SAPIENZA
UNIVERSITÀ DI ROMA

Classification and regression energy tree with network predictors

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea Magistrale in Data Science

Candidate
Gabriel Nespoli
ID number 1743585

Thesis Advisor
Prof. Pierpaolo Brutti

Co-Advisor
Riccardo Giubilei

Academic Year 2018/2019

Classification and regression energy tree with network predictors
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: gnmesquita1@gmail.com

Acknowledgement

I would like to thank firstly my Advisor Professor Pierpaolo Brutti, not only for all the support on this thesis, but specially for helping me to solve the many bureaucratic issues that I came across until the graduation.
My thanks go also to Riccardo Giubilei for the tips, corrections and suggestions from the beginning to the end of this work.
For sure I must thank all the friends I made in Italy. My times were easier and joyful with them.
I would like to express my sincere gratitude to my family, even far away, I know it was there with me.
For sure I would not have reached far without Sara. Her patience, support and encouragement throughout my years of study were essential for me to achieve this goal.
Above everything and everyone, I thank God for allowing me to have lived and learned in such wonderful times.

Abstract

Learning on graph data is a application of machine learning and a significant amount of recent research efforts have been devoted to it. Applications on this area is seen on identification of substructures and hidden patterns, signal recovery and prediction, for example. This work differs from standard graph learning methods because it uses graphs as covariates of a decision tree model. CART [7] and conditional inference trees [17] are also tree-based methods, but only accepts continuous or labeled data as variables. The tree-like model proposed in this work differs from *ctree* by applying an energy statistic [30] as measure of association between the covariates and the response. This statistic, named distance correlation [31], is a metric of statistical distance between observations that can be numerical, as classical problems, but also complex data, such as functional or networks. We extend the energy tree developed in [6] by enabling the use of network data. We tested our algorithm on simulated and real data examples, and compared to *ctree* and CART.

Contents

List of Figures

List of Tables

Introduction

The use of machine learning for predictions and inferences is a hype and still ascending nowadays due to the huge volume and the complexity of data generated in almost all day-to-day activities of people, machines, industries and governments. Two tasks in the machine learning field are of major importance: regression and classification. Regression models aim to predict continuous values, while in classifications, the goal is to predict a discrete output variable.

Decision tree is one family of such models that works for regression and classification problems. Decision tree imposes tests on attributes of the training data, splitting its observations in branches, that represent the outcome of the test, assigning a single value to all data points that belong to the same branch. This value can be a label or a continuous value, depending if it is a classification or a regression problem, respectively.

CART [7] is an example of such algorithms. It is a binary recursive partitioning method that processes labeled or continuous dependent or independent variables. CART has no stop conditions, so it grows tree to a stage that every data point in the training data becomes a leaf. To avoid overfitting, the algorithm employees a bottom-up pruning strategy, where branches that least contribute to the performance on training data are cut. CART decides whether to split or not based on Gini measure of impurity. This method has a drawback, as sentenced by its own author: *"variable selection is biased in favor of those variables having more values and thus offering more splits"*. To work around this problem, [17] developed a method that uses permutation test for an unbiased selection of covariates in order to decide for splits. Moreover, no pruning is needed to deal of overfitting because the procedure stops when there is not significant association between any of the covariates and the target variable. Many methods are proposed to extract information from graph data, like patterns and substructures, or aiming to predict graph signals. [23] summarizes graph-based induction as well as presents some applications. In [34], linear regression is used to approximate an underlying graph from its graph signals.

None of the aforementioned techniques can be used to make predictions when networks are used as covariates. In this work, we developed a tree-based model for classification and regression with unbiased method for feature and index selection for the splits when the covariates are graphs (network and graph have the same meaning and are used interchangeably). We extended the innovative method developed in [6] that proposed a machine learning algorithm for categorical and continuous inferences in the case that the covariates are functions.

Despite [17] that employees the permutation test as statistical significance test of association between independent variables and the response, this present work makes use of an energy statistics introduced in [31], known as distance correlation. Distance correlation is a measure to test whether two random vectors X and Y are independent and is zero only if this condition occurs.

The thesis is organized as follows. In Chapter 1 there is an introduction to the main concepts of tree-based methods and k-cores decomposition. In in Chapter 2 Energy Tree, energy statistics, distance covariance and correlation are introduced. Chapter 3 shows the application of the Energy Tree on several scenarios: classification and regression, univariate and multivariate, network and functional data.

Chapter 1

Preliminaries

Generally in predicting problems one aims to predict a dependent variable (or target variable) $y \in Y$ given a new value for the independent variable (or the input vector) $\mathbf{x} \in X^p$. If y comprises continuous values, this is a regression problem. Instead, if y accepts only k values (named classes or labels), then it is a classification problem.

There is a set of well-known algorithms for regression (Linear Regression, Ridge [16], Lasso [32], etc.) as well as for classification (Logistic Regression, SVM [11], Naive Bayes classifier [3] , etc.). Tree-based models are another famous and intuitive class of algorithms to solve both classification and regression problems.

Tree structured models or, more frequently, binary tree models, are built by recursive splits of X, the training data, into two subsets, let's say $X2$ and $X3$. In the tree structure, the initial non-partitioned X is called *root node* while the the two subsets are called *children nodes* of the upper node. $X2$ and $X3$ are disjoint and they form a partition of X. If one continues to split $X2$ and $X3$ repeatedly and stops at a certain point, the visual outcome of this successive binary splits is a tree (see Figure 1.1 [21]). When no further splits of a node are done, this is a *terminal node*. In consequence of series of splits in disjoint sets, the terminal subsets form a partition of X. In the example of Figure 1.1, we have

$$X = X8 \cup X14 \cup X15 \cup X10 \cup X11 \cup X6 \cup X12 \cup X16 \cup X17.$$

In a classification problem, a label is assigned to each terminal node. Different terminal nodes can have the same label. Analogously, for regression problems, a continuous quantity is assigned, instead of labels.

The strategy of how to make the split is what differentiates the tree-based algorithms. In Section 1.1, a famous tree-based model is presented: decision trees.

1.1 Decision Trees

Decision trees, or Recursive Partitioning [18], are sorts of machine learning algorithms for classification and regression that work splitting the data points of a data set into regions of the predictor space according to some conditions or tests. The observations that fall into the same region share a specific pattern with the neighbors and after a conditional test the outcomes are mutually exclusive [25]. To elucidate, check

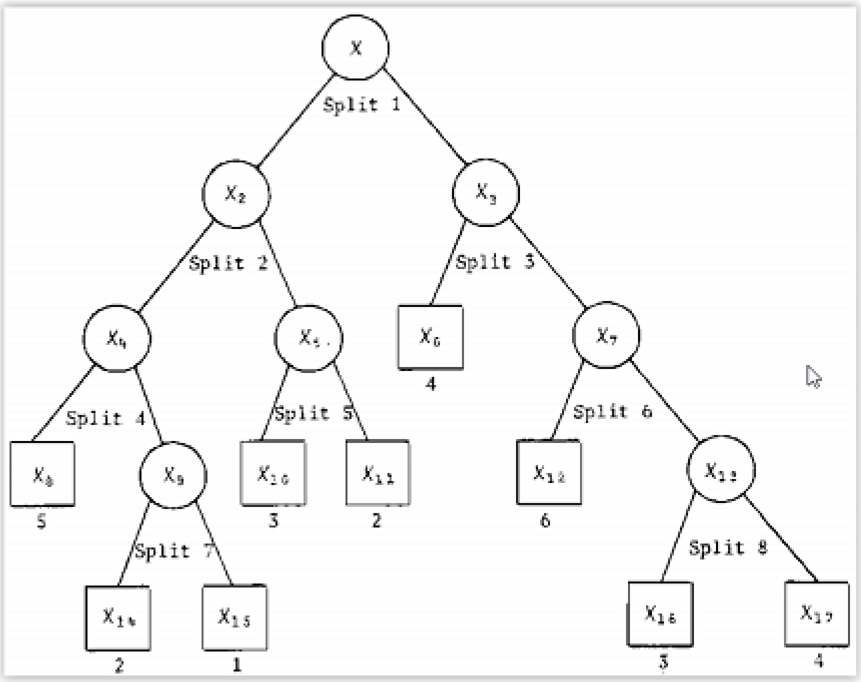


Figure 1.1. Result of 8 splits of set X

Figure 1.2 [5]. All data points in region A have the common properties of $x_1 < \theta_1$ and $x_2 < \theta_2$.

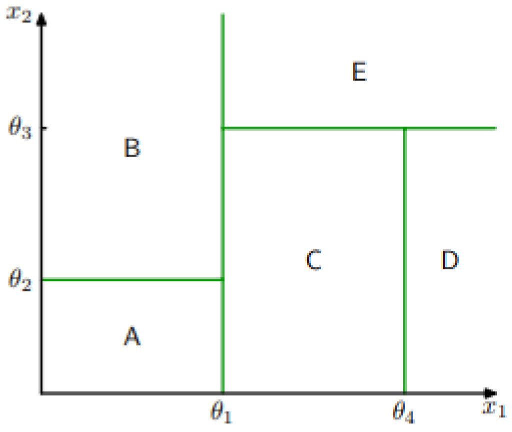


Figure 1.2. Illustration of a two-dimensional predictor space that has been partitioned into five regions

The splitting process of a branch is repeated exhaustively and recursively till one of the following condition is met:

(i) the data points in a region reach a predefined minimum number of observations;

(ii) the depth of the tree reaches a predefined maximum;

(iii) the data points in a region are all of the same label or range of values;

(iv) there is no more features to split;

(v) predefined stop condition - a statistical test of similarities [7] or an error tolerance of observations labels/values in a region.

When the recursion stops and the algorithm builds the tree, the learning phase ends. The prediction step traverses the same tree structure using the test set, splitting the data points into the regions known in the training phase. In other words, it divides the predictor space into the some regions found during the training. The predicted outcome of a region is calculated usually taking the mean (for regressions) or the mode (for classifications) and the same output is attributed to all observations of a region [14].

The first test node is called root and the regions are called terminal nodes or leaves of the tree. Intermediate tests are called internal nodes [14].

Each splitting condition of the predictor space can be univariate or multivariate. The former tests just one covariate, while the latter tests several features at once. If the splitting divides the vector space into two branches, the end result is a binary tree [25].

An example of binary classification decision tree, with univariate tests, is present in Figure 1.3 in which the algorithm classifies the quality of papayas based on two variables: color and softness [27].

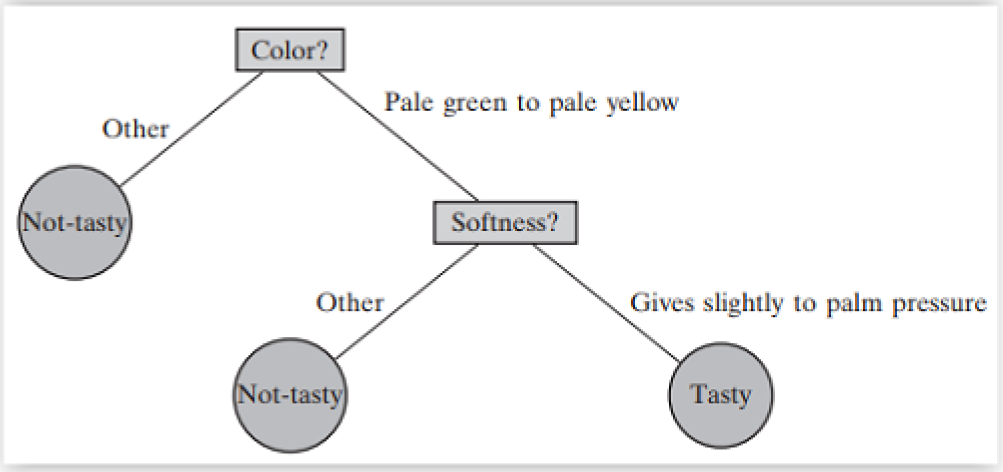


Figure 1.3. Example of decision trees applied to papayas quality problem

Even though modern machine learning algorithms outperform decision trees in terms of efficacy, they are still in use due to its simplicity and ease of interpretation [14].

There are two concerns regarding to decision trees: the selection of the covariate to split and the splitting point. A popular splitting rule is based on imposing a threshold θ on a single feature of x. Observations are moved to the right based on $1_{x_i>\theta}$, or to the left according to $1_{x_i<=\theta}$ where i is the feature selected [27].

Both x_i and θ are commonly chosen to minimize the *node impurity*, that can be calculated by the Residual Sum of Squares (RSS)(for regression problems) or the *classification error rate* (for classification problems), which is the fraction of the observations in R_j that do not belong to the most common class in j. RSS is given by

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where J is the set of regions and $\hat{y}_{R_j}$ is the mean output of the training observations in the j-th box [14].

A usual alternative metric to the classification error rate is the *Gini index*, which is used in CART (Classification and Regression Trees - a popular decision tree algorithm) [7]. Gini index measures the inequality or dispersion in a group or region and is given by

$$I_G(p) = 1 - \sum_{i=1}^{C} p_i^2$$

where C is the total number of classes and p_i is the proportion of class i in the set. The goal is to go through all covariates and all possible splits and choose the minimizing Gini index.

1.2 Pruning

Large and complex trees may lead to excellent performance on the training data, but may lead to poor predictions on the test set instead. This is a classic case of overfitting in tree-based methods, where these trees mimic the training pattern or reflect the presence of noise or outliers. If one leaves trees grow indefinitely, each node becomes a leaf.

Usually, the pruning performs a bottom-up walk on the tree, where internal nodes may be replaced by its lower subtrees or one of its leaf nodes [27].

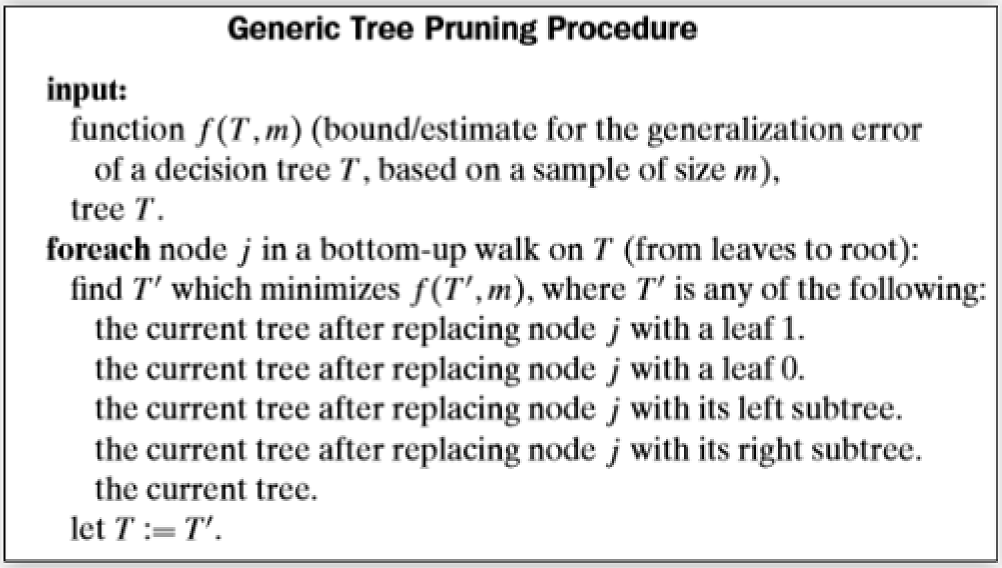
Taking the best subtree T' of T would need an evaluation of all possible subtrees of T, and this is costly. Instead, it is preferable to evaluate just a subset of the subtrees of T [14].

A solution is presented in [21] with the *weakest link pruning* technique. In this post-pruning approach, branches a removed from a fully grown tree. An internal node is replaced by the most-frequent node (for classification) or the mean value of the nodes (for regression) of the lower subtree, becoming a leaf node. The cost function is re-run and if the pruning leads to a predefined acceptable increase in the generalization error, then the pruned tree takes the place of original tree.

A pseudo-code of a generic post-pruning is present in Figure 1.4 [27].

1.3 Conditional Inference Trees

There are three major problems that affected the efficiency and the interpretability of tree-based models that employs the exhaustive search procedure to find splitting points [18]:



```
Generic Tree Pruning Procedure

input:
  function $f(T,m)$ (bound/estimate for the generalization error
    of a decision tree $T$, based on a sample of size $m$),
  tree $T$.
foreach node $j$ in a bottom-up walk on $T$ (from leaves to root):
  find $T'$ which minimizes $f(T',m)$, where $T'$ is any of the following:
    the current tree after replacing node $j$ with a leaf 1.
    the current tree after replacing node $j$ with a leaf 0.
    the current tree after replacing node $j$ with its left subtree.
    the current tree after replacing node $j$ with its right subtree.
    the current tree.
  let $T := T'$.
```

Figure 1.4. Pseudo-algorithm for post-tree-pruning

1. overfitting: in the lack of a good stop condition the tree may grow until every data point is a leaf node.

2. biased selection of features to split: exhaustive searching trees algorithms tend to biased select features with many possible splits or missing values

3. lack of statistical approach because looping through all features and all data points searching for the best split may approximate more to brutal force than to Statistics

The overfitting can be solved efficiently with pruning, but the other two problems remain unsolved for the exhaustive search procedure.

[18] proposed a tree-based machine learning model that does an unbiased selection of variables with a statistical test for the splits, solving two of the aforementioned deficiencies, without losing predictive performance.

The method is based on permutation tests [29] that measure several times the association between the response and covariates, leading to an unbiased selection of covariates independently of their scale.

Furthermore, a global hypothesis test of dependence expressed by H_0 is used to check if the recursion should stop or continue if there is still a significant similarity between any covariate and the response. This global hypothesis is expressed in terms of m partial hypotheses $H_0^j : D(Y|X_j) = D(Y)$, which is $H_0 = \bigcap_{j=1}^{m} H_0^j$.

Before each splitting in a node, the algorithm checks for all the m covariates a partial hypothesis $H_0^j : D(Y|X_j) = D(Y)$, where $D(Y|X_j)$ is the conditional distribution of Y given X_j.

Classification trees perform recursive binary partitioning of the covariate space by means of statistical tests. Consider the response variable $\mathbf{Y}$, the m-dimensional covariate vector $\mathbf{X} = (X_1, ..., X_m)$ and $\mathbf{w} = (w_1, ..., w_n)$, where n is the sample size and **w** is the vector of case weights. Each node of the tree is represented by **w**, that takes non-zero values in the index of n for observations that are elements of the node and zero otherwise. Since all observations belong to the root node, initially **w** has no zero value. The algorithm works as follow:

1. For **w**, test the global null hypothesis of independence between any of the m covariates and the response. Select the $j \star th$ covariate $X_{j\star}$ with strongest association to **Y** or stop if this hypothesis cannot be rejected.

2. Choose $A\star \subset \chi_{j*}$ splitting $\chi_{j\star}$ into two disjoint sets.

3. Modify **w** setting $w_{left,i} = w_i I(X_{j\star i} \in A\star)$ and $w_{right,i} = w_i I(X_{j\star i} \notin A\star)$, $i = 1, ..., n$ and I(.) denotes the indicator function.

4. Repeat 1 to 3.

The result is a tree structure which represent the partition of the predictor space χ into B_r disjoint cells, where $\chi = \bigcup\limits_{k=1}^{r} B_k$.

Even overcoming the problems of trees with exhausted searching strategy listed in the beginning of the session, according [18], the prediction performance is equivalent to optimally pruned trees.

1.4 K-Cores Decomposition

The use of cores in graph theory has as purpose to simplify the structure of graphs [4]. They can be used for many goals, such as graph coloring [12], graph clustering [15], protein function prediction [35] and identification of patterns and anomalies in graphs [28].

The k-core of a graph G is the maximal induced subgraph $H \subseteq G$ such that $\delta(G) \geq k$. Therefore, all nodes of H have at least k neighbors in H. The k-cores of a graph are unique and nested [4]. The k-shell is the set of vertices that are part of the k-core, but not of the $(k+1)$-core. The nestedness of the k-cores of a graph G is depicted in Figure 1.5 [13].

The core of maximum order is referred to as the main core or the highest k-core of the graph [2]. It is also known as the degeneracy of the graph. The degeneracy of a graph is the maximal value of k such that the k-core is not empty [10]. It is possible to find the k-core of a graph by removing iteratively all nodes (and their links) with degree less than k. The process of computing all k-cores of a graph G, for $k = 1, 2, 3, ..., k_{max}$ (where k_{max} is the degeneracy of the graph), is the called k-core decomposition.

In [20] a tool is presented to simplify graphs preserving the graph structure, using the shell distribution vector, a way of summarizing the k-core decomposition. This shell distribution is used to model the core structure of graphs, which is then useful to sample from the vector space of graphs given a specific shell distribution vector.

Each node of a graph can be contained in more than one k-core. The largest k that a node can reach is the *shell index* of that node. Given the vertex v of a graph

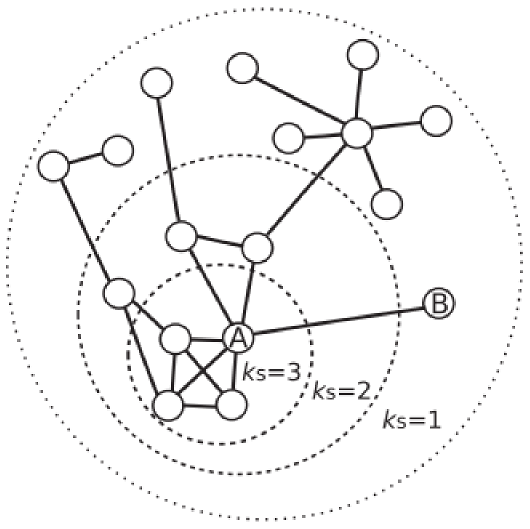


Figure 1.5. Nestedness of the k-core decomposition. Nodes inside the outer ring and outside the intermediate ring form the k-shell = 1. Vertices between the two extreme rings have k-shell = 2. The inner circle includes the nodes with k-shell = 3

g and its k-core H_k, the shell index of v is i if $v \in H_i(g)$ but $v \notin H_{i+1}(g)$. As stated in [20], the shell index of a vertex v indicates the highest core to which v belongs.

For example, on the left of Figure 1.6, the original graph g is presented, its 2-core in the middle, its 3- and 4-core on the right. Consider now the node painted red, it belongs to the 0-core (the original graph itself), 1-core, 2-core, 3-core and 4-core, but not to the 5-core (the empty graph). Thus, the shell index of this node is 4.

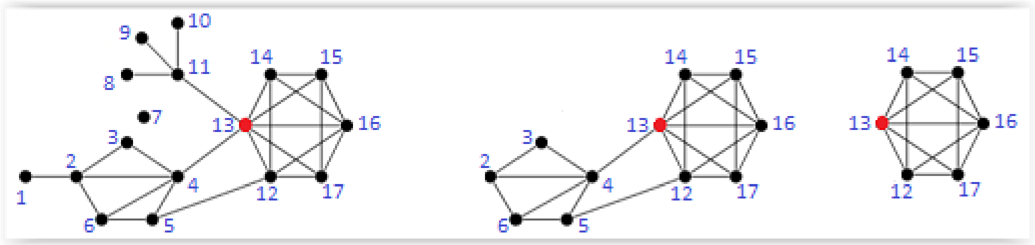


Figure 1.6. The original graph g (left), its 2-core (center), and its 3- and 4-core (right) (adapted from [20])

This present work employs the *shell distribution* to store all of the shell indices of a graph. The shell distribution $n(g)$ of a graph g with n vertices is a vector of length n whose i^{th} position $n_i(g)$ is the number of vertices of g with shell index equal to i, for $0 \leq i \leq n$ - 1.

[20] also represents the shell distribution as

$$n(g) := (n_0(g), n_1(g), ..., n_{n-1}(g)).$$

For example, the shell distribution of the graph in Figure 1.6 is given by the vector

$$n(g) = (1, 5, 5, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0).$$

In order to save memory space, in this present work the vector is cut before the tail of the vector full of zeros, because it is clear that the last non-zero value is the largest shell index of the graph. Therefore, $n(g)$ would be reduced to $n(g) = (1, 5, 5, 0, 6)$

Chapter 2

Energy Trees

None of the algorithms mentioned in Chapter 1 is general enough to be applied on graph data. In this work, a new method is proposed for classification and regression using energy distance to quantify the association between the response variable and network covariates. A comparison can be drawn between this new framework (energy trees) and the conditional inference trees, explained in Section 1.3. *Ctrees* avoid the exhaustive search of the best splits employing permutation tests between the covariates and the response variable, whilst energy trees make use of *distance correlation* (an energy statistic) [31], detailed in section 2.1.

2.1 Energy Statistics

Energy statistics is described as functions of distances between statistical observations. Statistical observations are considered heavenly bodies governed by a statistical potential energy, which is zero when an underlying statistical null hypothesis is true [30]. One should notice the proximity to Newton's gravitational potential energy concept, which is use to calculate the difference in stored energy of two objects in different heights. Usually, a zero value is assigned to the reference object. The authors in [30] state that energy statistics is more general and powerful than classical statistics, such as correlation, F-statistics, etc.

It is possible to make predictions using energy distance, this is called *energy inference*. In classical statistics, in many algorithms to make inferences, such as k-means [22], support vector machines (SVM) [11], Principal Component Analysis (PCA) [19], etc., there is the idea of distances between data points, or data points and the ground truth, that, in the end, is used to quantify the extent to which the model fits the data or is a good cluster. Specially for regression problems, the goal is to minimize the L1 or L2 distances between the predicted values and the target values. In other words, a loss function L is needed to penalize errors in prediction of the generated values [33]. This loss function could be written as

$$L(y, \hat{y}) = d(y, \hat{y})$$

where y is the ground truth, $\hat{y}$ is the predicted value and $d(.,.)$ is a distance function.

This idea can be applied when data is real numbers, but it can be cumbersome or even impossible for complex data, such as graphs or functions. If one wants to make predictions on these complex data, the problem can be overcome if one changes the vector space of the observations to a metric space [9] and work directly with non-negative distances [6]. In doing so, one can move from the graph vector space, for instance, to real numbers and then make inferences with graphs as predictors.

The energy distance between two independent random variables X and Y is defined as

$$\varepsilon(X, Y) = 2E|X - Y| - E|X - X'| - E|Y - Y'|. \tag{2.1}$$

X' and Y' are independent and identically distributed copies of X and Y respectively and $E|X| < \infty$, $E|Y| < \infty$. The two random variables must have the same dimension as well. Energy distance accepts only real and non-negative values or, in symbols,

1. $\varepsilon(X, Y) \geq 0$
2. $\varepsilon(X, Y) = 0 \iff X = Y$

The energy distance can be even more generalist replacing the differences inside the expected value equations by a generic distance function δ as defined in [30] as follows

$$\varepsilon(X, Y) = 2E|\delta(X, Y)| - E|\delta(X, X')| - E|\delta(Y, Y')|. \tag{2.2}$$

This last equation allows inferences in any complex metric space, needing only to build a notion of distance between observations. This idea was used in [6] to do classification and regression with functional data as predictors, and in this present work, with network covariates. In this work it will be detailed distance correlation and distance covariance in Section 2.2. Refer to [30] for more energy statistics.

2.2 Distance Covariance and Correlation

The distance correlation (*dCor*) is a measure to test the independence of two random vectors X and Y, written as $\mathcal{R}(X, Y)$. It is analogous to Pearson's correlation ρ, but the authors claim that distance correlation is more general than the former, because when the latter is zero, it does characterize independence of random vectors, while Pearson's correlation does not [31]. In detail of what happens with Pearson's correlation:

$$X, Y \ independent \implies \rho(X, Y) = 0$$

$$\rho(X, Y) = 0; \ X, Y \ uncorrelated \ \not\Longrightarrow \ X, Y \ independent$$

This problem does not happen with distance correlation, because $0 \leq \mathcal{R} \leq 1$ and $\mathcal{R} = 0$ only if X and Y are independent. Due to this, [24] considers that distance covariance (and by extension, distance correlation) is a *bona fide* measure for variables dependency.

Given two random vectors X and Y in arbitrarily dimensions $\mathbb{R}^p$ and $\mathbb{R}^q$, respectively, the distance covariance is given by the square root of

$$\mathcal{V}^2(X,Y) = \left\|\hat{f}_{X,Y}(t,s) - \hat{f}_X(t)\hat{f}_Y(s)\right\|^2 \tag{2.3}$$

where $\hat{f}_{X,Y}$ is the joint characteristic function of X and Y, $\hat{f}_X$ and $\hat{f}_Y$ are the characteristic function of X and Y, respectively. $\mathcal{V}^2(X,Y) \geq 0$, but assumes zero only if X and Y are independent. Similarly, the distance variance ($dVar$) is given by

$$\mathcal{V}^2(X) = \left\|\hat{f}_{X,X}(t,s) - \hat{f}_X(t)\hat{f}_X(s)\right\|^2. \tag{2.4}$$

Furthermore, the distance correlation is also defined in [30]:

$$\mathcal{R}^2(X,Y) = \begin{cases} \frac{\mathcal{V}^2(X,Y)}{\sqrt{\mathcal{V}^2(X)\mathcal{V}^2(Y)}}, & \text{if } \mathcal{V}^2(X)\mathcal{V}^2(Y) > 0. \\ 0, & \text{if } \mathcal{V}^2(X)\mathcal{V}^2(Y) = 0. \end{cases} \tag{2.5}$$

It is possible to numerically approximate to the true characteristic functions specified above by the sample characteristic function, denoted generically by $\hat{f}^n$. If $(\mathbf{X}, \mathbf{Y})$ is a sample from the joint distribution of (X, Y), the sample distance covariance is given by the square root of

$$\mathcal{V}_n^2(\mathbf{X},\mathbf{Y}) = \left\|\hat{f}^n_{X,Y}(t,s) - \hat{f}^n_X(t)\hat{f}^n_Y(s)\right\|^2, \tag{2.6}$$

where in [31] is further extended to

$$\mathcal{V}_n^2(\mathbf{X},\mathbf{Y}) = S_1 + S_2 - 2S_3, \tag{2.7}$$

$$S_1 = \frac{1}{n^2}\sum_{k,l=1}^{n} |X_k - X_l|_p |Y_k - Y_l|_q, \tag{2.8}$$

$$S_2 = \frac{1}{n^2}\sum_{k,l=1}^{n} |X_k - X_l|_p \frac{1}{n^2}\sum_{k,l=1}^{n} |Y_k - Y_l|_q, \tag{2.9}$$

$$S_3 = \frac{1}{n^3}\sum_{k=1}^{n}\sum_{l,m=1}^{n} |X_k - X_l|_p |Y_k - Y_m|_q. \tag{2.10}$$

The sample distance covariance $\mathcal{V}_n^2(\mathbf{X},\mathbf{Y})$ is also given by

$$\mathcal{V}_n^2(\mathbf{X},\mathbf{Y}) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{kl}B_{kl}. \tag{2.11}$$

Given a random sample $(\mathbf{X},\mathbf{Y}) = (X_k, Y_k) : k = 1, ..., n$ of n iid random vectors (X,Y) from the joint distribution of random vectors X in $\mathbb{R}^p$ and Y in $\mathbb{R}^q$, $(a_{kl}) = (|X_k - X_l|_p)$ and $(b_{kl}) = (|Y_k - Y_l|_q)$ are the Euclidean distance matrices, and the centered distances matrix is

$$A_{kl} = a_{kl} - \bar{a}_{k.} - \bar{a}_{.l} - \bar{a}_{..}\ k, l = 1, ..., n, \tag{2.12}$$

with

$$\bar{a}_{k.} = \frac{1}{n}\sum_{l=1}^{n} a_{kl},\ \bar{a}_{.l} = \frac{1}{n}\sum_{k=1}^{n} a_{kl},\ \bar{a}_{..} = \frac{1}{n^2}\sum_{k,l=1}^{n} a_{kl}, \tag{2.13}$$

In words, A_{kl} is the centered distance matrix of the X sample. B_{kl} is the same for the Y sample. $\bar{a}_{k.}$ is the row means, $\bar{a}_{.l}$ is the column means and $\bar{a}_{..}$ is the global means. It is important to point out that the rows and columns of A_{kl} and B_{kl} sum to zero.

Finally, the distance correlation is defined as

$$\mathcal{R}_n^2(X,Y) = \begin{cases} \frac{\mathcal{V}_n^2(X,Y)}{\sqrt{\mathcal{V}_n^2(X)\mathcal{V}_n^2(Y)}}, & \text{if } \mathcal{V}_n^2(X)\mathcal{V}_n^2(Y) > 0. \\ 0, & \text{if } \mathcal{V}_n^2(X)\mathcal{V}_n^2(Y) = 0. \end{cases} \tag{2.14}$$

If $E|X|_p < \infty$ and $E|Y|_q < \infty$, it is shown in [31] that almost surely

$$\lim_{n\to\infty} \mathcal{V}_n(\mathbf{X},\mathbf{Y}) = \mathcal{V}(X,Y)$$

$$\lim_{n\to\infty} \mathcal{R}_n(\mathbf{X},\mathbf{Y}) = \mathcal{R}(X,Y)$$

Some properties hold for $\mathcal{R}$, $\mathcal{R}_n$:

(i) If $E|X|_p < \infty$ and $E|Y|_q < \infty$, then $0 \leq \mathcal{R} \leq 1$, and $\mathcal{R}(X,Y) = 0$ if and only if X and Y are independent.

(ii) $0 \leq \mathcal{R}_n \leq 1$.

(iii) If $\mathcal{R}_n(\mathbf{X},\mathbf{Y}) = 1$, then there exist a vector a, a nonzero real number b and an orthogonal matrix C such that $\mathbf{Y} = a + b\mathbf{X}C$.

If the random vectors have finite first moments:

(i) $dVar(X) = 0$ implies that $X = E|X|$, almost surely.

(ii) $dVar(a + bCX) = |b|dVar(X)$ for all constant vectors a in $\mathbb{R}^p$, scalars b and $p \times p$ orthonormal matrices C.

(iii) $dVar(X+Y) \leq dVar(X)+dVar(Y)$ for independent random vectors $X \in IR^p$ and $Y \in IR^p$.

Moreover, [31] states that in case that X and Y are jointly distributed as bivariate normal, $\mathcal{R}$ is a function of $\rho = \rho(X,Y)$, the Pearson's correlation, and $\mathcal{R}(X,Y) \leq |\rho(X,Y)|$:

$$\mathcal{R}^2(X,Y) = \frac{\rho\ arcsin(\rho) + \sqrt{1-\rho^2} + \rho\ arcsin(\rho/2) - \sqrt{4-\rho^2} + 1}{1 + \pi/3 - \sqrt{3}} \tag{2.15}$$

. In Equation 2.15 $\mathcal{R}(X,Y) = \rho(X,Y)$ when $\rho = 0$ or $\rho = \pm 1$

2.3 The Splitting Rule and Stop Criterion

The first step of the algorithm is to represent the graphs in terms of their shell index distribution. The output is a matrix where each row represents a graph, the columns are shell indices and on cells are the shell index frequency count of graphs. After that, the algorithm enters in loop, being the first step of the cycle to make a global test of association between the graphical covariates and the response variable based on distance correlation,

$$H_0 = \bigcap_{j=1}^{m} H_0^j, \text{ such that}$$

$$H_0^j = D(Y|X_j) = D(Y).$$

The output of this step is sequence of p-value $P_1, ..., P_p$, where p is the number of predictors. The p-values are adjusted using Bonferroni correction (refer to [6], Equation 3.7). If one cannot reject the hypothesis, i.e., there is not any p-value lower than a predefined acceptance level α, the algorithm stops. Otherwise, the variable with lowest p-value is selected, suppose g_k^*. Once the covariate is selected, it is necessary to select the shell index in the shell distribution matrix that is most associated to the dependent variable using distance correlation. Again, here the result of this step will be a sequence of p-value, $P_1, ..., P_t$, where t is the maximum shell index found in the graphs of covariate k. These p-values are adjusted with Bonferrori correction. Then, it is necessary to select the shell index frequency count that best split the data into two subsets. A value is chosen to minimize the variance of the response variable. The method repeats iteratively on each subset until the stop condition is reached. A summary of the procedure is given bellow:

1. Global test if there is association between Y and $\{X_k\}_{k=1,...,p}$
2. Choose the network covariate g_k^* associated to the minimum adjusted p-value if it is lower than α
3. Choose the shell index of g_k^* associated to the minimum adjusted p-value if it is lower than α
4. Split data into two subsets, where the split point is the shell index frequency count that minimizes the variances of Y in the subsets
5. iterate on the two subsets

Chapter 3

Analysis

We tested the algorithm in 5 simulations, being 2 classification problems and 3 regressions. Moreover, one analysis was performed with real data for EEG (electroencephalogram) of patients [26]. The efficiency of our algorithm was compared to two well known tree-based methods: CART decision tree and conditional inference tree.

3.1 Simulation Study

In this section we present the application of Energy Tree in a simulated environment with univariate and multivariate scenarios, employing network and functional data. Both regression and classification tasks are performed in a train and test framework for a repeated simulated data. The efficacy of the Classification Energy Tree is measured with the accuracy metric, while root mean squared error is used to evaluate the Regression Energy Tree.

3.1.1 Classification Problem 1

In this simulation study, we classified two different types of graphs, bipartite and random graphs generated by Erdos-Renyi model. We executed 50 runs, with 100 graphs of each class. The training and test set had 80% and 20% of the full data set, respectively. The bipartite graphs have 200 nodes, 75 are bottom vertices and 125 top vertices with connection probability 0.07. The random graphs have 200 nodes and links were created with probability 0.03. All graphs are undirected.

A 10-rows sample of the shell index distribution matrix is present in Table 3.1 and we can see in a sample of the tree in Figure 3.1 how the energy tree built the structure and the tests done in order to classify the data points. The results are summarized in Table 3.2.

3.1.2 Classification Problem 2

In this classification set-up, we generated 400 ER-random graphs, but varying the parameter p. Of the total, edges in 134 graphs were created with probability 0.01, in 133 with probability 0.015 and in the rest with 0.02. We simulated 50 times. The training and test set had 80% and 20%, respectively, as before. All graphs have 200

Table 3.1. 10 rows of the shell index distribution matrix of the classification problem with ER-random and bipartite graphs

Shell index / Graph ID	0	1	2	3	4	5	Y
1	0	2	4	18	50	126	0
2	1	0	8	20	44	127	0
3	1	5	8	26	160	0	1
4	0	5	11	13	171	0	0
5	0	6	7	18	169	0	0
6	0	2	8	17	35	138	0
7	0	3	6	17	41	133	0
8	0	3	15	17	165	0	0
9	1	3	10	36	150	0	1
10	1	2	6	20	171	0	1

Table 3.2. Accuracy mean and standard deviation for 100 bipartite graphs and 100 ER-random graphs for different classification methods

Method	Mean Acc.	SD Acc.
Classification Energy Tree	0.7400	0.0751
CART	0.7390	0.0836
Conditional Inference Tree	0.7295	0.0735

nodes and are undirected. The response variable has three classes, one for each p value assigned during the graph creation.

Check in Table 3.3 10 rows of the shell index distribution matrix of one execution of this simulation. The results are present in Table 3.4. Our model performed slightly worse than the others. The confusion matrix of the second run of this problem is represented in Table 3.5. It can be seen that the algorithm was unsuccessful to correctly classify Class 2, resulting in an accuracy of 0.65 in this run. We can check in Figure 3.2 the tree structure of our method. It favoured the split using shell index 1, 2 and 3, while the conditional inference tree almost captured perfectly the hidden pattern of data using 0, 1 and 3 (Figure 3.3).

3.1.3 Regression Problem 1

We ran 50 times a regression problem with 100 Erdos-Renyi random graphs and 100 bipartite graphs. All graphs have 200 nodes, but in the bipartite graphs 75 of them are bottom vertices and 125 top vertices with connection probability 0.07. ER-random graphs have edges that were created with probability 0.03. The dependent variable is defined as:

$$Y \sim \begin{cases} \mathcal{N}(10, 2), & \text{if g is a ER-random graph.} \\ \mathcal{N}(30, 3), & \text{if g is a bipartite graph.} \end{cases}$$

As standard in this work, the Regression Energy Tree was compared to CART

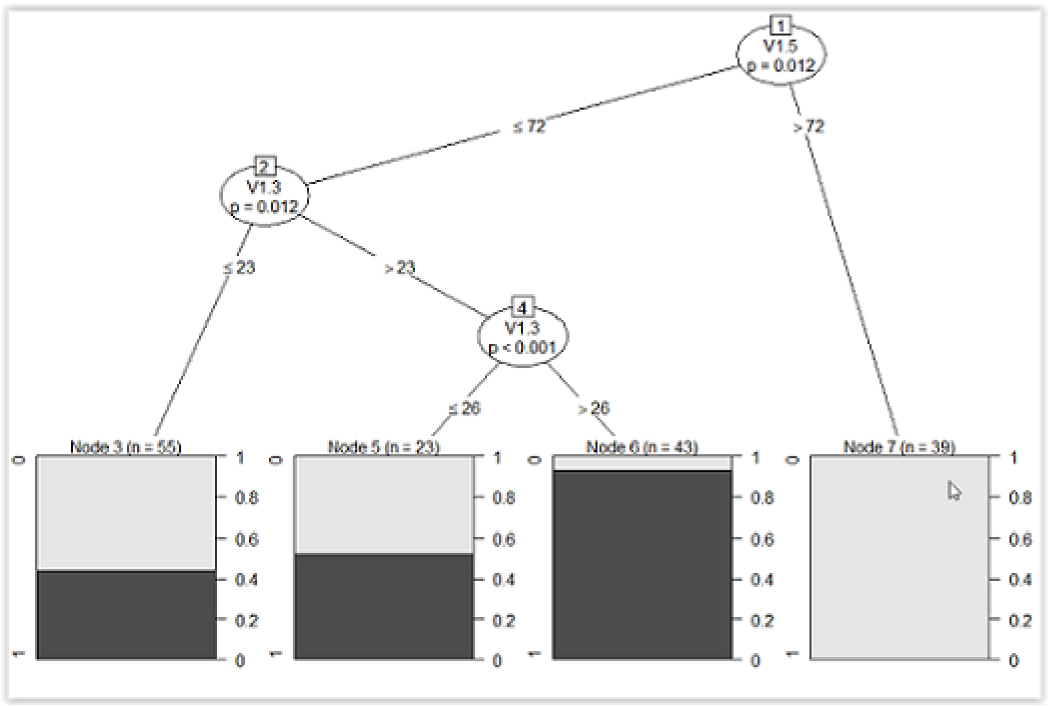


Figure 3.1. The energy tree structure sample for the classification problem of bipartite and ER-random graphs

decision tree and conditional inference tree. The metric used was the mean of the root mean square error over the 50 runs. It was also computed the standard deviation of the root mean squared error. The result of this simulation is present in Table 3.6. In Table 3.7 is listed a sample of shell index distribution of this simulation and Figure 3.4 depicts the tree structure of one execution of this simulation.

3.1.4 Regression Problem 2

In this simulation, we created 200 ER-random graphs. The edges in the graphs were created with probability that varied between 0.01, 0.07 and 0.15. Graphs created with $p = 0.01$ have Y $\sim \mathcal{N}(10, 2)$, whilst graphs with $p = 0.07$ and $p = 0.15$ have Y $\sim \mathcal{N}(30, 3)$ and Y $\sim \mathcal{N}(50, 2.5)$, respectively. Like previously, we simulated 50 times the same configuration. CART decision tree performed slightly better than Regression Energy Tree, which outweighed conditional inference tree. The results are shown in Table 3.8.

3.1.5 Regression Problem 3

We simulated the Regression Energy Tree in a multivariate scenario. The data set is composed by one network covariate and one functional. We generated 200 observations with the following configuration: 200 graph data points are ER-random undirected graphs with $p = 0.03$, while the functional covariate is divided in three classes of functions, being 66 data points generated by $cos(a+b\pi t)$, 67 by $sin(a+b\pi t)$ and 67 by $cos(a + b\pi t)sin(a + b\pi t)$, a and b is sampled from discrete uniform in

Table 3.3. 10 rows of the shell index distribution matrix of the classification problem with ER-random graphs generated with $p \in (0.01, 0.015, 0.02)$

Graph ID \ Shell index	0	1	2	3	Y
1	18	80	102	0	1
2	6	10	27	157	3
3	6	46	148	0	2
4	2	16	50	132	3
5	4	11	81	104	3
6	37	104	59	0	1
7	7	42	151	0	2
8	1	24	36	139	3
9	11	47	142	0	2
10	5	41	154	0	2

Table 3.4. Accuracy mean and standard deviation of the tree-based methods with three types of ER-random graphs ($p = 0.01, p = 0.015, p = 0.02$)

Method	Mean Acc.	SD Acc.
Classification Energy Tree	0.9155	0.1142
CART	0.9805	0.0145
Conditional Inference Tree	0.9795	0.0153

$[1, 4]$, $t = 1, ..., 100$. Refer to Section 3.4.2 in [6] for more on how the functional data is generated.

The response variable is calculated by summing two factors, Y_g coming from the graph data and Y_f from the functional data. Y_g is a linear combination of the shell indices in the shell index distribution matrix and is defined as:

$$Y_g(n_1, ..., n_{k_{max}}) = \sum_{i=1}^{k_{max}} i * n_i(g),$$

where $n_i(g)$ is the number of vertices of the graph g with shell index equal to i. The factor Y_f is defined as realizations from a normal distribution which the parameters depend on the class of the function in the form of $\mathcal{N}(10, 2)$, $\mathcal{N}(30, 3)$ and $\mathcal{N}(50, 2.5)$ for the same order of functions described in the above paragraph. The data is created following the rules summarized in Table 3.9.

In Figure 3.5, it can be seen that the Energy Tree performed three splits using both variables: firstly using the network variable $V1$ and shell index 3 and the last two splits by the functional variable $V2$. Our Regression Energy Tree was compared to the conditional inference tree and CART. The results are summarized in Table 3.10.

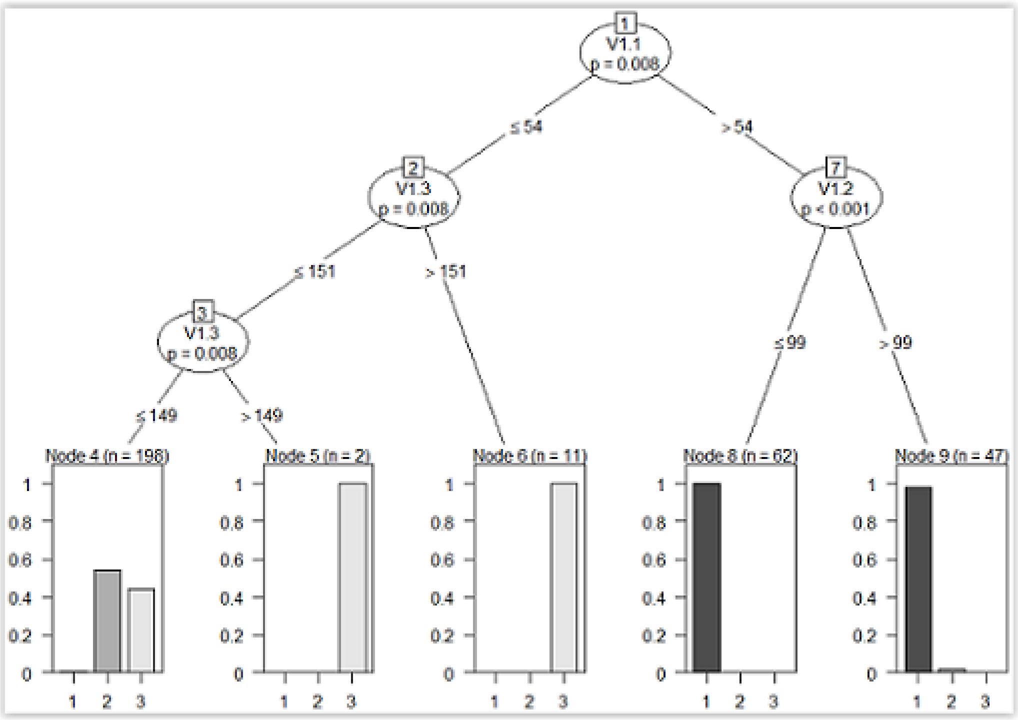


Figure 3.2. The energy tree structure sample for the classification problem with 400 ER-random graphs created with p varying between 0.01, 0.015 and 0.02

Table 3.5. Confusion matrix for predicted values of dependent variable versus the true values for the classification problem with 400 ER-random graphs created with p varying between 0.01, 0.015 and 0.02

Reference / **Prediction**	**1**	**2**	**3**
1	24	0	0
2	0	24	0
3	0	28	4

3.2 A Real Case Study

We analysed the performance of the Regression Energy Tree in a real case scenario. Our model was applied to study the association between the networks of roads of all 50 states and the District of Columbia in United States and the total fatal vehicle accidents in these territories.

The roads networks were downloaded from [8] and compose the independent variable of our data. The nodes in the graphs mark the end location of each complete line. Every line has two nodes, a start node and an end node. The node ID is assigned based on its position in relation of the line. To a node in the left is assigned an ID equal to i, while the ID of the node in the right is set as $i + 1$. The format of the data present in [8] is white-space separated list of numbers in the form of:

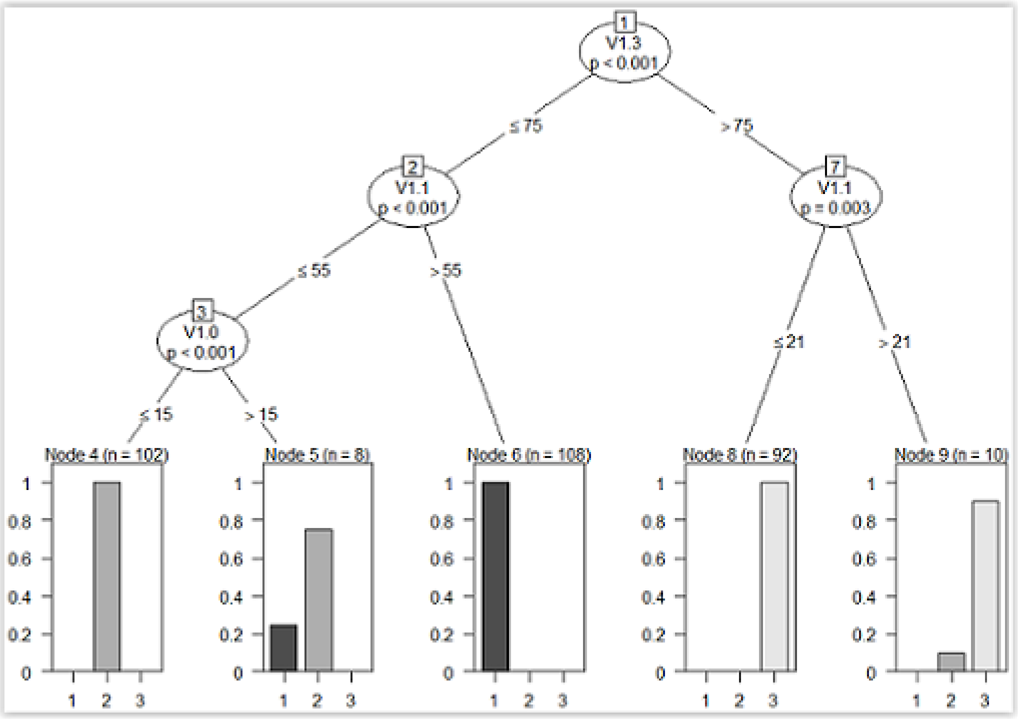


Figure 3.3. The conditional inference tree structure sample for the classification problem with 400 ER-random graphs created with p varying between 0.01, 0.015 and 0.02

Table 3.6. MRMSE of 50 simulations and standard deviation of the tree-based methods with 100 ER-random graphs ($p = 0.03$) and 100 bipartite ($p = 0.07$)

Method	MRMSE	SD RMSE
Regression Energy Tree	8.5071	0.7209
CART	8.7025	0.8201
Conditional Inference Tree	10.4867	0.3377

Number of nodes
For each node:
- id
- longitude
- latitude

Number of edges
For each edge:
- id of the source node
- id of the end node
- travel time
- spatial distance in meters
- road category.

The information of nodes were not important for this study, thus the first <*Number of nodes + 1*> rows were dropped in each file. Finally, *travel time*, *spatial distance*, *road category* were also removed.

Table 3.7. 10 rows of the shell index distribution matrix of the regression problem with 100 ER-random graphs and 100 bipartite graphs

Graph ID \ Shell index	0	1	2	3	4	5	Y
1	0	2	10	31	157	0	9.4888
2	0	4	16	26	154	0	36.5144
3	0	3	7	18	42	130	9.1542
4	1	3	13	21	162	0	28.0956
5	2	2	12	37	147	0	25.9537
6	2	5	7	20	166	0	9.8385
7	0	3	8	24	165	0	26.0276
8	0	5	17	36	142	0	30.1641
9	1	3	8	26	162	0	25.8522
10	0	6	8	24	44	118	10.3239

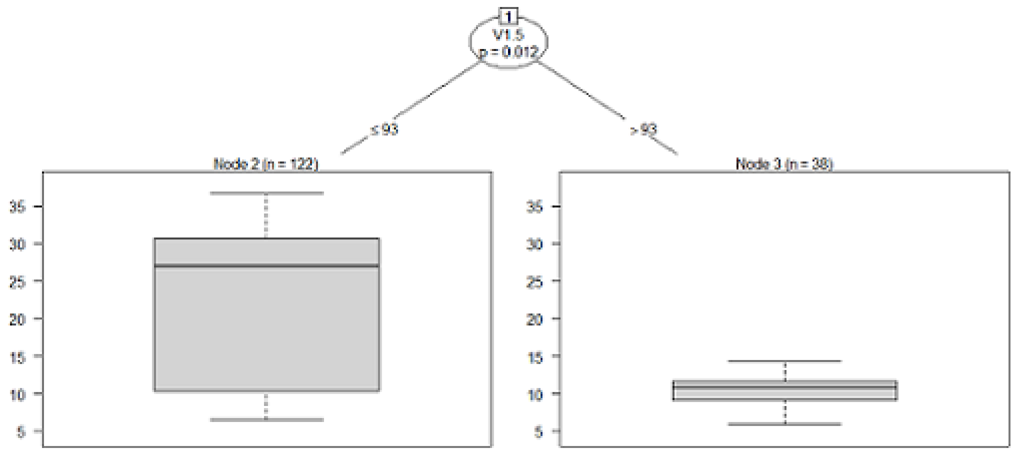


Figure 3.4. Sample of the Energy Tree structure for the regression problem with 100 ER-random graphs and 100 bipartite graphs

The official statistics of the total fatal vehicle accidents per state in United States is available in [1] and represents the response variable of our data set. The statistics were registered in 2017, while the roads networks refer to the year of 2000. Since the topology of cities and states do not change much over time, the difference of 17 years between the creation of the two data does not undermined the analysis. The entire data set has 2.26GB. The summary of it is present in Table 3.11.

We computed the RMSE for many values of *alpha*, from 0.01 to 0.4 in steps of 0.01. The best acceptance levels on the energy test are from 0.22 to 0.32. The study of the best *alpha* is present on Figure 3.6. The associated tree for $\alpha = 0.22$ is depicted on Figure 3.7. We compared the performance of our model in this scenario with *ctree* and CART and the results is summarized in Table 3.12.

Table 3.11. RMSE of the fatal vehicle crashes regression problem

State	Fatal Vehicle Crashes	Network Nodes	Network Edges
Alabama	864	566843	661487
Alaska	75	69082	78100
Arizona	919	545111	665827
Arkansas	457	483175	563036
California	3304	1613325	1989149
Colorado	600	448253	539295
Connecticut	260	153011	187318
Delaware	112	49109	60512
District of Columbia	29	9559	14909
Florida	2922	1048506	1330551
Georgia	1440	738879	869890
Hawaii	96	64892	76809
Idaho	223	271450	318761
Illinois	1005	793336	1012817
Indiana	836	497458	629750
Iowa	301	390002	502269
Kansas	407	474015	607391
Kentucky	721	467967	525995
Louisiana	696	413574	499254
Maine	163	194505	214921
Maryland	511	265912	317624
Massachusetts	336	308401	385164
Michigan	939	673534	845087
Minnesota	340	547028	670443
Mississippi	614	413250	483306
Missouri	863	675407	807892
Montana	169	317905	360936
Nebraska	210	308157	392008
Nevada	290	261155	311043
New Hampshire	98	116920	133415
New Jersey	591	330386	436036
New Mexico	340	467529	567084
New York	933	716215	897451
North Carolina	1306	887630	1009846
North Dakota	105	210801	260902
Ohio	1094	676058	842872
Oklahoma	611	540981	664215
Oregon	400	536236	628167
Pennsylvania	1083	874843	1088296
Rhode Island	76	53658	69213
South Carolina	924	463652	553599

South Dakota	111	212313	259622
Tennessee	959	583484	676080
Texas	3343	2073870	2584159
Utah	247	248730	295763
Vermont	63	97975	107558
Virginia	783	630639	714809
Washington	536	575860	675049
West Virginia	280	300146	328858
Wisconsin	557	519157	635436
Wyoming	105	253077	304014

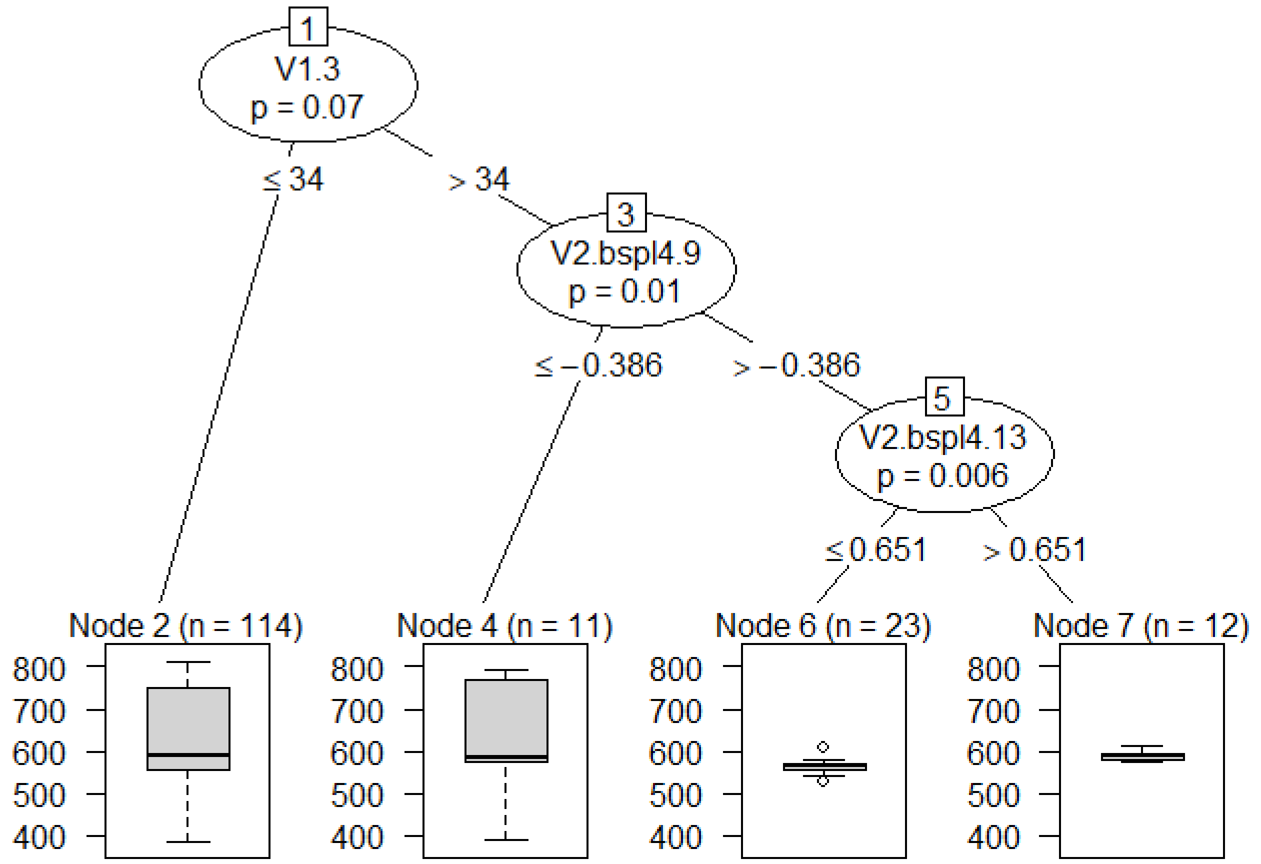


Figure 3.5. Structure of the energy regression tree of a sample of the multivariate problem, coupling network and functional data

Table 3.8. MRMSE of 50 simulations and standard deviation of the tree-based methods with 200 ER-random graphs with varying between 0.01, 0.07 and 0.15, and Y is normally distributed

Method	MRMSE	SD RMSE
Regression Energy Tree	5.1322	2.605
CART	2.7376	0.5825
Conditional Inference Tree	16.6943	0.4233

Table 3.9. Summary of the rules to generate the data set in the multivariate simulation

Network data	Functional data	Response
ER-random graph g_1	$cos(a + b\pi t)$	$\sum_{i=1}^{k_{max}} i * n_i(g_1) + \mathcal{N}(10, 2)$
ER-random graph g_2	$sin(a + b\pi t)$	$\sum_{i=1}^{k_{max}} i * n_i(g_2) + \mathcal{N}(30, 3)$
ER-random graph g_3	$cos(a + b\pi t)sin(a + b\pi t)$	$\sum_{i=1}^{k_{max}} i * n_i(g_3) + \mathcal{N}(50, 2.5)$

Table 3.10. MRMSE of 50 simulations of the regression problem with both functional and network covariate

Method	MRMSE	SD RMSE
Regression Energy Tree	108.2556	8.935
CART	126.3333	12.4294
Conditional Inference Tree	108.4542	9.2793

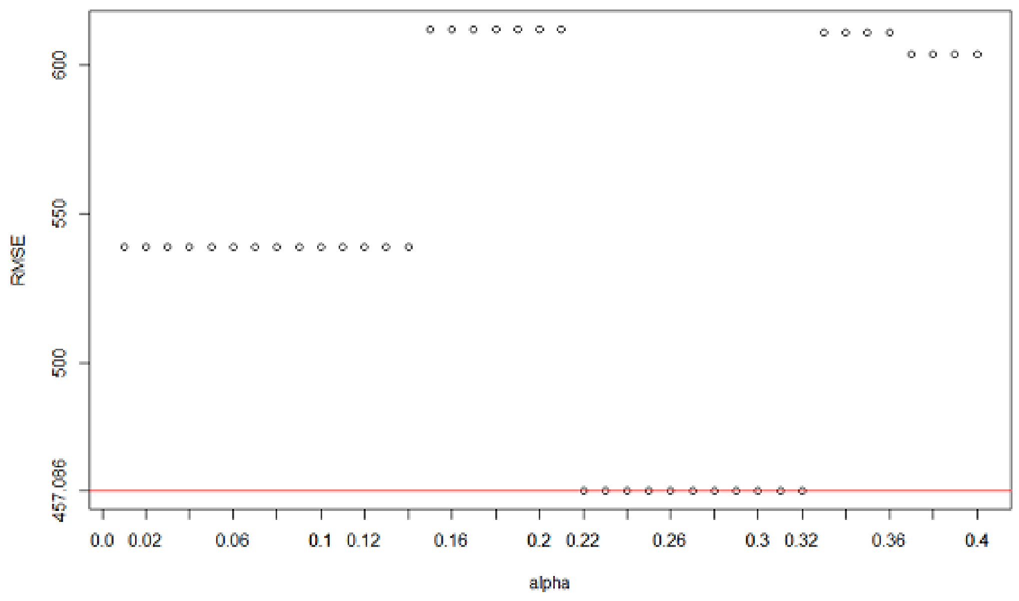


Figure 3.6. RMSE for many values of alpha

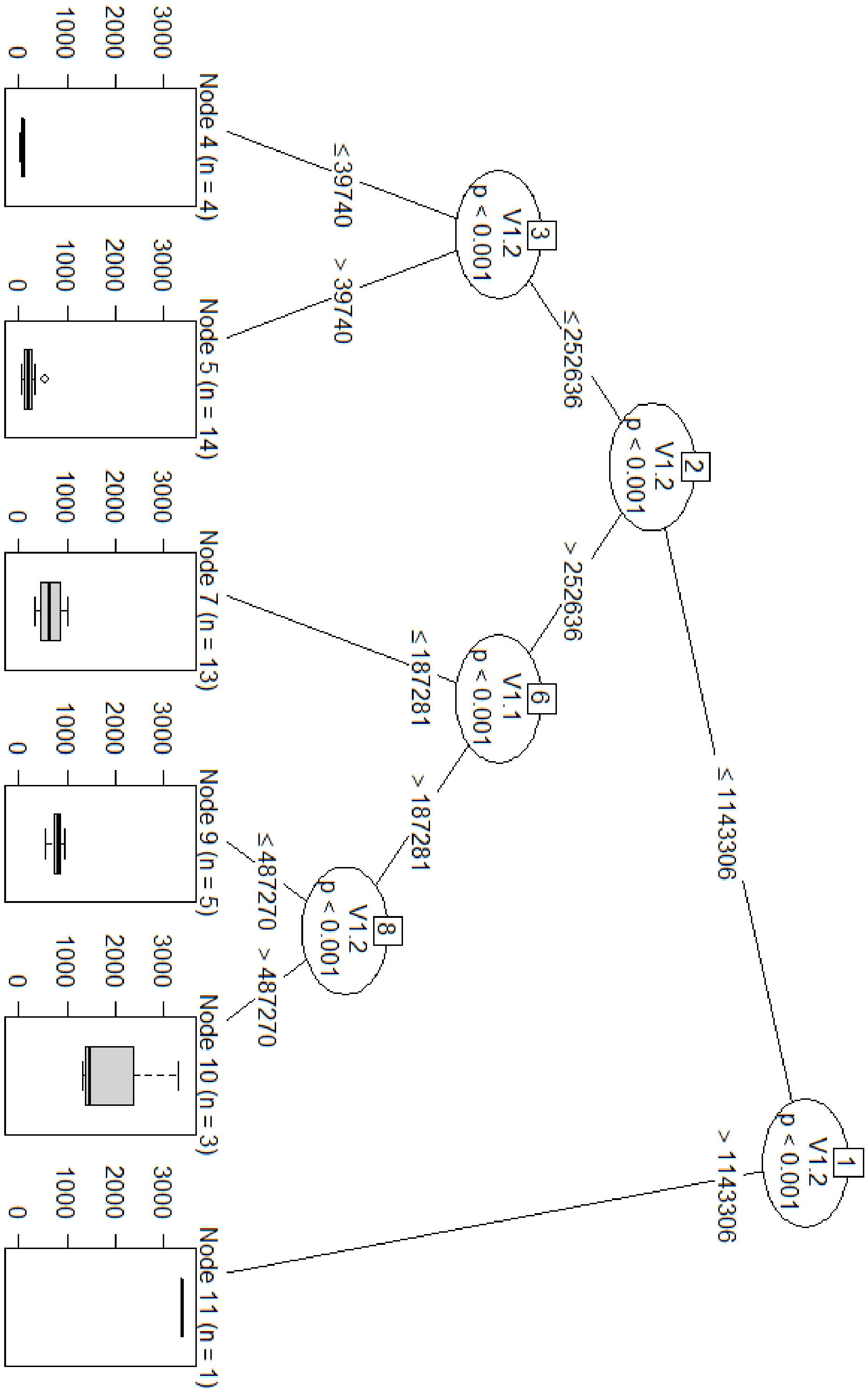


Figure 3.7. Structure of the energy regression tree applied to the fatal vehicle crashes problem in USA

Table 3.12. RMSE of the fatal vehicle crashes regression problem

Method	**RMSE**
Regression Energy Tree	457.086
CART	476.83
Conditional Inference Tree	567.0271

Conclusions and Final Considerations

In this present work we improved the Energy Tree for functional covariates developed in [6], enabling the use of network covariates. We enhanced the method to make classification and regression applying a tree-based model that works with complex data, such as graphs and functions. The algorithm tests the association between network covariates and the dependent variable using an energy statistic - the distance correlation. It works with continuous or categorical response variable.

We tested the efficiency of our method in six cases, including one with real data. Our model performed better than two famous tree-based methods (CART and conditional inference tree) in 4 scenarios. CART outperformed in one regression set-up, but Energy Tree had lower error than *ctree*. In one classification simulation, our model scored 91.55% of accuracy, but was behind CART and *ctree*.

Improvements can still be done on the algorithm. We do not consider weights on nodes or edges. It would be interesting to evaluate the biding force between nodes or even attributes of nodes and how these values influence the construction of the shell index distribution matrix. There is a lack of real data set with network as covariates as well. A future work could study the brain (and its associated graph) coupled with a functional covariate to make predictions. Further developments can be the use of directed graphs and another matrix representation of graphs that capture their structure as alternative to shell index distribution.

Appendices

Appendix A

Implementation

```
#### ver. "2019-03-17"

variable.importance <- function(response_name = "Y",
                                data,
                                deep = TRUE,
                                nb = 15,
                                R = 1000) {
  response <- data[[response_name]]
  datanew = data
  # extract the features from each variable of the list,
  # otherwise just check importance of the variables
  # of the list
  if(deep) {
    if(class(data) == "list") {
      datanew <- list2matrix(
        lst = data[which(names(data) != response_name)],
        struct2copy = NULL,
        Y = NULL,
        nb = nb)
    }
  }
  else {
    n.var <- which(names(data) != response_name)
    if (class(data) == "list") {
      datanew <- list("response" = response)
      for (j in n.var) {
        if (class(data[[j]]) == "fdata") {
          foo <- min.basis(data[[j]], numbasis = nb)
          fd3 <-
            fdata2fd(foo$fdata.est,
                     type.basis = "bspline",
                     nbasis = foo$numbasis.opt)
          foo$coef <- t(fd3$coefs)
          datanew[[j]] <- foo
        }
        else if (class(data[[j]]) == "list" &
                 class(data[[j]][[1]]) == "igraph") {
```

```
          datanew[[j]] <- graph.to.shellness.distr.df(data[[j]])
        }
        else if (class(data[[j]]) == "data.frame") {
          datanew[[j]] = data[[j]]
        }
      }
      names(datanew)[-1] <- names(data)[-1]
    }
    else if (class(data) == "data.frame") {
      colnames(datanew)[
        colnames(datanew) == response_name] <- "response"
    }

  }
  data <- NULL

  n.var <- which(names(datanew) != "response")
  p = sapply(n.var, function(i) {
    if (class(datanew[[i]]) == "list") {
      mytestREG(x = datanew[[i]]$fdata.est,
                y = response,
                R = R)
    }

    else if (class(datanew[[i]]) == "data.frame") {
      mytestREG(x = datanew[[i]], y = response, R = R)
    }

    else if (class(datanew[[i]]) == "numeric"
    | class(datanew[[i]]) == "integer") {
      mytestREG(x = datanew[[i]], y = response, R = R)
    }
  })
  feature <- names(datanew)[which(names(datanew) != "response")]
  p_value <- p[2,]

  # bonferroni correction
  p_value <- 1 - (1 - p_value) ^ sum(!is.na(p_value))
  dist.corr <- p[1,]
  result <- data.frame(feature, p_value, dist.corr)
  return(result[order(result$p_value, -rank(result$dist.corr)), ])
}

#Testing equal distributions
mytestREG <- function(x, y, R = 1000) {
  library(cluster)
  library(fda.usc)

  if (is.factor(x)) {
    d1 = daisy(as.data.frame(x))
  }
  if (is.numeric(x)) {
    d1 = metric.dist(as.data.frame(x))
  }
  if (is.data.frame(x)) {
    d1 = metric.dist(x)
  }
  if (is.fdata(x)) {
    d1 = metric.lp(x)
  }
  y = data.frame(y)
  d2 <- metric.dist(y)
  ct <- energy::dcor.test(d1, d2, R = R)
  if (!is.na(ct$statistic)) {
    return(c(ct$statistic, ct$p.value))
  } else{
    c(NA, NA)
  }
}

##test for split nominal variables
##perform chi-squared test of yvs.x
mychisqtest <- 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)
}

# See sample()'s surprise -- example in help file
resample <- function(x, ...)
  x[sample.int(length(x), ...)]

graph.to.shellness.distr.df <- function(data, shell.limit = NULL) {
  tot.graphs = length(data)

  list.df <- list()
  max.shellness = 0

  for (i in 1:tot.graphs) {
    g = data[[i]]
    coreness.distr = count(coreness(g)) # aggr. by count
    rownames(coreness.distr) <-
      coreness.distr$x # re-index the df by the shellness number

    # keep just the frequency column
    coreness.distr = coreness.distr[c('freq')]

    # transpose the df. Convert the column-df into row-df.
    #This will ease the join with df.shellness.distr
    coreness.distr = t(coreness.distr)
    list.df[[i]] <- coreness.distr
    this.max.shellness = colnames(coreness.distr)[
      ncol(coreness.distr)]

    # update the maximum shellness found so far (used to build
    #the df of shellness distr)
    if (this.max.shellness > max.shellness) {
      max.shellness = this.max.shellness
    }
  }

  if (!is.null(shell.limit)) {
    # calculates the max shellness between the number of
    # predictors used in train set and the one calculated in
    # test set
    max.shellness <-
      if (as.numeric(max.shellness) < shell.limit - 1)
        shell.limit - 1
    else
      max.shellness
  }

  col.names = seq(0, max.shellness, 1)
  col.names = lapply(col.names, function(x)
    as.character(x))  # convert to char
  df.shellness.distr = data.frame(matrix(
    data = NA_integer_,
    nrow = tot.graphs,
    ncol = length(col.names)
  )) #df with all graphs shellness distribution
  colnames(df.shellness.distr) <- col.names

  # fill in the df with the shellness distribution of each graph
  for (i in 1:tot.graphs) {
    updated.cols = colnames(list.df[[i]])

    for (x in updated.cols) {
      df.shellness.distr[i, x] = list.df[[i]][, x]
    }
  }

  df.shellness.distr[is.na(df.shellness.distr)] <-
    0 # replace NA by 0

  # converted the df columns to integer
  df.shellness.distr[, seq(1, ncol(df.shellness.distr))] <-
    sapply(df.shellness.distr[, seq(1, ncol(df.shellness.distr))],
           as.integer)

  return(df.shellness.distr)
}

mytree <- function(Y,
                   data,
                   weights = NULL,
                   minbucket = 1,
                   alpha = 0.05,
                   R = 1000,
                   rnd.sel = T,
                   rnd.splt = TRUE,
                   nb = 5) {
  # name of the response variable
  response <- data[[which(names(data) == Y)]]

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

  n.var <- which(names(data) != Y)
  if (class(data) == "list") {
    datanew <- list("response" = response)
    for (j in n.var) {
      if (class(data[[j]]) == "fdata") {
        foo <- min.basis(data[[j]], numbasis = nb)
        fd3 <-
          fdata2fd(foo$fdata.est,
                   type.basis = "bspline",
                   nbasis = foo$numbasis.opt)
        foo$coef <- t(fd3$coefs)
        datanew[[j]] <- foo
      }
      else if (class(data[[j]]) == "list" &
               class(data[[j]][[1]]) == "igraph") {
        datanew[[j]] <- graph.to.shellness.distr.df(data[[j]])
      }
      else if (class(data[[j]]) == "data.frame") {
        datanew[[j]] = data[[j]]
      }
    }
    names(datanew)[-1] <- names(data)[-1]
  }
  else if (class(data) == "data.frame") {
    datanew = data
    colnames(datanew)[colnames(datanew) == "Y"] <- "response"
  }

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

  # compute terminal node number for each observation
  response <- response
  response <- data.frame(response)
  y = response
  m.data <- c()

  for (j in n.var) {
    if (class(data[[j]]) == "fdata") {
      foo <- datanew[[j]]$coef
      colnames(foo) <-
        paste(names(data)[j], colnames(datanew[[j]]$coef),
              sep = ".")
    }
    else if (class(data[[j]]) == "data.frame" |
            (class(data[[j]]) == "list" &
             class(data[[j]][[1]]) == "igraph")) {
      foo <- datanew[[j]]
      colnames(foo) <-
        paste(names(data)[j], colnames(datanew[[j]]), sep = ".")
    }

    # fill in m.data or initialize it
    if (!is.null(m.data)) {
      m.data <- cbind(m.data, foo)
    }
    else {
      m.data <- foo
    }
  }

  data1 = cbind(response, m.data)
  m.data = m.data
  data1 = data1

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

  # return rich constparty object
  ret <- party(
    nodes,
    data = data.frame(m.data),
    fitted = data.frame(
      "(fitted)" = fitted,
      "(response)" = data1$response,
      "(weights)" = weights,
      check.names = FALSE
    ),
    terms = terms(formula, data = data1)
  )
```

```
  as.constparty(ret)
}

####

growtree <- function(id = 1L,
                     response,
                     data,
                     weights,
                     minbucket,
                     alpha,
                     R,
                     rnd.sel,
                     rnd.splt,
                     n.var) {
  # for less than <minbucket> observations stop here (for ctree()
  # is 7 in ?ctree_control)
  if (sum(weights) <= minbucket) {
    return(partynode(id = id))
  }

  # find best split
  res <- findsplit(
    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

  # no split found, stop here
  if (is.null(sp)) {
    return(partynode(id = id))
  }

  kidids <- c()

  if (class(data[[varselect]]) == "list") {
    kidids[which(data[[
      varselect]]$coef[, sp$varid] <= sp$breaks)] <- 1
    kidids[which(data[[
      varselect]]$coef[, sp$varid] > sp$breaks)] <- 2

    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))

    nb = 0
    for (i in n.var) {
      k = data[[i]]$numbasis.opt
      nb = c(nb, k)
    }

    # shift the varid of the tree based on the quantity of the
    # previous features/basis
    # Ex: if variable 3 is selected for splitting (variable 1
    # is the response, it's ignored), then shift varid by the
    # number of basis of variable 2 (if it's functional) or the
    # maximum k_core found in the graphs (if it's a graph)
    if (varselect != min(n.var)) {
      total_features <- c()
      lapply(n.var, function(v) {
        if (class(data[[v]]) == 'list')
          total_features[[v]] <<- data[[v]]$numbasis.opt
        if (class(data[[v]]) == 'data.frame')
          total_features[[v]] <<- ncol(data[[v]])
        if (class(data[[v]]) == 'numeric')
          total_features[[v]] <<- 1
      })
      step <-
        sum(total_features[n.var[
          which(n.var < varselect)]], na.rm = T)
      sp$varid = sp$varid + as.integer(step)
    }
  }
  else if (class(data[[varselect]]) == "data.frame") {
    kidids[(which(data[[
      varselect]][, sp$varid] <= sp$breaks))] <- 1
    kidids[(which(data[[
      varselect]][, sp$varid] > sp$breaks))] <- 2

    sum1 <-
      length(which(data[[
        varselect]][, sp$varid][which(weights == 1)] <=
        sp$breaks))
    sum2 <-
      length(which(data[[
        varselect]][, sp$varid][which(weights == 1)] >
        sp$breaks))
  }
  else if (class(data[[varselect]]) == "numeric") {
    kidids[(which(data[[varselect]] <= sp$breaks))] <- 1
    kidids[(which(data[[varselect]] > sp$breaks))] <- 2

    sum1 <-
      length(which(data[[
        varselect]][which(weights == 1)] <= sp$breaks))
    sum2 <-
```

```
      length(which(data[[
        varselect]][which(weights == 1)] > sp$breaks))
  }

  if (all(kidids == 1) | all(kidids == 2))
    return(partynode(id = id))

  if ((sum1 == 0 | sum2 == 0)) {
    return(partynode(id = id))
  }

  # setup all daugther nodes
  kids <- vector(mode = "list", length = max(kidids,
                                             na.rm = TRUE))

  for (kidid in 1:length(kids)) {
    # select observations for current node
    w <- weights
    w[kidids != kidid] <- 0

    # get next node id
    if (kidid > 1) {
      myid <- max(nodeids(kids[[kidid - 1]]))
    } else{
      myid <- id
    }

    # Start recursion on this child node
    kids[[kidid]] <-
      growtree(
        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))
  ))
}
```

```
findsplit <- function(response,
                      data,
                      weights,
                      alpha,
                      R,
                      rnd.sel,
                      rnd.splt,
                      n.var) {
  require(partykit)

  # 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) {
    if (class(data[[i]]) == "list") {
      mytestREG(x = data[[i]]$fdata.est[which(weights == 1)],
                y = y,
                R = R)
    }

    else if (class(data[[i]]) == "data.frame") {
      mytestREG(x = data[[i]][which(weights == 1),], y = y, R = R)
    }

    else if (class(data[[i]]) == "numeric") {
      mytestREG(x = data[[i]][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),]
  }
```

```
# split into two groups minimizing entropy
if (is.factor(x)) {
  # setup all possible splits in two kid nodes
  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)
      mychisqtest(x %in% q, y))
    splitpoint <- comb[[which.min(xlogp)]]
  }

  # split into two groups (setting groups that do not occur
  # to NA)
  splitindex <- !(levels(data[[xselect]]) %in% splitpoint)
  splitindex[!(levels(data[[xselect]]) %in% lev)] <- NA_integer_
  splitindex <- splitindex - min(splitindex, na.rm = TRUE) + 1L
}
if (is.data.frame(data[[xselect]])) {
  # select the column
  all.columns = seq(1, ncol(data[[xselect]]))
  # iterate over each column of the dataframe
  p = sapply(all.columns, function(i) {
    mytestREG(x = data[[
      xselect]][[i]][which(weights == 1)], y = y, R = R)
  })

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

  if (minp > alpha)
    return(NULL)

  # select the column to split
  cselect <- NA
  if (length(which(p[2,] == min(p[2,], na.rm = T))) > 1) {
    cselect <- which.max(p[1,])
  } else{
    cselect <- which.min(p[2,])
  }

  splitindex <-
    s.opt(response, x[which(weights == 1), cselect], rnd.splt)
}
if (is.numeric(x)) {
  splitindex <- s.opt(response, x[which(weights == 1)], rnd.splt)
}
if (is.list(x)) {
```

```r
    if (is.fdata(x$fdata.est)) {
      bselect <- 1:dim(x1)[2]
      p1 <- c()
      p1        <-
        sapply(bselect, function(i)
          mytestREG(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)
    }
  }

  # return split as partysplit object
  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.data.frame(x)) {
    temp = list (
      sp = partysplit(
        varid = as.integer(cselect),
        breaks = splitindex,
        info = list(p.value = 1 - (1 - p) ^ sum(!is.na(p)))
      ),
      varselect = xselect
    )
    return(temp)
  }
  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) {
  # find the split minimizing variance
  s  <- sort(X)
  obj <- c()
  for (i in 1:length(s)) {
    data1  <- y[which(X < s[i])]
    data2  <- y[which(X >= s[i])]
    v1 = var(data1)
    v2 = var(data2)
    n1 = length(data1)
    n2 = length(data2)
    n = n1 + n2

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

  if (all(is.na(obj)))
  {
    splitindex <- length(obj)
  }
  else {
    splitindex <- s[which.min(obj)]
  }

  return(splitindex)
}

# convert a list of graphs and/or functions to matrix
list2matrix <- function(lst,
                        struct2copy = NULL,
                        Y = NULL,
                        nb = NULL) {
  m.data <- c()
  if (!is.null(Y)) {
    n.var <- which(names(lst) != names(Y))
  }
  else {
    n.var <- seq(1, length(lst))
  }

  for (j in n.var) {
    if (class(lst[[j]]) == "fdata") {
      foo <- min.basis(lst[[j]], numbasis = nb)
      fd3 <-
        fdata2fd(foo$fdata.est,
                 type.basis = "bspline",
                 nbasis = foo$numbasis.opt)
      foo$coef <- t(fd3$coefs)
      foo <- foo$coef
      colnames(foo) <-
        paste(names(lst)[j], colnames(foo), sep = ".")
    }
    else if (class(lst[[j]]) == "data.frame" |
             (class(lst[[j]]) == "list" &
              class(lst[[j]][[1]]) == "igraph")) {
      if (!is.null(struct2copy)) {
        graph.data.train <-
          struct2copy[, grep(paste("V", j, sep = ""),
                             names(struct2copy))]
        foo <-
          graph.to.shellness.distr.df(lst[[j]],
                                      ncol(graph.data.train))
        foo.names <- names(graph.data.train)
      }
      else {
        foo <- graph.to.shellness.distr.df(lst[[j]], NULL)
        foo.names <- names(foo)
        # concatenates the variable prefix
        foo.names <- paste(names(lst)[j],
                           foo.names, sep = ".")
      }
      colnames(foo) <- foo.names
    }

    # fill in m.data or initialize it
    if (!is.null(m.data)) {
      m.data <- cbind(m.data, foo)
    }
    else {
      m.data <- foo
    }
  }
  if (!is.null(Y)) {
    m.data <- cbind(m.data, Y)
  }
  return(m.data)
}

my.predict <- function(Y = "Y",
                       model,
                       newdata,
                       nb = NULL) {
  m.data <- c()
  if (class(newdata) == "list") {
    m.data <- list2matrix(newdata,
                          struct2copy = model$data,
                          Y = NULL,
                          nb)
  }
  else if (class(newdata) == "data.frame") {
    m.data = newdata
  }

  response.pred <-
    partykit::predict.party(model, newdata = m.data)
  return(response.pred)
}
```

Bibliography

[1] Fatality facts. `https://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/state-by-state-overview`. Accessed: 2019-03-19.

[2] Altaf Amin, Kensaku Nishikata, T Koma, T Miyasato, Y Shinbo, M Arifuzzaman, C Wada, M Maeda, T Oshima, H Mori, and Shigehiko Kanaya. Prediction of protein functions based on k-cores of protein-protein interaction networks and amino acid sequences. *Genome Informatics*, 14:498–499, 01 2003.

[3] Thomas Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 1763.

[4] Allan Bickle. he k-cores of a graph. *Dissertations*, 505, 2010.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, page 663. Springer-Verlag, Berlin, Heidelberg, 2006.

[6] Marco Brandi. *Classication and Regression Energy Tree for Functional Data*. PhD thesis, Sapienza Università di Roma, 2018.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[8] U.S. Census Bureau. Ua census 2000 tiger/line files. http://users.diag.uniroma1.it/challenge9/data/tiger/, 2002.

[9] B. Choudhary. *The Elements of Complex Analysis*, page 20. J. Wiley, 1993.

[10] Giatsidis Christos. *Graph Mining and Community Evaluation with Degeneracy*. PhD thesis, École Polytechnique, 2013.

[11] Vapnik V. Cortes, C. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[12] P. Erdős and A Hajnal. On chromatic number of graphs and set-systems. 1966.

[13] Antonios Garas, Frank Schweitzer, and Shlomo Havlin. A k-shell decomposition method for weighted networks. *New Journal of Physics*, 14(8):083030, 2012.

[14] Trevor Hastie Robert Tibshirani Gareth James, Daniela Witten. *An Introduction to Statistical Learning with Applications in R*. Springer Texts in Statistics, Vol. 103. Springer, 2013.

[15] Christos Giatsidis, Fragkiskos Malliaros, Dimitrios M. Thilikos, and Michalis Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. *Proceedings of the National Conference on Artificial Intelligence*, 1, 07 2014.

[16] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[17] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. ctree: Conditional inference trees.

[18] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS*, 15(3):651–674, 2006.

[19] I.T. Jolliffe. *Principal Component Analysis.* Springer, 2002.

[20] Vishesh Karwa, Michael J. Pelsmajer, Sonja Petrovic, Despina Stasi, and Dane Wilburne. Statistical models for cores decomposition of an undirected random graph. *Electronic Journal of Statistics*, 11, 10 2014.

[21] Richard A. Olshen Charles J. Stone Leo Breiman, Jerome Friedman. *Classification and regression trees.* The Wadsworth statistics / probability series. CRC, 1984.

[22] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[23] Takashi Matsuda, Hiroshi Motoda, and Takashi Washio. Graph-based induction and its applications, 2002.

[24] Michael A. Newton. Introducing the discussion paper by székely and rizzo. *Ann. Appl. Stat.*, 3(4):1233–1235, 12 2009.

[25] Nilsson N.J. *Introduction to machine learning.* web draft edition, 1996.

[26] Kate Brody Nooner, Stanley Colcombe, Russell Tobe, Maarten Mennes, Melissa Benedict, Alexis Moreno, Laura Panek, Shaquanna Brown, Stephen Zavitz, Qingyang Li, et al. The nki-rockland sample: a model for accelerating the pace of discovery science in psychiatry. *Frontiers in neuroscience*, 6:152, 2012.

[27] Ben-David S. Shalev-Shwartz S. *Understanding Machine Learning: From Theory to Algorithms.* CUP, draft edition, 2014.

[28] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems*, 54(3):677–710, Mar 2018.

[29] Helmut Strasser and Christian Weber. On the asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 2, 1999.

[30] Gabor Szekely and Maria Rizzo. Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference*, 8, 08 2013.

[31] Gábor J Székely, Maria L Rizzo, Nail K Bakirov, et al. Measuring and testing independence by correlation of distances. *The annals of statistics*, 35(6):2769–2794, 2007.

[32] Robert Tibshirani. Regression shrinkage and selection via the lasso. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 58:267–288, 1994.

[33] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, 2 edition, 2013.

[34] Arun Venkitaraman, Hermina Petric Maretic, Saikat Chatterjee, and Pascal Frossard. Supervised linear regression for graph learning from graph signals. *CoRR*, abs/1811.01586, 2018.

[35] Stefan Wuchty and Eivind Almaas. Peeling the yeast protein network. *Proteomics*, 5:444–9, 02 2005.