Università
Ca'Foscari
Venezia

Master's Degree Programme
in **Computer Science - Data Management and Analytics**

Master's Thesis

Ca' Foscari
Dorsoduro 3246
30123 Venezia

ExplainableAI: on explaining forest of decision trees by using generalized additive models

Relatore
Chiar.mo prof. Claudio Lucchese

Laureanda
Martina De Zan
Matricola 846036

Anno Accademico
2019/2020

Anche questa è per te.
Anche adesso che non ci sei più,
sei sempre il motivo per cui sono arrivata a scrivere questa tesi.
Grazie.

Contents

Abstract

In recent years, decision support systems have become more and more pervasive in our society, playing an important role in our everyday life. But these systems, often called *black-box models*, are extremely complex and it may be impossible to understand or explain how they work in a human interpretable way. This lack of explainability is an issue: ethically because we have to be sure that our system is fair and reasonable; practically because people tend to trust more what they understand.
However, substituting black-box model with a more interpretable one in the process of decision making may be impossible: interpretable model may not work as good as the original one or training data may be no longer available.
In this thesis we focus on forests of decision trees, which are particular case of black-box models. In fact, trees are interpretable models, but forest are composed by thousand of trees that cooperate to take a decision, making the final model too complex to comprehend its behavior.
In this work we show that *Generalized Additive Models (GAMs)* can be used to explain forests of decision trees with a good level of accuracy. In fact, GAMs are linear combination of single-features or pair-features models, called *shape functions*. Since shape functions can be only one- or two-dimensional functions, they can be easily visualized and interpreted by user. At the same time, shape functions can be arbitrarily complex, making GAMs as powerful as other more complex models.

Introduction

Nowadays, every aspect of our daily life is potentially affected by decision support systems. A very simple example are the applications we use on our smartphone: Facebook uses a machine learning algorithm when it chooses to show us certain posts or certain advertisements; Spotify uses artificial intelligence to suggest new songs, Google Photos corrects colors of our shots and groups the photos according their subject. The curious thing is that we take advantage of these automated systems every day, but we have no idea how they take their decisions.
The situation becomes more complex when decision support systems takes decisions that have a much greater impact on our life. Often there are automatic systems behind the choice of a bank to grant a loan, an insurance to decide the price of a policy, a company to hire a candidate and so on. At this point, transparency of the automatic system is no longer a curiosity, but a necessity: ethically, because we must be sure that the system is fair and reasonable; practically, because people tend to trust more what they understand.
Some machine learning models are very simple and already interpretable, others - usually the most complex, accurate and used - cannot be interpreted at all. These models are often called black-box models. In the first chapter of this thesis, we analyze the concept of interpretability and see how we can tackle the problem of "opening the black-box" and understand its behaviour. Well-known technique used to interpret models are Lime [10] and Shap [9]. Both of them are agnostic local model explainers. Agnostic because they can be applied to any machine learning model since they do not analyse model internal component: they perturb input data and try to understand how prediction change. Local because they explain why was this prediction made and which variables caused the prediction.

In the second chapter of this thesis we see how forests of decision trees work. Forests are extremely accurate models and are therefore often used as decision support systems; they also have a very interesting feature: they are an

ensemble of decision trees. Decision trees are very simple and interpretable models. This particular characteristic of forests allows us not to treat them as black boxes, but allows us to observe their internal structure and get clues about their behavior. However, each tree cannot be studied independently of the other trees in the forest, since the decision-making power of the forest is given by the collaboration between the trees composing it. This makes it necessary to use another model to explain the logic used by the forest to take decisions. We choose to use Generalized Additive Models (GAMs).
GAMs are a non-linear extension of linear models: they are the sum of the results obtained from different models, called shape functions, each representing the relation between one - or maximum two - features and the response. Since each shape function is at most two-dimensional, it can be represented graphically, making the GAMs extremely intuitive and easily interpretable by humans. However, shape functions can be arbitrarily complex, making GAMs extremely powerful models.
The aim of this thesis is to find a way to obtain a GAM that acts as global, model-specific interpreter of a forest of decision trees, exploiting information that we can gather by observing a forest's internal structure. In order to act as interpreter of the forest, GAMs have to satisfy two conditions: first, GAM's shape functions plots have to grasp the relation between features; second, GAMs have to be accurate enough to imitate forest behaviour. In the fourth chapter, we propose our procedure to obtain a GAM that acts as interpreter of a forest, we explain how we conduct the tests and we illustrate different case studies, both a synthetic and real-world data.

Chapter 1

Explainable AI

The past decade has seen a resurgence of artificial intelligence mainly due to the computing power achieved by modern computers and the immense amount of data collected every day. Artificial intelligence has brought a significant improvement in decision support systems, which have become far more powerful and accurate. However, their improvement comes at the price of their interpretability. In fact, models have become so complex that they behave more and more like black boxes fed with input data and whose output is presented to us without having the slightest idea of the decision path that produced it. Lack of understanding exposes us to the risk of accepting predictions as correct, without being able to verify whether they have been compromised by an error or bias included in the data used to train the model. Explainable AI was born to investigate remedies to this problem.

In this thesis we analyze an extremely complex model, the forest of decision trees, and we try to provide an "interpreter" that is able to explain how the forest works. First, we give a definition of the explainable AI, the concept of interpretability and how we can tackle the problem of "opening a black box" and understanding its behavior.

1.1 Interpretability

Before discussing the possible explanation and interpretation of black-box predictors, we need to define what is a black-box model and what we mean with interpretable, explainable and comprehensible models.
A *black-box predictor* is a data-mining and machine-learning obscure model, whose internals are either unknown to the observer or they are known but uninterpretable by humans [3]. The meaning of *interpretability* can be defined

as the ability to explain the meaning of some concepts using understandable terms to a human [1]. In the end, an explanation is an "interface" between humans and decision maker.
The *reason* why a model should be interpretable is that we use those models to take critical decisions, like medical decisions or marketing strategies. For this reason, we need to be able to explain why a certain outcome have been returned.

1.1.1 Dimensions of iterpretability

Interpretability can be either global or local. We speak about *local interpretability* when we can understand only reasons of specific decisions; if we are able to understand the whole logic, then we are in a context of *global interpretability*[3].
Another way to measure the interpretability is *time*. User, actually, has a limited time available to understand model explanation [3]. The more time user need to understand model logic, the less the model is interpretable.
The last dimensions considered in [3] to measure interpretability is the *nature of user expertise*, which considers user background knowledge. Knowing the user experience in the task is an important aspect of the perception of interpretability of the model: if a users are expert in the task, they will prefer a larger and more sophisticated model over a smaller and more opaque one.

1.1.2 Desiderata of an interpretable model

To realize an interpretable model, we need to consider some quality that the model should have to be considered interpretable and to act as a good model. In [3] are listed some of such quality:

- *Interpretability:* model or its predictions should be human understandable;
- *Accuracy:* model should performs as good as a non-interpretable one;
- *Fidelity:* model should be able to accurately imitate the black-box predictor it is explaining;
- *Fairness:* model should protect groups against discrimination (see example of recism in data);
- *Privacy:* model should not reveal sensitive information about people;

- *Trustability:* this metric is based on two measures, which relate on how much a user can trust a model. The first one is the *monotonicity*: for example, an increase of a certain numerical attributes tends to increase (or decrease) the possibility of a record to belong to a certain class. The other measure of trustability is the *usability*: people tend to trust more models that guide them, using explanation of what is going on during task;

- *Reliability/robustness:* model should maintain certain lever of performance independently from small variations of parameters or input data;

- *Causality*: controlled changes in input due to perturbation affects model behaviour;

- *Scalability:* models should be able to scale to large input;

- *Generality:* models should not require special training regimes or restriction since models could be used with different data.

1.1.3 Already interpretable models

Actually there are models that are considered already interpretable and understandable by humans: decision tree, rules and linear models [3].

Decision Tree

A decision tree is basically a graph structured like a tree, whose internal nodes represent tests on features (ex: $x_1 \geq 3$) and whose leaf nodes represent class labels. The path from root to leaves represent the classification rules. For this reason, a decision tree can be linearized into a set of decision rules in the *if-then* form, considering the tests included in internal nodes.

$$\text{if } condition_1 \wedge condition_2 \wedge codition_3, \text{ then } outcome$$

Decision Rule

A decision rule is a function that maps an observation to an action. There are many types of decision rules:

- *if-then rule*: if combination of condition of input variables is true, then perform the correspondent action. Combination can formed by conjunction, negation and distinctions;

- *m-of-n rule*: given a set of n conditions, if m of them are verified, then rule is true;
- *list of rule*: given an ordered set of rules, all of them considered true, then the consequent of the first rule is verified;
- *falling rule list*: a list of *if-then* rules, ordered with respect to the probability of a particular outcome;
- *decision set*: unordered set of rules: each rule is an independent classifier.

Decision Rule vs Decision Tree: both have more or less the same logic, but they differ in three main aspects related to interpretability:

1. *textual representation vs. graphical representation.* Decision tree provides immediate information about feature importance: it has a hierarchical structure, in which test in the nodes closer to root are more important than tests in nodes closer to leaves;
2. *local patterns.* To analyze decision rules logic, it is sufficient to consider the set of rules; if we are using a decision tree, we have to consider every path from root to leaf nodes, which could be difficult if tree is big;
3. *ambiguity.* decision tree predict a single label for every input, every class is represented in a mutually exclusive and exhaustive way. Using decision rules to predict a label, a certain record can satisfy more than one rule, resulting in the prediction of more than one possible classes.

Linear models

Linear models are a way to visualize feature importance, they show both sign (positive or negative impact) and magnitude of the contribution of the attributes for a given prediction. It has an intrinsic problem when model does not optimally fit data, it may use spurious feature to optimize the error that may be hard to interpret for humans.

1.1.4 Model complexity

Model complexity is often tied to model comprehensibility: the more complex is the model, the harder is to explain it. For this reason, complexity is also connected to interpretability. Notice that we are considering a concept of complexity that is only related to the model and not to data, which is

generally unknown.
But how can we define complexity? In [3] a lot of definitions are provided, each of them related to a particular model: if we are considering a linear model, complexity is the number of attributes with non-zero weights; in decision trees is the tree depth; in decision rules can be the length of the rule or the number of rules in a list.
A more agnostic definition is provided by [4]: "complexity is identified by the number of regions, i.e., parts of the model, for which the boundaries are defined".

1.2 Open the Black Box problem

The problem of opening the black-box predictor is the problem of looking into the predictive model and understand the reasons behind a certain outcome when presenting a particular input. The following description of the problem is based on [3].
The problem is divided into two categories, based on two main solution approach: *reverse engineering* and *design of explanation*.
In the first case, the idea is that, given the decision record produced by the black box, we have to reconstruct an explanation for it. The other case considers building an already interpretable model, which solves the same problem as the model it is trying to explain.
As we can see in figure 1.1, the *black box explanation* problem, which correspond to the *reverse engineering* approach, can be further divided among *Model Explanation* when the explanation involves the whole logic of the obscure classifier, *Outcome Explanation*, when the target is to understand the reasons for the decisions on a given object, and *Model Inspection* when the target is to understand how internally the black box behaves changing the input. We investigate them in the next section.

1.2.1 Problem formulation

In [3] a formal and general description of the classification problem is provided, along with the formulation of all the problems we have presented in the previous introduction.

A *predictor* (or *model*, or *classifier*) is a function $b : \mathcal{X}^{(m)} \rightarrow \mathcal{Y}$, which maps data instances (tuples) x from a feature space $\mathcal{X}^{(m)}$, with m input features, to a decision y in a target space $\mathcal{Y}$. An instance x consist in a set of

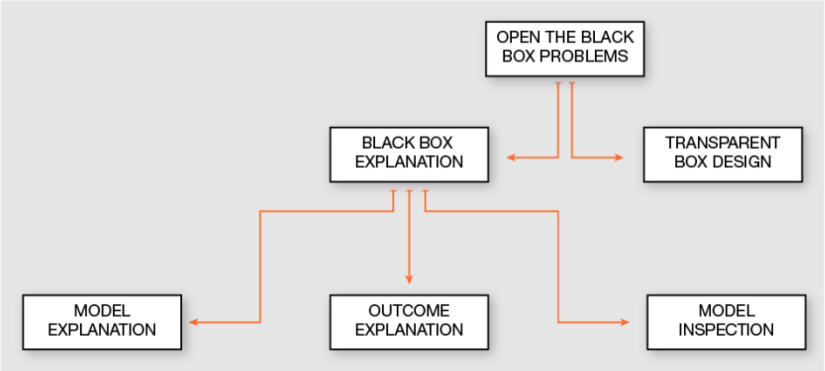

Figure 1.1: Open the black box taxonomy. The *black box explanation* part of the graph represents the approach of reverse engineering; while the *transparent box design* represents the design of explanation approach.

m attribute-value pairs (a_i, v_i), where a_i is the feature (or attribute) and v_i is a value from the domain of a_i. The domain of a feature can be continuous or categorical. The target space $\mathcal{Y}$, with dimensionality equals to one, contains the different labels and also in this case domain can be continuous or categorical.
A predictor b is a black-box predictor whose logic is unknown or uninterpretable by humans.
On the opposite site, we denote with c an interpretable predictor, with a known internal logic leading to a decision $c(x)$ that can be explained in a comprehensible way to a human.
Normally, to evaluate a model, we train it with a portion of dataset called *training set* and we evaluate the performance with the remaining portion, called *test set*. As a measure of goodness we use the *accuracy*, which is the number of matches between the prediction of the model and the true response in the test set, over the size of the test set itself.
To measure the performance of the interpretable predictor c, we have to measure how good it is in imitating the black box predictor b. This measure is called the *fidelity* and, similarly to the accuracy, counts the number of matching response between b and c, over the size of the test dataset.

We will now apply this formulation of the problem to all of the categories illustrated in figure 1.1. First of all, we will see in details the three specification of the *black box explanation* subcategory of the "open the black box" problem, which are *model explanation*, *outcome explanation* and *model inspection*. In the end, we will see the application of the problem formulation to the last subcategory, the *transparent box design*.

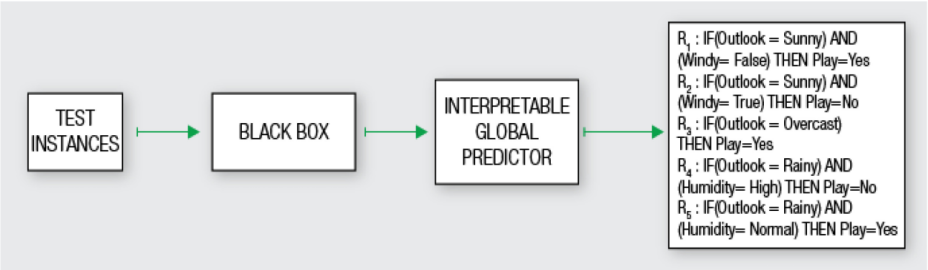

Figure 1.2: Model Explanation problem example. Starting from test instances in X, forst query the black box and then extract an interpretable global predictor from $X, b(X)$ in the form of a decision rule classifier.

1.2.2 Model explanation

The goal of the *model explanation* problem consist in providing a global explanation of the black box model b through an interpretable and transparent model c, which should be both able to mimic the black box behavior and understandable by humans. For this reason, we can say that the model c approximating the black box model b must be globally interpretable.
To formalize the problem, we assume that the interpretable model c is derived from model b and a dataset X, which is provided by user and obtained by sampling the domain $\mathcal{X}^{(m)}$. Dataset X may also include actual class values (to estimate accuracy of c), as we see in figure 1.2.

Definition. Given a black box predictor b and a set of instances X, the *model explanation problem* consist in finding an explanation $E \in \mathcal{E}$, belonging to a human interpretable domain $\mathcal{E}$, through an interpretable global predictor $c_g = f(b, X)$ derived from the black box b and the instances X using some process $f(\cdot, \cdot)$. An explanation $E \in \mathcal{E}$ is obtained through c_g, if $E = \varepsilon_g(c_g, X)$ for some explanation logic $\varepsilon_g(\cdot, \cdot)$, which reasons over c_g and X.

1.2.3 Outcome explanation

While model explanation aims to find an explanation of the whole black box model b, outcome explanation problem consists in providing an explanation e for the outcome of the black box b on a given instance x. It only needs to explain the reasons for the prediction on a specific input instance.
To formalize the problem, we assume that the whole process is divided into two stages: first, we build an interpretable local model c_l from the black box model b and the instace x, then an explanation e is derived from c_l, as we see in figure 1.3.

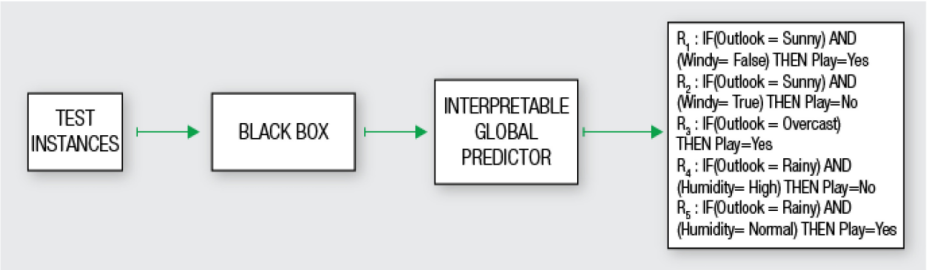

Figure 1.3: Outcome Explanation problem example. For a test instance x, the black box decision $b(x)$ is explained by building an interpretable local predictor c_l, e.g., a decision rule classifier. The local explanation $\varepsilon_l(c_l, x)$ is the specific rule used to classify x.

Definition. Given a black box predictor b and an instance x, the outcome explanation problem consists in finding an explanation $e \in \mathcal{E}$, belonging to a human-interpretable domain $\mathcal{E}$, through an interpretable local predictor $c_l = f(b, x)$ derived from the black box b and the instance x using some process $f(\cdot, \cdot)$. An explanation $e \in \mathcal{E}$ is obtained through c_l, if $e = \varepsilon_l(c_l, x)$ for some explanation logic $\varepsilon_l(\cdot, \cdot)$, which reasons over c_l and x.

1.2.4 Model inspection

Model inspection problem aims to provide a visual or textual representation r for understanding some specific property of the black box model b or of its predictions. For example, a representation r may be based on sensitive analysis: we observe how predictions of black box model b change when varying the input and we can obtain partial dependence plots or variable effect characteristic curve that indicates how predictions are influenced by features, see figure 1.4.
Model inspection problem differs from model explanation problem in the fact that model explanation problem requires the extraction of an interpretable global predictor, while model inspection problem focuses on the analysis of specific properties of the black box model b without requiring a global understanding of it.

Definition. Given a black box model b and a set of instances X, the model inspection problem consists in providing a visual or textual representation $r = f(b, X)$ of some property of b using some process $f(\cdot, \cdot)$.

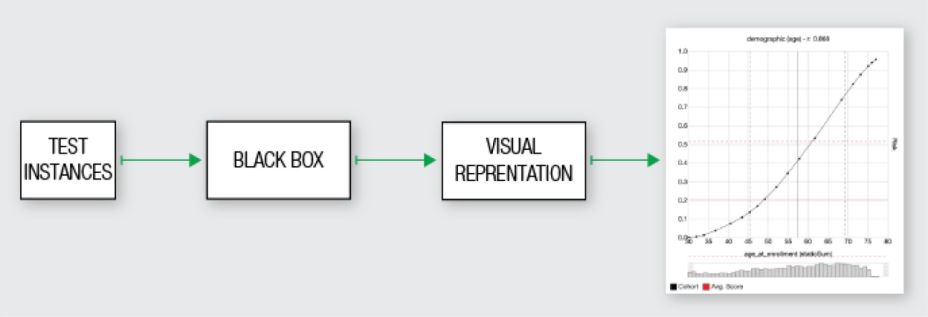

Figure 1.4: Model inspection problem example. Query the black box on test instances X, and then extract a sensitivity analysis plot.

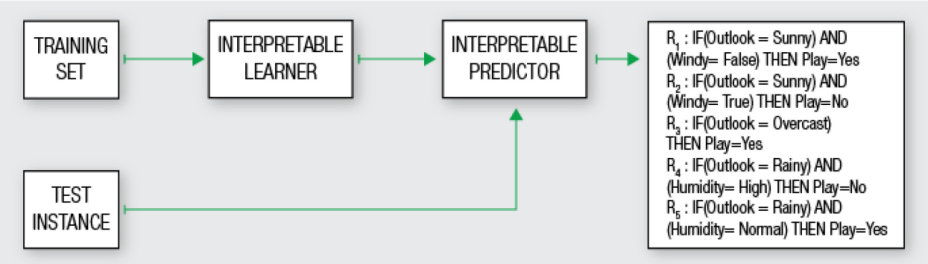

Figure 1.5: Transparent box design problem example. A decision rule classifier learned from a training dataset is globally interpretable predictor. Moreover, the rule that applies on a given test instance is a local explanation of the predictor's decision.

1.2.5 Transparent box design

While model explanation, outcome explanation and model inspection problems refer to the *reverse engineering* approach to the *Open the Black Box problem*, the transparent box design adopt the *design of explanation* approach. In fact, the transparent box design consists in directly providing a model that is locally or globally interpretable, for example a decision tree classifier.

Definition. Given a training set $D = (X, \hat{Y})$, the transparent box design problem consists in learning a locally or globally interpretable predictor c from D. For a locally interpretable predictor c, there exists a local explanator logic ε_l to derive an explanation $\varepsilon_l(c, x)$ of the decision $c(x)$ for an instance x. For a globally interpretable predictor c, there exists a global explanator logic ε_g to derive an explanation $\varepsilon_g(c, X)$

1.3 Conclusion

To summarize, according to the problem definition proposed at the beginning of this section, when stating that a method is able to "open the black box", we are referring to one of the following statements: (*i*) it explains the model, (*ii*) it explains the outcome, (*iii*) it can inspect the black box internally, (*iv*) it provides a transparent solution.

The aim of this thesis is to obtain a generalized additive model (GAM) that is able to imitate the behavior of a forest of decision trees.
Although decision trees are an extremely interpretable model, the composition of hundreds or more of them collaborating to take decisions creates an extremely complex model that can be seen as a black-box model.
The GAM, on the other hand, is a linear combination of one or two dimensional models that can be represented graphically, which makes GAMs models extremely interpretable for humans.
In the light of what we have seen in this chapter, our thesis is configured as a model explanation problem [1.2.2]. In fact, a GAM that can imitate a forest of decision trees is a great global explanation of the forest itself.

Chapter 2

Forest of decision trees

As we see in the previous chapter, decision trees are very simple models that can already be interpreted by humans.
Forests, on the other hand, are a composition of thousands of decision trees that cooperate to make a decision. They are extremely complex, yet extremely powerful models, which makes them very popular as decision support systems.
In this chapter we see how decision trees are structured and we describe some popular algorthms for building forests of decision trees.

2.1 Decision Trees

A decision tree is a hierarchical structure consisting of nodes and directed edges [11]. Tree has three types of nodes:

- root node: this node has no incoming edges and zero or more outgoing edges;
- internal nodes: these nodes have exactly one incoming edge and two or more outgoing edges;
- leaf or terminal nodes: these nodes have exactly one incoming edge and no outgoing edges.

In a decision tree, each leaf node is assigned a value: if the problem we are trying to solve is a classification problem, leaf node contains a class label; otherwise, if the problem we are facing is a regression problem, leaf node contains a numerical value. Non-terminal nodes - which include root and other internal nodes - contain attribute test conditions to separate records

that have different characteristics.
Once a decision tree has been constructed, classifying a test record is straightforward: starting from the root node, we apply the test condition to the record and follow the branch corrsponding to test outcome; this will lead us either to another internal node - and another test condition - or to a leaf node. Class label associated with the leaf node is then assigned to the record. Regression problem works exactly in the same way.

2.1.1 How to build a decision tree

There are exponentially many decision trees that can be constructed from a given set of attributes and, for this reason, finding the optimal tree is computationally infeasible [11]. However, there exist efficient algorithms to induce a reasonably accurate - even if suboptimal - decision tree in a reasonable amount of time. Usually, these algorithms employ a greedy strategy that grows a decision tree by making a series of locally optimum decisions about which attribute to use for partitioning the data. One such algorithm is Hunt's algorithm, which is the basis of many existing decision tree induction algorithms for classification problems.

Hunt's algorithm

In Hunt's algorithm, a decision tree is grown in a recursive fashion by partitioning the training records into successively subsets [11]. Let D_t be the set of training records that are associated with node t and $y = \{y_1, y_2, \ldots, y_c\}$ be the class labels. Recursively do:

1. If all records in D_t belong to the same class y_t, then t is a leaf node labeled as y_t.

2. If D_t contains records that belong to more than one class, select an attribute test condition to partition records into smaller subsets. A child node is created for each outcome at the test condition and records in D_t are distributed to the children based on the outcomes. Algorithm is then recursively applied to each child node.

Design issues of decision tree induction

Every decision tree learning algorithm - including Hunt's algorithm - must address the following two issues [11]:

How should the training records be split? Attribute values can have different types: they can be binary (yes/no), nominal (many different values, no order), ordinal (many different values, can be ordered) or continuous. Moreover, we can choose to split the node into two children nodes (binary split) or we can choose to design the attribute test condition to have more than two outcome (multiway splits). In the end, we need a way to select the best split: often measures developed for selecting the best split are based on the degree of impurity of the child nodes and depends on the type of problems we are facing, classification or regression. Typically, in order to find the best split, we enumerate all possible split for every possible predictor and we calculate the impurity measure of children nodes for every configuration. The best split is the one with the lowes impurity measure obtained.
Examples of impurity measures adopted in case of classification problems include:

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i|t) log_2 p(i|t),$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2,$$

$$Classification_error(t) = 1 - max_i[p(i|t)],$$

where c is the number of classes and $0log_20 = 0$ in entropy calculations.
Examples of impurity measures adopted in case of regression problems include:

$$least\_absolute\_deviations(t) = \sum_{i=1}^{n} |y_{true} - y_{predicted}|,$$

$$least\_square\_errors(t) = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2.$$

How should the splitting procedure stop? A possible strategy is to continue expanding a node until either all the records belong to the same class or all the records have identical attribute values. Both conditions are sufficient to stop any decision tree induction algorithm, but they are not enough to avoid the overfitting problem. There are two strategies for avoiding decision tree overfitting problem:

- Pre-pruning (or early stopping rule): algorithm is halted before generating a fully grown tree. For example, we can choose to stop expanding a leaf node when the observed gain in impurity measure falls below a certain threshold. This strategy has a big disadvantage: it is difficult

to choose the right threshold for early termination. A threshold too high will result in underfitted model, viceversa tree may suffer from overfitting anyway.

- Post-pruning: tree is initially grown to its maximum size, then we proceed with a tree-pruning step by trimming the fully grown tree in a bottom-up fashion. Trimming can be done by replacing a subtree with a new leaf node whose class label is determined from the majority class of records affiliated with the subtree, or the most frequently used branch of the subtree. The tree-pruning step terminates when no further improvement is observed.

2.2 Ensemble of trees: Random Forests and Boosting

Random forests and boosting use trees as building blocks to construct more powerful prediction models [7].

2.2.1 Random forests

Decision trees suffer from high variance: if we split the training data into two parts at random and fit a decision tree to both halves, the result we get could be quite different. A general-purpose procedure for reducing the variance of a statistical learning method is the *bootstrap aggregation* or *bagging* and it is based on the fact that given a set of n independent observation $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean Z_{avg} of the observation is given by σ^2/n. In other words, averaging a set of observations reduces variance [7].
Bagging works as follows: first we generate B different bootstrapped training dataset by repeatedly sample the original training dataset; then, we learn a decision tree from each one of the B bootstrapped dataset and, in the end, we average all the prediction of the trees (or we take the most voted prediction among all the predicted class if we are facing a classification problem).

Random forests provide an improvement over bagged trees trying to *decorrelate* the trees [7]. As in bagging, we build a number of decision trees on bootstrapped training samples. Peculiarity is that decision trees composing the random forest are built considering only a random sample of attributes each time a split in a tree is performed. This is done in order to force trees to consider every attributes when taking their decision. In fact, if we suppose

that there is an attribute that can be considered a very strong predictor, then in the collection of bootstrapped trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bootstrapped trees will look quite similar to each other and the prediction from the bagged trees will be highly correlated. However, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities [7]. Random forests overcome this problem by forcing each split to consider only a subset of m random predictors from the full set of p predictors. Therefore, on average $(p-m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance and we obtain less correlated predictions.

2.2.2 Boosting

Boosting procedure does not create multiple copies of the original training set using the bootstrap.
If we are facing a regression problem, algorithm works as follows: it starts by growing a single decision tree from the original dataset and it calculates the difference between the true response and the response predicted by the tree. Such difference is called *residual*. Then, it fits a second decision tree to the residuals from the first tree and so on. The prediction of the boosting is obtained by calculating the sum of each tree in the forest.
If we are facing a classification problem, clearly we cannot compute residuals. To overcome this problem, once the first tree is grown, we assign a weight $D > 1$ to mis-classified samples and we grow the second tree applying weight D to samples when calculating the impurity measure (Gini/Entropy) and so on for subsequent trees. In this way, boosting creates successive base classifiers that place greater emphasis on mis-classified samples. In the end, like in bagging, results from all boosting base classifiers are aggregated.
We can say that in the boosting procedure trees are grown sequentially [7]: each tree is grown using information from previously grown trees.
Boosting has three tuning parameters (four in case of classification boosting):

- **number of trees:** boosting can overfit if the number of trees is too large. This parameter can be selected using the cross validation;

- **shrinkage parameter:** this controls the rate at which boosting learns. This parameter's typical values are 0.01 or 0.001 and it is multiplied to the sum of trees prediction. Very small values of this parameter can require a large number of trees in order to achieve good performance;

- **number of splits in each tree:** it controls the complexity of the boosted ensemble. This parameter is called also *interaction depth* and controls the interaction order of the boosted model, since d splits can involve at most d variables;

- **weight:** weight D to be assigned to mis-classified example during training process (only in case of classification boosting).

Random forest vs boosing. key difference between boosting and random forest is how decision trees are built: in boosting, because the growth of a particular tree takes into account the other trees that have already been grown, smaller trees are typically sufficient.
Another difference lies in how they combine results: random forests combine results at the end of the process (by averaging or "majority rules") while gradient boosting combines results along the way.
In [7] performance of boosting and random forests are compared: if we carefully tune parameters, boosting can result in better performance that random forest, but random forest are more resistant to noise.

2.3 Conclusion

To summarize, forst we see how decision tree works and how they are organized. Then, two algorithms to learn forests are described: Random Forests and Boosting. Random forests is based on bootstrap aggregation: B different bootstrap training dataset are generated and a decision tree is learned from each one of them. Each decision tree considers a random subset of features. Boosting, on the other hand, has a more sequential behaviour. It starts by growing a single decision tree from the original dataset; then, it fits a second decision tree on the residuals from the first tree and so on.

In this thesis we choose to use the boosting algorithm to learn the forest of decision trees. We do not use the random forest because the random choice of features makes it difficult to replicate the experiment. In any case, the procedure described in this thesis to obtain a GAM capable of interpreting forest behavior can be applied to both random forests and boosting.
Also, we use binary trees when building the forest.

Chapter 3

Generalized additive models (GAM)

Forests of decision tree, as we see in chapter 2, are extremely accurate models, but they are so complex that they are no longer interpretable by users. This makes it necessary to have another, interpretable model that can explain forest decision process.
As we see in section 1.1.3, there are models that are already interpretable by humans: decision trees, decision rules and linear models; however, these models are extremely simple and totally inadequate in imitating the behavior of a forest of decision trees.
Generalized Additive Models (GAMs) can be seen as a non-linear extension of linear models which makes them capable of modeling very complex problems, but remaining easily interpretable by humans since they can be represented graphically. In this chapter we explain how they work.

3.1 Model description

Let's suppose we have a response random variable Y and a set of predictor random variables X_1, X_2, ..., X_p and suppose we collect n observations of these random variables, denoted by $(y_i, x_{i1}, x_{i2}, \ldots, x_{ip})$, for each $i \in (0, n)$.

The basic model for multiple linear regression is

$$Y = f(X_p) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p,$$

where β_i values are the values to be estimated.

Representing $f(X_p)$ by a liner model is usually a convenient approximation, since linear model are extremely easy to interpret. However, is extremely unlikely that the true function $f(X_p)$ is actually linear: it is much more likely that $f(X_p)$ will be nonlinear and nonadditive in X.
Generalized Additive Models extends linear models by replacing the linear function $\sum_1^p \beta_i X_p$ by an additive function $\sum_1^p s_i(X_p)$, where $s_i(\cdot)$ is an unspecified smooth function that can be estimated by any so-called *scatterplot smoother*, for example a running mean, running median, running least squares line or a spline [5].
Final formulation of the model is:

$$Y = g(X_p) = s_1(X_1) + s_2(X_2) + \cdots + s_p(X_p) + error,$$

where $g(\cdot)$ is called the *link function* and $s_i(\cdot)$ are called *shape functions*.

3.1.1 GA^{2}M: GAM with pairwise interactions

Standard GAMs are easy to interpret since users can visualize the relationship between the univariate terms of the GAM and the dependent variable through a plot $f_i(x_i)$ vs. x_i. However there is unfortunately a significant gap between the performance of the best standard GAMs and full complexity models.
In [8], authors suggest adding terms of interacting pairs of features to standard GAMs. In fact, two-dimensional interactions can still be rendered as heatmaps of $f_{ij}(x_i, x_j)$ on the two-dimensional x_i, x_j-plane, and thus a model that includes only one- and two-dimensional components is still intelligible.

The resulting models, called GA^{2}M-models, for *Generalized Additive Models plus Interactions*, consist of univariate terms and a small number of pairwise interaction terms and has the form:

$$Y = g(X_p) = \sum_{i=1}^{p} s_i(X_i) + \sum_{i=1}^{p} \sum_{j=i}^{p} s_{ij}(X_i, X_j) + error.$$

3.2 From linear regression to piecewise polynomials and splines

As we said before, it is extremely unlikely that the true function $f(X)$ is actually linear, we need then a way to move beyond linearity and extend linear regression to a non-linear relation between feature X and response Y.

Polynomial regression extends the linear model by augmenting the vector of inputs X with additional variables, which are transformations of X, and then use linear models in this new space of derived input features.
For example, let's suppose that X is one-dimensional. A linear model to fit data will be in the form:

$$f(X) = \beta_0 + \beta_1 X.$$

If we want to move into a polynomial regression problem, for example a *cubic regression*, we add two extra features obtained as transformation of X: the second degree polynomial transformation X^2 and the third degree polynomial transformation X^3. The final model have the form:

$$f(x) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3.$$

As we increase the polynomial degrees of the transformation of X, the curve obtained contains high oscillations which leads to shapes that are over-flexible.
However, polynomial regression has some issues: if we increase the polynomial degree of the transformation of X too much, we can run into an overfitting problem; furthermore polynomial regression is very susceptible to extreme values: if we add a value of Y extremely high or extremely low, it will change the shape of the whole polynomial, affecting also the fit of data that are very far away from the extreme Y value.

In order to overcome disadvantages of polynomial regression, we can use an improved regression technique which, instead of building one model for the entire dataset, divides the domain of X into contiguous intervals and represents f by a separate polynomial in each interval. Such technique is known as **piecewise polynomials** [6].
The points of the domain of X determining intervals are called *knots* and are represented by the notation ξ_i, with $i = 1, \ldots, n$ where n is the cardinality of the nodes, ie how many parts the domain of X has been divided into. The single functions are known as *piecewise functions* and are indicated by the notation $h_i(X)$. Piecewise functions can be constant functions, or they can be formulated as polynomial regression problems.
For example, let's suppose we divide the domain of X into three intervals with two knots at points ξ_1 and ξ_2 and suppose to use polynomials that reach at most grade 3. The model to be solved will be:

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i, & if\ x_i < \xi_1 \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i, & if\ \xi_1 < x_i \leq \xi_2 \\ \beta_{03} + \beta_{13} x_i + \beta_{23} x_i^2 + \beta_{33} x_i^3 + \epsilon_i, & if\ x_i \geq \xi_2 \end{cases} \tag{3.1}$$

Using more knots leads to a more flexible piecewise polynomial, as we use different piecewise functions for every interval of the domain and each piecewise function depends only on the distribution of data of that particular interval.

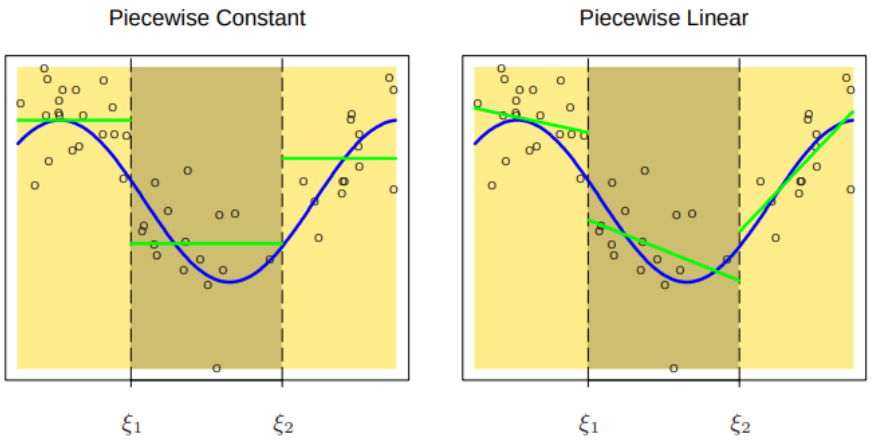

Figure 3.1: Left panel shows a piecewise constant function fit to some artificial data. Broken vertical lineas indicate positions of the two *knots* ξ_1 and ξ_2. Blue curve represents the true function, from which data were generated. Right panel shows piecewise linear functions to fit the same data.

Fig. 3.1 shows two simple piecewise polynomials: the first is piecewise constant with three piecewise functions and two knots; while the second shows a piecewise linear fit on the same data.
Still in figure 3.1 we can observe the behavior of the various piecewise functions at the knots: the green lines are not continuous at the extremes of the intervals. We can avoid this by adding an extra constraint that the polynomials on either side of a knot should be continuous at the knot. In the first two panels of figure 3.2 we can see the effect of this constraint: polynomials functions are now continuous, but the smoothness at the knots is still absent. To solve this problem, we can add another constraint: the first derivative of both polynomials functions must be the same.
It is important to notice that each constraint that we impose on a piecewise polynomial effectively frees up one degree of freedom and we reduce the complexity of the resulting piecewise polynomial fit.
Left-bottom panel of fig. 3.2 shows the effects of the last introduced constraint; while the bottom-right panel is the same polynomial with a new constraint: the second derivative of both polynomials functions at the knots must be the same. The final results is a good approximation of the true function and, in particular, at this step we obtain a *cubic spline*.

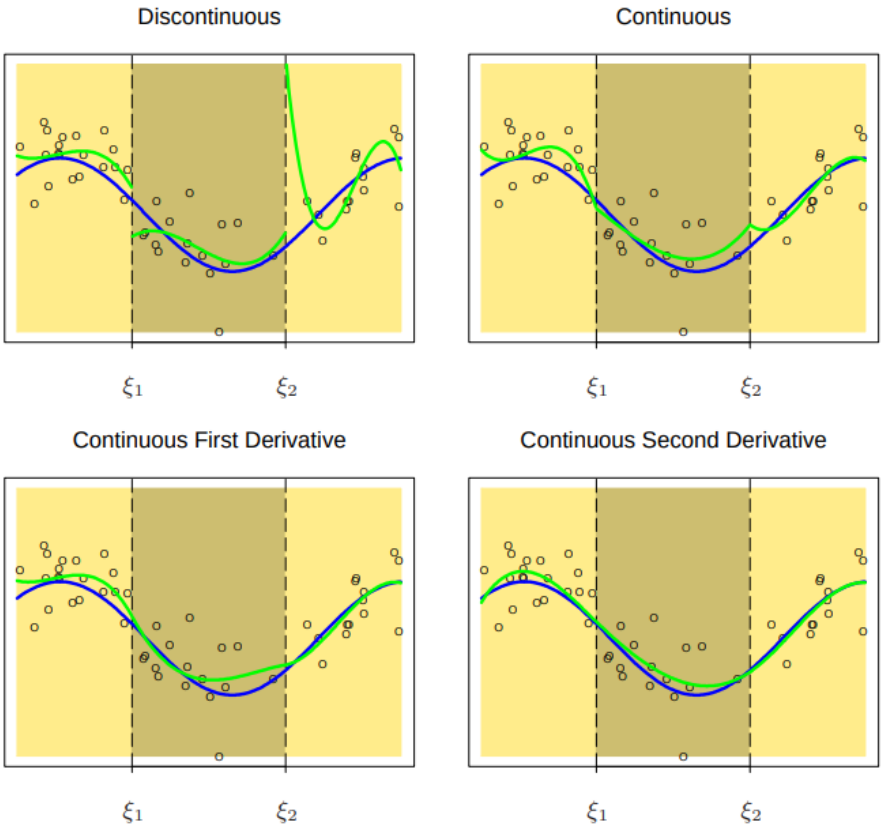

Figure 3.2: A series of piecewise-cubic polynomials with increasing orders of continuity.

More in general, an **order-M spline** with knots ξ_i, $i = 1, \ldots, K$ is a piecewise polynomial of order M, and has continuous derivatives up to order $M-2$ [6]. A cubic spline has $M = 4$ and continuous derivative up to the 2^{nd} order. There is seldom any good reason to go beyond cubic-splines, unless one is interested in smooth derivatives.
A final consideration concerns the behavior of the function at the boudaries of the intervals, where the data are less in number and more subject to variability. To mitigate this phenomenon, it can be assumed that the model is linear at these particular regions. In this case we speak about **natural splines**.

3.3 Solve the knot selection problem: *smoothing splines*

Now there remains the problem of deciding how many knots to use and where to place them. An option would be to place a knot in an area of high variability, since in those regions polynomial coefficients can change rapidly. Hence, one option is to place more knots in places where the function might vary most rapidly, and to place fewer knots where it seems more stable. Obviously, this requires prior knowledge which is not normally available. For this reason, knots are usually distributed on a uniform fashion and cross validation is used to elect the best number of knots.

It is important to notice that there exists a method that avoids the knot selection problem completely by using a maximal set of knots.
The method, known as *smoothing splines*, consists in solving the following problem: among all functions $f(x)$ with two continuous derivatives, find one that minimizes the penalized residual sum of squares

$$RSS(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 dt, \tag{3.2}$$

where λ is a fixed *smoothing parameter*. First term measures closeness to data, while second term penalizes curvature in the function, and λ establish a trade off between the two. Two special cases are:

- $\lambda = 0$: f can be any function that interpolates the data;
- $\lambda = \infty$: the simple least squares line fit, since no second derivative can be tolerated.

3.4 Conclusion

To summarize, Generalized Additive Models are a non-linear extension of linear models since they replace the linear function $\sum_1^p \beta_i X_p$ by an additive function $\sum_1^p s_i(X_p)$, where $s_i(\cdot)$ is an unspecified smooth function, usually a spline. Moreover, GAMs can be estended to consider also interacting pair of features.

In this thesis we use GAMs to build a model that is able to interpret and imitate the behaviour of a forest. For this reason, we use GAMs that considers also interacting pairs of features (GA^2M) to better imitate complex

forest behaviour.
We choose to use splines as shape functions and - in particular - we use smoothing splines to solve the knot selection problem and obtain a function that fits data as much as possible without overfitting the data.

Chapter 4

GAM as forest explainer

4.1 Introduction

In recent years, decision support systems have become more and more pervasive in our everyday life. They are used by companies to analyze sales and production costs, determining which products will be produced tomorrow and which, instead, will be removed from the market. Another application of decision support systems is the analysis of the risk to which a bank or insurance is exposed, which is also decisive in the process of evaluating a customer's request for a service. Other applications of decision support systems relate to telecommunications, healthcare, environment, advertising and many others. Every aspect of our private and social life is potentially influenced by these systems and, for this reason, it is crucial that decision support systems are as accurate and precise as possible in their predictions on which decisions depend.
However, the decision accuracy is not the only important aspect: interpretation of the model is also fundamental. Very often decision support systems are extremely complex models and this complexity makes it difficult to understand how they work. Interpretability, as we described in the section 1.1.2, allows to check whether the system used to make decisions is fair and reasonable, without bias or prejudices. Additionally, users tend to place more trust in tools they are able to understand.
Usually, when we are facing the problem of interpreting an extremely complex model, the process is the following: the model is managed as a *black box*, with no assumptions about its internal structure; the model is then fed with different inputs and the behavior of the output is studied as the input changes. For example, suppose we have a model that decides whether to grant a loan to a certain person. We feed the system with person's data

and observe the result. Then we perturb original data: how does the system output change assuming the person is a few years older? Or with a less profitable job? Or if he's wearing yellow socks? With this procedure it is possible to understand which features are important (i.e. age, job) and which are not (i.e. the color of the socks) and to what extent important features influence system decision.
However, the model we choose to interpret is very particular. Forests of decision trees are extremely complex and accurate models, but they are a composition of many extremely simple models. In fact, as we illustrated in section 1.1.3, decision trees are models interpretable by definition, along with decision rule and linear models. This property of decision trees allows us not to treat the forest as a *black box*: we can indeed observe the internal structures of the trees to get clues about model behavior. Unfortunately, each tree cannot be studied independently of the other trees in the forest, since the decision-making power of the forest is given by the collaboration between the trees composing it. This makes it very complicated to exploit the analysis of individual trees to get an interpretation of the forest, so we decide to use another model to explain the logic used by the forest to take decisions. In particular, we choose to use Generalized Additive Models (GAMs).
As in the previous chapters, GAMs are the sum of the results obtained from many very simple models, called *shape functions*, each representing the relationship between one - or maximum two - features and the response. Each *shape function* can be arbitrarily complex, making the final model almost as powerful as a forest of decision trees. However, since each *shape function* is at most two-dimensional, it can be represented graphically, making the GAMs extremely intuitive and easily interpretable by humans.

4.1.1 Overview of the procedure

The procedure for obtaining a GAM that acts as an interpreter of a forest of decision trees is divided into two phases:

1. **Forest Structure Analysys.** In this phase, we analyse internal structure of the trees composing the forest to get clues about forest behaviour, i.e. which features are used most often, which ones are likely to interact and which feature values have a greater impact in determining the response;

2. **GAM Extraction.** In this phase, the GAM is trained on the basis of the results obtained during previous phase.

Forest structure analysis. The first phase - see [4.3] - is the most important phase and has two main objectives:

- identify the most important features in the forest and how they interact with each other;
- build a dataset from which the GAM can be extracted.

In particular, during the forest structure analysis, all the tests contained in the internal nodes of the trees in the forest are extracted. Each test is a pair $< feature; threshold >$.
By observing where and how often a *feature* appears within the trees, it is possible to estimate the importance of that feature; while observing the behavior of two features at the same time allows to identify possible interactions between them. This aspect in detail in section 4.3.2.
On the other hand, by grouping the *thresholds* that appear for each features, it is possible to estimate the domain of the features, i.e. which values the feature can assume. This aspect in detail in section 4.3.3.

GAM extraction. The second phase deals with processing the results obtained from the forest structure analysis to obtain a GAM that acts as an interpreter of the forest.
In order to be able to train a GAM, we need three things:

- a dataset;
- which features to consider as *terms*, ie. which features to consider when training the *shape functions*.

The information relating to the most important features and their interactions is used to choose the *terms* of the GAM, while the information relating to the domain of the features is used to produce the datasets from which to extract the GAM.
Since there is no procedure valid for all forests, the second phase exploits different methods to compose datasets starting from the results obtained from the forest structure analysis. These datasets are then used to train different GAMs and we choose the model whose behavior is most similar to that of the forest. The details of this phase are described in section 4.4.3.

4.2 Experimental setup

In this section we explain the experimental setup that we apply to test and evaluate our procedure.
First of all, datasets used are described, then we illustrate how we prepare and conduct the tests and, at the end, the metric used to evaluate models is explained.

4.2.1 Datasets

To validate the proposed methodology, we chose to use five different datasets. Two of them are synthetically extracted from a known model - one with interactions, the other without interactions - in order to have valid references to evaluate the results of the first phase of the procedure, i.e. the extraction of the most important features and the estimate of the domain of the features. Other three datasets are based on real data and are used as *case studies* to illustrate the results that can be obtained by applying the procedure.
The five dataset we use are:

- the `data_no_inter` dataset, a synthetic dataset without interactions;
- the `data_inter` dataset, a synthetic dataset with an interaction;
- the `concrete` dataset from the UCI repository, with 8 features and 1030 entries. The aim is to predict the concrete compressive strength, which is a highly nonlinear function of age and ingredients;
- the `houses` dataset from Kaggle, with 21614 entries and 18 features. The aim is to predict the price of the houses by observing their characteristics.
- the `YearPredictionMSD` dataset from the UCI repository, with 90 features and 515345 entries. The aim is to predict the release year of a song from audio features;

Dataset	**Entries**	**Features**
Synthetic no inter	50.000	6
Synthetic w/ inter	50.000	6
Concrete	1.030	8
Houses	21.614	18
Years	515.345	90

Table 4.1: Sum up of dataset characteristics.

`Data_no_inter` **dataset.** In order to investigate the information contained in forest internal nodes, we create a synthetic dataset with 50.000 points, generated by the model:

$$y = x_0^3 - 12x_1^2 + \sqrt[3]{x_2^4} - \frac{x_3}{4} + x_4 + log(x_5^2 + 1)$$

Some of the features have a large impact on the final response, like feature x_0 or x_2 which grows super-exponentially. Other features, like x_3 or x_4 or x_5 are less significative. In general, all the features behave as non-complex functions, with a small number of maximum or minimum points and never oscillate. In fig. 4.1 we can see the functions of the model generating the artificial dataset used to investigate on forest internal node thresholds.

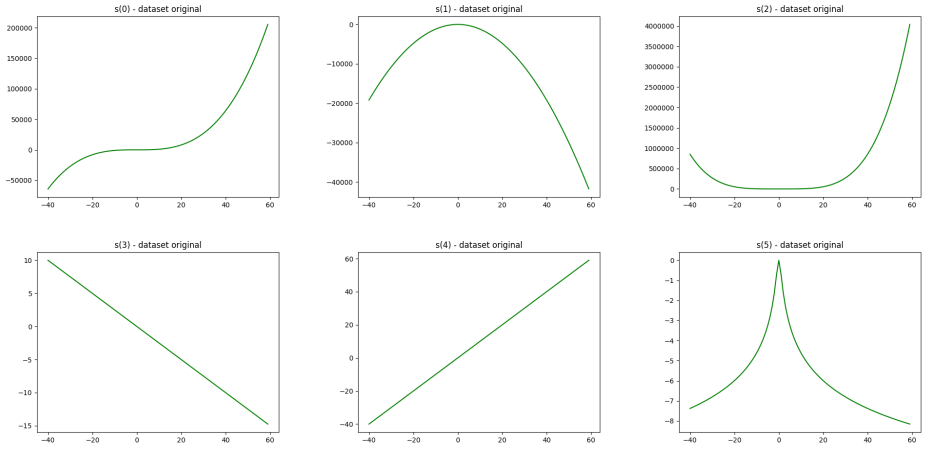

Figure 4.1: Functions of the model generating the artificial dataset used for investigate on forest internal node thresholds.

`data_inter` **dataset.** In order to test how procedure works when detecting interactions between features, we create another synthetic dataset with 50.000 points, generated by the model:

$$y = x_0 sin(\frac{x_0}{4}) + 7log(x_1^2 + 1) + 25(atan(x_2) + sin(\frac{x_2}{10}))-$$

$$-350000(Gauss(x_3, x_4, \mu = 0, \sigma^2 = 500) - 35) - (x_5^2)^{1/3},$$

where $Gauss(x_3, x_4, \mu = 0, \sigma^2 = 500)$ is a gaussian function with zero mean and variance equal to 500. This dataset contains features that have a more complex behavior than the dataset described in previous paragraph, but each

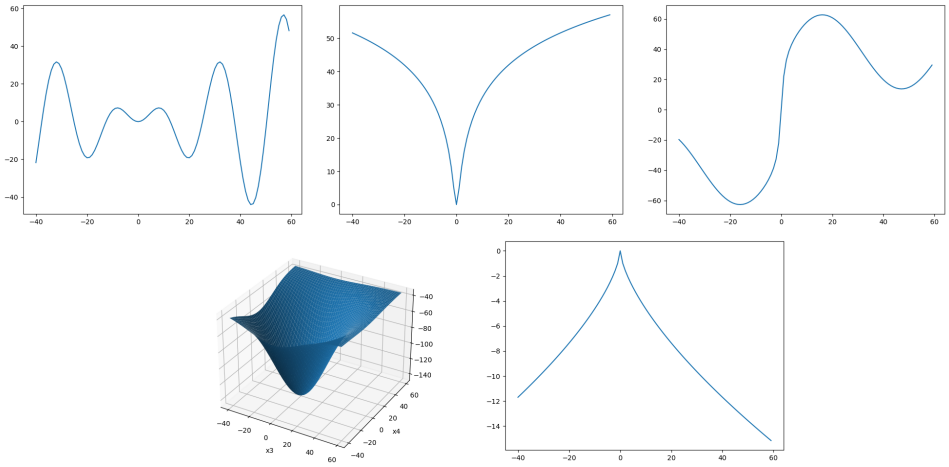

Figure 4.2: Functions of the model generating the second artificial dataset. This models has an interaction between features x_3 and x_4.

feature has more or less the same impact on the final response. In fig 4.2 we can see graphs of the functions generating the model.

Note about `houses` dataset. Dataset `houses` response distribution has a long right tail and this leads to an increase in the value of the global RMSE. In fact, the prices of most of the houses in the dataset lay between \$75,000 and \$1,500,000, while a subset of about 500 cases were priced between \$1,500,000 and \$7,500,000. Also due to the approximation made by the heuristics applied to reconstruct the dataset starting from the forest, some GAMs were unable to adequately model these particular cases and their performance was particularly affected. We therefore decided to apply the log function to house prices.

4.2.2 Metric

Since we are facing only regression problems, we decide to use RMSE as measure of accuracy of the models we are testing.

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

where y_i is GAM output when feeded with the i^{th} record of test dataset and $\hat{y}_i$ is the *true response*.

4.2.3 Methodology

Forest. From each dataset we train a forest using the `lightgbm`[1] library. Each tree in the forests has 32 leaves. We used *early stopping* as a stopping criterion, using accuracy as a measure of goodness and stopping training when no improvement is achieved for 50 consecutive rounds. We also fixed the number of trees in the forest not to exceed 10,000. In table 4.2 we can see, for every dataset, how many trees are in the forest and the value of the RMSE of the forest with respect to the original dataset.

Lightgbm

Dataset	n. of trees	RMSE
Synthetic no inter	111	11073.95
Synthetic w/ inter	5739	2.17
Concrete	331	3.91
Houses	341	0.16
Years	4612	8.68

Table 4.2: Forests extracted from datasets.

GAM. To train the GAM, we use the library `pyGAM`[2] . In particular, we use the object `LinearGAM`, which exploits the linear function as link function. To model terms of single features, we use `splines`; while, to model interaction terms, we use `tensor product`.
Moreover, library `pyGAM` implements the *smoothing spline* procedure, then we do not need to set the number of knots for each spline. We use the library default value $\lambda = 0.6$ as *smoothing parameter*.
It is important to note that while using this library we ran into some performance issues. In particular, when the dataset is large or with many features or the model involves the use of many nodes for shape functions, the process may fail due to excessive memory requirements. This limitation of the library affected some aspects of the procedure, in particular during the phase of creating the datasets necessary to train the GAMs [see section 4.3.3].

[1] `https://lightgbm.readthedocs.io/en/latest/`
[2] `https://pygam.readthedocs.io/en/latest/`

4.3 Forest structure analysis

The first step in the process of obtaining a GAM that can works as an interpreter for a forest of decision trees is the Forest Structure Analysis. A forest of decision trees is an extremely powerful and complex model, but it has the characteristic of being an ensemble of extremely simple and interpretable models: decision trees. The purpose of the Forest Structure Analysis is therefore to observe how decision trees that make up the forest are built, observing and cataloging their internal structures. Information gathered during this phase are fundamental to move on with the second phase of the procedure: the extraction of a GAM working as an interpreter of the forest of decision tree we are considering.

Forest analysis structure is divided into three steps:

1. Information gathering;
2. Estimate the most important features and identify possible interactions between features;
3. Estimate the domain of the features and identify subdomains that are particularly relevant in the decision making process.

Below, the three steps are explained in detail. In particular, for each step, we describe how the experiment is conducted and show the results obtained by applying the procedure to the two synthetic dataset described in the previous section.

4.3.1 Information gathering

The purpose of the first step of the forest structure analysis is to observe internal structure of decision trees composing the forest to gather information necessary to estimate the importance of the features, their possible interactions and their domain.
As we saw in previous chapter, in section 2.1, nodes inside a decision tree are divided into two categories: leaves and internal nodes. Leaves contain the final information, that is the *response*, while internal nodes, through their *tests*, define the path to reach the leaves. Tests contained in the internal nodes can be represented as pairs $< i, v >$, where i represents the index of the feature being tested and v represents the threshold. If the value of feature x_i exceeds the value v, the decision path continues along the right branch of the tree, otherwise it follows the left branch.

We can therefore say that leaves contain information about the response, while internal nodes contain information about features. Since this step of the forest structure analysis focuses on gather information about features, we focus only on internal nodes, excluding leaves.

Operationally, the information gathering step is configured as an exploration of the tree, enumerating all possible decision paths that connect the root of tree to all its leaves. Along the way, all test pairs $< i, v >$ are collected and some elaborations are produced:

- *feature dictionary:* for every feature, all the values v are collected and sorted;
- *same-path features:* how many times a pair of features appears along the same decision path;
- *adjacent features:* how many times each element of a pair of features appears one after the other along a decision path.

Example below shows the result obtained when gathering information on a twelve-leaf tree trained with the `lightgbm` library. In fig. 4.3 the obtained tree is shown: the tests, expressed in the form $x_i \leq v$, are reported in the internal nodes.
The following three tables show the results obtained. Table 4.3 shows the *features dictionary*, that is the list of values v that appear, for each feature, in the internal nodes of the tree. Table 4.4 shows the *same-path features* measure: for each pair of features, the number of decision paths in which it appears is indicated. Table 4.5 shows the *adjacent features* measure: for each pair of features, is indicated the number of times in which the terms of a pair of features appear one after the other within a decision path.
In next sections we see how this information is used.

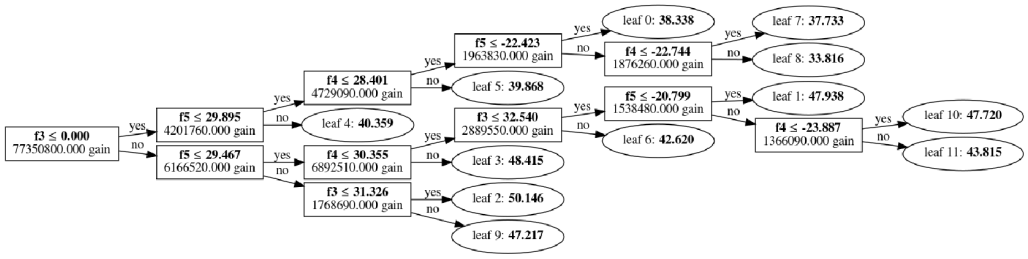

Figure 4.3: Example tree with twelve leaves, trained with the `lighgbm` library.

Feature dictionary

3	0	31.326	32.54	
4	-23.887	-22.744	28.401	30.355
5	-22.423	-20.799	29.467	29.895

Table 4.3: *feature-dictionary*: list of values v that appear, for each feature, in the internal nodes of the tree.

Same-path features

	3	**4**	**5**
3	0	9	12
4	0	0	9
5	0	0	0

Table 4.4: *same-path features*: the number of decision paths in which a pair of features appears.

Adjacent features

	3	**4**	**5**
3	0	1	4
4	0	0	5
5	0	0	0

Table 4.5: *adjacent features*: number of times in which terms of a pair of features appear one after the other within a decision path

4.3.2 Feature importance and features interaction

Second step of the forest structure analysis deals with estimating which are the most important features and the possible interactions between them within the decision making process of the forest.
At the basis of this step there is the notion described in section 2.1, where we describe how tests contained in the internal nodes of the trees influence the response that is produced at the end of decision making process. Below is a brief description of the concept.

Let's suppose that a tree contains a test on an attribute x_0, for example $x_0 \leq \xi_1$. Then, decision region is split into two parts and response of tree considers at most two values:

$$y = \begin{cases} v_1, & if\ x_0 \leq \xi_1 \\ v_2, & if\ x_0 > \xi_1 \end{cases}$$

This happens recursively for all the tests, for every tree in forest, segmenting decision region in a regular and precise pattern.

This theoretical approach suggests that the more often a feature appears as the subject of a tests, the more it is responsible for fragmentation of decision space and, therefore, the more relevant in determining outcome of the forest.
Therefore, to get an estimate of feature importance, we decided to count how many times a feature appears as the subject of a test. In particular, using information obtained from previous step, we can count how many values are associated with the feature in the *features dictionary*.
In fig. 4.4 are reported information about the two synthetic dataset, described in 4.2.1. Graph on the left is about the first synthetic dataset, in which first and third terms grow super-exponentially, unlike other terms which have less impact on final response of the model. In the graph we

can see that features considered most important are precisely the first and the third. Graph on the right is about the second synthetic dataset, which considers an interaction between the fourth and the fifth term. This dataset does not present terms that are clearly more important than the others and, in fact, graph does not show the preponderance of some features over others, unlike the graph relating to the first synthetic dataset.

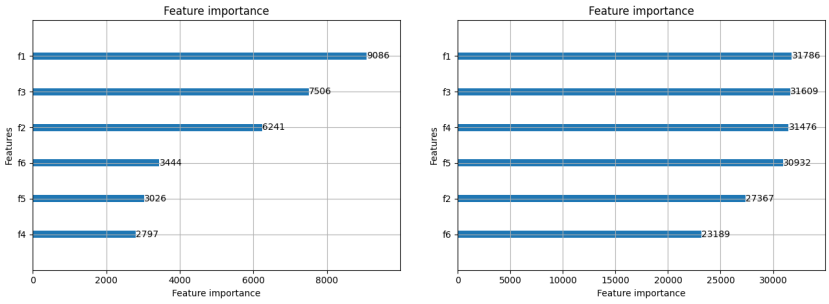

Figure 4.4: Feature importance calculated for the two synthetic datasets: on the left, the graph related to the dataset that does not include interactions; on the right the graph related to the dataset which includes an interaction between x_3 (f4) and x_4 (f5).

However, often forest outcome is the result of a collaboration of several features. It is possible to train GAM to grasp also the behavior of a function of two features at the same time, as indicated in [8] and reported in the section 3.1.1.
The simplest approach to identify pairs of interacting features is *brute forcing*: perturb behavior of all possible pairs of features to identify which ones have a more significant impact on model final outcome. Clearly this approach is highly time consuming, especially if dataset has a large number of features. Therefore we propose an heuristic to identify which pairs of features are most likely to interact. The heuristic is based on what is reported in [2]: *"If two variables X_1 and X_2 are used in different trees, but never in the same tree, then they do not interact: X_1 can only affect the prediction through the trees it occurs in, and since X_2 does not occur in these trees, the value of X_2 cannot possibly affect the effect of X_1, and vice versa."*.

This is the reason why - during the *information gathering* step - we count how many times a pair of features appears within the same decision path (*same-path features*) and how many times the terms of a pair of features appear one following the other (*adjacent features*). In table 4.6 we can see the *same-path features* information obtained from the forest trained using the synthetic dataset considering interactions and in table 4.7, we have the *adjacent features* information.

We know that dataset we used to train the forest is generated from a model containing an interaction between features x_3 and x_4. Looking at the results shown in tables 4.6 and 4.7, the score obtained by the interaction is highlighted in bold. In the table showing the results of the *same-path features* measure, we see that the result obtained for the interaction between x_3 and x_4 is not the highest, there are other pairs of features - highlighted in red - that have an higher score, such as (x_0, x_2), or (x_2, x_3), or (x_2, x_4). Instead, using the *adjacent features* measure, the interaction between x_3 and x_4 is correctly detected: it is the one with the highest score.
In light of the results obtained, we decide to discard the *same-path features* measure and use only the *adjacent features* measure to identify interactions between features.

	0	**1**	**2**	**3**	**4**	**5**
0	0	67540	78911	73553	73240	59361
1	0	0	70501	65303	65035	55525
2	0	0	0	80120	78820	58512
3	0	0	0	0	**78202**	57113
4	0	0	0	0	0	55489
5	0	0	0	0	0	0

Table 4.6: [*same-path features*] Interaction *scores* obtained counting how many times a pair of features appears in the same decision path, no matter how far, considering all trees in forest.

	0	**1**	**2**	**3**	**4**	**5**
0	0	7564	9269	8501	8520	6565
1	0	0	7943	7383	7231	6141
2	0	0	0	9602	9583	6139
3	0	0	0	0	**10083**	6388
4	0	0	0	0	0	6002
5	0	0	0	0	0	0

Table 4.7: [*adjacent features*] Interaction *scores* obtained counting how often terms in the feature pair appear in the same decision path one after the other, considering all trees in forest.

Boosting algorithm particular behaviour. The `lightgbm` library, which we used to train the forest, uses the boosting algorithm. As we see in section 2.2.2, boosting builds the trees in a sequential way and each tree is grown using information from previously grown trees. The first trees of the forest contain the most important information about the original dataset, while the last trees - obtained on the basis of the residuals of previous trees - gradually acquire less importance. For this reason, in the case of boosting algorithms, we choose to consider only a subset of trees - the most important ones - to identify the interactions between features. In this way, in addition to optimizing the algorithm, we obtain a score less influenced by the "noise" of the last trees. We calculate *same-path features* and *adjacent features* measures by considering only the first 10% of the trees in the forest. Results obtained are shown in table 4.8. Now, the score of the interaction between x_3 and x_4 is much higher than other possible interaction thus we can conclude that - if we are using the boosting algoritm - if we consider only the subset of the most important tree, we obtain a more precise result. Consequently, from now on, when we use the heuristic to identify possible interactions, we assume that it is always the score obtained from *adjacent features* information, applied to

only 10% of the trees in the forest.

	0	**1**	**2**	**3**	**4**	**5**
0	0	641	1015	964	978	336
1	0	0	687	528	540	293
2	0	0	0	995	936	292
3	0	0	0	0	**1639**	284
4	0	0	0	0	0	217
5	0	0	0	0	0	0

Table 4.8: [*boosting*] Interaction *scores* obtained counting how often terms in the feature pair appear in the same decision path one after the other *(adjacent features)* , considering 10% of the trees in forest.

4.3.3 Features domain

Third step of the forest structure analysis deals with estimating features domain and studies values present in*features dictionary* to identify parts of the domain that are particularly relevant during the decision making process.
This step is fundamental for the construction of datasets necessary to train GAMs.
To build datasets, it is obviously necessary to identify which values different features can assume, however this may not be sufficient. In fact, as we mention in section 4.2.3, the `pyGAM` library used to train GAMs has some performance problems that make it infeasible to train models when the dataset is too large, so we cannot create a dataset that combines all the possible values of all the features to reconstruct the decision space described by the forest: resulting dataset resulting would be too large.
For this reason, we introduce an analysis of the distribution of values assumed by different features: identifying the most relevant parts of domain in decision making process allows the creation of smaller datasets, but able to grasp essence of forest behavior, so that resulting GAM can take full advantage of it.

Starting point for this step is *features dictionary*, i.e. the collection, for each feature, of values assumed as *thresholds* in tests contained in the internal nodes of trees, ordered from the lowest to the highest.
The idea is to study the distribution of values for each feature to understand if any clues about the behavior of the forest can emerge from it.
Since models from which we extract the synthetic datasets are known, we are able to exactly represent the relationship between each feature x_i and the re-

sponse y. For example, if we consider the first artificial dataset described in 4.2.1, we know that the relationship existing between feature x_2 and the response y is expressed according to the formula $f(y|x_2) = \sqrt[3]{(x_2)^4}$. Since this relationship is a one-dimensional function, we can represent it with a graph and compare its trend with the plot of values corresponding to feature in the *features dictionary*, and with the histogram of threshold values frquency, as we can see in fig. 4.5.

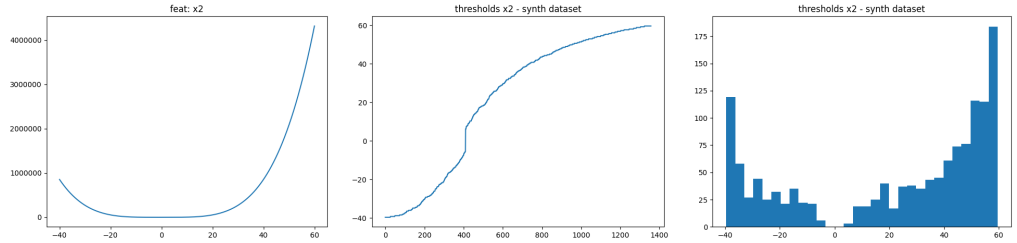

Figure 4.5: On the left, the plot of the function $f(y|x_2)$. Center image shows the sorted threshold values concerning tests on feature x_2, extracted from the forest. On the right, histogram regarding threshold values frequency.

Graph on the left shows graph of $f(y|x_2) = \sqrt[3]{x_2^4}$; graph in the middle shows values contained in *features dictionary* for feature x_2 and graph on the right shows thresholds values frequency. In particular, regarding middle graph, in the x-axes we have a progressive number, while on the y-axes we can find threshold values. The first thing we notice is that values of x_2 have the same domain in the first and in the last picture, meaning that threshold values span through all possible values assumed by the original model. We therefore have confirmation that distribution of values associated with a feature in *features dictionary* contains information about original domain of the feature in the original dataset.
Moreover, we can see that, in the first image, function firstly decreases, then for $-20 < x_2 < 20$ function remains stable and, in the final part, function increases its value. Similarly, the plot in the middle has values for all increasing/decreasing parts of the function, while there is a jump in correspondence to stationary interval. The histogram confirms that there is an higher frequency of thresholds where original function shows a non-stationary behaviour. These evidences suggest that forest is able to identify relationships between individual features and the response and concentrates a greater number of tests where changes in the feature influence the response the most.

However, due to the computational complexity of GAMs, we need to keep

datasets size under control, making it necessary to extrapolate from the feature domain those points where forest focuses its attention during decision making process. Using these points, we can obtain smaller datasets that can transmit to GAMs the most important information about forest decision power, then it will be GAM's task to approximate the parts of the domain where information is lacking.
Now let's see some ways to get samples from the feature domain. Some sampling methods take into account only the domain extension, others also consider values frequency of the features. In the next step of the procedure, when training GAMs, we will be able to evaluate which of the sampling methods works better.

We denote by N the parameter that defines the number of samples per features to be obtained when applying the different sample methods.

Method 0 - *all*: all thresholds.

Method 1 - *equi_width*: sampling the domain in regular intervals. We extract N values distributed at regular intervals in the range between maximum and minimum value assumed by feature. This is a very simple method that takes into account only estimated domain of features.

Method 2 - *k-means*: Sampling via clustering using `k-means` algorithm. We propose this method in order to exploit information given by the relation between y and the feature we are sampling. In fact, we know that a lot of threshold values are concentrated where y is more influenced by the feature. When clustering threshold values using `k-means`, we obtain k centroids which are representative of k groups of highly similar thresholds, with $|k| = N$. Then we can use centroids as samples to build the dataset.

Method 3 - *equi_size*: Sampling by getting the mean of set of fixed number of thresholds. The k-means algorithm incorporates the most similar elements into a cluster, determining sets that contain an arbitrary number of points. In our case this could be a disadvantage since we have seen that a large number of thresholds with similar values represent a trait of high variability in the relationship between the feature and the response. Representing a cloud of very close points with a single sample could make us lose information on the fragmentation of the decision space.
For this reason we tried a different approach: we create clusters by dividing

threshold values into N groups, each containing the same number of elements, and we calculate their centroid. In this way, a large set of thresholds with very similar values can produce several samples, while using k-means we would have obtained only one sample.

In order to see the effects of the three methods, we apply them to the *feature dictionary* obtained from a forest trained with the first synthetic dataset explained in 4.2.1.
In figure 4.6 we can see how the three sampling methods work when applied to the relation between y and x_2. The black points are the samples obtained, while the red points are the projection of the samples on the function representing the relation between y and x_2.
We can see that the first method (*equi_width*) has many points distributed in a stationary part of the function, while the more variable part of the function is described by a few samples.
Second method (*k-means*) extracts less points from the stationary part of the function, while distributes more samples where function is non-stationary. However, comparing the behavior of the second and the third method, we can see that the third method has a major concentration of samples when function increases more its value.
At first sight, third method (*equi_size*) seems to be the one performing better. In the next section (sect. 4.4) we will see the application of all three sampling methods on forests trained with four different datasets in order to understand the real efficiency of the proposed methods on real data.

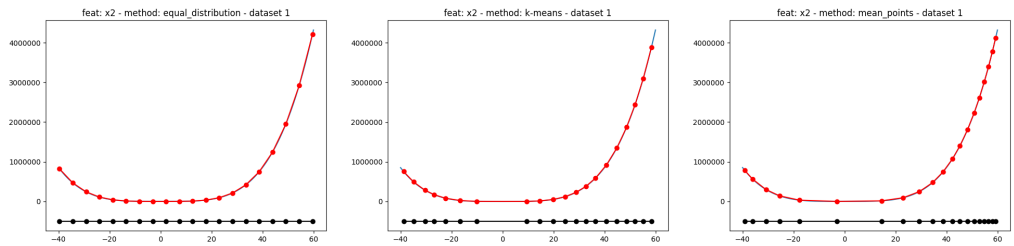

Figure 4.6: Three sample methods compared: in the first picture we show the behavior of the first method (*equi_width*), in the second picture we show the behavior of the second method (*k-means*) and in the third picture we show the behavior of the third method (*equi_size*).

4.4 GAM extraction

The second and last phase of procedure aims to use information obtained from f*orest structure analysis* to train the final model: a GAM that works as an interpreter of a forest of decision trees.
Let's summarize the information we get from the forest structure analysis:

- a score that allows us to sort the features from the most important to the least important;
- a score that allows us to identify possible interactions between the features;
- a sampling of the features domain.

Information regarding the importance of features and the possibility of their interaction are exploited to identify *terms* of the GAM that we need to train; sampling of the domain of the features is used to compose dataset.
Below, we describe how we proceed operationally to identify the terms of the GAM and to compose the dataset. At the end, we describe how we train the GAM and show the result obtained using the two synthetic dataset described in 4.2.1.

4.4.1 *Terms* identification

Theoretically, a GAM can have as many terms as we want, but practically, the performance problems of the pyGAM library forces us to choose only a few, the most important ones.
In particular, we have found that the performance problem occurs when the library has to manage a GAM with more than 10 or 12 terms, which become fewer if some terms are actually interactions between features. Of course, the number of terms the library can handle is also subject to the size of the dataset. The limit of 10 or 12 terms that we found refers to a dataset with 50,000 entries, which is the size of the dataset we have chosen for this phase of the procedure.

In order to identify the most important features, we plot the score obtained by each feature and we select a subset of 6-10 features among those that have the highest score. For example, if the first 8 features have an index that decreases more or less constantly, but between the 8^{th} feature and 9^{th} feature there is a larger gap between the indices, the choice of the most important features will be limited to the first 8 features, excluding the others.

In the case of the synthetic datasets that we use during the description of this phase, the problem of choosing the number of terms does not arise, as both datasets have 6 features. However, we still apply the feature selection process as a toy example, to show a practical demonstration.

4.4.2 Dataset composition

To create the dataset, we initially thought of composing all the possible combinations of the samples extracted from the forest. In this way we would have been able to totally recreate the decision space of the forest and represent all the possible responses of the forest. Clearly, the size of this dataset is huge and the algorithm to extract the GAM would have failed.
Therefore, we decide to build the dataset by randomly extracting a value for each feature from the set of sampled thresholds to compose each record, up to a certain number of records. In this way, we are able to create datasets that are more manageable in terms of space, but sufficiently exhaustive to represent the decision space defined by the forest.

```
# parameter n defines dataset size
def createDataset(sampled_features, n):
  dataset = []
  for feature_values in sampled_features:
    # randomly extract n values for that feature
    idx = np.random.choice(feature_values.shape[0], n, replace=True)
    feature = feature_values[idx]
    dataset.append(feature)

  dataset = np.array(dataset)
  dataset = dataset.T.reshape(-1, dataset.shape[0])
  return dataset
```

4.4.3 GAM training

Up to now we address the problems of identifying the most important features, identifying the possible interactions between them and building the datasets to train the GAM.
But still there are some open points:

- check if identified interaction terms bring improvements to the model;

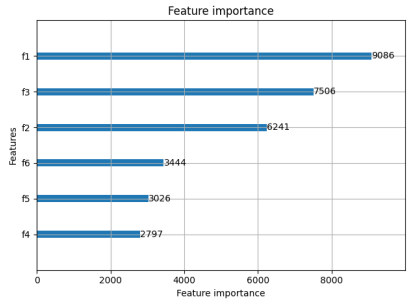

Figure 4.7: Feature importance calculated for the first synthetic dataset.

In figure 4.7, the score of the first three features drops steadily, while between feature f2 and feature f6 there is a greater jump. Last three features have more or less the same value. According to the procedure, the jump between the third and the fourth feature determines the division between the most important and less important features.

In figure 4.8, the first four features have approximately the same value, while, starting from the fifth feature, the value begins to decrease. The procedure chooses the first four features as terms and discards the last two, as they are less important.

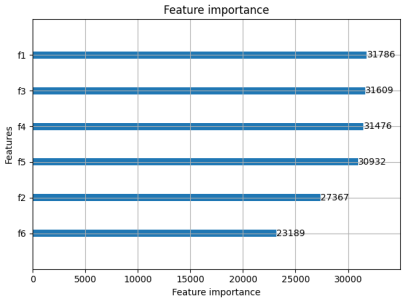

Figure 4.8: Feature importance calculated for the second synthetic dataset.

	0	**1**	**2**	**3**	**4**	**5**
0	0	641	1015	964	978	336
1	0	0	687	528	540	293
2	0	0	0	995	936	292
3	0	0	0	0	**1639**	284
4	0	0	0	0	0	217
5	0	0	0	0	0	0

Table 4.9: Interaction *scores* obtained counting how often terms in the feature pair appear in the same decision path one after the other, considering 10% of the trees in forest.

To identify feature interaction, we observe the *adjacent feature* information obtained during the forest structure analysis: possible interactions are pairs of features that have a much higher score than the others. To verify if pair of features really interact, however, it is necessary to compare the GAM with the interaction term with the GAM without the interaction term: only if accuracy of the first model improves, then the interaction is confirmed.

- identify correct value of N, the number of sample obtained per features, during the sampling process of the feature values.

To solve these open points we sample the values of the features using different values of N, the sample size. In particular, for each method, we sample the features with five different values of N: 20, 40, 60, 80, 100. We build a different dataset for each method and sample size N. In total, the built datasets are 16: for each one of the three methods (*equi_witdh*, *k-means* and *equi_size*) we extract 5 different datasets, one for each N; plus a dataset that uses method *all*, which considers all the possible values of the features and does not require a sample size. Then, with every dataset, we trained some GAMs and we compare them to find out the one with the best RMSE. At this point, we train another set of GAMs, this time considering possible features interaction. We select the model performing better and we compare it with the best model not considering interaction: the best model between those two is selected as forest interpreter.

Now let's apply the procedure just described to the second synthetic dataset described in 4.2.1, the one that models an interaction between features x_3 and x_4, and let's see how it works.
First of all, we train the GAMs that do not consider feature interactions. Figure 4.9 shows the plot of the value of the RMSE regarding GAMs trained with different dataset. RMSE value is obtained by comparing the predictions of the GAM with the predictions of the forest. Each of the four line plots refers to a different sampling method. Looking at the graph, we see that GAMs trained with dataset obtained by sampling methods *equi_size* and *all* perform better than those obtained with sampling methods *equi_width* and *k-means*. However, from the graph it is difficult to understand which sampling method produces the best GAM.
GAM obatined with the *all* method has $RMSE = 10.215$; while GAM obtained with *equi_size* method and $N = 60$ has $RMSE = 10.182$. We can therefore conclude that, among the models that do not consider interactions between features, the one trained with the dataset obtained via the sampling method *equi_size* is the model that imitates better the original forest.
Then, using the same procedure, we train GAMs having a *term* that considers the interaction between feature x_3 and feature x_4. Results are shown in figure 4.10. This time, sampling methods producing the best model are *equi_width* and *k-means*. In particular, the best model obtained using the *equi_width* method is the one with $N = 20$ as sample size, $RSME = 2.288$; while the best model using *k-means* method is the one with $N = 20$, and $RMSE = 2.95$.

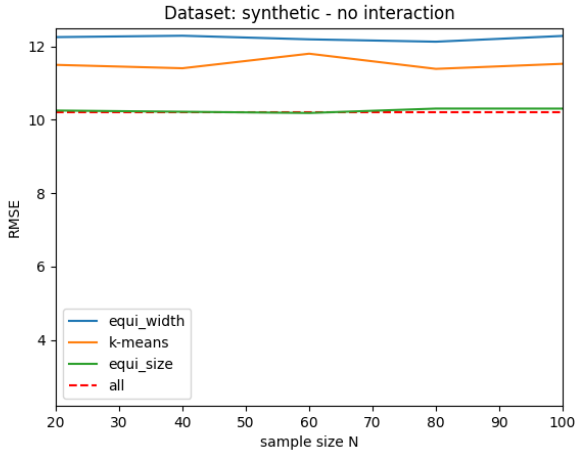

Figure 4.9: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `synthetic` All the models considers no interactions between features.

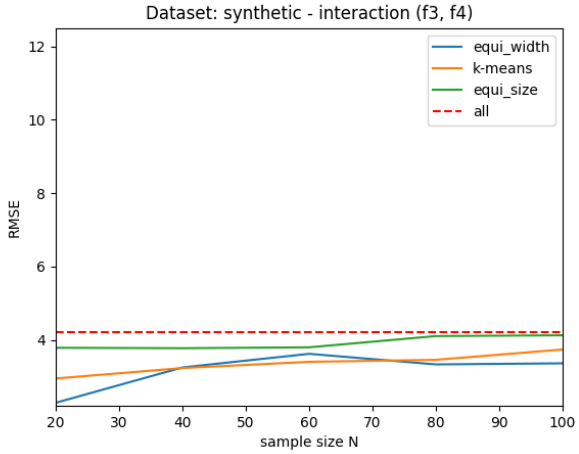

Figure 4.10: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `synthetic` All the models considers an interactions between features x_3 and x_4.

Now, let's compare the best two models obtained, one considering an interaction between features x_3 and x_4, the other considering no interaction at all. In figure 4.11 we can see that GAM actually considering the interaction between features x_3 and x_4 works much better. We already know that those two features interacts, thus this result is an evidence that the method we use to identify possible features interactions seems to correctly identify interactions between pair of features.

In figure 4.12 we can see the plot of the five *shape functions* composing the best GAM obtained from the forest trained with the `data_inter` synthetic dataset. In figure 4.13 we can see the plot of the original functions generating the `data_inter` synthetic dataset. Plot obtained from the GAM are quite similar to original one.

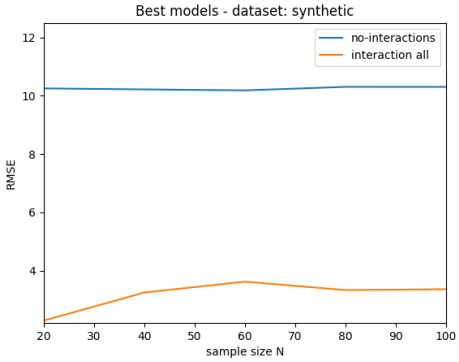

Figure 4.11: Comparison between best GAM considering an internaction between features x_3 and x_4 and best GAM not considering interaction to test whether interaction improve model or not.

In the next sections we apply the procedure to the three datasets described in section 4.2.1 to verify the behavior of the procedure on real data, of which we do not know the model generating them and that contain noise.

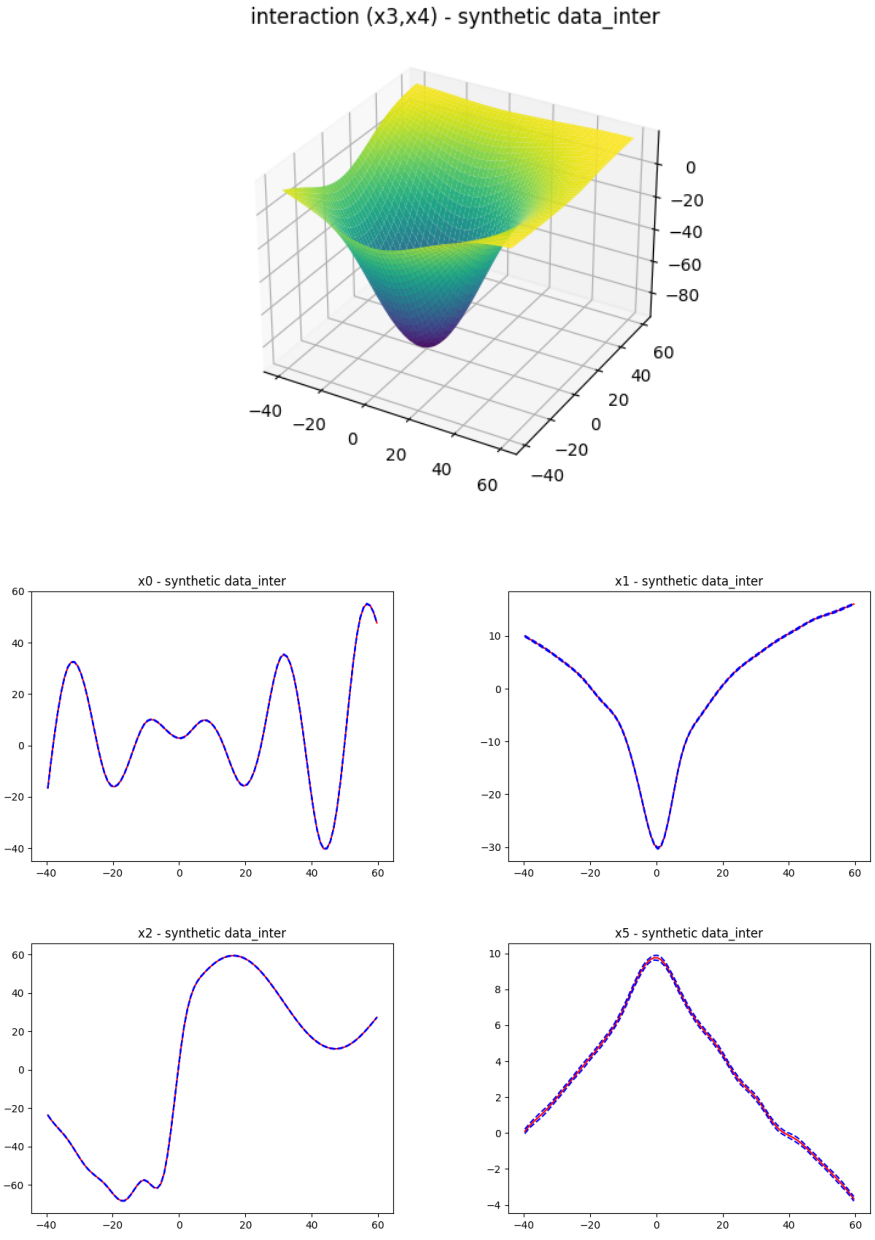

Figure 4.12: *Shape functions* of the best GAM obtained from the forest trained with the `data_inter` dataset: GAM is trained with a dataset built by sampling 20 values from thresholds using the *equi_width* method; it considers an interaction between features x_3 and x_4. Blue lines represent the 95% confidence interval for the estimated function.

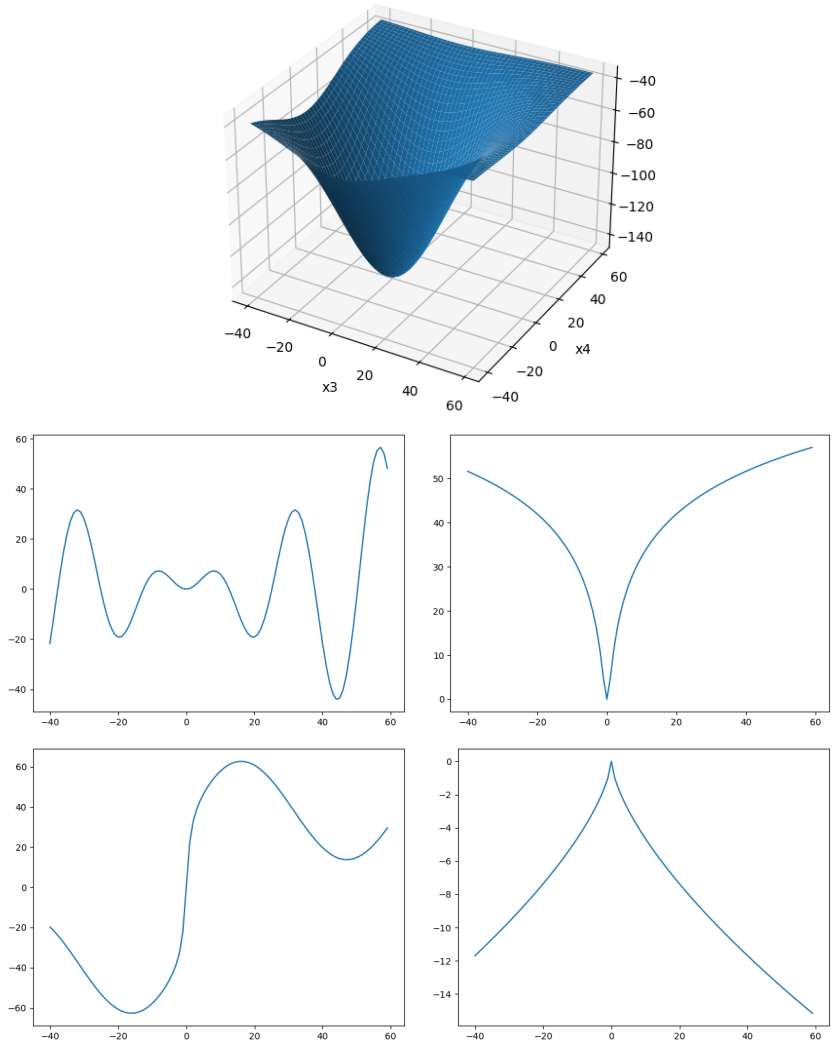

Figure 4.13: Functions of the model generating the artificial dataset `data_inter`.

4.5 Case study: concrete dataset

Concrete dataset contains information about the concrete compressive strength, which is an highly nonlinear function of age and ingredients.
Dataset contains 1030 instances and every instance contains 9 quantitative variables: 8 features and one response. There are no missing values. In table 4.10 we can find attribute description.

Nr.	Name	Data Type	Description
1	Cement	numerical	Input Variable
2	Blast Furnace Slag	numerical	Input Variable
3	Fly Ash	numerical	Input Variable
4	Water	numerical	Input Variable
5	Superplasticizer	numerical	Input Variable
6	Coarse Aggregate	numerical	Input Variable
7	Fine Aggregate	numerical	Input Variable
8	Age	numerical	Input Variable
–	Concrete compressive strength	numerical	Output Variable

Table 4.10: Features of concrete dataset.

We use 90% of the dataset as training set and 10% as testing set and we use the training set to train a forest of decision trees using the lightgbm library as described in 4.2.3. In the end, we obtain a forest having 331 trees and $RMSE = 3.91$.

4.5.1 Forest Structure Analysis

First of all, we collect all the pairs $< features, thresholds >$ from tests contained in the internal nodes of all the trees in forest, then we compute the *feature dictionary* and the two interaction measures: *same-path features* and *adjacent features*.

Feature importance In figure 4.14 we can see the score obtained for each feature. Features 1 (cement), 4 (water) and 6 (coarse aggregate) have the highest score: the forest gives a lot of importance to these features during the decision process. Features 2 (blast furnace slag) and 3 (fly ash), on the other hand, have a very low score, meaning that they are not frequently tested by the forest and have less impact on response prediction.

Features interaction In table 4.11 we can see the *adjacent features* information. Interaction between features 1 (cement) and 8 (fine aggregate) has

the highest score. We test whether features 1 and 8 actually interact or not when training the GAM.

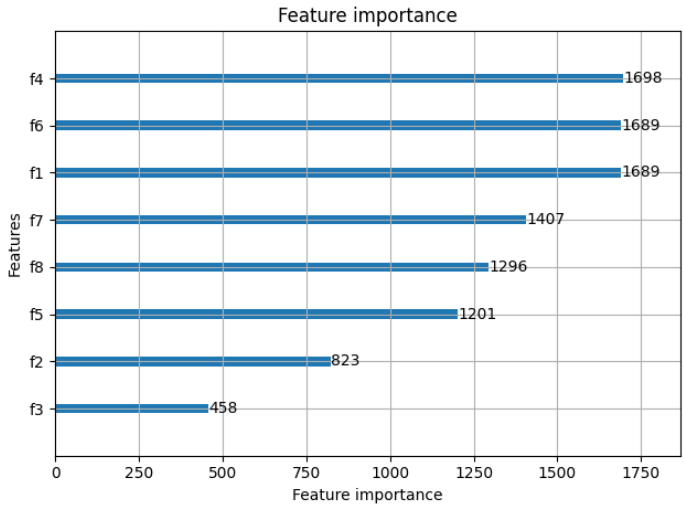

Figure 4.14: Importance of features of dataset `concrete`.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	82	15	78	46	31	25	**107**
2	0	0	15	73	28	26	13	33
3	0	0	0	5	14	4	8	8
4	0	0	0	0	15	26	52	46
5	0	0	0	0	0	15	31	51
6	0	0	0	0	0	0	33	19
7	0	0	0	0	0	0	0	32
8	0	0	0	0	0	0	0	0

Table 4.11: *Adjacent feature* information obtained from the forest structure analysis regarding `concrete` dataset.

4.5.2 GAM extraction

Using the different sampling methods seen in section 4.3.3, we sample thresholds contained in *features dictionary*. As sample size N we use different values: 20, 40, 60, 80, 100. We create different dataset as described in section 4.4.2 and train GAMs as described in section 4.4.3.

***Terms* identification** Dataset has 8 features, thus it is not necessary to choose which features to use as *terms* when training the GAM. Regarding the interaction term, *adjacent features* information shows that there is a possible interaction between features 1 and 8.

No interactions In figure 4.15 we see the RMSE values obtained from the different GAMs obtained without considering interactions between features. Methods *equi_width*, *equi_size* and *all* have more or less the same performances, while method *k-means* outperforms other methods, improving its results when parameter N grows. The latter observation is interesting when compared with the result obtained by the *all* method: in fact, looking at the results obtained using *k-means* as sampling method one can think that the bigger the sample size N, the better the model; however, results obtained using all the thresholds are much worst, demonstrating that sampling threshold values is important in order to obtain GAMs that better model the relationship between individual features and the response.

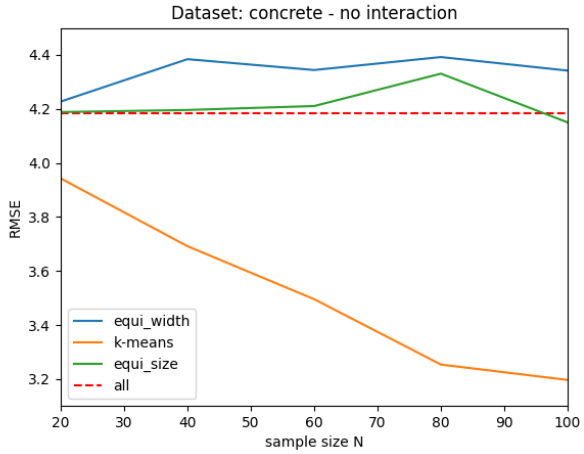

Figure 4.15: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `concrete`. All the models considers no interactions between features.

GAMs with interactions Figure 4.16 shows results of GAMs obtained with a term considering an interaction between features 1 (cement) and 8 (age). Method *k-means* is the best sampling methods even when considering interactions between features.

Choose best GAM We compare the best model obtained when considering an interaction between features 1 and 8 and the best model considering no interactions at all. We can see results in figure 4.17: model considering interaction is a little better than the other. This result confirms that the concrete compressive strength is a function where cement (feature 1) and age (feature 8) actually interact.

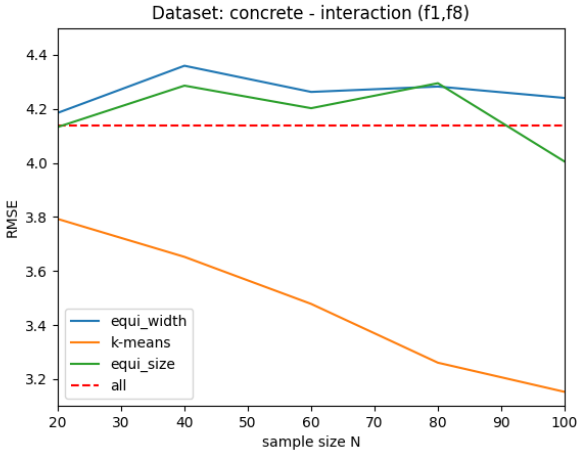

Figure 4.16: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `concrete`. All the models considers an interactions between cement (feature 1) and age (feature 8).

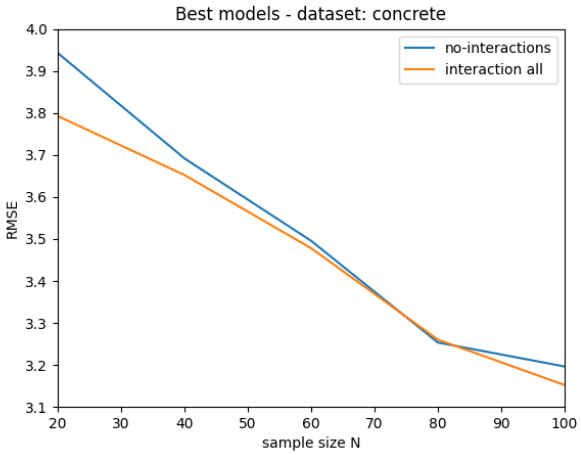

Figure 4.17: Comparison between best GAM considering an internaction between features 1 and 8 and best GAM not considering interaction to test whether interaction improve model or not.

Final GAM In figure 4.18 are represented the six shape functions composing the best GAM obtained from the forest trained with the `concrete` dataset.
First shape function concerns interaction between cement and age: the relationship shows that the more cement there is and the older the concrete is, the more the concrete compressive strength is high. Second shape function shows an inversely proportional relationship between water and concrete compressive strength, while the relationship between fine aggregate and response shows a more complex behavior. Fourth shape function shows that superplasticizer increases concrete compressive strength, as does the blast furnace slug of the fifth shape functions. Graph relating to last shape function, on the other hand, is less precise: we can see that the confidence interval (in blue) is wider and the trend of the function is quite stationary. This is

due to the fact that feature 3 is the least important, so we have fewer values to sample, as the forest did not concentrated many tests on this feature.

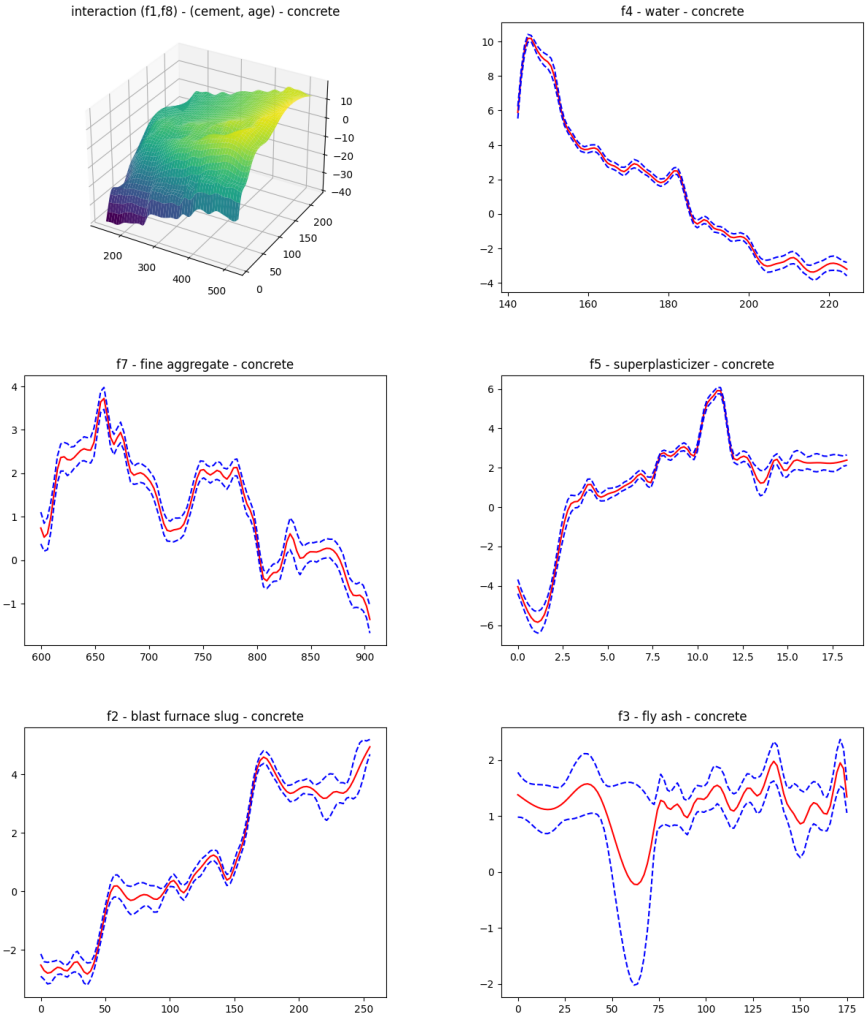

Figure 4.18: *Shape functions* of the best GAM obtained from the forest trained with the `concrete` dataset: GAM is trained with a dataset built by sampling 100 values from thresholds using the *k-means* method; it considers an interaction between features 1 and 8. Blue lines represent the 95% confidence interval for the estimated function.

4.6 Case study: `houses` dataset

`Houses` dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015.
Dataset contains 21.613 instances and every instance has 19 quantitative variables: 18 features and one response. Missing values have been removed from original dataset. In table 4.12 we can find attribute description.

Nr.	Name	Data Type	Description
1	date	numerical	Input Variable
2	bedrooms	numerical	Input Variable
3	bathrooms	numerical	Input Variable
4	sqft_living	numerical	Input Variable
5	sqft_lot	numerical	Input Variable
6	floors	numerical	Input Variable
7	waterfront	boolean	Input Variable
8	view	numerical	Input Variable
9	condition	numerical	Input Variable
10	grade	numerical	Input Variable
11	sqft_above	numerical	Input Variable
12	sqft_basement	numerical	Input Variable
13	yr_built	numerical	Input Variable
14	yr_renovated	numerical	Input Variable
15	lat	numerical	Input Variable
16	long	numerical	Input Variable
17	sqft_living15	numerical	Input Variable
18	sqft_lot15	numerical	Input Variable
–	price	numerical	Output Variable

Table 4.12: Features of `houses` dataset.

As we see in section 4.2.1, `houses` dataset response has a long right tail: there are a few houses sold at a very high price. Since the variability of those prices is very high, forest is not able to catch the underlying behaviour and its RMSE is affected. We therefore decide to apply the log function to house prices.

We use 90% of the dataset as training set and 10% as testing set. We use the training set to train a forest of decision trees using the `lightgbm` library as descrived in 4.2.3. We obtain a forest having 341 trees and $RMSE = 0.159$.

4.6.1 Forest Structure Analysis

First of all, we collect all the pairs $< features, thresholds >$ from tests contained in the internal nodes of all the trees in forest, then we compute

the *feature dictionary* and the two interaction measures: *same-path features* and *adjacent features*.

Feature importance In fig. 4.19 we can see the score obtained for each feature. Features 15 (lat) and 16 (long) have a score much higher than other features, which means that house geographical position has a large impact on house prices. Following important features regards house dimensions, building year and selling date. Less important features are f6 and f7, respectively the number of floors in the house and a boolean value recording whether it is possible to see the sea from the house or not.

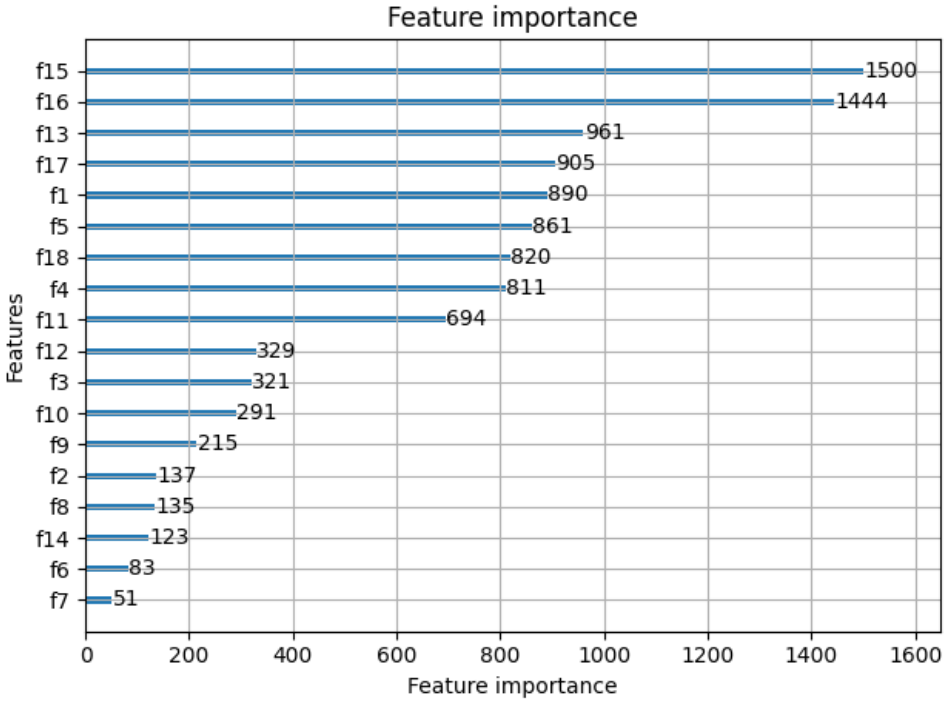

Figure 4.19: Importance of features of dataset `houses`.

Features interaction In table 4.13 we can see the *adjacent features* information. We can see that the highest score is reached by features f15 and f16, respectively latitude and longitude. Also features f4 (sqft_living) and f15 (lat) have an high interaction score. We test them all in the next section when training GAMs.

	f1	**f4**	**f5**	**f11**	**f13**	**f15**	**f16**	**f17**	**f18**
f1	0	3	8	6	5	7	5	5	0
f4	0	0	1	1	9	**91**	27	8	2
f5	0	0	0	2	5	43	19	6	2
f11	0	0	0	0	6	21	12	1	1
f13	0	0	0	0	0	21	7	9	4
f15	0	0	0	0	0	0	**134**	40	20
f16	0	0	0	0	0	0	0	21	3
f17	0	0	0	0	0	0	0	0	1
f18	0	0	0	0	0	0	0	0	0

Table 4.13: *Adjacent feature* information obtained from the forest structure analysis regarding `houses` dataset.

4.6.2 GAM extraction

Using the different sampling methods seen in section 4.3.3, we sample thresholds contained in *features dictionary*. As sample size N we use different values: 20, 40, 60, 80, 100, 150, 200. We create different dataset as described in section 4.4.2 and train GAM as described in section 4.4.3.

***Terms* identification** Dataset has 18 features, thus it is necessary to choose which features to use as *terms* when training the GAM. We take a look the first ten features of the histogram in fig. 4.19, i.e. until feature f11: excluding features f15 and f16, other features importance decreases constantly, while there is an higher jump between features f11 and f12, respectively the 9^{th} and the 10^{th} features. We therefore decide to use all the first 9 features in order of importance as *terms* of the GAMs.
Regarding interaction terms, we test first the interaction between features f15 and f16 and then we build another set of GAMs containing also the interaction between f4 and f15. In the end, we compare both set of GAMs with GAMs not considering interaction at all to determine whether features actually interacts or not.

No interactions In figure 4.20 we see the RMSE values obtained from the different GAMs obtained without considering interactions between features. Results obtained with methods *k-means* and *equi_size* are much better than those obtained with *equi_width* and *all*, even if *k-means* shows an extremely variable behavior. It is interesting because *k-means* and *equi_size* are both sampling methods that take into account also the distribution of threshold values along feature domain.

Moreover, sampling 80, 100 or 150 values works better than sampling 200 values and this underlines importance of sample size: a sample too big brings poor generalization and resulting final GAMs suffers from it.
In the end, we elect as best GAM with no interaction terms the one trained with the dataset obtained using the *equi_size* sampling method and sample size $N = 150$.

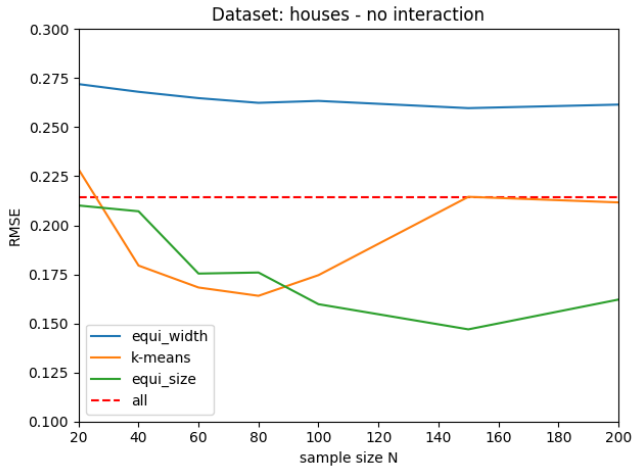

Figure 4.20: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `houses`. All the models considers no interactions between features.

Interaction between f15 and f16. Figure 4.21 shows results of GAMs obtained with a term considering an interaction between features f15 (latitude) and f16 (longitude). Model results are similar to those of the GAMs not considering interactions: best models are obtained when using methods *k-means* ($N = 80$) and *equi_size* ($N = 150$).
However, in this case, best model is the one obtained using the *k-means* sampling method and sample size $N = 80$.

All interactions Figure 4.22 shows results of GAMs obtained with a term considering an interaction between features f15 (latitude) and f16 (longitude) and another term considering interaction between feature f15 (latitude) and f4 (sqft_living). Model results are similar to those of the GAMs not considering interactions: best models are obtained are obtained when using methods *k-means* ($N = 80$) and *equi_size* ($N = 150$).
Also in this case we elect as best GAM considering both interaction term the one trained with the dataset obtained using the *k-means* sampling method and sample size $N = 80$.

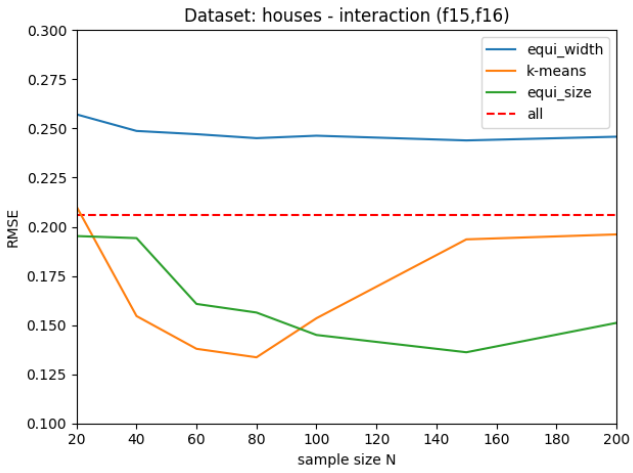

Figure 4.21: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `houses`. All the models considers no interactions between features.

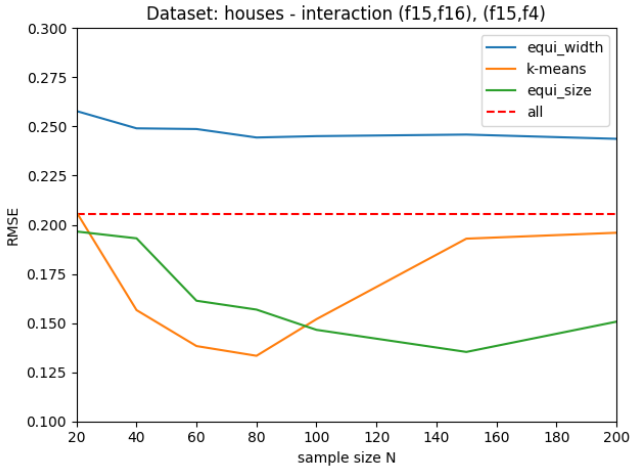

Figure 4.22: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `houses`. All the models considers no interactions between features.

Choose best GAM We compare the best model obtained for each one of the three cases analysed: best GAM considering no interaction at all, best GAM considering an interaction term between f15 and f16 and best GAM considering two interaction terms, one between f15 and f15 and one between f4 and f15. In figure 4.23 we can see the results: introducing an interaction term between features f15 and f16 brings an important improvement to the model. However, interaction between f15 and f4 seems not to be relevant.
We therefore decide to choose the GAM considering only the interaction between f15 and f16, trained with the dataset obtained with the *k-means* sampling method, $N = 80$.

Figure 4.23: Comparison between best GAM considering no interaction at all, best GAM considering an interaction term between f15 and f16 and best GAM considering two interaction terms, one between f15 and f15 and one between f4 and f15.

Final GAM In figures 4.24 and 4.25 are represented the 8 shape functions composing the best GAM obtained from the forest trained with the `houses` dataset.
First shape function concerns interaction between latitude and longitude: in a certain geographical area houses prices are higher. Shape functions of f4, f17, f5 and f11 are trivial: the bigger the house, the more it costs. Shape function of f13 is about year in which house have been built: obviously, houses built in more recent times have higher prices, while houses built before 1980 have a lower price, which peaks around 1915. What is surprising is price behaviour before 1960, in fact it shows an obscillating behaviour as house is older. This is probably due to the fact that some old houses may have already been renovated or may have some historical significance. From shape funtion of f1 seems that houses sold in 2015 has an higher price than houses sold in 2014. Shape function of f18 is kind of strange: it seems that house price have to be smaller as the value of f18 grows bigger. However, we can see that this feature 95% confidence interval (the blue line) is mich wider than other features. Probably feature 18 has a quite variable behavuoir that GAM was unable to represent effectively.

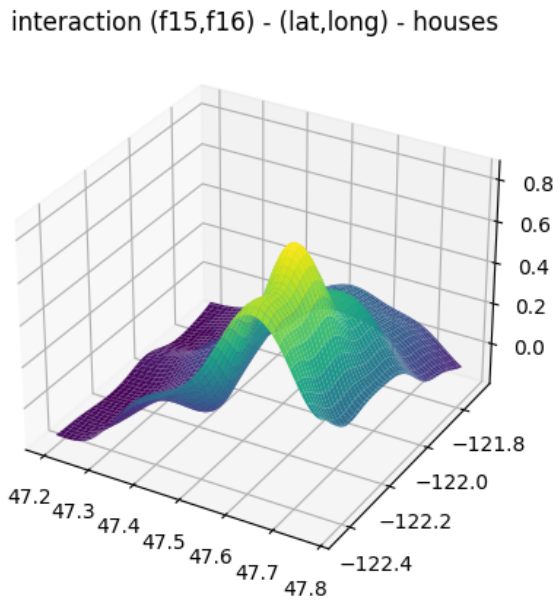

Figure 4.24: *Shape functions* of the best GAM obtained from the forest trained with the `houses` dataset: GAM is trained with a dataset built by sampling thresholds using the *k-means* method and 80 as sample size; it considers an interaction terms between features f15 and f16.

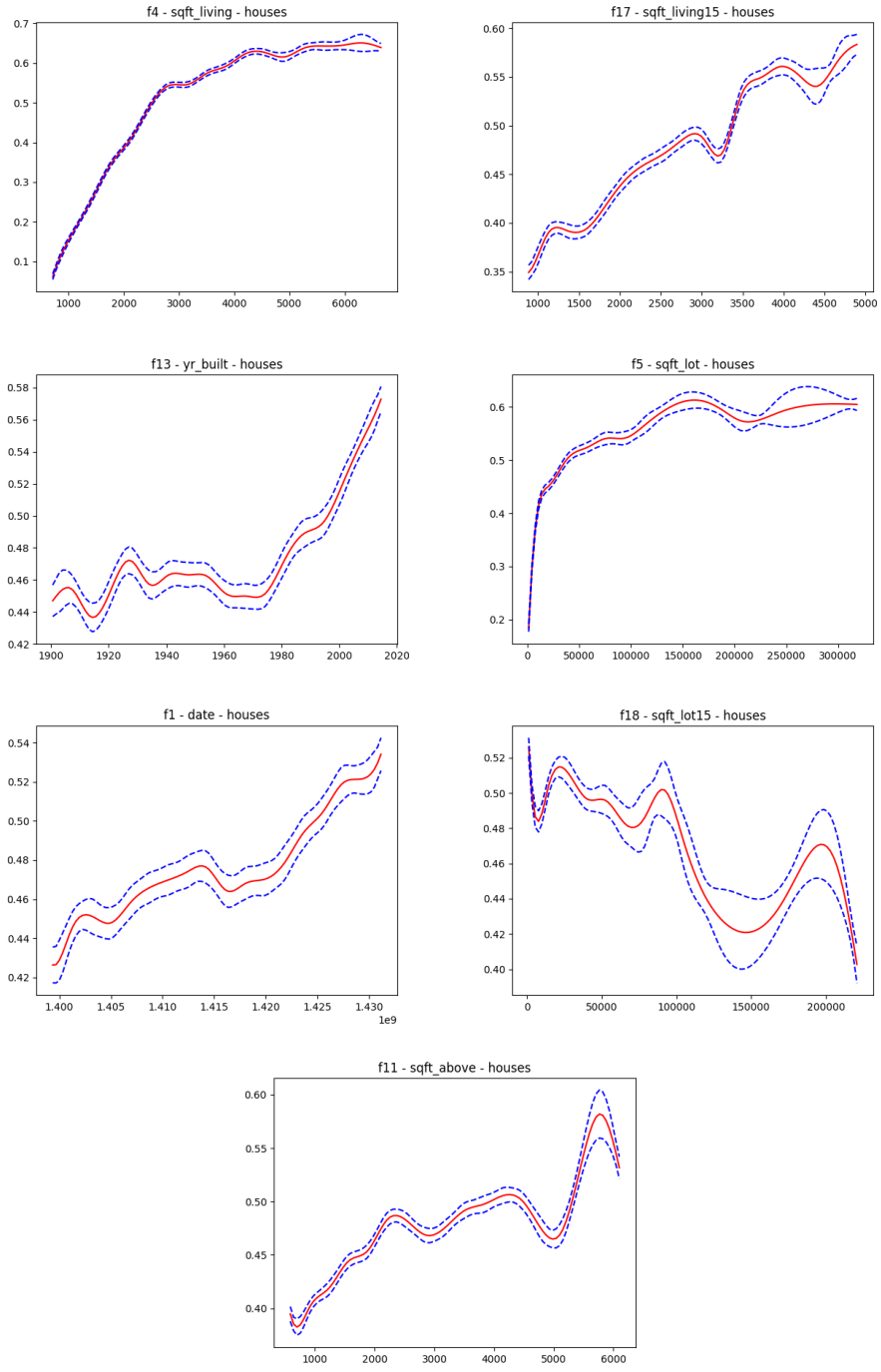

Figure 4.25: *Shape functions* of the best GAM obtained from the forest trained with the `houses` dataset: GAM is trained with a dataset built by sampling 80 values from thresholds using the *k-means* method; it considers an interaction term between features f15 and f16. Blue lines represent the 95% confidence interval for the estimated function.

4.7 Case study: `years` dataset

`Years` dataset is a subset of the Million Song Dataset. The aim it to predict the release year of a song from audio features. Songs are mostly western, commercial tracks ranging from 1922 to 2011, with a peak in the year 2000s. Dataset contains 515.345 instances and every instance has 91 numerical variables: 90 features and one response. All features are extracted from the 'timbre' features from The Echo Nest API: first 12 features are timbre average, last 72 features are timbre covariance. There are no missing values.

We use 412.275 instances as training set and 103.068 instances as testing set. We use the training set to train a forest of decision trees using the `lightgbm` library as described in 4.2.1. We obtain a forest having 4612 trees and $RMSE = 8.68$.

4.7.1 Forest Structure Analysis

First of all, we collect all the pairs $< features, thresholds >$ from tests contained in the internal nodes of all the trees in forest, then we compute the *feature dictionary* and the two interaction measures: *same-path features* and *adjacent features*.

Feature importance Since dataset has too many features, in fig. 4.26 we can see score obtained by the fifteen features with the highest score. In particular, most important features are f3, f6 and f1, all related to the timbre average.
We can also observe that 9 of the 10 most important features are actually related to timbre average, while features related to timbre covariance seems to have a less important impact on the final response.

Feature interactions In table 4.14 we can see the *adjacent features* information. We can see that the highest score is reached by features f1 and f2 and by features f1 and f3. In the next section, we train GAMs considering both possible interaction to test whether couple of features (f1, f2) and (f1, f3) actually interact.

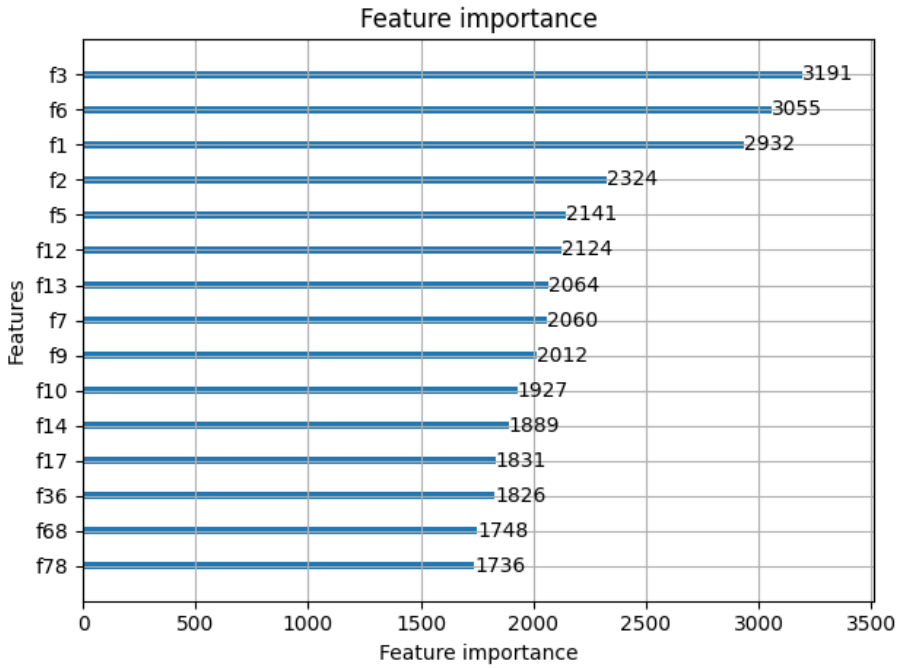

Figure 4.26: Importance of features of dataset `years`.

	f1	f2	f3	f5	f6	f7	f9	f10	f12	f13	f14	f17	f36	f68	f78
f1	0	**237**	**158**	18	29	14	9	10	12	41	43	4	11	7	11
f2	0	0	79	26	33	9	12	20	7	31	16	4	8	4	20
f3	0	0	0	40	29	42	20	15	23	20	32	7	22	18	14
f5	0	0	0	0	30	29	10	8	9	4	8	2	12	8	8
f6	0	0	0	0	0	9	19	23	16	16	8	10	19	11	6
f7	0	0	0	0	0	0	14	19	8	9	8	5	10	4	7
f9	0	0	0	0	0	0	0	13	5	3	4	3	1	5	2
f10	0	0	0	0	0	0	0	0	14	12	1	4	3	8	4
f12	0	0	0	0	0	0	0	0	0	6	3	1	4	2	5
f13	0	0	0	0	0	0	0	0	0	0	6	4	20	2	17
f14	0	0	0	0	0	0	0	0	0	0	0	0	6	5	9
f17	0	0	0	0	0	0	0	0	0	0	0	0	1	2	5
f36	0	0	0	0	0	0	0	0	0	0	0	0	0	9	7
f68	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
f78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.14: *Adjacent feature* information obtained from the forest structure analysis regarding `years` dataset.

4.7.2 GAM extraction

Using the different sampling methods seen in section 4.3.3, we sample thresholds contained in *features dictionary*. As sample size N we use different values: 20, 40, 60, 80, 100, 150, 200. We create different dataset as described in section 4.4.2 and train GAM as described in section 4.4.3.

***Terms* identification** Dataset has 90 features, thus it is necessary to choose which features to use as *terms* when training the GAM. We take a look at the first ten features of the histogram in fig. 4.26, i.e. until feature f10: excluding features f3, f6 and f1, other features decrease their importance in a constant trend. Highest jumps are between features f2 and f5 and between f9 and f10. We thus decide to consider features until feature f9, the 9^{th} feature. Regarding interaction terms, we test first the interaction between features f1 and f2 and then we build another set of GAMs containing also the interaction between f1 and f3. In the end, we compare both set of GAMs with GAMs not considering interaction at all to determine whether features actually interacts or not.

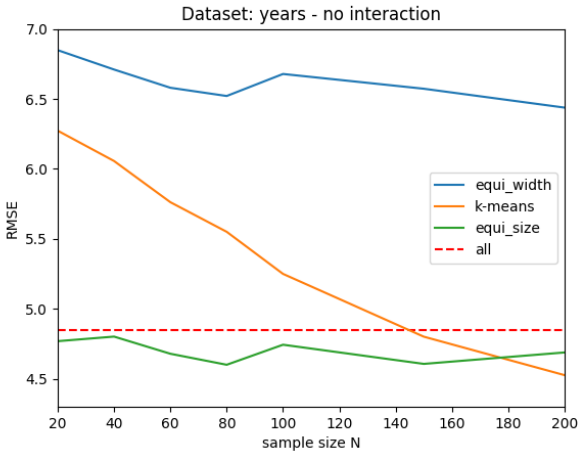

Figure 4.27: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `years`. All the models considers no interaction between features.

No interactions In figure 4.27 we see the RMSE values obtained by the different GAMs not considering interactions between features.
Models obtained with method *equi_width* have the worst results no matter the sample size N we are considering.
Models obtained with method *equi_size* performs better on average.
Models obtained with method *k-means* have results with an high variance, but we can see that the higher the sample size N, the better the results.

Models obtained with method *all* reach good results, but models obtained with *k-means* or *equi_size* sampling methods behave better.
It is hard to determine which model is the one performing better between those trained with *k-means* and *equi_size*, so we take a look at RMSE values. Model obtained with *k-means* sampling methods and sample size $N = 200$ has $RMSE = 4.527$, while best model obtained with *equi_size* and $N = 80$ has $RMSE = 4.602$. We can then conclude that the best model is the one obtained with *k-means* sampling method.

Interaction between f1 and f2. Figure 4.28 shows results of GAMs obtained with a term considering an interaction between features f1 and f2. Model results are similar to those of the GAMs obtained considering no interaction terms: best model is obtained when using method *k-means* and sample size $N = 200$.

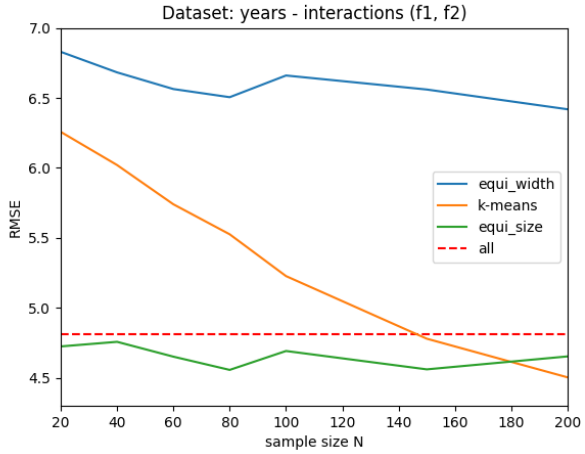

Figure 4.28: RMSE plot of best GAMs obtained with different sampling method extracted from the original forest trained with dataset `years`. Models do consider an interaction between features f1 and f2.

All interactions Figure 4.29 shows the behaviour in terms of RMSE of all the GAMs trained considering two interaction terms: one between f1 and f2, one between f1 and f3. Models results are similar to those of the models not considering interactions and models considering a single interaction between f1 and f2. Model performing better is obtained using *k-means* as sampling method, $N = 200$ as sample size.

Choose best GAM We compare the best model obtained for each one of the three cases analysed: best GAM considering no interaction at all, best GAM considering an interaction term between f1 and f2 and best GAM

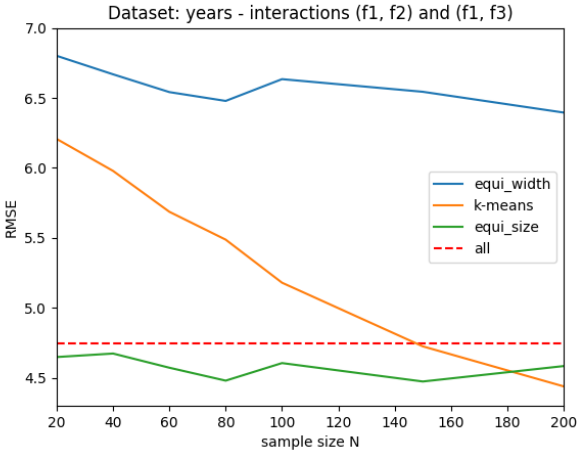

Figure 4.29: RMSE plot of GAMs trained using dataset extracted with different sampling method and size (N) from the original forest trained with dataset `years`. All the models considers has two interaction terms: one between f1 and f2, one between f1 and f3.

considering two interaction terms, one between f1 and f2 and one between f1 and f3. In figure 4.30 we can see the results: introducing an interaction term between features f1 and f2 brings some improvement to the model. Moreover, also interaction between f1 and f3 seems to be relevant.
We therefore decide to choose the GAM considering both interactions, trained with the dataset obtained with the *k-means* sampling method and $N = 200$.

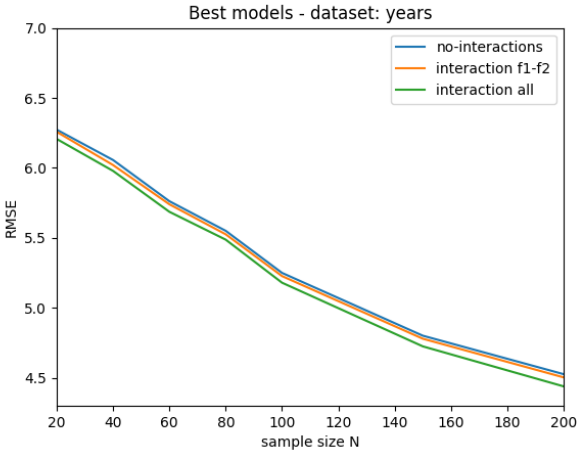

Figure 4.30: Comparison between best GAM considering no interaction at all, best GAM considering an interaction term between f1 and f2 and best GAM considering two interaction terms, one between f1 and f2 and one between f1 and f3.

Final GAM In figure 4.31 are represented the 8 shape functions composing the best GAM obtained from the forest trained with the `years` dataset. Shape function of interaction term between f1 and f2 indicates that the cooperation between the two features makes the response value increase when the value of f1 rises and the value of f2 falls. A similar behavior can also be

observed in the shape function of the interaction between f1 and f3. Graphs of features f5 and f4 show an inversely proportional relationship between features and response; while shape function of feature f12 shows an increase in the response value as the feature value increases. Shape functions of features f11, f6 and f8 show a wider confidence interval and, in general, a more stationary behavior than other features.

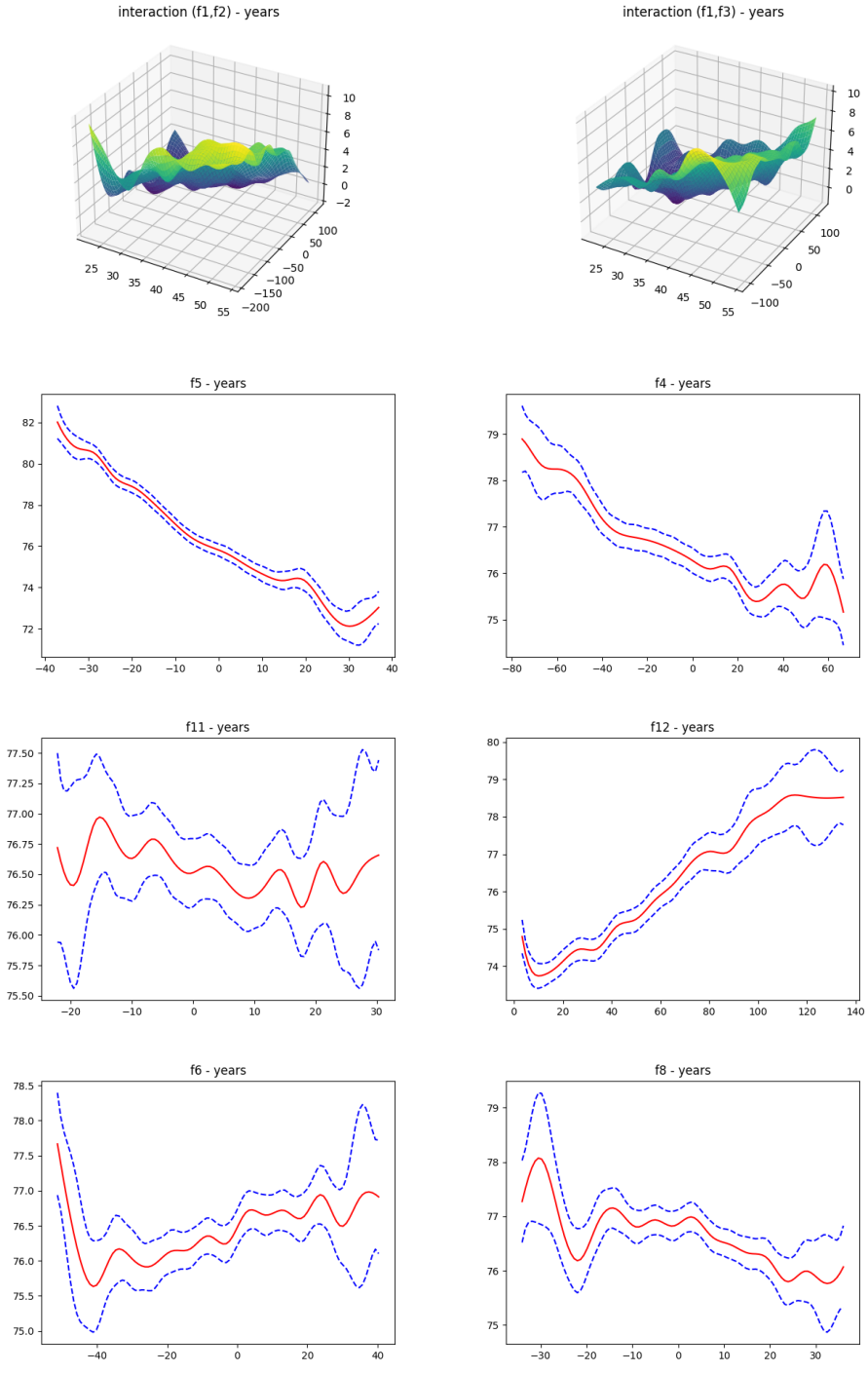

Figure 4.31: *Shape functions* of the best GAM obtained from the forest trained with the `years` dataset: GAM is trained with a dataset built by sampling 200 values from thresholds using the *k-means* method; it considers two interaction terms, one between features f1 and f2 and the other between f1 and f3. Blue lines represent the 95% confidence interval for the estimated function.

4.8 Conclusion

In this chapter we have seen the procedure to obtain a GAM that acts as an interpreter of a forest of decision trees, starting from the forest itself. Procedure is divided into two phases: forest structure analysis and GAM extraction. In the first phase, we observe trees internal structure to get clues about forest behaviour. In particular, we observe how tests on features are distributed in trees internal nodes in order to identify most important features and possible features interactions. Second phase processes results obtained from the forest structure analysis to obtain the GAM.
In the end, we apply the procedure to some case study in order to test how procedure works on real data. Forest learned on real data are a composition of hundreds or thousands of decision tree, hard to be interpreted and extremely powerful. GAMs are a linear combination of very simple models, each representing the relationship between one - or maximum two - features and the response. It is clear that GAMs have less decision-making power than forests. Another degree of approximation is introduced when reducing the number of features considered to train the model. However, as we can see in table 4.15, accuracy of GAMs towards forest predictions is excellent. In the case of the `concrete` dataset, we use all 8 features and we get a $RMSE = \pm 3.1$ MPa. The `houses` dataset has 18 features, 9 are used and we get a $RMSE = \pm 1.3$ (log transformation has been applied to original house prices). The `years` dataset aims to predict the year of publication of a song from audio features. The dataset has 90 features, we use 9 of them and we get a $RMSE = \pm 4.5$ years.

Case study	**n.of features**	**n. GAM terms**	**GAM RMSE**
`concrete`	8	8	3.1
`houses`	18	9	1.3
`years`	90	9	4.5

Table 4.15: Case studies results.

We can therefore conclude that GAMs can actually be used as interpreter of a forest of decision trees since it is an easily interpretable model and powerful enough to imitate the forest behavior.

Conclusions

Decision support systems are often very accurate models, but so complex that they are seen as *black boxes*: models whose internals are either unknown to the observer or they are known but uninterpretable by humans. In the first chapter of this thesis we talk about interpretability and explainability of machine leaning models; then we give a possible definition of the problem of "opening the black box": i.e. find a method that is able to understand how a model works. In particular, when stating that a method is able to open the black box, we are referring to one of the following statements: (*i*) it explains the model, (*ii*) it explains the outcome, (*iii*) it can inspect the black box internally, (*iv*) it provides a transparent solution.
In the second chapter of the thesis, we analyse a well-known and widely used prediction model: forests of decision trees. Forests are extremely accurate and complex models, but they have an interesting feature: they are a ensemble of decision trees. In the chapter dedicated to explainability, we see that decision trees are models that can be interpreted by definition, due to a very simple structure. The complexity of the forests of decision trees, therefore, lies not in the internal structures of the model, but in how decision trees interact with each other.
Third chapter is dedicated to generalized additive models (GAMs). GAMs can be seen as a non-linear extension of linear models which makes them capable of modeling very complex problems, but remaining easily interpretable by humans since they can be represented graphically.
The aim of this thesis is to find a way to obtain a GAM that acts as global interpreter of a forest of decision trees, exploiting information that we can gather by observing forest's internal structure. Fourth chapter contains the description of the procedure we propose to obtain forest global interpreter. Procedure is divided into two phases: *forest structure analysis* and *GAM extraction*. In the first phase, we observe trees internal structure to get clues about forest behaviour. In particular, we observe how tests on features are distributed in trees internal nodes in order to identify most important features and possible features interactions. Second phase processes results

obtained from the forest structure analysis to obtain the GAM. At the end of the chapter, we test the procedure on real-world data.
In order to state that we can actually obtain a GAM that works as global interpreter for a forest of decision trees, we have to satisfy two conditions: first, GAM's shape functions plots have to grasp the relation between features; second, GAMs have to be accurate enough to imitate forest behaviour. As we can see from case study results [figures 4.12, 4.18, 4.24, 4.31], GAMs are able to identify the interaction between features and response and the resulting shape functions are easily interpreted. As far as accuracy towards forest concerns, GAMs are not as accurate as forest and during procedure we introduce some approximations, however, if we analyse RMSE values of GAM computed with respect to forest predictions, we can see that GAM actually are able to imitate forest behaviour.
In the end, we can conclude that GAMs can actually be used as global interpreter of a forest of decision trees, since it is an easily interpretable model and powerful enough to imitate the forest behavior.

Bibliography

[1] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, 2017.

[2] Kshitij Goyal, Sebastijan Dumancic, and Hendrik Blockeel. Feature interactions in xgboost. July 2020.

[3] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), August 2018. ISSN 0360-0300. doi: 10.1145/3236009. URL `https://doi.org/10.1145/3236009`.

[4] Satoshi Hara and Kohei Hayashi. Making tree ensembles interpretable. June 2016.

[5] Trevor Hastie and Robert Tibshirani. Generalized additive models. *Statist. Sci.*, 1(3):297–310, 08 1986.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009. URL `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`.

[7] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R.* Springer Publishing Company, Incorporated, 2014. ISBN 1461471370.

[8] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. page 623–631, 2013. URL `https://doi.org/10.1145/2487575.2487579`.

[9] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach,

R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[10] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[11] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, us ed edition, May 2005. ISBN 0321321367.