directly because data can be collected over time in a discrete fixed or random time grid. Therefore, in such situations, the process is considered latent.

In the simple case, the functional data are considered as examples of fully observable trajectories. The general assumption for functional data is that they are written to the same dense time grid composed of consecutive times $t_1, ..., t_p$ for all n entities.

## 2.2 Summary statistics for functional data

All standard summary statistics for univariate date we can apply to functional data as well. The first function, which is well known to all, is the mean. To calculate the mean for functional data, we look for the mean with a function that looks like this

$$\bar{x}(t) = N^{-1} \sum_{i=1}^{N} x_i(t).$$

This is the average of the functions at the point during replications.

The value of the variance function is calculated in a similar aspect. The variance function is

$$Var_x(t) = (N-1)^{-1} \sum_{i=1}^{N} [x_i(t) - \bar{x}(t)]^2. \tag{1}$$

Then the standard deviation function is

$$\sigma = \sqrt{Var_x(t)} = \sqrt{(N-1)^{-1} \sum_{i=1}^{N} [x_i(t) - \bar{x}(t)]^2}. \tag{2}$$

In the case of functional data, we can also find the meaning of covariance. The covariance function is

$$cov_x(t_1, t_2) = (N-1)^{-1} \sum_{i=1}^{N} (x_i(t_1) - \bar{x}(t_1))(x_i(t_2) - \bar{x}(t_2)), \tag{3}$$

which summarizes the dependence on the different meanings of the arguments and is computed for all $t_1$ and $t_2$.

Then the correlation function is

$$corr_x(t_1, t_2) = \frac{cov_x(t_1, t_2)}{\sqrt{var_x(t_1)var_x(t_2)}}. \tag{4}$$

## 2.3 Functional regression models

Linear regression is one of the main statistical tools. Typically, a linear model looks like:

$$y_i = x_{i1}\beta_1 + x_{i2}\beta_2 + ... + x_{ip}\beta_p + \epsilon_i, i = 1, 2, ..., N, \tag{5}$$

where all variables and parameters are scalars. In the functional linear model, it appears that some of these quantities are functions. These models distributed into three categories: it could be scalar responses and functional predictors, functional responses and scalar predictors and functional responses and functional predictors. [7]

Scalar-on-function regression:

$$Y_i = \int \beta(s)X_i(s)ds + \epsilon_i. \tag{6}$$

In this case, the regressors are curves, but the responses are scalars. The parameter is the regression function $\beta$.

Function-on-scalar regression:

$$Y_i(t) = \sum_{k=1}^{p} x_{ik}\beta_k(t) + \epsilon_i(t). \tag{7}$$

The responses are curves, but the regressors are scalars. The parameters are the coefficient functions $\beta_k$.

The function-on-function regression:

$$Y_i(t) = \int \beta(t, s)X_i(s)ds + \epsilon_i(t). \tag{8}$$

The responses $Y_i$ and the regressors $X_i$ are curves. The parameter is the kernel (surface) $\beta$.

Most importantly, the functional parameters are in small-sized objects to be evaluated from a short sample. Without any constraints on the functional parameters, a perfect t is possible, and the resulting estimates are volatile, noisy functions that do not provide useful insight. Therefore, the assessment often involves additional restrictions, setting conditions for the smoothness or restricting the actions of the operators concerned in the relevant sub-operations.

### 2.3.1 Scalar-on-function regression

In this work, we examine only the case of scalar-on-function regression, so we will present it in more detail.

In order to increase the practical applicability, we consider the model 6 with an intercept term, i.e.

$$Y_i = \alpha + \int \beta(s) X_i(s) ds + \varepsilon, \qquad i = 1, ..., N. \tag{9}$$

The simplest way is to extend the function using deterministic basis functions. Assume, that

$$\beta(t) = \sum_{k=1}^{K} c_k B_k(t). \tag{10}$$

The number K is usually chosen to be less than the number of time points $t_j$ at which the functions $X_i$ are observed.

Using expansion of $\beta$, we obtain

$$\int \beta(t) X_i(t) dt = \sum_{k=1}^{K} c_k \int B_k(t) X_i(t) dt = \sum_{k=1}^{K} x_{ik} c_k.$$

After that, model reduces to

$$Y_i = \alpha + \sum_{k=1}^{K} x_{ik} c_k + \varepsilon_i, \qquad i = 1, ..., N. \tag{11}$$

The parameter vector $c = [\alpha, c_1, ..., c_K]^T$ is estimated by

$$\hat{c} = (X^T X)^{-1} X^T Y,$$

where

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nk} \end{bmatrix} \qquad Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix}.$$

Kokoszka and Reimherr provides that the disadvantage of this method is that the resulting estimate

$$\hat{\beta}(t) = \sum_{k=1}^{K} \hat{c}_k B_k(t)$$

depends on the shape of basis functions $B_k$ and on their number K, both being subjectively chosen. A reasonable approach is to use the same basis functions as expansions of the regressor

functions. Furthermore, the theory supporting this estimation procedure, assumes that $\beta$ is exactly equal to a linear combination of the basis functions. [7]

Reducing to the standard model also results in confidence intervals for approximately unknown regression function $\beta$.

We then know that if the errors $\varepsilon_i$ are independent $N(0; \sigma_\varepsilon^2)$, then the covariance matrix $\hat{c}$ looks like this:

$$var[\hat{c}] = \sigma_\varepsilon^2 (X^T X)^{-1}.$$

The variance of the error is

$$\hat{\varepsilon}_i = y_i - \hat{\alpha} - \sum_{k=1}^{K} x_{ik} \hat{c}_k. \tag{12}$$

Thus, the variance of $\hat{c}_k$ is calculated from the corresponding diagonal of the matrix $\sigma_\varepsilon^2 (X^T X)^{-1}$. With this estimate $\hat{\sigma}_k^2$, the approximately 95% confidence interval of the regression function $\beta$ is

$$\sum_{k=1}^{K} \hat{c}_k B_k(t) \pm 1.96 \sum_{k=1}^{K} \hat{\sigma}_k B_k(t). \tag{13}$$

# 3 Machine learning methods

## 3.1 Regression Energy Tree

Regression Energy tree model is prepared by Marco Brandi. [1] This regression method is developed using a tree-based model for functional data where covariates are functions. The whole algorithm works according to the procedure when a functional variable is selected that is more related to the response variable. The method is called the Regression Energy Tree, precisely because at this point the relationship between those variables is measured using an energy test such as the distance correlation test. In his work, Brandi compared other regression methods for functional data, such as the Functional Linear model. A comparison of the models showed that the best predictive results were obtained using the Regression Energy tree.

### 3.1.1 Regression energy tree algorithm

This regression tree using framework of decision tree where the testing procedure is implemented using energy statistics such as distance correlation. The first step in this method is transform and smooth data or simply put, we turn data into functional data. Once the data are sorted, then the relationship between the response variable $Y$ and the functional and smooth covariates $\hat{f}^J(t)$is measured. This is done using the distance correlation test of energy statistics. The distance correlation test is calculated as:

$$R_n^*(Y, \hat{f}_k^J(t)), \tag{14}$$

where $\sqrt{v-1}R_n^*$ with $v = \frac{n(n-3)}{2}$ asymptotically follow a standard normal distribution. This procedure is repeated for each variable. This forms a sequence of p values that must be correct to avoid a multiple problem with the Bonferroni correction. However, if the lowest adjusted value of p given by the test is less than the significance level $\alpha$, the hypothesis for the response variable $Y$ and the functional covariates $\hat{f}^J(t)$ is rejected. The following formulas are used to calculate the distances for the response variable and the distances for the functional variables:

$$d_{ij}^1 = ||Y_i - Y_j||^2 \tag{15}$$

$$d_{ij}^2(\hat{f}_k^J(t)) = ||f_{ik}(t) - f_{jk}(t)||^2 = \left( \frac{1}{\int_a^b w(t)dt} \int_a^b |f_{ik}(t) - f_{jk}(t)|^2 w(t)dt \right)^{\frac{1}{2}} \tag{16}$$

The split database expansion factor is then selected when recalculating the distance correlation test. It is also very important to take into account the problem of multiplicity and to correct the obtained p values by correcting the Bonferroni correction. The coefficient with the smallest corrected value of p shall be chosen, provided that it is less than $\alpha$. The last step is to choose

a value among the observed $n$ values of the coefficient $\hat{\theta}_{j*k*}$ by dividing the data into two subgroups, which reduces the variances of the response variable Y in the two subgroups. And this procedure is repeated in the resulting subsets until we accept the independence hypothesis.

Brandi [1] summarizes the following procedure as follows:

1. Transform data, where $i = 1, ..., n$ and $k = 1, ..., p$

$$(X_{ik})t_1), ...X_{ik})t_m)) \rightarrow (\hat{\theta}_{ik1}, ..., \hat{\theta}_{ikm})$$

2. Smooth data

$$\hat{f}_{ik}^{J} = \sum_{j=1}^{J} \hat{\theta}_{ijk}\phi_j(t)$$

3. Global test if there is association between Y and $\{X_k\}_{k=1,...,p}$.

4. Choose the functional covariates $\hat{f}_{ik*}^{J}$ associated the minimum adjusted P-value using Bonferroni correction conditionally that is less than $\alpha$.

5. Choose coefficient $\hat{\theta}_{j^*k^*}$ associated with the minimum adjusted P-value if less than $\alpha$.

6. Partition data for the observed coefficient value that minimazes the variances of $Y$ in the subsets.

7. Iterate the procedure on the two obtained subsets.

## 3.2 Bagging

Bagging is one of the latest machine learning algorithms designed to improve unstable classification and regression methods and their accuracy. Otherwise this method is called bootstrap aggregating. Bagging was introduced by Leo Breiman in 1996. Breiman developed the bagging algorithm as a variance reduction method for some basic procedure, such as decision trees or methods that perform variable selection and fitting to a linear model. This algorithm has received a lot of attention due to the simplicity of its implementation and the popularity of the bootstrap methodology.[2] However, in his work, he had presented only heuristic arguments as to why bagging should work. Subsequent work has shown that bagging is a smoothing operation that is certainly useful in improving the predictive properties of classification and regression trees. In the case of decision trees, the theory confirms Breinman's intuition that bagging is a dispersion reduction technique that also reduces the mean square error.

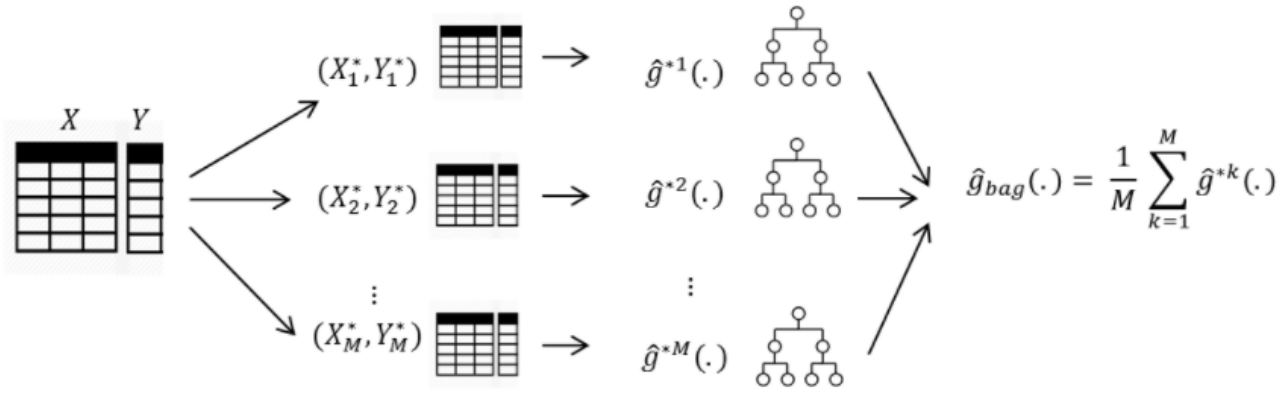


Figure 1: Bagging Procedure [6]

### 3.2.1 Bagging Algorithm

Assume that the data are presented as a set of p predictions $X_1, ..., X_n$ with response vector $Y = (Y_1, ..., Y_n)$. The relationship between X and Y is modeled using base procedure $\hat{g}(.)$. Then we have a bagged model $\hat{g}_{bag}(.)$. This is linear combination by some $\hat{g}(.)$ fit to bootstrapped samples of X. Bagging works like a smoothing operator for loss functions, which as mention before reduce model variance and improve model error.

Peter Buhlmann presents such a bagging algorithm in his work [3]:

- Step 1: Construct a bootstrap sample $(X_1^*, Y_1^*), ..., (X_n^*, Y_n^*)$ by randomly drawing $n$ times with replacement from the data $(X_1, Y_1), ..., (X_n, Y_n)$.
- Step 2: Compute the bootstrapped estimator $\hat{g}^*(.)$ by the plug-in principle:

$$\hat{g}^*(.) = h_n((X_1^*, Y_1^*), ..., (X_n^*, Y_n^*))(.).$$

- Step 3: Repeated steps 1 and 2 M times, where M is often chosen 50 or 100, yielding $\hat{g}^{*k}(.)$, where $k = 1, ..., M$. The bagged estimator is

$$\hat{g}_{bag}(.) = M^{-1} \sum_{k=1}^{M} \hat{g}^{*k}(.).$$

The operation of this algorithm is also illustrated in Figure 1. [6]

### 3.2.2 Advantages and disadvantages

Like any method, bagging has some major advantages and disadvantages that arise when the method is used to solve classification or regression problems. The main advantages of bagging are:

- The algorithm works well when using large amounts of data. Also, the performance of the algorithm is not affected in any way by the missing value in the data set.
- It reduces over-fitting of the model.
- Can be performed in parallel. Each individual boot file can be processed separately before combination.
- Bagging reduces the variance of the learning algorithm.

However, this algorithm also has drawbacks that pose certain challenges when using it. The main disadvantages are:

- The opportunity for a clearer interpretation is lost. It is difficult to make very accurate insights because only the average of the predictions is obtained. Although the output is more accurate than any single data point, a more comprehensive data set can sometimes provide more accuracy in a particular model.
- The increasing number of iterations requires more time as the algorithm slows down and intensifies.
- The model works particularly well with less stable algorithms. Therefore, an algorithm that is more stable or has a higher bias does not provide as much benefit.

## 3.3 K-fold Cross Validation

Cross-validation is a statistical method that is used to assess and compare the skills of machine learning models. The main aspect of the algorithm is to divide the data into two segments: one part is used to teach the model and the other part is used to validate the model. Also n this model, it is very important that each data point is validated, so training and validations sets must pass one after the other during validation. [14]

K-fold cross validation is the basic form of cross validation. In 1968, Mosteller and Tukey presented for the first time in literature a clear cross validation algorithm similar to the k-fold cross validation. [9] This algorithm has one parameter called k. This parameter specifies how many parts a particular dataset should be divided into, or more generally, the number of groups. Therefore, in the general case, this method is called k-fold cross validation. If a specific value of k is chosen, it can be used in the name of the method. For example, if k = 5, the name would be 5-fold cross validation.

### 3.3.1 K-fold Cross Validation algorithm

The general procedure for k-fold cross validation is as follows:

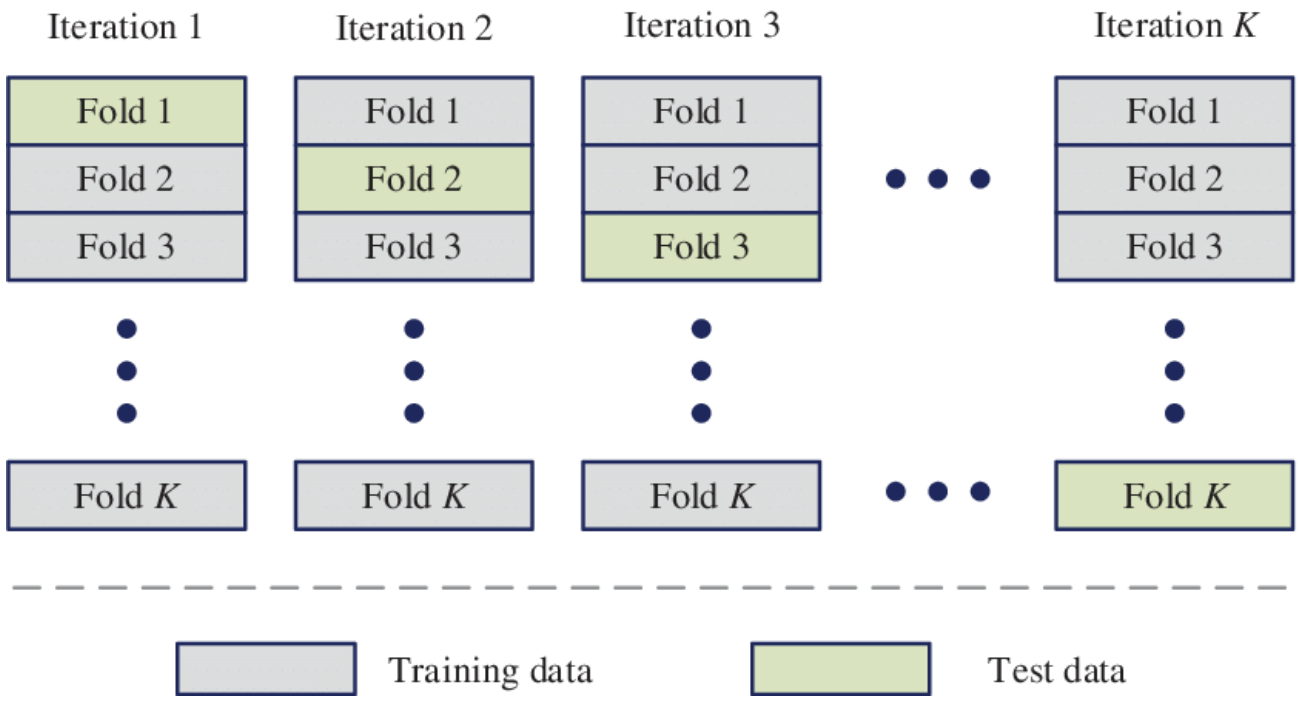


Figure 2: K-fold cross validation Procedure [11]

- First, the entire data set is shuffled in random order.
- Data are divided into equal-sized k groups or folds.
- Of the k groups, one segment is saved as model validation validation data, and the remaining parts of k-1 are used as training data.
- Fit a model on the training set and evaluate it on the test set. The results obtained are saved.
- The whole process is repeated k times so that each of the k samples is used once as validation data.
- Finally, the results obtained can be averaged to obtain a single estimate.

The principle of operation of k-fold cross validation is shown in the Figure 2.

### 3.3.2 Advantages and disadvantages

The main advantages of using the k fold cross validation algorithm are:

- Easy to understand method.
- Assessing model skills using k-fold cross validation is less biased and less optimistic.

- All observations are used for both training and validation, and each observation is used exactly once for validation.
- This method validates the performance of the model in several data folds.

But also this algorithm has not only pluses but also minuses. Thus, the main disadvantages are:

- While using k-fold cross validation, training takes more time because the model is trained in multiple training data sets.
- For the reason mentioned above, k fold cross validation is expensive in terms of computation.
- Due to the longer time consumption, the submission of feedback about the model and the search for its hyperparameters also slows down.

## 3.4 Repeated training

The Repeated training/test splits is also known as Monte Carlo cross-validation method has been presented by Picard and Cook [10]. Repeated training is quite similar to the bagging algorithm. In the bagging procedure, we chose a sample with replications, in which case our sample will be without replications. The basic idea of this algorithm is to create multiple data breaks into training and testing sets.

The ratio at which the data set will be shared can be varied and freely chosen. However, it is very important to keep in mind that the bias of the re-sampling method decreases as the amount of data in the subset approaches the amount of the simulation set. Therefore, it is recommended that training data be 70-80 percent.

The number of iterations is also very important in this algorithm. The higher the number, the more it will reduce the uncertainty of the estimates. Usually 50-200 reps are suggested as the optimal number of repetitions.

### 3.4.1 Repeated training algorithm

The repeated training algorithm looks like this:

- First we see the number m, how many times we will repeat the algorithm.
- It is then decided how the data set will be divided into training data and testing data. For example, 70% of the total data set will be training data, and the remaining 30% will be a testing set.
- A testing set is drawn at random.

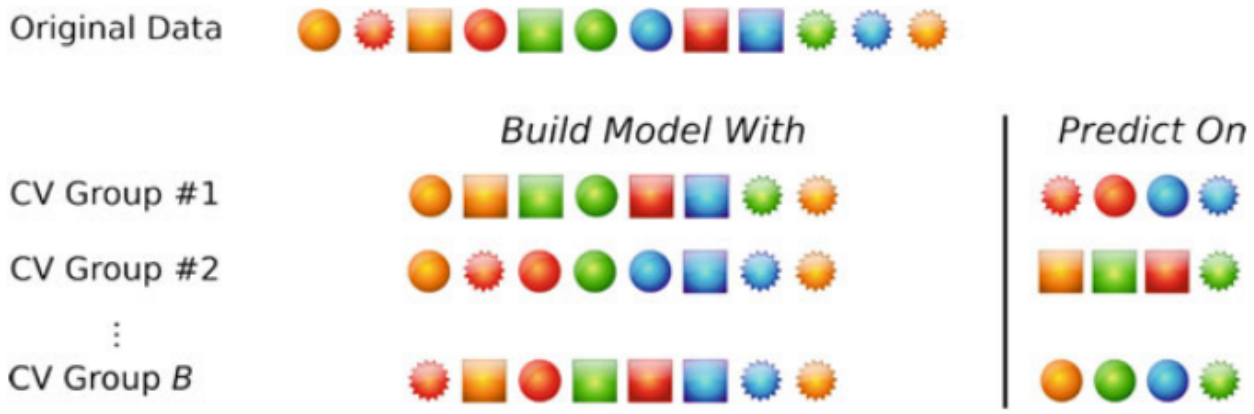


Figure 3: Repeated training Procedure [8]

- Fit a model on the training set and evaluate it on the test set.
- Repeat the whole procedure m times.
- We can summarize all the obtained results, for example, by averaging them.

The algorithm is also illustrated in the Figure 3.

### 3.4.2 Advantages and disadvantages

Like any method, repeated training also has benefits. The main advantages of the method are:

- The proportion of the training and testing dataset split is not dependent on the number of iterations.
- The method may give a less variable estimate.

But of course this method has its downsides. The main disadvantages are:

- Some observations may not be selected in the validation sample throughout the procedure. However, others may just be selected more than once. This means that the subsets of the approval may overlap.
- This method is also characterized by variations in Monte Carlo, the results may differ if the analysis is repeated with different random splits.
- The method provides a more biased estimate.

# 4 Practical Part

In the practical part, we combine the available regression energy tree model with the resampling techniques presented in the second section. The practical part consists of two parts. The first part presents the results obtained using simulated data, and the second part presents the results using real data.

## 4.1 Simulated dataset

We generate regression functional data from the regression model

$$Y_{ij} = f(t_{ij}) + \varepsilon_{ij}$$

with $j = 1, ..., m$ and $t_{jm} = j/m$. The regression functions for f are

$$F_1(t) = \left(\frac{5t-2}{3}\right)^2 - \cos 2\pi t \tag{17}$$

$$F_2(t) = -F_1(t) \tag{18}$$

$$F_3(t) = \sin \frac{5\pi t}{3} \tag{19}$$

$$F_4(t) = -F_3(t) \tag{20}$$

The data consist of 50 curves for each of the four curve shapes $F_1, F_2, F_3, F_4$. We consider the noise $\varepsilon_{ij}$ to be normally distributed. The response variable $Y_i = \mu_i(F(t_1), ..., F(t_m))$ when $F$ is one of the four forms $F_1, F_2, F_3, F_4$.

The first experiment is using a regression energy tree combined with a bagging algorithm. The first is to take a bootstrap sample with a replacement from the data set. We then use it in the model and find the meaning of prediction and MEP. The bagging parameter M is set to 100, so we repeat the last two steps 100 times. Example of tree is shown in the Figure 4. More examples are in Appendix A. From all the boxplots in Figures 4, 5 and 6, we see that in all cases the distribution of data is different at each terminal node. The number of nodes is also different in each case.

The second experiment is a combination of repeated training and a regression tree. In this case, we also repeat the process 100 times. We collect the training sample without repetition and it accounts for 70% of our total data, the rest is a test sample. Output of regression tree with repeated training is shown in Figure 5.

The third experiment is the application of the k-fold cross validation algorithm using a regression energy tree. Choose that k is 5. Output of regression tree with k-fold cross validation is shown in Figure 6.

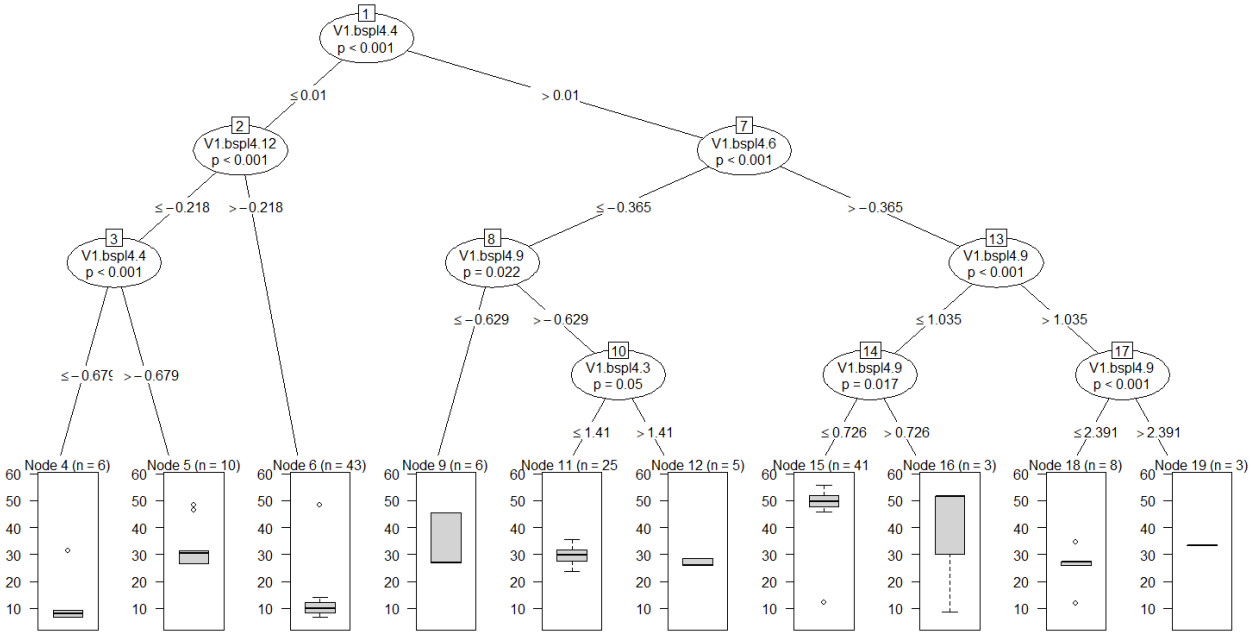


Figure 4: Output of regression tree with bagging for simulated dataset

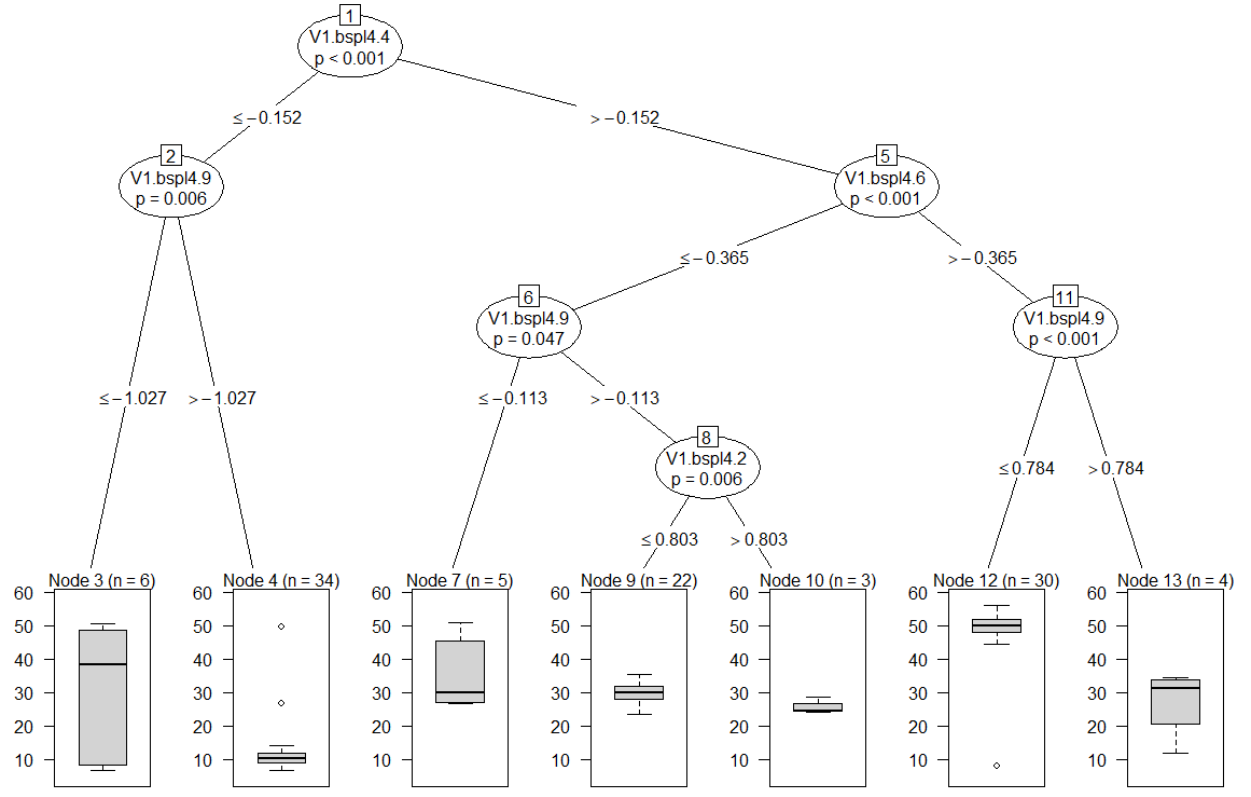


Figure 5: Output of regression tree with repeated training for simulated dataset

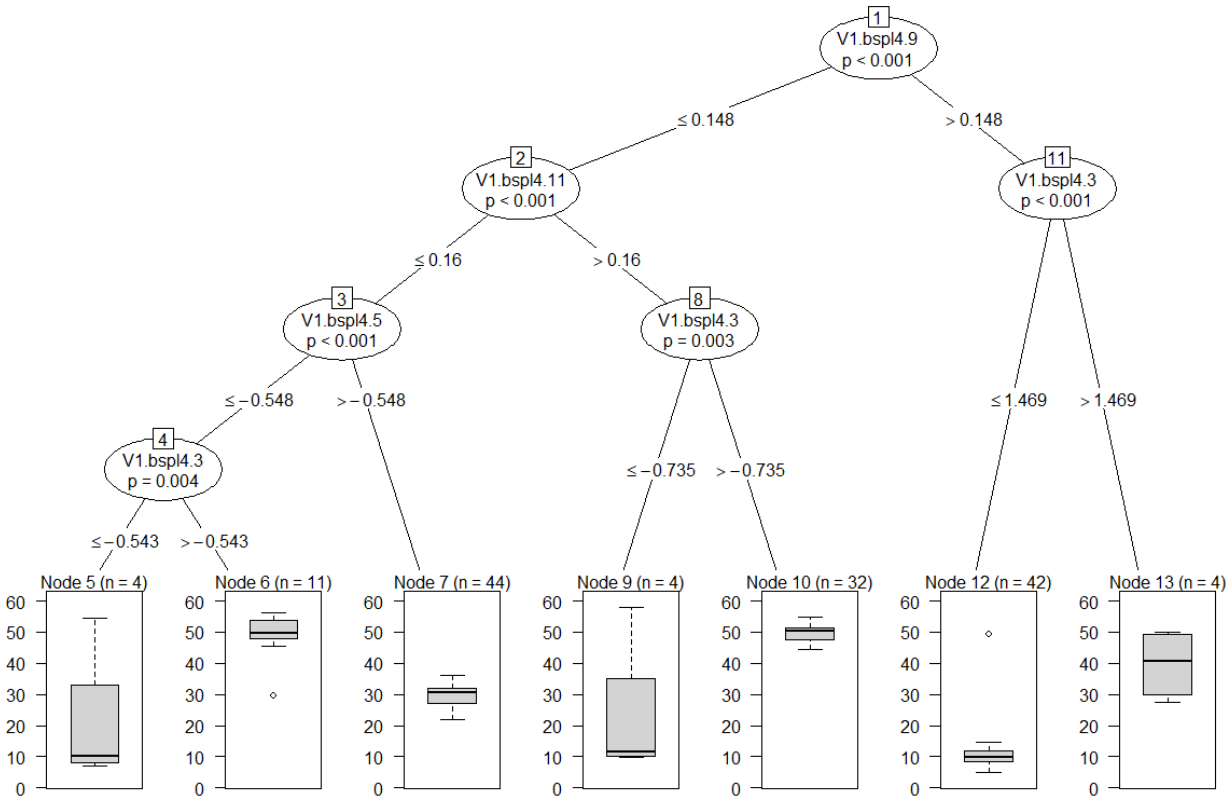


Figure 6: Output of regression tree with repeated training for simulated dataset

We compare the accuracy of the model using different resampling techniques using the mean square error prediction (MEP), whose formula looks like this:

$$MEP = \left( \sum_{i=1}^{n} \frac{(Y_i - \hat{Y}_i)^2}{n} \right) / Var(Y) \tag{21}$$

| Method | Mean MEP | Median MEP | Mode MEP |
|---|---|---|---|
| Bagging | 0.3710316 | 0.3270169 | 0.2827672 |
| Repeated training | 0.373086 | 0.3372972 | 0.2517172 |
| K-fold Cross validation | 0.295161 | 0.213247 | 0.213247 |

Table 1: Results with simulated data, when M=100 and k=5

For each experiment, we find the mean, median, and mode of their MEP sample. The results are shown in the Table 1.

We see that the lowest results for both the mean, median, and mode are in the case of k-fold cross validation, which means that it was best predicted in this case. In the case of bagging and repeated training, we obtained very similar averages, but slightly less, that is better predicted was using the bagging resampling technique. In the case of the median, the bagging results are also lower. This means that half of all reps were predicted to be better in the case of bagging than in repeated training. However, the most frequent recurrence of MEP in the case of repeated training is less than in the case of bagging.

| Method | Mean MEP | Median MEP | Mode MEP |
|---|---|---|---|
| Bagging (M=100) | 0.3710316 | 0.3270169 | 0.2827672 |
| Repeated training (M=100) | 0.373086 | 0.3372972 | 0.2517172 |
| K-fold Cross validation (k=5) | 0.295161 | 0.213247 | 0.213247 |
| Bagging (M=1000) | 0.281356 | 0.212614 | 0.234513 |
| Repeated training (M=1000) | 0.338699 | 0.296844 | 0.725881 |

Table 2: Results with simulated data, when M=100 and k=5

Let's look at a few more cases where we do 1000 reps for bagging and repeated training. The results obtained are shown in the Table 2. Now comparing the MEP means, we see that the lowest average of all the experiments is obtained when the model is combined with the bagging algorithm when 1000 repetitions are performed. In this case, the average is 0.281356. We also see that the increase in the number of repetitions resulted in better results in the case of Repeated training as well, average is 0.338699. The median bagging case is also the smallest compared to all the others. This means that half of the MEP is less than 0.212614.

For comparison, let's do another experiment and use only the regression energy tree method without any resampling technique. In this case, we get that the MEP is equal to 0.5501509. We see that the model gets quite a lot of error. Comparing this result with the MEP means of all methods, we can see that any resampling technique gave a better result.

## 4.2 Weather data

In this part we use dataset which contains 2007-2017 of daily weather observations from different locations in Australia. We use maximal temperature, number of hours of bright sunshine, the speed (km/h) of the strongest wind gust and the amount of rainfall recorded for the day in mm.

We choose that the response variable will be temperature and the other selected variables will be functional covariates. We will apply the same experiments to this data as we did to the simulated data. The results obtained using the bagging method and repeated training with 100 and 1000 repetitions and k-fold cross validation when k is 5 are shown in the Table 3. Some of the outputs of regression trees can be found in Appendix A.

We see very similar trends as in the case of simulated data. The lowest MEP mean value is obtained using the bagging algorithm. This time, the k-fold cross validation method was also well predicted. The highest mean is the Repeated training case, when M=100. However, different conclusions are observed with regard to the median. The smallest median was obtained for k-fold cross validation. The results of the bagging algorithm, when M is 1000 are not much different from it. This means that half of the predictions are accurate enough. Looking at the fashion values, it was the lowest in this case of repeated training. The biggest mode is 0.352471, when repeated training with M=1000 is used.

| Method | Mean MEP | Median MEP | Mode MEP |
|---|---|---|---|
| Bagging (M=100) | 0.342647 | 0.34621 | 0.32122 |
| Repeated training (M=100) | 0.35142 | 0.33196 | 0.32521 |
| K-fold Cross validation (k=5) | 0.325231 | 0.301571 | 0.301571 |
| Bagging (M=1000) | 0.31204 | 0.30621 | 0.29122 |
| Repeated training (M=1000) | 0.335471 | 0.312483 | 0.352471 |

Table 3: Results with Australian weather data

Compare the results obtained with the result obtained using only the regression energy tree. Without any resampling technique, an MEP of 0.513648 is obtained. This result is significantly higher than with any of the previously introduced resampling techniques.

# 5 Conclusions

In this work, we use the regression energy tree when the response variable is a number and the covariates are functions. This method is extended by three resampling techniques: bagging, k-fold cross validation, and repeated training. In each case where the original model is extended by some resampling technique, we calculate the means, median, mode, and mean square error prediction (MEP) of the resulting predictions. Also, each case was calculated with the generated data and the real data (Australian Weather Data).

The practical part showed that the best results were obtained using the mixing algorithm when M = 1000. Using both simulated data and real weather data, the regression energy tree advanced mixing technique performs the lowest MEP. Another method would be k-fold cross-validation when k is 5. In this case, a result very similar to the previous method was achieved. The third case of retraining turned out to be less good, with the MEP being the highest in most cases. The performed experiments show that a lower average MEP result is obtained in both bagging and repeated training cases, the model is recalculated more times. The advantage of bagging can be explained by the fact that it reduces dispersion. This is especially useful when using large amounts of data, where missing values may make a bigger difference, making them more likely to adapt too much and not allow accurate aggregation of new data sets.

Before concluding, some suggestions for further work. In this work, only one case is considered, where the response variable is a number and the covariates are functions. It would be interesting to look at other cases where the response variable is also a function and to find out what the prediction performance then looks like.

# References


[1] M. Brandi. *Classification and Regression Energy Tree for Functional Data*, Dipartimento di Statistica, Probabilit'a e Statistiche Applicate, Universita La Sapienza, Roma, 2017.

[2] L. Breiman. *Bagging predictors*, Machine Learning 24, 123–140, 1996.

[3] P. Bühlmann, B. Yu. *Analyzing bagging*, Ann. Statist. 30 (4) 927 - 961, 2002.

[4] U. Grenander, U. *Stochastic processes and statistical inference*, Arkiv för Matematik, 1950, 195–277.

[5] K. Karhunen. *Zur Spektraltheorie stochastischer Prozesse*, Annales Academiae scientiarum Fennicae, 1946.

[6] N. Kawa. *Bagging: an ensemble method for variance reduction in unstable models*, 2017, https://rpubs.com/nurakawa/bagging

[7] P.Kokoszka, M. Reimherr. *Introduction to Functional Data Analysis*, Taylor & Francis Group, LLC, 2017.

[8] M. Kuhn, K. Johnson. *Applied Predictive Modeling*, Springer Science+Business Media, New York, 2013.

[9] F. Mosteller, D.L. Wallace *Inference in an authorship problem*, J. Am. Stat. Assoc., 58:275–309, 1963.

[10] R. Picard and D. Cook. *Cross-validation of regression models*, Journal of the American Statistical Association, 79(387):575583, 1984.

[11] R. Qiubing, L. Mingchao, H. Shuai. *(2019). Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives*, Big Earth Data. 3. 1-18, 2019.

[12] J. Ramsay, C.J. Dalzell. *Some Tools for Functional Data Analysis*, Journal of the Royal Statistical Society, Series B (Methodological), 53(3), 539–572, 1991.

[13] J. Ramsay, BW. Silverman. *Functional Data Analysis*, 2nd ed. Springer, 2005.

[14] P. Refaeilzadeh, L. Tang, H. Liu. *Cross-Validation*, Encyclopedia of Database Systems. Springer, Boston, MA, 2009

[15] J. Wang, J. Chiou, H. Müller. *Functional Data Analysis*, Annu. Rev. Stat. Appl., 3:257–95, 2016.

# 6 Appendix A

## 6.1 Output of regression energy trees using bagging algorithm

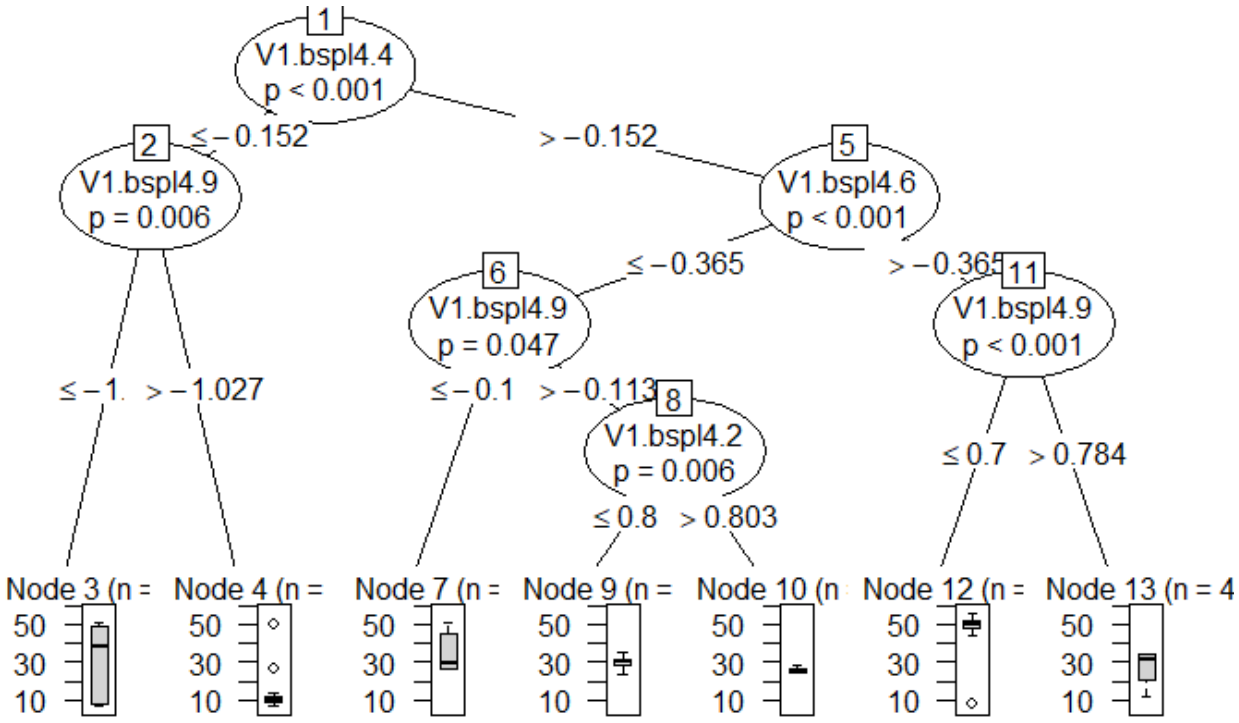


Figure 7: Output of regression tree with bagging for simulated dataset

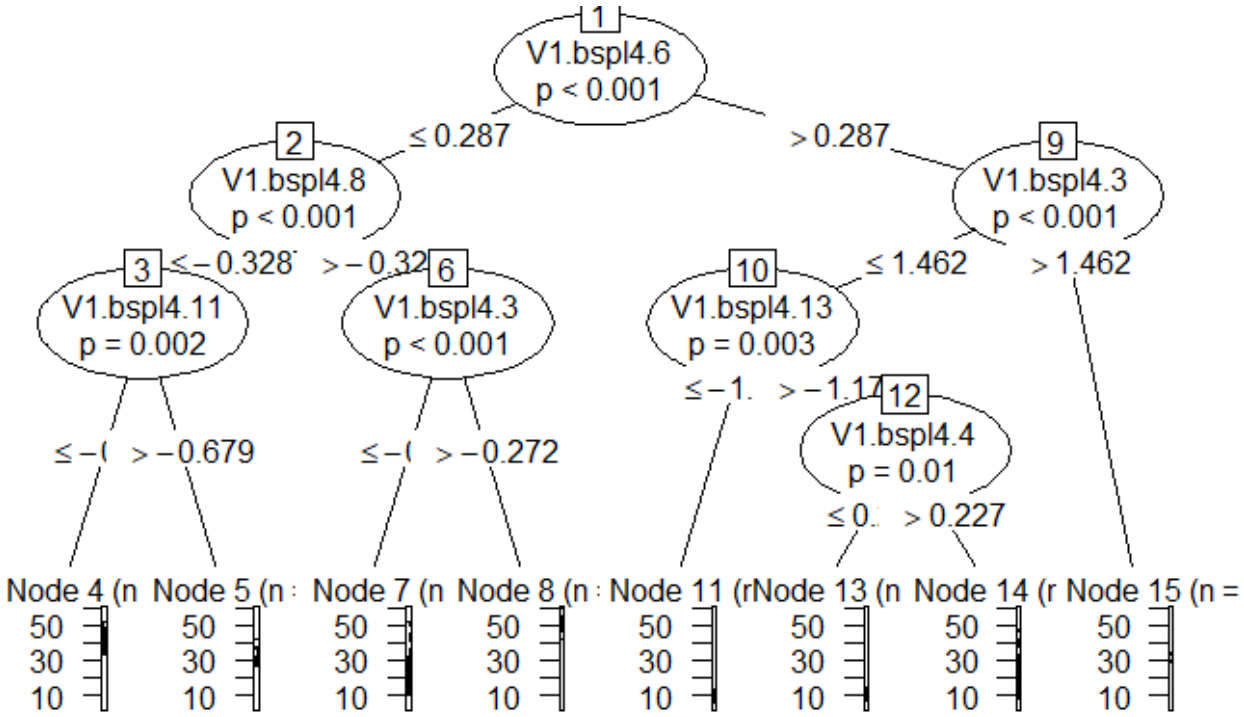


Figure 8: Output of regression tree with bagging for simulated dataset

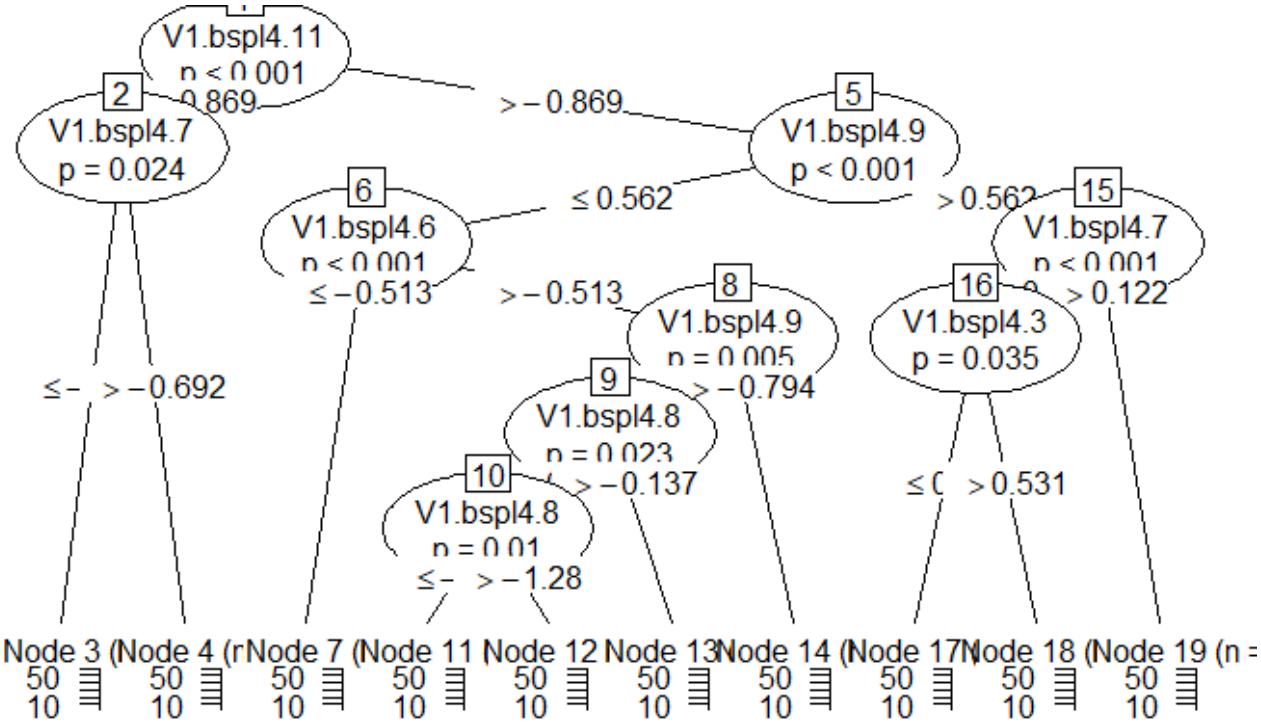


Figure 9: Output of regression tree with bagging for simulated dataset

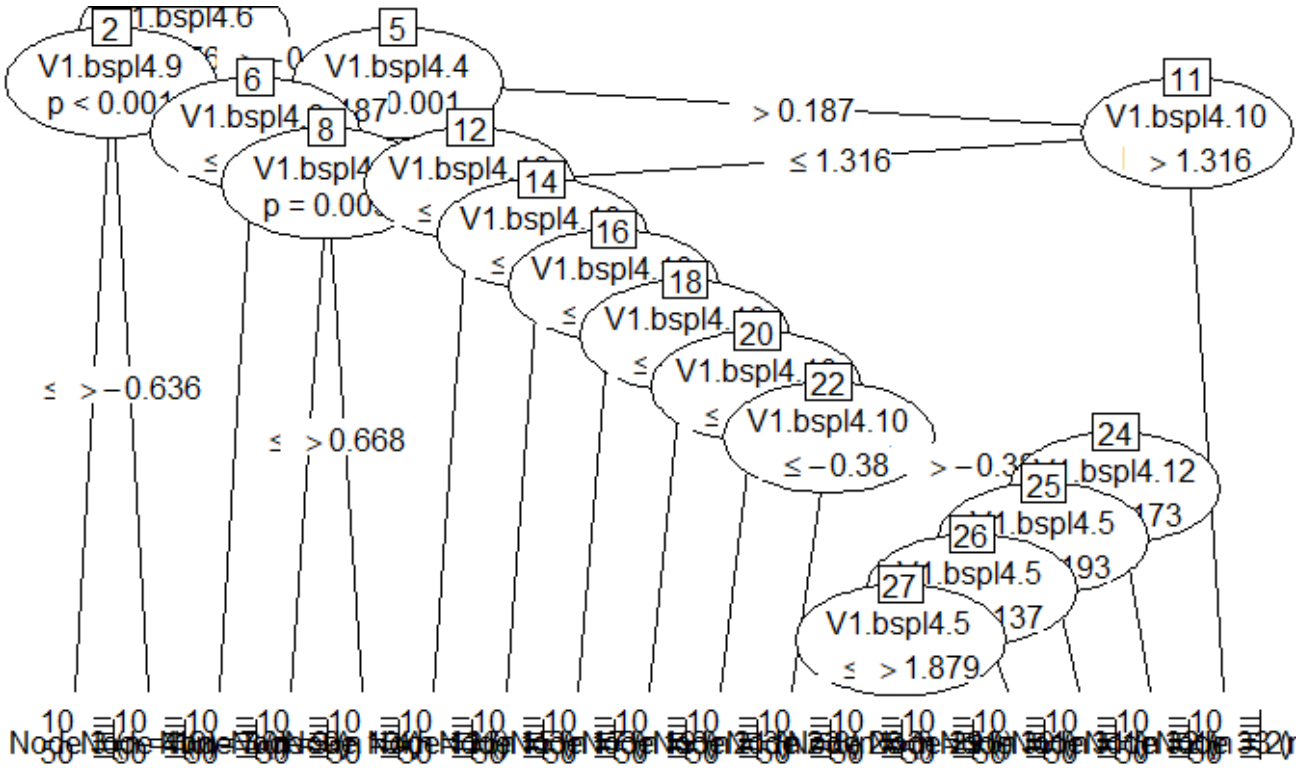


Figure 10: Output of regression tree with bagging for simulated dataset

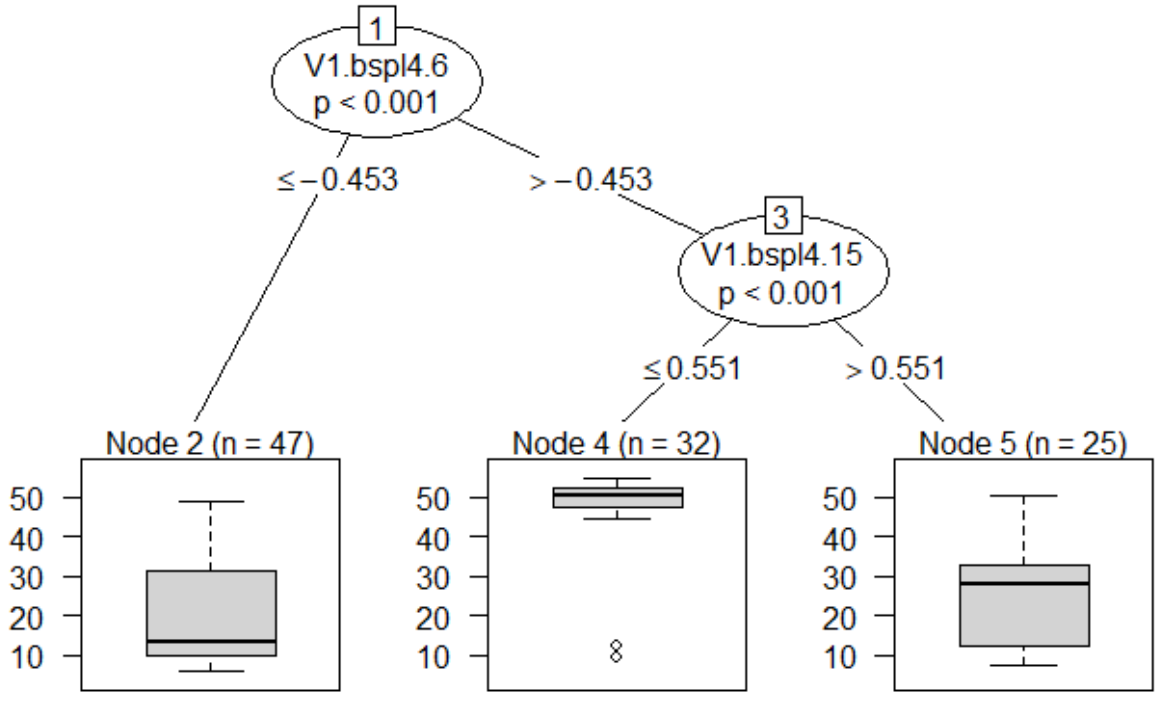


Figure 11: Output of regression tree with bagging for simulated dataset

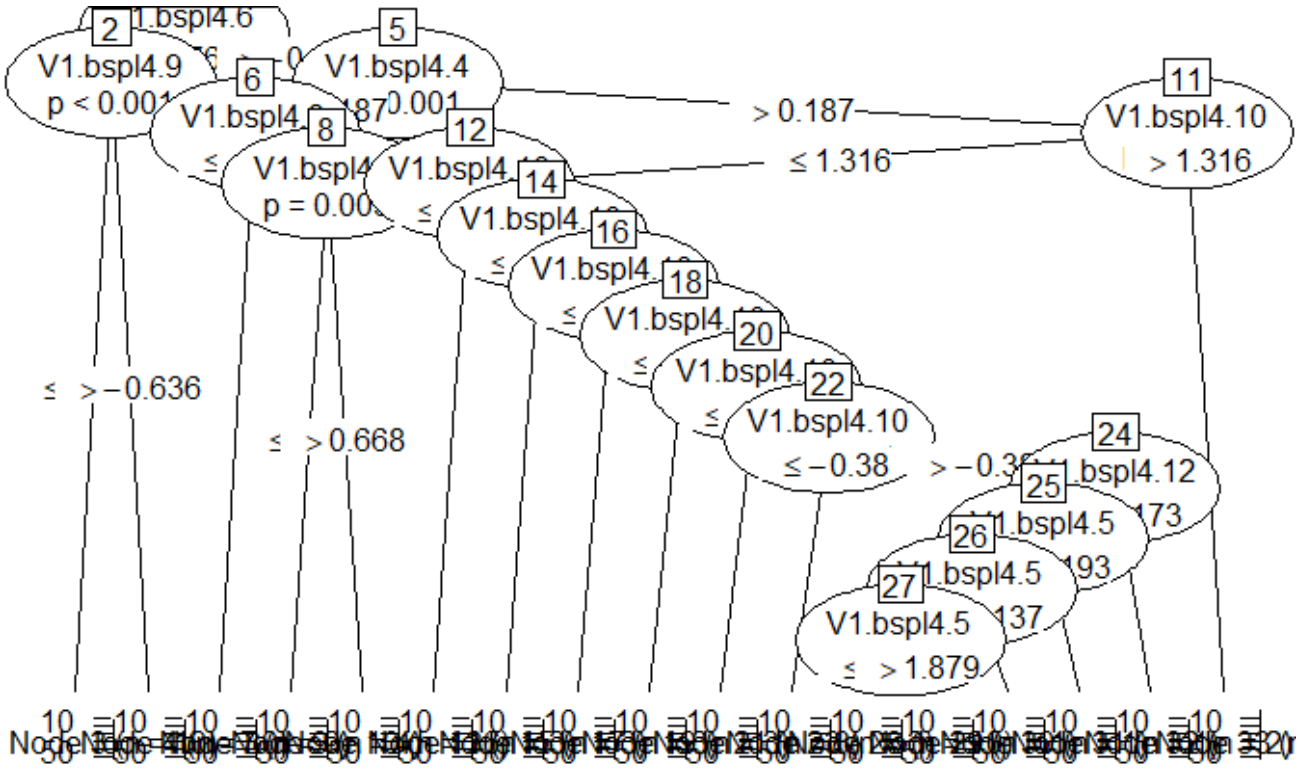


Figure 12: Output of regression tree with bagging for weather dataset

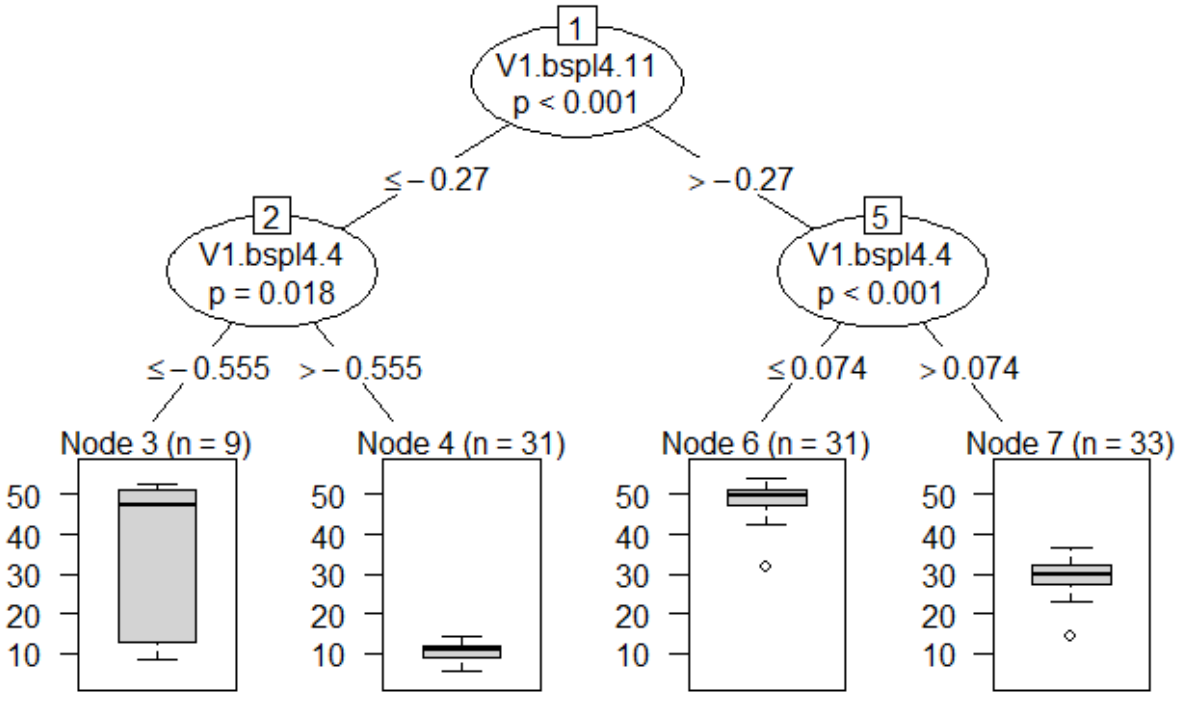


Figure 13: Output of regression tree with bagging for weather dataset

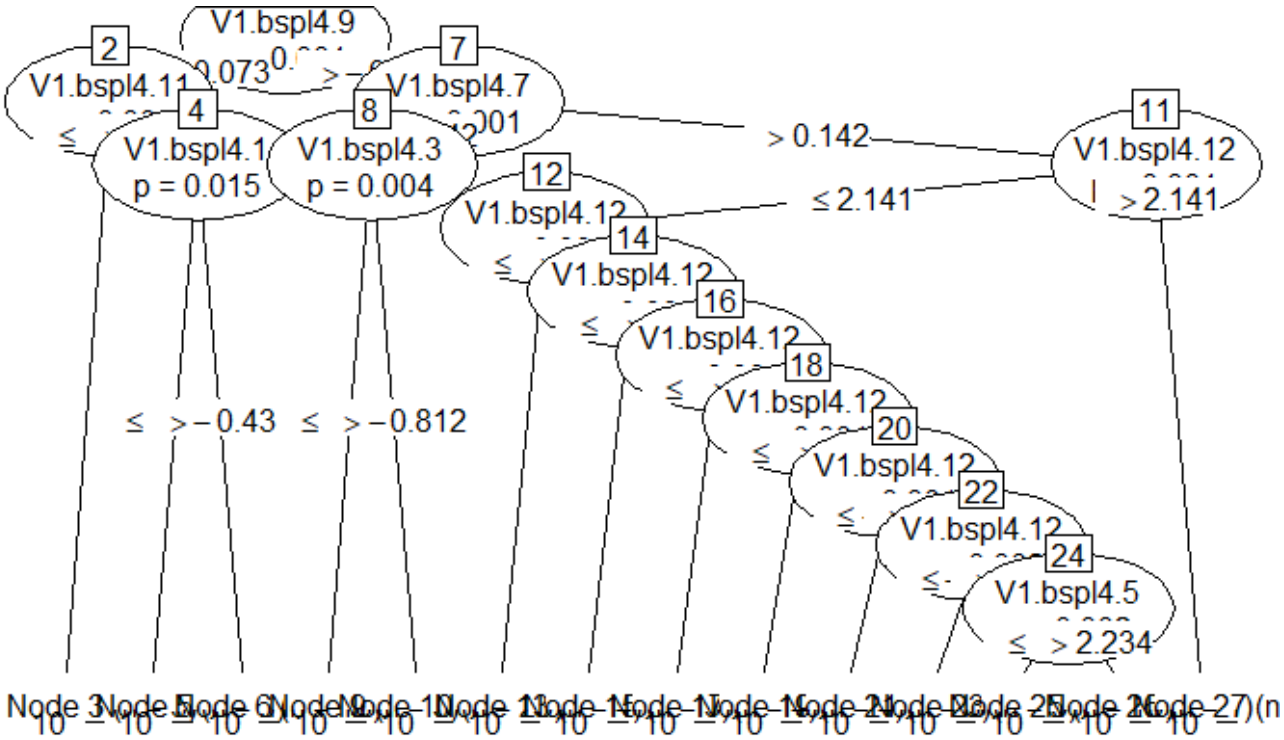


Figure 14: Output of regression tree with bagging for weather dataset

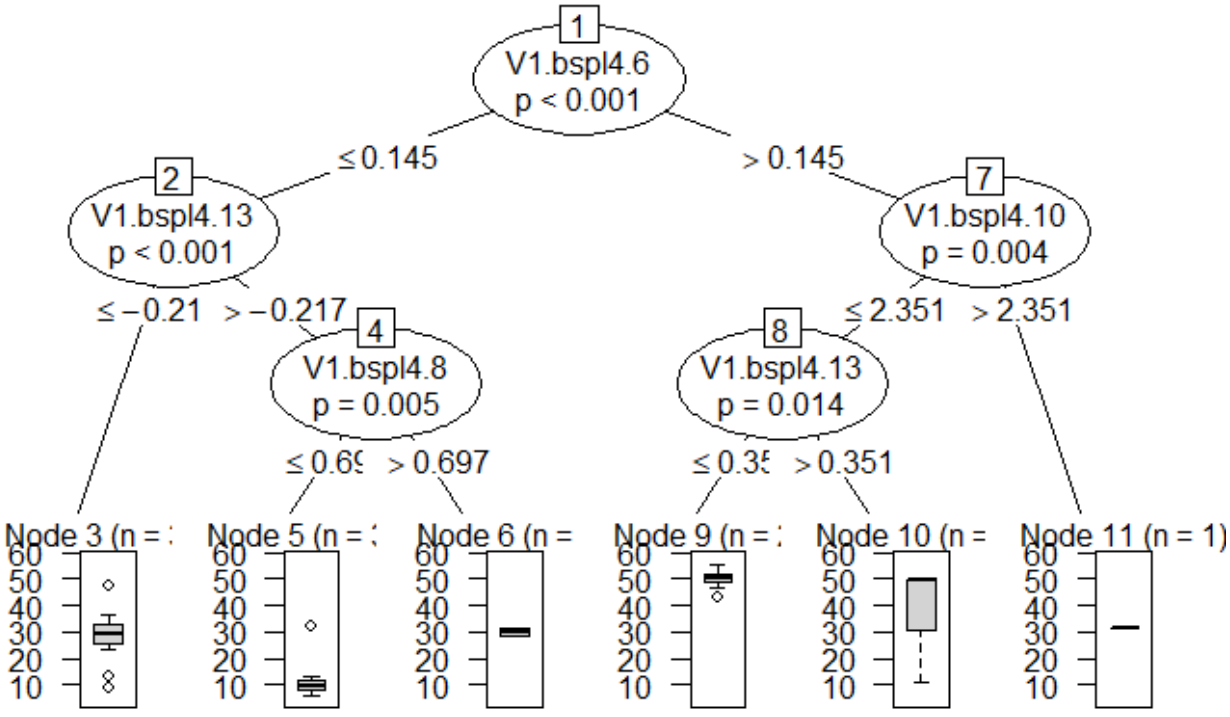


Figure 15: Output of regression tree with bagging for weather dataset

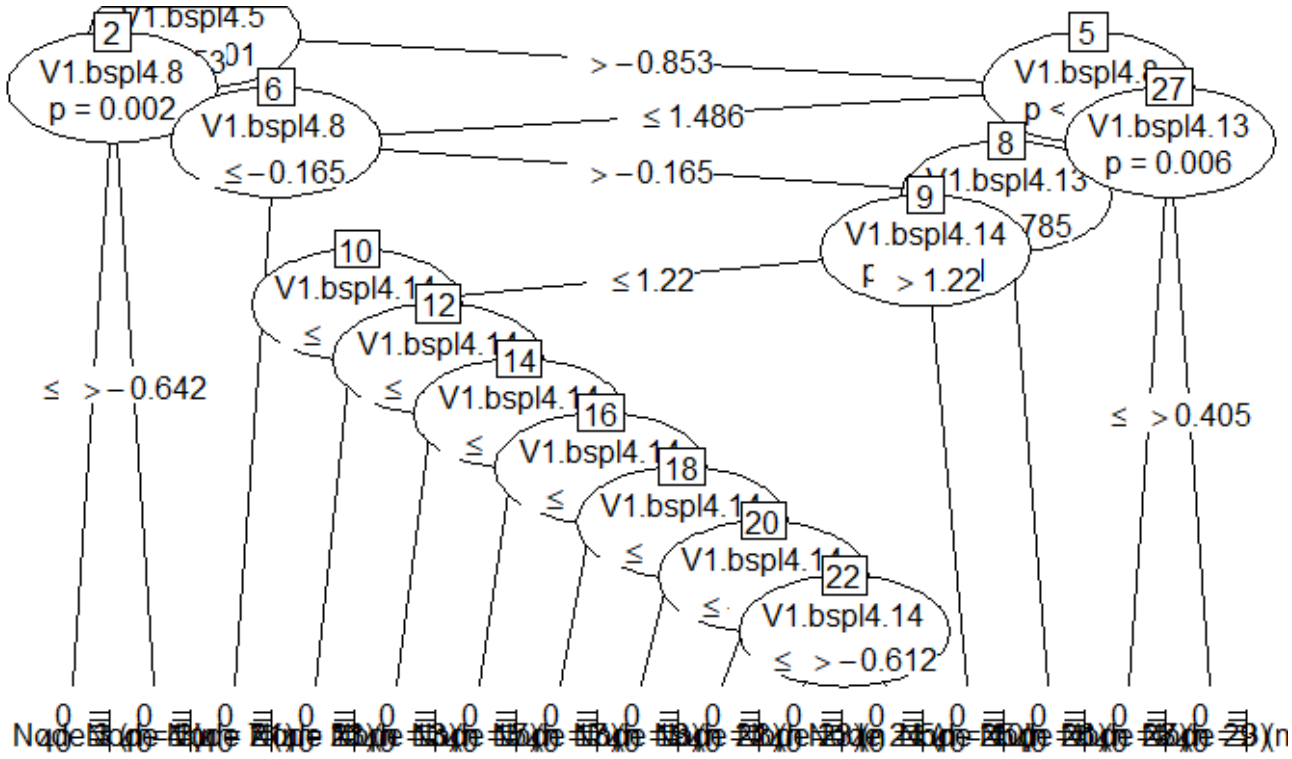


Figure 16: Output of regression tree with bagging for weather dataset

## 6.2 Output of regression energy trees using repeated training algorithm

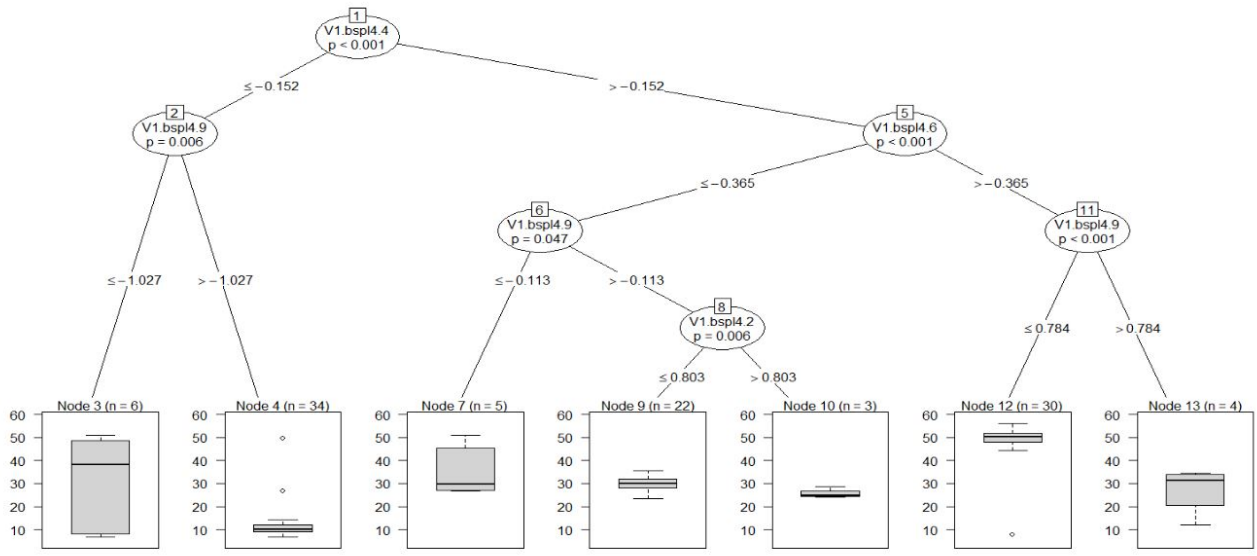


Figure 17: Output of regression tree with repeated training for simulated dataset

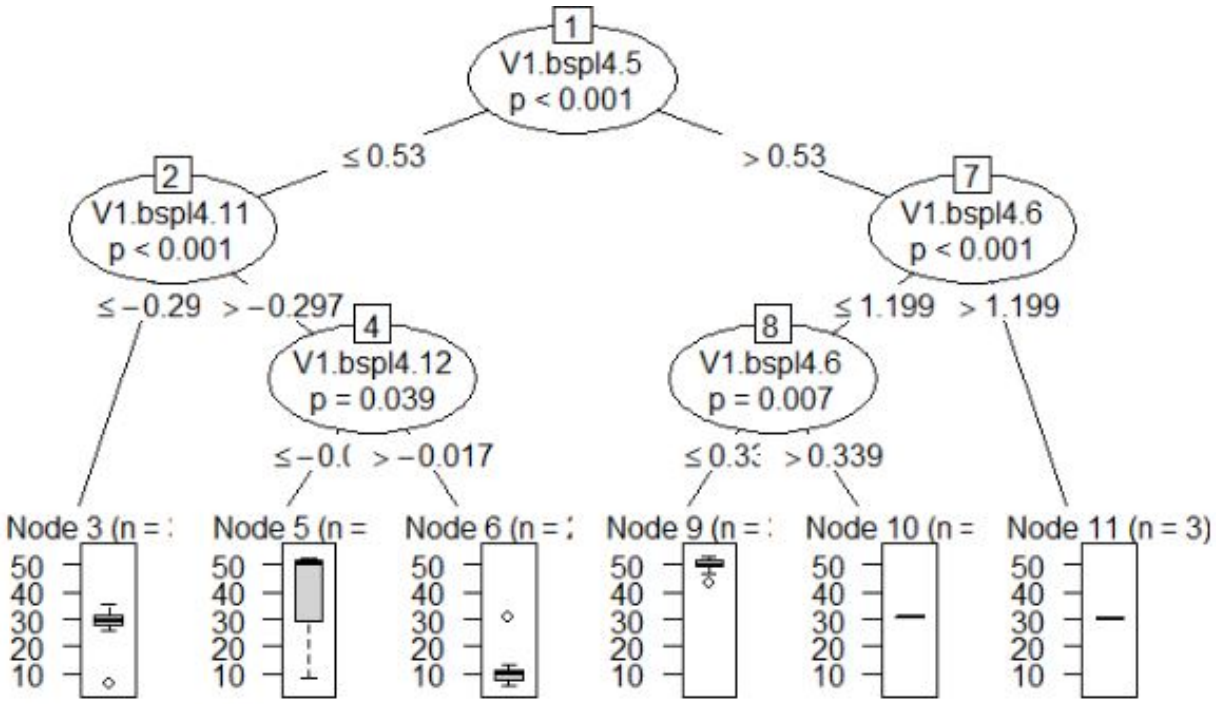


Figure 18: Output of regression tree with repeated training for simulated dataset

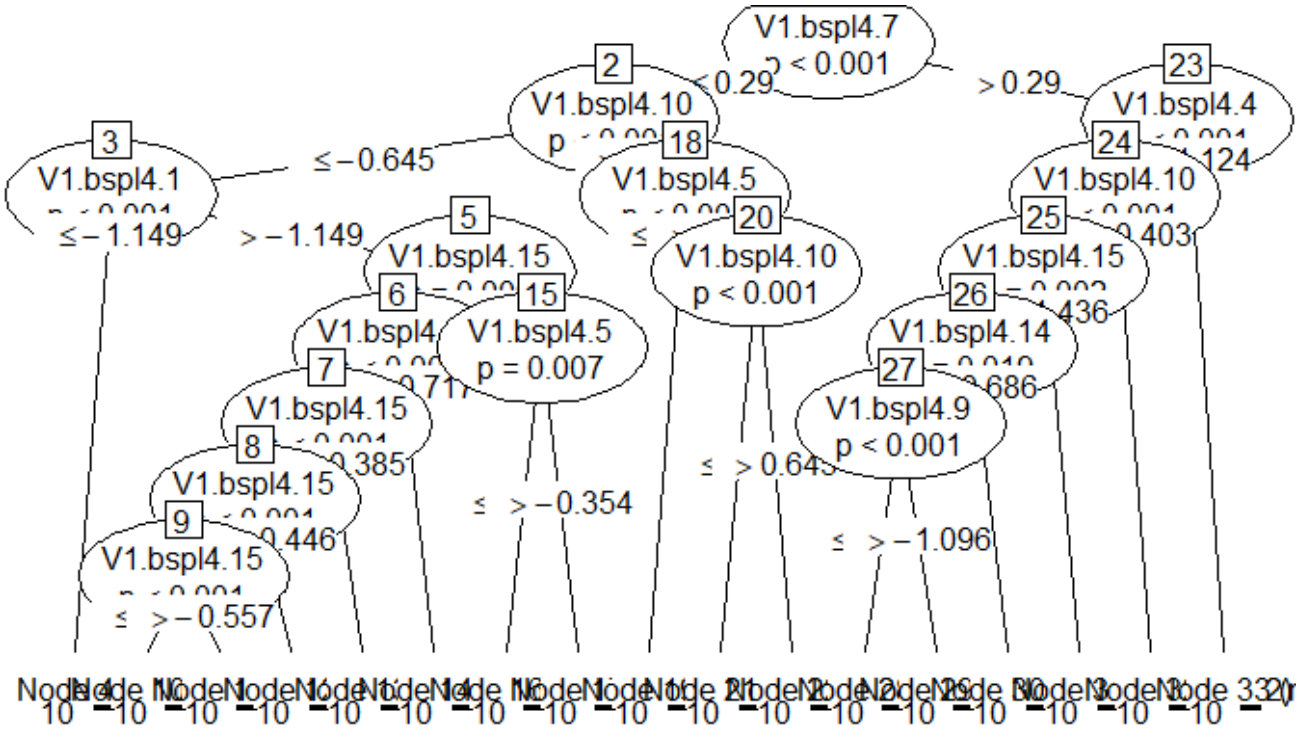


Figure 19: Output of regression tree with repeated training for simulated dataset

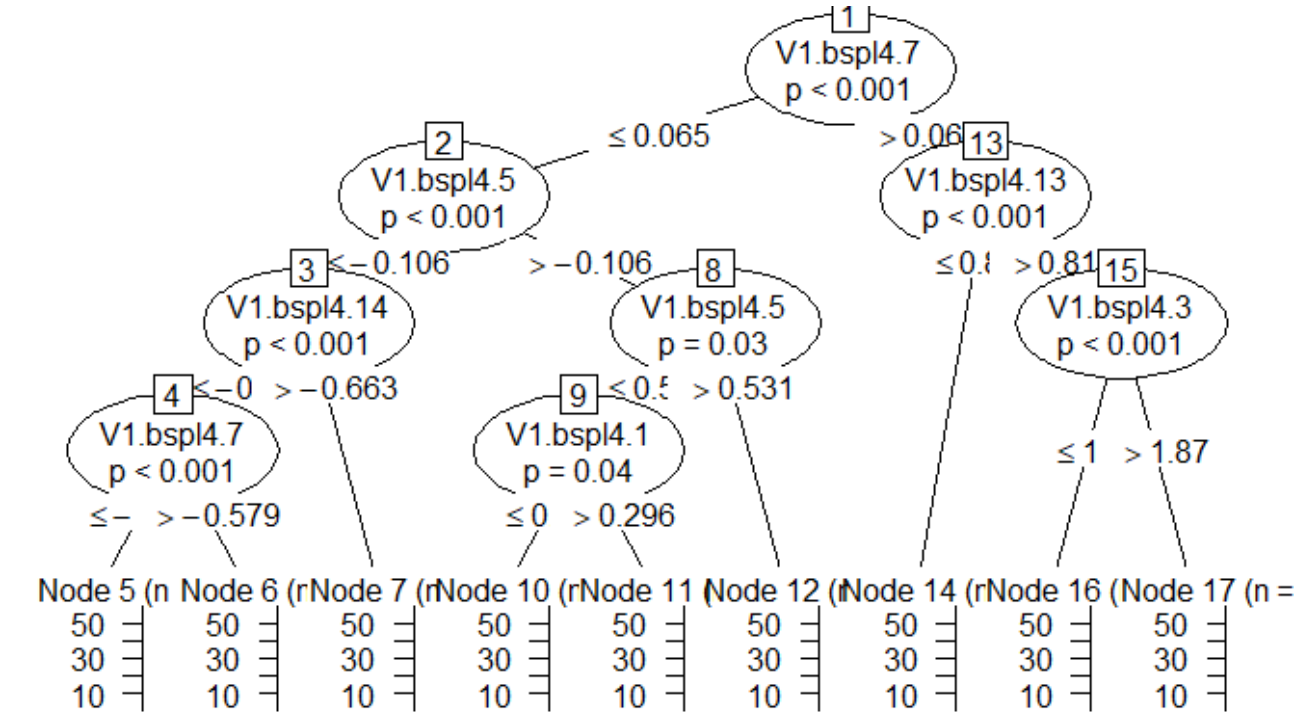


Figure 20: Output of regression tree with repeated training for simulated dataset

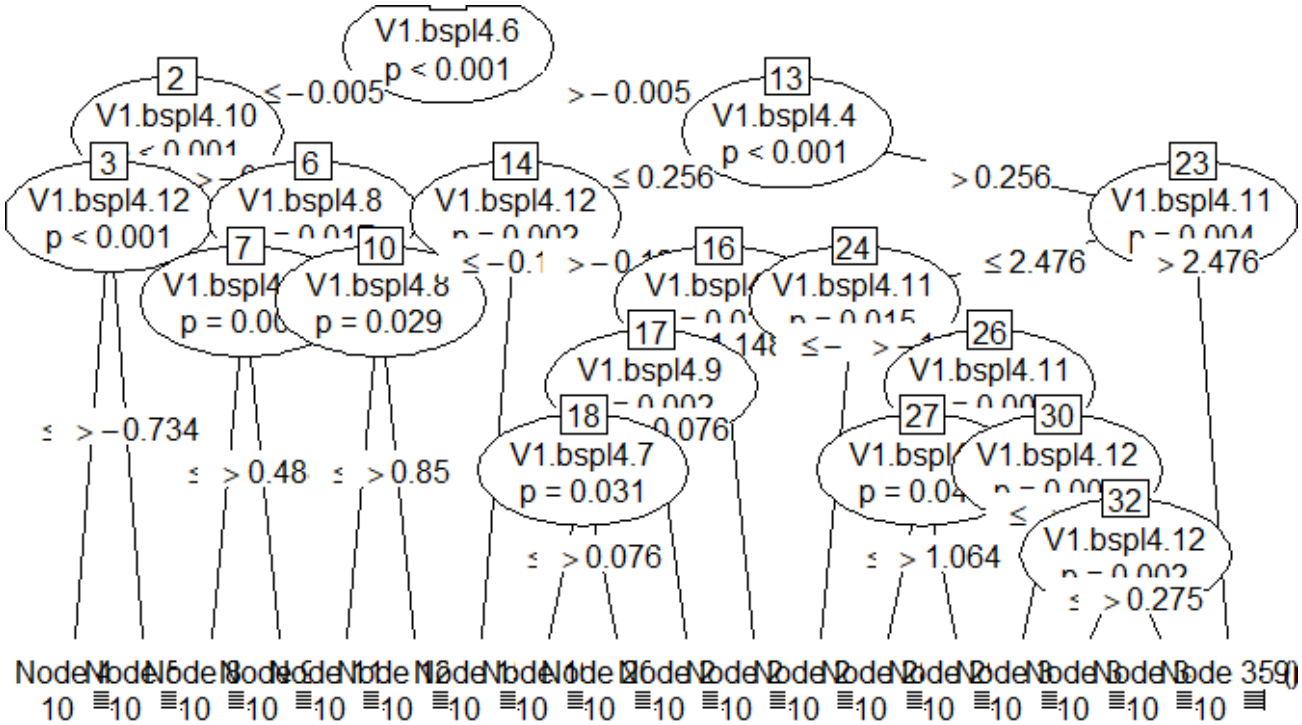


Figure 21: Output of regression tree with repeated training for simulated dataset

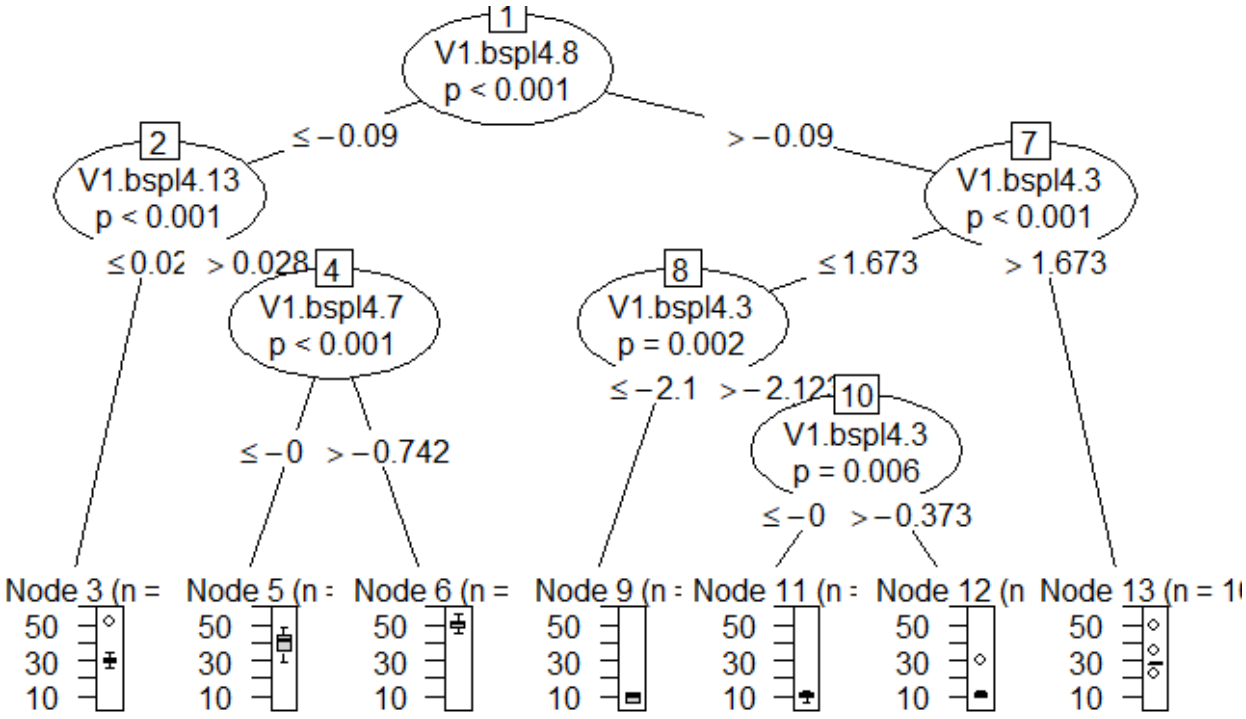


Figure 22: Output of regression tree with repeated training for weather dataset

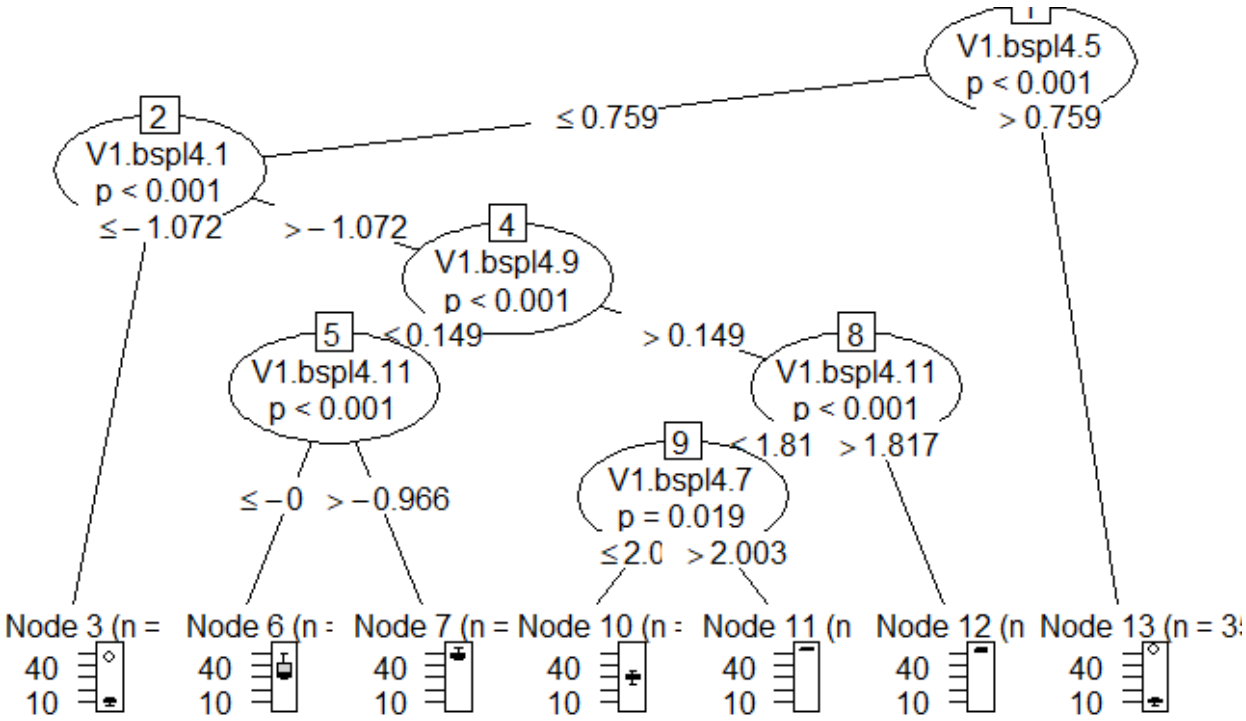


Figure 23: Output of regression tree with repeated training for weather dataset

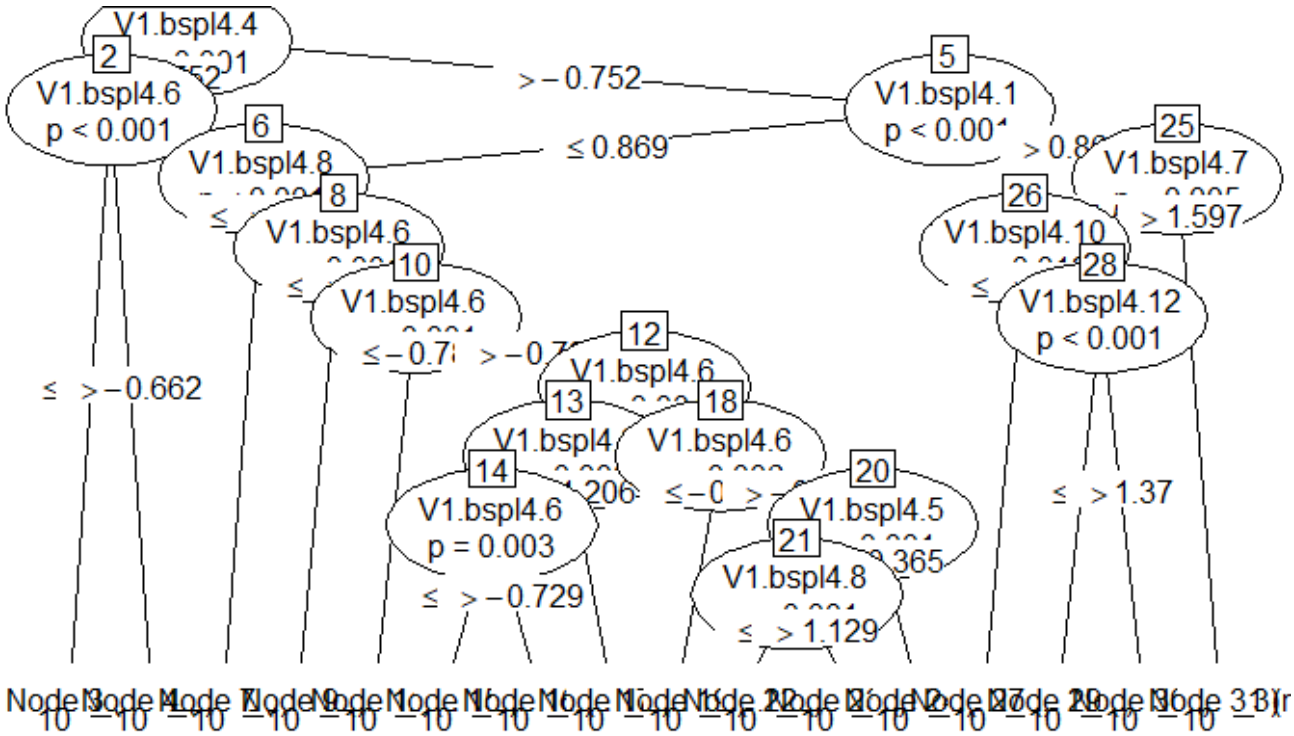


Figure 24: Output of regression tree with repeated training for weather dataset

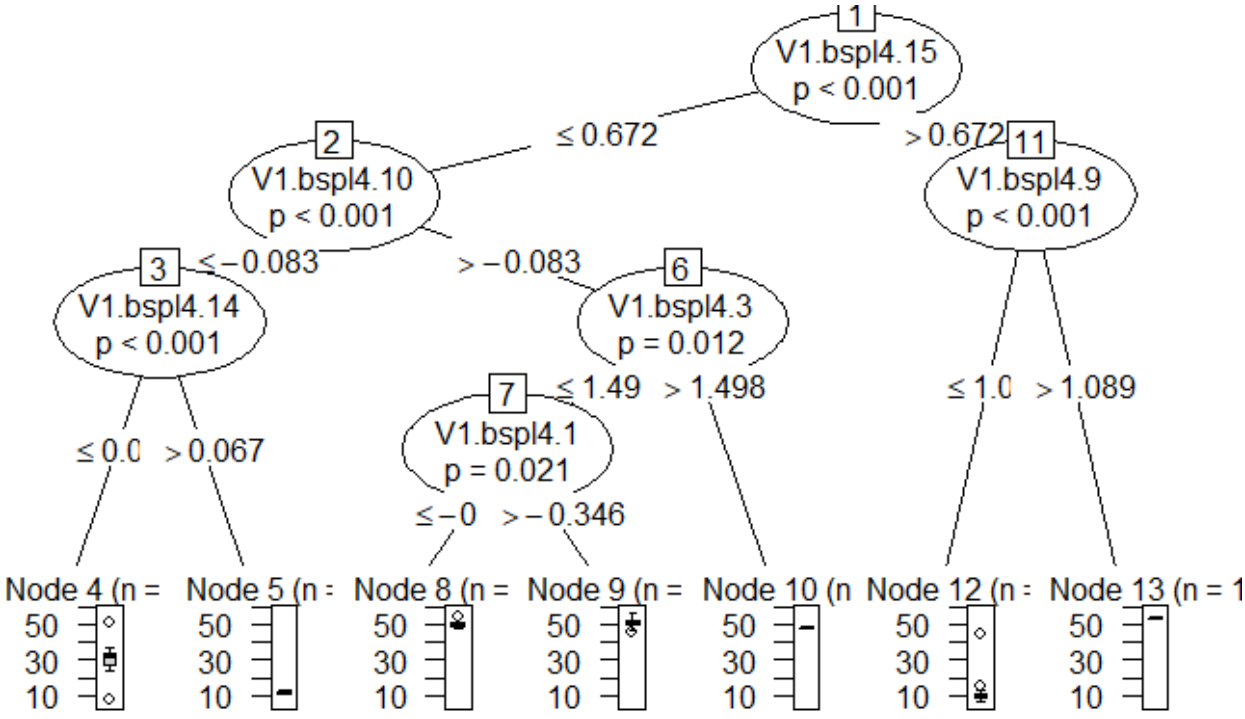


Figure 25: Output of regression tree with repeated training for weather dataset

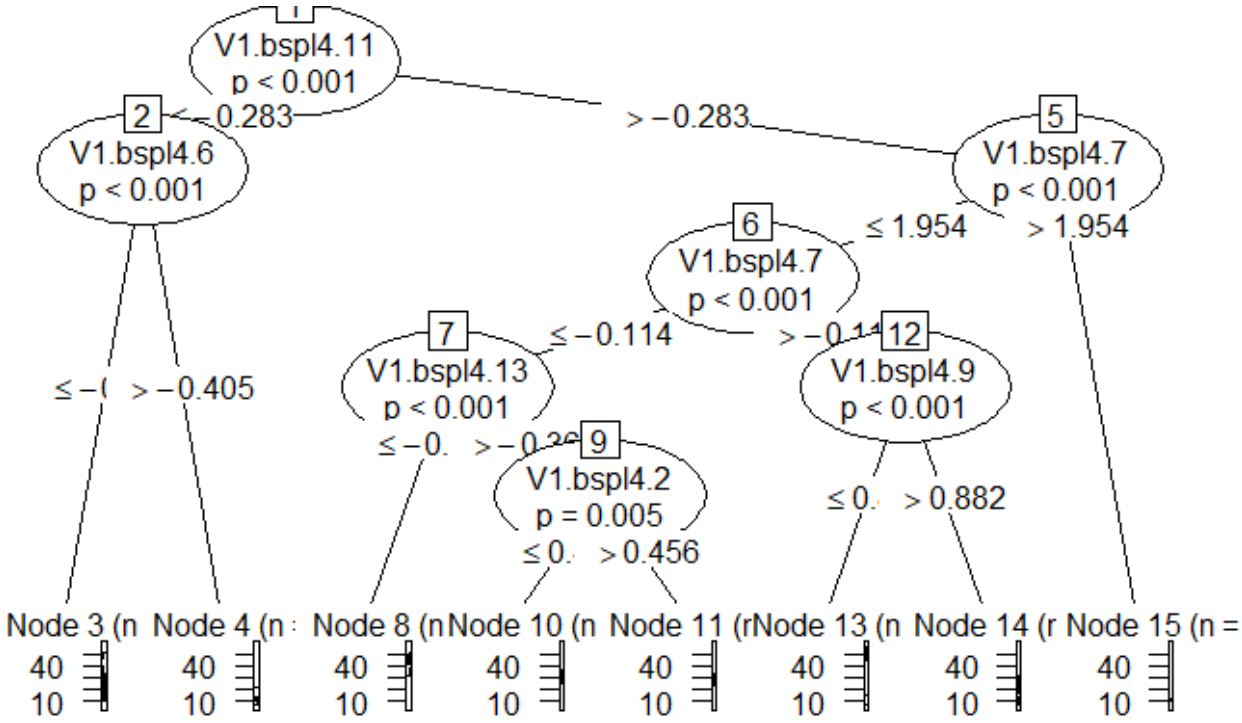


Figure 26: Output of regression tree with repeated training for weather dataset

## 6.3 Output of regression energy trees using k-fold cross validation algorithm

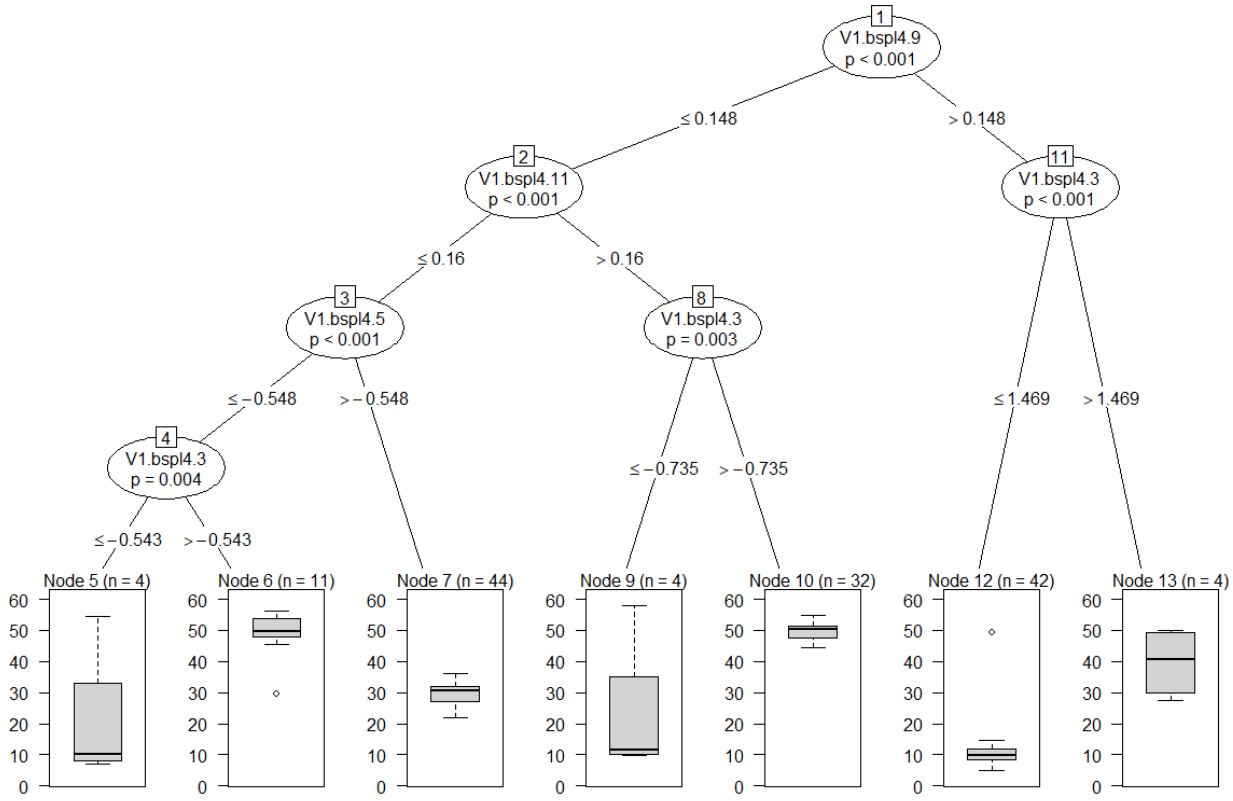


Figure 27: Output of regression tree with k-fold cross validation for simulated dataset

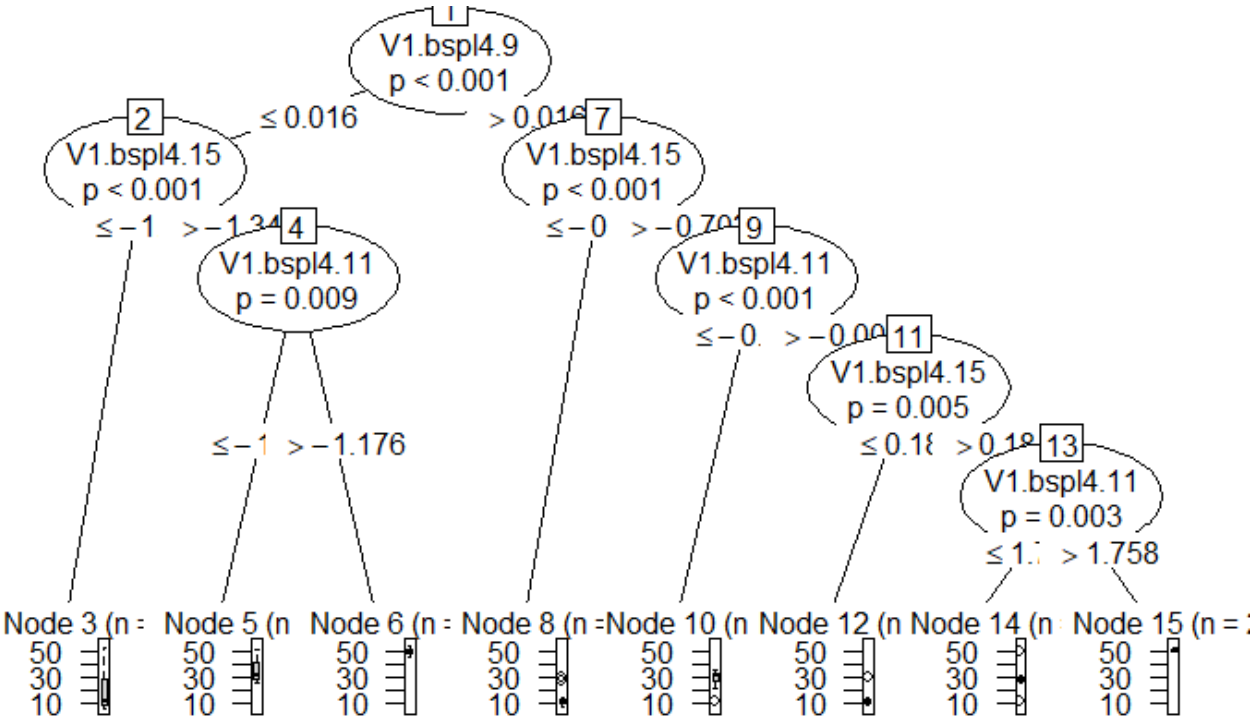


Figure 28: Output of regression tree with k-fold cross validation for simulated dataset

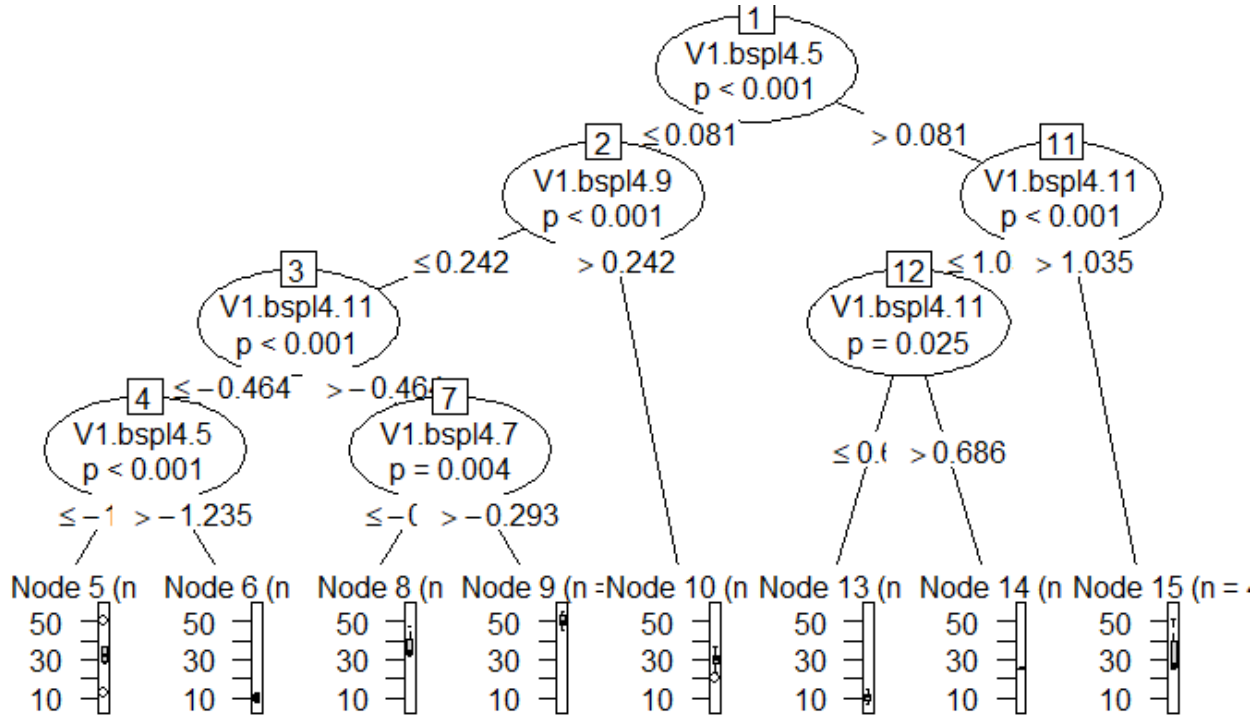


Figure 29: Output of regression tree with k-fold cross validation for simulated dataset

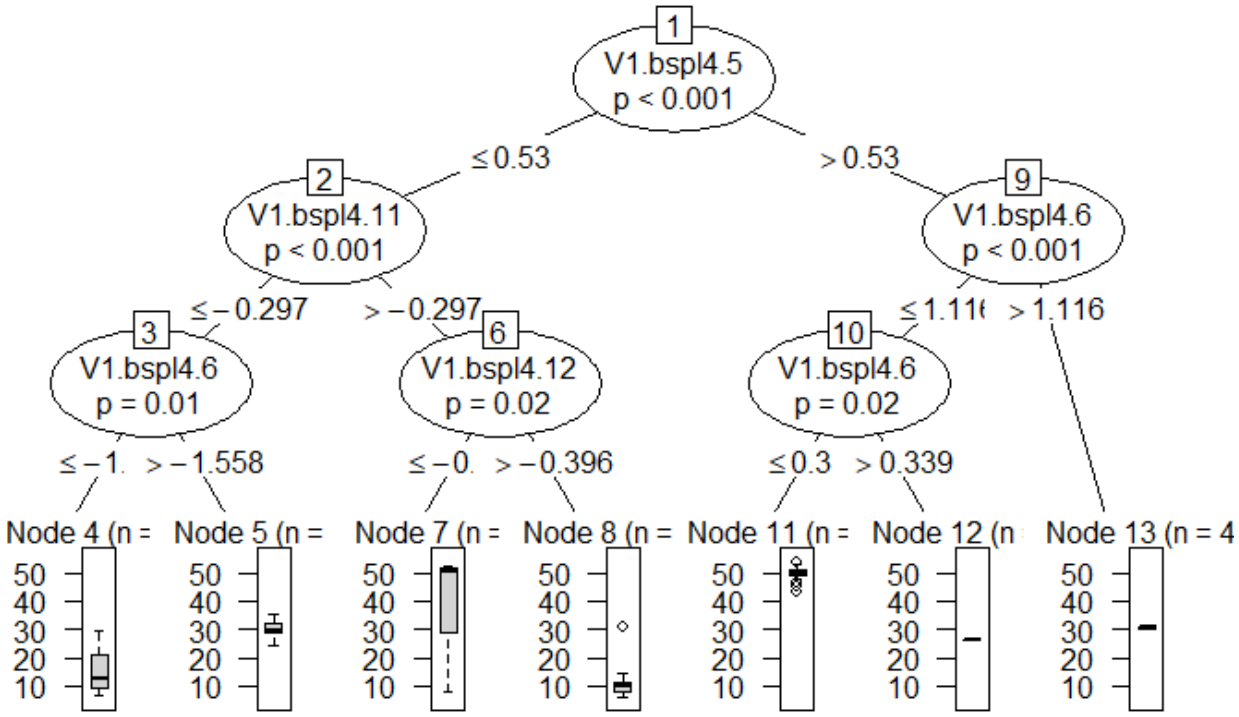


Figure 30: Output of regression tree with k-fold cross validation for simulated dataset

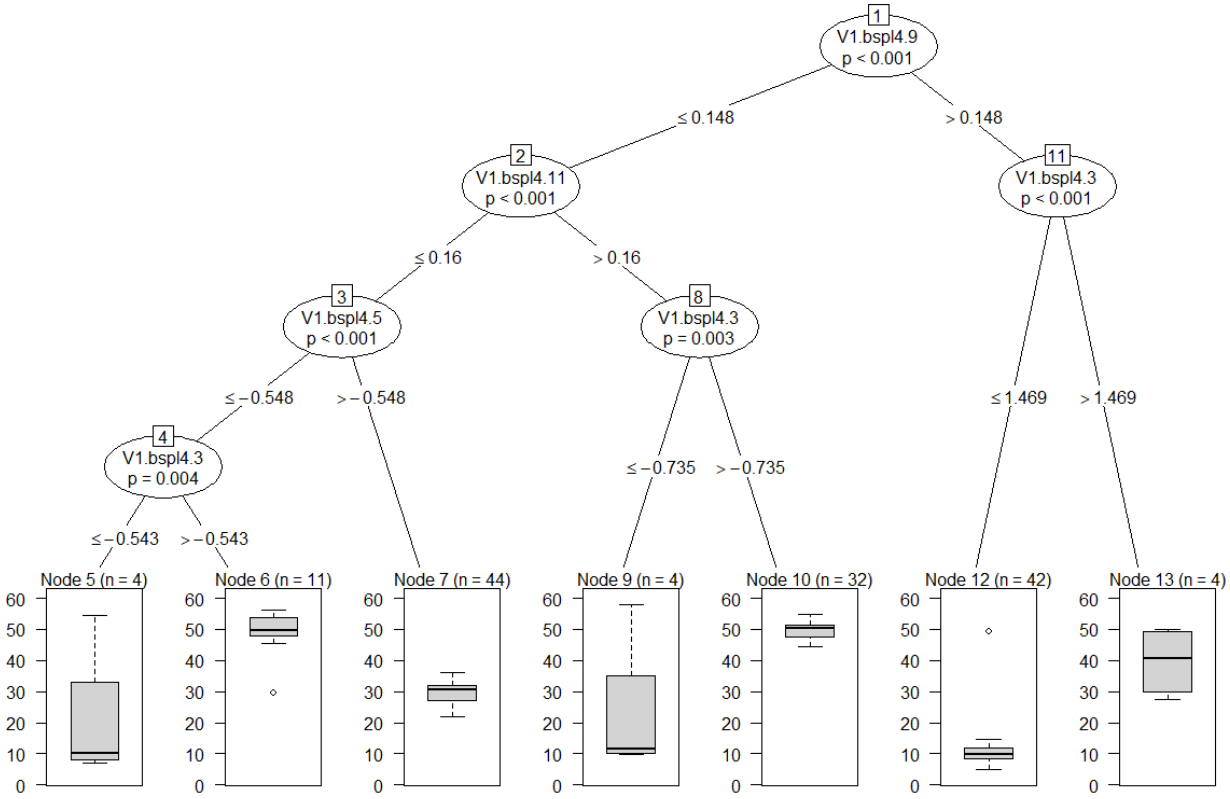


Figure 31: Output of regression tree with k-fold cross validation for simulated dataset

# 7 Appendix B

R code: [1]

```
library(fda.usc)
library(roahd)
library(energy)
library(entropy)
library(partykit)
library(cluster)

#Testing equal dsitributions
testingret <- function(x, y, R = 1000){
  if( is.factor(x) ){
    d_1 = daisy(as.data.frame(x))
  }
  if( is.numeric(x) ){
    d_1 = metric.dist(as.data.frame(x))
  }
  if( is.fdata(x) ){
    d_1 = metric.lp(x)
  }
  y = data.frame(y)


  d_2 <- metric.dist(y)
  ct <- dcor.test(d_1,d_2,R=R)
  if ( !is.na(ct$statistic) ){
    return(c(ct$statistic, ct$p.value))
  } else{
    NA
  }
}


#Testing for spliting of nominal variables
resample <- function(x, ...) x[sample.int(length(x), ...)]


Chitest<-function(x,y){
  x <- factor(x)
```

```
  if(length(levels(x))<2)return(NA)
  ct <- suppressWarnings(chisq.test(table(y,x), correct = FALSE))
  pchisq(ct$statistic, ct$parameter, log = TRUE, lower.tail = FALSE)
}

tree <- function(Y, data, weights = NULL,
                   minbucket = 1,
                   alpha = 0.05, R = 1000,
                   rnd.sel = T, rnd.splt = TRUE, nb=5) {


response <- data[[which(names(data)==Y)]]  # name of the response variable

  if ( is.null(weights) ) weights <- rep(1L,length(response))

  new_data <- list("response"=response)
  for(j in n.var){
    if(class(data[[j]])=="fdata")
      foo <- optim.basis(data[[j]], numbasis = nb)
    fd3 <- fdata_2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
    foo$coef <- t(fd3$coefs)
    new_data[[j]] <- foo
  }
  names(new_data)[-1] <- names(data)[-1]

  nodes <- tree_growing( id = 1L, response=new_data$response, data = new_data, weights,
                     minbucket = minbucket,
                     alpha = alpha, R = R,
                     rnd.sel = rnd.sel, rnd.splt = rnd.splt,
                     n.var = n.var)

  # Terminal node number for each observation
  response=data.frame(response)
  y=response
  m.coef <- c()
  for(j in n.var){
    foo <- new_data[[j]]$coef
    colnames(foo) <- paste(names(data)[j], colnames(new_data[[j]]$coef), sep = ".")
    m.coef <- cbind(m.coef,foo)
  }
```

```
  data_1=cbind(response,m.coef)
  m.coef=m.coef

  fitted <- fitted_node(nodes, data = data.frame(m.coef))
  formula=response~.

  ret <- party(nodes, data = data.frame(m.coef),
               fitted = data.frame("(fitted)" = fitted,
                                   "(response)" = data_1$response,
                                   "(weights)" = weights,
                                   check.names = FALSE),
               terms = terms(formula,data=data_1)
  )
  as.constparty(ret)
}

tree_growing <- function(id=1L, response, data, weights,
                     minbucket,
                     alpha, R,
                     rnd.sel, rnd.splt,n.var) {

  if (sum(weights) <= minbucket) return( partynode(id = id) )
  res <- split_find( response, data, weights,
                   alpha = alpha, R = R,
                   rnd.sel = rnd.sel, rnd.splt = rnd.splt,
                   n.var = n.var )

  sp <- res$sp
  varselect <- res$varselect

  if (is.null(sp)) return( partynode(id = id) )

  kid <- c()
  kid[which(data[[varselect]]$coef[,sp$varid]<=sp$breaks)] <- 1
  kid[which(data[[varselect]]$coef[,sp$varid]>sp$breaks)] <- 2

  if(all(kid==1) | all(kid==2)) return( partynode(id = id) )
  sum1 <- length(which(data[[varselect]]$coef[which(weights==1),
                                              sp$varid]<=sp$breaks))
  sum2 <- length(which(data[[varselect]]$coef[which(weights==1),
```

```
                                                sp$varid]>sp$breaks))
  if(sum1 == 0 | sum2 == 0) return( partynode(id = id) )
  nb=0
  for(i in n.var){
    k=data[[i]]$numbasis.opt
    nb=c(nb,k)
  }

  if(varselect!=min(n.var)){
    step=sum(nb[1:(max(n.var[which(n.var<varselect)]))])
    sp$varid=sp$varid+as.integer(step)
  }

  kids <- vector(mode = "list", length = max(kid, na.rm = TRUE) )

  for ( k_id in 1:length(kids) ){
    # select observations for current node
    w <- weights
    w[kid != k_id] <- 0
    # get next node id
    if(k_id > 1){
      myid <- max( nodeids( kids[[k_id - 1]] ) )
    } else{
      myid <- id
    }
    # Start recursion on this daugther node
    kids[[k_id]] <- tree_growing( id = as.integer(myid + 1), response, data, w,
                                  minbucket,
                                  alpha, R,
                                  rnd.sel, rnd.splt , n.var = n.var)
  }

  # return nodes
  return( partynode( id = as.integer(id), split = sp, kids = kids,
                     info = list( p.value = min( info_split(sp)$p.value, na.rm = TRUE ) )
  ) )
}


split_find <- function( response, data, weights,
```

```
                    alpha, R,
                    rnd.sel, rnd.splt, n.var ) {


# extract response values from data
y <- response[which(weights==1)]
p <- matrix(NA, nrow = length(n.var), ncol=2)
colnames(p) <- c("statistic","p-value")
p=sapply(n.var,function(i)testingret(x=data[[i]]$fdata.est[which(weights==1)],y=y,R=R))



# Bonferroni-adjusted p-value small enough
if ( all(is.na(p[2,])) ) return(NULL)

minp <- min(p[2,], na.rm = TRUE)
minp <- 1 - (1 - minp)^sum( !is.na(p[2,]) )
if ( minp > alpha ) return(NULL)

xselect <- n.var
if(length(which(p[2,] == min(p[2,], na.rm = T)))>1){
  xselect <- which.max(p[1,])+1
}else{
  xselect <- which.min(p[2,])+1
}

x <-  data[[xselect]]
if(is.list(x)){
  if(is.fdata(x$fdata.est)) x1=x$coef[which(weights==1),]
}

if(is.factor(x)){
  lev<-levels(x[drop = TRUE])
  if(length(lev) == 2){
    splitpoint <- lev[1]
  }else{
    comb <- do.call("c", lapply(1:(length(lev)-2),
                                function(x) combn(lev, x, simplify = FALSE)))
    xlogp <- sapply(comb, function(q) Chitest(x%in%q, y))
    splitpoint <- comb[[which.min(xlogp)]]
  }
```

```
  ##split into two groups
  splitindex <- !(levels(data[[xselect]])%in%splitpoint)
  splitindex[!(levels(data[[xselect]])%in%lev)] <- NA_integer_
  splitindex <- splitindex-min(splitindex, na.rm=TRUE)+1L
}
if(is.numeric(x)){
  splitindex <- s.opt(response, x[which(weights==1)], rnd.splt)
}
if(is.list(x)){
  if(is.fdata(x$fdata.est)){
    bselect <- 1:dim(x1)[2]
    p1 <- c()
    p1        <- sapply( bselect, function(i) testingret( x1[,i], y, R = R) )
    colnames(p1) <- colnames(x1)
    if(length(which(p1[2,] == min(p1[2,], na.rm = T))) > 1){
      bselect <- which.max(p1[1,])
    }else{
      bselect <- which.min(p1[2,])
    }


    splitindex <- s.opt(y=y, X=x1[, bselect], rnd.splt)
  }
}

#
if(is.numeric(x)){
  return( partysplit( varid = as.integer(xselect),
                      breaks = splitindex,
                      info = list( p.value = 1 - (1 - p)^sum( !is.na(p) ) )
  ) )
}
if(is.factor(x)){
  return(partysplit(varid = as.integer(xselect),
                    index = splitindex,
                    info = list( p.value = 1 - (1 - p)^sum( !is.na(p) ) )
  ) )
}
```

```
  if(is.list(x)){
    if(is.fdata(x$fdata.est)){
      return(list(sp = partysplit(varid = as.integer(bselect),
                                  breaks = splitindex,
                                  info=list( p.value = 1 - (1 - p[2,])^sum( !is.na(p[2,]) ) )
      ),varselect = xselect  ))
    }
  }
}

s.opt <- function(y, X, rnd = T){

  s  <- sort(X)
  obj <- c()
  for(i in 1:length(s)){
    data_1  <- y[ which(X <  s[i]) ]
    data_2  <- y[ which(X >= s[i]) ]
    v1=var(data_1)
    v2=var(data_2)
    n1=length(data_1)
    n2=length(data_2)
    n=n1+n2


    obj[i] = (n1*v1+n2*v2)/n
  }

  return( s[ which.min(obj) ] )
}

size <- c(50,50,50,50)

P <- 50
n.var <- 1

data_generation  <- function( size, P, n.var, alpha, beta,  sigma=0.1){


  nclass <- length(class)
```

```
data <- list()
for( i in 1:n.var){


  grid <- seq(0, 1, length.out = P)
  Cov <- exp_cov_function(grid, alpha = alpha[1,i], beta = beta[1,i])
  Data <- generate_gauss_fdata(size[1],
                               centerline = ((5*grid-2)/3)^2 - cos(2*pi*grid), Cov = Cov)
  fD_1 <- fData( grid, Data )

  Cov <- exp_cov_function(grid, alpha=alpha[2,i], beta=beta[2,i])
  Data <- generate_gauss_fdata(size[2],
                               centerline = -((5*grid-2)/3)^2 - cos(2*pi*grid, Cov = Cov)
  fD_2 <- fData( grid, Data )

  Cov <- exp_cov_function(grid, alpha=alpha[3,i], beta=beta[3,i])
  Data <- generate_gauss_fdata(size[3],
                               centerline = sin((5*pi*grid)/3), Cov = Cov)
  fD3 <- fData( grid, Data )

  Cov <- exp_cov_function(grid, alpha=alpha[4,i], beta=beta[4,i])
  Data <- generate_gauss_fdata(size[4],
                               centerline = -sin((5*pi*grid)/3), Cov = Cov)
  fD4 <- fData( grid, Data )

  eps1   <- matrix(rnorm(P*size[1], mean = 0, sd = sigma),
                   nrow = size[1], ncol = P)
  eps2   <- matrix(rnorm(P*size[2], mean = 0, sd = sigma),
                   nrow = size[2], ncol = P)
  eps3   <- matrix(rnorm(P*size[3], mean = 0, sd = sigma),
                   nrow = size[3], ncol = P)
  eps4   <- matrix(rnorm(P*size[4], mean = 0, sd = sigma),
                   nrow = size[4], ncol = P)

  k1 <- fD_1$values + eps1
  k2 <- fD_2$values + eps2
  k3 <- fD3$values + eps3
  k4 <- fD4$values + eps4
  k <- rbind(k1,k2,k3,k4)
```

```
    data[[paste("V",i,sep="")]] <- fdata(k)


    data[["Y"]] <- c(rnorm(50,-10,3),rnorm(50,10,3),rnorm(50,15,5),rnorm(50,-15,5))
  }

  return(data)

}

M=100
for(i in 1:M){
  alpha <- matrix( round(runif(4,0.1,1),2),
                   nrow=4, ncol=1)
  beta <- matrix( round(runif(4,0.1,1),2),
                  nrow=4, ncol=1)
  data[[i]]=data_generation(size=size,P=P,n.var=n.var,alpha=alpha,beta=beta,
                            sigma=0.5)
}

MEP_Bagg_100 <- c()

MEP_RT_100 <- c()

MEP_KF <- c()
#Bagging

for(i in 1:100){
  print(i)
  id.train_b <- sample(1:(4*size[1]), size=4*size[1],replace = TRUE)
  list.train_b <- lapply(data[[i]], function(x) {x[id.train_b]})
  list.test_b <- lapply(data[[i]], function(x) {x[-id.train_b]})
  nb <- 15
  REG_B <- mytree(Y="Y", data=list.train_b, weights = NULL,
                  minbucket = 5,
                  alpha = 0.05, R = 1000,
                  rnd.sel = T, rnd.splt = TRUE, nb=nb)
```

```
  plot(REG_B)

  test_y_b <- list.test_b$Y
  f.test_b <- list.test_b$V1
  foo <- optim.basis(f.test_b, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_b=predict(REG_b, newdata = m.coef)
  MEP_Bagg_100[i] <- (sum((test_y_b-y_pred_b)^2)/length(test_y_b))/(var(test_y_b))
}

getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

MeanMEP_Bagg100 <- mean(MEP_Bagg_100)
MedianMEP_Bagg_100 <- median(MEP_Bagg_100)
ModeMEP_Bagg_100 <- getmode(MEP_Bagg_100)
MeanMEP_Bagg_100
MedianMEP_Bagg_100
ModeMEP_Bagg_100



#Repeated Training
for(i in 1:100){
  print(i)

  id.train_r <- sample(1:(4*size[1]), size=0.7*4*size[1])
  list.train_r <- lapply(data[[i]], function(x) {x[id.train_r]})
  list.test_r <- lapply(data[[i]], function(x) {x[-id.train_r]})

  nb <- 15

  REG_RT <- mytree(Y="Y", data=list.train_r, weights = NULL,
                   minbucket = 10,
                   alpha = 0.05, R = 1000,
```

```
                  rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_RT)

  test_y_r <- list.test_r$Y
  f.test_r <- list.test_r$V1
  foo <- optim.basis(f.test_r, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_r=predict(REG_RT, newdata = m.coef)
  MEP_RT_100[i] <- (sum((test_y_r-y_pred_r)^2)/length(test_y_r))/(var(test_y_r))
}

MeanMEP_RT_100 <- mean(MEP_RT_100)
MedianMEP_RT_100 <- median(MEP_RT_100)
ModeMEP_RT_100 <- getmode(MEP_RT_100)
MeanMEP_RT_100
MedianMEP_RT_100
ModeMEP_RT_100

#K-fold Cross Validation
iddata<-1:length(data[[1]]$Y)
k<-5
ksize<-length(iddata)/5

for(i in 1:k){
  print(i)
  id.train <- sample(iddata, size=ksize, replace = FALSE)
  list.train <- lapply(data[[i]], function(x) {x[-id.train]})
  list.test <- lapply(data[[i]], function(x) {x[id.train]})
  iddata<-iddata[-id.train]
  nb <- 15

  REG_KF <- mytree(Y="Y", data=list.train, weights = NULL,
                   minbucket = 10,
                   alpha = 0.05, R = 1000,
                   rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_KF)
```

```
  test_y <- list.test$Y
  f.test <- list.test$V1
  foo <- optim.basis(f.test, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred=predict(REG_KF, newdata = m.coef)
  MEP_KF[i] <- (sum((test_y-y_pred)^2)/length(test_y))/(var(test_y))
}

MeanMEP_KF <- mean(MEP_KF)
MedianMEP_KF <- median(MEP_KF)
ModeMEP_KF <- getmode(MEP_KF)
MeanMEP_KF
MedianMEP_KF
ModeMEP_KF

#Original model
id.train_o <- sample(1:(4*size[1]), size=0.7*4*size[1])
list.train_o <- lapply(data[[i]], function(x) {x[id.train_o]})
list.test_o <- lapply(data[[i]], function(x) {x[-id.train_o]})

REG_o <- mytree(Y="Y", data=list.train_o, weights = NULL,
                  minbucket = 10,
                  alpha = 0.05, R = 1000,
                  rnd.sel = T, rnd.splt = TRUE, nb=nb)
plot(REG_o)


test_y_o <- list.test_o$Y
f.test_o <- list.test_o$V1
foo <- optim.basis(f.test_o, numbasis = nb)
fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
m.coef <- data.frame(t(fd3$coefs))
for(j in 1: dim(m.coef)[2]){
  names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
```

```
}
y_pred_o=predict(REG_o, newdata = m.coef)
MEP_o <- (sum((test_y_o-y_pred_o)^2)/length(test_y_o))/(var(test_y_o))
MEP_o


#Bagging M=1000
MEP_Bagg_1000 <- c()

MEP_RT_1000 <- c()

#Bagging

for(i in 1:1000){
  print(i)
  id.train_b <- sample(1:(4*size[1]), size=4*size[1],replace = TRUE)
  list.train_b <- lapply(data[[i]], function(x) {x[id.train_b]})
  list.test_b <- lapply(data[[i]], function(x) {x[-id.train_b]})
  nb <- 15
  REG_B_1 <- mytree(Y="Y", data=list.train_b, weights = NULL,
                    minbucket = 5,
                    alpha = 0.05, R = 1000,
                    rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_B_1)

  test_y_b <- list.test_b$Y
  f.test_b <- list.test_b$V1
  foo <- optim.basis(f.test_b, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_b=predict(REG_B_1, newdata = m.coef)
  MEP_Bagg_1000[i] <- (sum((test_y_b-y_pred_b)^2)/length(test_y_b))/(var(test_y_b))
}

MeanMEP_Bagg1000 <- mean(MEP_Bagg_1000)
MedianMEP_Bagg_1000 <- median(MEP_Bagg_1000)
ModeMEP_Bagg_1000 <- getmode(MEP_Bagg_1000)
```

```
MeanMEP_Bagg_1000
MedianMEP_Bagg_1000
ModeMEP_Bagg_1000


#Repeated Training
for(i in 1:1000){
  print(i)

  id.train_r <- sample(1:(4*size[1]), size=0.7*4*size[1])
  list.train_r <- lapply(data[[i]], function(x) {x[id.train_r]})
  list.test_r <- lapply(data[[i]], function(x) {x[-id.train_r]})

  nb <- 15

  REG_RT_1 <- mytree(Y="Y", data=list.train_r, weights = NULL,
                     minbucket = 10,
                     alpha = 0.05, R = 1000,
                     rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_RT_1)

  test_y_r <- list.test_r$Y
  f.test_r <- list.test_r$V1
  foo <- optim.basis(f.test_r, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_r=predict(REG_RT, newdata = m.coef)
  MEP_RT_1000[i] <- (sum((test_y_r-y_pred_r)^2)/length(test_y_r))/(var(test_y_r))
}

MeanMEP_RT_1000 <- mean(MEP_RT_1000)
MedianMEP_RT_1000 <- median(MEP_RT_1000)
ModeMEP_RT_100 <- getmode(MEP_RT_100)
MeanMEP_RT_1000
MedianMEP_RT_1000
ModeMEP_RT_1000
```

```
weather<-load("Weather_dataset.RData")
Y=weather$y$Temp
func.data=weather$reg.fdata
data=list(Y=Y, X=func.data)

MEP_Bagg_100 <- c()

MEP_RT_100 <- c()

MEP_KF <- c()
#Bagging

for(i in 1:100){
  print(i)
  id.train_b <- sample(1:(4*size[1]), size=4*size[1],replace = TRUE)
  list.train_b <- lapply(data[[i]], function(x) {x[id.train_b]})
  list.test_b <- lapply(data[[i]], function(x) {x[-id.train_b]})
  nb <- 15
  REG_B <- mytree(Y="Y", data=list.train_b, weights = NULL,
                  minbucket = 5,
                  alpha = 0.05, R = 1000,
                  rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_B)

  test_y_b <- list.test_b$Y
  f.test_b <- list.test_b$V1
  foo <- optim.basis(f.test_b, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_b=predict(REG_b, newdata = m.coef)
  MEP_Bagg_100[i] <- (sum((test_y_b-y_pred_b)^2)/length(test_y_b))/(var(test_y_b))
}

getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}
```

```
MeanMEP_Bagg100 <- mean(MEP_Bagg_100)
MedianMEP_Bagg_100 <- median(MEP_Bagg_100)
ModeMEP_Bagg_100 <- getmode(MEP_Bagg_100)
MeanMEP_Bagg_100
MedianMEP_Bagg_100
ModeMEP_Bagg_100


#Repeated Training
for(i in 1:100){
  print(i)

  id.train_r <- sample(1:(4*size[1]), size=0.7*4*size[1])
  list.train_r <- lapply(data[[i]], function(x) {x[id.train_r]})
  list.test_r <- lapply(data[[i]], function(x) {x[-id.train_r]})

  nb <- 15

  REG_RT <- mytree(Y="Y", data=list.train_r, weights = NULL,
                   minbucket = 10,
                   alpha = 0.05, R = 1000,
                   rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_RT)

  test_y_r <- list.test_r$Y
  f.test_r <- list.test_r$V1
  foo <- optim.basis(f.test_r, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_r=predict(REG_RT, newdata = m.coef)
  MEP_RT_100[i] <- (sum((test_y_r-y_pred_r)^2)/length(test_y_r))/(var(test_y_r))
}

MeanMEP_RT_100 <- mean(MEP_RT_100)
MedianMEP_RT_100 <- median(MEP_RT_100)
ModeMEP_RT_100 <- getmode(MEP_RT_100)
```

```
MeanMEP_RT_100
MedianMEP_RT_100
ModeMEP_RT_100

#K-fold Cross Validation
iddata<-1:length(data[[1]]$Y)
k<-5
ksize<-length(iddata)/5

for(i in 1:k){
  print(i)
  id.train <- sample(iddata, size=ksize, replace = FALSE)
  list.train <- lapply(data[[i]], function(x) {x[-id.train]})
  list.test <- lapply(data[[i]], function(x) {x[id.train]})
  iddata<-iddata[-id.train]
  nb <- 15

  REG_KF <- mytree(Y="Y", data=list.train, weights = NULL,
                   minbucket = 10,
                   alpha = 0.05, R = 1000,
                   rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_KF)


  test_y <- list.test$Y
  f.test <- list.test$V1
  foo <- optim.basis(f.test, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred=predict(REG_KF, newdata = m.coef)
  MEP_KF[i] <- (sum((test_y-y_pred)^2)/length(test_y))/(var(test_y))
}

MeanMEP_KF <- mean(MEP_KF)
MedianMEP_KF <- median(MEP_KF)
ModeMEP_KF <- getmode(MEP_KF)
MeanMEP_KF
```

```
MedianMEP_KF
ModeMEP_KF

#Original model
id.train_o <- sample(1:(4*size[1]), size=0.7*4*size[1])
list.train_o <- lapply(data[[i]], function(x) {x[id.train_o]})
list.test_o <- lapply(data[[i]], function(x) {x[-id.train_o]})

REG_o <- mytree(Y="Y", data=list.train_o, weights = NULL,
                minbucket = 10,
                alpha = 0.05, R = 1000,
                rnd.sel = T, rnd.splt = TRUE, nb=nb)
plot(REG_o)


test_y_o <- list.test_o$Y
f.test_o <- list.test_o$V1
foo <- optim.basis(f.test_o, numbasis = nb)
fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
m.coef <- data.frame(t(fd3$coefs))
for(j in 1: dim(m.coef)[2]){
  names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
}
y_pred_o=predict(REG_o, newdata = m.coef)
MEP_o <- (sum((test_y_o-y_pred_o)^2)/length(test_y_o))/(var(test_y_o))
MEP_o


#Bagging M=1000
MEP_Bagg_1000 <- c()

MEP_RT_1000 <- c()

#Bagging

for(i in 1:1000){
  print(i)
  id.train_b <- sample(1:(4*size[1]), size=4*size[1],replace = TRUE)
  list.train_b <- lapply(data[[i]], function(x) {x[id.train_b]})
  list.test_b <- lapply(data[[i]], function(x) {x[-id.train_b]})
```

```
  nb <- 15
  REG_B_1 <- mytree(Y="Y", data=list.train_b, weights = NULL,
                    minbucket = 5,
                    alpha = 0.05, R = 1000,
                    rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_B_1)

  test_y_b <- list.test_b$Y
  f.test_b <- list.test_b$V1
  foo <- optim.basis(f.test_b, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_b=predict(REG_B_1, newdata = m.coef)
  MEP_Bagg_1000[i] <- (sum((test_y_b-y_pred_b)^2)/length(test_y_b))/(var(test_y_b))
}

MeanMEP_Bagg1000 <- mean(MEP_Bagg_1000)
MedianMEP_Bagg_1000 <- median(MEP_Bagg_1000)
ModeMEP_Bagg_1000 <- getmode(MEP_Bagg_1000)
MeanMEP_Bagg_1000
MedianMEP_Bagg_1000
ModeMEP_Bagg_1000


#Repeated Training
for(i in 1:1000){
  print(i)

  id.train_r <- sample(1:(4*size[1]), size=0.7*4*size[1])
  list.train_r <- lapply(data[[i]], function(x) {x[id.train_r]})
  list.test_r <- lapply(data[[i]], function(x) {x[-id.train_r]})

  nb <- 15

  REG_RT_1 <- mytree(Y="Y", data=list.train_r, weights = NULL,
                     minbucket = 10,
                     alpha = 0.05, R = 1000,
```

```
                    rnd.sel = T, rnd.splt = TRUE, nb=nb)
  plot(REG_RT_1)

  test_y_r <- list.test_r$Y
  f.test_r <- list.test_r$V1
  foo <- optim.basis(f.test_r, numbasis = nb)
  fd3 <- fdata2fd(foo$fdata.est, type.basis = "bspline", nbasis = foo$numbasis.opt)
  m.coef <- data.frame(t(fd3$coefs))
  for(j in 1: dim(m.coef)[2]){
    names(m.coef)[j] <- paste("V1",names(m.coef)[j],sep = ".")
  }
  y_pred_r=predict(REG_RT, newdata = m.coef)
  MEP_RT_1000[i] <- (sum((test_y_r-y_pred_r)^2)/length(test_y_r))/(var(test_y_r))
}

MeanMEP_RT_1000 <- mean(MEP_RT_1000)
MedianMEP_RT_1000 <- median(MEP_RT_1000
ModeMEP_RT_1000 <- getmode(MEP_RT_1000)
MeanMEP_RT_1000
MedianMEP_RT_1000
ModeMEP_RT_1000
```