Boosting for Regression Problems with Complex Data

by

Xiaomeng Ju

B.Sc., Renmin University of China, 2013
M.Sc., University of Michigan, 2015

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Statistics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

December 2021

Abstract

Preface

You must include a preface if any part of your research was partly or wholly published in articles, was part of a collaboration, or required the approval of UBC Research Ethics Boards.

The Preface must include the following:

- A statement indicating the relative contributions of all collaborators and co-authors of publications (if any), emphasizing details of your contribution, and stating the proportion of research and writing conducted by you.
- A list of any publications arising from work presented in the dissertation, and the chapter(s) in which the work is located.
- The name of the particular UBC Research Ethics Board, and the Certificate Number(s) of the Ethics Certificate(s) obtained, if ethics approval was required for the research.

Examples

Chapter **??** is based on work conducted in UBC's Maple Syrup Laboratory by Dr. A. Apple, Professor B. Boat, and Michael McNeil Forbes. I was responsible for tapping the trees in forests X and Z, conducted and supervised all boiling operations, and performed frequent quality control tests on the product.

A version of chapter **??** has been published **?**. I conducted all the testing and wrote most of the manuscript. The section on "Testing Implements" was originally drafted by Boat, B. Check the first pages of this chapter to see footnotes with similar information.

Note that this preface must come before the table of contents. Note also that this section "Examples" should not be listed in the table of contents, so we have used the starred form: `\section*{Example}`.

Table of Contents

Abstract ii

Preface iii

Table of Contents iv

List of Tables vii

List of Figures ix

List of Programs xiii

Acknowledgements xiv

Dedication xv

1 Introduction 1
- 1.1 Background 1
- 1.2 Gradient boosting 4
  - 1.2.1 Regularization 6
  - 1.2.2 Regression trees as base learners 7
- 1.3 Complex data 10
  - 1.3.1 Inhomogeneous data 10
  - 1.3.2 Functional data 11
  - 1.3.3 Inhomogeneous functional data 12
- 1.4 Outline of the thesis 12

2 Robust gradient boosting 14
- 2.1 Robust regression 14
  - 2.1.1 Robust methods for linear regression 14
  - 2.1.2 Robust methods for nonparametric regression 18
- 2.2 Residual scale-based boosting (SBoost) 18
- 2.3 Two-stage robust gradient boosting (RRBoost) 22

2.3.1 Early stopping . . . 24
2.3.2 Robust variable importance . . . 26
2.4 Simulation studies . . . 27
2.4.1 Set up . . . 27
2.4.2 Implementation details . . . 29
2.4.3 Results and discussion . . . 31
2.5 Data analyses . . . 39
2.5.1 Original benchmark data . . . 39
2.5.2 Data with added atypical observations . . . 40
2.6 Summary . . . 43

3 Boosting for functional regression . . . 44
3.1 Functional regression . . . 44
3.1.1 Functional linear regression . . . 44
3.1.2 Functional nonparametric regression . . . 46
3.1.3 Functional index-based regression . . . 47
3.2 Tree-based functional boosting . . . 49
3.3 Functional multi-index tree . . . 51
3.3.1 Type A tree . . . 52
3.3.2 Type B tree . . . 54
3.4 Simulation studies . . . 58
3.4.1 Set up . . . 58
3.4.2 Implementation details . . . 60
3.4.3 Results and discussion . . . 62
3.5 German electricity data . . . 66
3.6 Summary . . . 69

4 Robust boosting for functional regression . . . 71
4.1 Robust functional regression . . . 71
4.1.1 Robust functional linear regression . . . 72
4.1.2 Robust functional nonparametric regression . . . 74
4.1.3 Robust functional index regression . . . 75
4.2 Robust tree-based functional boosting . . . 76
4.2.1 RTFBoost (LAD-M) . . . 77
4.2.2 RTFBoost (RR) . . . 80
4.3 Simulation studies . . . 82
4.3.1 Set up . . . 83
4.3.2 Implementation details . . . 85
4.3.3 Results . . . 89
4.4 Fruit fly data . . . 98

4.5 Summary 100

5 Conclusion and Future Work 101
5.1 Conclusion 101
5.2 Future work 101

Bibliography 102

Appendices

A First Appendix 113

B Second Appendix 114

List of Tables

2.1 Summary statistics of RMSEs on the test sets for Setting 1 ($g = g_1$, $S = S_0$, $n = 300$, $p = 10$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment. 36

2.2 Summary statistics of RMSEs on the test sets for Setting 2 ($g = g_2$, $S = S_1$, $n = 3000$, $p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment. 36

2.3 Summary statistics of RMSEs on the test sets for Setting 3 ($g = g_3$, $S = S_2$, $n = 300$, $p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment. 36

2.4 Summary statistics of the fractions of variables recovered for Setting 1 ($g = g_1$, $S = S_0$, $n = 300$, $p = 10$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment. 37

2.5 Summary statistics of the fractions of variables recovered for Setting 2 ($g = g_2$, $S = S_1$, $n = 3000$, $p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment. 37

2.6 Summary statistics of the fractions of variables recovered for Setting 3 ($g = g_3, S = S_2, n = 300, p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment. 37

2.7 Summary of the characteristics of the data sets used in the experiments. 39

2.8 Average and standard deviation of trimmed root-mean-square errors (TRMSEs) computed on 50 random training / validation / test data splits. 41

2.9 Average and standard deviation of trimmed root-mean-square errors (TRMSEs) computed on 50 random training / validation / test data splits with added outliers following the settings in Section 2.4.1. 42

3.1 Summary statistics of test MSPEs displayed in the form of mean (sd). The unit of MSPEs is 10^{-6}. 69

List of Figures

2.1 Tukey's bisquare functions and their derivatives 20
2.2 Boxplots of RMSEs on the test sets for 100 independent runs using data generated from Setting 1($g = g_1$, $S = S_0$, $n = 300$, $p = 10$). Panel (a) corresponds to clean training sets with Gaussian errors (D_0); panel (b) corresponds to symmetric gross error contaminated (D_1) data; panel (c) corresponds to asymmetric gross error contaminated (D_2) data; and panel (d) corresponds to skewed distributed (D_3) and heavy-tailed distributed (D_4) data. In panels (b) and (c), the yellow and blue boxplots correspond to 10% and 20% outliers, respectively. In panel (d), the yellow bloxplots correspond to data with log-normaly distributed errors (D_3) and the blue boxplots correspond to data with heavy-tailed distributed errors (D_4). 32
2.3 Boxplots of RMSEs on the test sets for 100 independent runs using data generated from Setting 2 ($g = g_2$, $S = S_1$, $n = 3000$, $p = 400$). Panel (a) corresponds to clean training sets with Gaussian errors (D_0); panel (b) corresponds to symmetric gross error contaminated (D_1) data; panel (c) corresponds to asymmetric gross error contaminated (D_2) data; and panel (d) corresponds to skewed distributed (D_3) and heavy-tailed distributed (D_4) data. In panels (b) and (c), the yellow and blue boxplots correspond to 10% and 20% outliers, respectively. In panel (d), the yellow bloxplots correspond to data with log-normaly distributed errors (D_3) and the blue boxplots correspond to data with heavy-tailed distributed errors (D_4). 33

2.4 Boxplots of RMSEs on the test sets for 100 independent runs using data generated from Setting 3 ($g = g_3$, $S = S_2$, $n = 300$, $p = 400$). Panel (a) corresponds to clean training sets with Gaussian errors (D_0); panel (b) corresponds to symmetric gross error contaminated (D_1) data; panel (c) corresponds to asymmetric gross error contaminated (D_2) data; and panel (d) corresponds to skewed distributed (D_3) and heavy-tailed distributed (D_4) data. In panels (b) and (c), the yellow and blue boxplots correspond to 10% and 20% outliers, respectively. In panel (d), the yellow bloxplots correspond to data with log-normaly distributed errors (D_3) and the blue boxplots correspond to data with heavy-tailed distributed errors (D_4). 34

3.1 Ten random samples generated from Model 1 (left panel) and Model 2 (right panel). 59
3.2 Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_1, M_1, S_1) in panel (a) and (r_1, M_2, S_1) in panel (b). 63
3.3 Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_2, M_1, S_1) in panel (a) and (r_2, M_2, S_1) in panel (b). 63
3.4 Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_3, M_1, S_1) in panel (a) and (r_3, M_2, S_1) in panel (b). 64
3.5 Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_4, M_1, S_1) in panel (a) and (r_4, M_2, S_1) in panel (b). 64
3.6 Visualization of German electricity data with 20 random samples of hourly electricity price curves displayed in panel (a) and histogram of daily average electricity demand of the whole data set in panel (b). 67
3.7 Boxplot of test MSPEs obtained from 100 random splits of the data from part (1) of the experiment. The unit of y-axis is 10^{-6}. 68
3.8 Boxplot of test MSPEs obtained from 100 random splits of the data from part (2) of the experiment. The unit of y-axis is 10^{-6}. 69

4.1 50 random samples generated from M_1 settings with the red curves representing outliers in the predictors. The top left panel shows data generated from D_0, D_1, and D_2; top right panel D_1 and D_2; bottom left panel D_3; and bottom right panel D_4. 86
4.2 50 random samples generated from M_2 settings with the red curves representing outliers in the predictors. The top left panel shows data generated from D_0, D_1, and D_2; top right panel D_1 and D_2; bottom left panel D_3; and bottom right panel D_4. 87
4.3 Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_1; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 93
4.4 Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_2; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 93
4.5 Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_3; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 94
4.6 Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_4; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 94
4.7 Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_5; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 95
4.8 Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_1; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 95
4.9 Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_2; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 96

4.10 Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_3; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 96
4.11 Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_4; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 97
4.12 Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_5; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted. 97
4.13 Test AADs based on 100 random partitions for the fruit fly data. 99
4.14 Test AADs based on 100 random partitions for the fruit fly data with vertical asymmetric outliers. 99

List of Programs

Acknowledgements

This is the place to thank professional colleagues and people who have given you the most help during the course of your graduate work.

Dedication

The dedication is usually quite short, and is a personal rather than an academic recognition. The *Dedication* does not have to be titled, but it must appear in the table of contents. If you want to skip the chapter title but still enter it into the Table of Contents, use this command `\chapter[Dedication]{}`.

Note that this section is the last of the preliminary pages (with lowercase Roman numeral page numbers). It must be placed *before* the `\mainmatter` command. After that, Arabic numbered pages will begin.

Chapter 1

Introduction

Regression analysis is a statistical technique that studies the relationship between variables, often a single response variable (or outcome) variable and one or more explanatory variables (predictors) This technique is widely used for predictive modelling, the primary goal of which is to predict outcomes of new data points based on values of their explanatory variables (Shmueli, 2010). Depending on the type of the responses (e.g. categorical, ordinal, continuous), regression models are defined and estimated differently. We focus on regression with a single continuous response and define the problem below.

1.1 Background

To set the stage, we introduce the notation to represent the variables. We denote the response variable $Y \in \mathbb{R}$ and the explanatory variables $\mathbf{X} \in \mathcal{X}$, where the space $\mathcal{X}$ determines the properties of $\mathbf{X}$ (e.g. real-valued variables $\mathbf{X} \in \mathbb{R}^p$ or square-integrable functional variables $\mathbf{X} \in L^2(\mathcal{I})$ over a set $\mathcal{I}$, typically $\mathcal{I} \subseteq \mathbb{R}$)

Consider a generic regression model

$$Y = F(\mathbf{X}) + \epsilon, \tag{1.1}$$

where the error term ϵ is independent of $\mathbf{X}$ and often assumed to have zero mean and finite variance. One important goal of regression analysis is to estimate the regression function $F : \mathcal{X} \to \mathbb{R}$ in order to make predictions for future observed $\mathbf{X}$.

Assume that we are given a non-negative loss function L that measures the difference between the prediction and the response, we define the risk associated with a function G as the expectation of the loss function over the join distribution of $\mathbf{X}$ and Y:

$$R(G) = E_{\mathbf{X},Y}(L(Y, G(\mathbf{X}))).$$

Many regression methods estimate F in (1.1) by finding a G that minimize this risk over a class of functions $\mathcal{G}$, the richness of which is determined by the assumptions on G:

$$\hat{F} = \underset{G \in \mathcal{G}}{\operatorname{argmin}} R(G). \tag{1.2}$$

In general, the objective function in (1.2) cannot be evaluated since the joint distribution of $\mathbf{X}$ and Y is unknown. However, assume that we have access to a training set containing independent samples $(\mathbf{x}_i, y_i)$, $i \in \mathcal{I}_{\text{train}}$ drawn from this joint distribution, and we can instead minimize the average of the loss function over the training data set, which is termed as the "empirical risk". This estimating procedure is known as empirical risk minimization (ERM), which learns

$$\underset{G \in \mathcal{G}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, G(\mathbf{x}_i)). \tag{1.3}$$

There are numerous statistical techniques in place for solving (1.3). They can be classified into parametric, nonparametric, and semi-parametric regression methods depending on how $\mathcal{G}$ in (1.3) is defined.

Parametric regression assumes that the functions in $\mathcal{G}$ can be characterized by a fixed number of parameters. The interest is in estimating the parameters, which are often associated with meaningful interpretations. It is usually easy to compute and interpret the estimated parameters, however, if a wrong functional form is chosen, it can result in severely biased predictions for the response.

Nonparametric regression offers a flexible alternative when there is little information available concerning the form of the regression function, and thus reduces the risk of model mis-specification. It does not make assumptions on the form of the regression function, and instead constructs it with the training data. Many nonparametric regression methods approximate the regression function using splines, which are piecewise polynomials with pieces connect at points called "knots". Depending on one's belief about the smoothness of the regression function, splines can be used to model continuous functions or functions with continuous derivatives of some orders. Roughness penalty is often applied to control the smoothness of the regression estimator. A comprehensive introduction of these spline methods can be found in Wahba (1990) and Eubank (1999).

Some other nonparametric methods use the data points near the point of interest to estimate the response at that point. Intuitively, the points in

the "neighbourhood" of the point of interest are more similar to that point and thus are made to carry more weight in the estimation procedure. The local-constant estimator (or Nadaraya-Watson estimator; Nadaraya (1964) and Watson (1964)) weights each training observation using a kernel function based on its distance from the point of interest. Similarly, K-nearest neighbours (KNN; Cover (1968)) takes the K closest training examples and makes prediction by averaging their responses. These methods do not directly fall under the ERM scheme of (1.3), but they typically choose the bandwidth or the number of neighbours to minimize a proxy of the risk $E_{\mathbf{X},Y}(L(Y, G(\mathbf{X})))$, taken as the loss evaluated on a hold-out validation set or combined over validation sets from different partitions, usually through a cross-validation procedure.

When dealing with datasets with a large number of explanatory variables or training examples, regression trees (Breiman, 1984) are a popular choice due to their fast computation and interpretability. The algorithm to fit a tree takes a top-down approach, recursively partitioning the training data into subsets called "nodes" and uses the observations at each node to make predictions for that node.

Boosting algorithms widely adopt regression trees as the building blocks, which are called "base learners", to construct an ensemble estimator. They combine predictions from multiple base learners to improve the performance of a single one. This approach is the basis of our study, and will be introduced in detail in Section 1.2. There are other methods proposed following the ensemble idea, which makes predictions through combining multiple estimators. Some popular ones include random forests, bagging, and stacking (see Zhou (2019) for a review).

Additive models (Stone, 1985) are another option to fit a nonparametric model with a large number of explanatory variables. For $\mathbf{X} \in \mathbb{R}^p$, these models assume that the conditional expectation of the response is a sum of p nonparametric components, each associated with one explanatory variable. For responses following exponential family distributions, Hastie and Tibshirani (1990) studied generalized additive models, relating Y and $\mathbf{X}$ using a known link function.

Besides parametric and nonparametric regression , there is semiparametric regression that combines the strengths of the two. It provides a useful compromise by reducing the risk of mis-specification of parametric models and avoiding the "curse of dimensionality" of nonparametric ones. The former occurs when the specified model deviates greatly from the true model, leading to seriously biased estimation. The latter is due to the sparseness of data in high-dimensional spaces, which requires the amount of data to

grows exponentially as the number of explanatory variables increases.

Partial-linear models and index models are the two most popular classes of semiparametric regression models. Partial-linear models (Engle et al., 1986) extend multivariate linear regression to include both linear and nonparametric components of the predictors. Index models assume that the response variable depends on linear combinations of the predictors, where each combination is called an "index". The link function between the response and the indices is unknown and needs to be estimated nonparametrically. In many cases, a single index can not fully capture the predictive information in the multivariate structure of $\mathbf{X}$. To address this problem, some methods are proposed to involve multiple indices. Friedman and Stuetzle (1981) developed "projection pursuit regression", assuming that the regression function is a sum of multiple single-index functions. Li (1991) and Xia et al. (2002) studied models that link Y with $\mathbf{X}$ through a multivariable smooth function which takes the input of multiple indices. Their methods are often used as tools for dimension reduction, which aim to find a subspace of the space spanned by multivariate predictors without loosing much information to predict Y.

To address this issue, Li (1991) and Xia et al. (2002) studied models that link Y with X through a multivariable function that takes the input of multiple indices. In Chapter 4, we will revisit some of these index models and introduce their extensions to data with functional explanatory variables.

1.2 Gradient boosting

Gradient boosting is a nonparametric regression approach that builds an estimator in the form of an ensemble of "base leaners", typically simple classifiers or regressors. In this thesis, we focus on its application to regression problems where the response is a real-valued variable.

Consider a training data set $(\mathbf{x}_i, y_i)$, where $i \in \mathcal{I}_{\text{train}}$, that consist of i.i.d. realizations of the pair $(\mathbf{X}, Y)$. The predictor variables $\mathbf{X}$ are in some feature space $\mathcal{X}$, typically $\mathbb{R}^p$ for some $p \geq 1$, and the response Y is a scalar variable in $\mathbb{R}$. We are interested in estimating a function $F : \mathcal{X} \to \mathbb{R}$ in order to make predictions for future observations of $\mathbf{X}$. Following Friedman (2001), we define the target function F as the minimizer of

$$F = \underset{G}{\operatorname{argmin}} \, E_{Y,X} L(Y, G(\mathbf{X})) \tag{1.4}$$

over the joint distribution of $(\mathbf{X}, Y)$, where L is a pre-specified loss function such as the squared loss $L(a, b) = (a - b)^2$. Gradient boosting (Friedman,

2001) assumes that the target function F can be approximated with a function $\hat{F}$ of the form

$$\hat{F}_T(\mathbf{x}) = \sum_{t=1}^{T} \hat{\alpha}_t \hat{h}_t(\mathbf{x}; \hat{\Theta}_t), \tag{1.5}$$

where the functions $\hat{h}_t : \mathcal{X} \to \mathbb{R}$ are "base learners" characterized by parameters $\hat{\Theta}_t$. If the base learners $\hat{h}_t(\mathbf{x}\,; \hat{\Theta}_t)$ are restricted to be functions of a single variable, (1.5) is akin to assuming an additive model. When the base leaners are nonparametric (e.g. regression trees), we omit $\hat{\Theta}_t$ and simply write $\hat{h}_t(\mathbf{x})$ in the expression above.

Gradient boosting can be seen as a step-wise iterative algorithm to minimize the empirical risk:

$$\underset{F}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, F(\mathbf{x}_i)), \tag{1.6}$$

over functions F of the form (1.5). For each of notation, we let $\mathcal{I}_{\text{train}} = \{1, ..., n\}$ and use this set to subscript elements in vectors. Consider the objective in (1.6) as a function of the vector $(F(\mathbf{x}_1), ..., F(\mathbf{x}_n))^T$. At the t-th iteration, we compute the negative gradient vector $\mathbf{u}_t = (u_{t,1}, ..., u_{t,n})^T$ of the objective function in (1.6) at the point obtained form the previous iteration $(\hat{F}_{t-1}(\mathbf{x}_1), ..., \hat{F}_{t-1}(\mathbf{x}_n))^T$:

$$u_{t,i} = -\frac{\partial L(y_i, b)}{\partial b}\Big|_{b=\hat{F}_{t-1}(\mathbf{x}_i)}, \; i \in \mathcal{I}_{\text{train}}. \tag{1.7}$$

Similar to the gradient descent algorithm that sequentially adds to the current point the negative gradient vector of the objective function, gradient boosting adds to the current function estimate an approximation to the negative gradient vector $(\mathbf{u}_t)$ using a base learner $\hat{h}_t(\cdot\,; \Theta_t)$. At the t-th iteration, $\hat{h}_t(\cdot\,; \Theta_t)$ is chosen to to minimize

$$\sum_{i \in \mathcal{I}_{\text{train}}} (u_{t,i} - h_t(\mathbf{x}_i; \Theta_t))^2,$$

over members h_t of a family of base learners $(\mathcal{H})$ and over Θ_t in the parameter space (Γ). We then calculate the optimal step size

$$\hat{\alpha}_t = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, \hat{F}_{t-1}(\mathbf{x}_i) + \alpha \hat{h}_t(\mathbf{x}_i; \hat{\Theta}_t)). \tag{1.8}$$

and update

$$\hat{F}_t(\mathbf{x}) = \hat{F}_{t-1}(\mathbf{x}) + \hat{\alpha}_t \hat{h}_t(\mathbf{x}; \hat{\Theta}_t). \tag{1.9}$$

The resulting algorithm is shown in Algorithm 1, where the shrinkage parameter γ will be introduced in Section 1.2.1. Note that this gradient boosting algorithm requires an initial estimate $\hat{F}_0(\mathbf{x})$, which is usually taken to be a constant that minimizes the training loss:

$$\hat{F}_0(x) = \underset{a \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, a).$$

Algorithm 1: Gradient boosting machines (GBM)

Input : Data set $(\mathbf{x}_i, y_i), i \in \mathcal{I}_{\text{train}}$
Number of iterations T
Class of the base learners $\mathcal{H}$
Parameter space of the base learners Γ
Shrinkage parameter γ
Initial estimate $\hat{F}_0(\mathbf{x})$

for $t = 1 : T$ **do**

$u_{t,i} = -\frac{\partial L(y_i, b)}{\partial b}\big|_{b=\hat{F}_{t-1}(\mathbf{x}_i)}, \ i \in \mathcal{I}_{\text{train}}$

$\hat{h}_t(\cdot\,; \hat{\Theta}_t) = \underset{h \in \mathcal{H}, \Theta_t \in \Gamma}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} (u_{t,i} - h_t(\mathbf{x}_i; \Theta_t))^2$

$\hat{\alpha}_t = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, \hat{F}_{t-1}(\mathbf{x}_i) + \alpha \hat{h}_t(\mathbf{x}_i; \hat{\Theta}_t))$

$\hat{F}_t(\mathbf{x}) = \hat{F}_{t-1}(\mathbf{x}) + \gamma \hat{\alpha}_t \hat{h}_t(\mathbf{x}; \hat{\Theta}_t)$

end

Output : $\hat{F}_T(\mathbf{x})$

1.2.1 Regularization

The boosting algorithm is intrinsically greedy: at each step it attempts to maximize the reduction in the value of the loss function evaluated on the training set. Consequently, overfitting the training set at the expense of an increased generalization error (i.e. a poor performance on future data) is a practical concern. To mitigate this problem, boosting algorithms generally apply two forms of regularization: early stopping and shrinkage.

Early stopping

As the number of iterations T grows, gradient boosting typically continues to reduce the training loss, so that for T large enough, it may eventually

lead to overfitting. To prevent this from happening, early stopping is widely used to determine when to stop running a boosting algorithm and select a T to construct an estimator that generalizes well to future data.

This simple yet effective strategy was originally introduced by Plaut et al. (1986) to determine the number of iterations to train a neural network, and later extended to gradient boosting and other iterative algorithms. A convenient way to find T is to monitor the performance on a validation set ($\mathcal{I}_{\text{val}}$) that follows the same model as the training set ($\mathcal{I}_{\text{train}}$). Usually, the validation set is randomly selected from all available data and set aside from the training set. Let $T_{\max}$ be the maximum number of iterations allowed in a boosting algorithm. The early stopping time is typically defined as the iteration that achieves the lowest loss on $\mathcal{I}_{\text{val}}$:

$$T_{\text{stop}} = \underset{t=1,\ldots,T_{\max}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{val}}} L(y_i, \hat{F}_t(x_i)). \tag{1.10}$$

If the sample size is small, the method described as (1.10) tends to be affected by random partitions of the training and validation set. When the computational budget permits it, one can instead determine the early stopping time based on a cross-validation estimate of the prediction error. This procedure trains and predicts on every data point available, and thus provides a more stable approximation of $E[L(Y, \hat{F}(\mathbf{X}))]$.

Shrinkage

The idea of shrinkage is to "slow down" the learning by scaling the contribution of each base learner by a factor $0 < \gamma < 1$. This corresponds to replacing (1.9) with

$$\hat{F}_t(\mathbf{x}) = \hat{F}_{t-1}(\mathbf{x}) + \gamma \hat{\alpha}_t \hat{h}_t(\mathbf{x}; \hat{\Theta}_t).$$

The inclusion of γ is expected to reduce the impact of each base learner $\hat{h}_t$ on $\hat{F}_t$ by approximating more finely the search path, and as a result, improve the prediction accuracy (Telgarsky, 2013).

There is a trade-off between the value of the shrinkage parameter (γ) and the number of iterations (T). A small γ typically improves the predictions, but requires a larger T to reach the early stopping time, and as a result, increases the computational cost. Empirically, Friedman (2001) found significant improvements in test errors when γ is set to be very small, typically smaller than 0.1.

1.2.2 Regression trees as base learners

Gradient boosting constructs an estimator by combining multiple base learners, and thus the choice of base learners is essential to the performance of the algorithm. Here, we introduce a special case where regression trees (also referred to as decision trees) are used as the base learners.

Regression trees (Breiman, 1984) are conceptually simple, and yet flexible and powerful. Compared with other methods, regression trees have several properties that make them favorable candidates for base learners. They are relatively fast to compute and can naturally incorporate data of mixed types (numerical, ordinal, or categorical). As nonparametric estimators, regression trees model nonlinear $\mathbf{X} \to Y$ mappings without imposing assumptions on the distribution of the predictors or the structure of the regression function. Scaling or transformations of the individual predictors are not an issue, since they are invariant under these operations. They also perform variable selection and easily scale to data with many predictors, including high dimensional settings where the number of predictors is greater than the sample size.

Regression trees partition the feature space into regions, and predict a constant value in each one. A regression tree h with J leaf nodes has an additive form:

$$h(\mathbf{x}; \{c_j, R_j\}_{j=1}^{J}) = \sum_{j=1}^{J} c_j I(\mathbf{x} \in R_j),$$

where R_j's are the regions partitioning the feature space, and c_j is the predicted value in region R_j. When regression trees are used as the base learners, at the t-th boosting iteration, one fits a tree $\hat{h}(\mathbf{x}; \{\hat{c}_{t,j}, \hat{R}_{t,j}\}_{j=1}^{J})$ to minimize the squared loss:

$$\sum_{i \in \mathcal{I}_{\text{train}}} (h(\mathbf{x}; \{c_{t,j}, R_{t,j}\}_{j=1}^{J}) - u_{t,i})^2, \tag{1.11}$$

where $u_{t,i}$'s are the negative gradients defined in (1.7).

The CART algorithm (Breiman, 1984) provides a way to fit a regression tree that minimizes a greedy approximation to (1.11). It takes a top-down approach to grow a tree through splitting data in a greedy fashion. Starting from the root node that contains all data points, the algorithm finds the best split and continues to split the resulted child nodes until some stopping criterion is satisfied, e.g. the tree reaching a user-specified maximum depth.

To find the best split at a given node that contains training observations $\mathbf{x}_i, i \in \mathcal{O}$, the CART algorithm searches through all possible splits. At the

root note, we have $\mathcal{O} = \mathcal{I}_{\text{train}}$. For the j-th feature $x_{\cdot j}$, there exist a finite number of possible splits: $x_{\cdot j} \leq \tilde{x}_1, ..., x_{\cdot j} \leq \tilde{x}_U$, where $\tilde{x}_1, ..., \tilde{x}_U$ are the unique values of $x_{i,j}$ for $i \in \mathcal{O}$. From all possible splits for all features, the algorithm selects the one that optimizes some criterion that evaluates the goodness-of-fit. Naturally one can choose the split that minimizes the prediction error at that node

$$\sum_{i \in R_1} (b_1 - u_i)^2 + \sum_{i \in R_2} (b_2 - u_i)^2 ,$$

where R_1 and R_2 are the regions partitioned by the split, and

$$b_1 = \text{ave}(u_i | \mathbf{x}_i \in R_1) \text{ and } b_2 = \text{ave}(u_i | \mathbf{x}_i \in R_2)$$

are the averages of y_i in each region. The CART algorithm then repeats this process in each of the resulting regions to build the regression tree. After obtaining the tree that approximates the negative gradient vector $\mathbf{u}_t$, we follow the usual boosting steps (1.8) and (1.9) to find the optimal step size $\hat{\alpha}_t$ and update $\hat{F}_t$.

Rather than using the same step size $\hat{\alpha}_t$ for the whole tree, Friedman (2001) introduces a variation that allows each region to have its own step size. Let $\hat{R}_{t,1}$,..., $\hat{R}_{t,J}$ be the regions of the fitted regression tree. This approach finds the constants in each region such that the training loss will be minimized:

$$\{\hat{c}_{t,j}\}_{j=1}^{J} = \underset{\{c_{t,j}\}_{j=1}^{J}}{\text{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L \left(y_i, \hat{F}_{t-1}(\mathbf{x}_i) + \sum_{j=1}^{J} c_{t,j} I(\mathbf{x}_i \in \hat{R}_{t,j}) \right). \quad (1.12)$$

Since the regions are disjoint, (1.12) reduces to

$$\hat{c}_{t,j} = \underset{c_{t,j}}{\text{argmin}} \sum_{i \in \hat{R}_{t,j}} L \left(y_i, \hat{F}_{t-1}(\mathbf{x}_i) + c_j \right) \quad (1.13)$$

for each region j. Compared with using a single step size that is the same for all regions, this approach finds the constant in each region that directly minimizes the training loss. Note that for the special case of using the squared loss, (1.13) becomes

$$\hat{c}_{t,j} = \text{ave}(y_i - \hat{F}_{t-1}(\mathbf{x}_i) | i \in \hat{R}_j),$$

which is the same as the estimates obtained from the previous approach. For other loss functions, this approach tends to make more progress at each iteration, requiring less iterations to reach the early stopping time, but at the same time, can be more prone to overfitting.

1.3 Complex data

Traditionally, gradient boosting is applied to data $(\mathbf{x}_i, y_i), i \in \mathcal{I}_{\text{train}}$ that follow the regression model (1.1) with finite-dimensional real-valued predictors $\mathbf{x}_i \in \mathbb{R}^p$. In real world applications, however, we often encounter data of a more complex nature. In our work, we consider data beyond classical settings, and more specifically, inhomogeneous data, data with functional variables, and a combination of the two.

1.3.1 Inhomogeneous data

Boosting methods generally assume that the data do not contain atypical observations. However, this is often not the case. Collected data can be noisy and may contain "outliers" due to various reasons: measurement errors, entry mistakes, sampling problems, or natural variation due to variable distributions. These "outliers" can be points that follow a different model that is remarkably different from the one that best describes the majority of the data, or simply "extreme" value that deviates from rest. For example, a data point can be regarded as an outlier if the variable is believed to follow a standard normal distribution, but it in fact comes from an exponential distribution with a large mean. Another example is skewed or heavy-tailed distributions where data points at the skewed-end or heavy tails are typically viewed as outliers.

Outlying values can occur in the predictors, responses, or both. A data point that has extreme values in the predictors is called a "high-leverage point". When data contain multiple predictors, these extremes can be particularly high or low values for one or more predictors, or "unusual" combinations of multiple predictors. If the response of a data point deviates from the model that applies to most data points, we call it a "regression outlier", which can be high-leverage or not. A regression outliers that is only outlying in the response is also called a "vertical outlier". In our context, "outliers" refer to regression outliers unless specified otherwise.

The leverage of outliers impacts regression estimators differently based on how the estimators are constructed. For linear regression estimators, high-leverage outliers are more damaging since they tend to pull the fit towards them at the cost of severely increased residuals for low-leverage points. By comparison, many nonparametric estimators construct local fits using neighbouring points, and thus they are less affected by high-leverage points that are far from the rest of the data.

In practice, the proportion and cause of outliers are usually unknown,

making it impossible to make assumptions on their distributions and model them explicitly. If one applies gradient boosting without accounting for outliers, the regression estimator can be highly biased, leading to unreliable predictions. To address this issue, we propose a robust boosting algorithm to fit nonparametric estimators for finite-dimensional data contaminated with outliers. This is the first contribution of our work, which we will present in Chapter 2.

1.3.2 Functional data

Functional data consist of a sample of functional variables, which are often smooth functions. For each subject in the sample, in principle, measurements could be taken at any points of their domain. In real applications, functional data are encountered in various scientific fields. Most commonly, they are random samples of independent real-valued functions $X_1(t)$, ..., $X_n(t)$ defined on a compact interval $\mathcal{I} \subseteq \mathbb{R}$. Such data can be viewed as curves generated from a one-dimensional stochastic process, often assumed to be in $L^2(\mathcal{I})$: the space of square-integrable functions that satisfy $E\left[\int_{\mathcal{I}} X^2(t)dt\right] < \infty$. Examples of such data include growth curves, temperature curves, and spectroscopic curves, amongst others.

More generally, the concept of functional data expands beyond curves to objects defined on a continuum, such as shapes, images (2D or 3D) or movement trajectories. They can be viewed as random elements taking values in a Hilbert space. See Ullah and Finch (2013) for a review of applications of functional data, including curves and other types of functional objects.

Functional variables are inherently infinite dimensional, which poses challenges to regression methods in the classical setting. When functional variables are evaluated at the same design points, the observed values have the same format as classical multivariate data. It may seem that multivariate regression can be applied directly to observed functional data. However, by doing so, one ignores the smoothness of the underlying functions and disregards other functional characteristics, such as the shape of the curves and their derivatives. In other situations, functional variables for each object may be observed at different design points, which makes multivariate methods inapplicable.

When fitting a regression model with functional data, a common strategy is to reduce the dimension of the functional variables, for example, through projecting them onto a finite-dimensional basis or using functional principal components (FPCs). Some other methods take a nonparametric approach that adapts kernel-based regression to functional regression. These meth-

ods generally involves using metrics generalized to evaluate the proximity of functional objects. In Chapter 3, we will provide an overview of these methods. As the second contribution of our work, we will propose a boosting algorithm for functional regression problems.

1.3.3 Inhomogeneous functional data

Similarly to finite-dimensional data, functional data may also contain outliers. These outliers can be "unusual" in different ways, often categorized into "shape outliers" and "magnitude outliers". Following Hyndman and Shang (2010), shape outliers are the curves that exhibit a different shape or pattern from the rest of the data. They may lie within the range of the majority of the sample and need not contain extreme values at any design point. Magnitude outliers are curves that lie outside the range of most data points, containing extreme values either across the whole domain or in some parts of the domain. Usually they can be easily detected by inspecting the plotted curves.

Hubert et al. (2015) present a different way to categorize functional outliers, which divides them into those that are outlying on a large part of the domain (which are called "persistent outliers"), or in a very short time interval (which are called "isolated outliers"). The former is further grouped into shift, shape, and amplitude outliers.

Ideally, a robust functional regression method should be able to deal with regression outliers which may or may not be functional outliers. Comparable to high-leverage outliers in finite-dimensional settings, functional outliers that are also outlying in the responses can be challenging to deal with. Most research on robust functional regression are focused on data that are only outlying in the responses (vertical outliers), and yet, rarely study functional outliers. In Chapter 4, we explore different ways to build boosting estimators for functional regression that is robust to outliers of various types, including vertical and functional outliers. This is the third contribution of our work.

1.4 Outline of the thesis

Our work covers boosting for regression with three types of complex data: (1) contaminated multivariate data, (2) functional data, and (3) contaminated functional data. The chapters in this thesis are organized as follows.

In Chapter 2 we begin by providing an overview of robust regression with finite-dimensional predictors, including robust linear and nonparametric regression estimators. We then develop the ideas behind robust boosting

in more detail, deriving our algorithm from MM-estimators. Since the loss functions involved in this algorithm are typically non-convex, we suggest a reliable initialization step by fitting an L_1 regression tree. To evaluate the importance of the variables, we recommend a robust variable importance measure that can be calculated via a permutation procedure. Lastly, we show results of simulation studies and benchmark data analyses that compare our method with gradient boosting with the squared loss and other robust boosting algorithms in the literature.

In Chapter 3 we first review regression methods for data with functional predictors and scalar responses, and discuss the challenges to fit complex regression functions. We then propose a boosting algorithm using regression trees constructed with multiple projections as the base learners, which we call "functional multi-index trees". We establish identifiability conditions for these trees and introduce two algorithms to compute them: one finds optimal projections for the entire tree, whereas the other searches for a single optimal projection at each split. The performance of our method is compared with linear and nonlinear regression estimators via numerical experiments, including simulation studies and a real example to predict electricity demands. The latter includes an extension of our method to partial-functional settings where data contain functional and real-valued explanatory variables.

Chapter 4 extends our proposal in Chapter 3 to fit robust functional regression estimators in the presence of outliers. Existing proposals on this topic are either restricted to linear models or not robust to high-leverage outliers. We explore several ways to robustify our method proposed in Chapter 3, including an obvious choice to use the L_1 loss, and other options derived from M-estimators or MM-estimators. The last idea can be viewed as a combination of our proposals in Chapter 2 and 3. We discuss several computational issues, including how to initialize the algorithms and choose their parameters. We then conduct simulated studies and real data analyses to access the performance of our methods and compare them against other existing proposals (robust or not).

Chapter 5 makes concluding remarks and points out areas of future work. We introduce several possible extensions of our work, including obvious next steps and directions that has yet to be explored. In particular, we provide ideas of extending our methods to deal with multiple functional predictors, sparse functional predictors, and classification problems.

Chapter 2

Robust gradient boosting

In this chapter, we propose a robust boosting algorithm based on a two-stage approach, similar to what is done for robust linear regression. We begin by reviewing robust regression methods in Section 2.1, including linear and nonparametric methods. We then introduce the proposed robust boosting algorithm and derive the first stage in Section 2.2. Since the loss functions involved in our robust boosting algorithm are typically non-convex, we introduce a reliable initialization step by fitting a L_1 regression tree. In Section 2.3, we derive the second stage and describe the combination of the two stages. To be able to perform variable selection, we present a robust variable importance measure that can be computed via a permutation procedure. In Sections 2.4 and 2.5, we report the results of our simulation studies and benchmark data analyses respectively. In Section 2.6, we summarize our findings and make conclusions. The content of this chapter is an extended version of our published paper Ju and Salibián-Barrera (2021).

2.1 Robust regression

Robust regression refers to a form of regression analysis that is designed to perform well even if data contain outliers. In the literature, most methods proposed for robust regression can be categorized into methods for linear and nonparametric models. We summarize them respectively in the two subsections below.

2.1.1 Robust methods for linear regression

Given data $(\mathbf{x}_i, y_i)$,..., $(\mathbf{x}_n, y_n)$ that are i.i.d. realizations of $(\mathbf{X}, Y)$, where $\mathbf{X} \in \mathbb{R}^p$ and $Y \in \mathbb{R}$, we define a linear model

$$y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \epsilon_i, \ i = 1, ..., n,$$

where $\boldsymbol{\beta} \in \mathbb{R}^p$ is the vector of regression coefficients to be estimated, and ϵ_i's are i.i.d errors that are independent of $\mathbf{x}_i$'s. For ease of notation, if the model contains an intercept term, $\mathbf{x}_{i,1} = 1$ for $i = 1, ..., n$.

The classical linear OLS regression estimator minimizes the sum of squared residuals:

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = \underset{\boldsymbol{\beta}}{\text{argmin}} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \boldsymbol{\beta})^2.$$

It is easy to compute $\hat{\boldsymbol{\beta}}_{\text{OLS}}$, which also has many desirable statistical properties. However, it is well known that the OLS estimator is extremely sensitive to outliers. A single outlier can dramatically change the fitted coefficients, resulting in severely biased predictions.

One way of quantifying the impact of the outliers on the regression estimator is through the notion of the breakdown point (Hampel, 1968). Roughly speaking, the breakdown point measures the smallest proportion of contamination that can cause an estimator to be unbounded. Assume a finite sample of size n, the smallest finite sample breakdown point is $1/n$ and the largest is 0.5. For the OLS estimator, a single outlier can carry the estimate arbitrarily far from the true values and yields an infinite bias. As a result, it has a low finite sample breakdown point of $1/n$, and asymptotically 0.

To improve the robustness of the OLS estimator, many methods have been proposed to reduce the impact of outliers. Among them, we will introduce the ones closely related to our proposals, namely M-estimators, S-estimators, and MM-estimators. A review that covers a wider class of robust regression methods can be found in Yu and Yao (2017). For mathematical details and computation algorithms, see the book of Maronna et al. (2019).

Regression M-estimators

M-estimators (Huber, 1973) for linear regression are defined as the solution to

$$\hat{\boldsymbol{\beta}}_{\text{M}} = \underset{\boldsymbol{\beta}}{\text{argmin}} \sum_{i=1}^{n} \rho\left(\frac{y_i - \mathbf{x}_i^T \boldsymbol{\beta}}{\hat{\sigma}}\right), \tag{2.1}$$

where ρ is a loss function that satisfies: $\rho(z)$ is a nondecreasing function of $|z|$ and $\rho(0) = 0$. Typically, we assume ρ to be symmetric. The auxiliary residual scale estimator $\hat{\sigma}$ in (2.1) is required to make $\hat{\boldsymbol{\beta}}_{\text{M}}$ scale equivariant, such that for any non-zero $a \in \mathbb{R}$, the M-estimator obtained from $(\mathbf{x}_1, ay_1)$, ..., $(\mathbf{x}_n, ay_n)$ is $a\hat{\boldsymbol{\beta}}_{\text{M}}$.

If we use the squared loss function $\rho(z) = z^2$, the corresponding M-estimator is the OLS estimator. To make the estimator more robust, we can make the loss function rise slower than a quadratic, penalizing less the large

absolute residuals. A natural choice is to use $\rho(z) = |z|$, which yields the LAD regression estimator

$$\hat{\boldsymbol{\beta}}_{\text{LAD}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} |y_i - \mathbf{x}_i^T \boldsymbol{\beta}|.$$

Due to $\hat{\boldsymbol{\beta}}_{\text{LAD}}$ minimizing a L_1 norm, this method is also known as L_1 regression. Dielman (2005) provides an overview of this type of estimators. Compared with the OLS estimator, the LAD estimator is more protective against outliers. If we only consider vertical outliers, the LAD estimator has a breakdown point of 0.5, the highest that can be achieved by any estimator. When data contain high-leverage outliers, however, the LAD estimator can be severely affected. In this case, it has a finite sample breakdown point of $1/n$, asymptotically 0, which is as lows as the OLS estimator.

Besides robustness that can be evaluated by the breakdown point, it is also important to consider the efficiency properties of an estimator. For a robust estimator $\hat{\boldsymbol{\beta}}$, unless stated otherwise, "efficiency" refers to the asymptotic efficiency defined as the ratio

$$\text{Eff}(\hat{\boldsymbol{\beta}}) = \tilde{v}(\hat{\boldsymbol{\beta}}_{\text{MLE}})/\tilde{v}(\hat{\boldsymbol{\beta}}),$$

where $\tilde{v}$ is the asymptotic variance, and $\hat{\boldsymbol{\beta}}_{\text{MLE}}$ is the MLE estimator. For regression models with Gaussian errors, $\hat{\boldsymbol{\beta}}_{\text{MLE}} = \hat{\boldsymbol{\beta}}_{\text{OLS}}$, and the LAD estimator has a very low efficiency of 0.64.

The OLS and LAD estimators are special cases of M-estimators defined in (2.1) with particular choices of the loss function ρ that do not require the estimation of the residual scale. For other loss functions, the residual scale estimator $\hat{\sigma}$ can either be computed in a separate step prior to fitting the M-estimator, or be estimated simultaneously with the regression coefficients (see Maronna et al. (2019)).

When ρ in (2.1) has a monotone derivative $\psi = \rho'$, the resulting estimator is called a "monotone regression M-estimator". In this case, the estimator is not protective against high-leverage outliers, having a finite sample breakdown point of $1/n$, asymptotically 0 (Maronna and Yohai, 2000). If ψ tends to zero at infinity, the corresponding estimator is called a "redescending regression M-estimator". That being the case, the loss function ρ is bounded, which offers more robustness to very large errors.

Regression S-estimators

Aiming at achieving a high breakdown point independent of the number of predictors, Rousseeuw and Yohai (1984) proposed S-estimators for linear re-

gression. Compared to the OLS estimator that can be viewed as minimizing the sample variance of the residuals, the regression S-estimator minimizes a robust scale of the residuals. For any $\beta \in \mathbb{R}^p$, let $r_i(\boldsymbol{\beta}) = y_i - \mathbf{x}_i^T\boldsymbol{\beta}$ and $\mathbf{r}(\boldsymbol{\beta}) = (r_1(\boldsymbol{\beta}), ..., r_n(\boldsymbol{\beta}))^T$ denote the vector of residuals. We define S-estimators for linear regression as

$$\hat{\boldsymbol{\beta}}_{\mathrm{S}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \hat{\sigma}(\mathbf{r}(\boldsymbol{\beta})),$$

where $\hat{\sigma}(\mathbf{r}(\boldsymbol{\beta}))$ is an M-scale estimator defined as the solution to

$$\frac{1}{n}\sum_{i=1}^{n} \rho\left(\frac{r_i(\boldsymbol{\beta})}{\hat{\sigma}(\mathbf{r}(\boldsymbol{\beta}))}\right) = \kappa,$$

where ρ is a bounded ρ-function that is continuously differentiable (see Rousseeuw and Yohai (1984)) and $\kappa \in (0, 1]$ controls the robust properties of $\hat{\sigma}$.

The value of κ controls the breakdown point of the S-estimator, which can be as high as 50%. In cases where ρ is characterized by a tuning constant c (e.g. Huber's or Tukey's bisquare loss), for a given κ one needs $\kappa = E_{F_0}[\rho_c(u)]$ to obtain a consistent scale estimator for residuals following F_0, which in turn, makes $\hat{\boldsymbol{\beta}}_s$ consistent. Customarily, F_0 is set to be the standard Gaussian distribution.

Rousseeuw and Yohai (1984) proved that S-estimators can achieve a high breakdown point, however, this high robustness comes at the price of low efficiency. At 50% breakdown point, the efficiency of an S-estimator is only up to approximately 33% for model with Gaussian errors (Hössjer, 1992).

Regression MM-estimators

To reconcile high robustness and high efficiency, Yohai (1987) proposed regression MM-estimators, which first compute a robust but possibly inefficient initial estimate, and then improve its efficiency by using it as the starting point to compute an M-estimate.

Denoted as $\hat{\boldsymbol{\beta}}_0$, the initial estimator is required to be regression, scale, and affine equivariant (see Yohai (1987) for details). Typically, one uses the regression S-estimator with a high breakdown point as the initial estimator, and then compute an M-estimate of scale using the residuals. We denote the M-scale estimator as $\hat{\sigma}$ defined as the solution to

$$\frac{1}{n}\sum_{i=1}^{n} \rho_0\left(\frac{r_i(\hat{\boldsymbol{\beta}}_0)}{\hat{\sigma}}\right) = \kappa,$$

where ρ_0 is a bounded ρ-function and κ is typically set to 0.5. Finally, we solve for

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho_1 \left(\frac{y_i - \mathbf{x}_i^T \boldsymbol{\beta}}{\hat{\sigma}} \right), \tag{2.2}$$

where ρ_1 is bounded ρ-function that satisfy $\rho_1 \leq \rho_0$.

We denote the objective function in (2.2) as $L(\boldsymbol{\beta})$. Yohai (1987) proved that any $\tilde{\boldsymbol{\beta}}$ satisfying

$$L(\tilde{\boldsymbol{\beta}}) \leq L(\boldsymbol{\beta}_0)$$

is consistent and achieves the same efficiency as the global minimum of $L(\boldsymbol{\beta})$. Therefore, it is sufficient to find a local solution to (2.2), which avoids the complexity of searching for the global minimum of a non-convex function. This local solution is called a regression MM-estimator, denote here as $\hat{\boldsymbol{\beta}}_{\text{MM}}$. Yohai (1987) showed that the breakdown point of $\hat{\boldsymbol{\beta}}_{\text{MM}}$ is at least that of $\hat{\boldsymbol{\beta}}_0$.

Based on the properties of $\hat{\boldsymbol{\beta}}_{\text{MM}}$, one can choose ρ_0 and ρ_1 to achieve the desired the breakdown point and efficiency . In practice, Tukey's bisquare function is commonly used, with ρ_0 associated with a high breakdown point and ρ_1 with high asymptotic efficiency for Gaussian errors. Starting from $\hat{\boldsymbol{\beta}}_0$, an iterative algorithm of weighted least squares can be applied to compute $\hat{\boldsymbol{\beta}}_{\text{MM}}$, in the same way as what we do to compute an M-estimator.

2.1.2 Robust methods for nonparametric regression

2.2 Residual scale-based boosting (SBoost)

As mentioned previously, prior attempts to robustify boosting replaced the squared loss with Huber's loss, which requires the use of a preliminary scale estimator of the residuals. However, to estimate the scale of the residuals we need a robust regression estimator with which to compute the residuals. To avoid this problem, similarly to what regression S-estimators do, we consider minimizing an M-estimator of the scale of the residuals.

Given a training data set $(\mathbf{x}_i, y_i)$, where $i \in \mathcal{I}_{\text{train}}$, $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$. Our goal is to estimate the regression function F that relates the explanatory variables $\mathbf{x}$ to the response y. We define an S-type boosting estimator

$$\hat{F} \;=\; \underset{F \in \mathcal{F}}{\operatorname{argmin}}\, \hat{\sigma}_n(F)\,, \tag{2.3}$$

where $\mathcal{F}$ is the class of functions that take the form of a boosting estimator

$$F(\mathbf{x}) = \sum_{t=1}^{T} \hat{\alpha}_t \hat{h}_t(\mathbf{x}), \tag{2.4}$$

where the step sizes $\hat{\alpha}_t \in \mathbb{R}$ (without loss of generality may include the shrinkage parameter, see Section 1.2.1) and the functions $\hat{h}_t : \mathbb{R}^p \to \mathbb{R}$ are base learners. In (14), $\hat{\sigma}_n(F)$ is a M-scale estimator of the residuals given as the solution of

$$\frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i \in \mathcal{I}_{\text{train}}} \rho_0 \left(\frac{y_i - F(\mathbf{x}_i)}{\hat{\sigma}_n(F)} \right) = \kappa \,. \tag{2.5}$$

where $\kappa > 0$, and $\rho_0 : \mathbb{R} \to [0, +\infty)$ is a symmetric function around zero. These two parameters control the robustness and efficiency of the resulting estimator (Maronna et al., 2019). Note that if we take $\rho_0(u) = u^2$ this approach reduces to the usual boosting with the squared loss function.

In what follows we use functions ρ_0 in Tukey's bisquare family (but our approach applies to any bounded differentiable function ρ_0):

$$\rho_{\text{Tukey},c}(u) = \begin{cases} 1 - \left(1 - (\frac{u}{c})^2\right)^3 & \text{if } |u| \leq c\,; \\ 1 & \text{if } |u| > c\,, \end{cases} \tag{2.6}$$

where $c > 0$ is a user chosen tuning constant. Figure 2.1 shows two examples of Tukey's bisquare functions and their derivatives defined with different c values. We can see that the parameter c determines the shape of these functions, and thus affects the statistical properties (e.g. robustness and efficiency) of their corresponding regression estimators.

In the robustness literature it is customary to assume that non-outlying observations follow a specific regression model, e.g. $Y = F(\mathbf{X}) + \sigma\varepsilon$, where $\sigma > 0$ is an unknown scale parameter and the residuals ε have a fixed distribution H. In that case, the above residual M-scale estimator will be consistent if ρ_0 satisfies $E_H \rho_0(\varepsilon) = \kappa$. When ρ_0 is bounded (and in that case we can assume without loss of generality that $\sup_t \rho_0(t) = 1$), using $\kappa = 0.5$ in (2.5) yields a residual scale estimator with maximal breakdown point. For clean data, a very common assumption is that ε is Gaussian. In that case, the choices $c = 1.547$ in (2.6) and $\kappa = 0.5$ in (2.5) result in a consistent and highly robust estimator. Similarly, highly robust and consistent estimators for other models can be obtained by choosing the tuning constant c in (2.6) as the solution to the equation $E_H \rho_0(\varepsilon) = 0.5$. We refer the interested reader to Maronna et al. (2019) for details.

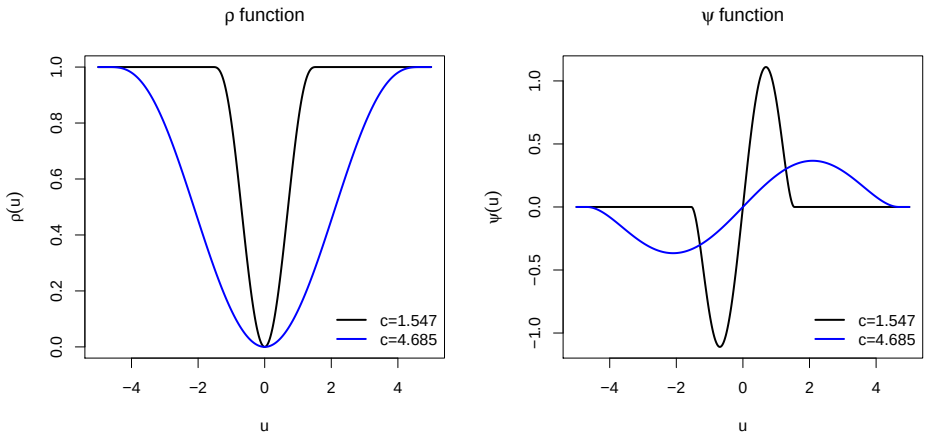


Figure 2.1: Tukey's bisquare functions and their derivatives

To compute our S-type boosting estimator, we follow the same approach used for gradient boosting, namely: at each step t we calculate the gradient of the objective function (14) and approximate the negative gradient with a base learner. More specifically, let training sample size $n = |\mathcal{I}_{\text{train}}|$, and the objective function (14) can be considered as a function of $(F(\mathbf{x}_1), ..., F(\mathbf{x}_n))^T$. At the t-th step, the i-th coordinate of the negative gradient of the objective function is given by:

$$u_{t,i} \;=\; -\frac{\partial \hat{\sigma}_n(F)}{\partial F(\mathbf{x}_i)}\Big|_{F(\mathbf{x}_i)=\hat{F}_{t-1}(\mathbf{x}_i)}\,, \qquad i \in \mathcal{I}_{\text{train}}\,,$$

where $\hat{\sigma}_n(F)$ is defined in (2.5).

The above can be computed explicitly by differentiating both sides of equation (2.5) with respect to $F(\mathbf{x}_\ell)$, and noting that

$$0 \;=\; \sum_{i\in\mathcal{I}_{\text{train}}} \frac{\partial}{\partial F(\mathbf{x}_\ell)} \left[\rho_0\left(\frac{y_i - F(\mathbf{x}_i)}{\hat{\sigma}_n(F)}\right)\right], \qquad \ell \in \mathcal{I}_{\text{train}}\,.$$

We have

$$0 \;=\; \sum_{i\in\mathcal{I}_{\text{train}}} -\psi_0\left(\frac{y_i - F(\mathbf{x}_i)}{\hat{\sigma}_n(F)}\right)\frac{(y_i - F(\mathbf{x}_i))u_{t,\ell}}{\hat{\sigma}_n^2(F)} + \psi_0\left(\frac{y_j - F(\mathbf{x}_j)}{\hat{\sigma}_n(F)}\right)\frac{1}{\hat{\sigma}_n(F)},$$

where $\psi_0 = \rho_0'$. Solving for $u_{t,\ell}$ in the equation above we get

$$u_{t,\ell} \;=\; C_t\, \psi_0 \left(\frac{y_\ell - \hat{F}_{t-1}(\mathbf{x}_\ell)}{\hat{\sigma}_n(\hat{F}_{t-1})} \right) ,$$

where the constant

$$C_t = \left[\sum_{i \in \mathcal{I}_{\text{train}}} \psi_0 \left(\frac{y_i - \hat{F}_{t-1}(\mathbf{x}_i)}{\hat{\sigma}_n(\hat{F}_{t-1})} \right) \left(\frac{y_i - \hat{F}_{t-1}(\mathbf{x}_i)}{\hat{\sigma}_n(\hat{F}_{t-1})} \right) \right]^{-1} .$$

Then we fit a base learner $\hat{h}_t$ that minimizes

$$\sum_{i \in \mathcal{I}_{\text{train}}} (u_{t,i} - h_t(\mathbf{x}_i))^2 ,$$

over members $h_t \in \mathcal{H}$ (e.g. regression trees of a prespecified maximum depth). Finally, we set

$$\hat{F}_t(\mathbf{x}) = \hat{F}_{t-1}(\mathbf{x}) + \hat{\alpha}_t \hat{h}_t(\mathbf{x}),$$

where $\hat{\alpha}_t$ is the optimal step size given by

$$\hat{\alpha}_t = \underset{\alpha}{\operatorname{argmin}}\, \hat{\sigma}_n \left(\hat{F}_{t-1} + \alpha \hat{h}_t \right) .$$

The resulting algorithm `SBoost` is shown in Algorithm 2.

The initial fit

Gradient boosting algorithms require an initial fit $\hat{F}_0(\mathbf{x})$ to calculate the gradient vector for the first iteration. The classical gradient boosting algorithm is generally initialized with a constant fit $\hat{F}_0(\mathbf{x}) \equiv \alpha_0$ where α_0 solves

$$\alpha_0 \;=\; \underset{a \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, a) .$$

Since the loss function in Algorithm 2 is non-convex, the choice of an initial fit for `SBoost` will typically be more important than for the usual gradient boosting with a squared loss function. A natural initial fit for a robust version of gradient boosting would be to take $\alpha_0 = \text{median}(y_i, ..., y_n)$. However, we have found that this choice may not work well when the response variable includes a wide range of values. The median initialization does not

Algorithm 2: SBoost

Input : A data set $(\mathbf{x}_i, y_i), i \in \mathcal{I}_{\text{train}}$
The number of iterations T_1
The class of base learners $\mathcal{H}$
Initialization $\hat{F}_0(\mathbf{x})$

for $t = 1 : T_1$ **do**

$\hat{\sigma}_n(\hat{F}_{t-1}) = \left\{\sigma : \frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i \in \mathcal{I}_{\text{train}}} \rho_0 \left(\frac{y_i - \hat{F}_{t-1}(\mathbf{x}_i)}{\sigma}\right) = \kappa\right\}$

$C_t = \left[\sum_{i \in \mathcal{I}_{\text{train}}} \psi_0 \left(\frac{y_i - \hat{F}_{t-1}(\mathbf{x}_i)}{\hat{\sigma}_n(\hat{F}_{t-1})}\right) \left(\frac{y_i - \hat{F}_{t-1}(\mathbf{x}_i)}{\hat{\sigma}_n(\hat{F}_{t-1})}\right)\right]^{-1}$

$u_{t,\ell} = C_t \psi_0 \left(\frac{y_\ell - \hat{F}_{t-1}(\mathbf{x}_\ell)}{\hat{\sigma}_n(\hat{F}_{t-1})}\right)$

$\hat{h}_t = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i \in |\mathcal{I}_{\text{train}}|} (u_{t,i} - h(\mathbf{x}_i))^2$

$\hat{\alpha}_t = \underset{\alpha}{\operatorname{argmin}}\, \hat{\sigma}_n \left(\hat{F}_{t-1} + \alpha \hat{h}_t\right)$

$\hat{F}_t(\mathbf{x}) = \hat{F}_{t-1}(\mathbf{x}) + \hat{\alpha}_t \hat{h}_t(\mathbf{x})$

end

Output : $\hat{F}_{T_1}(\mathbf{x})$, $\hat{\sigma}_n(\hat{F}_{T_1})$

consider any explanatory variables and can be far from the global optimum. Due to the non-convexity of the objective function and the gradient descent like approach that boosting takes, we may end up reaching a local optimum with a large objective value. Intuitively the problem is that in those cases many "good" observations may have large residuals $y_i - \alpha_0$ that result in zero initial negative gradients $u_{1,i} = 0$, and the algorithm may fail to update $\hat{F}(\mathbf{x}_i)$ in the following iterations.

Instead, we recommend using a regression tree computed with the L_1 loss (Breiman, 1984). This tree, which we call `LADTree`, at each split minimizes the average absolute value of the residuals. This initial fit is expected to be robust to outliers in the training set, it is able to handle data of mixed types (quantitative, binary and categorical), and it is scalable to data with a large number of explanatory variables.

2.3 Two-stage robust gradient boosting (RRBoost)

Although our numerical experiments confirm the robustness of the `SBoost` fit (see Section 2.4.3), they also show a very low efficiency when the data

do not contain outliers. Mimicking what is done to fix this issue in the case of S-estimators for linear regression (see Section 2.1.1), we propose to refine the `SBoost` estimator by using it as the initial fit for further gradient boosting iterations based on a bounded loss function that is closer to the squared loss function than ρ_0 in (2.5) (e.g. a Tukey's bisquare function with a larger tuning constant). Moreover, in this second step we can use the scale estimator associated with the final `SBoost` fit ($\hat{\sigma}_n(\hat{F}_{T_1})$).

This two-step procedure, which we call `RRBoost`, can be summarized as follows:

- **Stage 1:** compute an S-type boosting estimator (`SBoost`) with high robustness but possibly low efficiency;
- **Stage 2:** compute an M-type boosting estimator initialized at the function and scale estimators obtained in Stage 1.

With this approach we avoid the problem of many robust boosting proposals in the literature that require a preliminary residual scale estimator. Since it is known that highly robust S-estimators for linear regression have low efficiency (Hössjer, 1992), the second step attempts to improve the efficiency of `SBoost`, similarly to what MM-estimators do for linear regression models (Yohai, 1987).

The M-type boosting fit is given by

$$\hat{F} = \underset{F \in \mathcal{F}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \rho_1 \left(\frac{y_i - F(\mathbf{x}_i)}{\hat{\sigma}_n} \right) , \tag{2.7}$$

where F is of the form (2.4), and $\hat{\sigma}_n = \hat{\sigma}_n(\hat{F}_{T_1})$ is the residual scale estimator obtained from `SBoost`. In finite dimensional regression models, the shape of the function ρ_1 determines the robustness and efficiency of the resulting estimator with respect to the assumed model for the clean data. In what follows we will use a Tukey's bisquare function (2.6) for ρ_1. When the non-outlying observations are assumed to follow a regression model with normal errors, the choice $c = 4.685$ yields a highly-efficient robust estimator with maximal breakdown point (Maronna et al., 2019).

The algorithm for the second stage starts from the `SBoost` function estimator $\hat{F}_{T_1}(\mathbf{x})$, and sequentially adds base learners $\hat{h}_1(\mathbf{x})$, ..., $\hat{h}_{T_2}(\mathbf{x})$ that approximate the vector of negative gradients at each step. At the t-th iteration, the i-th coordinate of the negative gradient of the objective function

above is given by

$$\begin{aligned} u_{t,i} &= -\frac{\partial}{\partial F(\mathbf{x}_i)} \rho_1 \left(\frac{y_i - F(\mathbf{x}_i)}{\hat{\sigma}_n} \right) \Big|_{F(\mathbf{x}_i)=\hat{F}_{t-1}(\mathbf{x}_i)} \\ &= \frac{1}{\hat{\sigma}_n} \psi_1 \left(\frac{y_i - \hat{F}_{t-1}(\mathbf{x}_i)}{\hat{\sigma}_n} \right), \qquad i \in \mathcal{I}_{\text{train}}, \end{aligned}$$

where $\psi_1 = \rho_1'$.

Similar to what we do for `SBoost`, we find the optimal step size

$$\hat{\alpha}_t = \underset{\alpha}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \rho_1 \left(\frac{y_i - \hat{F}_{T_1+t-1}(\mathbf{x}_i) - \alpha \hat{h}_t(\mathbf{x}_i)}{\hat{\sigma}_n} \right),$$

and update the estimate

$$\hat{F}_{T_1+t}(\mathbf{x}) = \hat{F}_{T_1+t-1}(\mathbf{x}) + \alpha_t \, \hat{h}_t(\mathbf{x}).$$

The resulting `RRBoost` procedure is described in Algorithm 3.

As previously introduced in Section 1.2.1, many boosting algorithms include a "shrinkage" option (Friedman, 2001). The idea is to reduce the size of the function updates by multiplying the optimal step size $\hat{\alpha}_t$ by a small fixed constant $\gamma \in (0, 1)$. Our software implementation includes a shrinkage option, which corresponds to replacing line 7 in Algorithm 2 with

$$\hat{F}_t(\mathbf{x}) \;=\; \hat{F}_{t-1}(\mathbf{x}) + \gamma \, \hat{\alpha}_t \, \hat{h}_t(\mathbf{x}),$$

and line 5 in Algorithm 3 with

$$\hat{F}_{T_1+t}(\mathbf{x}) \;=\; \hat{F}_{T_1+t-1}(\mathbf{x}) + \gamma \, \hat{\alpha}_t \, \hat{h}_t(\mathbf{x}).$$

We ran `RRBoost` with shrinkage parameter $\gamma = 0.1, 0.3, 0.5$, and 1 in one of our simulated settings. As expected, we observed improvement in prediction error as γ decreases at the cost of requiring a significantly increased number of iterations to reach early stopping time. For more details on this experiment we refer the interested reader to the Appendix.

2.3.1 Early stopping

The boosting algorithm is intrinsically greedy: at each step it attempts to maximize the reduction in the value of the loss function evaluated on the training set. Consequently, overfitting the training set at the expense of

Algorithm 3: RRBoost algorithm

Input : A data set $(\mathbf{x}_i, y_i), i \in \mathcal{I}_{\text{train}}$
The number of stage 1 iterations T_1
The number of stage 2 iterations T_2
The class of base learners $\mathcal{H}$

Stage 1 : Run SBoost for T_1 iterations and obtain:
$\hat{F}_{T_1}(\mathbf{x}_i)$, and $\hat{\sigma}_n$

Stage 2 :

for $t = 1 : T_2$ **do**

$u_{t,i} = \frac{1}{\hat{\sigma}_n} \psi_1 \left(\frac{y_i - \hat{F}_{T_1+t-1}(\mathbf{x}_i)}{\hat{\sigma}_n} \right)$

$\hat{h}_t = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} (u_{t,i} - h(\mathbf{x}_i))^2$

$\hat{\alpha}_t = \underset{\alpha}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \rho_1 \left(\frac{y_i - \hat{F}_{T_1+t-1}(\mathbf{x}_i) - \alpha \hat{h}_t(\mathbf{x}_i)}{\hat{\sigma}_n} \right)$

$\hat{F}_{T_1+t}(\mathbf{x}) = \hat{F}_{T_1+t-1}(\mathbf{x}) + \hat{\alpha}_t \hat{h}_t(\mathbf{x})$

end

Output : $\hat{F}_{T_1+T_2}(\mathbf{x})$

an increased generalization error (i.e. a poor performance on future data) is a practical concern. To mitigate this problem, early stopping is widely adopted by boosting algorithms to stop training before they begin to overfit (see Section 1.2.1). For `RRBoost`, we propose to use an early stopping rule based on monitoring the performance of the predictor on a validation set. This approach can be seen as a form of regularization.

We define the early stopping time of each stage of `RRBoost` as the iteration that achieves the lowest loss function value on a validation set ($\mathcal{I}_{\text{val}}$). Let $T_{1,\max}$ and $T_{2,\max}$ be the maximum number of iterations allowed in `RRBoost` (Algorithm 3). In the first stage (`SBoost`), we refer to (2.5) and define the validation loss:

$$\mathrm{L}_{1,\text{val}}(t) = \hat{\sigma}_{\text{val}}(\hat{F}_t),$$

where $\hat{\sigma}_{\text{val}}(\hat{F}_t)$ satisfies

$$\frac{1}{|\mathcal{I}_{\text{val}}|} \sum_{i \in \mathcal{I}_{\text{val}}} \rho_0 \left(\frac{y_i - \hat{F}_t(\mathbf{x}_i)}{\hat{\sigma}_{\text{val}}(\hat{F}_t)} \right) = \kappa.$$

The early stopping time for `SBoost` is

$$T_{1,\text{stop}} = \underset{t\in\{1,\ldots,T_{1,\max}\}}{\operatorname{argmin}} \mathrm{L}_{1,\text{val}}(t).$$

In the second stage, referring to (2.7) we define the validation loss:

$$\mathrm{L}_{2,\text{val}}(t) = \frac{1}{|\mathcal{I}_{\text{val}}|} \sum_{i\in\mathcal{I}_{\text{val}}} \rho_1 \left(\frac{y_i - \hat{F}_t(\mathbf{x}_i)}{\hat{\sigma}_{\text{val}}(\hat{F}_{T_{1,\text{stop}}})} \right),$$

and the early stopping time

$$T_{2,\text{stop}} = \underset{t\in\{T_{1,\max},\ldots,T_{2,\max}\}}{\operatorname{argmin}} \mathrm{L}_{2,\text{val}}(t).$$

In practice, when running boosting for $T_{1,\max} + T_{2,\max}$ iterations is too costly, one can terminate the boosting iterations when the validation error ceases to improve after several iterations.

2.3.2 Robust variable importance

Variable importance methods typically rank the available explanatory variables in terms of their contribution to the prediction strength of the fit (see also Fisher et al. (2019) for a more general discussion). The variable importance measure suggested by Friedman (2001) for gradient boosting is based on averaging the improvement of a squared error at each tree split over all iterations, which cannot be generalized to arbitrary loss functions and is limited to tree base learners. Motivated by the permutation variable importance criterion used in random forests (Breiman, 2001), here we suggest a suitable robust modification that applies to `RRBoost`.

Permutation importance measures are typically based on randomly reordering the values of a explanatory variable in the validation data, and recording the resulting change in prediction accuracy. The idea is that a variable is "important" if breaking its relationship with the response (by randomly shuffling its values) increases the prediction error. Since we assume that our validation data may be contaminated, a robust prediction accuracy metric is needed to reduce the potential impact of outliers on our prediction error quantification. To this end, we propose to trim the prediction errors that deviate from the median by more than 3 MAD (median absolute deviations). More specifically, let $\tilde{\mathcal{I}}_{\text{val}}$ be the indices of the observations in the validation set that are not flagged as outliers:

$$\tilde{\mathcal{I}}_{\text{val}} = \left\{ i : i \in \mathcal{I}_{\text{val}} \text{ and } \left| \hat{F}(\mathbf{x}_i) - y_i - \hat{\mu}_r \right| < 3\hat{\sigma}_r \right\}, \tag{2.8}$$

where $\hat{\mu}_r = \text{median}(\hat{F}(\mathbf{x}_j) - y_j)$ and

$$\hat{\sigma}_r = \text{MAD}(\hat{F}(\mathbf{x}_i) - y_i, i \in \mathcal{I}_{\text{val}}), \tag{2.9}$$

with $\text{MAD}(r_i) = 1.438\,\text{median}_{1\leq i\leq n}|r_i - \hat{\mu}|$, and $\hat{\mu} = \text{median}_{1\leq j\leq n} r_j$. Finally, to evaluate the prediction error of our predictor on the validation set we use the trimmed root-mean-square error and define the importance of variable j as:

$$\text{VI}(X_j) = \sqrt{\frac{1}{|\tilde{\mathcal{I}}_{\text{val}}|}\sum_{i\in\tilde{\mathcal{I}}_{\text{val}}}\left(\hat{F}_{\pi_j}(\mathbf{x}_i) - y_i\right)^2} - \sqrt{\frac{1}{|\tilde{\mathcal{I}}_{\text{val}}|}\sum_{i\in\tilde{\mathcal{I}}_{\text{val}}}\left(\hat{F}(\mathbf{x}_i) - y_i\right)^2},$$

where $\hat{F}(\mathbf{x})$ is fitted using the validation data, and $\hat{F}_{\pi_j}(\mathbf{x})$ is fitted using validation data with the j-th variable permuted.

2.4 Simulation studies

We performed extensive numerical experiments to assess the performance of `RRBoost` and compare it with that of other proposals in the literature. In particular, we study the prediction properties of different regression boosting methods when the training and validation sets contain outliers, and also when no atypical observations are present.

2.4.1 Set up

Consider a data set $D_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots, N\}$, consisting of explanatory variables $\mathbf{x}_i \in \mathbb{R}^p$ and responses $y_i \in \mathbb{R}$ that follow the model

$$y_i = g(\mathbf{x}_i) + C\epsilon_i\,, \qquad i = 1, \ldots, N\,,$$

where the errors ϵ_i are independent from the explanatory variables $\mathbf{x}_i$, and $C > 0$ is a constant that controls the signal-to-noise ratio (SNR). Let $(\mathbf{x}_i, y_i)$, $i = 1, ..., N$, be i.i.d. realizations of $(\mathbf{X}, Y)$, where $\mathbf{X} = (X_1, ..., X_p)^T$, we define

$$\text{SNR} = \frac{\text{Var}(g(\mathbf{X}))}{\text{Var}(C\epsilon)}\,.$$

In our experiments, we considered three g functions:

$$g_1(\mathbf{X}) = 2\,X_1 - 2X_2 + 8\left(X_3 - \frac{1}{2}\right)^2 + \exp(X_4) + 0.5\,\cos(8\pi X_5)\,\exp(2X_5)\,,$$

$$g_2(\mathbf{X}) = 5\left(X_1^2 + \left(X_2X_3 - \frac{1}{X_2X_4}\right)^2\right)^{\frac{1}{2}}, \quad \text{and}$$

$$g_3(\mathbf{X}) = \left[\sum_{j=1}^{5}\Big(1 + (-1)^j 0.8X_j + \sin(6X_j)\Big)\right]\sum_{j=1}^{3}(1 + X_j/3)\,.$$

The first function (g_1) is additive, and we expect simple base learners to work well in this case. For g_2 we modified a function used in Friedman (1991) (see also Ridgeway et al. (1999) and Sigrist (2018)), while g_3 was adapted from Bühlmann and Yu (2003).

We considered the following error distributions:

- D_0: **Clean** - $\epsilon \sim N(0,1)$;
- $D_1(\alpha)$: **Symmetric gross errors**
  - $\epsilon \sim (1-\alpha)N(0,1) + 0.5\alpha N(20, 0.1^2) + 0.5\alpha N(-20, 0.1^2)$;
- $D_2(\alpha)$: **Asymmetric gross errors** - $\epsilon \sim (1-\alpha)N(0,1) + \alpha N(20, 0.1^2)$;
- D_3: **Skewed errors** - $\epsilon = W - \exp(1/2)$, where W has a standard log-normal distribution; and
- D_4: **Heavy-tailed symmetric errors** - $\epsilon \sim T_1$, a Student's T with 1 degree of freedom.

For the contaminated settings D_1 and D_2, we considered rates of contamination equal to 10% and 20% (which correspond to $\alpha = 10\%$ and $\alpha = 20\%$ respectively). The skewed error distribution in D_3 was centered so that its mean is equal to 0.

In each case, we constructed training, validation, and test sets. Test sets were always generated with uncontaminated data (D_0) and had 1000 observations. Each estimator was fitted using the training and validation sets, while the test sets were reserved to evaluate the performance of the different estimators. The training sets were of size 300 and 3000, corresponding to data of small and moderate sizes. The corresponding validation sets had 200 and 2000 observations, respectively. We used $p = 10$ and $p = 400$ explanatory variables. The combination of training size 300 and $p = 400$

corresponds to a "high-dimensional" case where the size of the training set is smaller than the number of features. The signal-to-noise ratio was set to SNR = 6 when $p = 10$ and SNR = 10 when $p = 400$.

The variables $\mathbf{X} = (X_1, ..., X_p)^T \in \mathbb{R}^p$, were assumed to have uniform marginal distributions between 0 and 1, except for the case $g = g_2$, where X_2 and X_4 in g_2 appear in the denominator term, and followed a $\mathcal{U}(1, 2)$ distribution. We considered three types of correlation structures for the variables in $\mathbf{X}$:

- S_0: Independent components;
- S_1: Decreasing correlation structure: $\text{cor}(X_i, X_j) = 0.8^{|i-j|}$; and
- S_2: Block correlation structure for active explanatory variables (Zou and Hastie, 2005):

$$Cor(X_i, X_j) = \begin{cases} 0.8 & X_i \text{ and } X_j \text{ are in the same block;} \\ 0 & \text{otherwise,} \end{cases}$$

  where the blocks were defined for each regression function g as follows:

  - for g_1 and g_3: Block 1: $[X_1, X_2, X_{.3}]$, Block 2: $[X_4, X_5]$, and
  - for g_2: Block 1: $[X_1, X_2]$, Block: 2: $[X_3, X_4]$.

2.4.2 Implementation details

For each possible combination of regression function, error distribution, variable correlation structure, sample size, and number of features, we conducted 100 independent experiments and compared the following non-parametric predictors:

- `L2Boost:` gradient boosting with squared error loss (Friedman, 2001);
- `LADBoost:` gradient boosting with absolute error loss (Friedman, 2001);
- `MBoost:` gradient boosting with Huber's loss (Friedman, 2001);
- `Robloss:` Robloss boosting (Lutz et al., 2008);
- `SBoost:` SBoost (see Section 2.2) intialized with `LADTree`;
- `RRBoost:` RRBoost (see Section 2.3) intialized with `LADTree`.

All methods under comparison were computed using decision trees of the maximum depth equal to 1, 2, and 3 as base learners (note that a tree of depth 1 is often called a "decision stump"). In order to facilitate comparison and manage the computational effort of our experiment, we did not use a shrinkage parameter.

`MBoost` and `Robloss` both used Huber's loss and the residual scale estimator is re-computed at each iteration. For `MBoost` we followed (Friedman, 2001) and used the 90% quantile of the residuals of the previous iteration, while `Robloss` used the MAD of residuals (Lutz et al., 2008). For `RRBoost`, we used the early stopping rules discussed in Section 2.3.1 with $T_{1,\max} = 500$ and $T_{2,\max} = 1000$ for $d = 1$ trees and $T_{1,\max} = 300$ and $T_{2,\max} = 500$ for $d = 2$ and $d = 3$ trees. For `L2Boost` and `LADBoost`, we tracked their loss functions evaluated on the validation set ($L_{\text{val}}(t)$) and calculated the early stopping time (T_{stop}):

$$T_{stop} = \underset{t\in\{1,\ldots,T_{\max}\}}{\operatorname{argmin}} \; \mathrm{L}_{\text{val}}(t)\,,$$

where the maximum number of iterations $T_{\max} = 1500$ for $d = 1$ trees and $T_{\max} = 800$ for $d = 2$ and $d = 3$ trees. For `MBoost` and `Robloss`, since the tuning constant of their loss functions varies in each iteration, we used average absolute deviation (AAD) of the validation residuals for early stopping and calculated

$$T_{stop} = \underset{t\in\{1,\ldots,T_{\max}\}}{\operatorname{argmin}} \; \mathrm{AAD}_{\text{val}}(t)\,,$$

where

$$\mathrm{AAD}_{\text{val}}(t) = \frac{1}{|\mathcal{I}_{\text{val}}|} \sum_{i\in\mathcal{I}_{\text{val}}} \left|\hat{F}_t(\mathbf{x}_i) - y_i\right|\,,$$

with $T_{\max} = 1500$ for $d = 1$ trees and $T_{\max} = 800$ for $d = 2$ and $d = 3$ trees.

To initialize `RRBoost` we considered `LADTree`s with depth 0, 1, 2, 3 and 4, and the minimum number of observations per node set at 10, 20 or 30 (note that an `LADTree` of depth 0 corresponds to using the median of the responses as the initial fit). Of the 13 different combinations we chose the one that performs the best on the validation set. As the validation set may also contain outliers, we first fitted a depth 0 tree and trimmed as potential outliers those observations with residuals deviating from their median by more than 3 times their MAD.

All our simulations were run using the `R` programming environment (R Core Team, 2019). Regression trees were fit using the `rpart` package, including the `LADTree`s and the base learners. We implemented our method in the `RRBoost` package, which is now available on CRAN.

We compared different methods in terms of their predictive performance and variable selection accuracy. For predictive performance, we recorded the root-mean-square error (RMSE) evaluated on the clean test set ($\mathcal{I}_{\text{test}}$) at the early stopping time (T_{stop}):

$$\text{RMSE}(T_{\text{stop}}) = \sqrt{\frac{1}{|\mathcal{I}_{\text{test}}|} \sum_{i \in \mathcal{I}_{\text{test}}} (\hat{F}_{T_{\text{stop}}}(\mathbf{x}_i) - y_i)^2}.$$

For variable selection accuracy, we measured the proportion of variables recovered (Kazemitabar et al., 2017). Specifically, let M be the set of indices of explanatory variables that were used to generate the data ($M = \{1, 2, \ldots, 5\}$). After ranking all p features using the robust permutation variable importance index, we recorded the proportion of true variables in the top $|M|$ of the ordered list:

$$\frac{1}{|M|} \left| \{j, \text{VI}(X_j) >= \text{VI}(X_{(|M|)})\} \cap M \right|,$$

where $\text{VI}(X_{(|M|)})$ is an order statistics, denoting the $|M|$-th largest value of $\text{VI}(X_1), ..., \text{VI}(X_p)$.

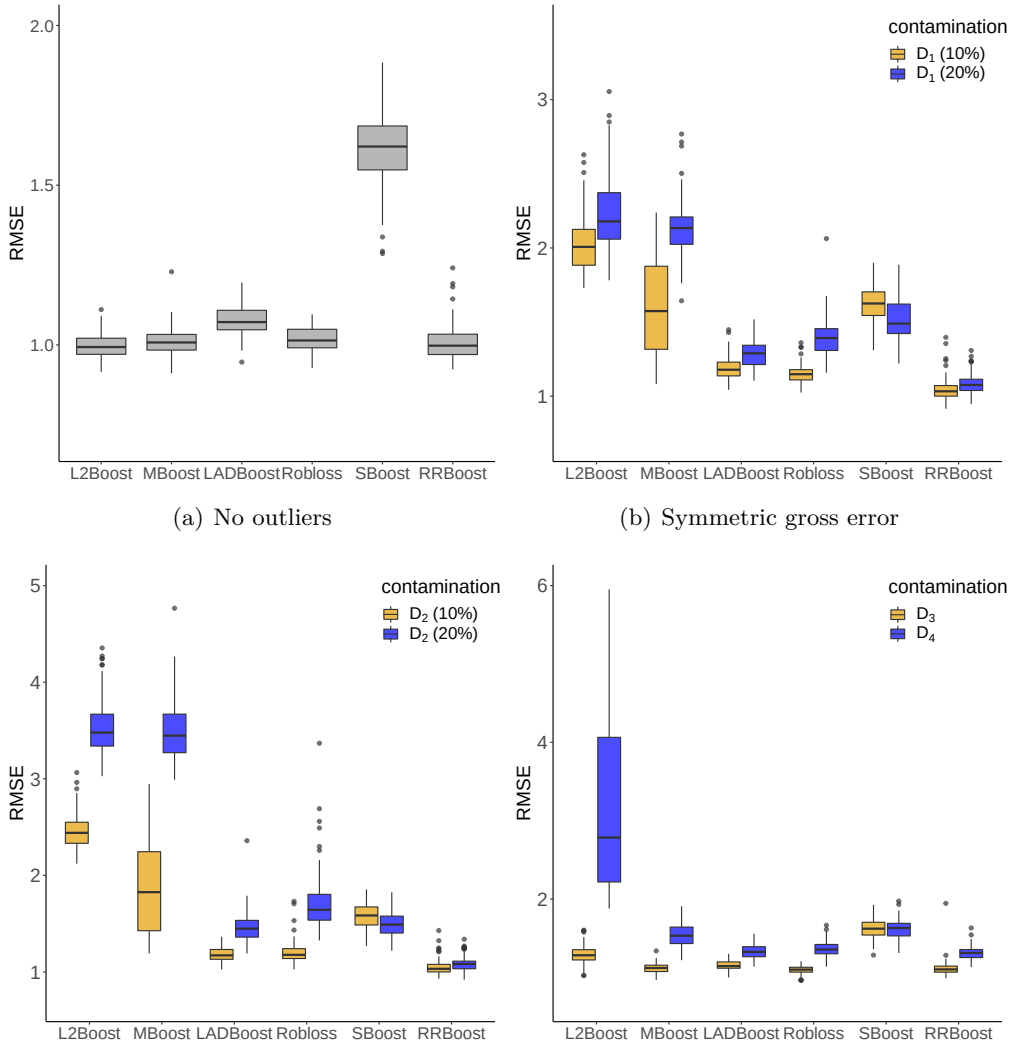
2.4.3 Results and discussion

There are 756 possible combinations of regression function, error distribution, feature correlation structure, sample size, number of features and depth of base learners. All our results can be found in the Appendix. Here we discuss in detail three representative settings, and below give general conclusions from our study. Consider the following three cases:

- Setting 1: $g = g_1$, $S = S_0$, $n = 300$, $p = 10$;
- Setting 2: $g = g_2$, $S = S_1$, $n = 3000$, $p = 400$; and
- Setting 3: $g = g_3$, $S = S_2$, $n = 300$, $p = 400$.

These settings were chosen to represent different regression functions g, correlation structures for the features, and data sizes.

Figures 2.2 to 2.4 show the boxplots of the test root-mean-square error (RMSE) at early stopping times for the 100 independent runs of the experiment, for each of the three settings above (the summary statistics are displayed in Tables 2.1 to 2.3). The depths of the tree learners that resulted in better performances for these three cases were: $d = 1$ for Settings 1 and 3, and $d = 2$ for Setting 2.



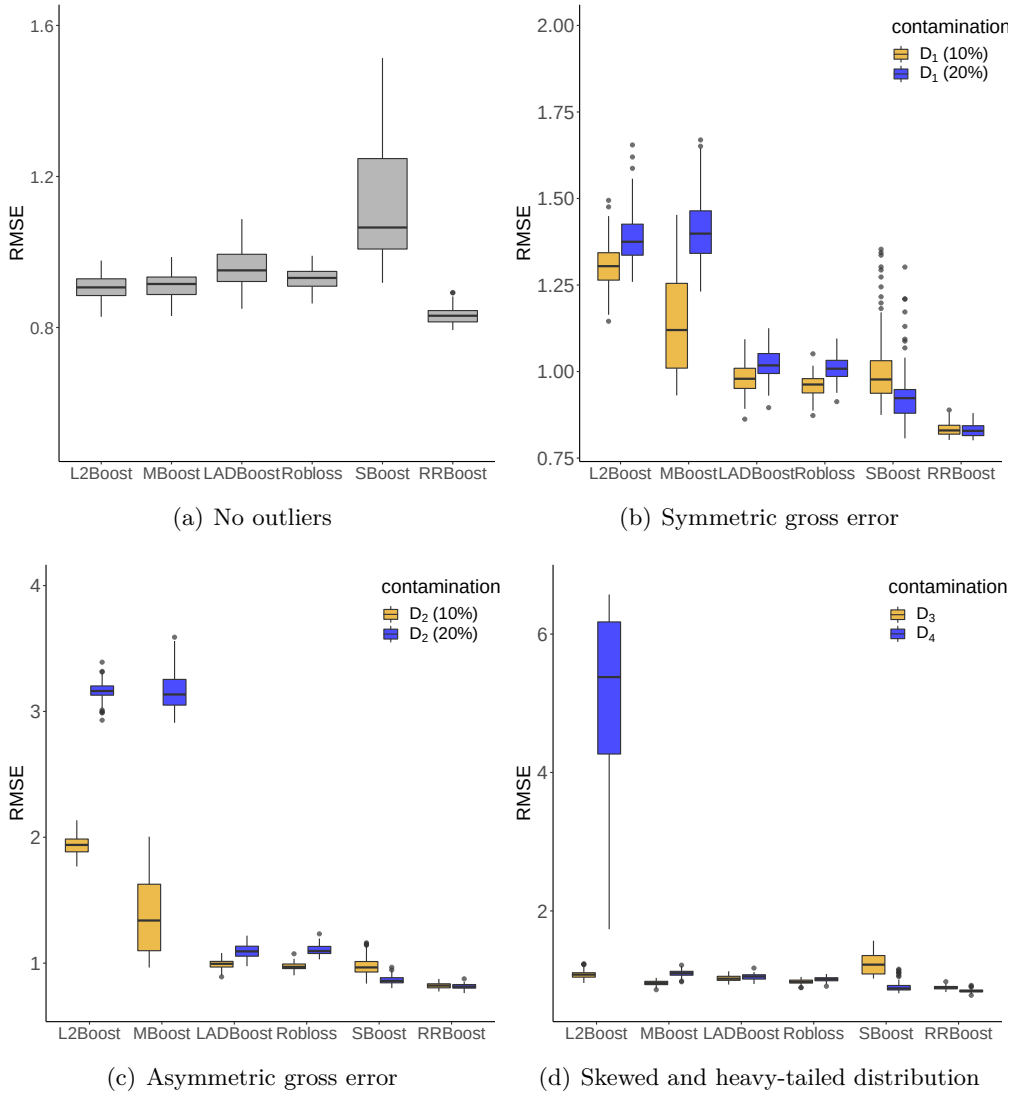
(a) No outliers

(b) Symmetric gross error

(c) Asymmetric gross error

(d) Skewed and heavy-tailed distribution

Figure 2.2: Boxplots of RMSEs on the test sets for 100 independent runs using data generated from Setting 1($g = g_1$, $S = S_0$, $n = 300$, $p = 10$). Panel (a) corresponds to clean training sets with Gaussian errors (D_0); panel (b) corresponds to symmetric gross error contaminated (D_1) data; panel (c) corresponds to asymmetric gross error contaminated (D_2) data; and panel (d) corresponds to skewed distributed (D_3) and heavy-tailed distributed (D_4) data. In panels (b) and (c), the yellow and blue boxplots correspond to 10% and 20% outliers, respectively. In panel (d), the yellow bloxplots correspond to data with log-normaly distributed errors (D_3) and the blue boxplots correspond to data with heavy-tailed distributed errors (D_4).



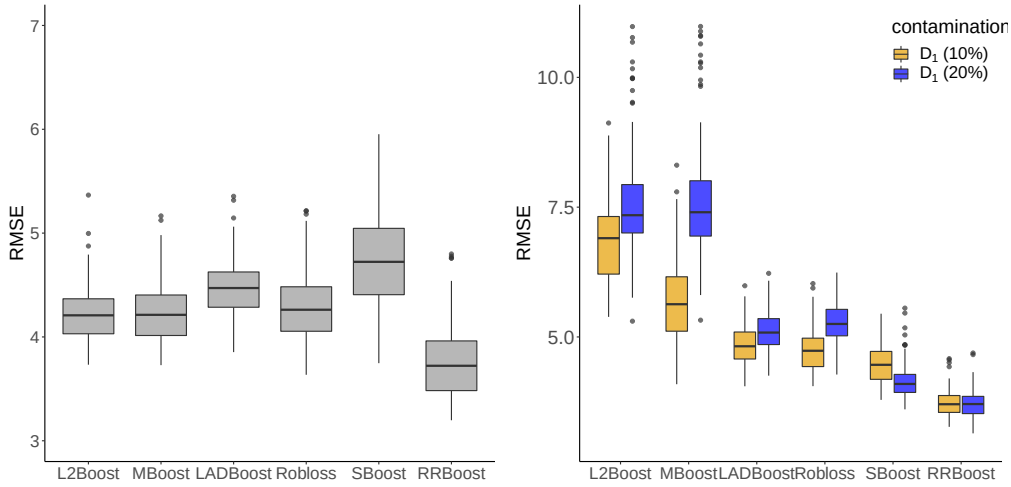
(a) No outliers

(b) Symmetric gross error

(c) Asymmetric gross error

(d) Skewed and heavy-tailed distribution

Figure 2.3: Boxplots of RMSEs on the test sets for 100 independent runs using data generated from Setting 2 ($g = g_2$, $S = S_1$, $n = 3000$, $p = 400$). Panel (a) corresponds to clean training sets with Gaussian errors (D_0); panel (b) corresponds to symmetric gross error contaminated (D_1) data; panel (c) corresponds to asymmetric gross error contaminated (D_2) data; and panel (d) corresponds to skewed distributed (D_3) and heavy-tailed distributed (D_4) data. In panels (b) and (c), the yellow and blue boxplots correspond to 10% and 20% outliers, respectively. In panel (d), the yellow bloxplots correspond to data with log-normaly distributed errors (D_3) and the blue boxplots correspond to data with heavy-tailed distributed errors (D_4).



(a) No outliers

(b) Symmetric gross error

(c) Asymmetric gross error

(d) Skewed and heavy-tailed distribution

Figure 2.4: Boxplots of RMSEs on the test sets for 100 independent runs using data generated from Setting 3 ($g = g_3$, $S = S_2$, $n = 300$, $p = 400$). Panel (a) corresponds to clean training sets with Gaussian errors (D_0); panel (b) corresponds to symmetric gross error contaminated (D_1) data; panel (c) corresponds to asymmetric gross error contaminated (D_2) data; and panel (d) corresponds to skewed distributed (D_3) and heavy-tailed distributed (D_4) data. In panels (b) and (c), the yellow and blue boxplots correspond to 10% and 20% outliers, respectively. In panel (d), the yellow bloxplots correspond to data with log-normaly distributed errors (D_3) and the blue boxplots correspond to data with heavy-tailed distributed errors (D_4).

We find that when there are no outliers in the training and validation sets `L2Boost`, `MBoost`, `Robloss` and `RRBoost` performed equally well. As expected the lack of efficiency of `SBoost` and `LADBoost` were clearly reflected in their increased prediction errors. Note that in Settings 2 and 3, `RRBoost` outperforms `L2Boost`, which we believe is likely due to the fact that `RRBoost` requires more iterations to converge than `L2Boost`, and thus may be resulting in a better approximation to the optimization path.

The performance of `L2Boost`, `MBoost`, `LADBoost`, and `Robloss` seriously deteriorated when outliers were present in the data (both in the symmetric and asymmetric cases), while `RRBoost` achieved the lowest average test RMSE among all methods, with a remarkably stable performance. The worst contamination setting for `MBoost`, `LADBoost`, and `Robloss` was D_2 with 20% outliers. It is interesting to note that for some contaminated settings the performance of `MBoost` was very similar to that of `L2Boost`. A possible explanation is that `MBoost` selects the tuning constant for the loss function as the 90% quantile of the residuals. It is not clear how to address this issue without assuming that the proportion of outliers is known.

Skewed errors (D_3) affected all predictors, but the damage was generally the least among all the contamination settings. Not surprisingly, heavy-tailed symmetric errors (D_4) produced the largest prediction error, with `L2Boost` the most negatively affected, and `RRBoost` achieving the best performance in all three cases.

We also report the fractions of variables recovered (see Section 2.4.2) using the variable importance measure in Section 2.3.2. The results can be found in Tables 2.4, 2.5 and 2.6. We note that for Settings 1 and 2 `RRBoost` an `LADBoost` recovered almost all variables for clean and contaminated data, whereas the other boosting methods (especially `L2Boost` and `MBoost`) recovered fewer variables when trained with contaminated data compared to clean data, with the exception of D_3 (which also least affected the prediction errors). In Setting 3 `RRBoost` recovered over 99% of the variables for all contaminations, except for D_4 where it identified 96% of them. This setting was more challenging for the other methods, especially for `L2Boost` and `MBoost`. In particular, `L2Boost` and `MBoost` recovered less than 30% of variables under $D_1(20\%)$ and $D_2(20\%)$, and `L2Boost` rarely selected the true variables when atypical observations were present.

Analyzing the complete set of results of our numerical experiments (reported in the Appendix) we can make the following general observations:

- Error distributions: The observations from the three cases discussed above extend to the rest of our experiments. Specifically, asymmet-

	L2Boost	MBoost	LADBoost	Robloss	SBoost	RRBoost
D_0	1.00 (0.04)	1.01 (0.04)	1.08 (0.05)	1.02 (0.04)	1.61 (0.12)	1.01 (0.06)
$D_1(10\%)$	2.03 (0.19)	1.61 (0.32)	1.18 (0.07)	1.15 (0.06)	1.61 (0.12)	1.05 (0.08)
$D_1(20\%)$	2.23 (0.25)	2.14 (0.18)	1.29 (0.09)	1.39 (0.13)	1.52 (0.14)	1.09 (0.06)
$D_2(10\%)$	2.46 (0.19)	1.86 (0.45)	1.18 (0.07)	1.20 (0.11)	1.58 (0.13)	1.05 (0.08)
$D_2(20\%)$	3.53 (0.29)	3.49 (0.31)	1.45 (0.16)	1.72 (0.31)	1.50 (0.12)	1.09 (0.07)
D_3	1.29 (0.12)	1.12 (0.06)	1.15 (0.06)	1.10 (0.05)	1.62 (0.12)	1.12 (0.10)
D_4	15.3 (37.0)	1.54 (0.14)	1.33 (0.09)	1.37 (0.10)	1.62 (0.12)	1.31 (0.09)

Table 2.1: Summary statistics of RMSEs on the test sets for Setting 1 ($g = g_1$, $S = S_0$, $n = 300$, $p = 10$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment.

	L2Boost	MBoost	LADBoost	Robloss	SBoost	RRBoost
D_0	0.90 (0.03)	0.91 (0.03)	0.96 (0.05)	0.93 (0.03)	1.11 (0.15)	0.83 (0.02)
$D_1(10\%)$	1.30 (0.07)	1.13 (0.15)	0.98 (0.04)	0.96 (0.03)	1.01 (0.10)	0.83 (0.02)
$D_1(20\%)$	1.39 (0.07)	1.40 (0.09)	1.02 (0.05)	1.01 (0.04)	0.93 (0.09)	0.82 (0.03)
$D_2(10\%)$	1.93 (0.07)	1.39 (0.29)	0.99 (0.04)	0.97 (0.03)	0.98 (0.08)	0.82 (0.02)
$D_2(20\%)$	3.16 (0.08)	3.18 (0.16)	1.09 (0.05)	1.10 (0.04)	0.86 (0.04)	0.82 (0.02)
D_3	1.09 (0.06)	0.96 (0.03)	1.02 (0.04)	0.98 (0.03)	1.23 (0.14)	0.89 (0.03)
D_4	115 (488)	1.10 (0.05)	1.05 (0.05)	1.01 (0.04)	0.89 (0.07)	0.84 (0.02)

Table 2.2: Summary statistics of RMSEs on the test sets for Setting 2 ($g = g_2$, $S = S_1$, $n = 3000$, $p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment.

	L2Boost	MBoost	LADBoost	Robloss	SBoost	RRBoost
D_0	4.22 (0.28)	4.23 (0.30)	4.47 (0.31)	4.31 (0.36)	4.77 (0.46)	3.77 (0.36)
$D_1(10\%)$	6.87 (0.92)	5.71 (0.86)	4.85 (0.41)	4.75 (0.41)	4.49 (0.39)	3.74 (0.28)
$D_1(20\%)$	7.79 (1.33)	7.90 (1.46)	5.11 (0.38)	5.27 (0.38)	4.16 (0.37)	3.71 (0.27)
$D_2(10\%)$	8.34 (0.74)	6.71 (1.31)	4.90 (0.37)	4.88 (0.38)	4.48 (0.46)	3.71 (0.26)
$D_2(20\%)$	13.0 (1.31)	13.2 (1.41)	5.65 (0.44)	6.09 (0.70)	4.16 (0.42)	3.69 (0.33)
D_3	5.07 (0.47)	4.54 (0.32)	4.74 (0.35)	4.57 (0.37)	4.73 (0.53)	3.93 (0.31)
D_4	81.8 (242)	5.47 (0.36)	5.11 (0.37)	5.15 (0.34)	4.59 (0.55)	4.07 (0.38)

Table 2.3: Summary statistics of RMSEs on the test sets for Setting 3 ($g = g_3$, $S = S_2$, $n = 300$, $p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment.

	L2Boost	MBoost	LADBoost	Robloss	RRBoost
D_0	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
$D_1(10\%)$	0.48 (0.27)	0.77 (0.30)	1.00 (0.00)	1.00 (0.02)	1.00 (0.02)
$D_1(20\%)$	0.39 (0.27)	0.27 (0.23)	1.00 (0.02)	0.97 (0.07)	1.00 (0.00)
$D_2(10\%)$	0.50 (0.30)	0.85 (0.21)	1.00 (0.02)	0.99 (0.03)	1.00 (0.00)
$D_2(20\%)$	0.31 (0.22)	0.45 (0.25)	0.99 (0.04)	0.91 (0.16)	1.00 (0.00)
D_3	0.99 (0.04)	1.00 (0.02)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
D_4	0.36 (0.27)	0.92 (0.14)	0.98 (0.06)	0.97 (0.07)	0.98 (0.06)

Table 2.4: Summary statistics of the fractions of variables recovered for Setting 1 ($g = g_1$, $S = S_0$, $n = 300$, $p = 10$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment.

	L2Boost	MBoost	LADBoost	Robloss	RRBoost
D_0	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
$D_1(10\%)$	0.74 (0.12)	0.87 (0.14)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
$D_1(20\%)$	0.66 (0.13)	0.66 (0.14)	0.97 (0.09)	0.98 (0.07)	1.00 (0.00)
$D_2(10\%)$	0.68 (0.12)	0.78 (0.10)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
$D_2(20\%)$	0.53 (0.11)	0.57 (0.11)	0.98 (0.07)	0.91 (0.12)	1.00 (0.00)
D_3	0.82 (0.12)	1.00 (0.00)	0.99 (0.04)	1.00 (0.00)	1.00 (0.00)
D_4	0.03 (0.08)	0.88 (0.13)	0.97 (0.08)	0.96 (0.09)	1.00 (0.00)

Table 2.5: Summary statistics of the fractions of variables recovered for Setting 2 ($g = g_2$, $S = S_1$, $n = 3000$, $p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment.

	L2Boost	MBoost	LADBoost	Robloss	RRBoost
D_0	0.96 (0.08)	0.97 (0.07)	0.91 (0.12)	0.95 (0.10)	0.99 (0.04)
$D_1(10\%)$	0.33 (0.10)	0.55 (0.22)	0.80 (0.16)	0.83 (0.17)	0.99 (0.03)
$D_1(20\%)$	0.23 (0.11)	0.24 (0.12)	0.67 (0.15)	0.64 (0.17)	1.00 (0.03)
$D_2(10\%)$	0.33 (0.09)	0.51 (0.19)	0.79 (0.15)	0.83 (0.15)	1.00 (0.00)
$D_2(20\%)$	0.28 (0.10)	0.29 (0.10)	0.65 (0.15)	0.57 (0.14)	0.99 (0.04)
D_3	0.68 (0.20)	0.89 (0.14)	0.87 (0.15)	0.92 (0.13)	0.99 (0.04)
D_4	0.05 (0.11)	0.53 (0.13)	0.68 (0.17)	0.66 (0.16)	0.96 (0.09)

Table 2.6: Summary statistics of the fractions of variables recovered for Setting 3 ($g = g_3, S = S_2, n = 300, p = 400$). Numbers are averages and standard deviations calculated from 100 independent runs of the experiment.

ric gross errors in $D_2(20\%)$ affected `MBoost`, `LADBoost`, and `Robloss` the most, while `L2Boost` was most sensitive to heavy-tailed errors (D_4). Skewed errors in D_3 only had a mild effect on `L2Boost`, `MBoost`, `LADBoost`, and `Robloss` compared to the other cases. The performance of `RRBoost` was very stable across different types of contamination, whereas the performance of the other boosting methods significantly worsened when the percentage contamination increases from 10% to 20% for D_1 and D_2.

- Sample sizes: As expected, larger training samples improved the prediction error of all boosting methods.

- Correlation structures of the features: it is not surprising that the effect of the correlation structure of the explanatory variables depended on the regression function. For g_1, all boosting methods tended to perform better with correlated features (S_1 and S_2) than in the independent case (S_0). For g_2 and g_3 the best scenario was S_0, and in most cases the second best was S_2.

- Depths of the tree learners: Although the optimal tree depth varied for different scenarios, we observed that for g_1 and g_3, trees with depth $d = 1$ worked the best, while for g_2 it was $d = 2$. The deepest trees considered here ($d = 3$) usually resulted in the worst performances.

- Variable selection results: we observed similar patterns for the fraction of variables recovered for different regression functions and correlation structures. We analyzed the results of different methods for the optimal tree depth at each setting ($d = 1$ for g_1 and g_3, and $d = 1$ or 2 for g_2, see Appendix), treating each sample size setting separately. Since the skewed error distribution D_3 did not affect visibly the variable recovery rate in general (except for a few cases where it mildly affected `L2Boost`) we do not include it in the discussion below.

  - When $n = 300$, `RRBoost` had the best variable recovery rate in almost all tables. `RRBoost` outperformed `L2Boost` and `MBoost` by a significant amount for different types of contaminations. The latter were negatively affected by D_1 and D_2 (more severely by 20% of outliers). `LADBoost` and `Robloss` produced similar results to those of `RRBoost` under $D_1(10\%)$ and $D_2(10\%)$ contaminations. When $p = 400$ the differences among methods were more prominent. Table 2.6 is representative of the situations where the contrast between `RRBoost` and the other methods was particularly

notable. `L2Boost` recovered very few variables under D_4, which could be expected given its poor predictive performance.

– When $n = 3000$, `RRBoost`, `LADBoost` and `Robloss` performed similarly, recovering almost all variables for clean and contaminated settings. `L2Boost` and `MBoost` were negatively affected by D_1 and D_2 (the case $g = g_1$, $S = S_0$ was an exception were all methods worked well, probably due to the combination of an additive function and independent features). `L2Boost` was again severely affected by D_4 (recovering only $20\% \sim 30\%$ features on average). As p increases from $p = 10$ to $p = 400$, we observed a decrease in variable recovery rates for `L2Boost` and `MBoost`. In particular, with $D_1(20\%)$ and $D_2(20\%)$ contaminations their average recovery rates usually dropped from $70\% \sim 90\%$ to $40\% \sim 60\%$.

Overall, `RRBoost` resulted in more accurate predictions for all data generation schemes considered in our experiment, and the associated variable importance measure can be used as a reliable variable selection method.

2.5 Data analyses

We evaluated the performance of our proposed algorithm on benchmark data sets, where we might find very different regression functions and correlation structures from the ones used in our simulation experiments. We considered 5 data sets (see Table 2.7 for details) where previous analyses found potential outliers.

Data	Size (n)	Dimension (p)	Source
airfoil	1503	5	UCI (Brooks et al., 1989)
abalone	4177	7	UCI (Nash et al., 1994)
wage	3000	8	`ISLR` R package (James et al., 2017)
nir	103	94	`pls` R package (Mevik et al., 2019)
glass	180	486	Serneels et al. (Serneels et al., 2005)

Table 2.7: Summary of the characteristics of the data sets used in the experiments.

2.5.1 Original benchmark data

Comparing different predictors on non-synthetic data sets is typically complicated by the fact that generally there is no fixed test set on which to

evaluate their performances. For our experiments we randomly split each data set into a training set (composed of 60% of the data), a validation set (20% of the data) and a test set with the remaining 20%. We ran the boosting algorithms on the training sets, and used the validation sets to select an early stopping time. To avoid the effect of potential outliers in the test set, prediction performance was evaluated with a trimmed root-mean-square error (TRMSE) as follows. Let $\tilde{\mathcal{I}}_{\text{test}}$ be the observations in the test set with prediction errors deviating from their median by less than 3 times of their MAD:

$$\tilde{\mathcal{I}}_{\text{test}} = \{i : i \in \mathcal{I}_{\text{test}} \text{ and } \left|\hat{F}_{T_{\text{stop}}}(\mathbf{x}_i) - y_i - \hat{\mu}_r\right| < 3\hat{\sigma}_r\},$$

where $\hat{\mu}_r = \text{median}(\hat{F}_{T_{\text{stop}}}(\mathbf{x}_i) - y_i)$ and $\hat{\sigma}_r = \text{MAD}(\hat{F}_{T_{\text{stop}}}(\mathbf{x}_i) - y_i, i \in \mathcal{I}_{\text{test}})$. The TRMSE is

$$\text{TRMSE} = \sqrt{\frac{1}{|\tilde{\mathcal{I}}_{\text{test}}|} \sum_{i \in \tilde{\mathcal{I}}_{\text{test}}} \left(\hat{F}_{T_{\text{stop}}}(\mathbf{x}_i) - y_i\right)^2}.$$

In order to reduce the natural variability introduced by the random partitions, we repeated this experiment 50 times with different random splits.

We used the same base learners and number of iterations as in Section 2.4, and show results for the tree depth that produced the best results. Where similar performances were observed for base learners of different depths, we show the results for the simpler one (corresponding to a smaller depth). We picked $d = 2$ for the "airfoil" data, and $d = 1$ for the rest of data sets. Results for all depths can be found in the Appendix.

We initialized `RRBoost` as in Section 2.4.1, using `LADTrees` with depth 0, 1, 2, 3 and 4. The minimum number of observations per node was set at 10, 20 or 30 for the "airfoil", "abalone", and "wage" data sets; and at 5, 10 or 15 for the "nir" and "glass" data sets (on account of their smaller sample sizes).

The results are summarized in Table 2.8. We note that `RRBoost` resulted in the smallest TRMSE for the "airfoil" and "abalone" data sets (in the latter case `LADBoost` is a close second). `RRBoost` also performed better than the alternatives with the "nir" data set, but the magnitudes of the differences are less notable considering the Monte Carlo uncertainty reflected in the standard errors. Finally, we note that all predictors performed similarly on the "wage" and "glass" data sets.

Data	L2Boost	MBoost	LADBoost	Robloss	RRBoost
airfoil	1.72 (0.13)	1.67 (0.11)	2.05 (0.16)	1.67 (0.12)	1.58 (0.12)
abalone	1.78 (0.06)	1.73 (0.05)	1.66 (0.06)	1.70 (0.05)	1.64 (0.05)
wage	24.7 (0.77)	24.0 (0.79)	23.9 (0.72)	23.9 (0.71)	23.9 (0.78)
nir	3.89 (1.00)	3.95 (1.08)	4.42 (1.21)	3.98 (0.93)	3.74 (1.01)
glass	0.10 (0.04)	0.11 (0.04)	0.12 (0.02)	0.13 (0.03)	0.10 (0.02)

Table 2.8: Average and standard deviation of trimmed root-mean-square errors (TRMSEs) computed on 50 random training / validation / test data splits.

2.5.2 Data with added atypical observations

To further explore the behaviour of these predictors on non-synthetic data (where we do not control the shape of the regression function, or the correlation structure among the available features), we performed another experiment by randomly adding outliers to the training and validations sets obtained before. Thus, this study can be considered as an extension of our experiments reported in Section 2.4, where now both the regression function and variable correlation structure remain unknown.

For each data set, new response variables were generated as follows:

$$y_i = y_i^* + C\epsilon_i \,,$$

where y_i^* denote the original responses, ϵ_i are random errors, and $C > 0$ controls the desired signal-to-noise ratio (SNR) as in Section 2.4.1. We used SNR $= 10$ for the high dimensional "glass" data and SNR $= 6$ for the other data sets. For the distribution of errors (ϵ_i) we followed the settings with $D_1(20\%)$, $D_2(20\%)$ and D_4 error distributions in Section 2.4.1, which were shown to significantly affect the performance of predictors in most simulated cases (see Section 2.4.3).

The results are shown in Table 2.9. We see that the performance of `RRBoost` was notably stable, and, for each data set, its results were closest to those for the original data set (Table 2.8) across different contamination settings. Both `L2Boost` and `MBoost` were seriously affected by contaminations (the worst being $D_2(20\%)$). The performance of `Robloss` was relatively good, but clearly worse than that of `LADBoost` for all data sets except for the "glass" data set. The deterioration with respect to the original data was most pronounced for the "airfoil" and "nir" data sets. It is interesting to note that, as it was the case for the original "wage" and "glass" data sets, `LADBoost`, `Robloss` and `RRBoost` produced similar results on their contam-

Data	Error	L2Boost	MBoost	LADBoost	Robloss	RRBoost
airfoil	$D_1(20\%)$	6.49 (0.66)	6.24 (0.50)	3.21 (0.28)	3.47 (0.31)	2.56 (0.20)
	$D_2(20\%)$	12.7 (0.53)	12.5 (0.68)	3.68 (0.27)	4.14 (0.34)	2.56 (0.20)
	D_4	9.04 (4.32)	4.15 (0.33)	3.34 (0.25)	3.49 (0.24)	3.06 (0.25)
abalone	$D_1(20\%)$	2.29 (0.20)	2.26 (0.18)	1.77 (0.06)	1.81 (0.05)	1.74 (0.06)
	$D_2(20\%)$	5.91 (0.17)	5.69 (0.14)	1.98 (0.07)	2.19 (0.07)	1.74 (0.06)
	D_4	4.43 (3.12)	1.93 (0.06)	1.78 (0.07)	1.83 (0.06)	1.75 (0.06)
wage	$D_1(20\%)$	29.9 (1.92)	29.3 (1.63)	24.4 (0.87)	24.5 (0.83)	24.1 (0.84)
	$D_2(20\%)$	77.6 (2.41)	75.2 (2.96)	26.9 (1.03)	29.1 (1.23)	24.1 (0.82)
	D_4	46.4 (16.1)	25.6 (0.87)	24.6 (0.82)	24.8 (0.92)	24.3 (0.80)
nir	$D_1(20\%)$	13.9 (4.74)	12.7 (3.60)	5.58 (1.43)	5.96 (1.32)	4.71 (1.20)
	$D_2(20\%)$	18.3 (3.23)	16.1 (3.45)	6.59 (1.64)	7.83 (2.54)	4.98 (1.28)
	D_4	10.3 (3.11)	6.95 (1.54)	5.48 (1.63)	6.06 (1.40)	5.36 (1.32)
glass	$D_1(20\%)$	0.43 (0.46)	0.33 (0.30)	0.28 (0.12)	0.23 (0.13)	0.22 (0.06)
	$D_2(20\%)$	1.16 (0.48)	1.12 (0.61)	0.30 (0.15)	0.22 (0.17)	0.22 (0.07)
	D_4	0.46 (0.50)	0.21 (0.16)	0.28 (0.13)	0.24 (0.15)	0.28 (0.14)

Table 2.9: Average and standard deviation of trimmed root-mean-square errors (TRMSEs) computed on 50 random training / validation / test data splits with added outliers following the settings in Section 2.4.1.

inated versions. However, Table 2.9 shows that RRBoost resulted in best predictions across contaminations and data sets, and also suffered the least performance degradation when compared with its behaviour on the original data sets (see Table 2.8). We provided top variables ranked using the robust variable importance (see Section 2.3.2) in the Appendix. There is no clear consensus among the rankings obtained with the different boosting methods, but we often observed overlaps in top ranked variables produced by RRBoost using the original data and data with added contamination.

2.6 Summary

In this chapter, we proposed a new robust boosting algorithm (RRBoost) based on best practices for robust regression estimation. Unlike previous robust boosting proposals our approach does not require computing an ad-hoc residual scale estimator in each boosting iteration. We use bounded loss functions, which are known to provide strong protection against outliers, and do not require that the possible proportion of outliers in the data be known or estimated beforehand. Extensive numerical experiments showed that our RRBoost algorithm can be much more robust than existing alternatives in the literature, while behaving very similarly to the standard L2Boost when the data do not contain atypical observations. In particular, the second stage of RRBoost significantly improves the prediction accuracy of the initial SBoost fit without affecting the overall robustness of the resulting predictor. Using a permutation-based variable importance ranking we also showed that in our experiments RRBoost generally achieved the highest variable selection accuracy.

To facilitate the application of RRBoost, we developed an R package available on CRAN (at https://cran.r-project.org/package=RRBoost). In the Appendix, we provide step-by-step instructions on how to apply our functions to a real data set.

Chapter 3

Boosting for functional regression

In this chapter we study a regression problem where the data consist of a scalar response and a functional predictor. We first provide a review of functional regression methods in Section 3.1, categorizing them into linear, non-parametric, and index-based methods. Following this we introduce a boosting algorithm using decision trees constructed with multiple projections as the "base-learners", which we call "functional multi-index trees". Their identifiability conditions and computation algorithms are presented in Section 3.3. We then report numerical results of a simulation study in Section 3.4 and a case study in Section 3.5 where we compare our method with existing alternatives. Finally, we provide a summary of our findings in Section 3.6.

3.1 Functional regression

Depending on the role played by functional variables, regression with functional data can be classified into three categories: scalar-on-function regression, function-on-scalar regression, and function-on-function regression. We focus on the first case, namely regression with a functional covariate X and a scalar response Y, where the main goal is to estimate a function $F : X \to Y$ in order to make predictions for observations of X. In the subsections below we review some common ways to solve this problem, including linear, nonparametric, and index-based regression methods.

3.1.1 Functional linear regression

Functional data are intrinsically infinite dimensional, and that poses challenges that vary with how the regression model is defined. Assuming that $X \in L^2(\mathcal{I})$ are square-integrable functions defined over an interval $\mathcal{I}$, the

linear functional regression model is defined as

$$F(X) = \mu + \langle X, \beta \rangle, \tag{3.1}$$

where $\beta \in L^2(\mathcal{I})$ is the regression coefficient and $\langle X, \beta \rangle = \int_{\mathcal{I}} X(t)\beta(t)dt$ denotes the usual inner product associated with the Hilbert space $L^2(\mathcal{I})$. In many cases, the main focus is to estimate β, which has a natural interpretation: $\beta(t)$ is the influence of the evaluation point at t to the response. This is different from our case, where the focus is to make predictions for future data.

Since β is infinite dimensional, its estimation requires some form of dimension reduction. A very common way of fitting this model is to represent β in some basis system, and control its smoothness by regularizing the expansion, typically through adding roughness penalties. More specifically, we assume $\beta = \sum_{l=1}^{d} c_l \psi_l$, which can also written as $\beta = \mathbf{c}^T \boldsymbol{\psi}$, where $\boldsymbol{\psi} = (\psi_1, ..., \psi_d)^T$ is a vector of basis functions in $L^2(\mathcal{I})$ and $\mathbf{c} = (c_1, ..., c_d)^T \in \mathbb{R}^d$. Given a data set (x_1, y_1),..., (x_n, y_n), we estimate μ and $\mathbf{c}$ to construct β in (3.1) by finding

$$(\hat{\mu}, \hat{\mathbf{c}}) = \underset{\mu, \mathbf{c}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (y_i - \mu - \langle x_i, \mathbf{c}^T \boldsymbol{\psi} \rangle)^2 + \lambda R(\mathbf{c}) \right\}$$

for some regularization parameter $\lambda > 0$ and penalty $R(\mathbf{c})$ that measures the roughness of $\beta = \mathbf{c}^T \boldsymbol{\psi}$ (see Ramsey and Silverman (2005)).

Some estimators consider using eigenfunctions as the basis to construct β, where the eigenfunctions can be obtained derived from Mercer's theorem see ((Adler et al., 2007)). For a stochastic process $X(t)$, we denote its mean $\mu_X(t) = E[X(t)]$ and covariance $C(s,t) = E[(X(s) - \mu_X(t))(X(t) - \mu_X(t)]$. Mercer's theorem implies a decomposition

$$C(s,t) = \sum_{l=1}^{\infty} v_l \phi_l(s) \phi_l(t), \tag{3.2}$$

where $v_1 \geq v_2, ..., \geq 0$ are ordered eigenvalues, and ϕ_j is the eigenfunction corresponding to v_j.

The eigenfunctions form a orthonormal basis in $L^2(\mathcal{I})$. We can then represent

$$X(t) = \mu_X(t) + \sum_{l=1}^{\infty} \xi_l \phi_l(t), \tag{3.3}$$

where

$$\xi_l = \int_{\mathcal{I}} (X(t) - \mu_X(t))\phi_l(t)dt \tag{3.4}$$

is the functional principal component (FPC) score associated with the eigenfunction ϕ_l. The expansion (3.3) is also referred to as the Karhunen-Loève expansion (Fukunaga and Koontz, 1970). In practice, the eigenfunctions are unknown and need to be replaced by suitable estimates. One can refer to Silverman (1996) and Yao et al. (2005) for algorithms to to perform functional principal component analysis (FPCA) based on data.

Similarly to what is usually done in multivariate settings, we can pick the eigenfunctions corresponding to the largest v_l's in (3.2) as the basis, which are the ones that explain the maximum variation of X.

In addition to what we have introduced, a number of proposals further explored the choice of basis expansions and regularization strategies to fit linear functional regression models. We can list Reiss and Ogden (2007), Zhao et al. (2012), Hall et al. (2007), and Cardot et al. (2003) amongst others. For non-Gaussian responses, several authors studied the generalized functional linear model $g\left(F(X)\right) = \mu + \langle X, \beta \rangle$ for a known link function g, see, e.g. James (2002), Cardot and Sarda (2005), Müller et al. (2005), and Dou et al. (2012).

3.1.2 Functional nonparametric regression

A well-specified linear model typically leads to a stable estimator associated with easy interpretation, however, a misspecified one can result in unreliable conclusions. Compared to linear models, nonparametric ones exhibit a higher degree of flexibility and have received recent attention in the functional context. We refer to Ferraty and Vieu (2006) and Ling and Vieu (2018) for general discussions on the topic of functional nonparametric regression. Various proposals adapted nonparametric regression in the finite dimensional case to functional data, many of which are based on kernels that require distances between pairs of explanatory variables.

For functional data, natural measures of proximity involve notions of shape as those given by the similarities of derivatives or FPC scores (Shang, 2016). This suggests the use of semi-metrics as distance measures for many kernel-based functional regression estimators. For example, a functional version of the Nadaraya-Watson (also called local-constant) estimator (Ferraty and Vieu, 2002) is defined as follows:

$$\hat{F}(x) = \frac{\sum_{i=1}^{n} K\left(h^{-1}s(x, x_i)\right) y_i}{\sum_{i=1}^{n} K\left(h^{-1}s(x, x_i)\right)},$$

where $K : \mathbb{R} \to \mathbb{R}$ is a single-dimensional kernel function, h is a positive-valued bandwidth, and s is a semi-metric that measures the distance between functional objects. Other examples include functional K-nearest neighbours (Burba et al., 2009; Kudraszow and Vieu, 2013; Kara et al., 2017), functional recursive kernel estimator (Amiri et al., 2014), and Reproducing Kernel Hilbert Space (RKHS) methods (Preda, 2007; Avery et al., 2014). To avoid having to choose a semi-metric if several are available, Ferraty and Vieu (2009) and Febrero-Bande and González-Manteiga (2013) suggested to combine kernel estimators constructed with different semi-metrics.

It is well known that in finite dimensional settings nonparametric estimators suffer from the "curse of dimensionality" due to the sparseness of data in high-dimensional spaces (Bellman, 1961; Hastie et al., 2009). Similarly, the performance of kernel estimators for functional data is affected by sparsity in the infinite dimensional space, making it difficult to select a bandwidth when a data point has very few neighbours (Geenens et al., 2011; Mas et al., 2012). Exceptions derived from (generalized) additive models avoid this issue by imposing some structure on the target function. For example, Müller and Yao (2008) introduced a functional additive model with each component associated to a FPC score, that is, assuming

$$F(X) = \mu + \sum_{l=1}^{\infty} f_l(\xi_l),$$

where $\mu = E[Y]$, f_l's are smooth functions to be estimated, and ξ_l's are FPC scores defined in (3.4). The fitted estimator takes the FPC scores corresponding to the largest eigenvalues and uses a local polynomial smoother to fit each f_l.

Another additive model is the one considered by Müller et al. (2013) and McLean et al. (2014) that assumes

$$F(X) = \mu + \int G(X(t), t)dt, \tag{3.5}$$

where $\mu = E[Y]$, $X \in L^2(\mathcal{I})$, and $G : \mathbb{R} \times \mathcal{I} \to \mathbb{R}$ is an unknown bivariate smooth function. This model is sometimes known as the "continuously additive model" since the additivity occurs in the time domain. In terms of estimation, Müller et al. (2013) approximates G by a step function, and McLean et al. (2014) with a tensor-product spline basis.

3.1.3 Functional index-based regression

Beyond linear and nonparametric models, there have been several developments towards semi-parametric regression models for functional data, particularly in the direction of index-based models. Ling and Vieu (2020) summarized advances in the field of functional semi-parametric regression in a recent survey. Ferraty et al. (2003) first proposed a functional single-index model

$$F(X) = r\left(\langle X, \beta \rangle\right), \tag{3.6}$$

where $r : \mathbb{R} \to \mathbb{R}$ is an unknown smooth function and $\beta \in L^2(\mathcal{I})$ is the functional index coefficient. Both r and β require to be estimated. The computational and theoretical aspects of this model were further explored by Ait-Saïdi et al. (2008) and Ferraty et al. (2011).

One convenient way to estimate r and β in model (3.6) is to use kernel estimators

$$\hat{r}(\langle x, \beta \rangle) = \frac{\sum_{i=1}^{n} K\left(h^{-1}\langle \hat{\beta}, x - x_i \rangle\right) y_i}{\sum_{i=1}^{n} K\left(h^{-1}\langle \hat{\beta}, x - x_i \rangle\right)}, \tag{3.7}$$

where $K : \mathbb{R} \to \mathbb{R}$ is a kernel function. The coefficient estimate ($\hat{\beta}$) and the bandwidth (h) in (3.7) can be chosen to minimize the cross-validation error.

To fit more complex regression functions, a few proposals studied multi-index models:

$$F(X) = r(\langle X, \alpha_1 \rangle, ..., \langle X, \alpha_q \rangle), \tag{3.8}$$

where q is the number of indices and $r : \mathbb{R}^q \to \mathbb{R}$ is an unknown smooth function. Amato et al. (2006) proposed an extension of the minimum average variance estimation to functional data. Their method uses a q-dimensional kernel smoother to estimate r, which requires a computationally expensive procedure to select q bandwidths. In order obtain a stable estimator when q is large, a large training sample size is needed. A few other proposals were derived from sliced inverse regression (Li, 1991), including functional sliced inverse regression (Ferré and Yao, 2003), functional k-means inverse regression (Wang et al., 2014), and functional sliced average variance estimation (Lian and Li, 2014). These methods avoid q-dimensional smoothing but require more restrictive assumptions on the distribution of X (see Ferré and Yao (2003) for more details).

To reduce the computation cost of fitting (3.8) while not imposing strong assumptions on X, some proposals approximate F with a finite sum of single-index terms:

$$F(X) \approx r_1(\langle X, \alpha_1 \rangle) + ... + r_q(\langle X, \alpha_q \rangle). \tag{3.9}$$

This additive approximation is attractive since each component can be estimated by a single dimensional smoother that can be computed efficiently, for example, by using the single-index kernel estimator defined in (3.7). Chen et al. (2011) and Ferraty et al. (2013) proposed iterative algorithms that estimate r_j's in (3.9) sequentially with kernel estimators, fitting one per iteration while keeping the previous ones fixed. For non-Gaussian responses, James and Silverman (2005) considered a generalized functional index model with a known link function and estimated the r_j's with spline smoothers.

3.2 Tree-based functional boosting

In this section we introduce a new boosting method for nonparametric regression with a functional explanatory variable and a scalar response. As we have seen in Section 3.1.2, most existing nonparametric methods for this problem are based on kernels, which involve selecting bandwidths and choosing semi-metrics. Different from these proposals, we develop a boosting algorithm using decision trees constructed with multiple projections as the "base-learners". We will describe these trees in detail in Section 3.3.

The proposed boosting estimator can be viewed as approximating

$$F(X) \approx r_1(\langle X, \beta_{1,1} \rangle, ..., \langle X, \beta_{1,P} \rangle) + ... + r_T(\langle X, \beta_{T,1} \rangle, ..., \langle X, \beta_{T,P} \rangle), \tag{3.10}$$

where each of the T components $r_j(\langle X, \beta_{j,1} \rangle, ..., \langle X, \beta_{j,P} \rangle)$ is characterized by P indices. The projection directions $\beta_{j,1}, ..., \beta_{j,P}$ and $r_j : \mathbb{R}^P \to \mathbb{R}$ are estimated at the j-th boosting iteration.

Compared to existing index-based functional regression estimators, ours has several advantages. Unlike directly fitting the multi-index model (3.8) using multi-dimensional kernel smoothers, our estimator is build from a combinations of trees, which are more efficient to compute. Different from additive single-index regression that considers an approximation (3.9), our proposal takes into account possible interactions between indices, making it capable of approximating more complex regression functions. Furthermore, our method does not pose restrictive distributional assumptions on the functional predictor, which are the bases to derive some proposals (e.g. Ferré and Yao (2003)). It only assumes that the explanatory variable is from a space where inner-products can be computed (such as a Hilbert space).

There exist other proposals that developed boosting-type algorithms for functional data, but they either apply to tasks different from ours (e.g. identifying the most predictive design points), or do not consider multi-index estimators (Ferraty et al., 2010; Tutz and Gertheiss, 2010; Fan et al., 2015; Ferraty and Vieu, 2009; Chen et al., 2011; Ferraty et al., 2013; Brockhaus et al., 2015; Greven and Scheipl, 2017). Compared to these proposals, our approach fits more "closely" into a boosting framework by minimizing a loss function with a gradient descent-like procedure and applying regularization via shrinkage and early stopping (see Section 1.2.1).

To introduce our algorithm, consider a training data set (x_i, y_i) where $i \in \mathcal{I}_{\text{train}}$, which is an i.i.d. sample of (X, Y), for X in a Hilbert space $\mathcal{L}$ (e.g. $L^2(\mathcal{I})$ the space of square-integrable functions over an interval $\mathcal{I}$) and $Y \in \mathbb{R}$. Our goal is to estimate the target function

$$F = \underset{G}{\operatorname{argmin}}\, E_{Y,X} L(Y, G(X)) \tag{3.11}$$

given a pre-specified loss function L.

Like gradient boosting introduced in Section 1.2, our algorithm minimizes the empirical risk (or training loss)

$$\sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, G(x_i))$$

over a class of G functions that takes a form of a linear combination of "base learners". Let $n = |\mathcal{I}_{\text{train}}|$ be the sample size of the training data . At the t-th iteration, we fit a base learner to approximate the negative gradient vector $(u_{t,1}, ..., u_{t,n})^T$ computed at the point $(\hat{F}_{t-1}(x_1), ..., \hat{F}_{t-1}(x_n))^T$ from the previous iteration, where the negative gradient for the i-th observation is defined as

$$u_{t,i} = -\frac{\partial L(y_i, b)}{\partial b}\Big|_{b=\hat{F}_{t-1}(x_i)}, \; i \in \mathcal{I}_{\text{train}}. \tag{3.12}$$

Core to the performance of a boosting algorithm is the choice of base learners. In the functional context, we consider a flexible class of base learners built from decision trees constructed with multiple projections, which we call "functional multi-index trees". These trees have the advantage of being capable of fitting functions of X that involve multiple projections and their possible interactions. Furthermore, they are scalable to large sample sizes and can easily incorporate additional real-valued explanatory variables. We will explain this in more detail in Section 3.3.

Algorithm 4: Tree-based functional boosting (TFBoost)

Input : Data set $(x_i, y_i), i \in \mathcal{I}_{\text{train}}$
Number of iterations T
Shrinkage parameter $\gamma \in (0, 1)$
Initial estimate $\hat{F}_0(x)$
Number of indices P

1 **for** $t = 1 : T$ **do**

2 $\quad u_{t,i} = -\frac{\partial L(y_i, b)}{\partial b}\big|_{b=\hat{F}_{t-1}(x_i)},\ i \in \mathcal{I}_{\text{train}}$

3 $\quad \hat{h}_t(\langle \cdot, \hat{\beta}_{t,1} \rangle, ..., \langle \cdot, \hat{\beta}_{t,P} \rangle) = \underset{h, \beta_1, ..., \beta_K}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle\right)\right)^2$

4 $\quad \hat{\alpha}_t = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, \hat{F}_{t-1}(x_i) + \alpha \hat{h}_t(\langle \cdot, \hat{\beta}_{t,1} \rangle, ..., \langle \cdot, \hat{\beta}_{t,P} \rangle))$

5 $\quad \hat{F}_t(x) = \hat{F}_{t-1}(x) + \gamma \hat{\alpha}_t \hat{h}_t(\langle x, \hat{\beta}_{t,1} \rangle, ..., \langle x, \hat{\beta}_{t,P} \rangle))$

6 **end**

Output : $\hat{F}_T(x)$

The resulting boosting algorithm is shown in Algorithm 4, where Line 3 approximates the negative gradient vector using a functional multi-index tree that we will introduce next. Line 4 finds the optimal step size $\hat{\alpha}_t$ to minimize the training loss, and Line 5 updates the function estimate where the shrinkage parameter $\gamma \in (0, 1)$ is included in order to reduce overfitting. One can refer back to Section 1.2.1 where we described this regularization strategy.

To determine the number of iterations T, we recommend an early stopping rule based on the performance on a validation set ($\mathcal{I}_{\text{val}}$) that follows the same model as the training set (see Section 1.2.1). Specifically, we define the early stopping time as the iteration that achieves the lowest loss on $\mathcal{I}_{\text{val}}$:

$$T_{\text{stop}} = \underset{t=1,...,T_{\max}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{val}}} L(y_i, \hat{F}_t(x_i)), \tag{3.13}$$

where $T_{\max}$ is the maximum number of iterations allowed in our algorithm. The resulting boosting estimator is given by

$$\hat{F}_{T\text{stop}}(x) = \hat{F}_0(x) + \sum_{t=1}^{T_{\text{stop}}} \gamma \hat{\alpha}_t \hat{h}_t(\langle \cdot, \hat{\beta}_{t,1} \rangle, ..., \langle \cdot, \hat{\beta}_{t,P} \rangle)),$$

where the initial function estimate is generally a constant defined as

$$\hat{F}_0(x) = \underset{a \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} L(y_i, a).$$

3.3 Functional multi-index tree

When the explanatory variables are vectors in $\mathbb{R}^p$ for some $p > 1$, decision trees are commonly used as base learners for gradient boosting, which select variables to make the best splits (see Section 1.2.2). Similarly for $X \in \mathcal{L}$, we "select" the optimal indices and use them to define the splits that partition the data.

Let $\beta_1, , ..., \beta_P \in \mathcal{L}$ be P functions which we view as projection directions. The inner-products $\langle x, \beta_1 \rangle, ..., \langle x, \beta_P \rangle$ are corresponding indices projecting $x \in \mathcal{L}$ onto these directions. At the t-th iteration, we compute negative gradients $u_{t,i}$ as in (3.12) and define a functional multi-index tree as the solution to

$$\underset{h, \beta_1, ..., \beta_P}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle \right) \right)^2, \tag{3.14}$$

where $h : \mathbb{R}^P \to \mathbb{R}$ is a decision tree. In what follows, we will introduce two strategies to approximate the solution to (3.14): one chooses the optimal set of P indices over the entire tree, and the other finds a best single index at each split. We refer to the resulting trees as Type A and Type B trees, respectively.

3.3.1 Type A tree

Type A trees take a two-level approach where we estimate h at the inner level and search for the optimal $\beta_1, ..., \beta_P$ at the outer level. It is easy to see this two-level structure if we re-state (3.14) as

$$\underset{\beta_1, ..., \beta_P}{\operatorname{argmin}} \left\{ \min_{h} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle \right) \right)^2 \right\}. \tag{3.15}$$

Given $\beta_1, ..., \beta_P$, the inner level is simply fitting a usual decision tree using the vector $(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle)^T \in \mathbb{R}^P$, $i = 1, ..., n$ as predictors. At the outer level, we find the β_1,..., β_P that minimize

$$\sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - \hat{h}(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle) \right)^2. \tag{3.16}$$

If $\beta_1, ..., \beta_P$ are unconstrained, the solution to (3.15) is not unique. For example, for any decision tree h and non-zero scalars $b_1, ..., b_P$,

$$h(\langle \cdot, \beta_1 \rangle, ..., \langle \cdot, \beta_P \rangle) = \tilde{h}(\langle \cdot, b_1\beta_1 \rangle, ..., \langle \cdot, b_P\beta_P \rangle),$$

where the value at which the j-th input of h is split is multiplied by b_j in $\tilde{h}$. To avoid this issue, we introduce conditions under which h, β_1, ..., and β_P are identifiable up to sign changes of each β_j:

Condition 1 *$\|\beta_j\| = 1$, for $j = 1, ..., P$.*

Condition 2 *Each index is chosen by at least one of the splits of a decision tree $h : \mathbb{R}^P \to \mathbb{R}$. For any set of indices $J \subset \{1, ..., P\}$, there exist a $\delta > 0$ and $x_0 \in B(x_0, \delta) = \{x | x \in \mathcal{L}, \|x - x_0\| \leq \delta\}$, such that for $x \in B(x_0, \delta)$,*

$$h(\langle L_1(x), \beta_1 \rangle ..., \langle L_j(x), \beta_j \rangle ..., \langle L_P(x), \beta_P \rangle) \tag{3.17}$$

is a non-constant function of x, where $L_j(x) = x$ if $j \in J$ and else $L_j(x) = x_0$, for $j = 1, ..., P$.

The following result shows that these conditions are sufficient to generate a unique solution to (3.15) up to sign changes of each β_j. The proof of Theorem 1 is included in the Appendix.

Theorem 1 *Suppose that Conditions 1 and 2 hold, then β_1,..., β_P are identifiable up to sign changes of each β_j. That means for any decision trees $h : \mathbb{R}^P \to \mathbb{R}$ and $\tilde{h} : \mathbb{R}^P \to \mathbb{R}$ if*

$$h(\langle x, \beta_1 \rangle ..., \langle x, \beta_P \rangle) = \tilde{h}(\langle x, \eta_1 \rangle, ..., \langle x, \eta_P \rangle) \tag{3.18}$$

hold for all $x \in \mathcal{L}$, then $\{\beta_1, ..., \beta_P\} = \{(-1)^{l_1}\eta_1, ..., (-1)^{l_P}\eta_P\}$ for some $l_1, ..., l_P \in \{0, 1\}$, and $h = \tilde{h}$.

A convenient approach to find the solution of (3.15) is to approximate the optimal directions using a flexible basis (e.g. splines). Let $\boldsymbol{\psi} = (\psi_1, ..., \psi_d)^T$ be an orthonormal set in $\mathcal{L}$ and write $\beta_j = \sum_{l=1}^{d} c_{j,l}\psi_l$ (or $\beta_j = \mathbf{c}_j^T\boldsymbol{\psi}$). Then Condition 1 becomes $\|\mathbf{c}_j\|_2 = 1$ where $\mathbf{c}_j = (c_{j,1}, ..., c_{j,d})^T$ for $j = 1, ..., P$, and $\|\cdot\|_2$ is the Euclidean norm. Condition 2 generally holds for functional multi-index trees except for very special situations, such as one of the indices being equal for all training data points.

With a slight abuse of notation, we define $\langle \cdot, \boldsymbol{\psi} \rangle = (\langle \cdot, \psi_1 \rangle, ..., \langle \cdot, \psi_d \rangle)^T$ and write the objective in (3.15) as

$$\sum_{i \in \mathcal{I}_{\text{train}}} (u_{t,i} - h(\mathbf{c}_1^T \langle x_i, \boldsymbol{\psi} \rangle, ..., \mathbf{c}_P^T \langle x_i, \boldsymbol{\psi} \rangle))^2. \tag{3.19}$$

To further simplify the computation to minimize (3.19) under the condition $\|\mathbf{c}_j\|_2 = 1$ for $j = 1, ..., P$, we represent $\mathbf{c}_j$ in a spherical coordinate system so that we can transform the optimization problem to a simpler problem with box constraints.

Referring to Blumenson (1960), Cartesian coordinates $\mathbf{c}_j$ and spherical coordinates $(r_j, \theta_{j,1}, ..., \theta_{j,d-1})$ are connected through the following equations:

$$\begin{aligned}
r_j &= \sqrt{\sum_{l=1}^{d} c_{j,l}^2} \\
c_{j,1} &= r_j \cos(\theta_{j,1}) \\
c_{j,l} &= r_j \cos(\theta_{j,l}) \prod_{k=1}^{l-1} \sin(\theta_{j,k}), \ l = 2, ..., d-1 \\
&\vdots \\
c_{j,d} &= r_j \prod_{k=1}^{d-1} \sin(\theta_{j,k}),
\end{aligned}$$

where $r_j \in [0, \infty)$, $\theta_{j,1} \in [-\pi, \pi)$, and $\theta_{j,l} \in [0, \pi]$ for $l = 2, ..., d-1$. As a result of this connection, we can transform $\mathbf{c}_j$ to $(r_j, \theta_{j,1}, ..., \theta_{j,d-1})$ and further reduce Condition 1 to $r_j = 1$ for $j = 1, ..., P$. In addition, we want to restrict $\theta_{j,1} \in [-\pi/2, \pi/2)$, which is equivalent to $c_{j,1} > 0$. This restriction ensures that (3.19) has a single unique solution as opposed to multiple solutions that are unique up to sign changes.

Finally, we treat the objective of (3.19) as a function of $\boldsymbol{\theta}_j = (\theta_{j,1}, ..., \theta_{j,d})^T$ and minimize it under conditions

$$\theta_{j,1} \in [-\pi/2, \pi/2), \ \theta_{j,2}, ..., \theta_{j,d-1} \in [0, \pi]. \tag{3.20}$$

This can be achieved by applying generic gradient-free optimization algorithms that allow box constraints.

Algorithm 5 summarizes the above-introduced procedure to fit a Type A tree. Note that the objective function in Line 8 of Algorithm 5 may not be convex and therefore multiple starting points are recommended when preforming the optimization (see Section 3.4.2 for details).

Algorithm 5: Type A tree

Input : Data $(x_i, u_{t,i}), i \in \mathcal{I}_{\text{train}}$
Number of indices P
Orthonormal basis $\boldsymbol{\psi} = \{\psi_1, ..., \psi_d\}$

Function `Loss`$(\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_P, h)$**:**

$\mathbf{c}_j = (c_{j,1}, ..., c_{j,d})^T$, for $j = 1, ..., P$, where
$c_{j,1} = \cos(\theta_{j,1})$
$c_{j,l} = \cos(\theta_{j,l}) \prod_{k=1}^{l-1} \sin(\theta_{j,k}),\ l = 2, ..., d-1$
$\vdots$
$c_{j,d} = \prod_{k=1}^{d-1} \sin(\theta_{j,k})$
return $\sum_{i \in \mathcal{I}_{\text{train}}} (u_{t,i} - h(\mathbf{c}_1^T \langle x_i, \boldsymbol{\psi} \rangle, ..., \mathbf{c}_P^T \langle x_i, \boldsymbol{\psi} \rangle))^2$

$$(\hat{\boldsymbol{\theta}}_1, ..., \hat{\boldsymbol{\theta}}_P) = \underset{\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_P}{\operatorname{argmin}} \min_h \texttt{Loss}(\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_P, h)$$

where `Loss` is minimized under conditions

$\theta_{j,1} \in [-\pi/2, \pi/2),\ \theta_{j,2}, ..., \theta_{j,d-1} \in [0, \pi]$, for $j = 1, ..., P$.

$$\hat{h}_t = \underset{h}{\operatorname{argmin}}\, \texttt{Loss}(\hat{\boldsymbol{\theta}}_1, ..., \hat{\boldsymbol{\theta}}_P, h)$$

Convert $\hat{\boldsymbol{\theta}}_1$, ..., $\hat{\boldsymbol{\theta}}_P$ to $\hat{\mathbf{c}}_1$, ..., $\hat{\mathbf{c}}_P$
$\beta_j = \hat{\mathbf{c}}_j^T \boldsymbol{\psi}$ for $j = 1, ..., P$.

Output : $\hat{h}_t(\langle \cdot, \hat{\beta}_1 \rangle, ..., \langle \cdot, \hat{\beta}_P \rangle)$

3.3.2 Type B tree

Unlike Type A trees which find K projection directions to minimize the prediction error of the whole tree, Type B trees adapt the CART algorithm (Breiman, 1984) to select an optimal direction at each split.

At each boosting iteration t, we start from $(x_i, u_{t,i})$, $i \in \mathcal{I}_{\text{train}}$ and make the first split at the root node by finding the optimal split and the projection direction that solve

$$\underset{\beta}{\operatorname{argmin}} \left\{ \min_{g \in G_1} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - g\left(\langle x_i, \beta \rangle\right)^2 \right\}, \tag{3.21}$$

where G_1 denotes the class of single split trees, also called decision stumps.

For any given β, the tree g partitions the data into two regions (nodes) where the dividing threshold is selected from $\langle x_i, \beta \rangle$, $i \in \mathcal{I}_{\text{train}}$ to minimize the squared prediction error. To make predictions for data points in each region, the average responses of the training data in that region is used.

A Type B tree can be fitted by repeatedly solving (3.21) at each split. We can view (3.21) as a special case of fitting a Type A tree with $P = 1$ and depth $d = 1$, and apply the algorithm we introduced in Section 3.3.1. Having found the first split at the root node, we repeat the same process in each of the two resulted regions, and continue until the tree reaches its specified maximum depth.

This iterative optimization procedure is straightforward, but it can be computationally costly. There is also the concern of performing optimization at nodes that contain very few examples, for instance, nodes at deep levels of the tree, as this may result in unstable estimation of β or even convergence failure of the optimization algorithm. To avoid these issues, we suggest an alternative approach to approximate the solution of (3.21).

Rather than solving (3.21) with respect to all possible β's at each split, we select β from a pool of randomly generated candidates. As we did for Type A trees, we expand β in the space of an orthonormal basis $\boldsymbol{\psi} = \{\psi_1,..., \psi_d\}$ and let $\beta = \sum_{l=1}^{d} c_{0,l}\psi_l$. We randomly generate a large pool of candidate $\mathbf{c}_0$ vectors and denote the pool as $\mathcal{C}_t = \{\tilde{\mathbf{c}}_t^{(1)}, ..., \tilde{\mathbf{c}}_t^{(Q)}\}$, where Q is the pre-specified number of candidates. Each $\tilde{\mathbf{c}}_t^{(j)} \in \mathcal{C}_t$ is an independently sampled vector $\tilde{\mathbf{c}}_t^{(j)} = (\tilde{c}_{t,1}^{(j)}, ..., \tilde{c}_{t,d}^{(j)})^T$ that satisfies $\|\tilde{\mathbf{c}}_t^{(j)}\|_2 = 1$ and $\tilde{c}_{t,1}^{(j)} > 0$. This ensures the candidate β's at the t-th iteration satisfy identifiability conditions $\|\tilde{\beta}_t^{(j)}\| = 1$ where $\tilde{\beta}_t^{(j)} = (\mathbf{c}_t^{(j)})^T\boldsymbol{\psi}$ for $j = 1, ..., Q$.

At each split, the coefficient vector of the projection direction is selected from $\mathcal{C}_t$. This corresponds to applying the CART algorithm to find a decision tree $h : \mathbb{R}^P \to \mathbb{R}$ that minimizes the squared error:

$$\underset{h}{\text{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left((\tilde{\mathbf{c}}_t^{(1)})^T \langle x_i, \boldsymbol{\psi} \rangle, ..., (\tilde{\mathbf{c}}_t^{(Q)})^T \langle x_i, \boldsymbol{\psi} \rangle \right) \right)^2. \tag{3.22}$$

The CART algorithm treats $(\tilde{\mathbf{c}}_t^{(1)})^T \langle x_i, \boldsymbol{\psi} \rangle, ..., (\tilde{\mathbf{c}}_t^{(Q)})^T \langle x_i, \boldsymbol{\psi} \rangle$ as the explanatory variables and $u_{t,i}$ as the response for the i-th individual. In order to reduce overfitting, we use different pools $\mathcal{C}_t$ randomly sampled at each boosting iteration. This is expected to introduce randomness to the algorithm and allow more directions to be considered.

We outline the procedure to fit a Type B tree in Algorithm 6. In Line 3, the number of selected indices P depends on the complexity of the decision tree, which is usually controlled by the maximum depth of the tree.

Algorithm 6: Type B tree

Input : Data $(x_i, u_{t,i}), i \in \mathcal{I}_{\text{train}}$
Number of random directions P
Orthonormal basis $\boldsymbol{\psi} = \{\psi_1, ..., \psi_d\}$

1 Uniformly sample $\tilde{\mathbf{c}}_t^{(j)} = (\tilde{c}_{t,1}^{(j)}, ..., \tilde{c}_{t,d}^{(j)})^T$, for $j = 1, ..., Q$ that each satisfies $\|\tilde{\mathbf{c}}_t^{(j)}\|_2 = 1$ and $\tilde{c}_{t,1}^{(j)} > 0$

2

$$\hat{h}_t = \underset{h}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left((\tilde{\mathbf{c}}_t^{(1)})^T \langle x_i, \boldsymbol{\psi} \rangle, ..., (\tilde{\mathbf{c}}_t^{(Q)})^T \langle x_i, \boldsymbol{\psi} \rangle \right) \right)^2$$

3 Let $\hat{\mathbf{c}}_1, ..., \hat{\mathbf{c}}_P$ be candidates selected by $\hat{h}_t$ from $\mathcal{C}_t = \{\tilde{\mathbf{c}}_t^{(1)}, ..., \tilde{\mathbf{c}}_t^{(Q)}\}$

4 $\beta_j = \hat{\mathbf{c}}_j^T \boldsymbol{\psi}$ for $j = 1, ..., P$.

Output : $\hat{h}_t(\langle \cdot, \hat{\beta}_1 \rangle, ..., \langle \cdot, \hat{\beta}_P \rangle)$

We point out that Type B trees cannot fully replace Type A trees as either type is preferred in different situations. If the target function is complex and requires a base learner with high complexity to achieve satisfactory prediction accuracy, we suggest using Type B trees, taking advantage of its fast computation and flexible structure. On the other hand, when the target function is simple, parsimonious base learners (e.g. Type A trees with few indices) are often preferred to prevent overfitting. Note that both Type A and B trees can easily include real-valued explanatory variables $\mathbf{v}_i \in \mathbb{R}^q$ by replacing $h(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle)$ in (3.14) with $h(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle, \mathbf{v}_i)$. More specifically, (3.19) becomes

$$\sum_{i \in \mathcal{I}_{\text{train}}} (u_{t,i} - h(\mathbf{c}_1^T \langle x_i, \boldsymbol{\psi} \rangle, ..., \mathbf{c}_P^T \langle x_i, \boldsymbol{\psi} \rangle, \mathbf{v}_i))^2,$$

for Type A trees, and the objective function in (3.22) becomes

$$\sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left((\tilde{\mathbf{c}}_t^{(1)})^T \langle x_i, \boldsymbol{\psi} \rangle, ..., (\tilde{\mathbf{c}}_t^{(Q)})^T \langle x_i, \boldsymbol{\psi} \rangle, \mathbf{v}_i \right) \right)^2$$

for Type B rrees. This extension allows our method to fit partial-functional regression models with mixed-type predictors. In Section 3.5, we will present an example that illustrates the usage of our method in a partial-functional setting.

3.4 Simulation studies

To assess the numerical performance of our proposed method, we conducted a simulation study comparing it with alternative functional regression methods in the literature.

3.4.1 Set up

We generated data sets $D = \{(x_i, y_i), i = 1, ..., N\}$, consisting of a predictor $x_i \in L^2(\mathcal{I})$ and a scalar response y_i that follow the model:

$$y_i = r(x_i) + C\epsilon_i, \tag{3.23}$$

where the errors ϵ_i are i.i.d $N(0, 1)$, r is the regression function, and $C > 0$ is a constant that controls the signal-to-noise ratio (SNR):

$$\text{SNR} = \frac{\text{Var}(r(X))}{\text{Var}(C\epsilon)}.$$

To sample the functional predictors x_i, we considered two models adopted from Ferraty et al. (2013) and Boente and Salibian-Barrera (2021) respectively:

- Model 1 (M_1):

  $$x_i(t) = a_i + b_i t^2 + c_i \exp(t) + \sin(d_i t),$$

  where $t \in [-1, 1]$, $a_i, b_i \sim \mathcal{U}(0, 1)$, $c_i \sim \mathcal{U}(-1, 1)$, and $d_i \sim \mathcal{U}(-2\pi, 2\pi)$; and

- Model 2 (M_2):

  $$x_i(t) = \mu(t) + \sum_{p=1}^{4} \sqrt{\lambda_j}\xi_{ij}\phi_j(t),$$

  where $t \in [0, 1]$, $\mu(t) = 2\sin(t\pi)\exp(1 - t)$, $\lambda_1 = 0.8, \lambda_2 = 0.3, \lambda_3 = 0.2$, and $\lambda_4 = 0.1$, $\xi_{ij} \sim N(0, 1)$, and ϕ_j's are the first four eigenfunctions of the "Mattern" covariance function $\gamma(s, t)$ with parameters $\rho = 3, \sigma = 1, \nu = 1/3$:

  $$\gamma(s, t) = C\left(\frac{\sqrt{2\nu}|s - t|}{\rho}\right), \; C(u) = \frac{\sigma^2 2^{1-\nu}}{\Gamma(\nu)} u^\nu K_\nu(u),$$

  where $\Gamma(.)$ is the Gamma function and K_ν is the modified Bessel function of the second kind.

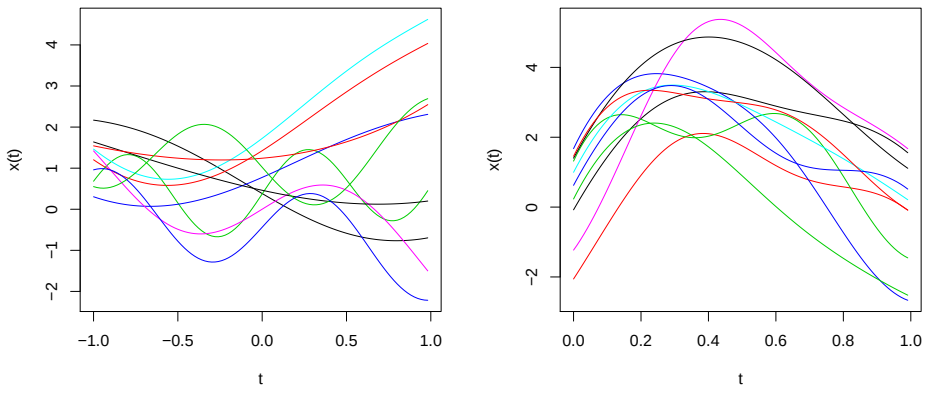


Figure 3.1: Ten random samples generated from Model 1 (left panel) and Model 2 (right panel).

For each subject i, we evaluated x_i on a dense grid $t_1, ..., t_{100}$ in $\mathcal{I} = [-1, 1]$ for M_1 and $\mathcal{I} = [0, 1]$ for M_2. Figure 3.1 shows an example of 10 randomly sampled x_i's generated from M_1 (left panel) and M_2 (right panel).

We considered five regression functions:

- $r_1(X) = \left(\int_{\mathcal{I}}(X(t) - \mu(t))(\phi_1(t) + \phi_2(t))dt\right)^{1/3}$, where the first two FPCs (ϕ_1 and ϕ_2) and mean (μ) for M_1 were calculated using a large sample of $x_i, i = 1, ..., 3000$, while for M_2 their true values were used,
- $r_2(X) = 5 \exp\left(-\frac{1}{2}\left|\int_{\mathcal{I}} x(t) \log(|x(t)|)dt\right|\right)$,
- $r_3(X) = 5\,\text{logistic}\left(2\int_{\mathcal{I}} X(t)^2 \sin(2\pi t)dt\right)$, where $\text{logistic}(u) = 1/(1 + \exp(-u))$, and
- $r_4(X) = 5\left(\sqrt{\left|\int_{\mathcal{I}_1} \cos(2\pi t^2)X(t)dt\right|} + \sqrt{\left|\int_{\mathcal{I}_2} \sin(X(t))dt\right|}\right)$, where $\mathcal{I}_1 = [-1, 0]$ and $\mathcal{I}_2 = (0, 1]$ for M_1, and $\mathcal{I}_1 = [0, 0.5]$ and $\mathcal{I}_2 = (0.5, 1]$ for M_2.

These regression functions were selected to represent a single-index model (r_1) and other nonlinear models (r_2, r_3, and r_4). To control the level of noise, we considered SNR = 5 and 20 as high and low noise levels and denoted them as S_1 and S_2, For each combination of the predictor model (M), regression function (r), and noise level (S), we generated i.i.d. samples

$(x_i, y_i), ..., (x_N, y_N)$ of size $N = 1600$. The dataset was randomly partitioned into a training set, a validation set, and a test set of size 400, 200, and 1000 respectively. In total, we simulated data from 16 settings, each defined as a combination (M, r, S) from the set $\{M_1, M_2\} \times \{r_1, ..., r_4\} \times \{S_1, S_2\}$.

3.4.2 Implementation details

For each setting, we used 100 independently generated datasets and compared the performance of the following estimators:

- `FLM1`: functional linear regression with cubic B-splines (Hastie and Mallows, 1993),
- `FLM2`: functional linear regression with FPC scores (Cardot et al., 1999),
- `FAM`: functional additive models (Müller and Yao, 2008),
- `FAME`: functional adaptive model estimation (James and Silverman, 2005),
- `FPPR`: functional projection pursuit regression (Ferraty et al., 2013),
- `FGAM`: functional generalized additive models (McLean et al., 2014),
- `TFBoost(A,P)`: tree-based functional boosting with Type A trees, where each tree is specified with $P = 1$, 2, or 3 indices, and
- `TFBoost(B)`: tree-based functional boosting with Type B trees.

`FLM1` and `FLM2` are classical functional regression methods for linear models which use different bases to estimate the regression coefficients. For `FLM1`, we used cubic B-splines with 7 basis functions (3 evenly spaced interior knots), which was found to work generally well compared to using less or more interior knots. We penalized the second derivative of the coefficient and selected the penalization parameter to minimize the mean-squared-prediction-error (MSPE) on the validation set. For `FLM2`, we used the top 4 FPCs required to explain 99% of the variance.

`FAM` constructs an additive estimator using FPC scores as predictors. We modified its implementation in the `fdapace` package (Carroll et al., 2021) to select the number of FPC scores that minimizes the MSPE on the validation set.

`FAME` estimates an extension of generalized additive models (GAM) to functional predictors. We used the code shared by the authors of James and Silverman (2005) and fitted the model using a Gaussian link function.

`FPPR` follows the principle of projection pursuit regression and assumes an additive decomposition

$$F(X) \approx r_1(\langle X, \alpha_1 \rangle) + ... + r_q(\langle X, \alpha_q \rangle). \tag{3.24}$$

In (3.24) each component r_j for $j = 1, ..., q$ is fitted by a functional single-index estimator

$$\hat{r}_j(x) = \frac{\sum_{i \in \mathcal{I}_{\text{train}}} K\left(h_j^{-1} \langle \beta_j, x - x_i \rangle\right) u_{j,i}}{\sum_{i \in \mathcal{I}_{\text{train}}} K\left(h_j^{-1} \langle \beta_j, x - x_i \rangle\right)}, \tag{3.25}$$

where $h_j \in \mathbb{R}$, $\beta_j \in L^2(\mathcal{I})$, and the residuals $u_{1,i} = y_i$ and $u_{j,i} = y_i - \sum_{l=1}^{j-1} \hat{r}_l(x_i)$ for $j = 2, ..., q$. Using the code shared by the authors of Ferraty et al. (2013) to compute a single $\hat{r}_j$, we built an estimator by adding multiple of these functional single-index estimators. Similarly to what was done with `FLM1`, for `FPPR` and `FAME` we used cubic B-splines with 7 functions (3 evenly spaced interior knots) as the basis to fit each β_j. For both methods, we selected the number of additive components between 1 and 15 to minimize the MSPE on the validation set. In our simulation settings, the performance of `FAME` and `FPPR` rarely improved beyond 15 additive components.

The implementation of `FGAM` is available from the `refund` package in R (Goldsmith et al., 2020). The method fits a model of the form

$$F(X) = \mu + \int G(X(t), t) dt,$$

where $\mu \in \mathbb{R}$, and G is estimated using bivariate B-splines with roughness penalties. A restricted maximum likelihood approach is applied to select the parameter that penalizes the second order marginal differences of the basis. See the documentation of `fgam` in the `refund` package for details of this approach. We chose to use tensor-type bivariate cubic B-splines of dimensions 15 by 15. In all our settings, this choice ensured a stable fit and almost always resulted in the best performance compared to using other B-splines bases.

We implemented `TFBoost` in R software, including `TFBoost(A.P)` for any positive integer P and `TFBoost(B)`. We fitted decision trees using the `rpart` package (Therneau and Atkinson, 2019) and performed optimization of `TFBoost(A.P)` using the `Nelder_Mead` function from the `lme4` package (Bates et al., 2015). The code implementing our method is publicly available online at `https://github.com/xmengju/TFBoost`. To make a fair comparison with its alternatives, we used cubic B-splines with 7 functions (3 evenly

spaced interior knots) as the basis for `TFBoost` methods, same as what we did for `FLM1`, `FPPR`, and `FAME`. The B-splines basis was orthonormalized as required by Type A and Type B trees.

We studied the performance of `TFBoost(A.P)` with $P = 1$, 2, and 3; and `TFBoost(B)` using 200 random directions at each iteration. For each of these methods, the maximum depth (d) of the functional multi-index trees was fixed for all iterations. We experimented with $d \in \{1, 2, 3, 4\}$ and for each method selected the value of d that achieved the lowest MSPE on the validation set at the early stopping time. We set the shrinkage parameter γ to 0.05 and the maximum number of iterations $T_{\texttt{max}}$ to 1000.

The estimation of a functional multi-index tree may result in a non-convex optimization problem. To avoid suboptimal local minima, we fitted each Type A tree with multiple starting points. More specifically, we first uniformly sampled 30 points in the spherical coordinates system that satisfied the box constrains in (3.20) and ran the Nelder-Mead algorithm for 10 steps using each of the 30 points as the starting point. We then chose the five ending points with the lowest objective values and continued running the Nelder-Mead algorithm until convergence. Out of the five resulting estimates, we chose the one that minimized the objective function.

The results for all methods under consideration were evaluated in terms of the MSPE on the test set ($\mathcal{I}_{\text{test}}$):

$$\text{MSPE} = \frac{1}{|\mathcal{I}_{\text{test}}|} \sum_{i \in \mathcal{I}_{\text{test}}} (\hat{F}(x_i) - y_i)^2. \tag{3.26}$$

For `TFBoost`, the estimate $\hat{F}$ was reported at the early stopping time.

3.4.3 Results and discussion

For each combination of the regression function $(r_1, ..., r_4)$ and the predictor model (M_1, M_2), the results for the high noise (S_1) settings look very similar as those for the low noise (S_2) settings. In S_2 settings, the differences between estimators are more pronounced, showing a more notable advantage of `TFBoost(A.P)` and `TFBoost(B)` over the others. We report here the results for S_1 settings and provide those for S_2 in the Appendix.

Figures 3.2 to 3.5 show the MSPEs on the test sets for r_1 to r_4 respectively. Since `FLM1`, `FLM2`, and `FAM` tend to produce very large errors, to be able to visualize the differences among the other methods, we excluded them from the figures and reported the summary statistics of their MSPEs in the Appendix. As expected, the linear estimators (`FLM1` and `FLM2`) do not

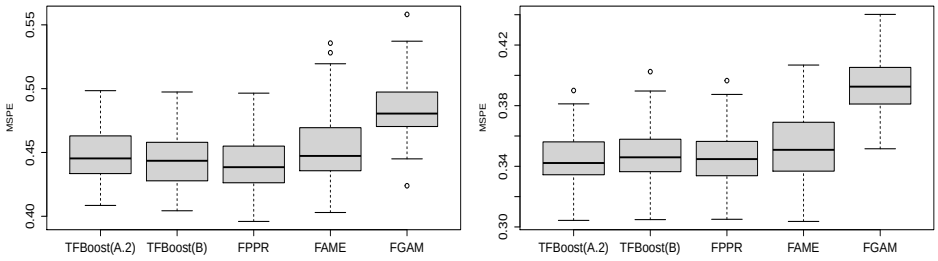
work well since r_1 to r_4 are nonlinear functions. `FAM` also performs poorly, likely due to its using one FPC as the projection direction for each additive component, making it less flexible compared with estimating the projection direction based on the data.

In each figure, panel (a) corresponds to data with the predictors generated from M_1 and panel (b) for those from M_2. Each panel shows the boxplot of MSPEs on the test sets from 100 independent runs of the experiment. For `TFBoost(A.P)`, we show the MSPEs for the value of P that gave the best results on the test sets, and include the rest in the Appendix. In the case of a tie we picked the simpler model (one with a smaller P). As a result, we selected

- $P = 2$ for all r_1 and r_3 settings, and (r_2, M_2),
- $P = 3$ for all r_4 settings and (r_2, M_1).

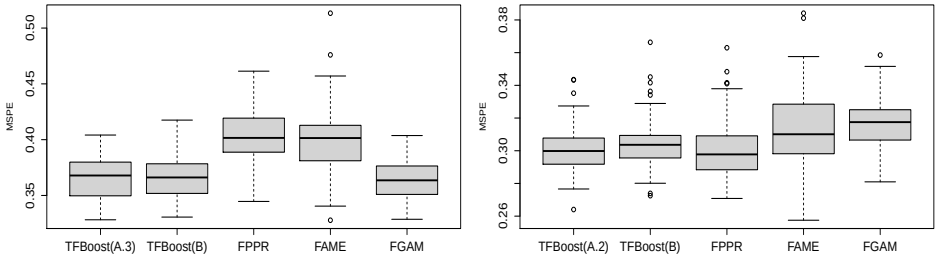
In general, `TFBoost(A.P)` and `TFBoost(B)` show stable and remarkable predictive performance in all these figures, with either or both achieving the lowest or second lowest average test MSPEs amongst all. In contrast, the performance of `FPPR`, `FAME`, and `FGAM` heavily depends on the regression function as well as the predictor model. For example, in panel (a) of Figure 3.3 where X was generated from M_1, `FPPR` produces the worst errors compared to the other methods. However, when X followed M_2 in panel (b), `FPPR` becomes one of the best. The opposite holds for `FGAM`, being among the best in panel (a) and the worst in panel (b). Similarly in Figure 3.4, the performance of `FGAM` differs greatly in M_1 and M_2 settings. In Figures 3.2 and 3.5, the performances of `FAME` and `FGAM` are similar in both M_1 and M_2 settings, but they differ substantially across figures. `FGAM` produces significantly worse errors compared to `FAME` in Figure 3.2, whereas in Figure 3.5 we observe the opposite.

It is worth noting that `TFBoost` methods are close to their alternatives in terms of test errors when the true model matches the one on which the other methods are based. For example, in Figure 3.2 `TFBoost(A.2)` and `TFBoost(B)` produce similar errors as `FPPR`, where the regression function r_2 matches with the true model of `FPPR` specified with a single additive component. In other cases where the alternatives deviate from the true model, `TFBoost(A.P)` and `TFBoost(B)` usually show the lowest errors and their performances are stable regardless of the setting. This illustrates the flexibility of our method, which helps to achieve low prediction errors without posing strong assumptions on the target function.



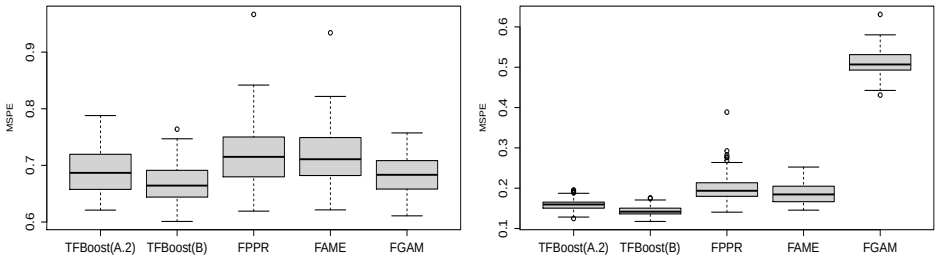
(a) M_1 (b) M_2

Figure 3.2: Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_1, M_1, S_1) in panel (a) and (r_1, M_2, S_1) in panel (b).



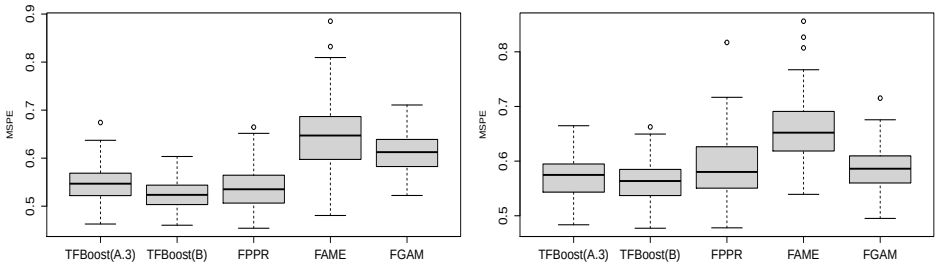
(a) M_1 (b) M_2

Figure 3.3: Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_2, M_1, S_1) in panel (a) and (r_2, M_2, S_1) in panel (b).



(a) M_1 (b) M_2

Figure 3.4: Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_3, M_1, S_1) in panel (a) and (r_3, M_2, S_1) in panel (b).



(a) M_1 (b) M_2

Figure 3.5: Boxplots of MSPEs on test sets from 100 runs of the experiment with data generated from (r_4, M_1, S_1) in panel (a) and (r_4, M_2, S_1) in panel (b).

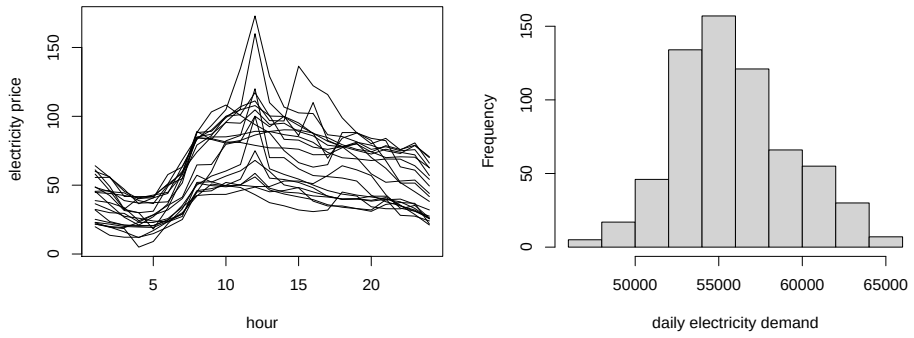
So far we have focused on the performance of `TFBoost` in nonlinear settings since the base learners are nonlinear. In the Appendix, we provide also results for an experiment fitting `TFBoost` methods with data generated from a linear model and comparing them with alternatives considered previously. As expected, the linear estimators `FLM1`, `FLM2` outperform the others. For `FGAM`, we note that the linear regression function matches with its model assumptions, which explains its good performance in linear settings. We note that the test errors of `TFBoost(A.1)` are only slightly worse than those produced by linear estimators: about 5% worse in the M_1 setting, and about 7% worse in the M_2 setting. This implies that even though `TFBoost` is an estimator for possibly nonlinear regression functions, we do not lose much applying it to data generated from linear models.

3.5 German electricity data

To further illustrate the use of `TFBoost`, we analyzed a data set consisting of electricity prices traded at the European Energy Exchange (EEX) in Leipzig and electricity demand reported by the European Network of Transmission System Operators for Electricity from January 1st 2006 to September 30th 2008. On each day, electricity prices and demands were recorded hourly and represented as vectors of dimension 24. Our objective is to predict the daily average of electricity demand using hourly reported electricity prices.

The whole dataset is available from the on-line supplementary materials of Liebl et al. (2013). To avoid days with known atypical price or demand values affecting our analysis, we removed weekends and holidays, and analyzed data collected on the remaining 638 days. For each of these days, we view the hourly reported electricity prices as discretized evaluations of a smooth price function. We show the data set in Figure 3.6, where the left panel plots 20 random samples of the electricity price curves and the right panel plotted the distribution of the daily electricity demand for the whole data set.

Same as in Section 3.4, we considered `TFBoost(A,P)`, for $P = 1, 2$, and 3, `TFBoost(B)`, and their alternatives `FLM1`, `FLM2`, `FAM`, `FPPR`, `FAME`, and `FGAM`. To adjust for potential seasonality effects, we created a "day of the year" variable defined as the number of days from January 1st of the year in which the data were collected. In order to compare the results before and after adjusting for seasonality, our analysis consisted of two parts: (1) predicting daily electricity demand with hourly electricity prices only, and (2) with both hourly electricity prices and the "day of the year" variable.



(a) (b)

Figure 3.6: Visualization of German electricity data with 20 random samples of hourly electricity price curves displayed in panel (a) and histogram of daily average electricity demand of the whole data set in panel (b).

In part (1), all methods under consideration were specified in the same way as in Section 3.4.2. In part (2), we added "day of the year" as an additional predictor to all methods and adjusted their algorithms accordingly. For `TFBoost` methods, at every boosting iteration, we added "day of the year" as an additional variable to the functional multi-index tree. For the competitors, we included the scalar predictor in the model following the same approach as we did for the functional predictor. We added "day of the year" as a linear term in `FLM1` and `FLM2`, and as a nonparametric term in `FAM`, `FPPR`, `FAME`, and `FGAM`. For each of `FAM` and `FPPR`, we first calculated the estimator in the same way as in part (1) and obtained the residuals. Then we added to the part (1) estimator a Nadaraya-Watson estimator with Gaussian kernel fitted to the residuals using "day of the year" as the predictor. For `FAME` and `FGAM` we specified a smooth term on "day of the year" represented as penalized cubic splines (Wood, 2017). Other than adding this additional predictor, the other parameters and fitting procedures were kept the same as in part (1).

We randomly partitioned the data into a training set (60%), a validation set (20%), and a test set (20%). As in Section 3.4, we fitted each model using the training and validation sets and recorded MSPE on the test set. The validation set was used to choose the regularization parameter of `FLM1`, the number of additive components of `FAM` and `FPPR`, and the maxi-

mum tree depth and early stopping time of `TFBoost(A,1)`, `TFBoost(A,2)`, `TFBoost(A,3)`, and `TFBoost(B)`.

To reduce the variability introduced by partitioning the data, we repeated the experiment with 100 random data splits. Figures 3.7 and 3.8 show test MSPEs obtained from 100 random data splits in part (1) and (2) of the experiment respectively. The summary statistics of these test MSPEs are provided in Table 3.1.

To better reveal the differences between the estimators under consideration, in Figures 3.7 and 3.8 the y-axes are truncated and represent test MSPE $\times 10^{-6}$. In both figures, methods of `TFBoost(A.1)`, `TFBoost(A.2)`, `TFBoost(A.3)`, and `TFBoost(B)` show superior performance compared to their alternatives. They achieve the lowest errors with the smallest standard errors. In Figure 3.7, we observe that `TFboost(B)` produces the smallest test errors, substantially better than `FLM1`, `FLM2`, `FAM`, and `FGAM`. The performance of `FPPR` is the closest to `TFBoost(B)`, however, it is much more expensive to fit kernel estimators compared to Type B trees, particularly with data of a large sample size. Comparing Figure 3.8 with Figure 3.7, we observe improvements in test errors for all methods except for `FLM1`. This implies the importance of adjusting for seasonality when predicting the electricity demand, which is likely due to higher electricity usage in the summer for cooling and in the winter for heating. In Figure 3.8, `TFBoost(A.1)`, `TFBoost(A.2)`, `TFBoost(A.3)` produce very similar results, and `TFBoost(A.2)` outperforms the other two by a small margin in terms of the average test MSPE.

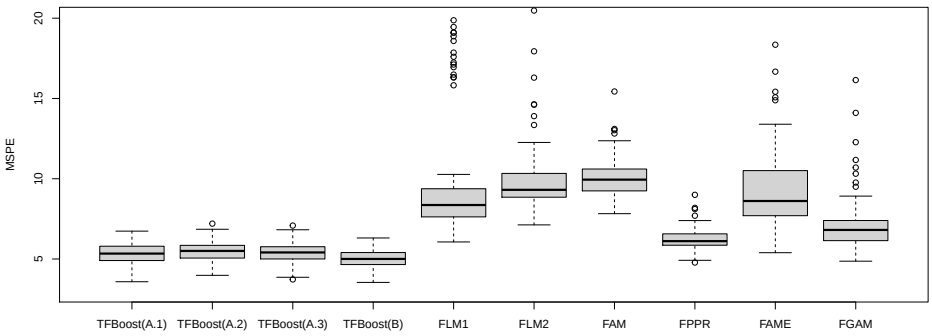


Figure 3.7: Boxplot of test MSPEs obtained from 100 random splits of the data from part (1) of the experiment. The unit of y-axis is 10^{-6}.

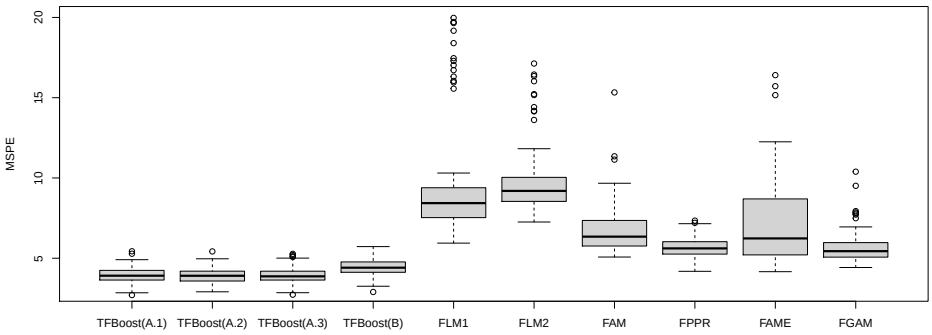


Figure 3.8: Boxplot of test MSPEs obtained from 100 random splits of the data from part (2) of the experiment. The unit of y-axis is 10^{-6}.

Methods	Part (1)	Part (2)
TFBoost(A.1)	5.333 (0.633)	3.880 (0.480)
TFBoost(A.2)	5.370 (0.627)	3.828 (0.446)
TFBoost(A.3)	5.259 (0.615)	3.887 (0.472)
TFBoost(B)	4.903 (0.568)	4.353 (0.513)
FLM1	10.614 (4.851)	10.679 (5.068)
FLM2	13.040 (11.654)	11.487 (6.058)
FAM	12.667 (16.582)	9.476 (17.453)
FPPR	6.235 (0.716)	5.721 (0.704)
FAME	13.888 (15.054)	8.581 (6.617)
FGAM	7.406 (2.202)	5.756 (1.050)

Table 3.1: Summary statistics of test MSPEs displayed in the form of mean (sd). The unit of MSPEs is 10^{-6}.

3.6 Summary

In this chapter we propose a novel boosting method for regression with a functional explanatory variable and a scalar response. Our approach uses functional multi-index trees as base learners, which offer flexibility and are relatively easy to fit. To compute these trees, we introduce two algorithms where either one is preferred in different situations: Type A trees find opti-

mal projections over the entire tree, while Type B trees search for a single optimal projection at each split. For parsimonious regression functions that suit simple base learners, we suggest Type A trees with few indices since they are less prone to overfitting. Otherwise, we recommend Type B trees to take advantage of their fast computation and flexible structure.

Compared with widely studied single-index estimators, our multi-index estimator enables modelling possible interactions between indices, allowing us to better approximate complex regression functions. Through extensive numerical experiments, we demonstrate that our method consistently produces one of the lowest prediction errors across different settings, while available alternatives can be seriously affected by model misspecification. `TFBoost` with Type B learners exhibits significant computational advantages over kernel-based methods, being able to provide accurate predictions at a much lower cost. In addition, we use a real example to show that our method compares favorably to existing methods with fully functional and partial-functional explanatory variables.

The methodology of `TFBoost` suggests interesting directions for next steps. In our experiments we only studied `TFBoost` applied with the squared loss. In cases where data contain outliers, it will be useful to consider fitting our method with different loss functions. In Chapter 4 we will study this problem and present several options to robustify `TFBoost`. Furthermore, in Chapter 5, we will provide ideas to generalize `TFBoost` to data with multiple or sparsely observed functional predictors.

Chapter 4

Robust boosting for functional regression

Tree-based functional boosting studied in Chapter 3 provides a way to fit a nonparametric regression model with data containing a functional predictor and a scalar response. In practical situations, measurements of the response and the functional predictor may either or both contain outlying values, which can cause serious damage to the regression estimator if not taken into consideration. In this chapter, we provide options to robustify tree-based functional boosting and investigate their performance through numerical experiments. In Section 4.1 we review existing robust functional regression methods, most of which were proposed for linear models. Following that, in Section 4.2 we present two robust tree-based functional boosting algorithms derived from M-estimators and MM-estimators respectively. In Section 4.3 we conduct simulation studies to compare these estimators with available robust functional regression methods and non-robust regression estimators that performed well in Chapter 3. In Section 4.4, we perform a case study on a real data set predicting the longevity of fruit flies based on their egg laying pattern. Lastly, we summarize our findings in Section 4.5.

4.1 Robust functional regression

In the context of scalar-on-function regression, consider a general model

$$Y = F(X) + \epsilon, \tag{4.1}$$

where X is in a Hilbert space $\mathcal{L}$, $F : \mathcal{L} \to \mathbb{R}$, and ϵ denotes the random error. If F is a linear function, (4.1) leads to the well-known functional linear model, which assumes $F(X) = \mu + \langle X, \beta \rangle$ for some unknown coefficient function $\beta \in \mathcal{L}$ and $\mu \in \mathbb{R}$ (also see Section 3.1.1). If the functional form for F is unknown, (4.1) implies a nonparametric model that can be estimated using kernels or splines (see Section 3.1.2). If F combines linear (or more generally parametric) and nonparametric components, (4.1) becomes a semi-parametric model often formulated as index models (see Section 3.1.3).

For linear, nonparametric, and semi-parametric models, one typically deals with estimating the conditional mean $E[Y|X]$, which is the minimizer of the expected squared loss $E[(Y - F(X))^2]$ over F from a pre-defined class of functionals. This procedure can be viewed as an extension of the classical (least squares) methods for multivariate regression. Like the classical methods, the resulting estimators are sensitive to outliers, where a single atypical observation can seriously affect the quality of these estimators. In recent years, robust alternatives have been studied to limit the impact of outlying values on functional regression. Below we will introduce these proposals for linear, nonparametric, and index-based regression models, respectively.

4.1.1 Robust functional linear regression

Research in the field of robust functional regression are concerned mainly with linear models. Suppose that we observe training data $(x_i, y_i), i = 1, ..., n$, where $x_i \in \mathcal{L}$ and $y_i \in \mathbb{R}$, and the regression function F takes a linear form: $F(X) = \mu + \langle X, \beta \rangle$, for $X, \beta \in \mathcal{L}$ and $\mu \in \mathbb{R}$. For estimating β and μ, some authors considered an M-regression problem

$$\underset{\beta \in \mathcal{L}, \mu \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i - \mu - \langle x_i, \beta \rangle), \tag{4.2}$$

where the loss function ρ is symmetric, $\rho(0) = 0$, and $\rho(z)$ is a nondecreasing function of $|z|$.

Huang et al. (2014) and Qingguo (2017) considered using convex loss functions as ρ, such as the L_1 loss and Huber's loss. They expanded the coefficient function β in terms of the FPC basis and transformed (4.2) into a finite dimensional regression problem.

If the loss function is defined as $\rho(u) = |u| + (2\tau - 1)u$ for some $\tau \in (0, 1)$, then (4.2) becomes functional quantile regression that estimates the τ-th conditional quantile of the response. When $\tau = 0.5$, this corresponds to finding the conditional median that minimizes the L_1 loss. To estimate β, Cardot et al. (2005) proposed a spline estimator, the smoothness of which is controlled by penalizing its derivatives. Kato (2012) suggested a FPC basis estimator for β and studied different criteria to choose the number of FPCs. For the same purpose to fit the conditional quantiles, Chen and Müller (2012) took an indirect approach to first estimate the conditional distribution function and then invert it to obtain the quantiles. They assumed a generalized linear model,

$$P(Y \leq y|X) = g^{-1}(\mu(y) + \langle x_i, \beta(y)\rangle),$$

where g is a known link function and $\beta(y) \in \mathcal{L}$ is estimated using FPCs.

Unlike the methods that use spline or FPC basis for estimation, Shin and Lee (2016) assumes β resides in a reproducing kernel Hilbert space (RKHS) and considered the problem

$$\underset{\mu\in\mathbb{R},\beta\in\mathcal{L}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho\left(\frac{y_i - \mu - \langle x_i, \beta\rangle}{\hat{\sigma}}\right) + \lambda J(\beta), \tag{4.3}$$

where ρ is a loss function, J is a penalization function on β, and $\lambda > 0$ is a penalization parameter. By the representer theorem (see e.g., Wahba (1990)), the explicit form of the minimizer of (4.3) can be derived. The auxiliary scale estimator $\hat{\sigma}$ is required to make $\hat{\beta}$ scale invariant, and it is suggested to be computed using residuals from fitting (4.3) with $\rho(u) = |u|$. This procedure to is similar to what is usually done to compute M-regression estimators for linear regression.

Several authors proposed to construct estimators following the principles of MM-estimation (Yohai, 1987). Maronna and Yohai (2013) considered a model of the same form as (4.3), where ρ is a bounded function, and $J(\beta)$ penalizes the roughness of the solution. The auxiliary residual scale $\hat{\sigma}$ is an M-scale defined as the solution to

$$\frac{1}{n}\sum_{i=1}^{n} \rho_0\left(\frac{y_i - \mu - \langle x_i, \beta\rangle}{\hat{\sigma}}\right) = \kappa, \tag{4.4}$$

where $\kappa \in (0, 1]$ controls the breakdown point of $\hat{\sigma}$ in the same way as it does for regression MM-estimators (see Section 2.1.1). The tuning constant of ρ_0 is selected to make $\hat{\sigma}$ consistent for Gaussian distributed residuals. When J is defined in a certain way (see Maronna and Yohai (2013)), the problem of (4.3) has an explicit solution after expanding β in a spline basis. Kalogridis and Van Aelst (2021) studied a more flexible class of methods, where the penalization J in (4.3) can be customized for different basis functions, and $\hat{\sigma}$ is defined in the same way as (4.4).

Kalogridis and Van Aelst (2019) considered an estimator with penalty on the second-derivative of β:

$$\underset{\beta\in\mathcal{L},\mu\in\mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} \rho\left(\frac{y_i - \mu - \langle x_i, \beta\rangle}{\hat{\sigma}}\right) + \lambda||\beta''||^2, \tag{4.5}$$

where $\lambda > 0$, $||\beta''||^2 = \langle \beta'', \beta'' \rangle$, and the residual scale is defined in (4.4). The coefficient β is estimated using the FPC basis where the number of basis functions is selected by robust cross-validation. Compared to the unpenalized estimator (corresponding to $\lambda = 0$ in (4.5)), this penalized estimator yields smoother estimates and shows better overall performance.

More recently, the development of functional linear regression methods leads to more complex estimators dealing with partial-functional settings with data containing functional and scalar predictors. These methods incorporate the functional predictor as a linear term, same as the aforementioned proposals, and include the scalar predictors as linear or nonparametric terms. We list Tang and Cheng (2014), Huang et al. (2015), Qingguo and Kong (2017), and Boente et al. (2020) as references. The first three were derived from M-estimators, and the last one from MM-estimators.

4.1.2 Robust functional nonparametric regression

In terms of nonparametric methods for robust functional regression, particular interest is devoted to kernel estimators. Same as in 3.1.2, we assume that x_i takes values in a semi-metric space $\mathcal{L}$ and $y_i \in \mathbb{R}$. For a training set $(x_i, y_i), i = 1, ..., n$, several authors considered a robust functional Nadaraya-Watson estimator

$$\hat{F}(x) = \underset{c}{\operatorname{argmin}} \sum_{i=1}^{n} K\left(h^{-1} s(x, x_i)\right) \rho(y_i - c), \tag{4.6}$$

or equivalently $\hat{F}(x)$ that satisfies

$$\sum_{i=1}^{n} K\left(h^{-1} s(x, x_i)\right) \psi(y_i - \hat{F}(x)) = 0,$$

where $h > 0$, K is a kernel function, and s is a semi-metric that measures the similarity between functions. As before, ρ is a loss function and ψ is the derivative of ρ.

When $\rho(u) = u^2$, we obtain the original functional Nadaraya-Watson estimator (Ferraty and Vieu, 2002) that is not resistant to outliers. To make the estimator more robust, we can use the L_1 loss $\rho(u) = |u|$ to estimate the conditional median. More generally, this approach extends to estimating the τ-th conditional quantile with the quantile loss $\rho(u) = |u| + (2\tau - 1)u$, which has been studied by Crambes et al. (2008), Chen and Zhang (2009), Laksaci et al. (2009) and Gheriballah et al. (2013).

Naturally, the Nadaraya-Watson estimator defined by (4.6) can be extended to local-linear estimators that solve

$$(\hat{a}, \hat{b}) = \underset{a\in\mathbb{R}, b\in\mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{n} K\left(h^{-1} s(x, x_i)\right) \rho(y_i - a - b\langle x_i, x\rangle). \tag{4.7}$$

where the estimator is given by $\hat{F}(x) = \hat{a}$. Compared to (4.6), the local-linear estimators tend to be more flexible, but require a higher computation cost. Belarbi et al. (2018) studied local-linear estimators computed with a convex loss function ρ. Kaid and Laksaci (2017) and Al-Awadhi et al. (2019) considered using the quantile loss ρ to estimate the conditional quantile of the response.

Ferraty et al. (2005) proposed another kernel estimator that indirectly models the cumulative distribution function of the response. Let H be a cumulative kernel such that $H\left(h_H^{-1}(y - y_i)\right)$ with the bandwidth $h_H > 0$ estimates $P(y_i \leq y)$. The kernel estimator is defined as

$$\hat{F}(y|x) = \frac{\sum_{i=1}^{n} K\left(h^{-1} s(x, x_i)\right) H\left(h_H^{-1}(y - y_i)\right)}{\sum_{i=1}^{n} K\left(h^{-1} s(x, x_i)\right)},$$

where h, K is a kernel function, and s is a semi-metric. The conditional quantiles can be obtained by inverting $\hat{F}(y|x)$.

Lastly, Crambes et al. (2013) proposed a conditional quantile estimator that can be computed via Support Vector Machines. Let $\mathcal{H}_K$ be the RKHS of a kernel K, the estimator solves

$$\underset{F\in\mathcal{H}_K}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i - F(x_i)) + \lambda ||F||^2, \tag{4.8}$$

where $\lambda > 0$, ρ is the quantile loss, and $||F||$ is a norm defined in $\mathcal{H}_K$. The explicit form of the solution to (4.8) can be obtained by the representer theorem (see e.g., Wahba (1990)).

4.1.3 Robust functional index regression

As far as we know, very few studies have investigated semiparametric models for robust functional regression. We found two recent proposals: one for single-index regression, and the other one for multi-index regression.

Sang and Cao (2020) considered a single-index model to estimate the conditional quantiles. The proposed estimator $\hat{F}(x) = \hat{r}(\langle x, \hat{\beta}\rangle)$ solves

$$\underset{\beta, r}{\operatorname{argmin}} \sum_{i=1}^{n} \rho(y_i - r(\langle x, \beta\rangle)) + \lambda J(r), \tag{4.9}$$

over $\beta \in \mathcal{L}$ and r in a Reproducing Kernel Hilbert Space. In (4.9), ρ is the quantile loss, $\lambda > 0$ is a regularization parameter, and J penalizes the complexity of r.

Wang et al. (2017) proposed a robust functional sliced inverse regression method for a multi-index model. Consider a P-index model

$$F(X) = r(\langle X, \beta_1 \rangle, ..., \langle X, \beta_P \rangle, \epsilon), \tag{4.10}$$

that satisfies the linearity assumption: let $\mathbf{B} = (\langle X, \beta_1 \rangle, ..., \langle X, \beta_P \rangle)^T$, there exist $\mathbf{c} \in \mathbb{R}^K$ such that

$$E(\langle X, \beta \rangle | \mathbf{B}) = \mathbf{c}^T \mathbf{B}$$

for any square integrable function β. The error ϵ in (4.10) is independent of X with mean zero and constant variance. Under these model assumptions, the index coefficients β_1, ..., β_K can be estimated based on the mean and covariance of X.

When data contain outlying values, Wang et al. (2017) suggested to robustly estimate the mean and covariance of X using a trimmed approach (4.10). Note that their implementation addresses the issue of having outlying values in the functional predictors, but not in the response.

4.2 Robust tree-based functional boosting

As we have introduced previously in Section 4.1, most robust functional regression methods were proposed for linear or nonparametric regression. In practice, linear models are rather restricted and can lead to misleading results when the model is misspecified. In comparison, nonparametric models offer greater flexibility with little assumptions on the target function.

Existing proposals to robustly fit a nonparametric regression model all use kernels, which rely on a semi-metrics to measure the proximity between functional objects. The choice of the semi-metric is crucial to the performance of these methods. Without prior knowledge, it is often difficult to know which one may work well. In addition, for kernel methods, one needs to determine the optimal bandwidth very carefully, which is usually conducted through a data-driven procedure (e.g. cross-validation) that can be computationally costly.

To address these issues, we propose alternatives that extend the tree-based functional boosting (TFBoost) introduced in Chapter 3 to perform robust functional regression. Many methods reviewed in Section 4.1 considered convex loss functions, particularly the quantile loss function, but it is

well-known that these loss functions are not robust to high-leverage outliers. Unlike these methods, we study two extensions of `TFBoost` that cover the use of bounded loss functions, one extension derived from M-estimators and the other one from MM-estimators. The latter can be viewed as a combination of `RRBoost` (see Chapter 2) and `TFBoost` (see Chapter 3).

We call these robust variants `RTFBoost`, which inherits the strengths of the original `TFBoost`. Compared to kernel estimators, our methods do not require semi-metrics or bandwidth selection. When applied with Type B trees, our method is very fast to compute and scalable to data with a large sample size.

Consider a nonparametric regression model:

$$Y = F(X) + \epsilon, \tag{4.11}$$

where $Y \in \mathbb{R}$, X is in a Hilbert space $\mathcal{L}$, and the error ϵ is independent of X. For a data set $D = \{(x_i, y_i), i \in \mathcal{I}_{\text{train}} \cup \mathcal{I}_{\text{val}}\}$, where $x_i \in \mathcal{L}$ and $y_i \in \mathbb{R}$, we assume that most data points are generated from model (4.11). The data set is allowed to be inhomogeneous such that it can include atypical values (in the predictor or the response, or both), or heavy-tail distributed errors. Our primary interest is to provide a reliable estimate of F that fits the majority of the data.

To achieve this goal, we study two extensions of the `TFBoost` estimator, both formulated as an ensemble of multi-index tree learners. Compared with available robust index-based methods introduced in Section 4.1.3, our methods use bounded loss functions and can deal with vertical or high-leverage outliers at the same time. The two methods, which we call `RTFboost (LAD-M)` and `RTFboost (RR)` respectively, are presented as follows.

4.2.1 RTFBoost (LAD-M)

Similarly to what M-estimation does in finite-dimensional settings (see Section 2.1), for data with functional predictors, we formulate a problem to minimize the empirical risk defined by a loss function ρ:

$$R(G) = \sum_{i \in \mathcal{I}_{\text{train}}} \rho\left(\frac{y_i - G(x_i)}{\hat{\sigma}}\right), \tag{4.12}$$

where $\hat{\sigma}$ is an auxiliary residual scale estimator, and ρ needs to satisfy: $\rho(z)$ is a nondecreasing function of $|z|$ and $\rho(0) = 0$.

Like `TFBoost`, we assume that G can be approximated by a sum of multi-index terms:

$$G(X) \approx r_1(\langle X, \beta_{1,1}\rangle, ..., \langle X, \beta_{1,P}\rangle) + ... + r_T(\langle X, \beta_{T,1}\rangle, ..., \langle X, \beta_{T,P}\rangle), \quad (4.13)$$

where P is the number of indices, and $\beta_{j,1}, ..., \beta_{j,P} \in \mathcal{L}$ are the index coefficients for the j-th term.

When fitting a regression M-estimator, it is common to first compute $\hat{\sigma}$ from a preliminary robust fit that does not require estimating the scale, typically the LAD fit that minimizes the L_1 loss. We adapt the same practice to `TFBoost` and construct a robust algorithm consisting of two stages:

- **LAD-stage**: apply `TFBoost` to minimize the L_1 loss, corresponding to letting $L(a, b) = |a - b|$ in Algorithm 4;
- **M-stage**: apply `TFBoost` to minimize $R(G)$ defined in (4.12), where $\hat{\sigma}$ and the initial function estimator are obtained from the **LAD-stage**.

The resulting algorithm called `RTFBoost(LAD-M)` is summarized in Algorithm 7.

From the LAD-stage, the robust scale estimator $\hat{\sigma}$ is taken as the MAD of the residuals. For simpler notation, we let $r_i = \hat{F}_{T_1}(x_i) - y_i$, $\hat{\mu} = \text{median}_{j \in \mathcal{I}_{\text{train}}} r_j$, and define

$$\hat{\sigma} = \text{MAD}(r_i) = 1.438 \underset{i \in \mathcal{I}_{\text{train}}}{\text{median}} |r_i - \hat{\mu}|.$$

The M-stage boosting fit is given by

$$\hat{F} = \underset{G}{\text{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \rho\left(\frac{y_i - G(x_i)}{\hat{\sigma}}\right). \quad (4.14)$$

The tuning constant of ρ (and its derivative ψ) is chosen for the corresponding location model to achieve a desired asymptotic efficiency under Gaussian errors. At every boosting iteration, we compute the negative gradient in Line 10, fit a base learner in Line 11, calculate the optimal step size in Line 12, and finally update the regression estimator in Line 13.

To avoid overfitting, we include the shrinkage parameter $\gamma \in (0, 1]$ in Line 13 and use early stopping to determine the number of iterations T_1 and T_2. For the latter, we define the early stopping time for each stage as the iteration that achieves the lowest loss on a validation set $\mathcal{I}_{\text{val}}$, which is set aside from the training set $\mathcal{I}_{\text{train}}$. The early stopping times are given by

$$T_{1,\text{stop}} = \underset{t=1,...,T_{1,\max}}{\text{argmin}} \sum_{i \in \mathcal{I}_{\text{val}}} |y_i - \hat{F}_t(x_i)|$$

for the LAD-stage, and

$$T_{2,\text{stop}} = \underset{t=1,\ldots,T_{2,\max}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{val}}} \rho\left(\frac{y_i - \hat{F}_{T_1+t}(x_i)}{\hat{\sigma}}\right),$$

for the M-stage, where $T_{1,\max}$ and $T_{2,\max}$ are the maximum number of iterations allowed in each stage.

Algorithm 7: RTFBoost (LAD-M)

Input : A data set $(x_i, y_i), i \in \mathcal{I}_{\text{train}}$
Number of iterations of the **LAD-stage** T_1
Number of iterations of the **M-stage** T_2
Shrinkage parameter $\gamma \in (0, 1)$
Number of indices P

LAD-stage: Run `TFBoost` with $L(a, b) = |a - b|$ for T_1 iterations to obtain $\hat{F}_{T_1}(\mathbf{x}_i)$, and $\hat{\sigma}$

M-stage :

for $t = 1 : T_2$ **do**

$u_{t,i} = \frac{1}{\hat{\sigma}}\psi\left(\frac{y_i - \hat{F}_{T_1+t-1}(x_i)}{\hat{\sigma}}\right)$

$\hat{h}_t(\langle\cdot, \hat{\beta}_{t,1}\rangle, ..., \langle\cdot, \hat{\beta}_{t,P}\rangle) = \underset{h,\beta_1,\ldots,\beta_K}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left(\langle x_i, \beta_1\rangle, ..., \langle x_i, \beta_P\rangle\right)\right)^2$

$\hat{\alpha}_t = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \rho\left(\frac{y_i - \hat{F}_{T_1+t-1}(x_i) - \alpha\hat{h}_t(\langle x_i, \hat{\beta}_{t,1}\rangle, ..., \langle x_i, \hat{\beta}_{t,P}\rangle))}{\hat{\sigma}}\right)$

$\hat{F}_{T_1+t}(x) = \hat{F}_{T_1+t-1}(x) + \gamma\hat{\alpha}_t\hat{h}_t(\langle x, \hat{\beta}_{t,1}\rangle, ..., \langle x, \hat{\beta}_{t,P}\rangle))$

end

Output : $\hat{F}_{T_1+T_2}(x)$

4.2.2 RTFBoost (RR)

As we have introduced in Section 2.1, M-regression estimators with convex loss functions are not robust to high-leverage outliers. Similarly, the LAD-stage of `RTFBoost(LAD-M)` can be affected by high-leverage outliers due to the use of the L_1 loss. To address this issue, we consider a different way to derive a boosting algorithm by adopting the idea of MM-estimators,

which are known to be robust to high-leverage outliers and can achieve high robustness and high efficiency at the same time. The resulting algorithm can be viewed as a combination of `RRBoost` and `TFBoost`, and it involves two stages:

- **S-stage**: compute a `TFBoost` estimator with high robustness but possibly low efficiency to minimize an M-scale of the residuals;
- **M-stage**: apply `TFBoost` to minimize $R(G)$ defined in (4.12), where $\hat{\sigma}$ and the initial function estimator are obtained from the **S-stage**.

Note that the M-stage of `RTFBoost(RR)` is computed in the same way as that of `RTFBoost(LAD-M)`. To avoid redundancy, we describe in detail the S-stage and include the M-stage in the pseudocode of Algorithm 8.

We define the S-stage estimator:

$$\hat{F} \;=\; \underset{F\in\mathcal{F}}{\operatorname{argmin}}\, \hat{\sigma}(F)\,,$$

where $F : \mathcal{L} \to \mathbb{R}$, and $\hat{\sigma}(F)$ is an M-scale that satisfies

$$\frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i\in\mathcal{I}_{\text{train}}} \rho_0\left(\frac{y_i - F(\mathbf{x}_i)}{\hat{\sigma}(F)}\right) = \kappa\,.$$

where $\kappa \in (0, 1]$ controls the breakdown point of $\hat{\sigma}(F)$, and ρ_0 is a symmetric differentiable bounded loss function. The derivative of ρ_0 is denoted as ψ_0.

Following the derivation of `RRBoost`, at the t-th iteration, the negative gradient for the i-th subject is given by

$$\begin{aligned} u_{t,i} \;&=\; -\frac{\partial\hat{\sigma}(F)}{\partial F(\mathbf{x}_i)}\Big|_{F(x_i)=\hat{F}_{t-1}(x_i)}\,, \\ &= C_t\psi_0\left(\frac{y_i - \hat{F}_{t-1}(x_i)}{\hat{\sigma}(\hat{F}_{t-1})}\right), \end{aligned}$$

where

$$C_t = \left[\sum_{i\in\mathcal{I}_{\text{train}}} \psi_0\left(\frac{y_i - \hat{F}_{t-1}(x_i)}{\hat{\sigma}(\hat{F}_{t-1})}\right)\left(\frac{y_i - \hat{F}_{t-1}(x_i)}{\hat{\sigma}(\hat{F}_{t-1})}\right)\right]^{-1}.$$

Like `TFBoost`, we fit a functional multi-index tree tp the negative gradients

$$\hat{h}_t(\langle\cdot,\hat{\beta}_{t,1}\rangle,\langle\cdot,\hat{\beta}_{t,P}\rangle) = \underset{h,\beta_1,\ldots,\beta_P}{\operatorname{argmin}} \sum_{i\in\mathcal{I}_{\text{train}}} \left(u_{t,i} - h\left(\langle x_i,\beta_1\rangle,\ldots,\langle x_i,\beta_P\rangle\right)\right)^2,$$

Algorithm 8: RTFBoost (RR)

Input : A data set $(x_i, y_i), i \in \mathcal{I}_{\text{train}}$
Number of iterations of the **S-stage** T_1
Number of iterations of the **M-stage** T_2
Shrinkage parameter $\gamma \in (0, 1)$
Initialization $\hat{F}_0(x)$
Number of indices P

S-stage :

for $t = 1 : T_1$ **do**

$$\hat{\sigma}(\hat{F}_{t-1}) = \left\{ \sigma : \frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i \in \mathcal{I}_{\text{train}}} \rho_0 \left(\frac{y_i - \hat{F}_{t-1}(x_i)}{\sigma} \right) = \kappa \right\}$$

$$C_t = \left[\sum_{i \in \mathcal{I}_{\text{train}}} \psi_0 \left(\frac{y_i - \hat{F}_{t-1}(x_i)}{\hat{\sigma}(\hat{F}_{t-1})} \right) \left(\frac{y_i - \hat{F}_{t-1}(x_i)}{\hat{\sigma}(\hat{F}_{t-1})} \right) \right]^{-1}$$

$$u_{t,i} = C_t \psi_0 \left(\frac{y_i - \hat{F}_{t-1}(\mathbf{x}_i)}{\hat{\sigma}(\hat{F}_{t-1})} \right)$$

$$\hat{h}_t(\langle \cdot, \hat{\beta}_{t,1} \rangle, ..., \langle \cdot, \hat{\beta}_{t,P} \rangle) = \underset{h, \beta_1, ..., \beta_K}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h \left(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle \right) \right)^2$$

$$\hat{\alpha}_t = \underset{\alpha}{\operatorname{argmin}}\, \hat{\sigma} \left(\hat{F}_{t-1} + \alpha \hat{h}_t(\langle \cdot, \hat{\beta}_{t,1} \rangle, ..., \langle \cdot, \hat{\beta}_{t,P} \rangle) \right)$$

$$\hat{F}_t(x) = \hat{F}_{t-1}(x) + \gamma \hat{\alpha}_t \hat{h}_t(\langle x, \hat{\beta}_{t,1} \rangle, ..., \langle x, \hat{\beta}_{t,P} \rangle)$$

end

M-stage :

for $t = 1 : T_2$ **do**

$$u_{t,i} = \frac{1}{\hat{\sigma}} \psi \left(\frac{y_i - \hat{F}_{T_1 + t - 1}(x_i)}{\hat{\sigma}} \right)$$

$$\hat{h}_t(\langle \cdot, \hat{\beta}_{t,1} \rangle, ..., \langle \cdot, \hat{\beta}_{t,P} \rangle) = \underset{h, \beta_1, ..., \beta_K}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \left(u_{t,i} - h \left(\langle x_i, \beta_1 \rangle, ..., \langle x_i, \beta_P \rangle \right) \right)^2$$

$$\hat{\alpha}_t = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in \mathcal{I}_{\text{train}}} \rho \left(\frac{y_i - \hat{F}_{T_1 + t - 1}(x_i) - \alpha \hat{h}_t(\langle x_i, \hat{\beta}_{t,1} \rangle, ..., \langle x_i, \hat{\beta}_{t,P} \rangle))}{\hat{\sigma}} \right)$$

$$\hat{F}_{T_1 + t}(x) = \hat{F}_{T_1 + t - 1}(x) + \gamma \hat{\alpha}_t \hat{h}_t(\langle x, \hat{\beta}_{t,1} \rangle, ..., \langle x, \hat{\beta}_{t,P} \rangle))$$

end

Output : $\hat{F}_{T_1 + T_2}(x)$

where P specifies the number of indices, $\beta_1, ..., \beta_P \in \mathcal{L}$, and $h : \mathbb{R}^P \to \mathbb{R}$ denotes a decision tree. Finally, we search for the optimal step size

$$\hat{\alpha}_t = \underset{\alpha}{\operatorname{argmin}}\, \hat{\sigma}\left(\hat{F}_{t-1} + \alpha \hat{h}_t(\langle \cdot, \hat{\beta}_{t,1}\rangle, \langle \cdot, \hat{\beta}_{t,P}\rangle)\right),$$

and update the function

$$\hat{F}_t(x) = \hat{F}_{t-1}(x) + \gamma \hat{\alpha}_t \hat{h}_t(\langle x, \hat{\beta}_{t,1}\rangle, \langle x, \hat{\beta}_{t,P}\rangle).$$

As before, the shrinkage parameter $\gamma \in (0, 1]$ is included above to avoid overfitting.

To determine T_1, the number of iterations for the S-stage, we suggest an early stopping time defined as

$$T_{1,\text{stop}} = \underset{t \in \{1,\ldots,T_{1,\max}\}}{\operatorname{argmin}} \hat{\sigma}_{\text{val}}(\hat{F}_t),$$

where the M-scale evaluated on the validation set $\mathcal{I}_{\text{val}}$ satisfies

$$\frac{1}{|\mathcal{I}_{\text{val}}|} \sum_{i \in \mathcal{I}_{\text{val}}} \rho_0\left(\frac{y_i - F(x_i)}{\hat{\sigma}_{\text{val}}(F)}\right) = \kappa\,.$$

The M-stage has been described in Section 4.2.1, where the scale estimator $\hat{\sigma}$ in (4.14) is the residual scale estimator $\hat{\sigma}(\hat{F}_{T_1})$ obtained from the S-stage.

4.3 Simulation studies

In this section, we explore the performance of the proposed `RTFBoost (LAD-M)` and `RTFBoost (RR)` and compare them with available alternatives, including `TFBoost`, selected robust functional regression methods introduced in Section 4.1, and `FPPR` and `FGAM` that performed well in the numerical studies of Chapter 3.

4.3.1 Set up

We considered a model similar to the one defined in Section 3.4.1. The data pairs $(x_i, y_i), i = 1, ..., N$ each consisting of a predictor $x_i \in L^2(\mathcal{I})$ and a scalar response y_i were generated to follow the model:

$$y_i = r(x_i) + C\epsilon_i, \tag{4.15}$$

where ϵ_i is the error term, r is the regression function, and $C > 0$ is a constant that controls the signal-to-noise ratio (SNR), which we defined later in (4.17).

Let (x_i, y_i), $i = 1, ..., N$ be i.i.d. realizations of (X, Y). To represent inhomogeneous data, we adopted the gross-error model where we observed "good" points $(x_i, y_i) \sim D_0$ most of the time, and contaminated points $(x_i, y_i) \sim \tilde{D}$ less often. We use α to denote the proportion of contamination and represent the joint distribution of (X, Y):

$$D = (1 - \alpha)D_0 + \alpha\tilde{D}. \tag{4.16}$$

To assess the noise level for the "good" points, we define the signal-to-noise ratio

$$\text{SNR} = \frac{\text{Var}(r(X))}{\text{Var}(C\epsilon)}, \tag{4.17}$$

where X follows the marginal distribution of D_0. For a given level of SNR, the value of C in (4.15) can be computed through (4.17).

For D_0, we adopted the same models (M_1 and M_2) to generate functional predictors as the ones considered in Section 3.4.1. For the i-th individual, we evaluated x_i on a dense grid $t_1, ..., t_{100}$ evenly spaced in $\mathcal{I} = [-1, 1]$ for M_1 and $\mathcal{I} = [0, 1]$ for M_2. The error term ϵ_i are i.i.d from $N(0, 1)$.

We considered the regression functions r_1, ..., r_4 used in Section 3.4.1, and added a linear function

$$r_5(X) = \int_{\mathcal{I}} (\sin\left(\frac{3\pi t}{2}\right) + \sin\left(\frac{\pi t}{2}\right))X(t)dt.$$

For contaminated settings, we considered six options with outliers defined as follows

- D_1: shape outliers with asymmetric contamination
  - For M_1, let $X(t) = at + b$, where $a \sim \mathcal{U}(3, 4)$ and $b \sim \mathcal{U}(0, 2)$;
  - For M_2, let $\xi_{j,2} \sim N(2, 0.25)$, $\mu(t) = 1 + 2\cos(2t\pi)$ with the other parameters staying the same (see the definition of these parameters in Section 3.4.1);
  - $\epsilon \sim N(-30, 0.25)$;
- D_2: shape outliers with asymmetric contamination
  - X follows the same distribution as in the D_1 setting;
  - $\epsilon \sim 0.5N(-30, 0.25) + 0.5N(30, 0.25)$;

- D_3: curve-type magnitude outliers with symmetric contamination
  - $X = 2\tilde{X} + 5$, where $\tilde{X}$ follows the same distribution as in the D_0 setting;
  - $\epsilon \sim 0.5N(-30, 0.25) + 0.5N(30, 0.25)$;
- D_4: interval-type magnitude outliers with symmetric contamination
  - For each object i, we randomly sampled one interval from $[t_1, ..., t_{10}]$, ..., $[t_{91}, ..., t_{100}]$ and denoted it as $\tilde{t}$. For each point $s \in \tilde{t}$, we let

    $$x_j(s) = \tilde{x}_j(s) + \eta_s,$$

    where $\eta_s \sim N(10, 0.25)$ and $\tilde{x}_j$ follow the distribution of X in the D_0 setting;
  - $\epsilon \sim 0.5N(-30, 0.25) + 0.5N(30, 0.25)$;
- D_5: vertical symmetric contamination
  - X follows the same distribution as in the D_0 setting;
  - $\epsilon \sim 0.5N(-30, 0.25) + 0.5N(30, 0.25)$;
- D_6: vertical asymmetric contamination
  - X follows the same distribution as in the D_0 setting;
  - $\epsilon \sim N(-30, 0.25)$.

To better understand these contaminated settings, we took a random sample of size 50 and plotted the predictors generated from D_0 to D_6 in Figure 4.1 for M_1 and Figure 4.2 for M_2. In each figure, there are four panels: the top left one for D_0, D_5 and D_6, the predictors of which are uncontaminated, the top right one for D_1 and D_2 that contain shape outliers, and the bottom two for D_3 and D_4 respectively, which contain magnitude outliers on the entire curve or selected intervals.

In these contaminated settings, D_1 to D_4 correspond to data with high-leverage outliers with D_1 concerning asymmetric contamination and the rest symmetric contamination. D_5 and D_6 include vertical outliers with symmetric and asymmetric contamination respectively. We choose not to include asymmetric contaminated settings for magnitude outliers since `TFBoost` with squared loss worked sufficiently well in those settings in our preliminary experiments. This is not surprising since these magnitude outliers

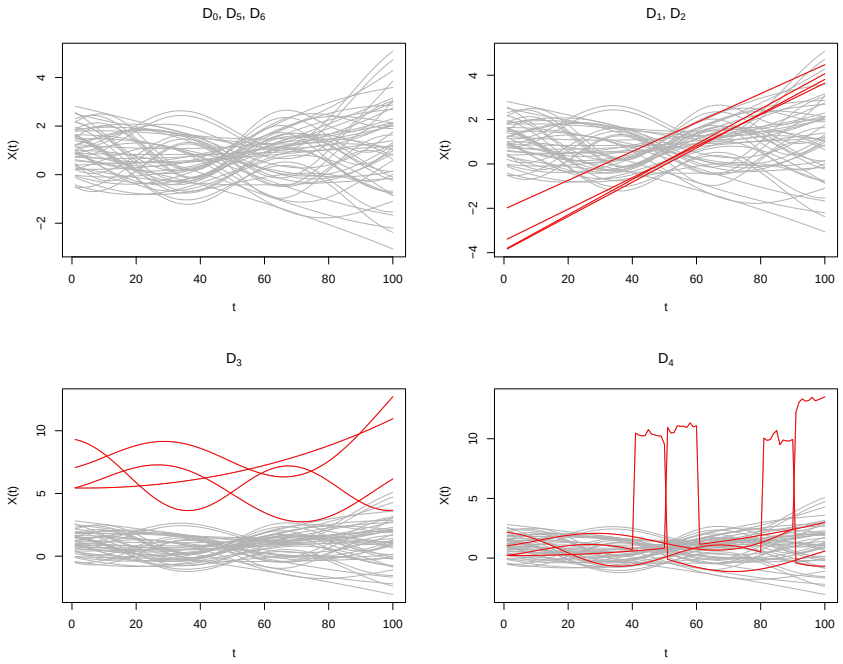


Figure 4.1: 50 random samples generated from M_1 settings with the red curves representing outliers in the predictors. The top left panel shows data generated from D_0, D_1, and D_2; top right panel D_1 and D_2; bottom left panel D_3; and bottom right panel D_4.

tend to produce index values far from the others. When they are contaminated in the same direction, it encourage the trees to fit these outliers locally without affecting the "good" points.

For D_1 to D_6, we considered rates of contamination equal to 10% and 30% (which corresponds to $\alpha = 10\%$ and 30% in (4.16) respectively). To study cases with low and medium noise levels, we considered SNR = 5 and 20 (denoted as S_1 and S_2 respectively). For each combination of the SNR (S_1 or S_2), predictor model (M_1 or M_2), regression function (r_1 to r_5), and contamination (D_0 to D_6), we generated data that contains a training set of size 400, validation set of size 200, and test set of size 1000. The settings of D_0 regard to data without outliers in the training, validation, and test sets. For D_1 to D_6 settings, the training and validation sets were generated from the gross-error model (4.16), and the test set contains only "good" points sampled from D_0.

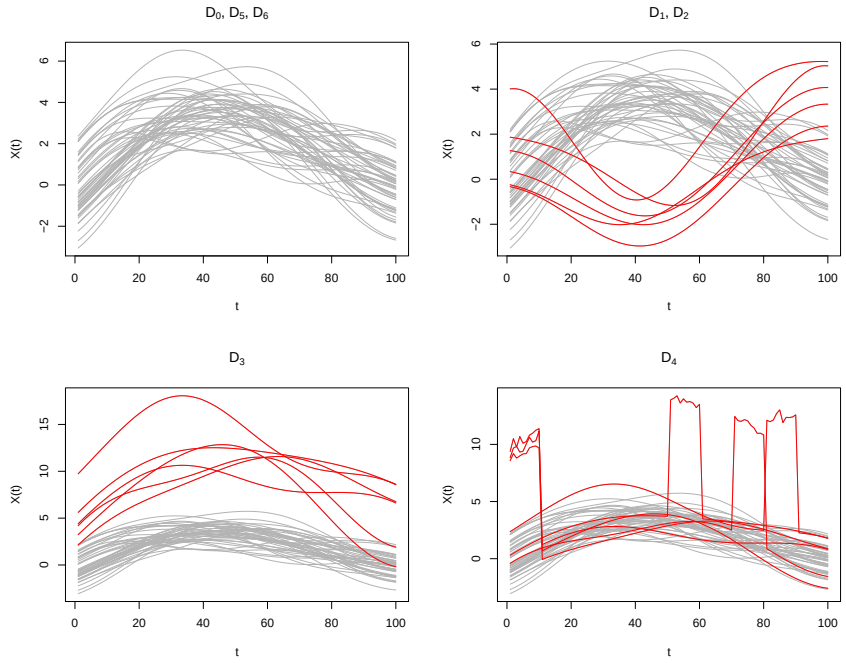


Figure 4.2: 50 random samples generated from M_2 settings with the red curves representing outliers in the predictors. The top left panel shows data generated from D_0, D_1, and D_2; top right panel D_1 and D_2; bottom left panel D_3; and bottom right panel D_4.

4.3.2 Implementation details

For each possible combination of (M, r, S, D) from the set $\{M_1, M_2\} \times \{r_1, ..., r_5\} \times \{D_0, ..., D_6\} \times \{S_1, S_2\}$, we conducted 100 independent experiments and compared the following estimators:

- `FPPR`: functional projection pursuit regression (Ferraty et al., 2013);
- `FGAM`: functional generalized additive models (McLean et al., 2014),
- `MFLM`: M-estimator for partial-functional linear regression models (Huang et al., 2015);
- `RFPLM`: MM-estimator for partial-functional linear regression models (Boente et al., 2020);
- `RFSIR`: robust functional sliced inverse regression (Wang et al., 2017);
- `FSIQ`: functional single-index quantile regression (Sang and Cao, 2020);
- `TFBoost(L2)`: `TFBoost` with squared error loss (see Section 3.2);
- `TFBoost(LAD)`: `TFBoost` with absolute error loss (see Section 3.2);
- `RTFBoost(LAD-M)`: robust extension of the `TFBoost` algorithm, involving the LAD-stage and the M-stage (see Section 4.2.1);
- `RTFBoost(RR)`: robust extension of the `TFBoost` algorithm, involving the S-stage and the M-stage (see Section 4.2.2).

The `FPPR` and `FGAM` estimators were computed in the same way as we did in Section 3.4.2. For `FPPR`, we used cubic B-spline with 7 functions (3 evenly spaced interior knots) as the basis and selected the number of additive components between 1 to 15 based on the prediction error on the validation set. Since the validation set also contains outliers (except for D_0 settings), we used a robust MSPE metric to measure the prediction error, which we call RMSPE and define as follows:

$$\text{RMSPE} = \mu_M^2(\{y_i - \hat{F}(x_i)\}|i \in \mathcal{I}_{\text{val}}) + \hat{\sigma}_M^2(\{y_i - \hat{F}(x_i)\}|i \in \mathcal{I}_{\text{val}}), \quad (4.18)$$

where $\hat{\mu}_M$ is a M-location estimator that achieves 95% asymptotic efficiency at the Gaussian model, and $\hat{\sigma}_M$ an M-scale estimator with 50% breakdown point. Both $\hat{\mu}_M$ and $\hat{\sigma}_M$ were were computed using Tukey's bisquare loss.

For `FGAM`, we adopted the implementation in the `refund` package (Goldsmith et al., 2020). Same as in Section 3.4.1, we used tensor-type bivariate cubic B-splines basis of dimension 15 by 15.

`MFLM` and `RFPLM` were available from the package `RobustFPLM`. Both methods were originally proposed for partial-functional regression, but their implementation can be easily adapted to fit regression with functional predictors only. We chose them as representatives of robust functional linear estimators, one derived from M-estimation and the other one from MM-estimation. As in Huang et al. (2015), `MFLM` was computed using Huber's loss, the tuning constant of which was selected to achieve 95% asymptotic efficiency for a location model with Gaussian errors. Following Boente and Salibian-Barrera (2021), `RFPLM` used Tukey's bisqaure loss function with the tuning constant selected for a location MM-estimator to achieve 50% breakdown point and 95% asymptotic efficiency with Gaussian errors.

`RFSIR` and `FSIQ` introduced in Section 4.1.3 were included as competing index-based estimators. For `RFSIR` we adopted the code shared by the authors of Wang et al. (2017) and selected the number of indices from 1 to 10 to minimize BIC. To compute `FSIQ`, we used the code available at https://github.com/caojiguo/FunSIQ from the authors of Sang and Cao (2020). The code used cubic splines as a special case of RKHS, which was assumed to be the space of the estimator in Sang and Cao (2020). We fitted `FSIQ` to estimate the conditional median of the response, where the dimension of the cubic splines was selected from 4 to 7 to minimize RMSPE defined in (4.18).

In our R package available at https://github.com/xmengju/RTFBoost, we implemented `RTFBoost` and `TFBoost`. The package allows using Type A or Type B trees as the base learners for `RTFBoost(LAD-M)`, `RTFBoost(RR)`, and `TFBoost` with user-defined loss functions. Considering the lower computation cost of fitting Type B trees, we compared `TFBoost` and `RTFBoost` computed with Type B tree base learners. The maximum depth of these trees were selected from 1 to 4 to achieve the lowest RMSPE (see (4.18)) on the validation set. We set the shrinkage parameter $\gamma = 0.05$ and the maximum number of iterations $T_{\texttt{max}} = 1000$ for `TFBoost(LS)` and `TFBoost(LAD)`, and $T_{1,\texttt{max}} = T_{2,\texttt{max}} = 1000$ for `RTFBoost(LAD-M)` and `RTFBoost(RR)`. With these values, in most cases the validation loss ceased to improve before reaching the maximum number of iterations.

For `RTFBoost(RR)` we controlled the level of robustness of the S-stage by considering $\kappa = 0.2$ and 0.5. To differentiate these two options, we denoted the resulting estimators as `RTFBoost(RR,0.2)` and `RTFBoost(RR,0.5)` respectively. We initialized `TFBoost(LS)`, `TFBoost(LAD)`, `RTFBoost(LAD-M)`, and `RTFBoost(RR)` at the median of all training responses. Like `RRBoost`, the loss function of `RTFBoost(RR)` is non-convex, and thus a good initialization point can avoid it reaching a local optimum with a large objective

value. We found that median initialization works well for `RTFBoost(RR)` in our numerical experiments and thus adopted it for its simplicity, even though in principle, an initial fit that accounts for the predictors may be beneficial in particular cases, e.g. when the response takes values within the wide range so that "good observations" may have large residuals in early iterations. Possible alternatives to initialize `RTFBoost(RR)` are addressed in Section 4.5 for future work.

4.3.3 Results

In Figures 4.3 to 4.12, the results are plotted for the mean-squared-prediction-error (MSPE) computed on the test sets based on 100 simulation runs. Each figure corresponds to one out of the 10 possible combinations of the predictor model (M_1 or M_2) and regression function (r_1 to r_5). For each combination, let α_1 and α_2 denote cases where the contaminated distributions (D_1 to D_6) have 10% or 30% outliers respectively. Signal to noise ratios of 5 and 20 are represented as S_1 and S_2.

In each figure, we considered four cases: (α_1, S_1), (α_1, S_2), (α_2, S_1), and (α_2, S_2). For each case, we often observed that some methods produced much higher MSPE compared to the others. To be able to see the differences amongst the other methods, we truncated the y-axis to exclude the estimators the preformed substantially worse than the rest. The reader can refer to the summary statistics of all methods that we considered in Section 4.3.2 from the Appendix.

In each panel of Figures 4.3 to 4.12, we connected the points for the methods that consistently produce moderate or low values of test MSPEs for D_0 to D_6, namely the ones for which no points was truncated in these cases. This allows us to see clearly the methods that produced stable results regardless of the distribution of the outliers.

Figures 4.3 to 4.12 show that `RTFBoost(RR,0.5)` performs well in D_0 to D_6 for all settings. In almost all cases, it achieves the lowest or second lowest average test MSPE. The performance of other methods varies for different proportion of outliers, data distributions, and regression functions. We summarize more detailed findings in the following aspects:

- Proportion of outliers (α_1 and α_2):

  Comparing the left panels (α_1 settings) with the right panels (α_2 settings) in each figure, we see that the performance of `RTFboost(RR,0.5)` remains stable. In α_1 settings, `RTFboost(RR,0.2)` is usually the best performing method, with `RTFboost(RR,0.5)` being the close second.

We think that in α_1 settings where only 10% outliers are present, the 50% high breakdown point used by `RTFboost(RR,0.5)` may mistakenly treat some "good points" as outliers, and thus negatively affected the fit. In α_2 settings, `RTFboost(RR,0.2)` can fail in various contaminated settings. This is due to its breakdown point only being 20%, and thus not as robust as `RTFBoost(RR,0.5)` in the S-stage. For the other competitors, increasing the proportion of outliers from 10% to 30% typically increases the test MSPE for all types of contamination. Exceptions are `RTFBoost(LAD-M)` and `RFPLM` which showed little change in a few settings.

- Data distributions (D_0 to D_6):

  We found that compared to D_0, the contaminated settings D_1 to D_6 tend to affect the performance of estimators differently. From the figures, we see that `RTFboost(LAD-M)` is generally more affected by D_1, and D_7 with 30% outliers, and in a few cases also by D_2. We know that `RTFboost(LAD-M)` used the L_1 in the first stage, which can be especially sensitive to asymmetric outliers. D_1 and D_7 with 30% outliers corresponds to this case. In these cases, M-stage failed to improve the performance of a poorly fitted LAD-stage.

  Note that `RTFBoost(LAD-M)` is less affected by D_3 and D_4 compared to D_1 and D_2. We think this is due to D_3 and D_4 defined as magnitude outliers, such that for most index coefficient functions, the index values of the outliers are likely to be extreme. Since our methods use tree learners, these trees can provide local fit to these outliers by isolating them into separate nodes, which do not affect the prediction error on clean data. Comparatively, D_1 and D_2 define curve outliers, of which the projections on the basis may not be extreme for some index coefficients, and thus neither do their corresponding index values. As a result, the LAD-stage of `RTFBoost(LAD-M)` may try to overfit these outliers rather than isolating them.

  The other competitors are usually more affected by high-leverage and asymmetric outliers, namely D_1, D_2, D_3, D_4, and D_6 settings.

- Regression functions (r_1 to r_5):

  For nonlinear regression functions r_1 to r_4, as expected, the linear M-estimation method `MFLM` perform poorly, for clean setting D_0 and contaminated settings D_1 to D_6. In some cases (see e.g. Figure 4.3, Figure 4.8, and Figure 4.9), the linear MM-estimation method `RFPLM`

appears to be insensitive to outliers across D_1 to D_6, however, its linear assumption limits the flexibility of the estimator and thus yields stable but high MSPEs. For linear regression function r_5, as expected, `RFPLM` outperforms the rest. `RTFBoost(RR,0.2)` or `RTFBoost(RR,0.5)` is the close second best for α_1 settings, and `RTFBoost(RR,0.5)` for α_2 settings. This implies that though `RTFBoost(RR)` was not proposed as a linear estimator, it works reasonably well to fit linear models.

- SNR (S_1 and S_2) and predictor model (M_1 and M_2):

  When the other parameter for data generation are the same, the conclusions draw from S_1 or S_2, and M_1 or M_2 settings are very similar. With a higher SNR in S_2 settings, the performance of all methods improved, especially the ones that already worked well in S_1 settings. The performance of `RTFBoost(RR,0.5)` appears to me more stable across D_1 to D_6 in S_2 compared to S_1 settings (see e.g. Figure 4.5 and Figure 4.7). In M_2, increasing the proportion of outliers seems to do more damage to methods other than `RTFBoost(RR, 0.5)` compared to in M_1 settings.

Overall for `RTFBoost`, we observed that `RTFBoost(RR,0.5)` performed consistently well in all settings, and `RTFBoost(RR,0.2)` in α_1 settings. We found that the performance of `RTFBoost(LAD-M)` is affected by data distribution. It generally works well in D_0, D_3, D_4, and D_5, but can be affected by outliers in D_1, D_2, and D_6, which are the settings with shape outliers or asymmetric outliers.

In most cases, `RTFBoost(LAD-M)` outperforms `RTFBoost(LAD)`. One may noticed that in α_1 settings with D_2 distributed data in Figure 4.5 and Figure 4.6, the contrary is observed: `RTFBoost(LAD)` performs better than `RTFBoost(LAD-M)`. This is due to the way that we selected the optimal depth of the trees using RMSPE, so that in each trial the LAD-stage of `RTFBoost(LAD-M)` and `RTFBoost(LAD)` may be computed using trees of different depths, and thus are not directly comparable. In addition, in a few trials, the points ignored by RMSPE very are "good" points that are poorly fitted, resulting in `RTFBoost(LAD-M)` selecting the non-optimal tree depth. Despite of this, the difference between these two methods is not significant because of the large standard decision of their test MSPEs (see Appendix).

Even though the performance of `RTFBoost(LAD-M)` varies for data distributions, considering its lower computing cost compared to `RTFBoost(RR)`, in practical situations, it is still good to have it as an alternative. Particularly for magnitude outlier or vertical symmetric outliers, `RTFBoost(LAD-M)`

usually performs second to `RTFBoost(RR)`.

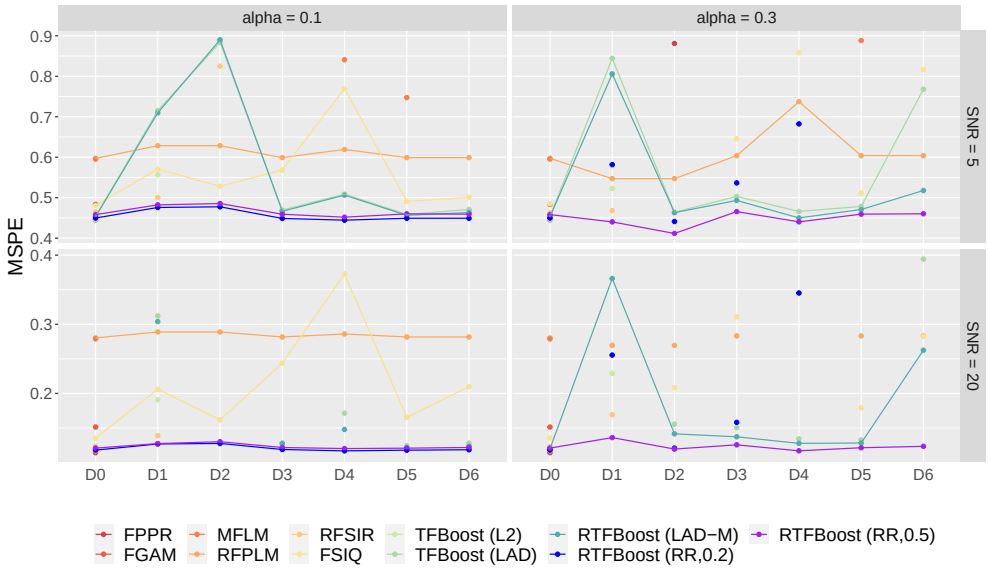


Figure 4.3: Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_1; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

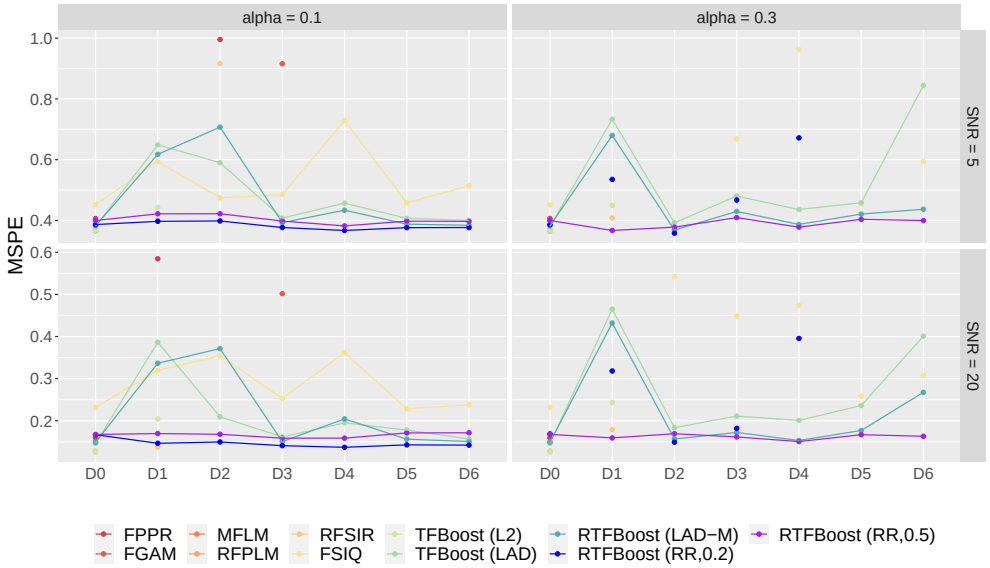


Figure 4.4: Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_2; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

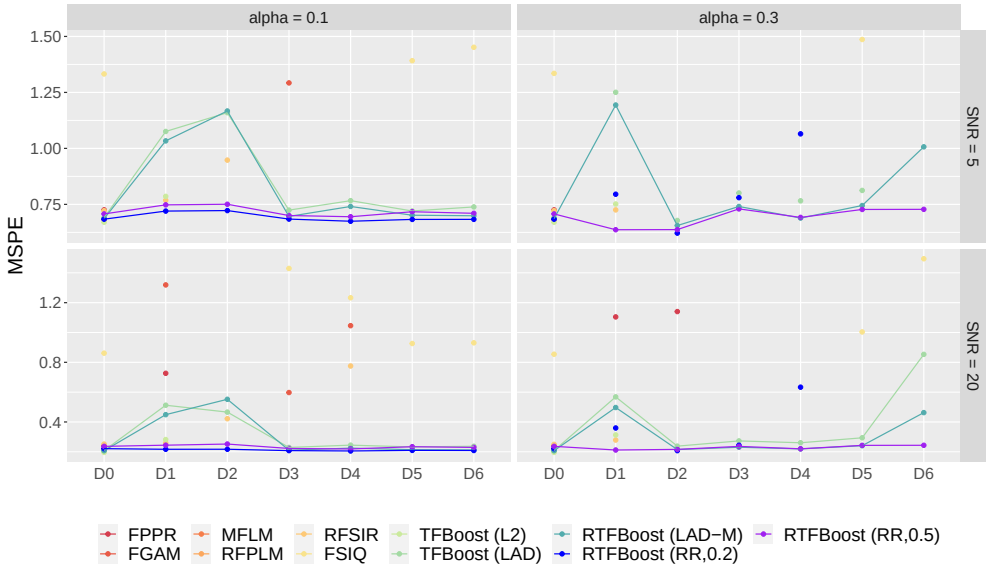


Figure 4.5: Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_3; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

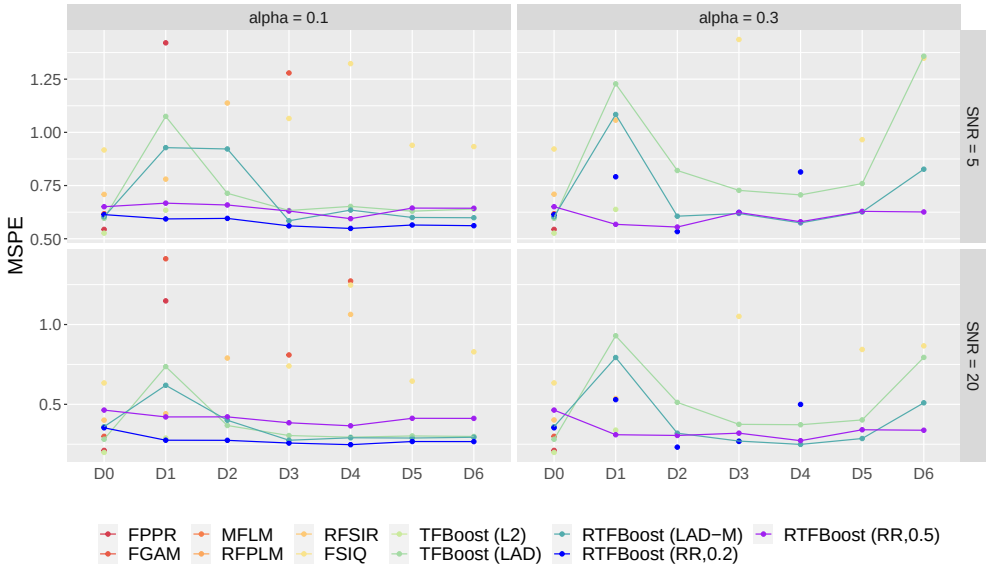


Figure 4.6: Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_4; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

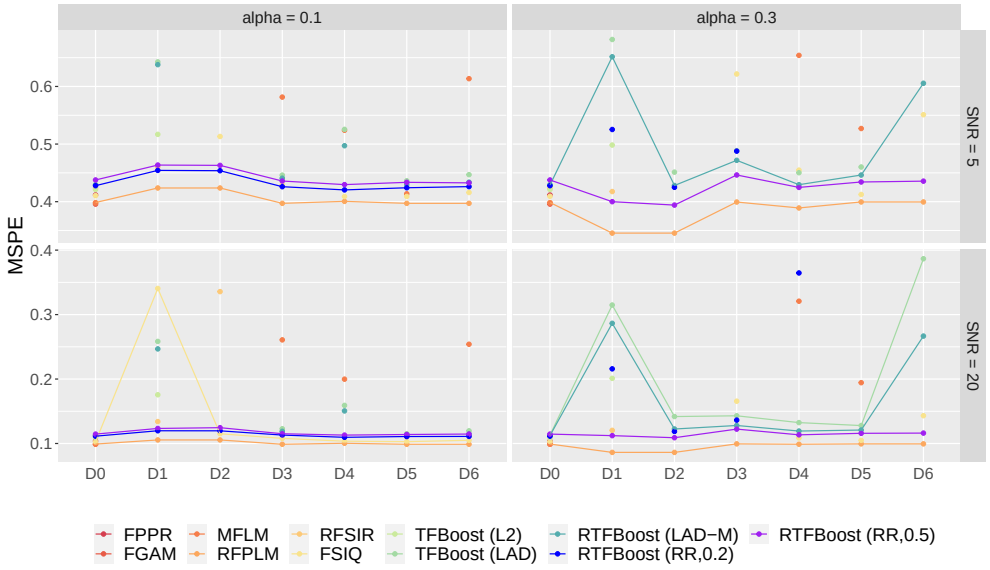


Figure 4.7: Average test MSPEs based on 100 simulation runs for data generated from M_1 and r_5; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

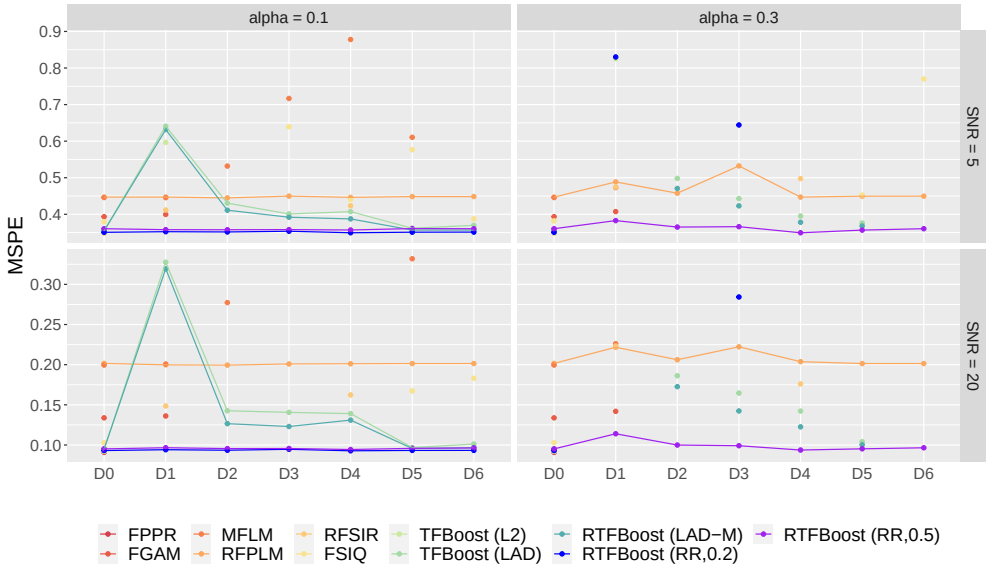


Figure 4.8: Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_1; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

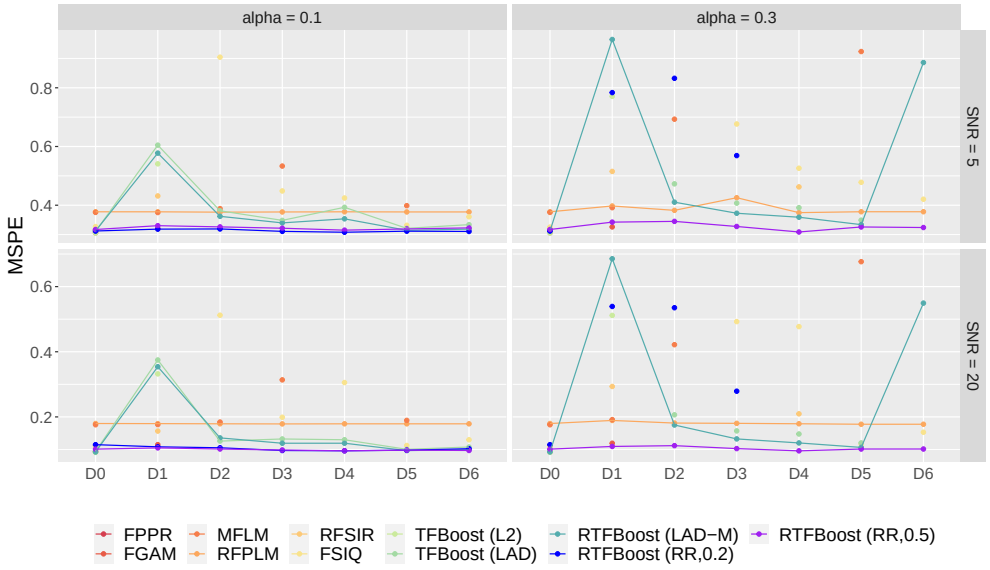


Figure 4.9: Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_2; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

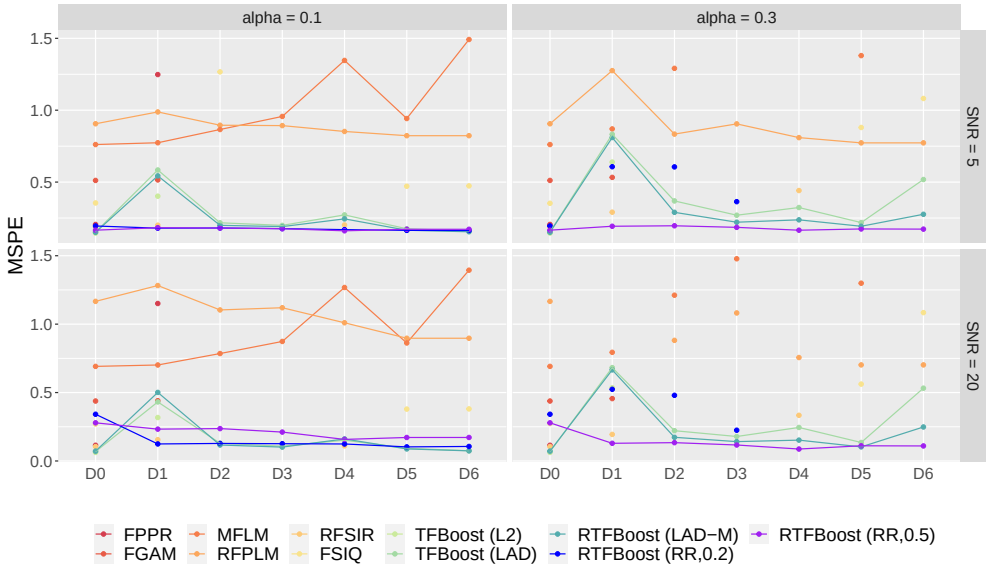


Figure 4.10: Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_3; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

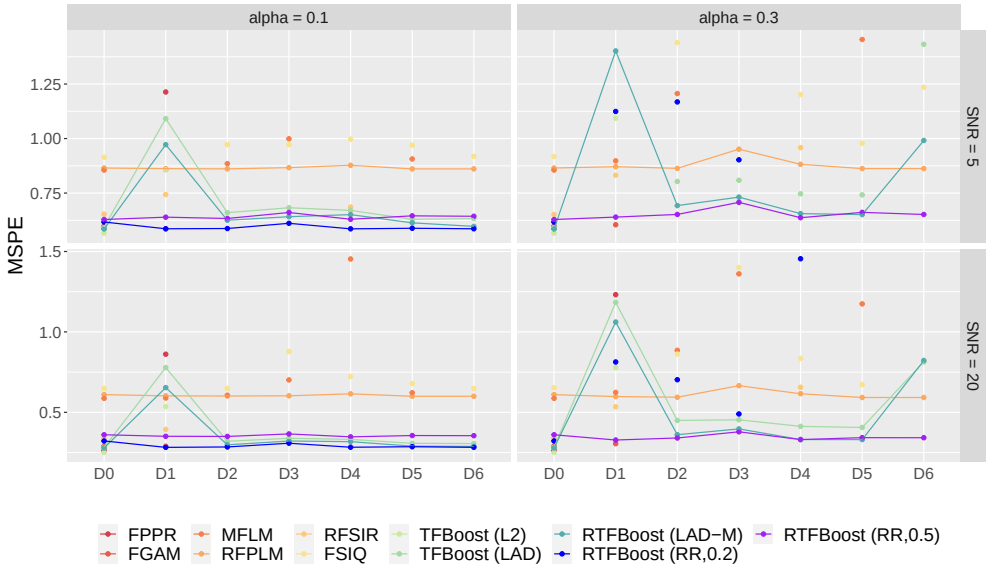


Figure 4.11: Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_4; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

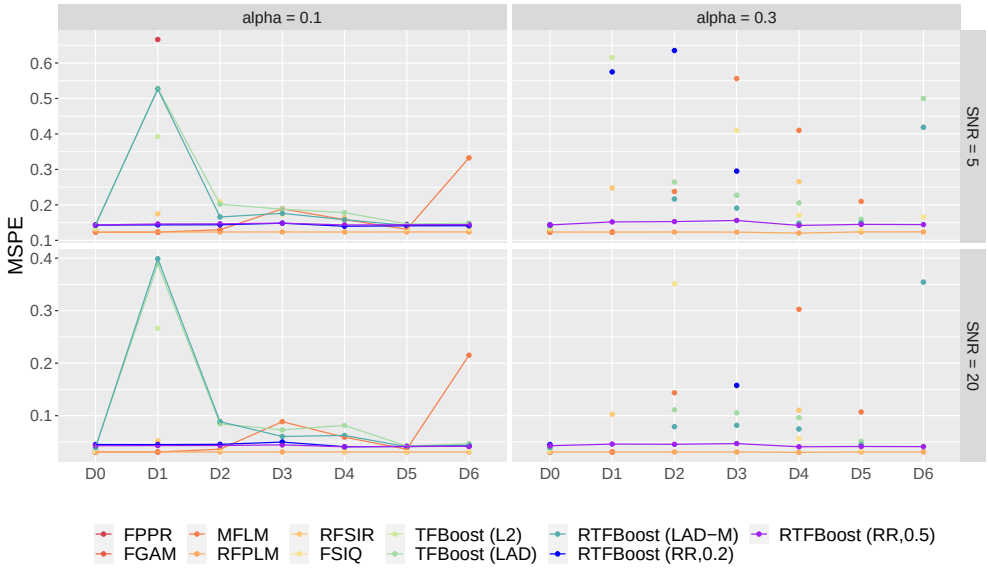


Figure 4.12: Average test MSPEs based on 100 simulation runs for data generated from M_2 and r_5; the panels represent combinations of $\{\alpha_1, \alpha_2\} \times \{S_1, S_2\}$; in each panel, results for D_0 to D_6 are plotted.

4.4 Fruit fly data

We applied our proposed methods to the fruit fly data, which is available at https://anson.ucdavis.edu/ mueller/data/data.html. The data set consists of number of eggs laid daily for each of 1000 medflies until the time of death. Details of the data collecting procedure can be found in Carey et al. (1998). We used a subset of the data including flies that lived at least 30 days to predict the number of living days for each fruit fly, and ended up with data from 667 flies. The number of eggs laid in the first 30 days is treated as the functional predictor (x_i) with values evaluated at each day. The life time is a numerical value treated as the response (y_i).

We randomly portioned the data into 60% training set ($\mathcal{I}_{\text{train}}$), 20% validation set ($\mathcal{I}_{\text{val}}$), and 20% test set ($\mathcal{I}_{\text{test}}$), and considered the same methods in Section 4.3.2. The description and set-ups of all methods follows the details in Section 4.3.2. Since the test set may contain outliers, average absolute values (AAD) of the residuals on the test set are used for comparison, which is defined as

$$AAD = \sum_{i \in \mathcal{I}_{\text{test}}} \left| \hat{F}(x_i) - y_i \right|$$

for any estimator $\hat{F}$.

In addition to using the original data, we considered two contaminated settings with 20% vertical asymmetric outliers, which appeared to be a very damaging case for many methods under consideration. The outliers are defined as

$$\tilde{y}_i = y_i + \eta_i,$$

where

$$\eta_i \sim 0.5N(-30, 2) + 0.5N(30, 2).$$

To avoid randomness introduced by a single partition, we repeated the experiment using 100 random partitions. The AADs The summary statistics of test robust MSPEs are shown in Figures 4.13 and 4.14. The y-axis of these figures are truncated to eliminate showing extreme errors. As a result, the linear estimators `MFLM` and `RFPLM` are not shown in these figures. Summary statistics of test AADs for all methods are included in the Appendix.

In Figure 4.13, we found that with the original data, `TFBoost` and `RTFBoost` methods work better than their competitors. We think this is because of the flexibility of our multi-index estimators to fix complex regression functions. Compared with `TFBoost`, the robust methods perform better, which likely indicate the existence of outliers in this data set.

For data with added asymmetric outliers, in Figure 4.14 we can see that `FPPR`, `FGAM`, `RFSIR`, `FSIQ` `TFBoost(L2)` are greatly affected. The methods of `TFBoost(LAD)` and `RTFBoost(RR,0.5)` achieved the best performance. `RTFBoost(LAD-M)` performed slightly worse than `TFBoost(LAD)`, likely due to it selected a different tree depth in different runs.

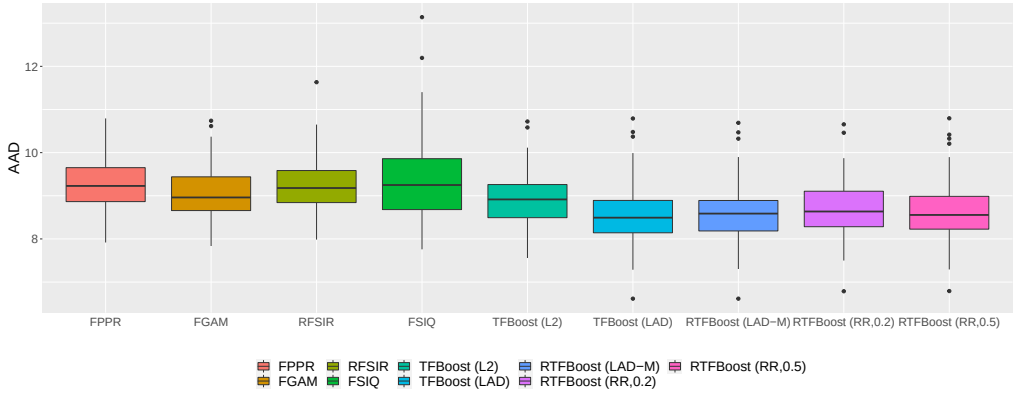


Figure 4.13: Test AADs based on 100 random partitions for the fruit fly data.

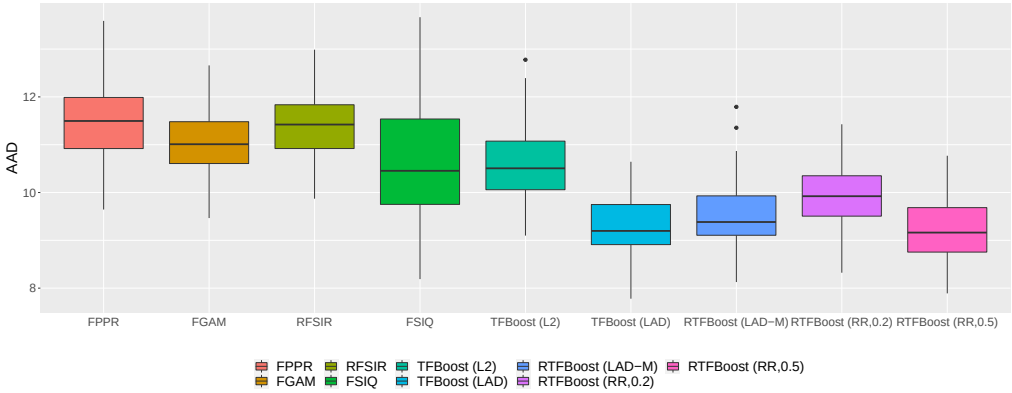


Figure 4.14: Test AADs based on 100 random partitions for the fruit fly data with vertical asymmetric outliers.

4.5 Summary

In this chapter, we explored two ways to robustify `TFBoost`, which include `RTFBoost(LAD-M)` derived from M-estimation, and `RTFBoost(RR)` derived from MM-estimation. In the numerical studies, we compared the performance of our method in various contaminated scenarios, including high-leverage outliers and vertical outliers. In the literature, most robust functional regression methods were proposed for linear models, and we found that those methods do not work well when fitting nonlinear functions. It is observed that `RTFBoost(RR,0.5)` produced stable and often the lowest errors in all settings under consideration, which we recommend in practice, especially when the proportion and distribution of the outliers are unknown. `RTFBoost(LAD-M)` performs reasonably well for data with magnitude outliers and vertical symmetric outliers. Even though it can be sensitive to some types of outliers, it has the advantage of requiring a lower computation cost. `RTFBoost(LAD-M)` can still be useful if one has prior knowledge of the type of outliers existing in the data.

In our numerical experiments, to initialize `RTFBoost(RR)`, we used the median of training responses. In some particular cases when the response range through a wide set of values, it can be helpful to consider a robust initialization that also considers the functional predictor. Like `RRBoost`, we consider using multi-index LAD trees that minimizes the L_1 loss. Though we found the median initialization worked well in our experiments, we have included this option of LAD Tree initialization in our R package hosted at

https://github.com/xmengju/RTFBoost. We found that the performance `RTFBoost(RR)` can be very sensitive to how the initial tree is specified, e.g. the number of indices, the maximum tree depth, the minimum number of observations per node. With the validation set being contaminated, it can be difficult to find a criteria for selecting these hyper-parameters. Proper ways to select these hyper-parameters needs to be further investigated in the future.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we address the problem of

5.2 Future work

Because of this, we elaborate on the directions of future work ..

Bibliography

Adler, R. J., Taylor, J. E., et al. (2007). *Random fields and geometry*, volume 80. Springer.

Ait-Saïdi, A., Ferraty, F., Kassa, R., and Vieu, P. (2008). Cross-validated estimations in the single-functional index model. *Statistics*, 42(6):475–494.

Al-Awadhi, F. A., Kaid, Z., Laksaci, A., Ouassou, I., and Rachdi, M. (2019). Functional data analysis: local linear estimation of the l1-conditional quantiles. *Statistical Methods & Applications*, 28(2):217–240.

Amato, U., Antoniadis, A., and De Feis, I. (2006). Dimension reduction in functional regression with applications. *Computational Statistics & Data Analysis*, 50(9):2422–2446.

Amiri, A., Crambes, C., and Thiam, B. (2014). Recursive estimation of nonparametric regression with functional covariate. *Computational Statistics & Data Analysis*, 69:154–172.

Avery, M., Wu, Y., Helen Zhang, H., and Zhang, J. (2014). Rkhs-based functional nonparametric regression for sparse and irregular longitudinal data. *Canadian Journal of Statistics*, 42(2):204–216.

Bates, D., Mächler, M., Bolker, B., and Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48.

Belarbi, F., Chemikh, S., and Laksaci, A. (2018). Local linear estimate of the nonparametric robust regression in functional data. *Statistics & Probability Letters*, 134:128–133.

Bellman, R. (1961). *Adaptive Control Processes*. Princeton University Press.

Blumenson, L. (1960). A derivation of n-dimensional spherical coordinates. *The American Mathematical Monthly*, 67(1):63–66.

Boente, G. and Salibian-Barrera, M. (2021). Robust functional principal components for sparse longitudinal data. *METRON*, pages 1–30.

Boente, G., Salibian-Barrera, M., and Vena, P. (2020). Robust estimation for semi-functional linear regression models. *Computational Statistics & Data Analysis*, 152:107041.

Breiman, L. (1984). *Classification and Regression Rrees*. Routledge.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Brockhaus, S., Scheipl, F., Hothorn, T., and Greven, S. (2015). The functional linear array model. *Statistical Modelling*, 15(3):279–300.

Brooks, T. F., Pope, D. S., and Marcolini, M. A. (1989). Airfoil self-noise and prediction. *NASA Reference Publication-1218, document id: 9890016302*.

Bühlmann, P. and Yu, B. (2003). Boosting with the L2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339.

Burba, F., Ferraty, F., and Vieu, P. (2009). k-nearest neighbour method in functional nonparametric regression. *Journal of Nonparametric Statistics*, 21(4):453–469.

Cardot, H., Crambes, C., and Sarda, P. (2005). Quantile regression when the covariates are functions. *Nonparametric Statistics*, 17(7):841–856.

Cardot, H., Ferraty, F., and Sarda, P. (1999). Functional linear model. *Statistics & Probability Letters*, 45(1):11–22.

Cardot, H., Ferraty, F., and Sarda, P. (2003). Spline estimators for the functional linear model. *Statistica Sinica*, pages 571–591.

Cardot, H. and Sarda, P. (2005). Estimation in generalized linear models for functional data via penalized likelihood. *Journal of Multivariate Analysis*, 92(1):24–41.

Carey, J. R., Liedo, P., Müller, H.-G., Wang, J.-L., and Chiou, J.-M. (1998). Relationship of age patterns of fecundity to mortality, longevity, and lifetime reproduction in a large cohort of mediterranean fruit fly females. *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences*, 53(4):B245–B251.

Carroll, C., Gajardo, A., Chen, Y., Dai, X., Fan, J., Hadjipantelis, P. Z., Han, K., Ji, H., Mueller, H.-G., and Wang, J.-L. (2021). *fdapace: Functional Data Analysis and Empirical Dynamics*. R package version 0.5.6.

Chen, D., Hall, P., Müller, H.-G., et al. (2011). Single and multiple index functional regression models with nonparametric link. *The Annals of Statistics*, 39(3):1720–1747.

Chen, J. and Zhang, L. (2009). Asymptotic properties of nonparametric m-estimation for mixing functional data. *Journal of Statistical Planning and Inference*, 139(2):533–546.

Chen, K. and Müller, H.-G. (2012). Conditional quantile analysis when covariates are functions, with application to growth data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(1):67–89.

Cover, T. (1968). Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(1):50–55.

Crambes, C., Delsol, L., and Laksaci, A. (2008). Robust nonparametric estimation for functional data. *Journal of Nonparametric Statistics*, 20(7):573–598.

Crambes, C., Gannoun, A., and Henchiri, Y. (2013). Support vector machine quantile regression approach for functional data: Simulation and application studies. *Journal of Multivariate Analysis*, 121:50–68.

Dielman, T. E. (2005). Least absolute value regression: recent contributions. *Journal of Statistical Computation and Simulation*, 75(4):263–286.

Dou, W. W., Pollard, D., Zhou, H. H., et al. (2012). Estimation in functional regression for general exponential families. *The Annals of Statistics*, 40(5):2421–2451.

Engle, R. F., Granger, C. W., Rice, J., and Weiss, A. (1986). Semiparametric estimates of the relation between weather and electricity sales. *Journal of the American statistical Association*, 81(394):310–320.

Eubank, R. L. (1999). *Nonparametric regression and spline smoothing*. CRC press.

Fan, Y., James, G. M., Radchenko, P., et al. (2015). Functional additive regression. *Annals of Statistics*, 43(5):2296–2325.

Febrero-Bande, M. and González-Manteiga, W. (2013). Generalized additive models for functional data. *Test*, 22(2):278–292.

Ferraty, F., Goia, A., Salinelli, E., and Vieu, P. (2013). Functional projection pursuit regression. *Test*, 22(2):293–320.

Ferraty, F., Hall, P., and Vieu, P. (2010). Most-predictive design points for functional data predictors. *Biometrika*, 97(4):807–824.

Ferraty, F., Park, J., and Vieu, P. (2011). Estimation of a functional single index model. In *Recent advances in functional data analysis and related topics*, pages 111–116. Springer.

Ferraty, F., Peuch, A., and Vieu, P. (2003). Modèle à indice fonctionnel simple. *Comptes Rendus Mathematique*, 336(12):1025–1028.

Ferraty, F., Rabhi, A., and Vieu, P. (2005). Conditional quantiles for dependent functional data with application to the climatic" el niño" phenomenon. *Sankhyā: The Indian Journal of Statistics*, pages 378–398.

Ferraty, F. and Vieu, P. (2002). The functional nonparametric model and application to spectrometric data. *Computational Statistics*, 17(4):545–564.

Ferraty, F. and Vieu, P. (2006). *Nonparametric functional data analysis: theory and practice*. Springer Science & Business Media.

Ferraty, F. and Vieu, P. (2009). Additive prediction and boosting for functional data. *Computational Statistics & Data Analysis*, 53(4):1400–1413.

Ferré, L. and Yao, A.-F. (2003). Functional sliced inverse regression analysis. *Statistics*, 37(6):475–488.

Fisher, A., Rudin, C., and Dominici, F. (2019). All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression. *Journal of the American statistical Association*, 76(376):817–823.

Fukunaga, K. and Koontz, W. L. (1970). Representation of random processes using the finite karhunen-loeve expansion. *Information and Control*, 16(1):85–101.

Geenens, G. et al. (2011). Curse of dimensionality and related issues in nonparametric functional regression. *Statistics Surveys*, 5:30–43.

Gheriballah, A., Laksaci, A., and Sekkal, S. (2013). Nonparametric m-regression for functional ergodic data. *Statistics & Probability Letters*, 83(3):902–908.

Goldsmith, J., Scheipl, F., Huang, L., Wrobel, J., Di, C., Gellar, J., Harezlak, J., McLean, M. W., Swihart, B., Xiao, L., Crainiceanu, C., and Reiss, P. T. (2020). *refund: Regression with Functional Data*. R package version 0.1-23.

Greven, S. and Scheipl, F. (2017). A general framework for functional regression modelling. *Statistical Modelling*, 17(1-2):1–35.

Hall, P., Horowitz, J. L., et al. (2007). Methodology and convergence rates for functional linear regression. *The Annals of Statistics*, 35(1):70–91.

Hampel, F. R. (1968). *Contributions to the theory of robust estimation*. University of California, Berkeley.

Hastie, T. and Mallows, C. (1993). A statistical view of some chemometrics regression tools. *Technometrics*, 35(2):140–143.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized additive models*. Chapman & Hall/CRC.

Hössjer, O. (1992). On the optimality of s-estimators. *Statistics & probability letters*, 14(5):413–419.

Hössjer, O. (1992). On the optimality of S-estimators. *Statistics and Probability Letters*, 14(5):413–419.

Huang, L., Wang, H., Cui, H., and Wang, S. (2015). Sieve m-estimator for a semi-functional linear model. *Science China Mathematics*, 58(11):2421–2434.

Huang, L., Wang, H., and Zheng, A. (2014). The m-estimator for functional linear regression model. *Statistics & Probability Letters*, 88:165–173.

Huber, P. J. (1973). Robust regression: asymptotics, conjectures and monte carlo. *The annals of statistics*, pages 799–821.

Hubert, M., Rousseeuw, P. J., and Segaert, P. (2015). Multivariate functional outlier detection. *Statistical Methods & Applications*, 24(2):177–202.

Hyndman, R. J. and Shang, H. L. (2010). Rainbow plots, bagplots, and boxplots for functional data. *Journal of Computational and Graphical Statistics*, 19(1):29–45.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2017). *ISLR: Data for an Introduction to Statistical Learning with Applications in R*. R package version 1.2.

James, G. M. (2002). Generalized linear models with functional predictors. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(3):411–432.

James, G. M. and Silverman, B. W. (2005). Functional adaptive model estimation. *Journal of the American Statistical Association*, 100(470):565–576.

Ju, X. and Salibián-Barrera, M. (2021). Robust boosting for regression problems. *Computational Statistics & Data Analysis*, 153:107065.

Kaid, Z. and Laksaci, A. (2017). Functional quantile regression: local linear modelisation. In *Functional Statistics and Related Fields*, pages 155–160. Springer.

Kalogridis, I. and Van Aelst, S. (2019). Robust functional regression based on principal components. *Journal of Multivariate Analysis*, 173:393–415.

Kalogridis, I. and Van Aelst, S. (2021). Robust penalized estimators for functional linear regression.

Kara, L.-Z., Laksaci, A., Rachdi, M., and Vieu, P. (2017). Data-driven knn estimation in nonparametric functional data analysis. *Journal of Multivariate Analysis*, 153:176–188.

Kato, K. (2012). Estimation in functional linear quantile regression. *The Annals of Statistics*, 40(6):3108–3136.

Kazemitabar, J., Amini, A., Bloniarz, A., and Talwalkar, A. S. (2017). Variable importance using decision trees. *Advances in Neural Information Processing Systems*, 30:426–435.

Kudraszow, N. L. and Vieu, P. (2013). Uniform consistency of knn regressors for functional variables. *Statistics & Probability Letters*, 83(8):1863–1870.

Laksaci, A., Lemdani, M., and Ould-Saïd, E. (2009). A generalized-approach for a kernel estimator of conditional quantile with functional regressors: Consistency and asymptotic normality. *Statistics and Probability Letters*, 79(8):1065.

Li, K.-C. (1991). Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, 86(414):316–327.

Lian, H. and Li, G. (2014). Series expansion for functional sufficient dimension reduction. *Journal of Multivariate Analysis*, 124:150–165.

Liebl, D. et al. (2013). Modeling and forecasting electricity spot prices: A functional data perspective. *The Annals of Applied Statistics*, 7(3):1562–1592.

Ling, N. and Vieu, P. (2018). Nonparametric modelling for functional data: selected survey and tracks for future. *Statistics*, 52(4):934–949.

Ling, N. and Vieu, P. (2020). On semiparametric regression in functional data analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, page e1538.

Lutz, R. W., Kalisch, M., and Bühlmann, P. (2008). Robustified l2 boosting. *Computational Statistics & Data Analysis*, 52(7):3331–3341.

Maronna, R. A., Martin, R. D., Yohai, V. J., and Salibián-Barrera, M. (2019). *Robust statistics: theory and methods (with R)*. John Wiley & Sons.

Maronna, R. A. and Yohai, V. J. (2000). Robust regression with both continuous and categorical predictors. *Journal of Statistical Planning and Inference*, 89(1-2):197–214.

Maronna, R. A. and Yohai, V. J. (2013). Robust functional linear regression based on splines. *Computational Statistics & Data Analysis*, 65:46–55.

Mas, A. et al. (2012). Lower bound in regression for functional data by representation of small ball probabilities. *Electronic Journal of Statistics*, 6:1745–1778.

McLean, M. W., Hooker, G., Staicu, A.-M., Scheipl, F., and Ruppert, D. (2014). Functional generalized additive models. *Journal of Computational and Graphical Statistics*, 23(1):249–269.

Mevik, B.-H., Wehrens, R., and Liland, K. H. (2019). *pls: Partial Least Squares and Principal Component Regression*. R package version 2.7-2.

Müller, H.-G., Stadtmüller, U., et al. (2005). Generalized functional linear models. *the Annals of Statistics*, 33(2):774–805.

Müller, H.-G., Wu, Y., and Yao, F. (2013). Continuously additive models for nonlinear functional regression. *Biometrika*, 100(3):607–622.

Müller, H.-G. and Yao, F. (2008). Functional additive models. *Journal of the American Statistical Association*, 103(484):1534–1544.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

Nash, W. J., Sellers, T. L., Talbot, S. R., Cawthorn, A. J., and Ford, W. B. (1994). The population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report*, 48.

Plaut, D. C. et al. (1986). Experiments on learning by back propagation.

Preda, C. (2007). Regression models for functional data by reproducing kernel hilbert spaces methods. *Journal of statistical planning and inference*, 137(3):829–840.

Qingguo, T. (2017). M-estimation for functional linear regression. *Communications in Statistics-Theory and Methods*, 46(8):3782–3800.

Qingguo, T. and Kong, L. (2017). Quantile regression in functional linear semiparametric model. *Statistics*, 51(6):1342–1358.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ramsey, J. O. and Silverman, B. W. (2005). Functional data analysis. *Springer Series in Statistics, New York: Springer Verlag*.

Reiss, P. T. and Ogden, R. T. (2007). Functional principal component regression and functional partial least squares. *Journal of the American Statistical Association*, 102(479):984–996.

Ridgeway, G., Madigan, D., and Richardson, T. (1999). Boosting methodology for regression problems. *Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics*, pages 152–161.

Rousseeuw, P. and Yohai, V. (1984). Robust regression by means of s-estimators. In *Robust and nonlinear time series analysis*, pages 256–272. Springer.

Sang, P. and Cao, J. (2020). Functional single-index quantile regression models. *Statistics and Computing*, pages 1–11.

Serneels, S., Croux, C., Filzmoser, P., and Van Espen, P. J. (2005). Partial robust M-regression. *Chemometrics and Intelligent Laboratory Systems*, 79(1-2):55–64.

Shang, H. L. (2016). A bayesian approach for determining the optimal semi-metric and bandwidth in scalar-on-function quantile regression with unknown error density and dependent functional data. *Journal of Multivariate Analysis*, 146:95–104.

Shin, H. and Lee, S. (2016). An rkhs approach to robust functional linear regression. *Statistica Sinica*, pages 255–272.

Shmueli, G. (2010). To explain or to predict? *Statistical science*, 25(3):289–310.

Sigrist, F. (2018). Gradient and newton boosting for classification and regression. *arXiv preprint arXiv:1808.03064*.

Silverman, B. W. (1996). Smoothed functional principal components analysis by choice of norm. *The Annals of Statistics*, 24(1):1–24.

Stone, C. J. (1985). Additive regression and other nonparametric models. *The annals of Statistics*, pages 689–705.

Tang, Q. and Cheng, L. (2014). Partial functional linear quantile regression. *Science China Mathematics*, 57(12):2589–2608.

Telgarsky, M. (2013). Margins, shrinkage, and boosting. In *International Conference on Machine Learning*, pages 307–315. PMLR.

Therneau, T. and Atkinson, B. (2019). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-15.

Tutz, G. and Gertheiss, J. (2010). Feature extraction in signal regression: a boosting technique for functional data regression. *Journal of Computational and Graphical Statistics*, 19(1):154–174.

Ullah, S. and Finch, C. F. (2013). Applications of functional data analysis: A systematic review. *BMC medical research methodology*, 13(1):1–12.

Wahba, G. (1990). *Spline models for observational data*. SIAM.

Wang, G., Lin, N., and Zhang, B. (2014). Functional k-means inverse regression. *Computational Statistics & Data Analysis*, 70:172–182.

Wang, G., Zhou, J., Wu, W., and Chen, M. (2017). Robust functional sliced inverse regression. *Statistical papers*, 58(1):227–245.

Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372.

Wood, S. N. (2017). *Generalized additive models: an introduction with R (2nd ed.)*. Chapman and Hall/CRC.

Xia, Y., Tong, H., Li, W. K., and Zhu, L.-X. (2002). An adaptive estimation of dimension reduction space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(3):363–410.

Yao, F., Müller, H.-G., and Wang, J.-L. (2005). Functional data analysis for sparse longitudinal data. *Journal of the American statistical association*, 100(470):577–590.

Yohai, V. J. (1987). High breakdown-point and high efficiency robust estimates for regression. *The Annals of statistics*, pages 642–656.

Yu, C. and Yao, W. (2017). Robust linear regression: A review and comparison. *Communications in Statistics-Simulation and Computation*, 46(8):6261–6282.

Zhao, Y., Ogden, R. T., and Reiss, P. T. (2012). Wavelet-based lasso in functional linear regression. *Journal of computational and graphical statistics*, 21(3):600–617.

Zhou, Z.-H. (2019). *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.

Appendix A

First Appendix

Here you can have your appendices. Note that if you only have a single appendix, you should issue `\renewcommand{\appendicesname}{Appendix}` before calling `\appendix` to display the singular "Appendix" rather than the default plural "Appendices".

Appendix B

Second Appendix

Here is the second appendix.

Additional Information

This chapter shows you how to include additional information in your thesis, the removal of which will not affect the submission. Such material should be removed before the thesis is actually submitted.

First, the chapter is unnumbered and not included in the Table of Contents. Second, it is the last section of the thesis, so its removal will not alter any of the page numbering etc. for the previous sections. Do not include any floats, however, as these will appear in the initial lists.

The `ubcthesis` LaTeX class has been designed to aid you in producing a thesis that conforms to the requirements of The University of British Columbia Faculty of Graduate Studies (FoGS).

Proper use of this class and sample is highly recommended—and should produce a well formatted document that meets the FoGS requirement. Notwithstanding, complex theses may require additional formatting that may conflict with some of the requirements. We therefore *highly recommend* that you consult one of the FoGS staff for assistance and an assessment of potential problems *before* starting final draft.

While we have attemped to address most of the thesis formatting requirements in these files, they do not constitute an official set of thesis requirements. The official requirements are available at the following section of the FoGS web site:

`http://www.grad.ubc.ca/current-students/dissertation-thesis-preparation`

We recommend that you review these instructions carefully.