UNIVERSITY OF CALIFORNIA
RIVERSIDE

Essays in the Causal Inference and Economic Forecasting Using Machine Learning

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Economics

by

Ran Wang

June 2020

Dissertation Committee:

Professor Tae-Hwy Lee, Co-Chairperson
Professor Aman Ullah, Co-Chairperson
Professor Ruoyao Shi

The Dissertation of Ran Wang is approved:

Committee Co-Chairperson

Committee Co-Chairperson

University of California, Riverside

Acknowledgments

The past five years witnessed the most fruitful period in my life. I would like to take this opportunity to express my sincere gratitude to many people for their contributions to my graduate study and research.

I would like to thank my advisors, Professor Tae-Hwy Lee and Professor Aman Ullah. Without their supports, this thesis would not have been contributed possibly. Professor Tae-Hwy Lee helped me in the econometrics and forecasting and supported my academic works and new ideas in causal inference. Professor Aman Ullah taught me how good econometric research is done and guided me in the foundation of nonparametric econometrics and statistics. I appreciate all their contributions of time and ideas to make my Ph.D. experience productive.

I also am grateful to the rest of my committee members, Professor Ruoyao Shi, for her great support and invaluable advice. Professor Shi provided me extensive personal and professional guidance and taught me a great deal about econometric researches and job market. I also appreciate my supervisor during the internship at the International Monetary Fund, Dr. Jorge A. Chan-Lau. His supports made my internship research work fruitful.

I extend my thanks to the cohort of 2014 and 2015 econ students. They are Yun Luo, Jianghao Chu, Mingyuan Jia, Feng Xiong, Shahnaz Parsaeian, Seolah Kim, and all the alumni who have helped me during my Ph.D. life. I also extend my thanks to my friends met in the academic conferences. They are Weiran Deng, Shuyang Du and Sihong Chen. They are supportive of both my life and research.

To my parents and family for all the support.

ABSTRACT OF THE DISSERTATION

Essays in the Causal Inference and Economic Forecasting Using Machine Learning

by

Ran Wang

Doctor of Philosophy, Graduate Program in Economics
University of California, Riverside, June 2020
Professor Tae-Hwy Lee, Co-Chairperson
Professor Aman Ullah, Co-Chairperson

This dissertation discusses the application of machine learning techniques on the economic causal inference and forecasting.

In Chapter 2, we develop a new method, the L_1-regularized soft decision tree, to identify the relevant features for the heterogeneous treatment effect and confounding effect under a very flexible nonlinear potential outcomes framework for causal inference. Compared to other methods, we show that our approach can identify relevant factors without the widely used assumption of additive nonlinearity. By embedding the debiased soft decision tree into the L_1-based variable selection framework, we show that the L_1-regularized soft decision tree outperforms other variable selection methods such as lasso in nonlinear settings.

Chapter 3 focuses on macroeconomic forecasting literature. This chapter introduces unFEAR, an unsupervised feature extraction clustering method aimed at facilitating crisis prediction tasks. We use unsupervised representation learning and a novel autoencoder method to extract from economic data information relevant to identify time-invariant

non-overlapping clusters comprising observed crisis and non-crisis episodes. Each cluster corresponds to a different economic regime characterized by an idiosyncratic crisis generating mechanism.

Chapters 4 and 5 summarize the literature of economic forecasting using two attractive machine learning techniques. Chapter 4 focuses on the Bagging and Random Forests methods. We explore Bagging, Random Forest, and their variants in various aspects of theory and practice. We also discuss applications based on these methods in economic forecasting and inference. Chapter 5 discusses Boosting. Boosting can estimate the variables of interest consistently under fairly general conditions given a large set of explanatory variables. Boosting is fast and easy to implement, which makes it one of the most popular machine learning algorithms in academia and industry.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

These years witness the dramatically fast development of data science and machine learning techniques are widely introduced on the economic causal inference and forecasting. On the one hand, there are many revised machine learning techniques emerging in the classic economic causal inference literature for better statistical inference and prediction. On the other hand, because machine learning can approximate unknown functions with fewer assumptions than economic structure models, economists introduce these powerful tools into economic forecasting and make very fruitful outcomes.

This dissertation aims to contribute to the theories and applications of machine learning techniques into economics. In Chapter 2, we introduce the L_1-regularized soft decision tree to identify the relevant features for the heterogeneous treatment effect and confounding effect under a very flexible nonlinear potential outcomes framework for causal inference. We firstly discuss the L_1-based variable selection framework in the nonlinear regression situation with oracle properties. Then, we propose the debiased soft decision

tree method and show its consistency and asymptotic normality. Next, we combine the L_1-based variable selection framework with the debiased soft decision tree to propose the L_1-regularized soft decision tree. By simulation studies, we show the importance of nonlinear variable selection in the heterogeneous treatment literature and show that the L_1-regularized soft decision tree outperforms other variable selection methods.

Chapter 3 is a joint work with Jorge A. Chan-Lau. This chapter introduces unFEAR, an unsupervised feature extraction clustering method aimed at facilitating crisis prediction tasks. We proposed a novel mode contrastive autoencoder extracts from economic data information relevant to identify time-invariant non-overlapping clusters comprising observed crisis and non-crisis episodes. Additionally, we illustrated that the common issues of the macroeconomic data, like missing values and nonlinear time trends, can be resolved by our proposed method.

Chapters 4 and Chapters 5 review the literature of economic forecasting using Bagging and Boosting. Chapter 4 is a joint work with Tae-Hwy Lee and Aman Ullah. This chapter focuses on the Bagging and Random Forests methods. We introduce theories of Bagging, Random Forest, and their variants in various aspects of economic forecasting and inference. Chapter 5 is a joint work with Jianghao Chu, Tae-Hwy Lee, and Aman Ullah. Chapter 5 discusses Boosting, which can estimate the variables of interest consistently under reasonably general conditions given a large set of explanatory variables. Boosting is fast and easy to implement, making it one of the most popular machine learning algorithms in academia and industry. Chapter 4 and Chapter 5 are published in the book *Macroeconomic Forecasting in the Era of Big Data: Theory and Practice* by Springer.

Chapter 2

Identifying Heterogeneous and Confounding Effect via an L_1-Regularized Soft Decision Tree

2.1 Introduction

Causal inference is a widely discussed topic in statistics, computer science, and many social sciences such as economics and politics. There are two main methods for causal inference: the structural causal model (SCM), which focuses on the identification of the causal structure [see Pearl (2009), Pearl (2018)], and the treatment effect model [see Rubin and Imbens (2017)], which concentrates on the estimation and inference of the treatment's magnitude. In the economic literature, the treatment effect method is usually the primary method for analyzing the causal effect in econometrics and applied economics.

In the setting of the classic random experiment, the potential outcomes framework is the foundation of many causal inference methods in economic studies discussed by Rubin (1974). To the observational data, propensity score matching [see Rosenbaum and Rubin (1983)] and doubly robust estimators [see Lunceford and Davidian (2004)] are proposed and discussed for the unbiased estimation of the treatment effect given the confounders.

Recently, machine learning techniques have been increasingly introduced into economics [see Athey (2017), Athey and Imbens (2019)]. Because of the available large economic datasets and low-cost computational resources, machine learning methods can approximate very complicated unknown functions and handle the high-dimensional inputs issue [see Varian (2014)]. For example, Richardson et al (2018) applied random forest, boosting, and neural network methods to economic nowcasting given a variety of covariates. Additionally, to study the topics in econometrics, such as GMM and instrumental variables, there has been an increasing number of new methods proposed by economists based on classic machine learning algorithms for economic inference [see Lewis and Syrgkanis (2018), Hartford, Lewis, Leyton-Brown and Taddy (2016), Athey, Tibshirani and Wager (2019)].

Early machine learning research in causal inference began in the 2000s. van der Laan and Rubin (2006) proposed the targeted learning framework for causal effect estimation. Chipman, George and McCulloch proposed Bayesian additive regression trees and estimated the causal effect based on the posterior distribution generated by the model. Athey and Imbens (2006) discussed an unbiased ML estimator, the honest tree, for testing the treatment effect. Wager and Athey (2018) proposed causal forests (CF) for esti-

mating the heterogeneous treatment effect with unbiasedness and asymptotic normality. Chernozhukov, Chetverikov, Demirer, Duflo, Hansen and Newey (2018) proposed double machine learning (DML) based on the partially linear model, which shares the similar idea with the residual-in-residual method discussed by Robinson (1988).

The recent developments in causal inference are mainly focusing on the treatment effect estimation. Consider a dataset from a randomized control trial (RCT) experiment with N samples. Sample i contains a treatment indicator $D_i \in \{0, 1\}$ and a response Y_i^D is a real number. The average treatment effect (ATE) τ is illustrated in

$$\tau = E(Y^1) - E(Y^0). \tag{2.1}$$

In Equation (2.1), Y^1, Y^0 are the responses corresponding to the treatment group and control group. The RCT is often unsatisfied, especially for observational data. One common case of violation of the RCT is confoundedness. There may exist one or more common factors distorting the true dependence between the treatment D and response Y. The partially linear regression (PLR) framework describes the issue:

$$\begin{aligned} Y &= \phi_0 D + m_y(C) + e, \\ P(D = 1) &= m_d(C). \end{aligned} \tag{2.2}$$

In Equation (2.2), Y is the outcome, e is a random error term where $E(e|C, D) = 0$, D is a binary treatment with the structural parameter ϕ_0, and C represents the confounders. $m_y(C)$ and $m_d(C)$ are nonlinear functions.

Hypothetically, if confounders C are observable and accessible, we can directly remove their effects. Chernozhukov, Chetverikov, Demirer, Duflo, Hansen and Newey (2017) proposed double machine learning (DML) based on the partially linear model. In the earily years, Robinson (1988) discussed a residual-in-residual method discussed by which provides a flexible estimator given many attributes. The two approaches share very similar inspiration in that they approximate the nonlinear functions $m_y(C)$ and $m_d(C)$ via nonparametric regression or machine learning methods.

In Equation (2.2), we assume all the features X as the confounders and assume that the treatment effect is a constant $\tau = \phi_0$. In contrast, the features X can lead to the heterogeneous treatment effect such that the treatment effect ϕ_0 is a function of attributes X, $\phi_0 = \tau(X)$. Based on this assumption, we have the heterogeneous treatment effect framework:

$$\begin{aligned} Y &= m(D, X) + u, \\ P(D = 1) &= e(X). \end{aligned} \tag{2.3}$$

In Equation (2.3), Y is the outcome, u is a random error term where $E(u|X, D) = 0$, and D is a binary treatment, $m(D, X)$ is a function for the outcomes. $e(X)$ is the propensity score. Thus, the heterogeneous treatment effect is $\tau(X) = m(1, X) - m(0, X)$. All the features X are treated as heterogeneous factors H. Athey and Imbens (2016) was the first paper to explore the application of decision tree in unbiased treatment effect. The key to the honest tree is sample splitting. That is, one group of samples is selected to grow a tree, and the other group is used for calculating the predicted value in each leaf. Wager and Athey (2018) discussed the Causal Forest (CF) such that the heterogeneous treatment

effect is estimated consistently with asymptotic normality.

Precisely, the DML method estimates the unbiased treatment effect by choosing all the factors as the confounders. The CF method considers all factors to estimate the heterogeneous effect. Thus, how to choose the variables for heterogeneous effects and confounding effects is a crucial question. To solve this issue, we consider a generalized partially linear regression framework (GPLR) in Equation (2.4).

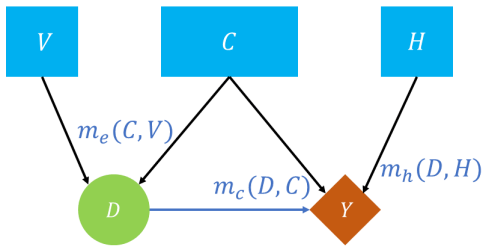

Figure 2.1: Generalized Partially Linear Regression

$$\begin{aligned} Y_D(H,C) &= m_h(D,H) + m_c(D,C) + u, \\ P(D=1) &= m_e(C,V), \end{aligned} \tag{2.4}$$

In Equation (2.4), Y is the outcome, u is a random error term where $E(u|C,D,H) = 0$, D is the treatment indicator, H, C and V represent the heterogeneous factors, confounders and instrumental variables. $m_h(D,H)$, $m_c(D,C)$ and $m_e(C,V)$ are nonlinear functions. Figure 2.1 illustrates the structure of the generalized partially linear regression framework.

In practice, the H, C and V are always contained in one variable set X with many candidates. Hence, the core problem is to select H and C from one group of candidates X. Belloni, Chernozhukov and Hansen (2013) discussed a related variable selection problem. The authors implemented a Double-Lasso method on equations Y_D and D to identify the relevant confounders C given many covariates. Then, the selected variables are combined to determine the correct C. Luo and Spindler (2017) discussed an L_2-boosting-based double selection method for the similar settings of high-dimensional controls under linear or additive linear settings. However, there are few researches discussing how to identify H and C when the functional form of $m_h(D, H)$, $m_c(D, C)$ and $m_e(C, V)$ are unknown. Thus, based on the generalized partially linear regression, we suggest the following procedure:

- Estimate propensity score $e(C, V)$ and identify the factors including confounders C and instrumental variables V.
- Regress $Y_D(H, C)$ on all the factors to select the confounders C and heterogeneous factors H.
- Regress $Y_D(H, C)$ on C to estimate $m_c(D, C)$.
- Estimate unbiased heterogeneous HTE $\tau(H)$ based on the selected variables for H.

The main contribution of this chapter is that we propose a new framework for identifying the three factors without assuming the form of the three unknown functions. Based on a generalized partially linear framework, we extend the Double Lasso estimator, Double Machine Learning (DML) and Causal Forest (CF) and propose our L_1-Regularized Soft Decision Tree (L_1 SDT), which can consistently identify the relevant attributes for

heterogeneous factors and confounders given a large number of covariates under a nonlinear setting. Instead of choosing all the covariates in other methods, our method resolves the issue of high-dimensionality by selecting the relevant variables for causal inference.

Our first contribution is that we construct the asymptotic properties of the soft decision tree. Under assumptions, we show that the soft decision tree is equivalent to a data-adaptive kernel regression with finite kernels and then can be unbiased to the true unknown function asymptotically. To our knowledge, that is the first time to discuss the related theories on the soft decision tree for inference instead of prediction. Compared to other decision tree methods, the soft decision tree provides a differentiable function for marginal analysis, which is essential in economic researches. More importantly, it supports the consistent nonlinear variable selection procedure we proposed in this chapter.

The second contribution is that we extend the linear adaptive Lasso method to the nonlinear scenario. We firstly discuss the traditional nonlinear variable selection methods and indicate that the two sources of bias introduced by these methods. Next, we propose a general framework for consistent nonlinear variable selection, inference and we also generalize the oracle properties in the linear case to our framework. Based on the framework, we move forward to propose the L_1-regularized soft decision tree and show that the L_1-regularized soft decision tree holds the oracle properties for consistently nonlinear variable selection. Compared to other variable selection techniques, we need much weaker conditions of the unknown function instead of the linear or additive nonlinear assumptions to the functions.

This chapter is organized as follows. In Section 2.2, we introduce the L_1-based

nonlinear variable selection framework and discuss the possible issues about bias affecting the consistency of variable selection. In Section 2.3, we introduce a debiased soft decision tree with its asymptotic properties. In Section 2.4 and we introduce the consistent variable selection via L_1 regularized soft decision tree and show its oracle properties for nonlinear variable selection. Then, we provide a simulation study for checking the performance in Section 2.5. Finally, an empirical study on unemployment bonus insurance is given based on our new method in Section 2.6. All proofs are gathered in Appendix A.

2.2 Nonlinear Variable Selection Framework

2.2.1 Variable Selection via L_1 Penalty

Variable selection is an essential topic in statistics, economics, and other modeling problems that suffer from the ultrahigh dimensional input space. In the linear variable selection case, L_1 regularization methods can simultaneously select essential variables.

We firstly define the data generating process for the following analysis. Given the sample $(y_i, x_i), i = 1, ..., N$, where y_i is a real number and $x_i = (x_{1,i}, ..., x_{p,i})$ is a p dimensional vector, we define that the data are generated from Equation

$$y_i = E(y_i|x_i) + u_i = f(x_i) + u_i. \tag{2.5}$$

In Equation (2.5), $f(x)$ is a unknown function and u_i is a random error following a distribution $u_i \sim p(u)$.

Consider the linear regression model $f(x) = \beta x$. $\hat{\beta}_{OLS} = (X^T X)^{-1} X^T Y$ is the optimal estimator when all the settings of the simple linear regression are satisfied. However,

if we assume that β is sparse, the OLS estimator is not optimal since it usually gives every element a nonzero value. The Lasso method was introduced in Tibshirani (1996), which suggests penalizing the L_1 norm of coefficient vector β for variable selection. Thus, the new loss function is

$$L_{LASSO} = \sum_{i=1}^{N}(y_i - \beta x_i)^2 + \lambda||\beta||_1. \tag{2.6}$$

Generally, consider the case when $f(x)$ is an unknown function. We can regress y on all x via a nonlinear regression $\hat{f}(X)$ such that

$$\hat{y} = \hat{f}(x). \tag{2.7}$$

Suppose only part of $x_{sparse} \subseteq x$ are relevant in the true data generating process $y = f(x_{sparse})$, we may introduce some irrelevant variables in the estimator. For example, assume the true model is $y = f(x) + u$ and $x = (x_1, x_2)$ is a two dimensional vector. The nonlinear regression model should be

$$y = f(x_1, x_2) + v. \tag{2.8}$$

However, if the true data generating process is $y = f(x_1) + u$, we cannot identify the correct $f(x)$. To select the relevant covariates given the unknown function $f(x)$, a natural way is to generalize the Lasso regression into the case when $f(x)$ is unknown.

Back to the example in Equation (2.8) , if x_2 is irrelevant with y, the partial derivative $\frac{\partial f(x_1,x_2)}{\partial x_2}$ should be zero. Then, $\frac{\partial f(x_1,x_2)}{\partial x_1} = \frac{\partial f(x_1)}{\partial x_1}$ to any x_1, x_2. Thus, we can select a variable by penalizing the norm of the first derivative of $\frac{\partial f}{\partial x}$ when the functions

given by nonlinear regression methods are differentiable.

$$Loss_f = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 + \lambda\sum_{p=1}^{P}\left|\left|\frac{\partial f_p(x)}{\partial x_p}\right|\right|_2 \tag{2.9}$$

Equation (2.9) illustrates a nonlinear variable selection framework, where the penalty term $\left|\left|\frac{\partial f_p(x)}{\partial x_p}\right|\right|_2$ is the L_2 norm of the first derivative function

$$\left|\left|\frac{\partial f}{\partial x_p}\right|\right|_2 = \sqrt{\int_{x_p}\left(\frac{\partial f}{\partial x_p}\right)^2 dx_p}. \tag{2.10}$$

In the linear Lasso, the penalty term $|\beta_1|+|\beta_2|+...+|\beta_P|$ is a sum of the absolutes of β. In the nonlinear Lasso in Equation (2.9), we substitute the absolute values to the L_2 norms of partial derivative f' and new penalty term is $||f_1'||_2 + ||f_2'||_2 + ... + ||f_P'||_2$. Since we use L_2 norm to measure the magnitude of f' and choose L_1 to compare different norms, this method is called "$l1/l2$" method [see Rosasco et al (2013)].

To calculate the $||f_p'||_2 = \left|\left|\frac{\partial f}{\partial x_p}\right|\right|_2$ on sample, we can implement the discrete version to approximate the L_2 norm

$$\left|\left|\frac{\partial f}{\partial x_p}\right|\right|_2 = \sum_{p=1}^{P}\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial f}{\partial x_p}\Big|_{x_p=x_{p,i}}\right)^2}. \tag{2.11}$$

Many nonlinear regression methods assume the sparse additive formation $f(x) = \sum_{p=1}^{P} f_p(x_{i,p})$ and the loss function in Equation (2.9) becomes:

$$Loss_f = \frac{1}{N}\sum_{i=1}^{N}(y_i - \sum_{p=1}^{P} f_p(x_{i,p}))^2 + \lambda\sum_{p=1}^{P}\left|\left|\frac{\partial f_p(x)}{\partial x_p}\right|\right|_2. \tag{2.12}$$

The nonlinear variable selection method based on loss function in Equation (2.12) is also called the Component Selection and Smoothing Operator (COSSO) proposed by

Lin and Zhang (2006). The sparse additive formation is widely used for nonlinear variable selection. However, the sparse additive assumption is one special and strong case for $f(x)$ and may not handle all the possibilities, especially the interaction term, x_1x_2, in $f(x)$. Thus, the sparse additive model-based nonlinear selection result may not have the same consistent selection results in linear variable selection.

2.2.2 Consistent Nonlinear Variable Selection

Necessarily, the Lasso type variable selection method may not give the correct subgroup of variables unless some essential conditions and assumptions are satisfied. Many studies have explored the consistency of the Lasso type linear variable selection methods [see Fan and Li (2001), Zou (2006)].

Given the Lasso regression in Equation (2.6), Zou (2006) introduces Lemma 1 and Lemma 2 related to the variable selection properties.

Lemma 1 *If* $\lambda/\sqrt{N} \to \lambda_0 \geq 0$, *then* $\limsup_n P(\beta \neq 0 | \beta \in A) \leq c < 1$, *where* A *is the ground truth set of the non-zero* βs *and* c *is a contant depending on the true model.*

Lemma 2 *If* $\lambda/N \to 0$ *and* $\lambda/\sqrt{N} \to \infty$, *then* $\frac{N}{\lambda}(\hat{\beta} - \beta) \to \arg\min(V_3)$, *where* $V_3(u) = u^T C u + \sum_{p=1}^{P} (u_p sign(\beta_p) I(\beta_p \neq 0) + |u_p| I(\beta_p \neq 0))$ *and* C *is the covariance matrix of regressor* x.

Based on Lemma 1 and 2, the variable selection is not consistent if $\lambda \sim O(\sqrt{N})$. Also, when the λ increases faster than than $\sqrt{N}$ but slower than N, the convergence rate of $\hat{\beta}$ is $\frac{N}{\lambda}$ and it is slower than $\sqrt{N}$. In other words, the fast convergence rate and consistent

variable selection cannot hold for the same λ. Furthermore, Yuan and Lin (2005) derived the necessary condition for the consistent variable selection.

Unfortunately, the issues become even worse in the nonlinear variable selection situation. Considered the Lasso-based nonlinear variable selection in Equation (2.9). Since the ground truth $f_0(x)$ is unknown, we may not know if the nonlinear approximator $f(x;\theta)$ can correctly fit the true $f(x) = E(y|x)$. If $f(x;\theta)$ is a inconsistent approximator to $f_0(x)$, the norm of partial derivative $\frac{\partial f(x;\theta)}{\partial x_p}$ would also be inconsistent and the related nonlinear variable selection results are inconsistent.

Even though there exists a θ such that $f(x;\theta) = f_0(x) = E(y|x)$ and $\frac{\partial f(x;\theta)}{\partial x_p} = \frac{\partial f_0(x)}{\partial x_p}$, the nonlinear variable selection framework may suffer from the low convergence rate as same as the Lasso-based linear variable selection. Suppose $|f(x;\hat{\theta}) - f_0(x)| = O_p(N^{-\alpha})$ and $0 < \alpha < 1/2$. We can generalize the Lemma 2 to the nonlinear situation. Now we need the $\lambda/N^{\alpha} \to \infty$ and $\lambda/N \to 0$ such that the nonlinear variable selection is consistent. However, under these conditions, the rate of convergence is even slower than $N^{-\alpha} < \sqrt{N}$.

To resolve the two issues in the nonlinear variable selection, we need to bound the first bias via constructing an unbiased and consistent nonlinear approximation $E(f(x;\hat{\theta})) \to f_0(x)$. Then, the slower convergence rate can be resolved by the idea of the Adaptive Lasso method proposed by Zou (2006).

Definition 3 *A variable selection estimator satisfies oracle properties if and only if the two following conditions are satisfied:*

- *Consistency in variable selection:* $\lim_{n\to\infty} P(A_n^* = A) = 1$.
- *Asymptotic normality:* $\sqrt{n}(\hat{\beta}_A^{*(n)} - \beta_A^*) \to_d \ N(0, \Sigma)$, *where A is a set including all*

the relevant variables and Σ *is the covariance matrix for* $\hat{\beta}_A^{*(n)}$.

Many variable selection methods were developed with oracle properties, such as Adaptive Lasso proposed by Zou (2006) and SCAD introduced by Fan and Li (2001). The main idea about these methods are similar: for different magnitudes of the coefficients, the penalty terms can give different weights.

$$L_{ADALASSO} = \sum_{i=1}^{N}(y_i - \beta x_i)^2 + \lambda \sum_{p=1}^{P} \frac{1}{w_p}|\beta_p| \tag{2.13}$$

Equation (2.13) illustrates the adaptive lassos loss function, where $w_p = ||\hat{\beta}_{OLS}||^{\gamma}$. Adaptive lasso gives a large penalty to the coefficient with a smaller estimated OLS value. Zou (2006) proved that under some conditions on γ and λ, adaptive lasso could consistently select correct variables. Thus, these oracle methods guarantee the variable selection and asymptotic normality.

To our knowledge, consistent nonlinear variable selection and the related oracle properties have not been widely discussed. However, this is a very important problem. In linear model selection, we often select the best subgroup of variables and then regress on the chosen variables, which is called post-selection analysis. The oracle properties can support the analysis by guaranteeing the variable selection. In nonlinear variable selection, selecting the correct variable is even more important in terms of the curse of dimensionality. Hence, consistent variable selection can resolve this issue by identifying correct regressors.

In Equation (2.9), similar to the original lasso in the linear regression, there is no guarantee that the L_1-based nonlinear variable selection method can consistently retrieve all the relevant variables. Thus, we introduce the adaptive L_1 penalty term to the loss

function and conclude the result in Proposition 4.

Proposition 4 *Suppose the data generating process is $y = f(x) + \epsilon$, where $f(x)$ is a continuous function that can be consistently approximated by a function $f(x;\theta)$. θ is optimized via the following regularized loss function in Equation (2.14).*

$$Loss_f = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i;\theta))^2 + \lambda\sum_{p=1}^{P}\hat{w}_p\left|\left|\frac{\partial f(x;\theta)}{\partial x_p}\right|\right|_2. \tag{2.14}$$

In Equation (2.14), $\hat{w}_p = \frac{1}{\left|\left|\frac{\partial \hat{f}}{\partial x_p}\right|\right|_2^{\gamma}}$. If $|f(x;\hat{\theta}) - f(x)| \sim O_p(N^{-\alpha})$ and there exists λ and γ such that $\lambda(N^{-\alpha})^{\frac{\gamma-1}{2}} \to \infty$ and $\lambda N^{-\alpha} \to 0$, the estimator $\hat{f} = f(x;\hat{\theta})$ has the following oracle properties:

- *Consistency in variable selection:*

$$\lim_n \; P\left(\left|\left|\frac{\partial \hat{f}}{\partial x_p}\right|\right|_2 \neq 0 \;\middle|\; p \in A\right) = 1.$$

- *Pointwise asymptotic normality:*

$$\frac{\hat{f}(x;\theta) - f(x)}{\sqrt{Var(\hat{f}(x;\theta))}} \to_d N(0,1).$$

The framework in Equation (2.15) provides a very flexible way for consistent nonlinear variable selection. Compared to other methods, our approach does not require strong assumptions on the nonlinear function, such as the additive nonlinear form. More importantly, our approach can guarantee to select the most relevant variables when the sample size goes to infinity. Finally, based on the penalty term, the flexibility of the nonlinear approx-

imator is bounded via penalizing the derivative, which gives a more reasonable estimation for the marginal change analysis, which is widely discussed in the economic literature.

2.3 Nonlinear Regression via a Soft Decision Tree

2.3.1 From Hard Decision Tree to Soft Decision Tree

The decision tree is a classical machine learning algorithm. The classification and regression tree (CART), a revised decision tree method, has been widely used in data mining, machine learning, and other related research fields[1]. Figure 2.2 illustrates the structure of a decision tree with one layer.

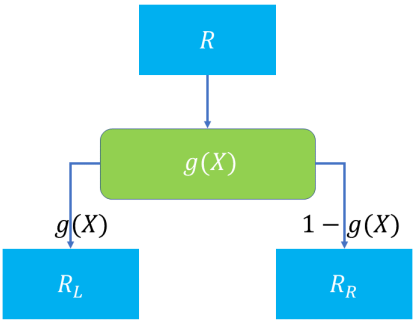

Figure 2.2: The Structure of a Decision Tree with 1 Layer

The output of the decision tree in Figure 2.2 is

$$R = R_L g(x) + R_R(1 - g(x)). \tag{2.15}$$

[1]In this chapter, the name of the decision tree is equivalent to CART.

In Equation (2.15), $g(x)$ is a threshold function and it is usually an indicator function:

$$R = \begin{cases} R_L & if\ g_b(x) = 1 \\ R_R & if\ g_b(x) = 0. \end{cases} \tag{2.16}$$

In terms of $g(x)$, the decision tree is "hard". A decision tree can also be "soft" when $g(x)$ has a continuous output as $0 < g(x) < 1$. In the machine learning literature, the logistic function $g(x) = \frac{1}{1+\exp(-(w^T x+w_0))}$ is usually selected to return a continuous output from 0 to 1. Thus, the soft decision tree (SDT) returns the average output combining R_L and R_R, as in Equation (2.18).

$$R = R_L g(x) + R_R(1 - g(x)). \tag{2.17}$$

Jordan and Jacob (1994) discussed the soft decision tree as a hierarchical mixture of experts (HME) method for the first time. For the HME, we can designate the structure of the tree and optimize the parameters via the expectation-maximization (EM) algorithm.

Figure 3 shows a two-level HME. The HME covers all the intuitions behind the soft decision tree. First, this method considers regression as a recursive binary problem. To any given space of variables X, each node splits the area into two subspaces. Second, it introduces a soft gating function instead of a hard step function in the hard decision tree. Finally, the soft decision tree designates the tree's structure and then optimizes the tree's parameters. Since all the parameters are adjusted simultaneously, a soft decision tree often outperforms a hard decision tree in many cases.

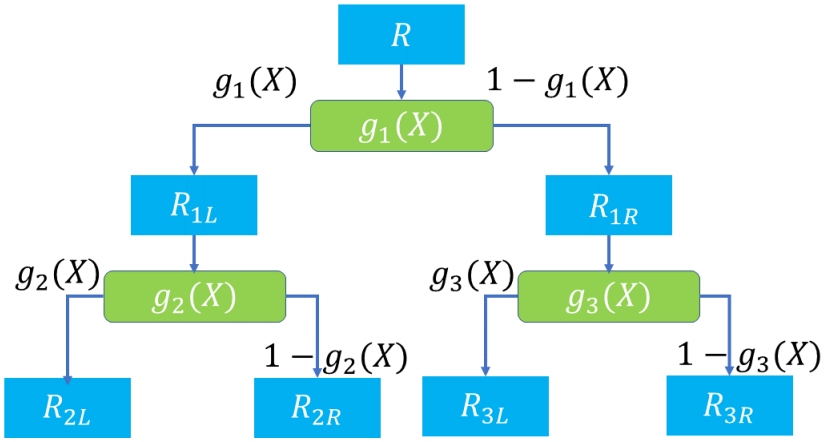

Figure 2.3: A Graph of Hierarchical Mixture of Experts with Two Levels

In this chapter, we implement the Quasi-Most likelihood Estimation (QMLE) to optimize the soft decision tree. According to the definition of the soft decision tree, Equation (2.18) gives the probability of y given x and all the parameters w

$$P(y|x,w) = \sum_{s=1}^{S} \prod_{p \to s} g_p(x; w_p) P_s(y|x). \tag{2.18}$$

In Equation (2.18), S is the number of leaf nodes, $p \to s$ represents all the internal nodes ($p = 1, 2, ...$) on the path from the root node to the sth leaf node. Specifically, we mainly focus on the conditional mean of distribution in Equation (2.18) and then define the conditional mean as the output of a soft decision tree in Equation (2.19)

$$\mu_{tree}(x) = E(y|x,w) = \sum_{s=1}^{S} \prod_{p \to s} g_p(x; w_p) E_s(y|x) \tag{2.19}$$

Equation (2.20) provides a compact form of Equation (2.19) where $\alpha_s(x_i) = \prod_{p \to s} g_p(x; w_p)$

$$\mu_{tree}(x) = \sum_{s=1}^{S} \alpha_s(x_i)\mu_s \tag{2.20}$$

Simply, assume that u_i follows a Gaussian distribution $N(0, \sigma^2)$ in the DGP $y_i = f(x_i) + u_i$. The final likelihood function in Equation is:

$$L(y|x; w, \mu, \sigma) = -\sum_{i=1}^{N} \frac{1}{2} log\sigma^2 - \sum_{i=1}^{N}\sum_{s=1}^{S} \alpha_s(x_i)\frac{(y_i - \mu_s)^2}{\sigma^2} \tag{2.21}$$

Since the optimal solution based on the likelihood function in Equation (2.21) does not have a closed-form, the numerical methods are often considered. According to Jordan and Jacob (1994), they optimized the soft decision tree via expectation-maximization (EM) on the original likelihood function without normality assumption. In this chapter , we optimize the parameters in Equation (2.21) via gradient descend. Additionally, since our main task is variable selection based on penalty, we introduce a penalized log-likelihood method from Fan and Li (2001) and generalize it into our L_1 regularized soft decision tree. We summarize this method in Proposition 5.

Proposition 5 *Given the sample* $\{y_i, x_i\}, i = 1, ..., N$ *with data generating process* $y_i = E(y_i|x_i) + u_i = f(x_i) + u_i$ *where* $u_i \sim N(0, \sigma^2)$*, define the* L_1 *penalized MLE of soft decision tree as*

$$L(y|x; w, \mu, \sigma) - \lambda\Omega = -\sum_{i=1}^{N} \frac{1}{2} log\sigma^2 - \sum_{i=1}^{N}\sum_{s=1}^{S} \alpha_s(x_i)\frac{(y_i - \mu_s)^2}{\sigma^2} - \lambda\Omega(\mu_{tree}(x)), \tag{2.22}$$

where $\Omega(\theta) = \left|\left|\frac{\partial \mu_{tree}(x)}{\partial x_p}\right|\right|_2$ *and*

$$\mu_{tree}(x) = \sum_{s=1}^{S} \alpha_s(x_i) E(y_i|x_i; \mu_s, \sigma^2) = \sum_{s=1}^{S} \alpha_s(x_i)\mu_s.$$

Based on Proposition 5, we can optimize the penalized likelihood function of the soft decision tree easily and it also supports the discussion about the asymptotic properties.

2.3.2 Soft Decision Tree as a Nonparametric Kernel Regression

Firstly, we show that the soft decision tree method is equivalent to nonparametric kernel regression. Without loss of generalization, we set the leaf node $f_{leaf}(x)$ as a constant μ for each leaf node. The formula for a soft decision tree is

$$\begin{aligned}
\mu_{tree}(x) &= \sum_{s=1}^{S} \left(\prod_{p \to s} g(x; w_p) \right) \times \mu_s \\
&= \sum_{s=1}^{S} \left(\prod_{p \to s} \frac{1}{1 + exp(-(w_p x + w_{p0}))} \right) \times \mu_s \\
&= \sum_{s=1}^{S} \left(\prod_{p \to s} \frac{1}{1 + exp(-(w_p x + w_{p0}))} \right) \times \mu_s.
\end{aligned} \tag{2.23}$$

Proposition 6 *For a soft decision tree* $\mu_{tree}(x) = \sum_{s=1}^{S} \left(\prod_{p \to s} g(x; w_p) \right) \times \mu_s$. *There exist a equivalent nonparameteric regression with S kernels in Equation (2.24)*

$$\begin{aligned}
\mu_{tree}(x) &= \sum_{s=1}^{S} \left(\prod_{p \to s} g(x; w_p) \right) \times \mu_s \\
&= \frac{\sum_{s=1}^{S} K(x; \beta_s, c_s)\mu_s}{\sum_{s=1}^{S} K(x; \beta_s, c_s)}.
\end{aligned} \tag{2.24}$$

We illustrate the Proposition 6 in an univariate regression case. Given $\mu_{tree}(x) = \sum_{s=1}^{S} \left(\prod_{p \to s} g(x; w_p) \right) \times \mu_s$, rewrite the gate function $g(x; w_p, w_{p0})$ into Equation (2.25):

$$
\begin{aligned}
g(x; w_p, w_{p0}) &= \frac{1}{1 + exp(-(w_p x + w_{p0}))} \\
&= \frac{exp(1/2(w_p x + w_{p0}))}{exp(-1/2(w_p x + w_{p0})) + exp(1/2(w_p x + w_{p0}))}.
\end{aligned} \tag{2.25}
$$

Next, let us consider another possible method for deriving the gate function. Assume we have a Gaussian kernel $exp(-\beta(x - c_0)^2)$; a related gate function is

$$
g(x; \beta, c) = \frac{exp(-\beta(x - c_0)^2)}{exp(-\beta(x - c_0)^2) + exp(-\beta(x - c_1)^2)}. \tag{2.26}
$$

In Equation 2.26, β, c_0, c_1 are parameters for the Gaussian kernel function. Intuitively, the gate function in (12) contains a pair of symmetric functions. Rewriting this function, we have

$$
\begin{aligned}
g(x; \beta, c) &= \frac{exp(-\beta(x - c_0)^2)}{exp(-\beta(x - c_0)^2) + exp(-\beta(x - c_1)^2)} \\
&= \frac{exp(2\beta(c_1/2 + c_0/2)x + \beta c_1^2/2 - \beta c_0^2/2)}{exp(-\beta(x - c_0)^2) + exp(-\beta(x - c_1)^2)}
\end{aligned}
$$

Let $2\beta(c_1/2 + c_0/2) = w_p$ and $\beta c_1^2/2 - \beta c_0^2/2 = w_{p0}$, we have

$$
\begin{aligned}
g(x; \beta, c) &= \frac{exp(2\beta(c_1/2 + c_0/2)x + \beta c_1^2/2 - \beta c_0^2/2)}{exp(-\beta(x - c_0)^2) + exp(-\beta(x - c_1)^2)} \\
&= g(x; w_p, w_{p0})
\end{aligned} \tag{2.27}
$$

Since a logistic function can be represented by two kernel functions with the same window width β and symmetric centers c_0, c_1, a soft decision tree can be represented by a nonparametric kernel regression function with finite kernels in Equation (2.24).

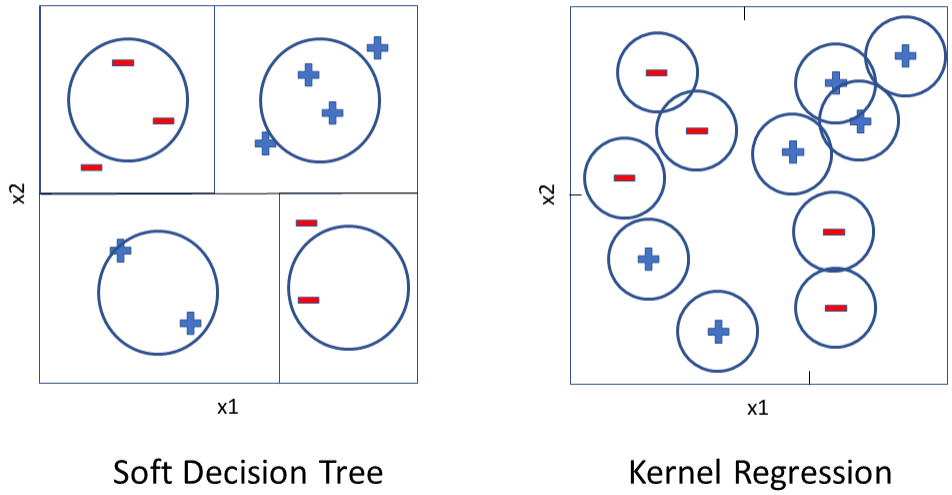

Figure 2.4: The Connection between Soft Decision Tree and Kernel Regression

Figure (2.24) illustrates the connection between the soft decision tree and kernel regression. Each split can generate two kernels, and the total number of kernels depends on the number of leaf nodes. The center in each kernel μ_s is estimated. In the kernel regression, the sample size determines the number of kernels, and each kernel is centered on a sample. Thus, the two methods can share the similar asymptotic properties. However, the soft decision tree can be more adaptive to the sample than the kernel regression since it determines the position of the kernel on the distribution of samples.

2.3.3 Asymptotic Properties of Soft Decision Trees

Based the Proposition 6, we can analyze the unbiasedness of the soft decision tree based on mixture models with finite kernels. We present Theorem 7 to show the asymptotic unbiasedness of the soft decision tree:

Theorem 7 *Given the following assumptions satisfied*

- *The unknown function $f_0(x)$ is Lipschitz continuous:*

$$d(f_0(x) - f_0(x')) \leq Dd(x - x'), \tag{2.28}$$

  where D is the Lipschitz constant, and d is a measure of distance.

- *x follows a distribution with a finite support $d(x - x') \leq d_{max} < \infty, \forall x, x'$.*

If the parameters in $K(x; \beta_s, c_s)$ and the parameter μ_s are estimated in two seperate sub samples with N_1 and N_2 observations, to a soft decision tree $\hat{\mu}_{tree}(x) = \frac{\sum_{s=1}^{S} K(x;\hat{\beta}_s,\hat{c}_s)\hat{\mu}_s}{\sum_{s=1}^{S} K(x;\hat{\beta}_s,\hat{c}_s)}$ where $K(x; \beta_s, c_s) = e^{(-lg_2^S (x-c_s)^T \Sigma_\beta (x-c_s)))}$,the bias is bounded by

$$|f_0(x) - E(\hat{f}(x))| \leq \frac{C^*}{\sqrt{lg_2^S \beta_{max}} S^{(1/2p-1/2)}} \asymp \frac{1}{\sqrt{lg_2^S \beta_{max}} S^{(1/2p-1/2)}} \to 0,$$

where $C^* = \frac{D}{\sqrt{2\pi}}$ and D is the Lipchitz constant. p is the dimension of the input x, β_{max} is the max eigenvalue of the weight matrix Σ_β of input x.

Specifically, let $\beta_{max} = \frac{S}{lg_2^S}$, we have

$$|f_0(x) - E(\hat{f}(x))| \leq \frac{C^*}{\sqrt{lg_2^S \frac{S}{lg_2^S}} S^{(1/2p-1/2)}} \asymp S^{-1/2p} \to 0, \tag{2.29}$$

We present proof in Appendix A. Based on Theorem 7, if $S \to \infty$ and $\frac{S}{N} \to 0$ are satisfied when $N \to \infty$, the bias of the soft decision tree is bounded.

Based on the results in Theorem 7, the soft decision tree can approximate any Lipschitz continuous functions with finite D. Thus, we can conclude that the likelihood function of the soft decision tree can approximate the actual likelihood function based on the right data generating process such that $E(L(\hat{\theta})) \to L(\theta)$. Hence the asymptotic results based on the actual likelihood function can be generalized to the soft decision tree.

The next questions is: can the QMLE-based soft decision tree estimator $\hat{\mu}_{tree}(x)$ be consistent with the true $f(x)$? To solve it, we need to construct the asymptotic theory of $\hat{\mu}_{tree}(x)$. Consider the soft decision tree model in Equation (2.30)

$$\begin{aligned} E(y|x) &= \frac{\sum_{s=1}^{S} K(x;\beta_s,c_s)\mu_s}{\sum_{s=1}^{S} K(x;\beta_s,c_s)} \\ &= \sum_{s=1}^{S} \alpha_s(x)\mu_s \end{aligned} \tag{2.30}$$

The likelihood function is as follows:

$$L(y,x;\theta) = \sum_{i=1}^{N} Logp\left(y_i, \sum_{s=1}^{S} \alpha_s(x_i)\mu_s\right) = \sum_{i=1}^{N} Logp\left(y_i, \mu_{tree}(x_i)\right) \tag{2.31}$$

According to Kiefer and Wolfowitz (1956), the consistency of the nonparametric MLE requires four conditions. Because of $E(L(\hat{\theta})) \to L(\theta)$, the likelihood function of soft decision tree should share the same conditions of the true likelihood function. However, the soft decision tree suffers from the label switching problem. Given an object function $f(x)$, there exists more than one set of parameters θ converging to the same function.

Nevertheless, since the object is $f(x)$ instead of the parameters, we can revise the parameters to be locally identifiable, which means that at an open set of parameters around one optimal local parameter, the parameters can be identified. Given the satisfying condition for MLE, we can have Lemma 8.

Lemma 8 *Assume the true likelihood function is $L(y, x; \theta)$. We implement the QMLE on estimating θ and asymptotic normality on the parameters*

$$\sqrt{N}(\hat{\theta} - \theta^*) \sim N(0, \Sigma_{\theta^*}),$$

where $\theta^* = E(\hat{\theta})$, $\Sigma_{\theta^*} = I^{-1}(\theta^*)E(S(\theta^*)S(\theta^*)')I^{-1}(\theta^*)$. $S(\theta^*)$ is the score function and $I(\theta)$ is the Fisher Information matrix.

In Theorem 7, we discuss the debiased procedure for the soft decision tree via sample-splitting. To optimize the debiased soft decision tree, we need to consider a sample-splitting MLE. Firstly, we split the N samples into two parts, $N_1 = wN$ and $N_2 = (1-w)N$, where $w \in (0, 1)$ is a weight for sample-splitting. Then, implement the MLE procedure on optimizing the soft decision tree's parameters with the first sample N_1. Next, by fixing all the parameters within the root node and intermediate nodes, we estimate all the parameters on the leaf nodes μ_s via MLE on the second sample N_2. Finally, combine the two parts of parameters to construct the debiased soft decision tree. Theorem 9 illustrates that the QMLE-based debiased soft decision tree estimator should be asymptotic normal distributed.

Theorem 9 *Given the soft decision tree $\mu_{tree}(x; \theta)$. If the θ is estimated via QMLE and has the asymptotic normal distribution $\sqrt{n}(\hat{\theta} - \theta^*) \sim N(0, \Sigma_{\theta^*})$, the soft decision tree estimator is asymptotically distributed as*

$$(\hat{\mu}_{debiased\ tree}(x;\theta) - E(\hat{\mu}_{debiased\ tree}(x;\theta))) \to N(0, \sigma^2_{debiased\ tree}(x)), \tag{2.32}$$

where $\sigma^2_{debiased\ tree}(x) = \frac{1}{N} J(x)' \Sigma_{debiased,\theta^*} J(x) \to O(\frac{pS^2}{N})$, *$S$ is the number of leaf of the soft decision tree and p is the dimensioa of x.*

Finally, we combine the Theorem 7 and tTheorem 9 to construct the asymptotic normality of the debiased soft decision tree $\hat{\mu}_{debiased,tree}(x)$. Specifically, we need that the convergence rate of bias is faster than the variance such that the $\hat{\mu}_{debiased,tree}(x)$ converges to normal distribution centered at $f(x)$. To sum up, we propose the Theorem 10.

Theorem 10 *Given the number of leaf nodes $S = N^\alpha$ and $\frac{1}{(1/p_0+1)} < \alpha < 1$, the asymptotic normality of $\hat{\mu}_{debiased\ tree}(x)$ is*

$$\frac{\hat{\mu}_{debiased\ tree}(x) - f(x)}{\sigma_{debiased\ tree}(x)} \to N(0,1) \tag{2.33}$$

Theorem 9 and 10 are proved in Appendix A.

Additionally, we provide the standard error estimator:

$$\hat{\sigma}_{debiased\ tree}(x) = \sqrt{\frac{1}{N} \hat{J}(x)' \widehat{\Sigma}_{debiased,\theta} \hat{J}(x)},$$

where $\widehat{\Sigma}_{debiased,\theta} = I(\theta)^{-1} \hat{E}(S(\theta)S(\theta)')I(\theta)^{-1}$, $S(\theta)$ is the score function and $I(\theta)$ is the information matrix.

2.4 Variable Selection via Soft Decision Tree

In the discussions of Section 3, we derived the nonlinear variable selection framework with oracle properties. Section 4 illustrated the consistency and asymptotic normality

of the debased soft decision tree. In Section 5, we introduce the debiased soft decision tree as the nonlinear approximator and embed it into the adaptive L_1-regularization log-likelihood function in Equation (2.34).

$$-L(y,x;\beta,c)-\lambda\Omega(f)=-\sum_{i=1}^{N}\frac{1}{2}log\sigma^2-\sum_{i=1}^{N}\sum_{s=1}^{S}\alpha_s(x_i)\frac{(y_i-\mu_s)^2}{\sigma^2}-\lambda\sum_{p=1}^{P}\hat{w}_p\left|\left|\frac{\partial f(x;\theta)}{\partial x_p}\right|\right|_2 \tag{2.34}$$

In Equation (2.34), $\left|\left|\frac{\partial f(x;\theta)}{\partial x_p}\right|\right|_2=\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial\hat{f}}{\partial x_p}|_{x=x_i}\right)^2}$ and $\hat{w}_p=\frac{1}{||\frac{\partial\hat{f}}{\partial x_p}||_2^{\gamma}}$. Specifically, we set $f(x)=\mu_{tree}(x)$.

According to Proposition 4 and Theorem 10, we present Theorem 11 about consistent nonlinear variable selection via a soft decision tree.

Theorem 11 *Given the likelihood function of a soft decision tree $\mu_{tree}(x)$ with an adaptive L_1 penalty in Equation (2.34), if $\lambda\left(\frac{pS^2}{N}\right)^{(\gamma-1)/2}\to\infty$ and $\lambda\left(\sqrt{\frac{pS^2}{N}}\right)\to 0$,we have the following conclusions of oracle properties:*

- $P\left(\left|\left|\frac{\partial\mu_{tree}(x)}{\partial x_p}\right|\right|_2\neq 0\ |p\in A\right)=1.$
- $(\hat{\mu}_{tree}(x)-f(x))\to_d N(0,\sigma^2_{tree}(x))$, *where* $\sigma^2_{tree}(x)\sim O\left(\frac{pS^2}{N}\right)$.

The proof is showed in Appendix A.

Additionally, we provide the standard error estimator:

$$\hat{\sigma}_{tree}(x)=\sqrt{\frac{1}{N}\hat{J}(x)'\widehat{\Sigma}_{\hat{\theta},L_1}\hat{J}(x)},$$

where

$$\widehat{\Sigma}_{\theta,L_1} = \left(I(\theta) + \lambda \sum_p \frac{1}{\hat{w}_p} \frac{\partial^2 \left|\left| \frac{\partial \hat{\mu}_{tree}(x,\theta)}{\partial x_p} \right|\right|_2}{\partial\theta\partial\theta'} \right)^{-1} \hat{E}(S(\theta)S(\theta)')$$

$$\times \left(I(\theta) + \lambda \sum_p \frac{1}{\hat{w}_p} \frac{\partial^2 \left|\left| \frac{\partial \hat{\mu}_{tree,\theta}(x)}{\partial x_p} \right|\right|_2}{\partial\theta\partial\theta'} \right)^{-1},$$

$S(\theta)$ is the score function and $I(\theta)$ is the information matrix.

2.5 Simulation Study

Based on the discussion from Section 2.2 to 2.4, we illustrate that the soft decision tree can have the asymptotic property and the L_1-regularized soft decision tree can consistently identify the relevant variables. In this section, we implement several simulation experiments to show the performance of the L_1-regularized soft decision tree. In Section 6.1, we explore the asymptotic properties of the soft decision tree by comparing the convergence in different situations. In Section 6.2, we test the nonlinear variable selection outcomes of the performance of the L_1-regularized soft decision tree comparing with other methods. Finally, we examine the results of the heterogeneous treatment effect estimation with or without the variable selection procedures.

2.5.1 Simulation Results of the Asymptotic Properties

We explore the convergence of soft decision tree given a large sample size. Firstly, we test the mean square error and average bias in 4 different data generating situations with different sample sizes. The classic random forest method is introduced as the benchmark for the comparison. We choose four data generating processes as follows

- DGP1:

  $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + u.$

- DGP2:

  $y = \beta_0 + \beta_1 sin(x_1) + \beta_2 x_2 \times exp(\beta_3 x_1 x_2) + \beta_4 x_3^2 + u.$

- DGP3:

  $y = \beta_1 I(x_1 > 1) + \beta_2 I(x_2 > 1) + \beta_3 I(x_3 > 1) + \beta_4 I(x_1 > 0) \times I(x_2 > 0) + u.$

- DGP4:

  $y = \beta_1 I(x_1 > 1) \times x_1 + \beta_2 I(x_2 > 1) \times x_2 + \beta_3 I(x_3 > 1) \times x_3 + \beta_4 I(x_1 > 0) \times I(x_2 > 0) + u.$

where $u \sim N(0, 1)$. DGP 1 and 2 are continuous functions and DGP 3 and 4 are discrete functions. Also, we compare the soft decision tree (SDT) and random forests (RF) when $n = 500$, $n = 1000$ and $n = 5000$. Additionally, we consider two situations. In situation 1, we set the dimension of $x = 3$. In situation 2, we consider $x = 30$ but only the first 3 variables are relevant. Table 2.1 and Table 2.2 give the simulation results.

Based on the simulation results in Table 2.1 and Table 2.2, firstly, we can conclude that the SDT can beat RF in the first and second DGP. RF works better than SDT in the DGP 3 and 4. These conclusions are reasonable since DGP 1 and 2 are smooth functions and easily captured by SDT. At the same time, DGP 3 and 4 are discrete functions satisfying the assumptions of RF. Additionally, to DGP 4, we notice that SDT works better and can finally touch the comparable results as RF when n increases. Therefore, SDT can be more adaptive than RF given more samples.

Table 2.1: MSE and Bias of SDT and RF ($p = 3$)

MSE	**SDT (DGP 1)**	**RF (DGP 1)**	**SDT (DGP 2)**	**RF (DGP 2)**
$n = 500$	1.2474	5.4607	1.5491	4.5156
$n = 1000$	1.1660	4.9113	1.1819	3.5488
$n = 5000$	1.0520	5.0816	1.1735	3.5930
Bias	**SDT (DGP 1)**	**RF (DGP 1)**	**SDT (DGP 2)**	**RF (DGP 2)**
$n = 500$	0.7359	1.6850	0.9188	1.12485
$n = 1000$	0.8089	1.8771	0.8569	1.2747
$n = 5000$	0.7507	1.7793	0.8547	1.3098
MSE	**SDT (DGP 3)**	**RF (DGP 3)**	**SDT (DGP 4)**	**RF (DGP 4)**
$n = 500$	2.3396	1.3140	2.7985	1.6602
$n = 1000$	1.9739	1.2101	1.4776	1.3549
$n = 5000$	1.6010	0.9717	1.1552	1.1351
Bias	**SDT (DGP 3)**	**RF (DGP 3)**	**SDT (DGP 4)**	**RF (DGP 4)**
$n = 500$	0.8531	0.2420	1.0352	0.6464
$n = 1000$	1.1057	0.7762	0.8592	0.6180
$n = 5000$	0.9327	0.1090	1.0071	0.4295

Table 2.2: MSE and Bias of SDT and RF ($p = 30$)

MSE	**SDT (DGP 1)**	**RF (DGP 1)**	**SDT (DGP 2)**	**RF (DGP 2)**
$n = 500$	1.6317	6.2609	5.1188	5.1840
$n = 1000$	1.3280	5.5948	1.5331	3.6131
$n = 5000$	1.0520	5.0816	1.4870	3.5930
Bias	**SDT (DGP 1)**	**RF (DGP 1)**	**SDT (DGP 2)**	**RF (DGP 2)**
$n = 500$	1.0668	1.9884	1.3797	1.6532
$n = 1000$	0.9418	1.7417	0.9723	1.4895
$n = 5000$	0.7507	1.7793	0.9698	1.3098
MSE	**SDT (DGP 3)**	**RF (DGP 3)**	**SDT (DGP 4)**	**RF (DGP 4)**
$n = 500$	8.7303	1.3323	8.5138	2.5808
$n = 1000$	7.5761	1.0084	7.9889	1.9482
$n = 5000$	3.6183	0.9717	7.1745	1.5959
Bias	**SDT (DGP 3)**	**RF (DGP 3)**	**SDT (DGP 4)**	**RF (DGP 4)**
$n = 500$	2.2951	0.9018	2.2220	1.2283
$n = 1000$	2.1093	0.7552	2.2635	1.1120
$n = 5000$	1.5707	0.7564	2.1588	0.9606

Then, to check the pointwise asymptotic distribution of SDT, we implement a simulation study with DGP 1 and DGP3 and check the asymptotic distribution at $x = 1.5$ given $n = 500, 5000$. Figure 2.5 and Figure 2.6 illustrate the simulation results.

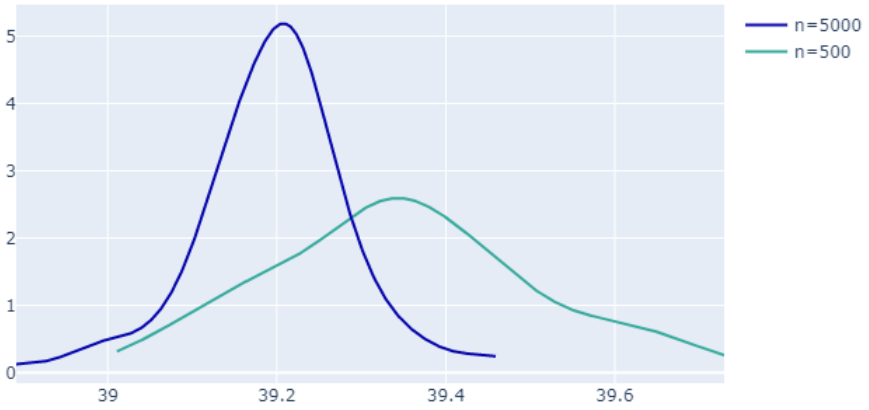

Figure 2.5: The Asymptotic Distribution of Soft Decision Tree (DGP1)

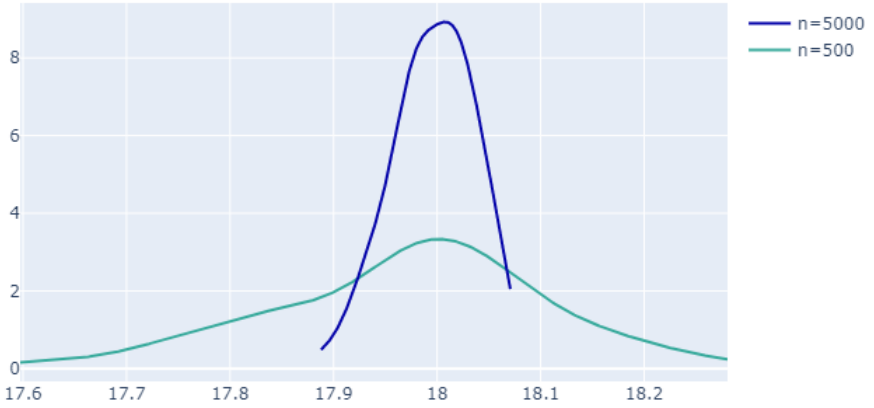

Figure 2.6: The Asymptotic Distribution of Soft Decision Tree (DGP3)

Based on the simulation results in Figure 2.5 and Figure 2.6, we can see that the SDT is pointwise consistent and the asymptotic distributions converge to the actual values

with a decreasing variance. Therefore, we conclude that the debiased SDT has the pointwise asymptotic normality.

2.5.2 Simulation Results of the Nonlinear Variable Selection

In this section, we test the nonlinear variable performance of L_1-regularized SDT. In the first study, we choose lasso and adaptive lasso as our baseline methods. We compare their variable selection results in two settings.

In the first study, we choose two data generating processes as follows:

- DGP1:

  $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + u.$

- DGP2:

  $y = \beta_0 + \beta_1 sin(x_1) + \beta_2 x_2 \times exp(\beta_3 x_1 x_2) + \beta_4 x_3^2 + u.$

DGP 1 is simple, and DGP 2 is relatively complex. Specifically, we choose the input variable with 30 dimensions, and only the first three dimensions are relevant. Then, the sample size increases from 500 to 5000. Finally, we determine the hyperparameters λ and γ via 10-fold cross-validation.

Table 2.3 shows the results of the variable selection based on the soft decision tree, lasso, and adaptive lasso in the situation of DGP1. To the simple DGP 1, SDT works worse than Lasso, and adaptive lasso when n is small. Since DGP 1 is very closed to the sparse linear model, Lasso and Adaptive Lasso work very well in small samples. SDT can attach the same performance of variable selection when n is large.

Table 2.3: Variable Selection for Nonlinear Regression (DGP1)

$\mathbf{P}(\beta_x \neq 0 \mid x \in A)$	**SDT**	**Lasso**	**Adaptive Lasso**
$n = 500$	0.78	0.90	0.91
$n = 1000$	0.84	0.91	0.93
$n = 5000$	0.98	0.94	0.99

Table 2.4: Variable Selection for Nonlinear Regression (DGP2)

$\mathbf{P}(\beta_x \neq 0 \mid x \in A)$	**SDT**	**Lasso**	**Adaptive Lasso**
$n = 500$	0.53	0.68	0.78
$n = 1000$	0.78	0.70	0.79
$n = 5000$	0.97	0.70	0.81

Table 2.4 shows the results of the variable selection based on the soft decision tree, lasso, and adaptive lasso in the situation of DGP 2. To the complicated and highly nonlinear DGP 2, the SDT method works better than other candidates and illustrates the consistency when n increases. However, lasso and adaptive lasso do not work on the highly nonlinear model.

2.5.3 Nonlinear Variable Selection in the Treatment Effect Model

In the second study, we compare their variable selection results in the partially linear regression settings. We design the data generating process based on the generalized partially linear regression:

- $g_h(D,H) = \beta_{1,D} sin(H_1) + \beta_2 H_2 \times H_3 + \beta_{3,D} exp(H_3)$(Heterogeneous function in the outcomes)
- $g_c(D,C) = \alpha_{0,D} + \alpha_{1,D} \times exp(C_1) + \alpha_{2,D} \times C_2$(Confounding function in the outcomes)
- $e(C,V) = \frac{1}{1+exp(-k(C,V))}$(Propensity Score)
- $k(C,V) = 1 + C_1 \times \gamma_1 C_2 + \gamma_2 V_1$
- $y(D,H,C) = g_h(D,H) + g_c(D,C) + u,\ u \sim N(0,1)$(Outcomes)

For each regression, we choose the input dimension as 30, and the sample size is 5000.

Table 2.5 shows the results of the variable selection based on the generalized partially linear regression. We can see that SDT works better than Lasso and adaptive Lasso on both outcome model $y(D,H,C)$ and propensity score model $e(C,V)$. Precisely, Lasso

Table 2.5: Variable Selection for Generalized Partially Linear Regression ($p = 30$)

$\mathbf{P}(\beta_x \neq 0 \| x \in A)$	**SDT**	**Lasso**	**Adaptive Lasso**
H	0.94	0.53	0.62
C	0.95	0.56	0.71

and adaptive Lasso work better on the propensity score model $e(C, V)$, which is simpler than the outcome model $y(D, H, C)$. This result is consistent with the conclusions from Section 2.5.2.

2.5.4 Simulation Results of the Heterogenous Treatment Effect with Variable Selection

In this section, we focus on the heterogeneous treatment effect estimation after nonlinear variable selection. We compare the post-selection results of SDT to the benchmark model, Causal Forests, by Athey and Wager (2018). We choose the following data generating processes

- $g_h(D, H) = \beta_{1,D} sin(H_1) + \beta_2 H_2 \times H_3 + \beta_{3,D} exp(H_3)$(Heterogeneous)
- $e(C, V) = 0.5$(Constant Propensity Score)
- $y(D, H) = g_h(D, H) + u,\ u \sim N(0, 1)$(Outcomes)

The objective is estimating $\tau(H) = y(1, H) - y(0, H)$. The outcomes of the comparison are in Table 6. We show the bias at $x = 1.5$ and calculate the standard deviation in the simulated samples.

Table 2.6: Causal Forest and SDT with/without SDT variable selection (3/30 relevant variables)

	with SDT Varaible Selection	without SDT Variable Selection
Causal Forest	Bias (s.d.)	Bias (s.d.)
$n = 500$	0.401 (0.691)	0.639 (0.926)
$n = 1000$	0.334 (0.556)	0.456 (0.736)
$n = 5000$	0.104 (0.346)	0.145 (0.509)
SDT	Bias (s.d.)	Bias (s.d.)
$n = 500$	0.201 (0.454)	1.249 (1.874)
$n = 1000$	0.147 (0.232)	0.513 (0.762)
$n = 5000$	0.051 (0.101)	0.221 (0.520)

In Table 2.6, we can see that both SDT and Causal Forest can converge to zero in the bias with the decreasing standard deviations. Then, our SDT-based variable selection method can largely increase the performance of convergence to Causal Forests and SDT. Interestingly, we find that the SDT is more sensitive to the variable selection results than the Causal Forest, especially in the small samples. We think the reason is that SDT is a smooth approximator. Causal Forest is based on discrete decision trees, which is relatively robust to the variable selection given the large samples.

2.6 Empirical Analysis: the Effect of an Unemployment Insurance Bonus on Unemployment Duration

In this example, we reanalyze the Pennsylvania Reemployment Bonus experiment conducted by the US Department of Labor in the 1980s to test the incentive effects of alternative compensation schemes for unemployment insurance (UI). This experiment was previously studied by Bilias (2000) and Bilias and Koenker (2002). In these experiments, UI claimants were randomly assigned either to a control group or one of five treatment groups. In the control group, the standard rules of the UI system applied. Individuals in the treatment groups were offered a cash bonus if they found a job within some prespecified period (qualification period), provided that the job was retained for a specified duration. The treatments differed in the level of the bonus, the length of the qualification period, and whether the bonus declined over time in the qualification period; see Bilias and Koenker (2002) for further details.

After cleaning the data and deleting all the irrelevant features, the sample size is 14,068. For the five treatment groups, we find that group 4 and group 6 are often combined into one treatment group since they represent very similar treatments. We perform the same processing so that our result is comparable to the results from other papers. For the covariates, the input dimension is 33, including gender, age, sites, and dates of the treatment and more.

Our main objective is to identify two groups of related attributes for heterogeneous effects and confounding effects. We compare the soft decision tree method with

Table 2.7: Variables in the Unemployment Insurance Bonus Data

Variable	**Meaning**
AGEGT54	Age ¿ 54 Indicator
AGEGT35	Age ¡ 35 Indicator
AT1,AT2,AT3	Attend workshop 1,2,3
BLACK	Black Ethnic Indicator
HISPANIC	Hispanic Ethnic Indicator
SEX	Gender Indicator
JAN, FEB,...,DEC	Month Indicators
BONUS	Bonus Amount
SITE1,SITE2,...,SITE12	Location Indicators

Table 2.8: Variable Selection for the Unemployment Insurance Bonus Data

Methods	**Heterogeneous**	**Confounding**
SDT	JAN, FEB, ... ,DEC	SITE 1, SITE 2, ... ,SITE 12
Lasso	0	SITE 1, SITE 2, ... ,SITE 12
Adaptive Lasso	AUG, MAR	SITE 1, SITE 2, ... ,SITE 12

Lasso and adaptive Lasso. Then, the 10-fold validation method is used to determine the hyperparameters λ and γ. Our results are shown in Table 8.

Based on the selection result, we can see that the program's location is highly correlated with the treatment and outcome. Intuitively, geographic information can distort the treatment effect. Then, the month of the program determines the heterogeneous treatment effect. The other factors are irrelevant since it is a random experiment. Notably, even though the experiment is randomly applied, there are still some confounders that can distort the randomness.

Additionally, the heterogeneous effect can be very significant and indicate that the program's time point is essential to the outcomes. More importantly, compared to the SDT method, lasso and adaptive lasso work well in selecting the confounders. Nevertheless, both linear approaches cannot identify essential factors for the heterogeneous treatment effect. Thus, given the unknown functions for outcomes y and propensity score $e(x)$, it is more reasonable to use our nonlinear variable selection method than linear methods.

2.7 Conclusions

In summary, this chapter proposed a L_1-regularized soft decision tree, a new method for identifying heterogeneous factors and confounders. In Section 2.2, we discuss a consistent variable selection framework in the situation of nonlinear regression. We show that nonlinear variable selection can also feature the oracle properties under certain conditions. In Section 2.3, we prove that the soft decision tree is equivalent to the kernel regression and can approximate the unknown nonlinear function with consistency and asymptotic normality. In Section 2.4, we show that the L_1-regularized soft decision tree can consistently identify the relevant variables. In Section 2.5 and Section 2.6, we find that it works in the simulation data and empirical study. Because of the complexity of heterogeneous treatment effect and confounding effect, we think it is very important to select important variables before estimating and inference on the heterogeneous effect. Hence, L_1-regularized soft decision tree provides a new way to identify the critical variables and it can be applied in a variety of applications in nonlinear regression and causal inference.

Chapter 3

Unsupervised Feature Extraction Clustering for Crisis Prediction

3.1 Introduction

Economic crises inflict substantial damage to the economy. Long-term economic costs, measured in terms of output foregone, are on average 5 percent for balance of payments crises; 10 percent for banking crises, and 15 percent for twin crises (Cerra and Saxena, 2008). Following a financial crisis, a country needs on average eight years to return to its pre-crisis level of income (Reinhard and Rogoff, 2014). Societal costs are also staggering, as average life expectancy declines, primary school enrollment drops, and infant mortality increases (Alexander et al. 2009; van Dijk, 2013).

Macroprudential policy has an important role in crisis prevention and crisis mitigation. The policy effectiveness, however, hinges on whether the macroprudential tools can

target the root causes of economic crises. Crisis prediction models, hence, need to support macroprudential policy. By flagging in advance economic and financial conditions leading to an economic crises, the models can guide policy actions aimed at reducing the crisis likelihood.

This chapter proposes unFEAR, an unsupervised feature extraction clustering method aimed at facilitating economic crisis prediction. The approach in unFEAR is quite different from that in other machine learning-based crisis prediction models. The latter adopt a supervised learning framework: at any time period, the models assign a crisis or no crisis label to a country's observed economic and financial predictor data based on whether the observation was followed or not by a crisis n periods ahead.

The reliance on labeled data gives rise to the biased label problem. Briefly, two countries characterized by similar economic and financial data may receive different labels as only one of them experienced a crisis in the near term. A supervised learner would try to separate both countries even though from a vulnerability perspective both countries belong to the same class. We explain the biased label problem in detail below.

unFEAR avoids the biased label problem using unsupervised learning to find clusters using information in the distribution of the economic and financial data. Rather than working with the raw data unFEAR leverages on the use of autoencoders to reduce the dimensionality of the original data set and generates time-invariant clusters using a novel mode contrastive autoencoder. The crisis and non-crisis observations in a cluster do not correspond to a specific time period, a finding that suggests that a time-invariant economic regime and crisis generating mechanism characterizes each cluster.

Once the clusters are identified it is possible to assess a country's crisis vulnerability at a given point in time. The simplest approach is to assign a country to its closest neighboring cluster. The crisis vulnerability is then calculated as the empirical crisis frequency in the cluster. A second approach, which is normally used in applied machine learning work, is to fit separate supervised learning classifiers to each cluster.

Both approaches for measuring crisis vulnerability could help guide macroprudential policy. Analysts could project the effect of policies on economic and financial fundamentals to determine whether a country may migrate to a safer or riskier cluster. Even if a country's cluster assignment does not change a supervised classifier estimated for the cluster could help to assess whether policies may contribute to reduce or to increase crisis risk.

This chapter is organized as follows: Section 3.2 offers a overview of the literature on early warning and crisis prediction models and provides the needed background to understand the differences between previous machine learning-based crisis prediction models and the unFEAR method proposed here. Section 3.3 describes unFEAR in detail both at the conceptual and technical level. To illustrate unFEAR capability, in Section 3.4, we apply it to a group of advanced economies using a data set of economic and financial variables covering the period 1980 - 2018. Crisis risk and crisis prediction is examined next, and the concluding section examines possible extensions of unFEAR.

3.2 Machine learning-based crisis prediction models

Work on crisis prediction models have largely side-stepped the use of the standard macroeconomic workhorse, the dynamic stochastic general equilibrium model (DSGE). While useful for conducting policy experiments the models do not perform well for forecasting crisis events partly due to the fact that these events are out-of-equilibrium states.[1] Unsurprisingly, most crisis prediction models are formulated as econometric and/or statistical models where economic theory serves to narrow the selection of predictive variables, or features.

The wave of speculative currency attacks on countries with fixed or pegged exchange rates experienced in the 1990s prompted the development of a first generation of crisis prediction models, also known as early warning models. Examples of such models include Frankel and Rose (1996), Kaminsky et al. (1998), and Berg and Patillo (1999) among others. Research on crisis prediction tapered off in the early 2000s as the Great Moderation brought a large decline in macroeconomic volatility (Bernanke, 2004).

Research resumed in the aftermath of the Great Recession in 2008, an event not foreseen by central banks, policy makers, and a majority of market participants. The ensuing studies focus on the reassessment of existing models and on the development of more accurate early warning systems. Example of such work include, among others, Babecky et al. (2012), Chamon and Crowe (2013), Christofides et al. (2016), and Ahuja et al. (2017).

More recently there has been much interest in developing machine learning based models for crisis prediction. The interest sparks from the success of machine learning

[1] See Stiglitz (2017) for a critique, and Christiano et al. (2018) for a rebuttal.

models in prediction tasks in a vast range of knowledge domains outside economics. Recent examples include Alessi et al. (2014), Holpainen and Sarlin (2017), Beutel et al. (2018), Lang et al. (2018), and a number of studies conducted at the International Monetary Fund, with models specialized to predict external crises, financial crises, and fiscal crises.

The machine learning models cited above are supervised learning models. First, the set of explanatory variables (covariates or attributes) include observable economic and financial variables. In some cases, economic theory guides the selection of variables. In other cases, a large number of variables is included with the expectation that the machine learning algorithm will sort out what variables matter the most for crisis prediction. A data point is simply the set of attributes of a country at a given point in time.

Second, since the goal of the models is to predict crisis events, each data point is labeled as a crisis (or non-crisis) point when a crisis occurs (or does not occur) n periods ahead, that is, data points at time t serve to predict crisis and non-crisis events at time $t+n$. In models developed for policy making purposes, n typically ranges from one to two years. If the model flags a future crisis such relatively long prediction horizon leaves time for the authorities to implement preventive or mitigating measures. Finally, crisis definitions and the timing of the crisis are determined outside the model using expert domain knowledge.

3.2.1 Challenges in supervised learning crisis prediction models

Model estimation presents analysts with several challenges. First, despite the widespread perception in the popular press that economic crises recur frequently, crises are still rare events. Compared to non-crisis episodes, the number of non-crisis events largely exceeds that of crisis events, raising the issue of *imbalanced data* (Kotsiantis et al. 2006).

Second, the data sample includes as many countries as possible so that the learner algorithm can observe a non-negligible number of crisis events. Many countries, however, may lack observations for several of the attributes which raises the issue of *missing data*. One solution is to include only attributes with complete observations at the cost of discarding attributes containing useful information. Another solution is to eliminate observations for which the set of attributes is incomplete, which may drastically reduce the number of crisis observations and further worsen the imbalanced data problem. The third option, is to use *data imputation* methods to complete the data set by assigning values to any missing data observation, which raises the question on whether the imputed values truly represent the missing data. A final option is to allow the classifier to learn a set of functions, each one specialized to classify the data points using a subset of covariates.

Third, the set of attributes may include information extraneous for crisis prediction, a likely situation when the number of covariates is large. Extraneous information represents noise and makes model estimation more difficult. Adding to the model estimation complications is that two or more covariates may be strongly dependent. While covariate dependence may not harm the predictive ability of the model, it makes difficult to evaluate a particular covariate importance to predict an economic crisis.

3.2.2 The biased label problem and its unsupervised learning solution

Last but equally important, the biased label problem, may impair the predictive ability of a supervised learning model. Figure 3.1 illustrates this problem. Figure 3.1 is a simple two-dimensional representation of crisis and non-crisis points where there are only

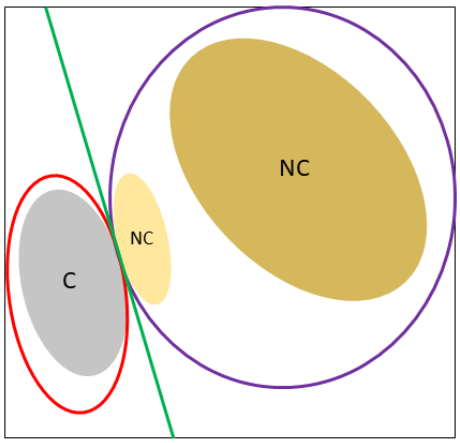

Figure 3.1: The biased label problem.

two features. Each point is represented by its features coordinates. The large circular purple cluster contains all the non-crisis points and the red elliptical cluster all the crisis points. A supervised learning classifier generates a separating hyperplane depicted as the the green line. The hyperplane imposes a hard separation between the crisis and non-crisis points. Ignoring the data points labels would yield a different separating hyperplane, one that separates the large non-crisis cluster from the crisis cluster and the small non-crisis cluster. The latter two clusters belong to a same class different from the non-crisis class. This situation reflects the fact that in this example the information the features convey cannot be used to discriminate properly between crisis and non-crisis labeled points.

The biased label problem is prevalent in policy crisis prediction models due to their long forecasting horizon. Two data points sharing the same characteristics, i.e. two different countries with the same economic fundamentals possibly measured at different times, may suffered a different fate two years ahead as only one of them would experience a crisis. There might be several explanations on why the countries' fates were so different, none of which

the features are able to capture. For instance, the lucky country may experience a favorable commodity price movement that strengthens its fundamentals. Or economic policies may have been put in place that prevented the crisis.[2]

This is a situation unsupervised learning could handle adequately. An unsupervised learner, rather than forcing a hard separation between crisis and non-crisis points, assigns countries with similar features or economic fundamentals to different clusters. The crisis risk in the cluster corresponds roughly to the frequency of its crisis observations. Hence, it becomes possible to rank clusters in terms of crisis risk and to assign a crisis frequency to a data point even if the point label was not used to identify the cluster structure. One natural interpretation is that clusters represent different economic regimes, each with a different propensity to generate an economic crisis.

Mathematically a supervised classifier tries to estimate the conditional probability distribution of the crisis/non-crisis label, y, conditional on the features, x, i.e. $P(y|x)$. In contrast, the unsupervised classifier attempts to learn the unconditional probability distribution of the features, $P(x)$. From a computational and estimation perspective, an added advantage of the unsupervised classifier is that its estimation requires fewer data points (or examples) than a supervised learner to produce a reliable cluster structure. Also, there is no need to separate the available data into training, validation, and test sets.

It is also worth noting that adopting an unsupervised approach is consistent with economic intuition since we expect, given current knowledge of crisis dynamics and the partial information of economic and financial data, that in a group of countries with similar

[2]The biased label problem in crisis prediction is somewhat similar to the problem of label bias and fairness: data points are falsely attributed to a certain class even if the features may not justify it. See for instance, Jiang and Nachum (2019).

economic fundamentals some may experience a crisis and others may not. Hence, on a first pass, it makes sense to identify the clusters first using unsupervised learning and then to fit a cluster-specific supervised classifiers.

3.3 unFEAR: Unsupervised Feature Extraction Clustering for Crisis Prediction

Conceptually, unFEAR is a simple method:

- First, it performs feature engineering (also known as feature learning or representation learning) to extract relevant information from the raw data set useful for clustering analysis.
- Second, once an appropriate representation is built, unFEAR identifies separate clusters and the corresponding data point assignments.

At first it may seem odd to perform feature engineering on the raw data since this is not yet usual practice in econometrics. Figures 3.2 and 3.3 illustrate why this step is necessary to obtain a suitable data representation. Namely:

- *Raw data attributes do not generate separable clusters.* The left panel in Figure 3.2 shows clusters obtained with K-means using annual data on data points comprising 75 attributes collected for 34 countries during the period 1970-2018. The raw data was used without any pre-processing prior to the application of the clustering algorithm, and the number of clusters was selected with a scree plot. The two-dimensional

representation, generated using the t-SNE algorithm (van der Maaten and Hinton, 2008), shows that raw data attributes do not have enough discriminatory power.

- *Raw data capture different time periods rather than different economic fundamentals, i.e. the data exhibits time trends.* The right panel in Figure 3.2 simply places time labels, corresponding to different periods, over the data points without any attempt to assign them to clusters. The time labeled data points overlap substantially with the raw data-based clusters in the left panel of the figure (i.e. cluster 8 corresponds to the most recent data points). Absent feature engineering, an unsupervised learner may only pick data points in different time periods.

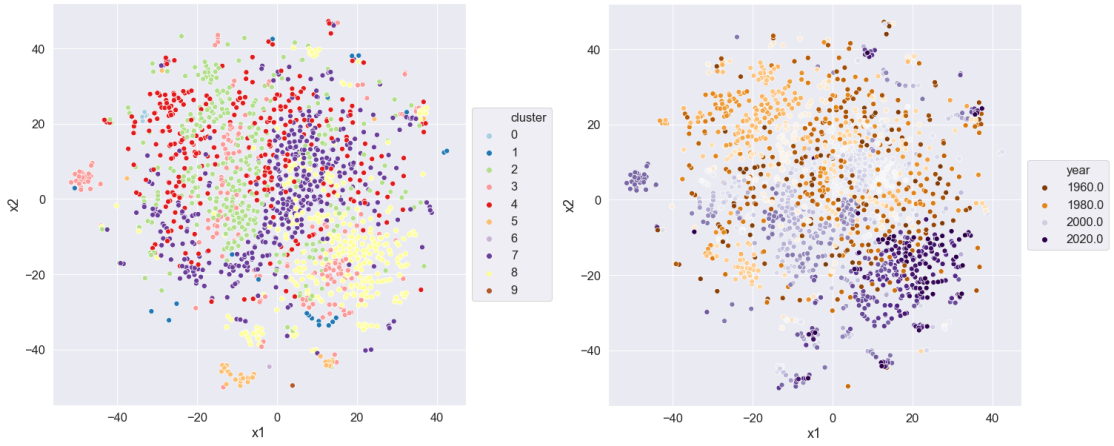

Figure 3.2: Raw data clusters: non-separability and time clustering

- *Raw data tends to group data points corresponding to the same country.* The left panel in Figure 3.3 shows the data points colored by countries. While the cluster structure remains badly defined, near neighbor points tend to correspond to the same country. Reliance on the raw attributes may yield clusters with a majority of data

points corresponding to the same country.

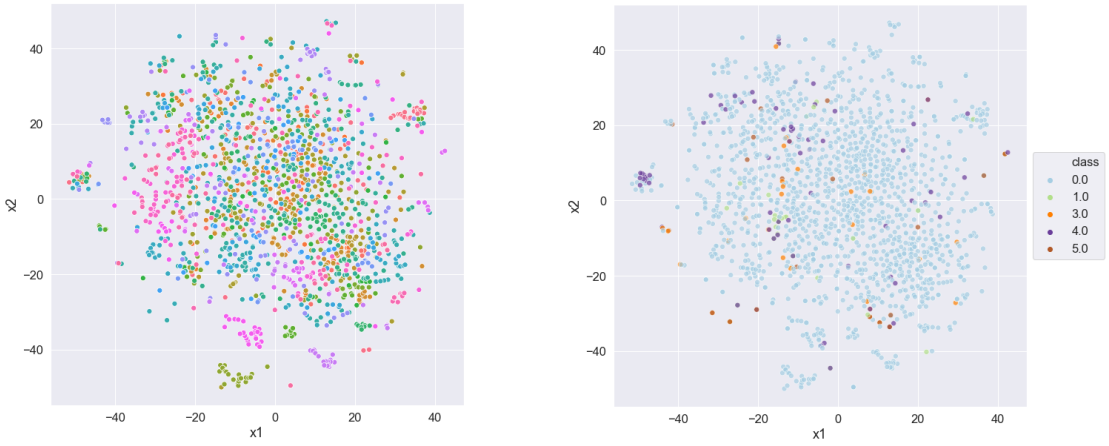

Figure 3.3: Raw data clusters: country bias and imbalanced data

- *Raw data is imbalanced, i.e. few crisis observations.* The right panel in Figure 3.3 shows the non-crisis points (in light blue) and the crisis points, which comprise 90 percent and 10 percent of the observations. An algorithm may find clusters biased to reflect the distribution of the non-crisis observations.

- *Missing data is a big issue.* It is not uncommon to find several missing and incomplete data points when putting together a common data set of economic and financial data for a large panel of countries. In our data set, for all observed data points only five variables do not have any missing value and for about two thirds of the variables (58), missing values could be found as in as much as fifty percent of the data points.

To perform feature engineering step we use Auto Encoders, which are commonly used in machine learning and deep learning, with a suitable loss function designed with the purpose to to address the first three issues described above, i.e. lack of separability in the

data points, time clustering, and country clustering. After some experimentation, we fall back on the Synthetic Minority Over-sampling Technique (SMOTE) to address the data imbalance issue (Chawla et al. 2002). The next section describes in detail the technical details of the method.

The main tool in the feature engineering task is the Autoencoder. To understand the logic behind it we first present its foundation, the multilayered network. Next we examine how Autoencoders work using as an analogy principal components analysis. Once the intuition is established it becomes straightforward to understand why autoencoders serve to input missing data, to remove time clusters, and to identify the different data point clusters.

3.3.1 Multilayer neural networks

The multi-layered neural network is the basic workhorse of deep learning methods (Goodfellow, Bengio, and Courville, 2016). Figure 3.4 illustrates two neural networks.

The neural network in the left panel consists of three layers: the input layer, the hidden layer, and the output layer. From the outside, only the input, x, and the final output, y, are observed. The input layer collects the input, x, and feeds it to the hidden layer, whose units transform the input into an intermediate output, h, via a function g_1, i.e. $h = g_1(x)$. The intermediate output is then fed to the output layer, which processes it and produces the final output, y, using the function g_2, i.e. $y = g_2(h)$. Allowing the functions g_1 and g_2 to be non-linear allows the neural network to capture nonlinearities present in the data. The number of units in the hidden layer is a hyperparameter which is tuned

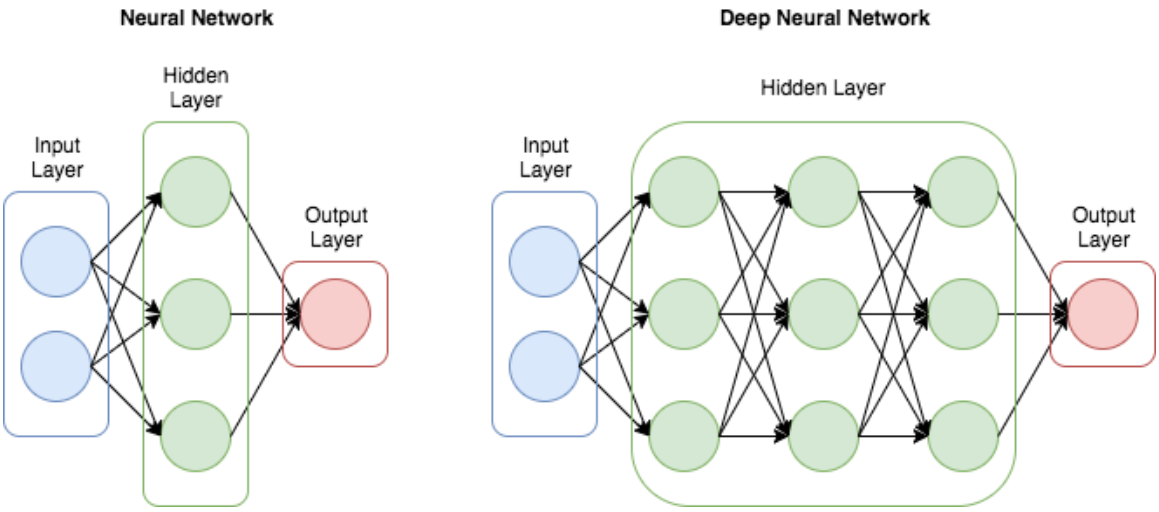

Figure 3.4: Two neural networks

(calibrated) using the data points.

It is possible to use multilayer networks, which contain several hidden layers instead of a single hidden layer. For instance, the right panel shows a three layer network. In this case, the output of the first hidden layer is the input of the second hidden layer. In turn, its output serves as an input to the third layer, whose output is then fed to the output layer. More generally, the transforming functions of the hidden layers can be specified recursively:

$$h_l = g_l(h_{l-1}),$$

where l is the l-th hidden layer, and g_l is a nonlinear transformation. Including several layers enables a deep learning network to captures the dependence between the input data and the output data in complex cases (Pascanu et al., 2014; Arora et al., 2018). Autoencoders exploit this property to find data patterns, such as the joint probability distribution of the attributes, which then serves to input missing data; and the time clustering information in the data, which then allows removing time effects as explained later.

3.3.2 Understanding autoencoders

To understand autoencoders it is useful to examine their conceptual similarity with principal components analysis (PCA), a standard method for dimensionality reduction widely applied in statistics as well as in finance and economics. PCA transforms the data input from its original space into an orthogonal space via a projection matrix, or in other words, it decomposes the data attributes along vectors (directions) orthogonal to each other (upper panel, Figure 3.5). It is possible to go from the orthogonal space to the original space, or reconstruct the inputs, if the projection matrix is known. To reduce the dimensionality of the original data input, we only keep a few components of the orthogonal space provided the retained components explain a substantial amount of the data total variance.

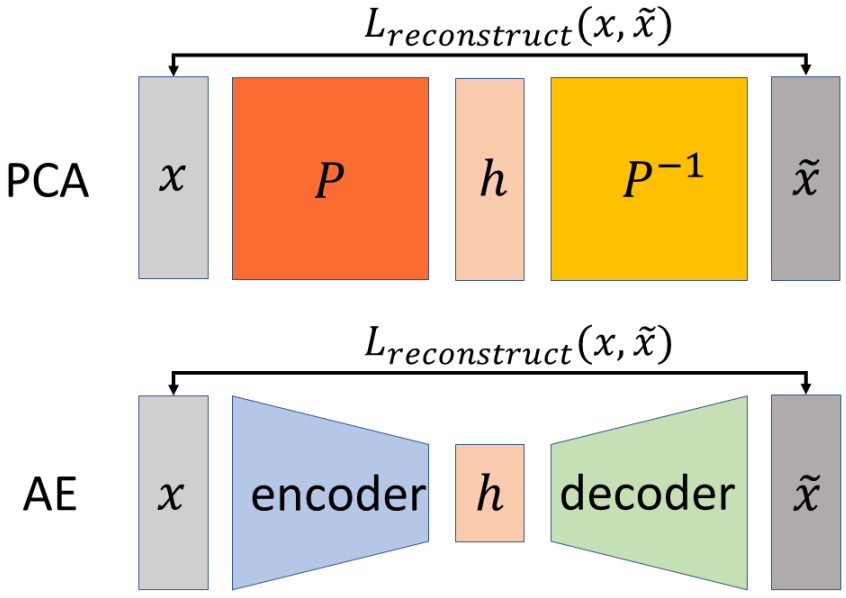

Figure 3.5: The analogy between principal components analysis and the autoencoder

Functionally, the matrix P is an encoder, i.e, it *encodes* the data using a few components and yields a low-dimensional representation, h. or the code in the transformed space. The inverse of the matrix P, P^{-1}, is a decoder, i.e. it *decodes* the coded, h, and returns an approximation of x, $\tilde{x}$. Ideally, what we want is to encode the data to obtain a good but simpler data representation while at the same time retain enough information such that the data approximation in the original space is adequate. The process of encoding and decoding the data yields the matrix P, which captures the relevant characteristics of the data input.

The autoencoder generalizes the PCA coding and decoding function beyond linear transformations and it comprises an encoder and a decoder, which are typically specified as multilayer neural networks (bottom panel, Figure 5). The encoder learns a function, $g_{encoder}$ by projecting the original input x onto h, with h contained in a lower dimensional space:

$$h = g_{encoder}(x).$$

We require the encoder to reduce the dimensionality of the data input in order to simplify any subsequent classification or learning process applied to the encoded data. In turn, the decoder learns the $g_{decoder}$ function that enables the autoencoder to reconstruct an approximation $\tilde{x}$ of the the original input x from the encoded representation h:

$$\tilde{x} = g_{decoder}(h).$$

To find the best data representation, it is necessary to specify a loss function associated with the reconstruction error, $L_{reconstruct}(x, \tilde{x})$:

$$L_{reconstruct}(x, \tilde{x}) = L(x, g_{decoder}(h)) = L(x, g_{decoder}(g_{encoder}(x))).$$

Optimization of the loss function yields the optimal encoder function $g_{encoder}(x)$ for the nonlinear transformation. By using multilayer networks, the autoencoder easily performs PCA when nonlinearities are present in the data. Stacking more layers and introducing noise in the encoder and decoder functions enable autoencoders to deliver complex but robust data transformations (Vincent et al., 2008 and 2010). We exploit the autoencoder properties to perform missing data imputation.

3.3.3 Missing data imputation with autoencoders

Missing data imputation is often performed using one of the three following methods: replacing the missing value by a constant value, typically the median (median imputation) or the mean (mean imputation); resampling from the empirical distribution of the non-missing values; or exploiting the dependence among variables by regressing observed values on other variables and replacing the missing data by the predictions of the regression equations, such as done in the multivariate imputation by chained equations (MICE) method (Raghunathan et al., 2001, Van Buuren, 2007).

unFEAR introduces an autoencoder-based missing data imputation strategy using the Mean Squared Error loss function (MSE) to measure the reconstruction error in Figure 3.6.

The use of the autoencoder builds on the assumption that all the attributes (variables) in a high dimensional data exhibit dependence. The dependence assumption typically holds in reality especially for economic data. This fact enables the autoencoder strategy to recover missing attribute values using the observed values of other attributes. It is worth

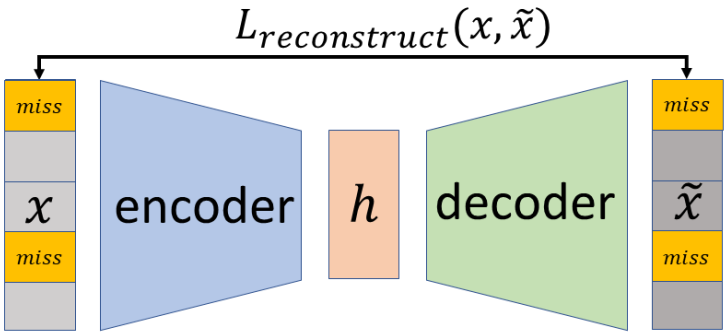

Figure 3.6: Missing data imputation using an autoencoder

noting that the MICE also exploits the dependence assumption to justify regressing an attribute on other attributes. Contrary to MICE, the autoenconder does not need to specify whether an attribute with missing data can or cannot be used as a regressor. The autoencoder automatically assigns higher weights to attributes with more observations. Hence, the autoencoder method is equivalent to a data driven MICE method, combining resampling and dependence exploitation.[3]

The autoencoder missing data imputation method then reduces to:

- First, draw samples randomly from the non-missing data points.
- Second, train an autoencoder on the randomly drawn data sample.
- Third, use the estimated autoencoder to fill the missing data values.[4]

[3] A related method is the Markov Chain Monte Carlo variational autoencoder-based of Rezende et al. (2015).

[4] The activation function of this autoencoder, as well as the others unFEAR uses, is an exponential linear unit (ELUs) (Clevert et al., 2016). The convergence speed of ELUs outperforms that of rectified linear units (ReLUs) (Klambauer et al., 2017).

3.3.4 Removing time trends using autoencoders

Data exploration shows that the raw data exhibits time clusters or time trends, i.e. data points in certain time periods tend to be close to each other. For low dimensional data sets it is feasible to remove trends using univariate methods but they become burdensome as the number of attributes increases.

unFEAR uses a *Boosted Autoencoder* to remove time trends. The procedure is performed in several rounds. Each round starts with a trained autoencoder which allows us to reconstruct the approximated data input, $\tilde{x}$. The resulting reconstruction error, $r_i = x_i - \tilde{x_i}$ is then fed as an input for training a new autoencoder in the next round (see Figure 3.7).

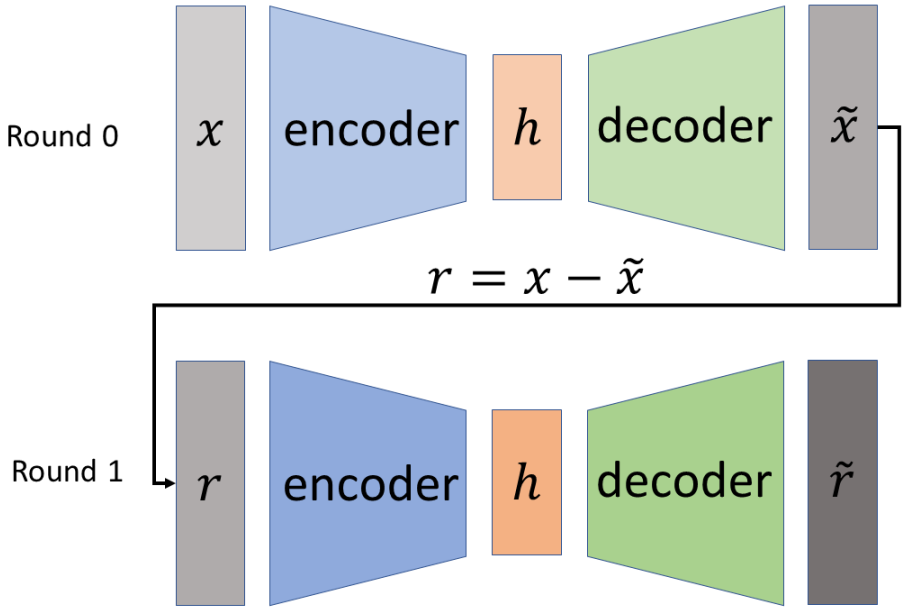

Figure 3.7: A Boosted Autoencoder

Time trends, either linear or non-linear, characterize the variables in the data set. The boosted autoencoder, in a first pass, learns to project the data input x onto a space containing the time trends. Hence, the reconstruction error $r_i = x_i - \tilde{x_i}$ does not exhibit a time trend but still retains other useful information contained in the data input. The unsupervised clustering approach we review next exploits this information.

3.3.5 Unsupervised clustering using the mode constrastive autoncoder

This section introduces and explains the mode contrastive autoencoder, which is the key element in the unFEAR. Raw data, as Figures 3.2 and 3.3 illustrate, are not suitable for clustering analysis. A proper use of an autoencoder could enable us to find a feature representation that facilitates separability. The feature representation should meet two requirements:

- The transformed features should retain a substantial amount of the variation in the original data set to remain informative.
- In the transformed space, the data points concentrate in to several separable clusters.

The first requirement is a common one in the construction of multilayer networks. It forces the autoencoder to learn the best representation of the data that yields a low reconstruction error. The second requirement is necessary since the first one, by itself, does not ensure the autoencoder learns to separate the data into clusters.

Enforcing the first requirement needs the autoencoder to minimize a regularized loss function that balances the reconstruction error and the need to group the data points in

the transformed space into separate clusters. The regularized loss function, L_{AE} is showed in Equation (3.1):

$$L_{AE} = L_{reconstruct}(x, \tilde{x}) + \lambda L_{cluster}(h, \mu), \tag{3.1}$$

where x collects all the data points $(x_1, \ldots, x_N)$, $L_{reconstruct}$ is the standard reconstruction error, λ is a weight tuning parameter, and $L_{cluster}$ is the regularization term forcing the autoencoder to separate the data into clusters using as inputs the output of the encoder, $h = g_{encoder}(x)$, and the centroids of the clusters, $\mu = (\mu_1, \ldots, \mu_K)$.[5]

The regularized loss function induces the autoencoder to learn a encoder $g_{encoder}(x)$ such that an original data point, x_i, when transformed into the encoder output, $h_i = g_{encoder}(x_i)$, can be assigned to a single cluster with centroid μ_c.

The specification of the regularization term is the key element in the unFEAR method. To specify it, we follow an approach similar to the one van der Maaten and Hinton (2008) used to derive their t-Distributed Stochastic Neighbor Embeddings method (t-SNE). We start by specifying the conditional probability that the data point x_i belongs to the c-th cluster, $P(\mu_c|x_i)$ (or equivalently, that the closer neighbor of the data point x_i is the c-th cluster) as:

$$P(\mu_c|x_i) = \frac{(1 + ||\mu_c - g_{encoder}(x_i)||^2)^{-1}}{\sum_{j=1,\ldots,K}(1 + ||\mu_c - g_{encoder}(x_j)||^2)^{-1}}, \tag{3.2}$$

where K, a hyperparameter, is the number of clusters and $||.||^2$ is the Euclidean or L_2 norm. Ideally, we want to assign the transformed data point $h_i = g_{encoder}(x_i)$ to a

[5] A more complex alternative to the use of a regularized loss function, as done here, is to use a denoising autoencoder incorporating the cluster requirement into the reconstruction error. On denoising autoencoders, see Alain and Bengio (2014).

single cluster to ensure the clusters do not overlap and are separable. This implies that the conditional probability distribution in Equation (3.2) should peak at a single value μ_c and take low values, ideally zero, at other cluster centroids. In other words, we want $P(\mu_c|x_i)$ to be a one-peaked probability distribution as close as possible to a delta distribution.

This is equivalent to perform K-means clustering by maximizing the likelihood function:

$$\mathcal{L}(\mu, g; x) = \prod_{c=1}^{K} \prod_{i=1}^{N} P(\mu_c|x_i) P(x_i \in \text{cluster } c) \tag{3.3}$$

or its log-likelihood. The expectation-maximization algorithm of Dempster et al. (1977) yields the following iterative procedure to maximize the log-likelihood:

E-step: given the centroids $\mu = (\mu_1, \ldots, \mu_K)$ and the encoder g, assign to data point x_i the cluster c_i with the maximum log-probability value:

$$c_i = \arg \max_{c_i \in (1,\ldots,K)} \log(P(\mu|x_i; g)),$$

where the conditional probability is given by Equation (3.2) and we have made explicit its dependence on the encoder g.

M-step: given the cluster assignments for each data point, find the new centroid μ_c of cluster c solving the minimization problem below:

$$\mu, g = \arg\min_{\mu,g} \left(-\sum_i label_i \odot \log(P(\mu|x_i; g)) \right), \quad c = 1, \ldots, K.$$

where $label_i = (I(x_i \in \text{cluster } 1), I(x_i \in \text{cluster } 2), ¨, I(x_i \in \text{cluster } K))$ is the one-hot encoded label for x_i, and $I(x_i \in \text{cluster } c)$ is the indicator function.

If follows naturally to set the $L_{cluster}$ equal to $(-\sum_i \mu_c \log(P(\mu_c|x_i)))$ since we want the autoencoder to perform K-means clustering. The autoenconder, hence, is a mode

contrastive autoencoder (MCAE) since it tries to separate the different modes of the clusters.The mode contrastive loss function L_{MCAE} is:

$$L_{MCAE} = L_{reconstruct}(x_i, \tilde{x_i}) + \lambda \left(- \sum_{c=1}^{K} \sum_{i=1}^{N} I(x_i \in \text{cluster } c) \log(P(\mu_c | x_i; g)) \right). \quad (3.4)$$

Figure 3.8 illustrates the role of the loss function terms in the mode contrastive autoencoder.

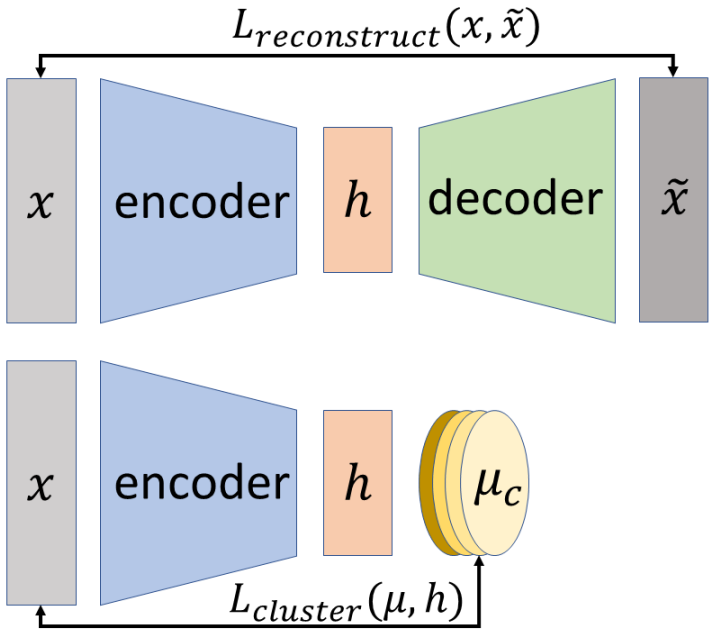

Figure 3.8: The Mode Contrastive Autoencoder

Minimizing the loss function in Equation (3.4) is possible using expectation maximization iteration for a given encoder g:

E-step: this step is similar to the E-step in the log-likelihood maximization. Given the centroids μ_c, $c = 1, \ldots K$, and the encoder g, assign each data point x_i to a cluster c_i with

the maximum log-probability value:

$$c_i = \arg \max_{c_i \in (1,\ldots,K)} \log(P(\mu_c | x_i; g)).$$

M-step: Find the new cluster centroids $\mu_c,\ c = 1, \ldots, K$ that minimize the loss function L_{MCAE}:

$$L_{MCAE} = L_{reconstruct}(x_i, \tilde{x_i}) + \lambda \left(- \sum_c \sum_i I(x_i \in \text{cluster } c) \log(P(\mu_c | x_i; g)) \right).$$

3.4 Application: Identification of Economic Crisis Clusters

This section illustrates the use of unFEAR to identify economic crisis clusters, which in turn, could facilitate the task of crisis prediction. Predicting an economic crisis in advance matters to policy makers and macro-strategists. The goal of the former group is to put in place policy measures to prevent the crisis from realizing, and the goal of the latter is to profit from the event by betting against falling asset prices.

3.4.1 Data

The data in the analysis covers 34 countries during the period 1970 - 2018 (Table 3.1).

The data comprises 1688 data points where each data point is a country-year observation, with 75 attributes. The attributes are constructed using levels, differences, and Hodrick-Prescott trends of the following variables:[6]

[6]A detailed description of the attributes is available upon request from the authors. Most variables

Table 3.1: Country List

Australia	Austria	Belgium	Canada	Cyprus
Czech Republic	Denmark	Estonia	Finland	France
Germany	Greece	Hong Kong S.A.R	Iceland	Ireland
Israel	Italy	Japan	Korea	Luxembourg
Malta	Netherlands	New Zealand	Norway	Portugal
San Marino	Singapore	Slovakia	Slovenia	Spain
Sweden	Switzerland	United Kingdom	United States	

Global variables

- Oil prices
- 10-year U.S. real interest rate
- 3-month U.S. Treasury bill rate, constant maturity
- Trade-weighted dollar currency index, major currencies

Domestic economic variables

- GDP growth
- Total external debt
- Output gap
- Debt revenue
- Inflation
- Exports and Imports
- Reserves
- Capital flows

are available from public IMF databases and/or private data providers. Probabilities of default are from the Credit Research Initiative at the Risk Managment Institute, National University of Singapore (https://rmicri.org). Researchers can access PD data upon registration.

- Exchange rate against the U.S. dollar
- Purchasing power parity
- Real exchange rate
- Terms of trade
- Fiscal balance
- Fiscal revenue
- Fiscal expenses

Domestic financial variables

- Probability of default, banking sector
- Probability of default, non-financial sector
- Probability of default, non-bank financial sector
- Investment grade securities, share in total stock of debt securities
- Long-term bond yields
- Stock prices
- Price to income ratio, housing sector
- Price to rent ratio, housing sector
- Aggregate bank capitalization ratio,
- Bank assets to GDP ratio
- Credit to GDP ratio
- Loan to deposit ratio, banking sector
- Short-term deposit rates
- Private sector indebtedness to GDP ratio
- Financial access
- Financial efficiency in the financial sector

Other variables

- Natural disasters, material impact on GDP

- Years elapsed since the end of a crisis episode
- Cumulative number of years recorded as a crisis episode since country entered the database

A data point is labeled as a crisis if an economic crisis affects the country two years after the data point is observed and recorded. The crisis labels correspond to one of each of the following categories: external crisis, as defined in Basu et al. (2017); financial crisis, as defined in Laeven and Valencia (2017); fiscal crisis, as defined in Medas et al. (2018); and real sector crisis, as defined in Basu e et al. (2017). The crisis/no-crisis labels are not used to find the economic crisis clusters to avoid the biased label problem. The labels are used ex-post: once the clusters are identified and data points assigned to them, the labels are disclosed to assess a cluster's crisis frequency.

3.4.2 Feature representation with autoencoders

As explained earlier in Section 3.2 and illustrated in Figure 3.2 above, the information conveyed by the data in a raw form does not generate clearly separable clusters while tending to cluster data points in time periods, a trivial result. It is possible to achieve a better feature representation using autoencoders, as shown below.

Removing time trends

To remove the time trends or effects we implement a standard autoencoder with five dimensional hidden vectors h, which is trained using the original raw data input x. The autoencoder residuals are obtained subtracting the reconstructed data, $\tilde{x}$, from x, i.e. $r = x - \tilde{x}$. K-means clustering serves to assign the data point residuals, corresponding to

country-year observations, to one of ten clusters, where the number of clusters is determined using a scree plot. Figures 3.9 and 3.10 show the results.

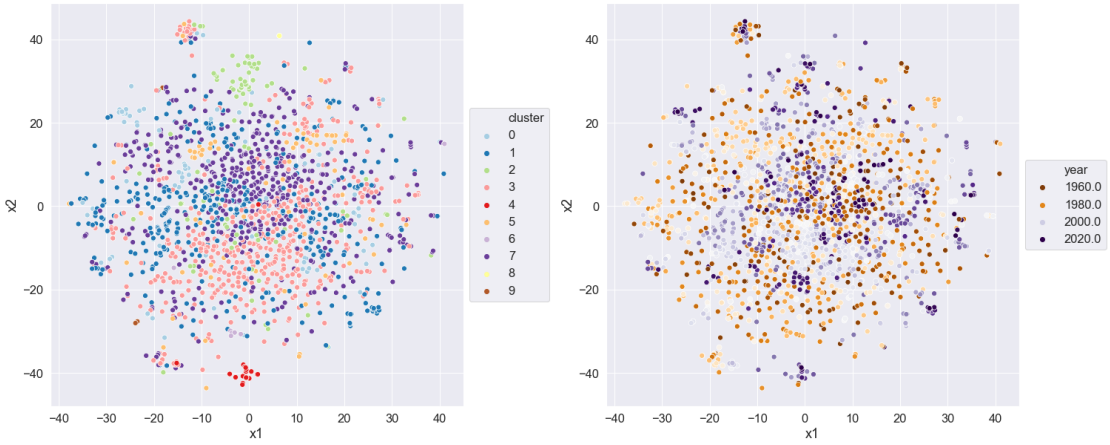

Figure 3.9: Time detrended data clusters: K-mean clusters and time periods

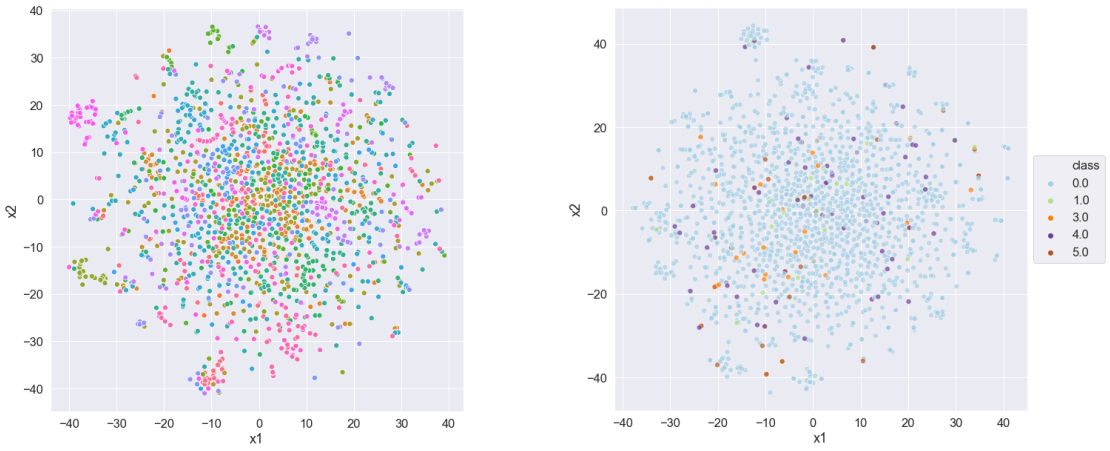

Figure 3.10: Time detrended data clusters: country and crisis presence

The residuals obtained from the first pass of the autoencoder tend to cluster in groups not clearly separable (Figure 3.9, left panel), except for one cluster (cluster 4, bottom center of the figure). However, time clustering has mostly vanished (Figure 3.9, right panel). We can also examine whether clusters group data points corresponding to the same country:

the answer is negative as data clusters comprise data points from different countries (Figure 3.10, left panel). Similarly, the clusters do not seem to be mainly comprised by data points corresponding to the same label (Figure 3.10, right panel).[7]

Balancing the data using SMOTE

After removing the time trend, it is necessary to address the imbalanced data problem. In supervised learning imbalanced data could often produce inaccurate predictions. While the problem is less severe in unsupervised learning since the learner does not use the label information. In our setup, however, it is still the case that since the number of data points labeled as no-crisis points is large, the learner may be biased to use mostly these points to identify the clusters.

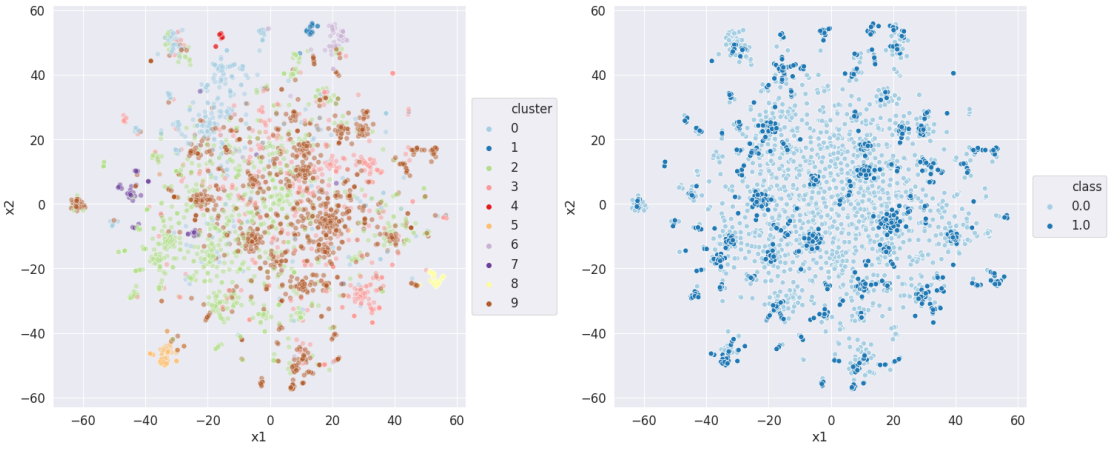

Figure 3.11: Time detrended balanced data: clusters and crisis/non-crisis observations

To resolve this issue we implement the SMOTE method to create synthetic crisis data points and to improve the accuracy of the unsupervised learner when applied to crisis

[7]Class 0 corresponds to the no crisis label, class 1 to financial crisis, class 2 to a sudden stop crisis, class 3 to an exchange rate market pressure index event, class 4 to a real sector crisis, and class 5 to a fiscal crisis.

prediction. Using SMOTE assumes that the feature distribution of data points labeled as crisis, for all crisis labels, is very similar when contrasted with the features distribution of data points labeled as non-crisis.

Figure 3.11, left panel, shows the K-means clusters obtained after applying SMOTE to the time-detrended features, i.e. the residuals after applying the autoencoders to the raw data. The cluster structure still suggests that there is scope for improving the feature representation. Nevertheless, as the right panel shows, crisis-labeled data points start to separate from the non-crisis labeled data points.

3.4.3 Clustering via Mode Contrastive Autoencoder (MCAE)

The standard autoencoder attempts to minimize a loss function proportional to the difference between the original data points and the reconstructed data points without regard for whether the residuals exhibit a multicluster structure. The mode contrastive autoencoder presented in Section 3.3 is able to capture the data structure, by minimizing the residuals, and to assign the data points to unique clusters, thanks to the inclusion of a negative log-likelihood term associated with cluster assignments as shown in Equation (3.4).

The number of clusters is a hyperparameter in the MCAE. In the absence of specific selection rules in the clustering literature we apply the elbow method to the scree plot of the mean squared distance between the data points and their centroid assignment for different number of clusters. Figure 3.12 shows the scree plot obtained applying MCAE for a number of clusters ranging from 2 to 20. We base our analysis on nine clusters since

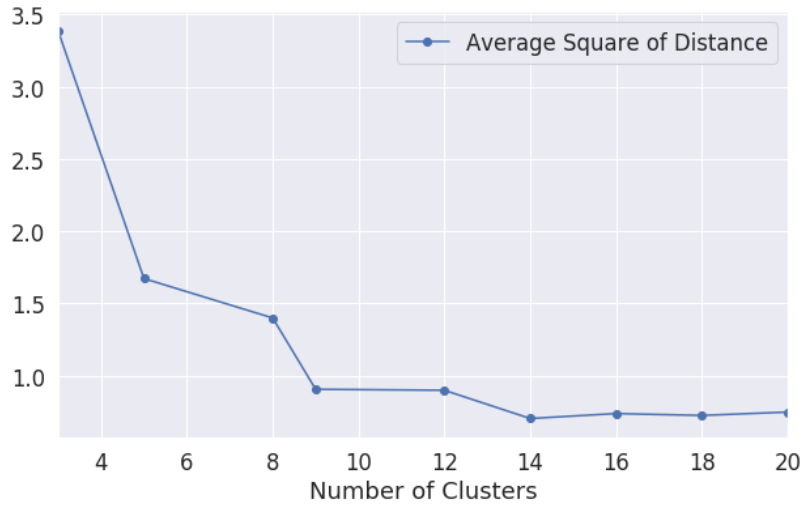

Figure 3.12: Scree plot for cluster selection

there are no substantial gains by adding more clusters.

Figure 3.13 illustrates the results obtained applying the 9-cluster MCAE to the residuals obtained by the first pass of a standard autoencoder. The left panel shows nine well differentiated clusters. Each cluster could be interpreted as a different economic regime. Under the assumption of ergodicity, i.e. the past economic regimes are recurrent, we could expect a current or future data point to belong to one of the clusters.

Recall that the MCAE does not use the labels when performing data reconstruction and clustering assignment. When labels are applied, they reveal that the MCAE clusters contain both crisis and non-crisis points coexist (Figure 3.13, right panel). This finding indicates that there are no risk-free clusters but some are safer than others in terms of crisis frequency. In addition, compared with raw data clusters, the MCAE clusters show a clear separation between crisis and non-crisis data points.

Figure 3.13: Mode Contrastive Autoencoder: clusters and crisis data points

Before discussing in more detail the crisis prediction task we assess whether some important information may be missing after applying the MCAE. The assessment is based on the distribution of the MCAE residuals. When viewed within the cluster structure (Figure 3.14, left panel), some of the residuals still tend to aggregate into three small separate clusters, suggesting MCAE may have missed some clustering information. When viewed from the perspective of crisis and non-crisis labeled data points (Figure 3.14, right panel), the spatial distribution of the residuals is very similar for both classes. These results indicate that the unFEAR method is able to extract an appropriate feature representation useful for identifying recurrent economic regimes and their crisis generating mechanisms.

3.4.4 Crisis risk measurement and crisis prediction

unFEAR, after learning an appropriate feature representation, produces clear and well separated cluster, each characterizing one of ten possible crisis clusters (Figure 3.13, right panel). For instance, the two larger clusters are safer, from a crisis realization perspective, than the smaller clusters since the number of crisis points is small relative to the

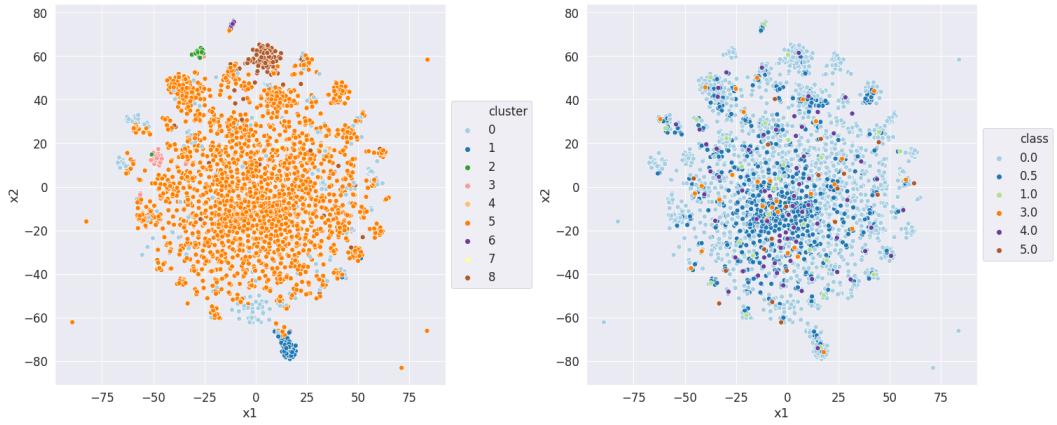

Figure 3.14: Mode Contrastive Autoencoder: residuals

number of non-crisis points. We could consider a country has a low crisis risk if its data point falls into any of these two clusters.

Table 3.2 summarizes the crisis frequencies of each cluster using two different measures. The empirical frequency is the ratio of observed crisis data points to the total number of observed data points. The shadow frequency is the ratio of observed and synthetically generated crisis data points to the total number of data points.

Crisis and non-crisis data points correspond to the number of data points, both observed and synthetic, classified as crisis and non-crisis respectively. Empirical frequency is the ratio of the number of observed crisis data points to the total number of observed data points, and the shadow empirical frequency is the ratio of the number of observed crisis data points to the total number of data points.

From an empirical frequency perspective two clusters, clusters 1 and 3, are low crisis risk clusters, in which 6 percent and 2 percent of the observed data points are crisis observations. From the shadow crisis frequency perspective only cluster 1 can be characterized as low crisis risk (12 percent). Tables 3.3 to 3.6 show the crisis observations in each cluster and highlight that a country could experience several crisis types in the same year.

Table 3.2: Crisis clusters: empirical and shadow crisis frequencies

	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Crisis data points	190	307	211	143	168
Non-crisis data points	24	452	14	1003	6
Empirical crisis frequency	43	6	61	2	71
Shadow crisis frequency	89	40	94	12	97
	Cluster 5	Cluster 6	Cluster 7	Cluster 8	
Crisis data points	232	120	94	74	
Non-crisis data points	33	5	0	2	
Empirical crisis frequency	30	71	100	85	
Shadow crisis frequency	88	96	100	97	

Table 3.3: High empirical crisis frequency clusters

Cluster 0: non-crisis observations = 24, crisis observations = 190,

empirical crisis frequency = 43 percent, shadow crisis frequency = 89 percent

Year	Country	Class	Year	Country	Class	Year	Country	Class
1974	Greece	4	1981	Belgium	4	2008	*Denmark*	4
1974	Japan	4	1985	New Zealand	5	2008	*Denmark*	1
1977	Sweden	4	2001	Iceland	3	2008	*Estonia*	4
1978	Norway	4	2006	Slovenia	3	2008	*Sweden*	4
1979	New Zealand	4	2008	Sweden	1	2009	*Spain*	4
1981	Spain	4	2008	Estonia	5	2012	*Spain*	5

Cluster 2: non-crisis observations = 14, crisis observations = 211,

empirical crisis frequency = 61 percent, shadow crisis frequency = 94 percent

Year	Country	Class	Year	Country	Class	Year	Country	Class
1980	Korea	4	2002	Israel	3	2008	United States	4
1991	Slovakia	5	2007	United States	1	2009	Iceland	4
1991	Sweden	1	2008	Japan	4	2012	*Cyprus*	3
1991	Sweden	4	2008	Portugal	1	2012	*Cyprus*	4
1997	*Korea*	1	2008	Switzerland	1	2012	*Cyprus*	5
1997	*Korea*	3	2008	Netherlands	1	2012	Iceland	3
1997	*Korea*	5	2008	Germany	1			
1998	Korea	4	2008	Belgium	1			

The cluster structure idenfied by unFEAR, and illustrated in Figure 3.13, serves as the starting point for crisis risk measurement and crisis prediction. Specifically, a country's economic fundamentals place it in one of the clusters. The crisis frequency serves as a measure of crisis risk and can be further decomposed by crisis type. As an example, suppose a country is assigned to cluster 8. In this case, crisis risk is high since the empirical frequency is 85 percent, which can be decomposed into financial crisis risk (24 percent or 2/7 of 85 percent), exchange market pressure crisis (24 percent), real sector crisis (24 percent) and fiscal sector crisis (13 percent). The risk of simultaneous crises seems negligible.

We want to point here two other extensions not undertaken in this study. First, the examination of crisis and non-crisis observations in a cluster could also serve to understand why some countries may not experience a crisis despite sharing the same economic fundamentals as crisis-affected countries. Second, Figure 3.13 shows that fitting supervised classification models for each cluster is relatively straightforward compared with fitting a global classification model on all the data set. unFEAR hence provides an adequate feature representation which can improve the precision of the crisis prediction task. In a first stage clusters are identified, and in a second stage, supervised learning models are fitted to each cluster.

3.5 Conclusions

Crisis prediction in policy making institutions benefits greatly from the increased adoption of machine learning-based predictive models. One potential concern in supervised learning-based models is the biased label problem: countries sharing similar weak economic

Table 3.4: High empirical crisis frequency clusters (continued)

Cluster 4: non-crisis observations = 6, crisis observations = 168,

empirical crisis frequency = 71 percent, shadow crisis frequency = 97 percent

Year	Country	Class	Year	Country	Class	Year	Country	Class
1992	Israel	3	2008	*Ireland*	1	2011	*Portugal*	5
1998	Singapore	4	2008	*Ireland*	4	2011	*Spain*	3
2001	Singapore	4	2009	Netherlands	4	2011	*Spain*	4
2008	*Iceland*	1	2010	*Ireland*	3	2011	Cyprus	1
2008	*Iceland*	3	2010	*Ireland*	5			
2008	*Iceland*	5	2011	*Portugal*	3			

Cluster 5: non-crisis observations = 33, crisis observations = 232,

empirical crisis frequency = 30 percent, shadow crisis frequency = 88 percent

Year	Country	Class	Year	Country	Class	Year	Country	Class
1974	United Kingdom	4	2008	*Greece*	1	2009	Slovakia	4
1975	Italy	4	2008	Greece	4	2010	*Greece*	3
1980	United States	4	2009	Germany	4	2010	*Greece*	5
1980	United Kingdom	4	2009	Czech Republic	4	2012	Italy	4
1981	Greece	4	2009	Slovenia	4			

Table 3.5: High empirical crisis frequency clusters (continued)

Cluster 6: non-crisis observations = 5, crisis observations = 120,

empirical crisis frequency = 71 percent, shadow crisis frequency = 96 percent

Year	Country	Class	Year	Country	Class	Year	Country	Class
1980	Denmark	4	1993	Spain	4	2008	*New Zealand*	3
1990	Malta	3	1998	New Zealand	3	2008	*New Zealand*	4
1993	Sweden	3	2008	*United Kingdom*	3	2008	Korea	3
1993	Spain	3	2008	*United Kingdom*	4	2013	Japan	3

Notes: Crisis and non-crisis observations correspond to the number of data points, both observed and synthetic, classified as crisis and non-crisis respectively. Country names in italic denote countries that experienced multiple crises in the same year. Crises: financial (1), sudden stop (2), exchange rate market pressure (3), real (4), fiscal (5).

fundamentals may or may not experience a future crisis due either to luck or policy actions. The biased label problem is more severe the longer the prediction horizon is. The more time elapses since when the prediction was made, the likelier that random events or policies may alter the outcome.

Unsupervised learning methods can avoid the biased label problem and cluster countries based on the similarity of their economic fundamentals. To this end, we introduced a new unsupervised feature extraction clustering method, unFEAR, where a novel mode contrastive autoencoder helped to idenfity observation clusters. Moreover, unFEAR can handle time effects and missing data efficiently.

To illustrate unFEAR's usefulness, we applied it to a sample of advanced economies. The data points to the existence of eight different clusters we associate with economic

Table 3.6: High empirical crisis frequency clusters (continued)

Cluster 7: non-crisis observations = 0, crisis observations = 94,

empirical crisis frequency = 100 percent, shadow crisis frequency = 100 percent

Year	Country	Class	Year	Country	Class	Year	Country	Class
1993	Italy	3	2008	France	1	2008	*Italy*	4
2008	Austria	1	2008	*Italy*	1			

Cluster 8: non-crisis observations = 2, crisis observations = 74,

empirical crisis frequency = 85 percent, shadow crisis frequency = 97 percent

Year	Country	Class	Year	Country	Class	Year	Country	Class
1986	Norway	5	2008	Slovenia	1	2015	Norway	3
1992	Slovenia	1	2009	Norway	4			
1998	Norway	3	2009	Canada	4			

Notes: Crisis and non-crisis observations correspond to the number of data points, both observed and synthetic, classified as crisis and non-crisis respectively. Country names in italic denote countries that experienced multiple crises in the same year. Crises: financial (1), sudden stop (2), exchange rate market pressure (3), real (4), fiscal (5).

regimes, only one of which comprising most of the observations could be considered a low risk. A country cluster assignment serve to assess its crisis risk, and the cluster per se could serve as building blocks for simpler and more precise supervised learning-based crisis prediction models.

Chapter 4

Bootstrap Aggregating and Random Forest in Economic Forecasting

4.1 Introduction

The last 30 years witnessed the dramatic developments and applications of Bagging and Random Forests. The core idea of Bagging is model averaging. Instead of choosing one estimator, Bagging considers a set of estimators trained on the bootstrap samples and then takes the average output of them, which is helpful in improving the robustness of an estimator. In Random Forest, we grow a set of Decision Trees to construct a 'forest' to balance the accuracy and robustness for forecasting. This chapter is organized as follows. In Section 4.2 we introduce Bagging and some variants. Section 4.3 discuss Decision Trees

in details. In Section 4.4, we move to Random Forest which is one of the most attractive machine learning algorithms combining Decision Trees and Bagging. Finally, in Section 4.5 and 4.6 several economic applications of Bagging and Random Forest are discussed. As we mainly focus on the regression problems rather than classification problems, the response y is a real number, unless otherwise mentioned.

4.2 Bootstrap Aggregating and Its Variants

Since the Bagging method combines many base functions in an additive form, there are more than one strategies to construct the aggregating function. In this section, we introduce the Bagging and its two variants, Subbaging and Bragging. We also discuss the Out-of-Bag Error as an important way to measure the out-of-sample error for Bagging methods.

4.2.1 Bootstrap Aggregating (Bagging)

The first Bagging algorithm was proposed in Breiman (1996). Given a sample and an estimating method, he showed that Bagging can decrease the variance of an estimator compared to the estimator running on the original sample only, which provides a way to improve the robustness of a forecast.

Let us consider a sample $\{(y_1, x_1), ..., (y_N, x_N)\}$, where y_i is the dependent variable and x_i is the p independent variables. Suppose the data generating process is $y = E(y|x) + u = f(x) + u$ where $E(u|x) = 0$ and $Var(u|x) = \sigma^2$. To estimate the unknown conditional mean function of y given x, $E(y|x) = f(x)$, we choose a function $\hat{f}(x)$ as an approximator,

such as linear regression, polynomial regression or spline, via minimizing the L_2 loss function

$$\min_{\hat{f}} \sum_{i=1}^{N} \Big(y_i - \hat{f}(x_i) \Big)^2 . \tag{4.1}$$

A drawback of this method is that, if $\hat{f}(x)$ is a nonlinear function, the estimated function $\hat{f}(x)$ may suffer from the over-fitting risk. Consider the Bias-Variance decomposition of Mean Square Error (MSE)

$$\begin{aligned} MSE &= E(y - \hat{f}(x))^2 \\ &= \Big(E\hat{f}(x) - f(x) \Big)^2 + Var(\hat{f}(x)) + Var(u) \\ &= Bias^2 + Variance + \sigma^2. \end{aligned} \tag{4.2}$$

There are three components included in the MSE: the bias of $\hat{f}(x)$, the variance of $\hat{f}(x)$, and $\sigma^2 = Var(u)$ is the variance of the irreducible error. The bias and the variance are determined by $\hat{f}(x)$. The more complex the forecast $\hat{f}(x)$ is, the lower its bias will be. But a more complex $\hat{f}(x)$ may suffer from a larger variance. By minimizing the L_2 loss function, we often decrease the bias to get the "optimal" $\hat{f}(x)$. As a result, $\hat{f}(x)$ may not be robust as it may result in much larger variance and thus a larger MSE. This is the over-fitting risk. To resolve this problem, the variance of $\hat{f}(x)$ needs to be controlled. There are several ways to control the variance, such as adding regularization term or adding random noise. Bagging is an alternative way to control the variance of $\hat{f}(x)$ via model averaging.

The procedure of Bagging is as follows:

- Based on the sample, we generate bootstrap sample $\{(y_1^b, x_1^b), ..., (x_N^b, y_N^b)\}$ via randomly drawing with replacement, with $b = 1, ..., B$.

- To each bootstrap sample, estimate $\hat{f}_b(x)$ via minimizing the L_2 loss function

$$\min_{\hat{f}_b(x)} \sum_{i=1}^{N} \left(y_i^b - \hat{f}_b(x_i^b) \right)^2.$$

- Combine all the estimated forecasts $\hat{f}_1(x), ..., \hat{f}_B(x)$ to construct a Bagging estimate

$$\hat{f}(x)_{bagging} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x).$$

Breiman (1996) proved that Bagging can make prediction more robust. Several other papers have studied why/how Bagging works. Friedman and Hall (2007) showed that Bagging could reduce the variance of the higher order terms but have no effect on the linear term when a smooth estimator is decomposed. Buja and Stuetzle (2000a) showed that Bagging could potentially improve the MSE based on second and higher order asymptotic terms but do not have any effects on the first order linear term. Buja and Stuetzle (2000b) also showed that Bagging could even increase the second order MSE terms. Bühlmann and Yu (2002) studied in the Tree-based Bagging, which is a non-smooth and non-differentiable estimator, and found that Bagging does improve the first order dominant variance term in the MSE asymptotic terms. In summary, Bagging works with its main effects on variance and it can make prediction more robust by decreasing the variance term.

4.2.2 Sub-sampling aggregating (Subagging)

The effectiveness of Bagging method is rooted in the Bootstrap method, the resampling with replacement. Sub-sampling, as another resampling method without replacement, can also be introduced to the same aggregating idea. Compared to the Bootstrap method, the Sub-sampling method often provides a similar outcome without relatively heavy com-

putations and random sampling in Bootstrap. Theoretically, Sub-sampling needs weaker assumptions than the Bootstrap method.

Comparing to the Bootstrap, Sub-sampling method needs extra parameters. Let d be the number of sample points contained in each sub-sample. Since Sub-sampling method draws samples without replacement from the original sample, the number of sub-sample is $M = \binom{N}{d}$. Thus, instead of aggregating the base predictors based on Bootstrap, we consider **Sub-sampling Aggregating**, or **Subagging**, which combines predictors trained on samples from Sub-sampling.

The procedure of Subagging is as follows:

- Based on the sample, construct $M = \binom{N}{d}$ different sub-samples $\{(y_1^m, x_1^m), ..., (y_d^m, x_d^m)\}$ via randomly drawing M times without replacement, where $m = 1, ..., M$.

- To each sub-sample, estimate $\hat{f}_m(x)$ via minimizing the L_2 loss function

$$\min_{\hat{f}_m(x)} \sum_{i=1}^{d} \left(y_i^m - \hat{f}_m(x_i^m) \right)^2.$$

- Combine all the estimated models $\hat{f}_1(x), ..., \hat{f}_M(x)$ to construct a Subagging estimate

$$\hat{f}(x)_{subagging} = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_m(x).$$

Practically, we choose $d = \alpha \times N$ where $0 < \alpha < 1$. There are several related research papers considered the similar settings for d [see Buja and Stuetzle (2000a), Buja and Stuetzle (2000b)]. Since the d is related to the computational cost, $d = N/2$ is widely used in practice.

4.2.3 Bootstrap robust aggregating (Bragging)

In Section 4.2.1 and 4.2.2, we discussed Bagging and Subagging that are based on bootstrap samples and sub-sampling samples respectively. Although they are shown to improve the robustness of a predictor, both of them are based on the mean for aggregation, which may suffer from the problem of outliers. A common way to resolve the problem of outliers is to use median instead of the mean. To construct an outlier-robust model averaging estimator, a median-based Bagging method is discussed by Bühlmann (2004), which is called **Bootstrap Robust Aggregating** or **Bragging**.

The procedure of Bragging is the following:

- Based on the sample, we generate bootstrap samples $\{(y_1^b, x_1^b), ..., (y_N^b, x_N^b)\}$ via random draws with replacement, with $b = 1, ..., B$.
- With each bootstrap sample, estimate $\hat{f}_b(x)$ via minimizing the L_2 loss function

$$\min_{\hat{f}_b(x)} \sum_{i=1}^{N} \left(y_i^b - \hat{f}_b(x_i^b) \right)^2 .$$

- Combine all the estimated models $\hat{f}_1(x), ..., \hat{f}_B(x)$ to construct a Bragging estimate

$$\hat{f}(x)_{bragging} = \text{median} \left(\hat{f}_b(x);\ b = 1, ..., B \right) .$$

To sum up, instead of taking the mean (average) on the base predictors in Bagging, Bragging takes the median of the base predictors. According to Bühlmann (2004), there are some other robust estimators, like estimating $\hat{f}_b(x)$ based on Huber's estimator, but Bragging works slightly better in practice.

4.2.4 Out-of-Bag Error for Bagging

In Sections 4.2.1 to 4.2.3, we have discussed Bagging and its two variants. In the Bootstrap-based methods like Bagging and Bragging, when we train $\hat{f}_b(x)$ on the bootstrap sample, there are many data points not selected by resampling with replacement with the probability

$$P\left((x_i, y_i) \notin Boot_b\right) = \left(1 - \frac{1}{N}\right)^N \to e^{-1} \approx 37\%,$$

where $Boot_b$ is the bth bootstrap sample. There are roughly 37% of the original sample points not included in the bth bootstrap sample. Actually, this is very useful since it can be treated as a 'test' sample for checking the out-of-sample error for $\hat{f}_b(x)$. The sample group containing all the samples not included in the bth bootstrap sample is called the Out-of-Bag sample or OOB sample. The error that the $\hat{f}_b(x)$ has on the bth out-of-bag sample is called the Out-of-Bag Error, which is equivalent to the error generated from the real test set. Breiman (1996) discussed the OOB error in details.

The bth Out-of-Bag error is calculated by

$$\begin{aligned} \widehat{err}_{OOB,b} &= \frac{\sum_{i=1}^{N} I\left((y_i, x_i) \notin Boot_b\right) \times Loss(y_i, \hat{f}_b(x_i))}{\sum_{i=1}^{N} I\left((y_i, x_i) \notin Boot_b\right)} \\ &= \frac{1}{N_b} \sum_{i=1}^{N_b} Loss\left(y^b_{i,OOB}, \hat{f}_b(x^b_{i,OOB})\right). \end{aligned} \tag{4.3}$$

The procedure of implementing the Out-of-Bag Error is the following:

- Based on the sample, we generate B different bootstrap samples $\{(y_1^b, x_1^b), ..., (y_N^b, x_N^b)\}$ via randomly drawing with replacement.

- To each bootstrap sample, estimate $\hat{f}_b(x)$ via minimizing the Loss function

$$\min_{\hat{f}_b(x)} \sum_{i=1}^{N} Loss\left(y_i^b - \hat{f}_b(x_i^b)\right).$$

- Compare the bth bootstrap sample to the original sample to get the the bth Out-of-Bag sample $\{(y_{1,OOB}^b, x_{1,OOB}^b), ..., (y_{N_b,OOB}^b, x_{N_b,OOB}^b)\}$, where N_b is the number of data points for the bth Out-of-Bag sample.

- Calculate the Out-of-Bag error of $\hat{f}_b(x)$ among all the Out-of-Bag samples

$$\begin{aligned}\widehat{err}_{OOB} &= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_b}\sum_{i=1}^{N_b} Loss\left(y_{i,OOB}^b, \hat{f}_b(x_{i,OOB}^b)\right) \\ &= \frac{1}{B}\sum_{b=1}^{B}\widehat{err}_{OOB,b}.\end{aligned}$$

4.3 Decision Trees

Although many machine learning methods, like spline and neural networks, are introduced as the base predictors in Bagging method, the most popular Bagging-based method is the so-called Random Forest proposed by Breiman (2001). Random Forest has been applied to many studies and becomes an indispensable tool for data mining and knowledge discovery. Intuitively, the main idea behind Random Forest is combining a large number of decision trees into a big forest via Bagging. In this section, we concentrate on how to construct the base learner, **Decision Tree**, for Random Forest. In Section 4.4, we discuss the Random Forest in detail. Several effective variants of Random Forest are discussed in detail in Section 4.5.

4.3.1 The structure of a decision tree

The basic idea of the decision tree has a long history and has been used in many areas including biology, computer science, and business. Biologists usually introduce a very large tree chart to describe the structure of classes containing animals or plants; in computer science, tree structure is a widely used data type or data structure with a root value and sub-trees of children with a parent node, represented as a set of linked nodes; in business, the decision tree is a usual structure choice for a flowchart that each internal node has a series of questions based on input variables.

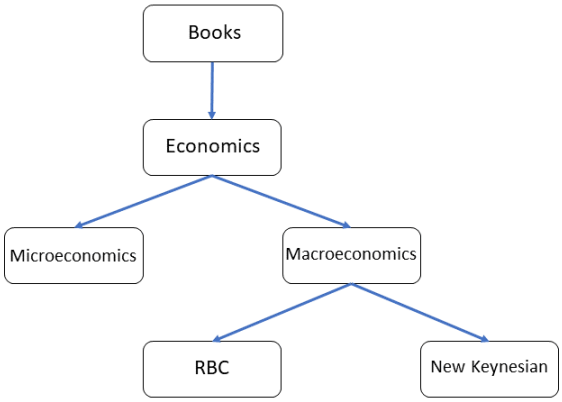

Figure 4.1: A Tree of Structured Data about Economic Books

Figure 4.1 gives an example of book data with the tree structure. Firstly, in all kinds of books, we have economic books. Then, economic books contain books about

macroeconomics, microeconomics, and others. If we concentrate on macroeconomic books, it contains books about Real Business Cycle (RBC) theory, New Keynesian theory, etc.

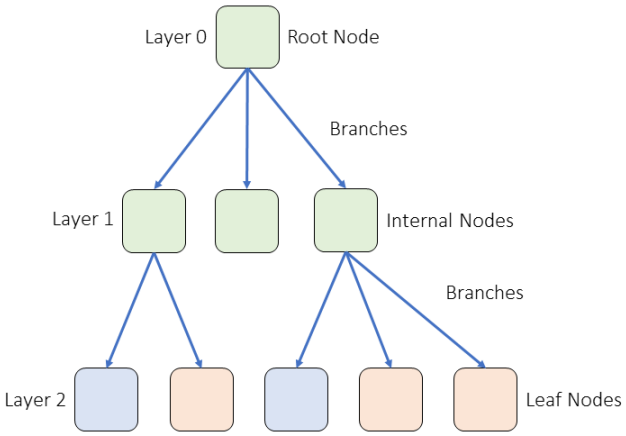

Figure 4.2: The Components in a Decision Tree

First of all, let us explore the structure of the decision tree and clarify the names of components in the decision tree. Figure 4.2 illustrates a decision tree with three layers. We can see that there are 4 components in a decision tree: root nodes, internal nodes, leaf nodes, and branches between every two layers. The root node is the beginning of a decision tree. From the only one root node, there could be two or more branches connecting to the internal nodes in the next layer. Each internal node is also called the parent node to the connected nodes in the next layer. The nodes in the next layer are called child nodes or sub-nodes. Also, every internal node contains a decision rule to decide how to connect to its sub-nodes in the next layer. At the bottom, there are several leaf nodes. They are the end of one decision tree and they represent different outputs for prediction. For example, to a regression problem, each leaf node contains a continuous output. To a classification

problem, each leaf node contains a discrete output corresponding to the labels of classes.

Intuitively, all the tree structure methods share the same intuition: the recursive splitting. Given a node, we split it into several branches connecting to its sub-nodes in the next layer. Then, to each sub-node, we split it again to get more sub-nodes in the next layer until the end of the decision tree.

In data mining and machine learning, the decision tree is widely used as a learning algorithm called Decision Tree Learning. We first construct the structure of a decision tree structure. Each node contains a decision rule. To compute the prediction of a decision tree, we feed the input to the root node and then propagate through all the layers to a leaf node, which outputs the final prediction of the decision tree. We discuss this procedure in detail via the following two examples.

Example 1: People's health

Let us consider a classification problem about people's health. Suppose a people's health $Heal$ depends on two explanatory variables, weight W and height H. Health is a binary variable with two potential outcomes: $Heal = 1$ means healthy and $Heal = 0$ means not healthy. The function of $Heal$ given H and W is

$$Heal = h(W, H).$$

Now suppose we can represent this function via several decision rules. Based on our experience, to a people with a large height, it is not healthy if this people have a relatively small weight; to a people with a small height, it is not healthy if this people have a large weight. We can write down these rules:

$$\begin{cases} Heal = 1 \quad if\ H > 180\ cm\ and\ W > 60\ kg \\ Heal = 0 \quad if\ H > 180\ cm\ and\ W < 60\ kg \\ Heal = 1 \quad if\ H < 180\ cm\ and\ W < 80\ kg \\ Heal = 0 \quad if\ H < 180\ cm\ and\ W > 80\ kg. \end{cases}$$

We first consider height H. Based on the outcome of H, there are different decision rules for weight W. Thus, it is straightforward to construct a tree to encode this procedure.

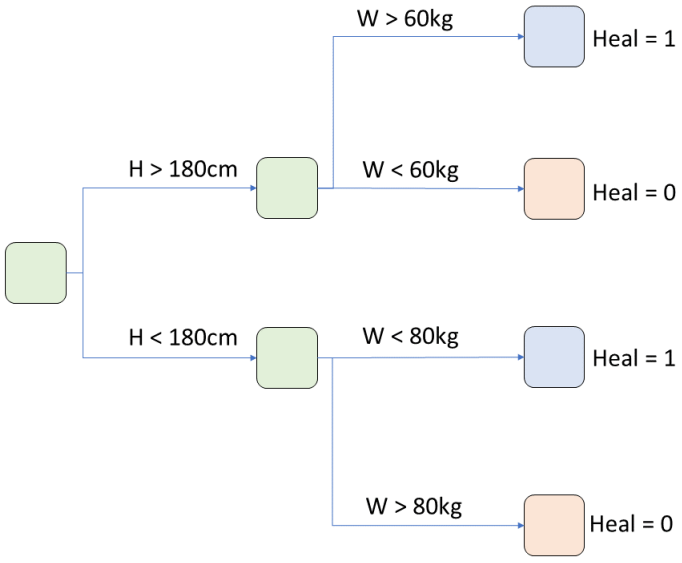

Figure 4.3: A Tree of People's Health

In Figure 4.3, the node containing H is the root node, which is the beginning of the decision procedure. The node containing W is the internal node in the first layer. In the second layer, there are four leaf nodes that give the final prediction of health. For example, to a sample ($H = 179cm,\ W = 60kg$), according to the decision rule in the root node, we choose the lower part of branches since $179 < 180$. Then, since $60 < 80$ based on the

decision rule in the internal node, we go to the third leaf node and output $Heal = 1$ as the prediction. This decision tree encodes the four decision rules into a hierarchical decision procedure.

Example 2: Women's wage

Another example is about the classic economic research: women's wage. Suppose women's wage depends on two factors: education level Edu and working experience $Expr$. Thus, this is a regression problem. The nonlinear function of women's wage is

$$Wage = g(Edu, Expr).$$

If a woman has higher education level or a longer working experience, it is much possible that woman have higher wage rate. As in Example 1, we suppose the nonlinear function g can be represented by the following rules:

$$\begin{cases} Wage = 50 \quad if\ Expr > 10\ years\ and\ Edu = college \\ Wage = 20 \quad if\ Expr > 10\ years\ and\ Edu \neq college \\ Wage = 10 \quad if\ Expr < 10\ years\ and\ Edu = college \\ Wage = 0 \quad if\ Expr < 10\ years\ and\ Edu \neq college. \end{cases}$$

In this case, we first consider the experience $Expr$. Based on it, we use different decision rules for education Edu. This procedure can also be encoded into a decision tree.

Figure 4.4 illustrates the decision tree for predicting women's wage. To a woman who has 11 years of working experience with a college degree, it is more likely that she has a higher wage rate. Thus the decision tree outcomes 50; if a woman has 3 years of

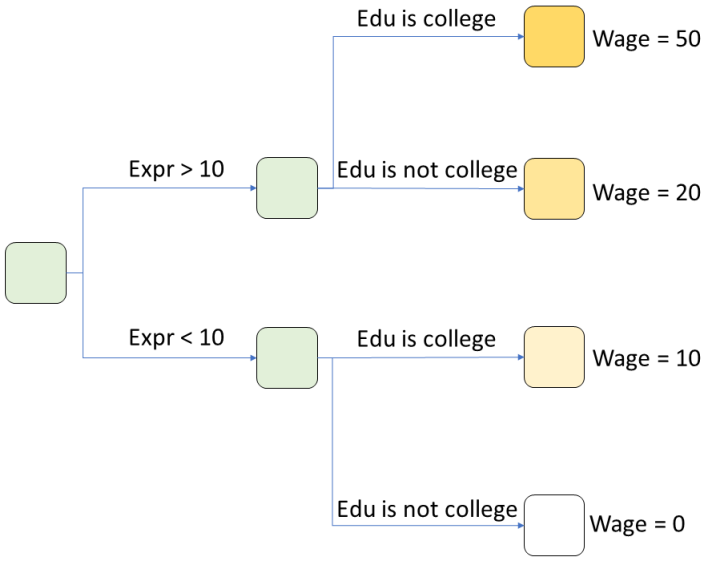

Figure 4.4: A Tree of Woman's Wage

working experience without a college degree, we expect the woman could have a hard time in searching for her job. Thus, the decision tree reports 0.

4.3.2 Growing a decision tree for classification: ID3 and C4.5

In Section 4.3.1, we have discussed how a decision tree works. Given the correct decision rules in the root and internal nodes and the outputs in the leaf nodes, the decision tree can output the prediction we need. The next question is how to decide the decision rules and values for all the nodes in a decision tree. This is related to the learning or growing of a decision tree. There are more than 20 methods to grow a decision tree. In this chapter, we only consider two very important methods. In this section, we discuss ID3 and C4.5 methods for the classification problem. In the Section 4.3.3 and 4.3.4, we will introduce the Classification and Regression Tree (CART) method for the classification problem and the

regression problem, respectively.

In the example 1 about weight, height and health, since there are two explanatory variables, H and W, we can visualize the input space in a 2D plot. Figure 13.5 illustrates all the data points $\{(Heal_1, W_1, H_1), ..., (Heal_N, W_N, H_N)\}$ in a 2D plot. The horizontal axis is the weight W and the vertical axis represents the height H. The red minus symbol means $Heal = 0$ and the blue plus symbol represents $Heal = 1$.

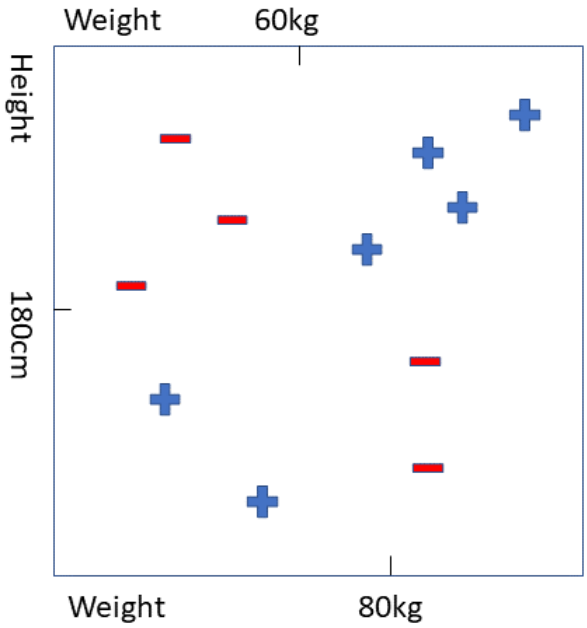

Figure 4.5: Health Data in 2D Plot

Figure 4.5 illustrates the implementation of a decision tree in a 2D plot to predict a person's health. First of all, in level 1, the decision rule at the root node is $Height > 180$ or not. In the 2D plot, this rule could be represented as a **decision stump** which is a horizontal line at $H = 180cm$. The decision stump splits the sample space into two

sub-spaces that are corresponding to the two sub-nodes in level 1. The upper space is corresponding to $H > 180cm$ and the lower space represents $H < 180cm$.

Next, we have two sub-spaces in level two. To the upper spaces, we check the rule at the right internal node, $W > 60kg$ or not. This can be represented as another vertical decision stump at $W = 60kg$ to separate upper space to two sub-spaces. Similarly, to the lower space, we also draw another vertical decision stump, which is corresponding to the decision rule at the left internal node.

Finally, we designate the final output for each of the four sub-spaces that represent the four leaf nodes. In classification problems, given a sub-space corresponding to a leaf node, we consider the number of samples for each class and then choose the class with the most number of samples as the output at this leaf node. For example, the upper left space should predict $Heal = 0$, the upper right space is corresponding to $Heal = 1$. For the regression problems, we often choose the average of all the samples at one sub-space as the output of this leaf node.

To sum up, each node in a decision tree is corresponding to space or a sub-space. The decision rule in each node is corresponding to a decision stump in this space. Then, every leaf node computes its output based on the average outputs belonging to this leaf. To grow a decision tree, there are two kinds of 'parameters' need to be figured out: the positions of all the decision stumps corresponding to the non-leaf nodes and the outputs of all the leaf nodes.

In decision tree learning, we often grow a decision tree from the root node to leaf nodes. Also in each node, we usually choose only one variable for the decision stump.

Thus, the decision stump should be orthogonal to the axis corresponding to the variable we choose. At first, we decide that the optimal decision stump for the root node. Then, to two internal nodes in layer 1, we figure out two optimal decision stumps. Then, we estimate the outputs to four leaf nodes. In other words, decision tree learning is to hierarchically split input space into sub-spaces. Comparing the two plots at the bottom of Figure 4.6, we can see the procedure of hierarchical splitting for a decision tree learning.

To sum up, each node in a decision tree is corresponding to space or a sub-space. The decision rule in each node is corresponding to a decision stump in this space. Then, every leaf node computes its output based on the average outputs belonging to this leaf. To grow a decision tree, there are two kinds of 'parameters' need to be figured out: the positions of all the decision stumps corresponding to the non-leaf nodes and the outputs of all the leaf nodes.

In decision tree learning, we often grow a decision tree from the root node to leaf nodes. Also in each node, we usually choose only one variable for the decision stump. Thus, the decision stump should be orthogonal to the axis corresponding to the variable we choose. At first, we decide that the optimal decision stump for the root node. Then, to two internal nodes in layer 1, we figure out two optimal decision stumps. Then, we estimate the outputs to four leaf nodes. In other words, decision tree learning is to hierarchically split input space into sub-spaces. Comparing the two plots at the bottom of Figure 13.6, we can see the procedure of hierarchical splitting for a decision tree learning.

Thus, the core question is how to measure the goodness of a decision stump to a node. An important measure of this problem is called **impurity**. To understand it, we

consider two decision stumps for one sample set.

Figure 4.7 shows the different cases of the sub-spaces split by two decision stumps. To the left panel, H is selected for the decision stump. In two sub-spaces, the samples have two labels. To the right panel, W is selected. The left sub-space only contains samples with label $Heal = 0$ and the right sub-space only contains samples with label $Heal = 1$. Intuitively, we can say that the two sub-spaces in the left panel are impure compared to the sub-spaces in the right panel. The sub-spaces in the right panel should have lower impurity. Obviously, the decision stump in the right panel is better than the left panel since it generates more pure sub-spaces.

Mathematically, the information entropy is a great measure of impurity. The more labels of samples are contained in one sub-space, the higher entropy of the sub-space has. To discuss the entropy-based tree growing clearly, we introduce a new definition: information gain. The information or entropy for an input space S is

$$Info(S) = -\sum_{c=1}^{C} p_c \log_2(p_c), \tag{4.4}$$

where C is the total number of classes or labels contained in space S. p_c is the frequency of samples for one class in the space S. It can be estimated by

$$p_c = \frac{1}{N_S} \sum_{x_i \in S} I(y_i = c), \tag{4.5}$$

where N_S is the total number of samples in space S. $I(y_i = c)$ is an indicator function measuring the label y_i is the cth class or not.

Suppose we choose D as a decision stump and it separates the space S into two

sub-spaces. For example, if we choose D as $x = 5$, the two sub-spaces are corresponding to $x < 5$ and $x > 5$. Then, we calculate the distinct entropies for two sub-spaces. Thus, if the space S is separated into v different sub-spaces, the average entropy of S after splitting is

$$Info_D(S) = \sum_{j=1}^{v} \frac{N_{S_j}}{N_S} \times Info(S_j), \tag{4.6}$$

where v is the number of sub-spaces generated by D. To binary splitting, $v = 2$. S_j is the jth sub-space and it satisfies: $S_i \cap S_j = \emptyset$ if $i \neq j$ and $\bigcup_i S_i = S$. N_{S_j} and N_S are the number of samples contained in S_j and S.

Obviously, the information or entropy for space S changes before and after splitting based on decision stump D. Thus, we define the information gain of D as

$$Gain(D) = Info(S) - Info_D(S). \tag{4.7}$$

Example 3: Predicting economic growth

Consider an example of predicting economic growth G based on two factors: Inflation Rate I and Net Export NX. Suppose G is a binary variable where $G = 1$ for expansion and $G = 0$ for recession. Then, the growth G is an unknown function of the inflation rate I and the Net Export NX

$$G = G(I, NX).$$

From the left panel in Figure 4.8, we can see the sample distribution of economic growth G. For example, if there is high inflation rate I and high net export NX, we observe the economic expansion where $G = 1$; if there are high inflation rate I but low net export

NX, the economy will be in recession with $G = 0$.

Let us consider a decision tree with only the root node and two leaf nodes to fit the samples. In the right panel, we choose $D : I = 10\%$ as the decision stump in the root node. Thus, the space S is splitted into two sub-spaces S_1 and S_2. According to Equation (4.4), the information to the original space S is

$$\begin{aligned} Info(S) &= -\sum_{c=1}^{2} p_c \log_2(p_c) \\ &= -(p_1 \log_2(p_1) + p_2 \log_2(p_2)) \\ &= -\left(\frac{4}{8}\log_2\left(\frac{4}{8}\right) + \frac{4}{8}\log_2\left(\frac{4}{8}\right)\right) \\ &= 1, \end{aligned}$$

where class 1 is corresponding to $G = 0$ and class 2 to $G = 1$. And $p_1 = \frac{4}{8}$ means that there are 4 samples with $G = 0$ out of 8 samples.

After splitting, the information to the sub-space S_1 is

$$\begin{aligned} Info(S_1) &= -\sum_{c=1}^{2} p_c \log_2(p_c) \\ &= -p_1 \log_2(p_1) + 0 \\ &= -\left(\frac{2}{2}\right)\log_2\left(\frac{2}{2}\right) = 0. \end{aligned}$$

The information to the sub-space S_2 is

$$
\begin{aligned}
Info(S_2) &= -\sum_{c=1}^{2} p_c \log_2(p_c) \\
&= -(p_1 \log_2(p_1) + p_1 \log_2(p_1)) \\
&= -\left(\frac{2}{6}\log_2\left(\frac{2}{6}\right) + \frac{4}{6}\log_2\left(\frac{4}{6}\right)\right) \\
&= 0.92.
\end{aligned}
$$

$$
\begin{aligned}
Info_D(S) &= \sum_{j=1}^{v} \frac{N_{S_j}}{N_S} \times Info(S_j) \\
&= \frac{N_{S_1}}{N_S} \times Info(S_1) + \frac{N_{S_2}}{N_S} \times Info(S_2) \\
&= \frac{2}{8} \times 0 + \frac{6}{8} \times 0.92 \\
&= 0.69.
\end{aligned}
$$

After splitting, the information decreases from 1 to 0.69. According to Equation (4.6), the information gain of D is

$$Gain(D) = Info(S) - Info_D(S) = 0.31.$$

To sum up, we can find the best decision stump to maximizing the information gain such that the optimal decision stump can be found. From the root node, we repeat finding the best decision stump to each internal node until stopped at the leaf nodes. This method for tree growing is called **ID3** introduced by Quinlan (1986).

Practically, the procedure of implementing the decision tree for classification based on ID3 is the following:

- Suppose the sample is $\{(y_1, x_1), ..., (y_N, x_N)\}$ where $y_i \in (0, 1)$ and x_i is a p dimensional vector. To the first dimension, gather all the data orderly as $x_{1,(i)}, ..., x_{1,(N)}$.

- Search the parameter d_1 respect to $D_1 : x_1 = d_1$ through $x_{1,(i)}$ to $x_{1,(N)}$ such that

$$\max_{D_1} \ Gain(D_1) = \max_{D_1} \ \left(Info(S) - Info_{D_1}(S)\right).$$

- Find the best $D_2 : x_2 = d_2, ..., D_p : x_p = d_p$ and then choose the optimal D such that

$$\max_{D} \ Gain(D) = \max_{D} \ \left(Info(S) - Info_{D}(S)\right).$$

- Repeatedly run the splitting procedure until every node containing one label of y. Finally, take the label of y from one leaf node as its output.

One problem this method suffer from is related to over-fitting. Suppose we have N data points in space S. According to the rule that maximizing the information gain, we can find that the optimal result is separating one sample into one sub-space such that the entropy is zero in each sub-space. This is not a reasonable choice since it is not robust to noise in the samples. To prevent that, we can introduce a revised version of information gain from **C4.5** method.

C4.5 introduces a measure for information represented via splitting, which is called

Splitting Information

$$Split\ Info_D(S) = -\sum_{j=1}^{v} \frac{N_{S_j}}{N_S} \times \log_2 \frac{N_{S_j}}{N_S}, \tag{4.8}$$

where $v = 2$ for the binary splitting.

Obviously, this is an entropy based on the number of splitting or the number of sub-spaces. The more the sub-spaces are, the higher the splitting information we will get. To show this conclusion, let us go back to the economic growth case illustrated in Figure 4.9.

To the left case, the splitting information is computed based on Equation (4.8) as

$$\begin{aligned} Split\ Info_D(S) &= -\sum_{j=1}^{v} \frac{N_{S_j}}{N_S} \times \log_2 \frac{N_{S_j}}{N_S} \\ &= -\left(\frac{N_{S_1}}{N_S} \times \log_2 \frac{N_{S_1}}{N_S} + \frac{N_{S_2}}{N_S} \times \log_2 \frac{N_{S_2}}{N_S} \right) \\ &= -\left(\frac{2}{8} \times \log_2 \frac{2}{8} + \frac{6}{8} \times \log_2 \frac{6}{8} \right) \\ &= 0.81. \end{aligned}$$

To the right case, the splitting information is

$$\begin{aligned} Split\ Info_D(S) &= -\sum_{j=1}^{v} \frac{N_{S_j}}{N_S} \times \log_2 \frac{N_{S_j}}{N_S} \\ &= -\left(\frac{N_{S_1}}{N_S} \times \log_2 \frac{N_{S_1}}{N_S} + \frac{N_{S_2}}{N_S} \times \log_2 \frac{N_{S_2}}{N_S} + \frac{N_{S_3}}{N_S} \times \log_2 \frac{N_{S_3}}{N_S} \right) \\ &= -\left(\frac{2}{8} \times \log_2 \frac{2}{8} + \frac{4}{8} \times \log_2 \frac{4}{8} + \frac{2}{8} \times \log_2 \frac{2}{8} \right) \\ &= 1.5. \end{aligned}$$

Thus, when there are more sub-spaces, the splitting information increases. In

other words, splitting information is the 'cost' for generating sub-spaces. Now, instead of information gain, we can use a new measure called **Gain Ratio(D)**

$$Gain\ Ratio(D) = \frac{Gain(D)}{Split\ Info(D)}. \tag{4.9}$$

When we generate more sub-spaces, the information gain increases but splitting information is higher at the same time. Thus, to maximize the Gain Ratio of D, we can make great trade-offs. This is the main idea of **C4.5**, an improved version of ID3 introduced by Quinlan(1994).

Summarizing, the procedure of implementing the decision tree for classification based on C4.5 is the following:

- Suppose the sample is $\{(y_1, x_1), ..., (y_N, x_N)\}$ where $y_i \in (0, 1)$ and x_i is a p dimensional vector. To the first dimension, gather all the data orderly as $x_{1,(i)}, ..., x_{1,(N)}$.

- Search the parameter d_1 respect to $D_1 : x_1 = d_1$ through $x_{1,(i)}$ to $x_{1,(N)}$ such that

$$\max_{D_1}\ Gain\ Ratio(D_1) = \max_{D_1}\left(\frac{Gain(D_1)}{Split\ Info(D_1)}\right).$$

- Find the best $D_2 : x_2 = d_2, ..., D_p : x_p = d_p$ and then choose the optimal D such that

$$\max_{D}\ Gain\ Ratio(D) = \max_{D}\left(\frac{Gain(D)}{Split\ Info(D)}\right).$$

- Repeatedly run the splitting procedure until the Gain Ratio is less than 1. Finally, take the most frequency label of y from one leaf node as its output.

4.3.3 Growing a decision tree for classification: CART

In Section 4.3.2, we have discussed related methods about how to grow a tree based on ID3 and C4.5 methods. In this section, we introduce another way to construct a decision tree, the **Classification and Regression Tree** (**CART**), which not only features great performance but very easy to implement in practice for both classification and regression tasks.

The main difference between ID3, C4.5, and CART is the measure of information. ID3 and C4.5 choose the entropy to construct the Information Gain and Gain Ratio. In CART, we introduce a new measure for deciding the best decision stump called the **Gini Index** or **Gini Impurity**. The definition of Gini Impurity is

$$Gini(S) = \sum_{j=1}^{M} p_j(1 - p_j) = 1 - \sum_{j=1}^{M} p_j^2, \tag{4.10}$$

where M is the number of classes in node spaces S and p_j is the frequency of class j in node space S. Intuitively, this is the variance of the binary distribution. That is, CART chooses the variance as the impurity measure.

Figure 4.10 illustrates the difference between Entropy and Gini Impurity. Given x-axis as the proportion of sample belonging to one class, we can see that two curves are very similar. Then, we have the new Gini Impurity after binary splitting

$$Gini_D(S) = \frac{N_{S_1}}{N_S} Gini(S_1) + \frac{N_{S_2}}{N_S} Gini(S_2). \tag{4.11}$$

where the N_S, N_{S_1}, N_{S_2} are the numbers of sample points in space S, S_1, S_2 respectively.

Similarly to the information gain in ID3, we consider the difference of Gini impurity as the measure of goodness of decision stump

$$\Delta Gini_D(S) = Gini(S) - Gini_D(S). \tag{4.12}$$

As we discuss in ID3 method, if we grow a decision tree via maximizing the information gain in each node, it is the best choice that we split all the data points in one space such that each subspace contains one sample point. ID3 and CART may suffer from this risk. C4.5 should be a better choice than ID3 and CART, but it has a fixed rule to prevent over-fitting which cannot be adaptive to data.

To solve this problem, let us consider the total cost of growing a decision tree

$$Total\ Cost = Measure\ of\ Fit + Measure\ of\ Complexity. \tag{4.13}$$

The total cost contains two main parts: the measure of fit is related to the goodness of the model, as the error rate in classification problem; the measure of complexity describes the power of the model. To balance the two measures in growing a decision tree, we often choose the following function as the objective:

$$L = Loss(y_i, x_i; tree) + \lambda\Omega(numbers\ of\ leaf\ nodes).$$

The first term is related to the loss of the decision tree. To classification problem, we can use the error rate on the samples as the loss. The second term is a measure of complexity based on the number of leaf nodes. Ω is an arbitrary function like the absolute function. λ is a tuning parameter balancing the loss and the complexity. Many machine learning and

regressions like Lasso and Ridge Regression follow this framework. Also, since the second term penalizes on the number of leaf nodes, this is also called **pruning** a decision tree.

The procedure of implementing the decision tree for classification based on CART is the following:

- Suppose the sample is $\{(y_1, x_1), ..., (y_N, x_N)\}$ where $y_i \in \{0, 1\}$ and x_i is a p dimensional vector. To the first dimension, gather all the data orderly as $x_{1,(i)}, ..., x_{1,(N)}$.

- Search the parameter d_1 respect to $D_1 : x_1 = d_1$ through $x_{1,(i)}$ to $x_{1,(N)}$ such that

$$\max_{D_1} \ \Delta Gini_{D_1}(S) = \max_{D_1} \left(Gini(S) - Gini_{D_1}(S) \right).$$

- Find the best $D_2 : x_2 = d_2, ..., D_p : x_p = d_p$ and then choose the optimal D such that

$$\max_{D} \ \Delta Gini_{D}(S) = \max_{D} \left(Gini(S) - Gini_{D}(S) \right).$$

- Based on the new decision stump, calculate the error rate for the decision tree and the total loss function

$$L = error\ rate(y_i, x_i; tree) + \lambda \Omega(numbers\ of\ leaf\ nodes).$$

- Repeatedly run the splitting procedure until the total loss function starting to increase. Finally, take the most frequency label of y from one leaf node as its output.

4.3.4 Growing a decision tree for regression: CART

The Information Gain and Gini Impurity are a very important measures when we are implementing a classification problem. In economic research, we often consider more regression problems with the continuous response. Thus, instead of the information gain, we choose the variation to measure the goodness of a decision stump

$$Variation(S) = \sum_{i=1}^{N}(y_i - \bar{y})^2, \tag{4.14}$$

where N is the number of data points belong to the space S. After several splitting, the space S is separated into v sub-spaces $S_1, ..., S_v$, we can define the average variance after splitting the space S

$$Variation_D(S) = \frac{1}{v}\sum_{j=1}^{v} Variation_j(S), \tag{4.15}$$

where v is the number of the sub-spaces separated by D. Again, to binary splitting, we have $v = 2$. Thus, we have a new information gain for regression method

$$Gain(D) = Variation(S) - Variation_D(S). \tag{4.16}$$

Based on the total cost in Equation (4.13), we choose the same formula for regression

$$L = Loss(y_i, x_i; tree) + \lambda\Omega(numbers\ of\ leaf\ nodes),$$

where $Loss(y_i, x_i; tree)$ is the L_2 loss function.

Thus, the procedure of implementing the decision tree for regression based on CART is the following:

- Suppose the sample is $\{(y_1, x_1), ..., (y_N, x_N)\}$ where y_i is a real value and x_i is a p dimensional vector. To the first dimension, gather all the data orderly as $x_{1,(i)}, ..., x_{1,(N)}$.

- Search the parameter d_1 respect to $D_1 : x_1 = d_1$ through $x_{1,(i)}$ to $x_{1,(N)}$ such that

$$\max_{D_1}\ Gain(D) = \max_{D_1} \left(Variation(S) - Variation_{D_1}(S)\right).$$

- Find the best $D_2 : x_2 = d_2, ..., D_p : x_p = d_p$ and then choose the optimal D such that

$$\max_{D}\ Gain(D) = \max_{D} \left(Variation(S) - Variation_{D}(S)\right).$$

- Based on the new decision stump, compute the loss for the decision tree and the total loss function

$$L = Loss(y_i, x_i; tree) + \lambda\Omega(numbers\ of\ leaf\ nodes).$$

- Repeatedly run the splitting procedure until the total loss function starting to increase. Finally, take an average of y from one leaf node as its output.

4.3.5 Variable importance in a decision tree

In Sections 4.3.2 to 4.3.4, we discussed how to grow a decision tree. In this section, we consider another problem: how to measure the importance of the variable.

In the procedure of growing a decision tree, each time we split one internal node into two child nodes, one variable should be selected based on the information gain or variation gain. Thus, for an important variable, the decision tree should choose it frequently among all the internal nodes. Conversely, the variables may be selected just a few times if the variables are not very important. To the jth variable, [31] defined a **relative importance** as

$$I_j^2 = \sum_{t=1}^{T-1} e_t^2 I(v(t) = j), \tag{4.17}$$

where T is the number of internal nodes (non-leaf nodes) in a decision tree, $v(t)$ is the variable selected by node t. e_t is the error improvement based on before and after splitting the space via variable $v(t)$. To regression task, it can be a gain of variation. To classification problem, it is related to information gain of entropy or the difference of Gini Impurity.

For example, let us consider a CART tree to a regression problem. Suppose we split the tth node into two nodes based on variable j selected by the decision stump D. Then, we can calculate the value of the information gain $Gain(D) = Variation(S) - Variation_D(S)$.

This is the error improvement e_t. Thus, considering all the internal nodes, we compute all the e_t^2 to get I_j^2.

If variable j is very important, the error improvement should be very large and $I(v(t) = j)$ often equals to 1 since variable j is usually selected. As a result, the measure I_j^2 is relatively large; conversely, it a variable is not very important, the error improvement based on this variable cannot be so large, which leads to a small I_j^2. After we growing a decision tree on a training set, we often calculate it on the test set.

4.4 Random Forests

Random Forest is a combination of many decision trees based on Bagging. In the first paper about Random Forest, Breiman (2001) discussed the theories behind the Random Forest and compared Random Forest with other ensemble methods. From this section, we start to discuss Random Forests in detail.

4.4.1 Constructing a random forest

As we discussed in Section 4.2, Bagging method can generate a lot of base learners trained on bootstrap samples and then combine them to predict. If we consider combining a set of unbiased estimators or predictors, Bagging works by decreasing the variances of the predictors but keeping the means unaffected.

For example, let us consider B numbers of unbiased estimators $f_1, f_2, ..., f_B$ with same variance σ^2. If they are i.i.d, it is easy to show that the variance of average estimator is

$$Var(g) = Var\left(\frac{1}{B}\sum_{b=1}^{B} f_b\right) = \frac{1}{B}\sigma^2. \tag{4.18}$$

But if the unbiased estimators are correlated, the variance of the average estimator is

$$\begin{aligned} Var(g) &= \frac{1}{B^2} Var\left(\sum_{b=1}^{B} f_b\right) \\ &= \frac{1}{B^2}\left(\sum_{b=1}^{B} Var(f_b) + 2\sum_{b\neq c} cov(f_b, f_c)\right) \\ &= \frac{1}{B^2}(B\sigma^2 + (B^2 - B)\rho\sigma^2) \\ &= \rho\sigma^2 + \frac{(1-\rho)}{B}\sigma^2, \end{aligned} \tag{4.19}$$

where ρ is the correlation coefficient between two estimators.

The variance of average estimator depends on the number of base estimators and the correlation between estimators. Even if we can decrease the second term to zero via adding increasingly large numbers of estimators, the first term remains at the same level if the estimators are not independent. Similarly, in Bagging, even if we can combine a lot of predictors based on Bootstrap, the variance cannot keep decreasing if the predictors are dependent with each other.

In practice, since most of the bootstrap sample are very similar, the decision trees trained on these sample sets are often similar and highly correlated with others. Thus, average estimators of similar decision trees can be more robust but do not perform much better than a single decision tree. That is the reason why Bagging decision trees or other base learners may not work so well in prediction.

Compare to Bagging decision trees, which only combines many trees based on

Bootstrap to decrease the second term $\frac{(1-\rho)}{B}\sigma^2$, Random Forest also considers controlling the first term $\rho\sigma^2$. To decrease the correlation between decision trees, Random Forest introduces the so-called **random subset projection** or **random feature projection** during growing a decision tree. That is, instead of applying all the variables in one tree, each decision tree chooses only a subset of variables at each potential split in Random Forest. Also, comparing to the classic decision tree, in Random Forest, decision trees are not necessarily pruned by penalizing the number of leaf nodes but grow all the way to the end. Random subset projection can significantly decrease the correlations between trees since different trees grow on different sets of attributes, which leads to a smaller $\rho\sigma^2$. But it could affect the second term $\frac{(1-\rho)}{B}\sigma^2$ and the unbiasedness of decision trees since they cannot predict dependent variables based on all the attributes. Thus, we need to select the number of variables to select in each split to balance the first and the second term.

The procedure of constructing a Random Forest is the following:

- Generate B number of bootstrap sample sets.
- On each sample set, grow a decision tree all the way to the end.
- During growing a tree, randomly select m variables at each potential split (random feature projection).
- Combine the B decision trees to a Random Forest. To regression, take the average output among all the trees; to classification, consider the vote of all the trees.

We can choose the hyper-parameter m based on cross-validation but this is very time-consuming when B is very large. Thus, to the classification task, m is often chosen as

$1 \leq m \leq \sqrt{p}$; to regression task, we choose m as $1 \leq m \leq p/3$, where p is the number of variables. To the node size, for every decision tree, we grow it all the way to the end for the classification task, while we grow to that every leaf node has no more than $n_{min} = 5$ sample inside for regression task.

4.4.2 Variable importance in a random forest

In Section 4.3.5, we discussed the relative importance I_j^2 to measure the importance of a variable in a decision tree. Since Random Forest is a linear combination of decision trees, we can introduce an average relative importance

$$I_j^2 = \frac{1}{B} \sum_{b=1}^{B} I_j^2(b), \tag{4.20}$$

where $I_j^2(b)$ is the relative importance for the bth decision tree

$$I_j^2(b) = \sum_{t=1}^{T_b - 1} e_t^2 I(v(t)_b = j). \tag{4.21}$$

A drawback for this measure is that we need to check every node in a decision tree. This is not very efficient if there is too many samples or large numbers of the decision tree in a Random Forest.

Surprisingly, Random Forest provides a much simpler but very effective way to measure the importance of variables via **random permutation**. That is, for one variable, we perturb the samples by random permutation. For example, after constructing a Random Forest, to the jth variable along all the samples $x_j = (x_{j,1}, x_{j,2}, ..., x_{j,i}, ..., x_{j,N})$, we randomly rearrange all the x to generate a new series of samples $x_j^* = (x_{j,1}^*, x_{j,2}^*, ..., x_{j,i}^*, ..., x_{j,N}^*) =$

$(x_{j,2}, x_{j,10}, ..., x_{j,N-4}, ..., x_{j,i+5})$, which is the original x_j with random sample order. Then, we test the Random Forest on that to get the error rate or mean square error under random permutation. Intuitively, if one variable is not important, comparing the test error on the original test sample, the test error on permutation test samples should not change a lot since this variable may not usually be selected by the nodes in a decision tree. Given a test set with N_t samples, the variable importance under random permutation is

$$\begin{aligned} VI_j &= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_t}\sum_{i=1}^{N_t} Loss(y_i, tree_b(x_{1,i}, ..., x^*_{j,i}, ...)) - Loss(y_i, tree_b(x_{1,i}, ..., x_{j,i}, ...)) \\ &= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_t}\sum_{i=1}^{N_t} \Delta Loss(y_i, tree_b(x_{1,i}, ..., x^*_{j,i}, ...)). \end{aligned} \tag{4.22}$$

In practice, one way to estimate the test error is sample splitting. We split one data set into a training set and a validation set and then estimate the test error on the validation set. But this is not efficient because of the loss of samples. When we discussed in Bagging in Section 4.2.4, in terms of Bootstrap sampling, all the Bagging methods could leave about one third sample points untouched, that are the Out-of-Bag samples. Since Random Forest is a Bagging method, we can use the OOB error as the test error. This is a very efficient way to implement since each time we add a decision tree based on a new bootstrap sample, we can test the variable importance on the new OOB samples.

Based on the OOB error, we redefine the measure of variable importance as

$$
\begin{aligned}
VI_j^{OOB} &= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_b}\sum_{i=1}^{N_b}(Loss(y_{i,OOB}, tree_b(x_{1,i,OOB}, ..., x^*_{j,i,OOB}, ...)) \\
&\qquad\qquad - Loss(y_{i,OOB}, tree_b(x_{1,i,OOB}, ..., x_{j,i,OOB}, ...; tree_b))) \\
&= \frac{1}{B}\sum_{b=1}^{B}\left(\widehat{err}^*_{OOB,b} - \widehat{err}_{OOB,b}\right) \\
&= \frac{1}{B}\sum_{b=1}^{B}\Delta\widehat{err}_{OOB,b}
\end{aligned}
\tag{4.23}
$$

where N_b is the sample size of the bth OOB sample.

The implementing procedure is the following:

- To bth bootstrap sample set, grow a decision tree.
- Find the sample point not contained in the sample set and construct the bth Out-of-Bag sample set.
- Compute OOB error for the bth decision tree based on the OOB sample with and without random permutation.
- Calculate VI_j^{OOB} to measure the jth variable importance.

One related topic is about the variable selection in Random Forest. Based on the variable importance, we can compare the importance between two variables. Thus, could we select relevant variables based on this measure? A simple way to implement is designating a threshold value for variable importance and select the variables with high importance only. But there is no theory about how to decide the threshold value such that we can select relevant variables correctly. Strobl, Boulesteix, Kneib, Augustin and Zeileis

(2008) and Janitza, Celik and Boulesteix (2016) considered the hypothesis testing to select variables in Random Forest.

The last but not the least, the issue of variable dependence need to be considered when we measure the variable importance via random permutation. For example, if we implement a linear model by regressing the level of health on weight and height, the coefficient on the weight could be very unstable if weight and height are highly correlated. Similarly, in random forest, if two variables are correlated, we cannot get an accurate measure of importance via random permuting on the variable. Strobl, Boulesteix, Zeileis and Hothorn (2007) discussed the topics about the bias in random forest variable importance.

To resolve this issue, we need to check the dependence among all the variables. Some methods like PCA could be introduced to decorrelate the variables, but they may affect the interpretations of the variables. Strobl, Boulesteix, Kneib, Augustin and Zeileis (2008) proposed a method called the **conditional variable importance**.

The implementing procedure is the following:

- Given variable x_j, find a group of variables $Z = \{z_1, z_2, ..., \}$ that are correlated with x_j.
- To the bth decision tree, find out all the internal nodes containing the variables in Z.
- Extract the cutpoints from the nodes and create a grid by means of bisecting the sample space in each cutpoint.
- In this grid, permute the x_j to compute the OOB accuracy. The OOB error of the bth tree is the difference between OOB accuracy with and without permutation given

Z.

- Consider the average of all the trees' OOB error as the forest's OOB error.

4.4.3 Random forest as the adaptive kernel functions

Now we start to discuss some related theories behind Random Forest to uncover why Random Forest works. Basically, Random Forest or decision tree ensemble methods can be seen as a local method. For example, it is easy to find that the predicted value of a given data totally depends on the average of y_i in one of the leaf node. In other words, the predicted value only depends on a 'neighborhood' samples belong to the leaf node. Similarly, Breiman (2000) showed that Random Forest which is grown using i.i.d random vectors in the tree construction are equivalent to a kernel acting on the true margin.

Without loss of generality, let us consider a Random Forest with B decision trees for a binary classification task. To one decision tree, suppose R as the area of one of leaf node with the responses as $R = +1$ or $R = -1$. We have the labeling rule for $R = +1$ to this leaf node

$$\int_R P(+1|z)P(dz) \geq \int_R P(-1|z)P(dz), \tag{4.24}$$

where z represents all the possible inputs inculded in the leaf node. Intuitively, by considering all the samples in R, if more samples with the label as $+1$, the response of R is $+1$. Otherwise, we label the response of R as -1.

Based on Equation (4.24), we have the output $+1$ from a decision tree given an input x when Equation (4.25) holds

$$\int_{R_x(\theta)} P(1|z)P(dz) \geq \int_{R_x(\theta)} P(-1|z)P(dz), \tag{4.25}$$

where $R_x(\theta)$ is the area of the leaf node containing x and θ is the parameter of the decision tree. Let $D(z) = P(1|z) - P(-1|z)$, then Equation (4.25) can be written as

$$\int_{R_x(\theta)} D(z)P(dz) \geq 0. \tag{4.26}$$

According to Equation (4.26), the prediction of the bth decision tree is

$$\widehat{y} = \begin{cases} 1 & if \int_{R_x(\theta_b)} D(z)P(dz) \geq 0 \\ -1 & if \int_{R_x(\theta_b)} D(z)P(dz) \leq 0. \end{cases}$$

Now let us introduce an indicator function $I(x, z \in R(\theta_b))$ to represent the event $z \in R_x(\theta_b)$, we have

$$\widehat{y} = \begin{cases} 1 & if \int I(x, z \in R(\theta_b))D(z)P(dz) \geq 0 \\ -1 & if \int I(x, z \in R(\theta_b))D(z)P(dz) \leq 0. \end{cases}$$

Obviously, the indicator function $I(x, z \in R(\theta_b))$ can be seen as a kernel weighted function $K(x, z)$. Also, this kernel function is not smooth since it only considers the sample in the leaf node $R(\theta_b)$. Intuitively, it means that one decision tree can learn to construct a distribution plot and then works via the 'hard' kernel weighting.

Let us consider a Random Forest. Compare to a single decision tree, Random Forest contains B decision trees. Assume in bth decision tree, the number of leaf nodes is T_b. Thus, we can derive a kernel function for Random Forest

$$K_{RF}(x,z) = \frac{1}{B}\sum_{b=1}^{B}\sum_{t=1}^{T_b} I(x,z \in R^t(\theta_b)). \tag{4.27}$$

This is a discrete kernel combining all the leaf nodes from B decision trees. Additionally, when $B \to \infty$, we have

$$\begin{aligned} K_{RF}(x,z) &= \frac{1}{B}\sum_{b=1}^{B}\sum_{t=1}^{T_b} I(x,z \in R^t(\theta_b)) \\ &\to P_\theta(x,z \in A(\theta)), \end{aligned} \tag{4.28}$$

where $A(\theta)$ is the area based on the Random Forest and it contains infinite number of leaf nodes from infinite decision trees. When $B \to \infty$, we can see that the kernel function will converge to a probability measure. That is, the hard kernel function will be a smoother kernel function when we have increasingly number of decision trees. Thus, the final output for Random Forest in this case should be

$$\widehat{y}_{RF} = \begin{cases} 1 & if \int K_{RF}(x,z)D(z)P(dz) \geq 0 \\ -1 & if \int K_{RF}(x,z)D(z)P(dz) \leq 0. \end{cases}$$

From another perspective, Lin and Jeon (2006) discussed Random Forest from a point of view of K-Nearest Neighbor (KNN). To show the connection between Random Forest and KNN, they proposed a new method called Potential Nearest Neighbor (PNN). They also showed that Random Forest could be converted to an adaptive kernel smooth method described by PNN.

To sum up, Random Forest not only combines a large number of decision trees to reduce the variance of prediction like bagging, but also decreases the dependence among

decision trees via random feature projection to get a much lower prediction error than Bagging Decision Tree. Theoretically, Random Forest makes prediction via constructing an adaptive kernel function. That is very similar to other local methods such as non-parametric kernel method and KNN. Figure 4.11 illustrates the difference between Decision Tree and KNN.

4.5 Recent Developments of Random Forest

As one of the most effective ensemble method in solving real-world issues, the random forest also has many variants for different modeling tasks in statistics, data mining, and econometrics literature. In this section, we introduce several attractive variants of Random Forest.

4.5.1 Extremely randomized trees

For Bagging method, we discussed its effectiveness related to the variance of ensemble model. According to Equation (4.19), we have

$$Var(g) = \rho\sigma^2 + \frac{(1-\rho)}{B}\sigma^2,$$

the variance is decomposed into two parts: the first term $\rho\sigma^2$ depends on the correlation among base models and the second term $\frac{(1-\rho)}{B}\sigma^2$ is related to the number of base models.

Since we often combine a large number of base models, we can assume B goes to infinity and the main part of the variance converges to $\rho\sigma^2$. Thus, bagging can largely

decrease the second term.

Random Forest, besides controlling the second term via Bagging, also controls the first term by decreasing the ρ via random feature projection simultaneously. Because of the random feature projection, the decision tree suffers from a higher bias. It means that we need to focus on decreasing correlations among decision trees such that the ensemble model becomes more effective.

Random feature projection is not the only way to decrease the correlations. Geurts, Pierre, Ernst and Wehenkel (2006) introduced another way to achieve the goal and derived a new technique called the Extremely Randomized Trees (Extra-Trees). Compare to Random Forest, Extra-Trees works on the original samples instead of bootstrap samples. More importantly, Extra-Trees method generates the base decision tree via a more random way to split sample space than the random feature projection in Random Forest.

The Extra-Trees splitting algorithm is the following:

- To a node space in decision tree, first choose K variables $(x_1, x_2, ..., x_K)$ among all the p variables.
- To all K attributes, randomly choose a splitting point to each one of them via choosing a uniform number from (x_{min}, x_{max}) belong to this node.
- Compare the criteria among all the random splitting point and choose the attribute x_k giving the best splitting outcome.
- Choose variable x_k and the random splitting point as the final decision stump in this node.

Table 4.1: A Summary of Three Ensemble Methods

Names	Main Part of Variance	Bootstrap	Hyper-parameters
Bagging Decision Trees	$\rho\sigma^2$	Yes	B, n_{min}
Random Forest	$\rho\sigma^2$	Yes	B, m, n_{min}
Extremely Randomized Trees	$\frac{(1-\rho)}{B}\sigma^2$	No	B, K, n_{min}

- Stop splitting when the number of sample points = n_{min}.

Practically, we set $K = \sqrt{p}$ and $n_{min} = 5$ by default. But we can tune them based on the cross-validation.

The key difference of constructing base decision trees between Random Forest and Extra-Trees is the splitting rule for each node. In Random Forest, we choose m variables and then find the optimal decision stump directly. But in Extra-Trees, we choose K variables to randomly generate decision stump and then choose the 'optimal' decision stump. As a consequence, randomly growing decision trees in extra-trees will be less dependent than the trees in Random Forest, which leads to lower correlations ρ. Thus, even though Extra-Trees do not introduce Bootstrap, it works well in many data mining and predicting tasks. This idea about being 'random' is also used in many other machine learning algorithms such as Extreme Learning Machine proposed by Huang, Zhu and Siew (2006) and Echo State Networks designed by Jaeger (2001).

We summarize Bagging Decision Trees, Random Forest and Extremely Randomized Trees in Table 4.1.

4.5.2 Soft decision tree and forest

Based on the previous discussion, we find that Random Forest and its variants are based on the decision tree. The decision tree is growing via splitting the space into optimal sub-spaces recursively and the function defined by a decision tree is a non-smooth step function. The decision tree is naturally suitable for implementing the classification problem because of the discrete outputs. Since most economic problems are related to the regression problems, we could expect that the decision tree should be so large that it can handle a smooth function 'non-smoothly'.

To resolve this problem, we can consider a 'soft' decision tree instead of the 'hard' decision tree. Given a decision tree with only one root node and two leaf nodes, it can have two possible outcomes

$$f(x) = \begin{cases} \mu_1 & if\ g(x) > 0 \\ \\ \mu_2 & if\ g(x) < 0, \end{cases}$$

where μ_1 and μ_2 are correspond to the first and second leaf nodes. $g(x)$ is called gate function. It decides which leaf node should be selected. We can also rewrite the formula based on an indicator function

$$f(x) = \mu_1 \times I(g(x) > 0) + \mu_2 \times (1 - I(g(x) > 0)) .$$

For example, in women wage case we have discussed, the decision stump is $D : Expr = 10$. Given that, we can use a gate function $g(Expr) = Expr - 10$ to represent the decision stump

$$f(Expr) = \mu_1 \times I(g(Expr) > 0) + \mu_2 \times (1 - I(g(Expr) > 0)) . \tag{4.29}$$

That is, if $Expr > 10$, we choose the first leaf node and $Expr < 10$ choose the second one.

Generally, to the mth node, we can use a similar function to represent its output

$$F_m(x) = F_m^L(x) \times I(g_m(x) > 0) + F_m^R(x) \times (1 - I(g_m(x) > 0)). \tag{4.30}$$

If $F_m^L(x)$ and $F_m^R(x)$ are leaf nodes, we have $F_m^L(x) = \mu_L$ and $F_m^R(x) = \mu_R$. If not, they are corresponding to the child-nodes in the next layer $F_m^L(x) = F_{m+1}^L(x)$. Because of the indicator function, the $F_m(x)$ is a step function with two outcomes, $F_m^L(x)$ or $F_m^R(x)$. It is a hard decision tree.

In Equation (4.30), we can use a smooth gate function instead of the identity function such that the decision tree is 'soft' and $F_m(x)$ is a smooth function. Let us change the indicator function $I(h)$ to a logistic function $L(h)$, we have

$$F_m(x) = F_m^L(x) \times L(g_m(x)) + F_m^R(x) \times (1 - L(g_m(x))), \tag{4.31}$$

where $L(h) = \frac{1}{1+e^{-h}}$ is a logistic function and $g_m(x) = \beta^T x$ is a linear single index function of input variables. In the soft decision tree, instead of selecting one from two child nodes, a smooth $F_m(x)$ is taking weighed average between $F_m^L(x)$ and $F_m^R(x)$. In Figure 4.12, we compare the hard decision tree with the soft decision tree.

Back to the women's wage example, we choose $L(g(Expr)) = \frac{1}{1+e^{-(Expr-10)}}$. That is, if $Expr > 10$, we consider the left node more and consider the right node more when

$Expr < 10$.

Compared to the hard decision tree, the soft decision tree has many advantages:

- Since soft decision tree can represent any smooth function, it is more suitable to handle the regression problem than the original decision tree. That may be the most important advantage since economic research often cares more about the regression problem, such as economic growth rate prediction and derivative estimation for partial effect analysis.

- Soft decision tree contains a bunch of differentiable gate functions, which means we can train all the parameters via the Expectation Maximization (EM) method very quickly.

- In all the leaf nodes of a soft decision tree, we could not only choose a constant μ, but consider more flexible methods, like the linear formula or even the neural networks.

- Because of its hierarchical structure, the soft decision tree is a local method as the hard decision tree. Thus, it has similar theories and properties as other local methods like kernel regression.

There are many research papers related to the soft version of the decision tree. This first soft decision tree model is called Hierarchical Mixtures of Experts (HME) discussed by Jordan and Jacob (1994). Instead of growing a decision tree via splitting recursively, in the HME method, we first designate the structure of a soft decision tree, like the number of layers, then optimize all the parameters in this tree.

Consider a soft decision tree with S layers and one split in each node. Thus, the number of total leaf nodes is 2^S and the function of this soft decision tree is

$$\begin{aligned} f_{HME}(x) &= \sum_{leaf=1}^{2^S} P_{leaf}(x)\mu_{leaf} \\ &= \sum_{leaf=1}^{2^S} \prod_{p \to leaf} G_p(x)\mu_{leaf}, \end{aligned}$$

where $p \to leaf$ means all the gate functions contained in the nodes located on the path to the sth leaf node. According to Equation (4.31), we have

$$G_p(x) = I(p = left) \times L\left(g_p(x)\right) + I(p = right) \times \left(1 - L\left(g_p(x)\right)\right)$$

It decides the gate function for each node on the path. μ_{leaf} represents the function in each leaf, which could be a constant, a simple linear function or other nonlinear models.

For example, Figure 4.13 shows the structure of an HME with two layers. Let us consider the path to the first leaf node $p \to 1$. The path starts from the root node in layer 0. Since the path chooses the left node, the node0, $I(p = left) = 1$ and $G_0(x)$ should be

$$\begin{aligned} G_0(x) &= I(0 = left) \times L(g_0(x)) + (1 - I(0 = left)) \times (1 - L(g_0(x))) \\ &= L(g_0(x)). \end{aligned}$$

Then, the path contains the node1 at layer 1 and then choose the left node, the leaf1. Thus, $G_1(x)$ should be

$$G_1(x) = I(1 = left) \times L(g_{00}(x)) + (1 - I(1 = left)) \times (1 - L(g_{00}(x)))$$

$$= L(g_{00}(x)).$$

Thus, to $P_{leaf=1}(x)$, we have

$$\begin{aligned} P_1(x) &= \prod_{p \to 1} G_p(x) = G_0(x) \times G_1(x) \\ &= L(g_0(x)) \times L(g_{00}(x)). \end{aligned} \tag{4.32}$$

Similarly, to the path to leaf 2, we have

$$\begin{aligned} P_2(x) &= \prod_{p \to 2} G_p(x) = G_0(x) \times G_1(x) \\ &= L(g_0(x)) \times (1 - L(g_{00}(x))). \end{aligned} \tag{4.33}$$

Now we find that HME is similar to a mixture model since $\sum_{leaf} P_{leaf}(x) = 1$. Suppose the μ_{leaf} is a parameter, like mean, of a distribution $P_{leaf}(y|x)$. Then we have the conditional probability of y given x

$$\begin{aligned} P(y|x) &= \sum_{leaf=1}^{2^S} \prod_{p \to leaf} G_p(x) P_{leaf}(y|x) \\ &= \sum_{leaf=1}^{2^S} P_{leaf}(x) P_{leaf}(y|x). \end{aligned}$$

Thus, we can have the log likelihood function of HME with unknown parameter β

$$\begin{aligned} L(y|x;\beta) &= \sum_{i=1}^{N} \log P(y_i|x_i;\beta) \\ &= \sum_{i=1}^{N} \log \sum_{leaf=1}^{2^S} P_{leaf}(x_i;\beta) P_{leaf}(y_i|x_i;\beta). \end{aligned}$$

To optimize the likelihood function, [90] considered a the Expectation-Maximization (EM) to optimize it. The main idea behind EM is based on the so-called complete log likelihood function

$$L_c(y|x;\beta) = \sum_{i=1}^{N} \sum_{leaf=1}^{2^S} z_{leaf} \prod_{p \to leaf} G_p(x_i;\beta) P_{leaf}(y_i|x_i;\beta),$$

where z_{leaf} are implicit variables that represent the indicators of leaf nodes. Take the expectation of $L_c(y|x;\beta)$, we have

$$Q(y|x;\beta) = E_z(L_c(y|x;\beta)) = \sum_{i=1}^{N} \sum_{leaf=1}^{2^S} E(z_{leaf}) \prod_{p \to leaf} G_p(x_i;\beta) P_{leaf}(y_i|x_i;\beta).$$

To $E(z_{leaf})$, we have

$$\begin{aligned} E(z_{leaf}) &= P(z_{leaf} = 1|y, x, \beta) \\ &= \frac{P(y|z_{leaf} = 1, x, \beta) P(z_{leaf} = 1|x, \beta)}{P(y|x, \beta)} \\ &= \frac{\prod_{p \to leaf} g_p(x;\beta) P(y|x,\beta)}{\sum_{leaf=1}^{S^2} \prod_{\to leaf} g_p(x;\beta) P(y|x,\beta)}. \end{aligned}$$

We can see that $Q(y|x;\beta)$ is the lower bound of $L(y|x;\beta)$ because of Jensen's Inequality. The log-likelihood function $L(y|x)$ is optimized if we can optimize the lower bound $Q(y|x;\beta)$. This is the key to the EM method.

To sum up, the training procedure for HME is as follows:

- Randomly initializes all the parameters β, then propagate forward the input to get the distribution of x_i.
- To each mini-batch, propagate forward all the x to the leaves to get the predicted outputs. Then compute all the $E(z_{i,leaf})$ (E-step).

- Optimize the expectation likelihood function $Q(x, y; \beta)$ (M-step).

- Redo E-step and M-Step until that all the parameters converge.

One possible drawback to the soft decision tree method is that the HME could lead to a long-time training process. More importantly, since HME needs a pre-determined structure of a soft decision tree, it is not adaptive to data. To resolve this issue, another way to implement soft decision trees was discussed by Irsoy, Yldz and Alpaydn (2012). The authors introduced a new way to grow a soft decision tree. In each node, they used gradient descent to find the optimal splitting line then compare the predicting outcome between the two trees with and without the new splitting line to decide that this new node should be added or not. Thus, this method can adaptively learn the structure of soft decision tree and could be faster. Similarly to Random Forest, Yıldız,Írsoy and Alpaydın (2016) constructed an ensemble of soft decision trees via Bagging to explore the ensemble of soft decision trees.

4.6 Applications of Bagging and Random Forest in Economics

4.6.1 Bagging in economics

Recently, Bootstrap Aggregating is widely used in macroeconomic analysis and forecasting. Panagiotelis, Athanasopoulos, Hyndman, Jiang and Vahid (2019) explored the performance of the ensemble a large number of predictors in predicting macroeconomic series data in Australia. Precisely, they compared Bagging LARS with Dynamic Factor Model, Ridge Regression, LARS, and Bayesian VAR respectively on GDP growth, CPI inflation and IBR (the interbank overnight cash rate equivalent to the Federal funds rate

in the US). They found that Bagging method can help in more accurate forecasting.

As discussed in this chapter, Bagging has been proved to be effective to improve on unstable forecast. Theoretical and empirical works using classification, regression trees, variable selection in linear and non-linear regression have shown that bagging can generate substantial prediction gain. However, most of the existing literature on bagging has been limited to the cross sectional circumstances with symmetric cost functions. Lee and Yang (2006) extend the application of bagging to time series settings with asymmetric cost functions, particularly for predicting signs and quantiles. They use quantile predictions to construct a binary predictor and the majority-voted bagging binary prediction, and show that bagging may improve the binary prediction. For empirical application, they presented results using monthly S&P500 and NASDAQ stock index returns.

Inoue and Kilian (2008) considered the Bagging method in forecasting economic time series of US CPI data. They explored how the Bagging may be adapted to application involving dynamic linear multiple regression for the inflation forecasting. And then they compare several models' performances, including correlated regressor models, factor models and shrinkage estimation of regressor models (with LASSO) with or without Bagging. Their empirical evidence showed that Bagging can achieve large reductions in prediction mean squared error, even in challenging applications such as inflation forecasting.

Lee, Tu and Ullah (2014), Lee, Tu and Ullah (2015) and Hillebrand, Lee and Medeiros (2014) consider parametric, nonparametric, and semiparametric predictive regression models for financial returns subject to various hard-thresholding constraints using indicator functions. The purpose is to incorporate various economic constraints that are

implied from economic theory or common priors such as monotonicity or positivity of the regression functions. They use bagging to smooth the hard-thresholding constraints to reduce the variance of the estimators. They show the usefulness of bagging when such economic constraints are imposed in estimation and forecasting, by deriving asymptotic properties of the bagging constrained estimators and forecasts. The advantages of the bagging constrained estimators and forecasts are also demonstrated by extensive Monte Carlo simulations. Applications to predicting financial equity premium are taken for empirical illustrations, which show imposing constraints and bagging can mitigate the chance of making large size forecast errors and bagging can make these constrained forecasts even more robust.

Jin, Su and Ullah (2014) propose a revised version of bagging as a forecast combination method for the out-of-sample forecasts in time series models. The revised version explicitly takes into account the dependence in time series data and can be used to justify the validity of bagging in the reduction of mean squared forecast error when compared with the unbagged forecasts. Their Monte Carlo simulations show that their method works quite well and outperforms the traditional one-step-ahead linear forecast as well as the nonparametric forecast in general, especially when the in-sample estimation period is small. They also find that the bagging forecasts based on misspecified linear models may work as effectively as those based on nonparametric models, suggesting the robustification property of bagging method in terms of out-of-sample forecasts. They then re-examine forecasting powers of predictive variables suggested in the literature to forecast the excess returns or equity premium and find that, consistent with Welch and Goyal (2008), the historical aver-

age excess stock return forecasts may beat other predictor variables in the literature when they apply traditional one-step linear forecast and the nonparametric forecasting methods. However, when using the bagging method or the revised version, which help to improve the mean squared forecast error for unstable predictors, the predictive variables have a better forecasting power than the historical average excess stock return forecasts.

Audrino and Medeiros (2011) proposed a new method called smooth transition tree. They found that the leading indicators for inflation and real activity are the most relevant predictors in characterizing the multiple regimes' structure. They also provided empirical evidence of the model in forecasting the first two conditional moments when it is used in connection with Bagging.

Hirano and Wright (2017) considered forecasting with uncertainty about the choice of predictor variables and compare the performances of model selection methods under Rao-Blackwell theorem and Bagging respectively. They investigated the distributional properties of a number of different schemes for model choice and parameter estimation: in-sample model selection using the Akaike Information Criterion, out-of-sample model selection, and splitting the data into sub-samples for model selection and parameter estimation. They examined how Bagging affected the local asymptotic risk of the estimators and their associated forecasts. In their numerical study, they found that for many values of the local parameter, the out-of-sample and split-sample schemes performed poorly if implemented in a conventional way. But they performed well if implemented in conjunction with model selection methods under Rao-Blackwell theorem or Bagging.

4.6.2 Random forest in economics

To introduce Random Forest into economic research, many economic and statistic researchers studied in extending the theory of random forest not only for forecasting but for inference.

In the literature of economic inference, Biau, Devroye and Gabor (2008) discussed the consistency of Random Forest in the context of additive regression models, which sheds light on the forest-based statistical inference. Wager and Athey (2008) studied in the application of random forest in economic research. They proposed the Causal Forest, an unbiased random forest method for estimating and testing the heterogeneous treatment effect. They first showed that classic Random Forest cannot have unbiasedness because of Bagging. Then, they proposed the Causal Forest which combines a bunch of unbiased Honest Tree based on Sub-sampling aggregating. They also showed that Causal Forest is unbiased and has asymptotic normality under some assumptions. Finally, they discussed the importance and advantage of Causal Forest in applications to economic causal inference.

To the application of economic forecasting, Hothorn and Zeileis (2017) discussed a new Random Forest method, the Transformation Forest. Based on a parametric family of distributions characterized by their transformation function, they proposed a dedicated novel transformation tree and transformation forest as an adaptive local likelihood estimator of conditional distribution functions, which are available for inference procedures. In macroeconomic forecasting, Random Forest is applied in Euro area GDP forecasting [see Biau and D'Elia (2011)] and financial volatility forecasting [see Luong and Dokuchaev (2018)]. Finally, Fischer, Krauss and Treichel (2018) assess and compare the time series

forecasting performance of several machine learning algorithms such as Gradient Boosting Decision Trees, Neural Networks, Logistic Regression, Random Forest and so on in a simulation study. Nyman and Ormerod (2016) explore the potential of Random Forest for forecasting the economic recession on the quarterly data over 1970Q2 to 1990Q2.

4.7 Conclusion

In this chapter, we discuss the Bagging method and Random Forest. In Section 4.2 we begin with introducing Bagging and its variants, the Subbaging and Bragging. In Section 4.3, we introduce Decision Tree, which provides the foundation of the Random Forest. In Section 4.4, we introduce the related theories about Random Forest and its important variants like Extreme Random Trees and Soft Decision Tree. In Section 4.5 and 4.6 we discussed many applications of Bagging and Random Forest in macroeconomic forecasting and economic causal inference.

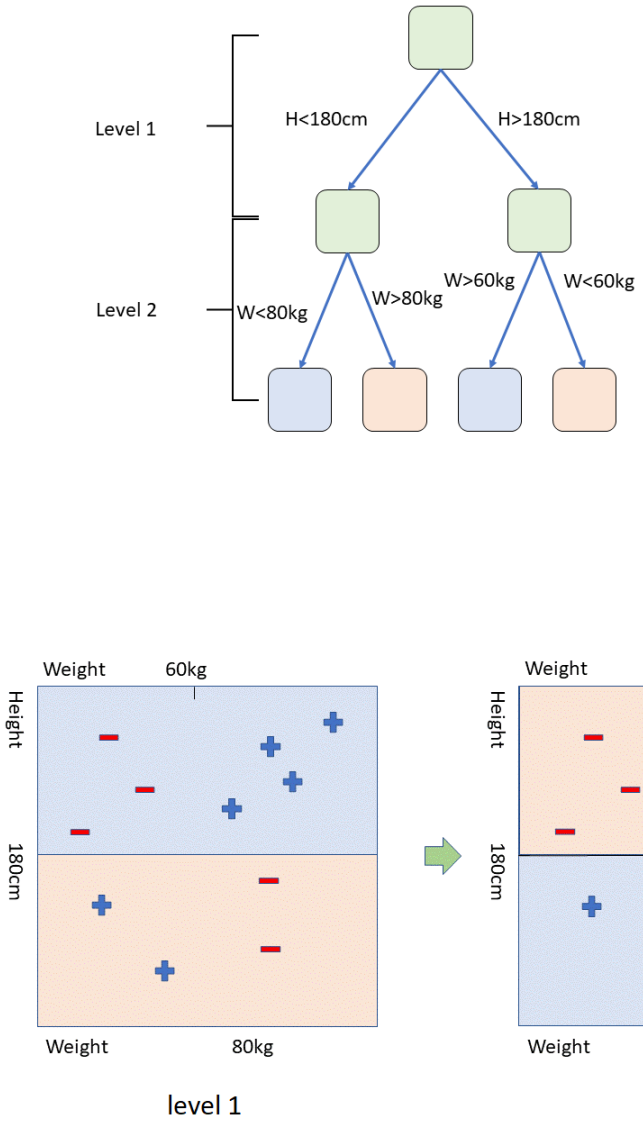

Figure 4.6: Grow a Tree for Health Data

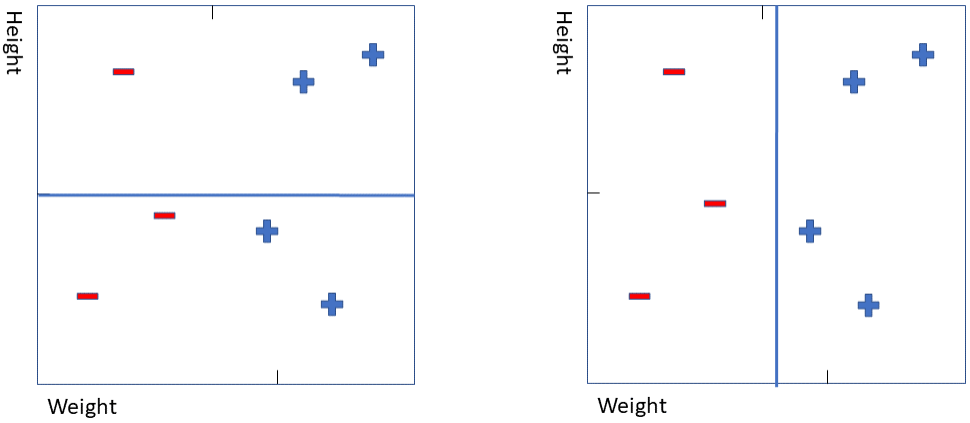

Figure 4.7: Sub-spaces Generated by Decision Stumps

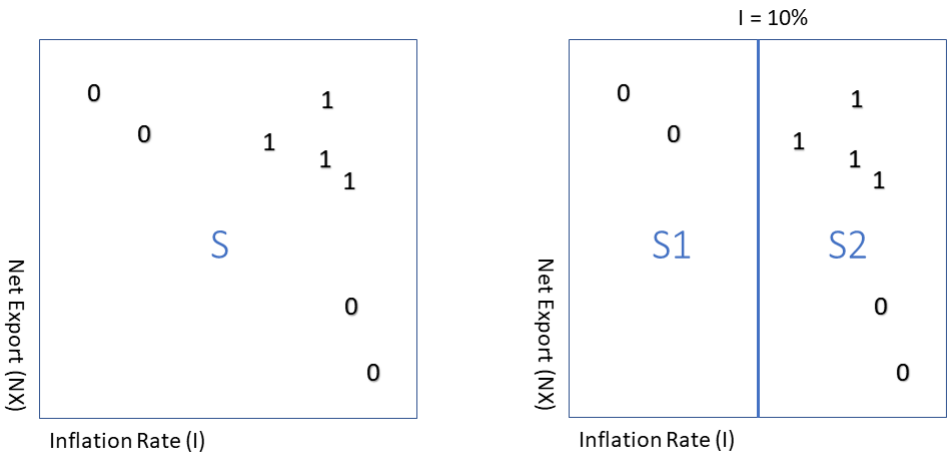

Figure 4.8: Plots for Economic Growth Data

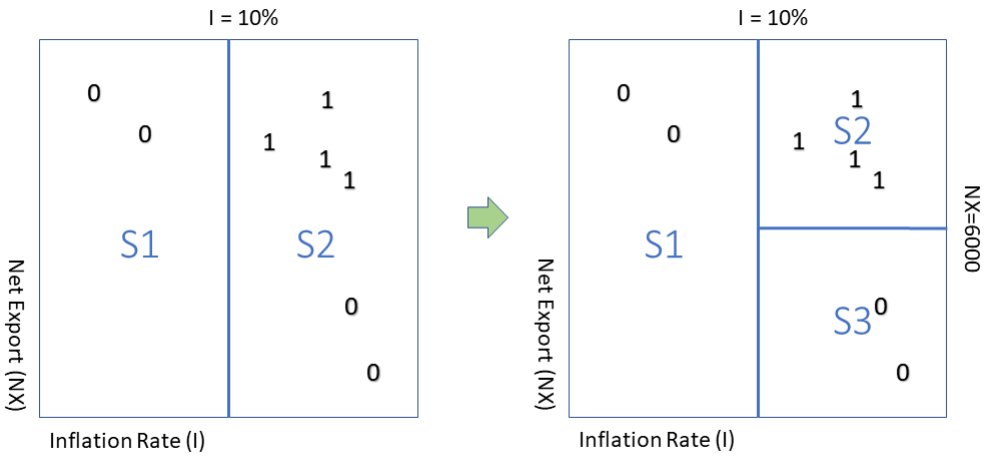

Figure 4.9: Grow a Tree for Economic Growth Prediction

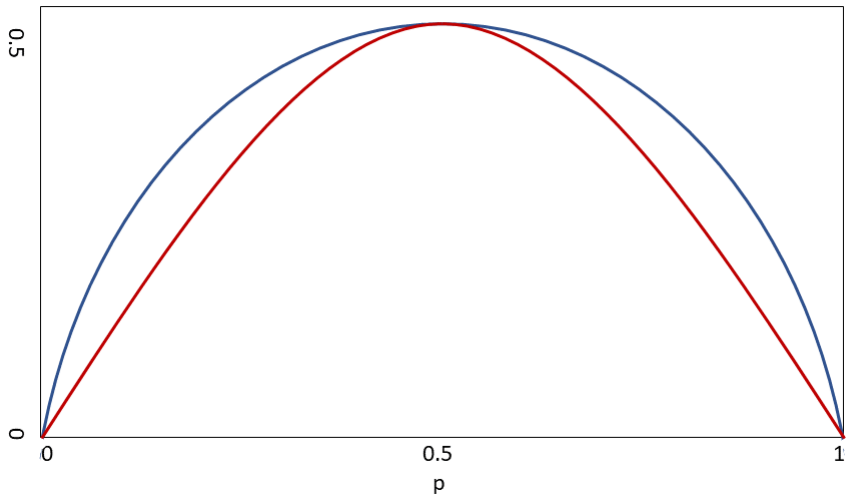

Figure 4.10: Entropy (blue) and Gini Impurity (red)

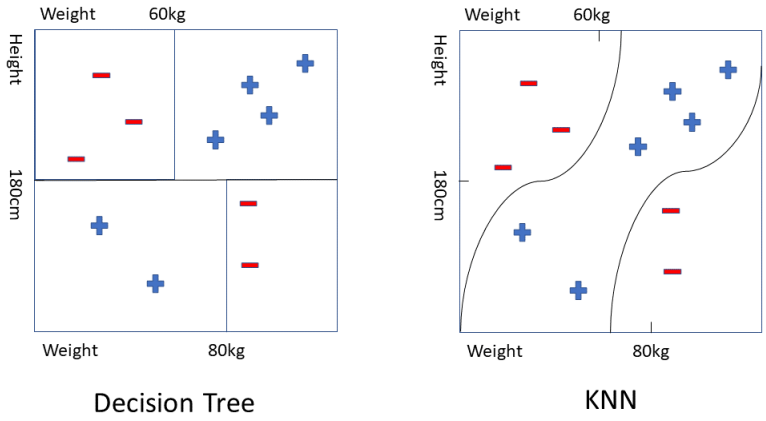

Figure 4.11: 2D Plot of Decision Tree and KNN

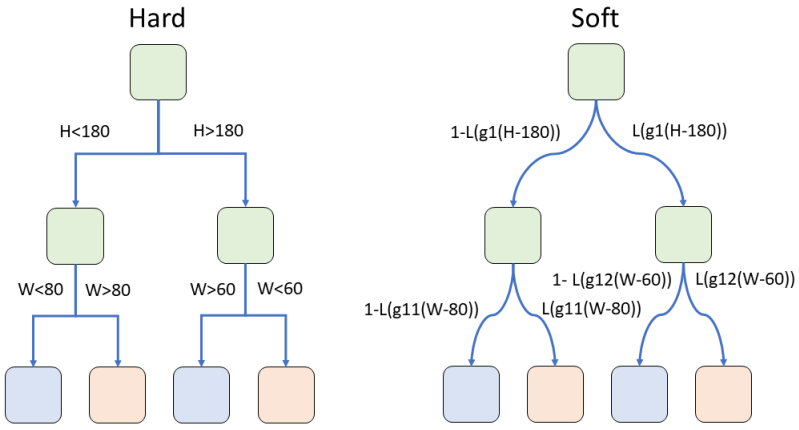

Figure 4.12: Hard Decision Tree and Soft Decision Tree

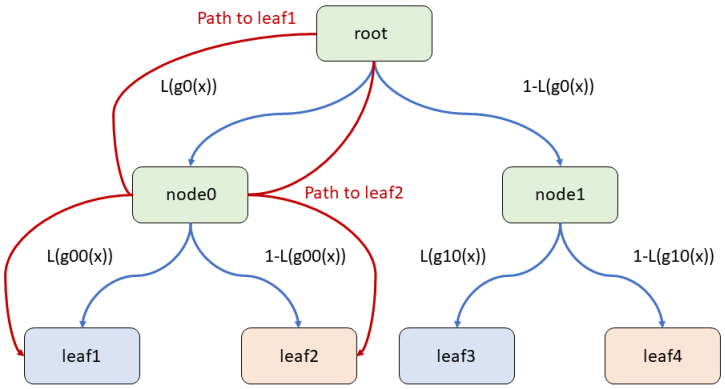

Figure 4.13: Hierarchical Mixtures of Experts

Chapter 5

Boosting in the Economic Forecasting

5.1 Introduction

The term Boosting originates from the so-called hypothesis boosting problem in the distribution-free or probably approximately correct model of learning. In this model, the learner produces a classifier based on random samples from an unknown data generating process. Samples are chosen according to a fixed but unknown and arbitrary distribution on the population. The learner's task is to find a classifier that correctly classifies new samples from the data generating process as positive or negative examples. A weak learner produces classifiers that perform only slightly better than random guessing. A strong learner, on the other hand, produces classifiers that can achieve arbitrarily high accuracy given enough samples from the data generating process.

In a seminal paper, Schapire (1990) addresses the problem of improving the accuracy of a class of classifiers that perform only slightly better than random guessing. The paper shows the existence of a weak learner implies the existence of a strong learner and vice versa. A boosting algorithm is then proposed to convert a weak learner into a strong learner. The algorithm uses *filtering* to modify the distribution of samples in such a way as to force the weak learning algorithm to focus on the harder-to-learn parts of the distribution.

Not long after the relation between weak learners and strong learners are revealed, Freund and Schapire (1997) propose the Adaptive Boost (AdaBoost) for binary classification. AdaBoost performs incredibly well in practice and stimulates the invention of boosting algorithms for multi-class classifications. On the other hand, researchers try to explain the success of AdaBoost in a more theoretical way, e.g. Friedman, Hastie and Tibshirani (2000), Bartlett, Jordan and McAuliffe (2006), Bartlett and Traskin (2007). Further understanding of the theory behind the success of boosting algorithms in turn triggers a bloom of Boosting algorithm with better statistical properties, e.g. Friedman (2001), Bühlmann and Yu (2003), Mease, Wyner and Buja (2007).

Boosting is undoubtedly the most popular machine learning algorithm in the online data science platform such as Kaggle. It is efficient and easy to implement. There are numerous packages in Python and R which implement Boosting algorithms in one way or another, e.g. *XBoost*. In the following sections, we will introduce the AdaBoost as well as other Boosting algorithms in detail together with examples to help the readers better understand the algorithms and statistical properties of the Boosting methods.

This chapter is organized as follows. Section 5.1 provides an overview on the

origination and development of Boosting. Sections 5.2 and 5.3 are an introduction of AdaBoost which is the first practically feasible Boosting algorithm with its variants. Section 5.4 introduces a Boosting algorithm for linear regressions, namely L_2Boosting. Section 5.5 gives a generalization of the above mentioned algorithms which is called Gradient Boosting Machine. Section 5.6 gives more variants of Boosting, e.g. Boosting for nonlinear models. Section 5.7 provides applications of the Boosting algorithms in macroeconomic studies.

5.2 AdaBoost

The first widely used Boosting algorithm is AdaBoost which solves binary classification problems with great success. A large number of important variables in economics are binary. For example, whether the economy is going into expansion or recession, whether an individual is participating in the labor force, whether a bond is going to default, and etc. Let

$$\pi(\mathbf{x}) \equiv \Pr(y = 1|\mathbf{x}),$$

so that y takes value 1 with probability $\pi(\mathbf{x})$ and -1 with probability $1 - \pi(\mathbf{x})$. The goal of the researchers is to predict the unknown value of y given known information on $\mathbf{x}$.

5.2.1 AdaBoost algorithm

This section introduces the AdaBoost algorithm of Freund and Schapire (1997). The algorithm of AdaBoost is shown in Algorithm 12.

Let y be the binary class taking a value in $\{-1, 1\}$ that we wish to predict. Let $f_m(\mathbf{x})$ be the weak learner (weak classifier) for the binary target y that we fit to predict

using the high-dimensional covariates $\mathbf{x}$ in the mth iteration. Let err_m denote the error rate of the weak learner $f_m(\mathbf{x})$, and $E_w(\cdot)$ denote the weighted expectation (to be defined below) of the variable in the parenthesis with weight w. Note that the error rate $E_w\left[1_{(y\neq f_m(\mathbf{x}))}\right]$ is estimated by $err_m = \sum_{i=1}^{n} w_i 1_{(y_i\neq f_m(x_i))}$ with the weight w_i given by step 2(d) from the previous iteration. n is the number of observations. The symbol $1_{(\cdot)}$ is the indicator function which takes the value 1 if a logical condition inside the parenthesis is satisfied and takes the value 0 otherwise. The symbol $\text{sign}(z) = 1$ if $z > 0$, $\text{sign}(z) = -1$ if $z < 0$, and hence $\text{sign}(z) = 1_{(z>0)} - 1_{(z<0)}$.

Algorithm 12 *1. Start with weights* $w_i = \frac{1}{n}, i = 1, \ldots, n$.

2. For $m = 1$ *to* M

(a) For $j = 1$ *to* k *(for each variable)*

i. Fit the classifier $f_{mj}(x_{ij}) \in \{-1, 1\}$ *using weights* w_i *on the training data.*

ii. Compute $err_{mj} = \sum_{i=1}^{n} w_i 1_{(y_i\neq f_{mj}(x_{ji}))}$.

(b) Find $\hat{j}_m = \arg\min_j err_{mj}$

(c) Compute $c_m = \log\left(\frac{1-err_{m,\hat{j}_m}}{err_{m,\hat{j}_m}}\right)$.

(d) Set $w_i \leftarrow w_i \exp[c_m 1_{(y_i\neq f_{m,\hat{j}_m}(x_{\hat{j}_m,i}))}]$, $i = 1, \ldots, n$, *and normalize so that* $\sum_{i=1}^{n} w_i = 1$.

3. Output the binary classifier $\text{sign}[F_M(\mathbf{x})]$ *and the class probability prediction* $\hat{\pi}(\mathbf{x}) = \frac{e^{F_M(\mathbf{x})}}{e^{F_M(\mathbf{x})}+e^{-F_M(\mathbf{x})}}$ *where* $F_M(\mathbf{x}) = \sum_{m=1}^{M} c_m f_{m,\hat{j}_m}(x_{\hat{j}_m})$.

Note that the presented version of Discrete AdaBoost in Algorithm 12 as well as Real AdaBoost (RAB), LogitBoost (LB) and Gentle AdaBoost (GAB) which will be

introduced later in the next section are different from their original version when they were first introduced. The original version of these algorithms only output the class label. In this chapter, we follow the idea of Mease, Wyner and Buja (2007) and modified the algorithms to output both the class label and the probability prediction. The probability prediction is attained using

$$\hat{\pi}(\mathbf{x}) = \frac{e^{F_M(\mathbf{x})}}{e^{F_M(\mathbf{x})} + e^{-F_M(\mathbf{x})}},$$

where $F_M(\mathbf{x})$ is the sum of weak learners in the algorithms.

The only hyperparameter, i.e. the user specified parameter, in the AdaBoost as well as other Boosting algorithms is the number of iterations, M. It is also known as the stopping rule and is commonly chosen by cross-validation as well as information criterion such as AICc in Bühlmann and Yu (2003). The choice of the stopping rule is embedded in most implementation of AdaBoost and should not be a concern for most users. Interesting readers could check Hastie, Tibshirani and Friedman (2009) for more details of cross-validation.

The most widely used weak learner is the classification tree. The simplest classification tree, the stump, takes the following functional form

$$f(x_j, a) = \begin{cases} 1 & x_j > a \\ -1 & x_j < a, \end{cases}$$

where the parameter a is found by minimizing the error rate

$$\min_a \sum_{i=1}^{n} w_i 1\left(y_i \neq f\left(x_{ji}, a\right)\right).$$

The other functional form of the stump can be shown as exchanging the greater and smaller

sign in the previous from

$$f(x_j, a) = \begin{cases} 1 & x_j < a \\ -1 & x_j > a, \end{cases}$$

where the parameter a is found by minimizing the same error rate.

5.2.2 AdaBoost: statistical view

After AdaBoost is invented and shown to be successful, numerous papers have attempted to explain the effectiveness of the AdaBoost algorithm. In an influential paper, Friedman, Hastie and Tibshirani (2000) show that AdaBoost builds an additive logistic regression model

$$F_M(\mathbf{x}) = \sum_{m=1}^{M} c_m f_m(\mathbf{x})$$

via Newton-like updates for minimizing the exponential loss

$$J(F) = E\left(e^{-yF(\mathbf{x})} \middle| \mathbf{x}\right).$$

We hereby show the above statement using the greedy method to minimize the exponential loss function iteratively as in Friedman, Hastie and Tibshirani (2000).

After m iterations, the current classifier is denoted as $F_m(\mathbf{x}) = \sum_{s=1}^{m} c_s f_s(\mathbf{x})$. In the next iteration, we are seeking an update $c_{m+1} f_{m+1}(\mathbf{x})$ for the function fitted from previous iterations $F_m(\mathbf{x})$. The updated classifier would take the form

$$F_{m+1}(\mathbf{x}) = F_m(\mathbf{x}) + c_{m+1} f_{m+1}(\mathbf{x}).$$

The loss for $F_{m+1}(\mathbf{x})$ will be

$$\begin{aligned} J(F_{m+1}(\mathbf{x})) &= J(F_m(\mathbf{x}) + c_{m+1} f_{m+1}(\mathbf{x})) \\ &= E\left[e^{-y(F_m(\mathbf{x}) + c_{m+1} f_{m+1}(\mathbf{x}))}\right]. \end{aligned} \tag{5.1}$$

Expand w.r.t. $f_{m+1}(\mathbf{x})$

$$\begin{aligned} J(F_{m+1}(\mathbf{x})) &\approx E\left[e^{-yF_m(\mathbf{x})}\left(1 - yc_{m+1} f_{m+1}(\mathbf{x}) + \frac{y^2 c_{m+1}^2 f_{m+1}^2(\mathbf{x})}{2}\right)\right] \\ &= E\left[e^{-yF_m(\mathbf{x})}\left(1 - yc_{m+1} f_{m+1}(\mathbf{x}) + \frac{c_{m+1}^2}{2}\right)\right]. \end{aligned}$$

The last equality holds since $y \in \{-1, 1\}$, $f_{m+1}(\mathbf{x}) \in \{-1, 1\}$, and $y^2 = f_{m+1}^2(\mathbf{x}) = 1$. $f_{m+1}(\mathbf{x})$ only appears in the second term in the parenthesis, so minimizing the loss function (5.1) w.r.t. $f_{m+1}(\mathbf{x})$ is equivalent to maximizing the second term in the parenthesis which results in the following conditional expectation

$$\max_f E\left[e^{-yF_m(\mathbf{x})} y c_{m+1} f_{m+1}(\mathbf{x}) | \mathbf{x}\right].$$

For any $c > 0$ (we will prove this later), we can omit c_{m+1} in the above objective function

$$\max_f E\left[e^{-yF_m(\mathbf{x})} y f_{m+1}(\mathbf{x}) | \mathbf{x}\right].$$

To compare it with the Discrete AdaBoost algorithm, here we define weight $w = w(y, \mathbf{x}) = e^{-yF_m(\mathbf{x})}$. Later we will see that this weight w is equivalent to that shown in the Discrete AdaBoost algorithm. So the above optimization can be seen as maximizing a weighted conditional expectation

$$\max_f E_w\left[y f_{m+1}(\mathbf{x}) | \mathbf{x}\right], \tag{5.2}$$

where $E_w(y|\mathbf{x}) := \frac{E(wy|\mathbf{x})}{E(w|\mathbf{x})}$ refers to a weighted conditional expectation. Note that (5.2) can be re-written as

$$\begin{aligned} & E_w\left[y f_{m+1}(\mathbf{x})\,|\mathbf{x}\right] \\ = \; & P_w(y=1|\mathbf{x})\, f_{m+1}(\mathbf{x}) - P_w(y=-1|\mathbf{x})\, f_{m+1}(\mathbf{x}) \\ = \; & \left[P_w(y=1|\mathbf{x}) - P_w(y=-1|\mathbf{x})\right] f_{m+1}(\mathbf{x}) \\ = \; & E_w(y|\mathbf{x})\, f_{m+1}(\mathbf{x}), \end{aligned}$$

where $P_w(y|\mathbf{x}) = \frac{E(w|y,\mathbf{x})P(y|\mathbf{x})}{E(w|\mathbf{x})}$. Solve the maximization problem (5.2). Since $f_{m+1}(\mathbf{x})$ only takes 1 or -1, it should be positive whenever $E_w(y|\mathbf{x})$ is positive and -1 whenever $E_w(y|\mathbf{x})$ is negative. The solution for $f_{m+1}(\mathbf{x})$ is

$$f_{m+1}(\mathbf{x}) = \begin{cases} 1 & E_w(y|\mathbf{x}) > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Next, we minimize the loss function (5.1) w.r.t. c_{m+1}

$$c_{m+1} = \arg\min_{c_{m+1}} E_w\left(e^{-c_{m+1} y f_{m+1}(\mathbf{x})}\right),$$

where

$$E_w\left(e^{-c_{m+1} y f_{m+1}(\mathbf{x})}\right) = P_w\left(y = f_{m+1}(\mathbf{x})\right) e^{-c_{m+1}} + P_w\left(y \neq f_{m+1}(\mathbf{x})\right) e^{c_{m+1}}.$$

The first order condition is

$$\frac{\partial E_w\left(e^{-c y f_{m+1}(\mathbf{x})}\right)}{\partial c_{m+1}} = -P_w\left(y = f_{m+1}(\mathbf{x})\right) e^{-c_{m+1}} + P_w\left(y \neq f_{m+1}(\mathbf{x})\right) e^{c_{m+1}}.$$

Let

$$\frac{\partial E_w\left(e^{-c_{m+1} y f_{m+1}(\mathbf{x})}\right)}{\partial c_{m+1}} = 0,$$

and thus we have

$$P_w\left(y = f_{m+1}\left(\mathbf{x}\right)\right) e^{-c_{m+1}} = P_w\left(y \neq f_{m+1}\left(\mathbf{x}\right)\right) e^{c_{m+1}}.$$

Solving for c_{m+1}, we obtain

$$c_{m+1} = \frac{1}{2}\log\frac{P_w\left(y = f_{m+1}\left(\mathbf{x}\right)\right)}{P_w\left(y \neq f_{m+1}\left(\mathbf{x}\right)\right)} = \frac{1}{2}\log\left(\frac{1 - err_{m+1}}{err_{m+1}}\right),$$

where $err_{m+1} = P_w\left(y \neq f_{m+1}\left(\mathbf{x}\right)\right)$ is the error rate of $f_{m+1}\left(\mathbf{x}\right)$. Note that $c_{m+1} > 0$ as long as the error rate is smaller than 50%. Our assumption $c_{m+1} > 0$ holds for any learner that is better than random guessing.

Now we have finished the steps of one iteration and can get our updated classifier by

$$F_{m+1}\left(\mathbf{x}\right) \leftarrow F_m\left(\mathbf{x}\right) + \left(\frac{1}{2}\log\left(\frac{1 - err_{m+1}}{err_{m+1}}\right)\right) f_{m+1}\left(\mathbf{x}\right).$$

Note that in the next iteration, the weight we defined w_{m+1} will be

$$w_{m+1} = e^{-yF_{m+1}(\mathbf{x})} = e^{-y(F_m(\mathbf{x}) + c_{m+1}f_{m+1}(\mathbf{x}))} = w_m \times e^{-c_{m+1}f_{m+1}(\mathbf{x})y}.$$

Since $-yf_{m+1}\left(\mathbf{x}\right) = 2 \times 1_{\{y \neq f_{m+1}(\mathbf{x})\}} - 1$, the update is equivalent to

$$w_{m+1} = w_m \times e^{\left(\log\left(\frac{1-err_{m+1}}{err_{m+1}}\right)1_{[y \neq f_{m+1}(\mathbf{x})]}\right)} = w_m \times \left(\frac{1 - err_{m+1}}{err_{m+1}}\right)^{1_{[y \neq f_{m+1}(\mathbf{x})]}}.$$

Thus the function and weight update are of an identical form to those used in AdaBoost. AdaBoost could do better than any single weak classifier since it iteratively minimizes the loss function via a Newton-like procedure.

Interestingly, the function $F\left(\mathbf{x}\right)$ from minimizing the exponential loss is the same

as maximizing a logistic log-likelihood. Let

$$\begin{aligned} J\left(F\left(\mathbf{x}\right)\right) &= E\left[E\left(e^{-yF(\mathbf{x})}\middle|\,\mathbf{x}\right)\right] \\ &= E\left[P\left(y=1|\mathbf{x}\right)e^{-F(\mathbf{x})} + P\left(y=-1|\mathbf{x}\right)e^{F(\mathbf{x})}\right]. \end{aligned}$$

Taking derivative w.r.t. $F\left(\mathbf{x}\right)$ and making it equal to zero, we obtain

$$\frac{\partial E\left(e^{-yF(\mathbf{x})}|\mathbf{x}\right)}{\partial F\left(\mathbf{x}\right)} = -P\left(y=1|\mathbf{x}\right)e^{-F(\mathbf{x})} + P\left(y=-1|\mathbf{x}\right)e^{F(\mathbf{x})} = 0.$$

Therefore,

$$F^{*}\left(\mathbf{x}\right) = \frac{1}{2}\log\left[\frac{P\left(y=1|\mathbf{x}\right)}{P\left(y=-1|\mathbf{x}\right)}\right].$$

Moreover, if the true probability is

$$P\left(y=1|\mathbf{x}\right) = \frac{e^{2F(\mathbf{x})}}{1+e^{2F(\mathbf{x})}},$$

for $Y = \frac{y+1}{2}$, the log-likelihood is

$$E\left(\log L|\mathbf{x}\right) = E\left[2YF\left(\mathbf{x}\right) - \log\left(1+e^{2F(\mathbf{x})}\right)\middle|\,\mathbf{x}\right].$$

The solution $F^{*}\left(\mathbf{x}\right)$ that maximizes the log-likelihood must equal the $F\left(\mathbf{x}\right)$ in the true model $P\left(y=1|\mathbf{x}\right) = \frac{e^{2F(\mathbf{x})}}{1+e^{2F(\mathbf{x})}}$. Hence,

$$\begin{aligned} e^{2F^{*}(\mathbf{x})} &= P\left(y=1|\mathbf{x}\right)\left(1+e^{2F^{*}(\mathbf{x})}\right) \\ e^{2F^{*}(\mathbf{x})} &= \frac{P\left(y=1|\mathbf{x}\right)}{1-P\left(y=1|\mathbf{x}\right)} \\ F^{*}\left(\mathbf{x}\right) &= \frac{1}{2}\log\left[\frac{P\left(y=1|\mathbf{x}\right)}{P\left(y=-1|\mathbf{x}\right)}\right]. \end{aligned}$$

AdaBoost that minimizes the exponential loss yields the same solution as the logistic regression that maximizes the logistic log-likelihood.

From the above, we can see that AdaBoost gives high weights to and thus, focuses on the samples that are not correctly classified by the previous weak learners. This is exactly what Schapire (1990) referred to as *filtering* in Section 5.1.

5.3 Extensions to AdaBoost Algorithms

In this section, we introduce three extensions of (Discrete) AdaBoost (DAB) which is shown in Algorithm 12: namely, Real AdaBoost (RAB), LogitBoost (LB) and Gentle AdaBoost (GAB). We discuss how some aspects of the DAB may be modified to yield RAB, LB and GAB. In the previous section, we learned that Discrete AdaBoost minimizes an exponential loss via iteratively adding a binary weaker learner to the pool of weak learners. The addition of a new weak learner can be seen as taking a step on the direction that loss function descents in the Newton method. There are two major ways to extend the idea of Discrete AdaBoost. One focuses on making the minimization method more efficient by adding a more flexible weak learner. The other is to use different loss functions that may lead to better results. Next, we give an introduction to three extensions of Discrete AdaBoost.

5.3.1 Real AdaBoost

Real AdaBoost that Friedman, Hastie and Tibshirani (2000) propose focuses solely on improving the minimization procedure of Discrete AdaBoost. In Real AdaBoost, the weak learners are continuous comparing to Discrete AdaBoost where the weak learners are binary (discrete). Real AdaBoost is minimizing the exponential loss with continuous

updates where Discrete AdaBoost minimizes the exponential loss with discrete updates. Hence, Real AdaBoost is more flexible with the step size and direction of the minimization and minimizes the exponential loss faster and more accurately. However, Real AdaBoost also imposes restriction that the classifier must produces a probability prediction which reduces the flexibility of the model. As pointed out in the numerical examples by Chu, Lee and Ullah (2018), Real AdaBoost may achieve a larger in-sample training error due to the flexibility of its model. On the other hand, this also reduces the chance of over-fitting and would in the end achieve a smaller out-of-sample test error. Algorithm 13 illustrates the procedure of Real AdaBoost.

Algorithm 13 *1. Start with weights* $w_i = \frac{1}{n}, i = 1, \ldots, n$.

2. For $m = 1$ *to* M

(a) For $j = 1$ *to* k *(for each variable)*

i. Fit the classifier to obtain a class probability estimate $p_m(x_j) = \hat{P}_w(y = 1|x_j) \in [0, 1]$ *using weights* w_i *on the training data.*

ii. Let $f_{mj}(x_j) = \frac{1}{2} \log \frac{p_m(x_j)}{1-p_m(x_j)}$.

iii. Compute $err_{mj} = \sum_{i=1}^{n} w_i 1_{(y_i \neq \texttt{sign}(f_{mj}(x_{ji})))}$.

(b) Find $\hat{j}_m = \arg\min_j err_{mj}$.

(c) Set $w_i \leftarrow w_i \exp\left[-y_i f_{m,\hat{j}_m}(x_{\hat{j}_m,i})\right]$, $i = 1, \ldots, n$, *and normalize so that* $\sum_{i=1}^{n} w_i = 1$.

3. Output the classifier sign$[F_M(\mathbf{x})]$ *and the class probability prediction* $\hat{\pi}(\mathbf{x}) = \frac{e^{F_M(\mathbf{x})}}{e^{F_M(\mathbf{x})}+e^{-F_M(\mathbf{x})}}$ *where* $F_M(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x})$.

5.3.2 LogitBoost

Friedman, Hastie and Tibshirani (2000) also propose LogitBoost by minimizing the Bernoulli log-likelihood via an adaptive Newton algorithm for fitting an additive logistic regression model. LogitBoost extends Discrete AdaBoost in two ways. First, it uses the Bernoulli log-likelihood instead of the exponential loss function as a loss function. Furthermore, it updates the classifier by adding a linear model instead of a binary weak learner.

In LogitBoost, continuous weak learner is used similarly to Real AdaBoost. However, LogitBoost specifies the use of a linear weak learner while Real AdaBoost allows any weak learner that returns a probability between zero and one. A more fundamental difference here is that LogitBoost uses the Bernoulli log-likelihood as a loss function instead of the exponential loss. Hence, LogitBoost is more similar to logistic regression than Discrete AdaBoost and Real AdaBoost. As pointed out in the numerical examples by Chu, Lee and Ullah (2018), LogitBoost has the smallest in-sample training error but the largest out-of-sample test error. This implies that while LogitBoost is the most flexible of the four, it suffers the most from over-fitting. Algorithm 14 illustrates the procedure of LogitBoost.

Algorithm 14 *1. Start with weights* $w_i = \frac{1}{n}, i = 1, \ldots, n$, $F(\mathbf{x}) = 0$ *and probability estimates* $p(x_i) = \frac{1}{2}$.

2. For $m = 1$ *to* M

(a) Compute the working response and weights

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))}$$

$$w_i = p(x_i)(1 - p(x_i))$$

(b) For $j = 1$ to k (for each variable)

i. Fit the function $f_{mj}(x_{ji})$ by a weighted least-squares regression of z_i to x_{ji} using weights w_i on the training data.

ii. Compute $err_{mj} = 1 - R^2_{mj}$ where R^2_{mj} is the coefficient of determination from the weighted least-squares regression.

(c) Find $\hat{j}_m = \arg\min_j err_{mj}$

(d) Update $F(\mathbf{x}) \leftarrow F(\mathbf{x}) + \frac{1}{2} f_{m,\hat{j}}(x_{\hat{j}})$ and $p(\mathbf{x}) \leftarrow \frac{e^{F(\mathbf{x})}}{e^{F(\mathbf{x})}+e^{-F(\mathbf{x})}}$.

3. Output the classifier sign$[F_M(\mathbf{x})]$ and the class probability prediction $\hat{\pi}(\mathbf{x}) = \frac{e^{F_M(\mathbf{x})}}{e^{F_M(\mathbf{x})}+e^{-F_M(\mathbf{x})}}$ where $F_M(\mathbf{x}) = \sum_{m=1}^{M} f_{m,\hat{j}_m}(x_{\hat{j}_m})$.

5.3.3 Gentle AdaBoost

In Friedman, Hastie and Tibshirani (2000), Gentle AdaBoost extends Discrete AdaBoost in the sense that it allows each weak learner to be a linear model. This is similar to LogitBoost and more flexible than Discrete AdaBoost and Real AdaBoost. However, it is closer to Discrete AdaBoost and Real AdaBoost than LogitBoost in the sense that Gentle AdaBoost, Discrete AdaBoost and Real AdaBoost all minimize the exponential loss while LogitBoost minimizes the Bernoulli log-likelihood. On the other hand, Gentle AdaBoost is more similar to Real AdaBoost than Discrete AdaBoost since the weak learners are continuous and there is no need to find an optimal step size for each iteration because the weak learner is already optimal. As pointed out in the numerical examples by Chu, Lee and Ullah (2018), Gentle Boost often lies between Real AdaBoost and LogitBoost in terms of in-sample training error and out-of-sample test error. Algorithm 15 illustrates the

procedure of Gentle AdaBoost.

Algorithm 15 *1. Start with weights* $w_i = \frac{1}{n}, i = 1, \ldots, n$.

2. For $m = 1$ *to* M

(a) For $j = 1$ *to* k *(for each variable)*

i. Fit the regression function $f_{mj}(x_{ji})$ *by weighted least-squares of* y_i *on* x_{ji} *using weights* w_i *on the training data.*

ii. Compute $err_{mj} = 1 - R^2_{mj}$ *where* R^2_{mj} *is the coefficient of determination from the weighted least-squares regression.*

(b) Find $\hat{j}_m = \arg\min_j err_{mj}$

(c) Set $w_i \leftarrow w_i \exp[-y_i f_{m,\hat{j}_m}(x_{\hat{j}_m,i})]$, $i = 1, \ldots, n$, *and normalize so that* $\sum_{i=1}^n w_i = 1$.

3. Output the classifier $sign[F_M(\mathbf{x})]$ *and the class probability prediction* $\hat{\pi}(\mathbf{x}) = \frac{e^{F_M(\mathbf{x})}}{e^{F_M(\mathbf{x})} + e^{-F_M(\mathbf{x})}}$ *where* $F_M(\mathbf{x}) = \sum_{m=1}^M f_{m,\hat{j}_m}(x_{\hat{j}_m})$.

5.4 L_2Boosting

In addition to classification, the idea of boosting can also be applied to regressions. Bühlmann and Yu (2003) propose L_2Boosting that builds a linear model by minimizing the L_2 loss. Bühlmann (2006) further proved the consistency of L_2Boosting in terms of predictions. L_2Boosting is the simplest and perhaps most instructive Boosting algorithm for economists and econometricians. It is very useful for regression, in particular in the presence of high-dimensional explanatory variables.

We consider a simple linear regression

$$y = \mathbf{x}\beta + u,$$

where y is the dependent variable, $\mathbf{x}$ is the independent variable and $u \sim N(0,1)$. Note that the number of independent variables $\mathbf{x}$ could be high-dimensional, i.e. the number of independent variables in $\mathbf{x}$ can be larger than the number of observations.

This model, in the low dimension case, can be estimated by the ordinary least squares. We minimize the sum of squared errors

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where

$$\hat{y}_i = \mathbf{x}_i \hat{\beta}.$$

The solution to the problem is

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}.$$

The residual from the previous problem is

$$\hat{u}_i = y_i - \hat{y}_i.$$

In the high-dimension case, the ordinary least squares method falls down because the matrix $(\mathbf{X}'\mathbf{X})$ is not invertible. Hence, we need to use a modified least squares method to get over the high-dimension problem.

The basic idea of L_2Boosting is to use only one explanatory variable at a time. Since the number of variables p is larger than the length of the sample period n, the matrix $\mathbf{X}'\mathbf{X}$ is not invertible. However, if we use only one variable in one particular iteration,

the matrix $\mathbf{x}_j'\mathbf{x}_j$ is a scalar and thus invertible. In order to exploit the information in the explanatory variables, in the following iterations, we can use other explanatory variables to fit the residuals which are the unexplained part from previous iterations. L_2Boosting can be seen as iteratively use the least squares technique to explain the residuals from the previous least squares regressions. In the L_2Boosting algorithm, we use the least squares technique to fit the dependent variable y with only one independent variable $\mathbf{x}_j$. Then, we iteratively take the residual from the previous regression as the new dependent variable y and fit the new dependent variable with, again, only one independent variable $\mathbf{x}_j$. The detailed description of L_2Boosting is listed in Algorithm 16.

Algorithm 16 *1. Start with y_i from the training data.*

2. For $m = 1$ to M

(a) For $j = 1$ to k (for each variable)

i. Fit the regression function $y_i = \beta_{m,0,j} + \beta_{m,j}x_{ji} + u_i$ by least-squares of y_i on x_{ji}.

ii. Compute $err_{mj} = 1 - R^2_{mj}$ where R^2_{mj} is the coefficient of determination from the least-squares regression.

(b) Find $\hat{j}_m = \arg\min_j err_{mj}$

(c) Set $y_i \leftarrow y_i - \hat{\beta}_{m,0,\hat{j}_m} - \hat{\beta}_{m,\hat{j}_m}x_{\hat{j}_m,i}$, $i = 1, \ldots, n$.

3. Output the final regression model $F_M(\mathbf{x}) = \sum_{m=1}^{M} \beta_{m,0,\hat{j}_m} + \hat{\beta}_{m,\hat{j}_m}x_{\hat{j}_m}$.

5.5 Gradient Boosting

This section discusses the Gradient Boosting Machine first introduced by Friedman (2001). Breiman (2004) shows that the AdaBoost algorithm can be represented as a steepest descent algorithm in function space which we call functional gradient descent (FGD). Friedman, Hastie and Tibshirani (2000) and Friedman (2001) then developed a more general, statistical framework which yields a direct interpretation of boosting as a method for function estimation. In their terminology, it is a 'stage-wise, additive modeling' approach. Gradient Boosting is a generalization of AdaBoost and L_2Boosting. AdaBoost is a version of Gradient Boosting that uses the exponential loss and L_2Boosting is a version of Gradient Boosting that uses the L_2 loss.

The Functional Gradient Descent minimizes the risk function $R(F)$ at each $\mathbf{x}$ directly with respect to $F(\mathbf{x})$. In each iteration m, like in gradient descent, we look for a pair of optimal direction $f_m(\mathbf{x})$ and step size c_m. The optimal direction at $\mathbf{x}$ is the direction that the loss function $R(F)$ decreases the fastest. Hence, the optimal direction

$$f_m(\mathbf{x}) = E_y \left[-\frac{\partial L(y, F(\mathbf{x}))}{\partial F(\mathbf{x})} \middle| \mathbf{x} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}.$$

The optimal step size c_m can be found given $f_m(\mathbf{x})$ by a line search

$$c_m = \arg\min_{c_m} E_{y,\mathbf{x}} L(y, F_{m-1}(\mathbf{x}) + c_m f_m(\mathbf{x})).$$

Next, we update the estimated function $F(\mathbf{x})$ by

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + c_m f_m(\mathbf{x}).$$

Thus, we complete one iteration of Gradient Boosting. In practice, the stopping iteration, which is the main tuning parameter, can be determined via cross-validation or

some information criteria. The choice of step size c is of minor importance, as long as it is 'small', such as $c = 0.1$. A smaller value of c typically requires a larger number of boosting iterations and thus more computing time, while the predictive accuracy will be better and tend to over-fit less likely. The algorithm of Gradient Boosting is shown in Algorithm 17.

Algorithm 17 *1. Start with* $F_0(\mathbf{x}) = \arg\min_{const} \sum_{i=1}^{n} L(y_i, const)$.

2. For $m = 1$ *to* M

(a) calculate the pseudo-residuals $r_i^m = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$, $i = 1, \ldots, n$.

(b) $f_m(\mathbf{x}) = \arg\min_{f_m(\mathbf{x})} \sum_{i=1}^{N} (r_i^m - f_m(\mathbf{x}_i))^2$.

(c) $c_m = \arg\min_c \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + c_m f_m(\mathbf{x}_i))$.

(d) $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + c_m f_m(\mathbf{x})$.

3. Output $F_M(\mathbf{x}) = \sum_{m=1}^{M} c_m f_m(\mathbf{x})$.

In theory, any fitting criterion that estimates the conditional expectation could be used to fit the negative gradient at step $1(a)$. In Gradient Boosting, the negative gradient is also called 'pseudo-residuals' r_i^m and Gradient Boosting fits this residuals in each iteration.

5.5.1 Gradient boosting decision tree

Gradient Boosting Decision Tree (GBDT) or Boosting Tree is one of the most important methods for implementing nonlinear models in data mining, statistics, and econometrics. According to the results of data mining tasks at the data mining challenges platform, *Kaggle*, most of the competitors choose Boosting Tree as their basic technique to model the data for predicting tasks.

Obviously, Gradient Boosting Decision Tree combines the decision tree and gradient boosting method. The gradient boosting is the gradient descent in functional space,

$$f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \lambda_m \left(\frac{\partial L}{\partial f}\right)_m,$$

where m is the number of iteration, L is the loss function we need to optimize, λ_m is the learning rate. In each round, we find the best direction $-\left(\frac{\partial L}{\partial f}\right)_m$ to minimize the loss function. In gradient boosting, we can use some simple function to find out the best direction. That is, we use some functions to fit the 'pseudo-residuals' of the loss function. In AdaBoost, we often use the decision stump, a line or hyperplane orthogonal to only one axis, to fit the residual. In the Boosting Tree, we choose a decision tree to handle this task. Also, the decision stump could be seen as a decision tree with one root node and two leaf nodes. Thus, the Boosting Tree is a natural way to generalize Gradient Boosting.

Basically, the Boosting Tree learns an additive function, which is similar to other aggregating methods like Random Forest. But the decision trees are grown very differently among these methods. In the Boosting Tree, a new decision tree is growing based on the 'error' from the decision tree which grew in the last iteration. The updating rule comes from Gradient Boosting method and we will dive into the details later.

Suppose we need to implement a regression problem given samples $(y_i, x_i), i = 1, ..., n$. If we choose the square loss function, the 'pseudo-residual' should be $r_i^m = -\left(\frac{\partial L}{\partial f}\right)_m = -\left(\frac{\partial (y-f)^2}{\partial f}\right)_m = 2(y - f_m)$. The algorithm of Gradient Boosting Decision Tree is shown in Algorithm 18.

Algorithm 18 *1. Initially, estimate the first residual via* $r_i^0 = -2(y_i - \bar{y}) = -2(y_i -$

$f_1(x_i))$.

2. *For* $m = 1$ *to* M

   (a) *Based on new samples* $(r_i^m, x_i), i = 1, ..., n$, *fit a regression tree* $h_m(\mathbf{x})$.

   (b) *Let* $f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \lambda_m h_m(\mathbf{x})$, *then,* $\lambda_m = \arg\min_\lambda L(y, f_m(\mathbf{x}) + \lambda h_m(\mathbf{x}))$.

   (c) *Update* $f_{m+1}(\mathbf{x})$ *via* $f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \lambda_m h_m(\mathbf{x})$.

   (d) *Calculate the new residual* $r_i^{m+1} = -2(y_i - f_{m+1}(x_i))$, *then update the new samples as* $(r_i^{m+1}, x_i), i = 1, ...$

3. *Output the Gradient Boosting Decision Tree* $F_M(\mathbf{x}) = \sum_{m=1}^{M} \lambda_m f_m(\mathbf{x})$.

According to Algorithm 18, the main difference between Gradient Boosting and Boosting Tree is at step $1(a)$. In Boosting Tree, we use a decision tree to fit the 'residual' or the negative gradient. In other words, Boosting Tree implement the Functional Gradient Descent by following the functional gradient learned by the decision tree.

Additionally, to implement Gradient Boosting Decision Tree, we need to choose several hyperparameters: (1) N, the number of terminal nodes in trees; (2) M, the number of iterations in the boosting procedure.

Firstly, N, the number of terminal nodes in trees, controls the maximum allowed level of interaction between variables in the model. With $N = 2$ (decision stumps), no interaction between variables is allowed. With $N = 3$, the model may include effects of the interaction between up to two variables, and so on. Hastie, Tibshirani and Friedman (2009) comment that typically $4 < N < 8$ work well for boosting and results are fairly insensitive to the choice of N in this range, $N = 2$ is insufficient for many applications, and $N > 10$ is

unlikely to be required. Figure 5.1 shows the test error curves corresponding to the different number of nodes in Boosting Tree. We can see that Boosting with decision stumps provides the best test error. When the number of nodes increases, the final test error increases, especially in boosting with trees containing 100 nodes. Thus, practically, we often choose $4 < N < 8$. Secondly, the number of iterations M is related to the regularization method in Boosting Tree. We discuss the M in Section 5.5.2 in details.

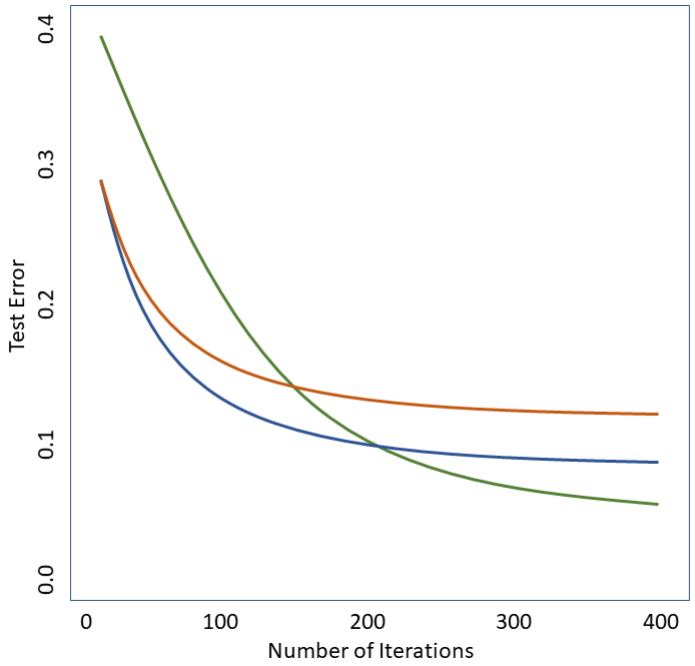

Figure 5.1: Illustration of Gradient Boosting Decision Trees with Different Nodes (green: decision stump; red line: tree with 10 leaf nodes; blue: tree with 100 leaf nodes)

5.5.2 Regularization

By following the discussion above, the Gradient Boosting Decision Trees method contains more trees when M is larger. A further issue is related to over-fitting. That is,

when there are increasingly large numbers of decision trees, Boosting Tree can fit any data with zero training error, which leads to a bad test error on new samples. To prevent the model from over-fitting, we will introduce two ways to resolve this issue.

Early stopping

A simple way to resolve this issue is to control the number of iterations M in the Boosting Trees. Basically, we can treat M as a hyper-parameter during the training procedure of Boosting Trees. Cross-Validation is an effective way to select hyperparameters including M. Since Boosting Trees method is equivalent to the steepest gradient descent in functional space, selecting the optimal M means that this steepest gradient descent will stop at the Mth iteration.

Shrinkage method

The second way to resolve the problem of over-fitting is shrinkage. That is, we add a shrinkage parameter during the training process. Let us consider the original formula for updating Boosting Trees:

$$f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \lambda_m h_m(\mathbf{x}). \tag{5.3}$$

In the Boosting Trees, we first fit $h_m(\mathbf{x})$ based on a decision tree. Then, we optimize λ_m for the best step size. Thus, we can shrink the step size by adding a shrinkage parameter ν:

$$f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \nu\lambda_m h_m(\mathbf{x}). \tag{5.4}$$

Obviously, if we set $\nu = 1$, Equation (5.4) is equivalent to Equation (5.3). Suppose we set $0 \leq \nu \leq 1$, it can shrink the optimal step size λ_m to $\nu\lambda_m$, which leads to a slower optimization. In other words, comparing to the original Boosting Tree, Shrinkage Boosting Tree learns the unknown function slower but more precise in each iteration. As a consequence, to a given $\nu < 1$, we need more steps M to minimize the error. Figure 5.2 shows this consequence. To a binary classification problem, we consider two measures: the test set deviations, which is the negative binomial log-likelihood loss on the test set, and the test set misclassification error. In the left and right panels, we can see that, with the shrinkage parameter less than 1, Boosting Tree typically need more iterations to converge but it can hit a better prediction result. Friedma (2001) found that a smaller ν will lead to a larger optimal M but the test errors in the new datasets are often better than the original Boosting Tree. Although large M may need more computational resources, this method may be inexpensive because of the faster computers.

5.5.3 Variable importance

After training Boosting Tree, the next question is to identify the variable importance. Practically, we often train boosting tree on a dataset with a large number of variables and we are interested in finding important variables for analysis.

Generally, this is also an important topic in tree-based models like Random Forest discussed in Chapter 4. Since Boosting Tree method is also an additive trees aggregating, we can use I_j^2 to measure the importance of a variable j:

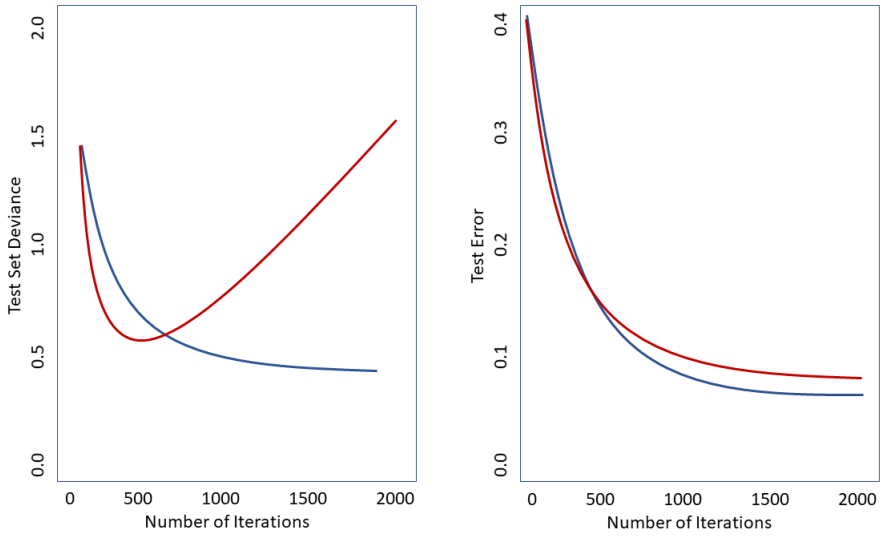

Figure 5.2: Gradient Boosting Decision Tree (6 leaf nodes) with Different Shrinkage Parameters (blue: Shrinkage $\nu = 0.6$; red: No shrinkage)

$$I_j^2 = \frac{1}{M} \sum_{m=1}^{M} I_j^2(m),$$

and $I_j^2(m)$ is the importance of variable j for the mth decision tree:

$$I_j^2(m) = \sum_{t=1}^{T_m - 1} e_t^2 I(v(t)_m = j),$$

where T_m is the number of internal nodes (non-leaf nodes) in the mth decision tree, $v(t)_m$ is the variable selected by node t, and e_t is the error improvement based on before and after splitting the space via variable $v(t)_m$.

In Random Forest or Bagging Decision Tree method, we can measure the variable importance based on the so-called Out-of-Bag errors. In Boosting Tree, since there are no Out-of-Bag samples, we can only use I_j^2. In practice, OOB-based method and I_j^2 method often provide similar results and I_j^2 works very well especially when M is very large.

Let us consider an example about the relative importance of variables for predicting spam mail via Boosting Trees. The input variable $\mathbf{x}$ could be a vector of counts of the keywords or symbols in one email. The response y is a binary variable ($Spam,\ Not\ Spam$). We regress y on $\mathbf{x}$ via Boosting Tree and then calculate the variable importance for each word or symbol. On one hand, the most important keywords and symbols may be '!', '$', 'free', that is related to money and free; on the other hand, the keywords like '3d', 'addresses' and 'labs' are not very important since they are relatively neutral. Practically, the variable importance measure often provides a result consistent with common sense.

5.6 Recent Topics in Boosting

In this section, we will focus on four attractive contributions of Boosting in recent years. First of all, we introduce two methods that are related to Boosting in time series and volatility models respectively. They are relevant topics in macroeconomic forecasting. The third method is called Boosting with Momentum (BOOM), which is a generalized version of Gradient Boosting and is more robust than the Gradient Boosting. The fourth method is called Multi-Layered Gradient Boosting Decision Tree, which is a deep learning method via non-differentiable Boosting Tree and shed light on representation learning in tabular data.

5.6.1 Boosting in nonlinear time series models

In macroeconomic forecasting, nonlinear time series models are widely used in the last 40 years. For example, Tong and Lim (1980) discuss the Threshold Autoregressive (TAR) model to describe the time dependence when the time series is higher or lower than

a threshold value. Chan and Tong (1986) propose the Smooth Transition Autoregressive (STAR) model to catch the nonlinear time dependence changing continuously between two states over time. Basically, nonlinear time series models not only perform better than linear time series models but also provide a clear way to analyze the nonlinear dependence among time series data.

Although nonlinear time series models are successful in macroeconomic time series modeling, we also need to consider their assumptions and model settings so that they can work for time series modeling. Unfortunately, in the era of big data, they cannot handle the large datasets since they often contain more complicated time dependence and higher dimensional variables along time that does not satisfy the assumptions. Essentially, the Boosting method provides an effective and consistent way to handle the time series modeling among big datasets especially with relatively fewer assumptions required.

Robinzonov, Tutz and Hothorn (2010) discuss the details of Boosting for nonlinear time series models. Suppose we have a bunch of time series dataset

$$z_t = (y_{t-1}, ..., y_{t-p}, x_{1,t-1}, ..., x_{q,t-1}, ..., x_{1,t-p}, ..., x_{q,t-p}) = (y_{t-1}, ..., y_{t-p}, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-p})$$

as a $(q+1)p$ dimensional vector, where z_t is the information set at time t, y is a series of endogenous variable with lags of p and $(\mathbf{x}_{t-1}, ..., \mathbf{x}_{t-p})$ is a q dimensional vector series with lags of p. Consider a nonlinear time series model for the conditional mean of y_t:

$$E(y_t|z_t) = F(z_t) = F(y_{t-1}, ..., y_{t-p}, x_{1,t-1}, ..., x_{q,t-1}, ..., x_{1,t-p}, ..., x_{q,t-p}),$$

where $F(z_t)$ is an unknown nonlinear function. Chen and Tsay (1993) discuss an additive

form of $F(z_t)$ for nonlinear time series modeling, which is called Nonlinear Additive Auto Regressive with exogenous variables (NAARX):

$$\begin{aligned} E(y_t|z_t) &= F(z_t) \\ &= \sum_{i=1}^{p} f_i(y_{t-i}) + \sum_{i=1}^{p} f_{1,i}(x_{1,t-i}) + ... + \sum_{i=1}^{p} f_{q,i}(x_{q,t-i}) \\ &= \sum_{i=1}^{p} f_i(y_{t-i}) + \sum_{j=1}^{q} \sum_{i=1}^{p} f_{j,i}(x_{j,t-i}). \end{aligned}$$

To optimize the best $F(z_t)$ given data, we need to minimize the loss function:

$$\hat{F}(z_t) = \arg\min_{F(z_t)} \frac{1}{T} \sum_{t=1}^{T} L(y_t, F(z_t)).$$

For example, we can use L_2 loss function $L(y_t, F(z_t)) = \frac{1}{2}(y_t - F(z_t))^2$. If we consider a parametric function $F(z_t, \beta)$, we can have the following loss function:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{T} \sum_{t=1}^{T} L(y_t, F(z_t; \beta)).$$

Since the true function of $E(y|z)$ has the additive form, the solution to the optimization problem should be represented by a sum over a bunch of estimated functions. In Boosting, we can use M different weak learner to implement:

$$F(z_t; \hat{\beta}^M) = \sum_{m=0}^{M} \nu h(z_t; \hat{\gamma}^m),$$

where ν is a shrinkage parameter for preventing over-fitting. Similar to original gradient boosting, in each iteration, we can generate a 'pseudo residual' term $r^m(z_t)$ which is:

$$r^m(z_t) = -\left.\frac{\partial L(y_t, F)}{\partial F}\right|_{F=F(z_t;\hat{\beta}^{m-1})}.$$

Thus, we can optimize $\hat{\gamma}^m$ based on the loss function:

$$\hat{\gamma}^m = \arg\min_{\gamma} \sum_{t=1}^{T} L(r^m(z_t), h(z_t;\gamma)).$$

After that, we update the $F(z_t;\hat{\beta}^m)$ as:

$$F(z_t;\hat{\beta}^m) = F(z_t;\hat{\beta}^{m-1}) + \nu h(z_t;\hat{\gamma}^m).$$

Now go back to the NAARX model. Since each function f only contains one variable, y_{t-i} or $x_{j,t-i}$, we can construct same additive form via L_2 Boosting. That is, in each iteration, we only choose one variable from the whole vector $z_t = (y_{t-1}, ..., y_{t-p}, ..., x_{q,t-1}, ..., x_{q,t-p})$ and then fit a weak learner. This is called Component-wise Boosting.

Robinzonov, Tutz and Hothorn (2010) discussed two methods of component-wise boosting with different weak learners: linear weak learner and P-Spline weak learner. The first method is called component-wise linear weak learner. For this method, we choose a linear function with one variable of z_t as a weak learner in each iteration. The algorithm of Component-wise Boosting with linear weak learner is shown in 19.

Algorithm 19 *1. Start with y_t from training data.*

2. For $m = 1$ to M

(a) For $j = 1$ to $(1 + q)p$ (for each variable)

i. Fit the regression function $y_t = \beta_{m,0,j} + \beta_{m,j} z_{j,t} + u_t$ *by least-squares of* y_t *on* $z_{j,t}$ *on the training data.*

ii. Compute $err_{mj} = 1 - R^2_{mj}$ *from the weighted least-squares regression.*

(b) Find $\hat{j}_m = \arg\min_j err_{mj}$.

(c) Set $y_t \leftarrow y_t - \hat{\beta}_{m,0,\hat{j}_m} - \hat{\beta}_{m,\hat{j}_m} z_{t,\hat{j}_m}$, $t = 1, \ldots, T$.

3. Output the final regression model $F_M(z) = \sum_{m=1}^{M} \beta_{m,0,\hat{j}_m} + \hat{\beta}_{m,\hat{j}_m} z_{\hat{j}_m}$.

Obviously, this method only provides a linear solution like an Autoregressive model with exogenous variables (ARX). We can also consider more complicated weak learner such that the nonlinear components could be caught. In the paper, P-Spline with B base learners is considered as the weak learner. The algorithm of Component-wise Boosting with P-Spline weak learner is shown in Algorithm 20.

Algorithm 20 *1. Start with* y_t *from training data.*

2. For $m = 1$ *to* M

(a) For $j = 1$ *to* $(1 + q)p$ *(for each variable)*

i. Fit the P-Spline with B Base learners $\hat{y}_t = Spline_m(z_{j,t})$ *by regressing* y_t *on* $z_{j,t}$ *on the training data.*

ii. Compute $err_{mj} = 1 - R^2_{mj}$ *from the P-Spline regression.*

(b) Find $\hat{j}_m = \arg\min_j err_{mj}$.

(c) Set $y_i \leftarrow y_t - \hat{y}_t$, $t = 1, \ldots, T$.

3. Output the final regression model $F_M(z) = \sum_{m=1}^{M} Spline_m(z_{\hat{j}_m})$.

5.6.2 Boosting in volatility models

Similarly to Boosting in nonlinear time series models for the mean, it is possible to consider Boosting in volatility models, like GARCH. Audrino and Bühlmann (2003) discussed volatility estimation via functional gradient descent for high-dimensional financial time series. Matías, Febrero-Bande, González-Manteiga and Reboredo (2010) compare Boost-GARCH with other methods, like neural networks GARCH.

Let us begin with the classic $GARCH(p,q)$ model by Bollerslev (1986):

$$y_t = \mu + e_t, t = 1, ..., T$$

$$e_t \sim N(0, h_t)$$

$$h_t = c + \sum_{i=1}^{p} \alpha_i e_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}.$$

We can implement a Maximum Likelihood Estimation (MLE) method to estimate all the coefficients. Generally, consider a nonlinear formula of the volatility function h_t:

$$h_t = g(e_{t-1}^2, ..., e_{t-p}^2, h_{t-1}, ..., h_{t-q}) = g(E_t^2, H_t),$$

where $E_t^2 = (e_{t-1}^2, ..., e_{t-p}^2)$ and $H_t = (h_{t-1}, ..., h_{t-q})$. Similarly to NAARX model, we can consider a additive form of the function g:

$$h_t = \sum_{m=1}^{M} g_m(E_t^2, H_t).$$

For simplicity, let $p = q = 1$, we have:

$$h_t = \sum_{m=1}^{M} g_m(e_{t-1}^2, h_{t-1}).$$

Thus, we can use L_2 Boosting to approximate the formula above. Since we use MLE to estimate the original GARCH model, for Boost-GARCH, we can also introduce the likelihood function for calculating the 'pseudo residual' $r_{t,m}$ instead of using the loss function. Finally, Boost-GARCH can fit an additive nonlinear formula as the estimation of h_t:

$$\hat{h}_t = \sum_{m=1}^{M} f_m(e_{t-1}^2, h_{t-1})$$

The algorithm of Boost-GARCH $(1,1)$ is shown in Algorithm 21.

Algorithm 21 *1. Start with estimating a linear GARCH (1,1) model:*

$$y_t = \mu + e_t, t = 1, ..., T$$

$$e_t \sim N(0, h_t)$$

$$h_t = c + \alpha_1 e_{t-1}^2 + \beta_1 h_{t-1}$$

2. Getting the $\hat{\mu}, \hat{h}_{t-1,0}$

3. For $m = 1$ *to* M

(a) Calculate the residual:

$$e_{t,m}^2 = (y_t - \hat{\mu})^2,$$
$$r(h_{t,m}) = -\left(\frac{\partial L}{\partial h_t}\right)_m = \frac{1}{2}\left(\frac{(y_t - \hat{\mu})^2}{\hat{h}_{t,m}^2} - \frac{1}{\hat{h}_{t,m}}\right),$$

where $L = -\sum_i \frac{1}{2}\log h_t - \sum_i \frac{(y_i-\mu)^2}{2h_t}$.

(b) Fit a nonlinear base learner $\hat{y}_t = f_m(e_{t-1}^2, h_{t-1})$ *by regressing* $r(h)_{t,m}$ *on* $e_{t-1}^2, \hat{h}_{t-1,m}$.

(c) Set $\hat{h}_{t,m} \leftarrow \hat{h}_{t,m-1} + f_m(e_{t-1}^2, \hat{h}_{t-1,m})$.

4. Output the final regression model $\hat{h}_t = \widehat{GARCH}(1,1) + \sum_{m=1}^{M} f_m(e_{t-1}^2, h_{t-1})$.

5.6.3 Boosting with momentum (BOOM)

In Section 5.5, we show that Gradient Boosting can be represented as a steepest gradient descent in functional space. In the optimization literature, gradient descent is widely discussed on its properties. First, gradient descent is easily revised for many optimization problems. Second, gradient descent often finds out good solutions no matter the optimization problem is convex or nonconvex.

But gradient descent also suffers from some drawbacks. Let us consider the plots of loss surface in Figure 5.3. Suppose the loss surface is convex. Obviously, gradient descent should converge to the global minimum eventually. But what we can see in the panel (a) is that the gradient descent converges very slow and the path of gradient descent is a zig-zag path. Thus, original gradient descent may spend a long time on converging to the optimal solution. Furthermore, the convergence is worse in a non-convex optimization problem.

(a) Gradient Descent without Momentum

(b) Gradient Descent with Momentum

Figure 5.3: Gradient Descent without and with Momentum

To resolve this issue, a very practical way is to conside 'momentum' term to the gradient descent updating rule:

$$\theta_{m+1} = \theta_m - \lambda V_m,$$
$$V_m = V_{m-1} + \nu \left(\frac{\partial L}{\partial \theta} \right)_m,$$

where θ_m is the parameter we want to optimize at mth iteration, V_m is the momentum term with another corresponding updating rule.

In the original gradient descent method, we have $V_m = \left(\frac{\partial L}{\partial \theta}\right)_m$. In $(m+1)$th iteration, the parameter θ_{m+1} is updated by following the gradient $\left(\frac{\partial L}{\partial \theta}\right)_m$ only. But when we consider momentum term, the parameter θ_m is updated by following the updating direction in previous iteration V_{m-1} and the gradient $\left(\frac{\partial L}{\partial \theta}\right)_m$ together. Intuitively, this is just like the effect of momentum in physics. When a ball is rolling down from the top, even though it comes to a flat surface, it keeps rolling for a while because of momentum.

Panel (b) in Figure 5.3 illustrates the difference between Gradient Descent without and with Momentum. Comparing to the path of convergence in the panel (a), if we consider momentum in gradient descent, the path becomes better and spends less time on moving to the optimal solution which is shown in panel (b).

As the generalized version of gradient descent in function space, gradient boosting may also suffer from the same problem when the loss surface is complicated. Thus, a natural way to improve the gradient boosting method is considering the momentum term in its updating rule. Mukherjee, Canini, Frongillo and Singer (2013) discuss a general analysis of a fusion of Nesterov's accelerated gradient with parallel coordinate descent.

The resulting algorithm is called Boosting with Momentum (BOOM). Namely, BOOM retains the momentum and convergence properties of the accelerated gradient method while taking into account the curvature of the objective function. They also show that BOOM is especially effective in large scale learning problems. Algorithm 22 provides the procedure of BOOM via Boosting Tree.

Algorithm 22 *1. Initially, estimate the first residual via* $r_i^0 = -2(y_i - \bar{y}) = -2(y_i - f_1(x_i))$.

2. For $m = 1$ *to* M

(a) Based on new samples $(r_i^m, x_i), i = 1, ..., n$*, fit a regression tree* $h_m(\mathbf{x})$.

(b) Let $V_m = V_{m-1} + \lambda_m h_m(\mathbf{x})$.

(c) Let $f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \nu V_m$*, then optimize* λ_m *via* $\lambda_m = \arg\min_\lambda L(y, f_m(\mathbf{x}) + \nu V_m) = \arg\min_\lambda L(y, f_m(\mathbf{x}) + \nu(V_{m-1} + \lambda h_m(\mathbf{x})))$.

(d) Update $f_{m+1}(\mathbf{x})$ *via* $f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \nu V_m$.

(e) Calculate the new residual $r_i^{m+1} = -2(y_i - f_{m+1}(x_i))$*, then update the new samples as* $(r_i^{m+1}, x_i), i = 1, ..., n$.

3. Output the Gradient Boosting Decision Tree $F_M(\mathbf{x}) = \sum_{m=1}^M \nu V_m$.

The main difference between Boosting with Momentum and ordinary Boosting Tree is a step to update V_m. Also, we have one more hyperparameter to decide ν, which decides the fraction of gradient information saved for next iteration updating of $f_m(\mathbf{x})$. Practically, we set $0.5 < \nu < 0.9$ but it is more reasonable to tune ν via cross-validation.

This method can be generalized to Stochastic Gradient Boosting discussed by Friedman (2002). Algorithm 23 shows the procedure of BOOM via Stochastic Gradient Boosting Tree.

Algorithm 23 *1. Initially, randomly select a subset of the samples* $(y_i, x_i), i = 1, ..., n_s$, *where* $0 < n_s < n$.

2. Estimate the first residual via $r_i^0 = -2(y_i - \bar{y}) = -2(y_i - f_1(x_i))$.

3. For $m = 1$ *to* M.

(a) Based on new samples $(r_i^m, x_i), i = 1, ..., n_s$, *fit a regression tree* $h_m(\mathbf{x})$.

(b) Let $V_m = V_{m-1} + \lambda_m h_m(\mathbf{x})$.

(c) Let $f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \nu V_m$, *then optimize* λ_m *via* $\lambda_m = \arg\min_\lambda L(y, f_m(\mathbf{x}) + \nu V_m) = \arg\min_\lambda L(y, f_m(\mathbf{x}) + \nu(V_{m-1} + \lambda h_m(\mathbf{x})))$.

(d) Update $f_{m+1}(\mathbf{x})$ *via* $f_{m+1}(\mathbf{x}) = f_m(\mathbf{x}) + \nu V_m$.

(e) Calculate the new residual $r_i^{m+1} = -2(y_i - f_{m+1}(x_i))$, *then update the new samples as* $(r_i^{m+1}, x_i), i = 1, ..., n_s$.

4. Output the Gradient Boosting Decision Tree $F_M(\mathbf{x}) = \sum_{m=1}^{M} \nu V_m$.

There some differences between BOOM with Boosting Tree and Stochastic Boosting Tree. In Boosting Tree, we use all the n samples to update the decision tree in each iteration. But Stochastic Boosting Tree randomly selects $\frac{n_s}{n}$ fraction of samples to grow a decision tree in each iteration. When the sample size n is increasingly large, selecting a subset of samples could be a better and more efficient way to implement the Boosting Tree algorithm.

5.6.4 Multi-layered gradient boosting decision tree

Last 10 years witnessed the dramatic development in the fields about deep learning, which mainly focus on distilling hierarchical features via multi-layered neural networks automatically. From 2006 deep learning methods have changed so many areas like computer vision and natural language processing.

The multi-layered representation is the key ingredient of deep neural networks. Thus, the combination of multi-layered representation and Boosting Tree are expected in handling very complicated tabular data analysis tasks. But there are few research papers exploring multi-layered representation via non-differentiable models, like Boosted Decision Tree. That is, the gradient-based optimization method which is always used in training multi-layered neural networks cannot be introduced in training multi-layered Boosting methods.

Feng, Yu and Zhou (2018) explored one way to construct Multi-Layered Gradient Boosting Decision Tree (mGBDT) with an explicit emphasis on exploring the ability to learn hierarchical representations by stacking several layers of regression GBDTs. The model can be jointly trained by a variant of target propagation across layers, without the need to derive back-propagation or to require differentiability.

Figure 5.4 provides the structure of a Multi-Layered Gradient Boosting Decision Tree. $F_m, m = 1, ..., M$ are the M layers of a mGBDT. Similar to the multi-layered neural networks, the input o_0 is transformed to $o_1, ..., o_M$ via $F_1, ..., F_M$. Then, the final output o_M is the prediction of the target variable y. But all the F_m are constructed via gradient boosting decision tree, we cannot training them via back-propagation method used in train-

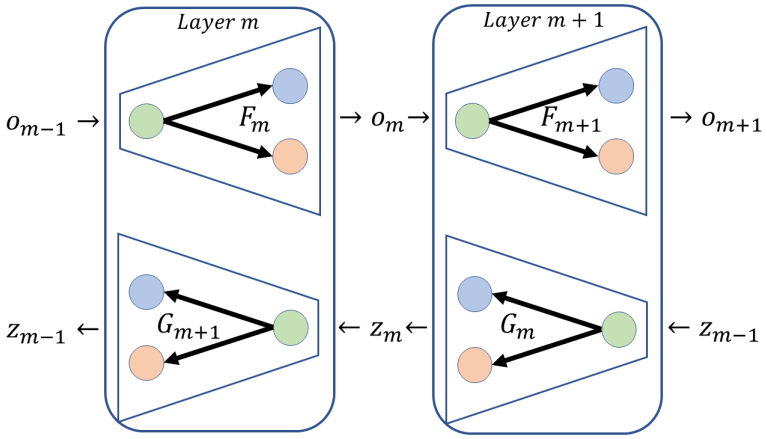

Figure 5.4: Illustration of Multi-Layered Gradient Boosting Decision Tree

ing multi-layered neural networks. Feng, Yu and Zhou (2018) introduced another group of functions $G_m, m = 1, ..., M$ and corresponding variables $z_m, m = 1, ..., M$.

Intuitively, the group of function G_m are introduced for achieving back-propagation algorithm in non-differentiable Boosting Tree. To train Multi-layered Gradient Boosting Decision Tree, firstly, we use 'forward propagation' method to calculate all the $o_m, m = 1, ..., M$. Secondly, to (o_m), G_m are trained to reconstruct o_m via optimizing the loss function $L(o_m, G_m(F_m(o_m)))$. That is, we train G_m to learn 'back-propagation'. Then, after training all the $G_m, m = 1, ..., M$, we can do 'back-propagation' to generate $z_m, m = 1, ..., M$, that represents the information to each layer. Next, to the pairs of (z_m, z_{m-1}), we train F_m to optimize another loss function $L(z_m, F_m(z_{m-1}))$. Finally, we can update all the F_m and G_m via Boosting Tree method. Algorithm 24 shows the procedure of Multi-Layered Gradient Boosting Decision Tree.

Algorithm 24 *1. Input: Number of layers M, layer dimension d_m, samples $(y_i, x_i), i = 1, ..., n$. Loss function L. Hyper-parameters α, γ, K_1, K_2, T, σ^2.*

2. *Initially, set* $F_m^0 = Initialize(M, d_m), m = 1, ..., M;$

3. *For* $t = 1$ *to* T

   (a) *Propagate the* o_0 *to calculate* $o_m = F(o_{m-1}), m = 1, ..., M$

   (b) $z_M^t = o_M - \alpha \frac{\partial L(y, o_M)}{\partial o_M}$

   (c) *For* $m = M$ *to* 2

      i. $G_m^t = G_m^{t-1}$

      ii. $o_{m-1}^{noise} = o_{m-1} + \epsilon, \epsilon \sim N(0, diag(\sigma^2))$

      iii. $L_m^{inv} = L(o_m^{noise}, G_m^t(F_m^{t-1}(o_m^{noise})))$

      iv. *for* $k = 1$ *to* K_1

         A. $r_k = -\frac{\partial L_m^{inv}}{\partial G_m^t(F_m^{t-1}(o_m^{noise}))}$

         B. *Fit a decision tree* h_k *to* r_k

         C. $G_m^t = G_m^t + \gamma h_k$

      v. $z_{m-1} = G_m^t(z_m)$

   (d) *For* $m = 1$ *to* M

      i. $F_m^t = F_m^{t-1}$

      ii. $L_m = L(z_m^t, F_m^t(o_{m-1}))$ *using gradient boosting decision tree*

      iii. *for* $k = 1$ *to* K_2

         A. $r_k = -\frac{\partial L_m}{\partial F_j^t(o_m)}$

         B. *Fit a decision tree* h_k *to* r_k

         C. $F_m^t = F_m^t + \gamma h_k$

      iv. $o_m = F_m^t(o_{m-1})$

4. Output the trained multi-layered gradient boosting decision tree.

Feng, Yu and Zhou (2018) suggested to optimize $L^{inv} = L(o_m^{noise}, G_m^t(F_m^{t-1}(o_m^{noise})))$ instead of $L_m^{inv} = L(o_m, G_m^t(F_m^{t-1}(o_m)))$ to make the training of G_m more robust. Also, the authors found that the multi-layered gradient boosting decision tree is very robust to most hyper parameters. Without fine-tuning the parameters, this method can achieve very attractive results.

Furthermore, consider the noisy loss function from the perspective of minimizing the reconstruction error, this process could be seen as an encoding-decoding process. First, in each layer F_m encodes the input via a nonlinear transform. Then, G_m learns how to decode the transformed output back to the original input. This is similar to the Auto Encoder method in deep learning. Thus, we can also use the Multi-layered Gradient Boosting Decision Tree to do encoding-decoding, which shed a light on implementing unsupervised learning tasks in the tabular data in economics.

5.7 Boosting in Macroeconomics and Finance

Boosting methods are widely used in classification and regression. Gradient Boosting implemented in the packages, like *XGBoost* and *LightGBM*, is a very popular algorithm among data science competitions and industrial applications. In this section, we discuss four applications of boosting algorithms in macroeconomics.

5.7.1 Boosting in predicting recessions

Ng (2014) uses boosting to predict recessions 3, 6, and 12 months ahead. Boosting is used to screen as many as 1,500 potentially relevant predictors consisting of 132 real and financial time series and their lags. The sample period is 1961:12011:12. In this application, boosting is used to select relevant predictors from a set of potential predictors as well as probability estimation and prediction of the recessions. In particular, the analysis uses the Bernoulli loss function as implemented in the *GBM* package of Ridgeway (2007). The package returns the class probability instead of classifications. For recession analysis, the probability estimate is interesting in its own right, and the flexibility to choose a threshold other than one-half is convenient.

5.7.2 Boosting diffusion indices

Bai and Ng (2009) use boosting to select and estimate the predictors in factoraugmented autoregressions. In their application, boosting is used to make 12 months ahead of forecast on inflation, the change in Federal Funds rate, the growth rate of industrial production, the growth rate of employment, and the unemployment rate. A sample period from 1960:1 to 2003:12 was used for a total of 132 times series. They use two boosting algorithms, namely L_2Boosting and Block Boosting.

5.7.3 Boosting with Markov-switching

Adam, Mayr and Kneib (2017) propose a novel class of flexible latent-state time series regression models called Markov-switching generalized additive models for location, scale, and shape. In contrast to conventional Markov-switching regression models, the

presented methodology allows users to model different state-dependent parameters of the response distribution - not only the mean, but also variance, skewness and kurtosis parameters - as potentially smooth functions of a given set of explanatory variables. The authors also propose an estimation approach based on the EM algorithm using the gradient boosting framework to prevent over-fitting while simultaneously performing variable selection. The feasibility of the suggested approach is assessed in simulation experiments and illustrated in a real-data setting, where the authors model the conditional distribution of the daily average price of energy in Spain over time.

5.7.4 Boosting in financial modeling

Rossi and Timmermann (2015) construct a new procedure for estimating the covariance risk measure in ICAPM model. First, one or more economic activity indices are extracted from macroeconomic and financial variables for estimating the covariance matrix. Second, given realized covariance matrix as the covariance matrix measure, Boosting Regression Tree is applied in projecting realized covariance matrix on the indices extracted in the first step. Lastly, predictions of the covariance matrix are made based on the nonlinear function approximated by Boosting Regression Tree and applied into the analysis of ICAPM method.

5.8 Conclusion

In this chapter, we focus on Boosting method. We start with an introduction of the well known AdaBoost. Several variants of AdaBoost, like Real AdaBoost, LogitBoost and

Gentle AdaBoost are also discussed. Then, we consider in regression problem and introduce L_2 Boosting. Next, Gradient Boosting and Gradient Boosting Decision Tree are discussed in theory and practice. Then, we introduce the several variants of Gradient Boosting such as Component-wise Boosting and Boost-GARCH for nonlinear time series modeling, Boosting with Momentum and multi-layered Boosting Tree. Finally, we discuss several applications of Boosting in macroeconomic forecasting and financial modeling.

Chapter 6

Conclusions

The development of data science and machine learning techniques provides powerful tools for economic and econometric researches. Economists can deeply explore and study the individual's actions, market equilibrium, and economic crises. Given that, our works focus on using machine learning in economics and we try to resolve the related applications' issues from the perspective of theory and practice. In Chapter 2, we introduce a new nonlinear variable selection method, L_1-regularized soft decision tree (SDT), for resolving the nonlinear factor selection in the causal treatment effect framework. We also discuss the oracle properties in the nonlinear regression, which guarantees consistent variable selection. Based on the chosen variables, most nonlinear regression can provide a higher convergence rate and more precise estimation results. Our method can support the econometric and empirical economic researches under the unknown regression functions. In Chapter 3, we explore the unsupervised learning method in the literature of economic crisis forecasting. We propose a new Mode Constrastive Auto Encoder (MCAE) method for detecting the

clustering distribution and extracting the features of high-dimensional regressors x. Additionally, MCAE can resolve the issue of missing values by Auto Encoder (AE) and time trends by Boosted Auto Encoder (BAE) in the high dimensional time series data. When the target economic variables y are noisy or ambiguous, our method provides a new perspective for understanding the reason behind the crisis and makes the crisis forecasting more robust.

Chapter 4 and Chapter 5 focus on the economic forecasting using the Bagging and Boosting methods and provide a reference to economic researchers when using machine learning methods. Chapter 4 firstly introduces the Bootstrap Averaging (Bagging) with its variants, such as Subagging and Bragging. Next, we present the Decision Tree and discuss one of the most popular tree-based machine learning techniques, the Random Forest. Then, we introduce several variants of Random Forests, including Extreme Random Trees and Soft Decision Tree. Finally, we explore several applications of economic forecasting and inference using Random Forests and its variants. Chapter 5 firstly explore the Boosting algorithm and its variants, including AdaBoost, Real AdaBoost, LogitBoost, and Gentle AdaBoost. Then, the definitions of Gradient Boosting and L_2 Boosting are introduced. Next, we explore some recent developments of Boosting, including Boost-GARCH, Boosting with Momentum (BOOM), and multi-layered Gradient Boosting Decision Tree (mGBDT). Finally, we discuss several applications of Boosting in macroeconomic forecasting and financial modeling.

Bibliography

[1] Timo Adam, Andreas Mayr, and Thomas Kneib. Gradient boosting in Markov-switching generalized additive models for location, scale and shape. 2017.

[2] Ashvin Ahuja, Kevin Wiseman, and Murtaza H Syed. Assessing country risk — selected approaches. *IMF Technical Notes and Manuals 17/08*, 2017.

[3] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *Journal of Machine Learning Research*, (15):3743—3773, 2014.

[4] Lucia Alessi, Antonio Antunes, Jan Babecky, Simon Baltussen, Markus Behn, Diana Bonfim, Oliver Bush, Carsten Detken, Jon Frost, Rodrigo Guimaraes, Tomas Havranek, Mark Joy, Karlo Kauko, Jakub Mateju, Nuno Monteiro, Benjamin Neudorfer, Tuomas A. Peltonen, Marek Rusnak, Paulo Manuel Marques Rodrigues, Willem Schudel, Michael Sigmund, Hanno Stremmel, Katerina Smidkova, Ruben van Tilburg, Borek Vasicek, and Diana Zigraiova. Comparing different early warning systems: results from a horse race competition among members of the Macro-prudential Research Network. *Mimeo*, 2014.

[5] Marcus Alexander, Matthew Harding, and Carlos Lamarche. The human cost of economic crises. *SIEPR Working Paper*, pages 08–029, 2008.

[6] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding Deep Neural Networks with Rectified Linear Units. *International Conference on Learning Representations*, 2018.

[7] Susan Athey. *The Impact of Machine Learning on Economics, The Economics of Artificial Intelligence: An Agenda*. University of Chicago Press, 2019.

[8] Susan Athey and Guido Imbens. Recursive Partitioning for Heterogeneous Causal Effects. *Proceedings of the National Academy of Sciences of the United States of America*, 113(27):7353–7360, 2016.

[9] Susan Athey and Guido W. Imbens. Machine Learning Methods Economists Should Know About. *arXiv*, 2019.

[10] Susan Athey, Julie Tibshirani, and Stefan Wager. Generalized Random Forests. *Annals of Statistics*, 47(2):1179–1203, 2019.

[11] Francesco Audrino and Peter Bühlmann. Volatility estimation with functional gradient descent for very high-dimensional financial time series. *The Journal of Computational Finance*, 6(3):65–89, 2003.

[12] Francesco Audrino and Marcelo C. Medeiros. Modeling and forecasting short-term interest rates: the benefits of smooth regimes, macroeconomic variables, and Bagging. *Journal of Applied Econometrics*, 26(6):999–1022, 2011.

[13] Jan Babecky, Tomas Havranek, Jakub Mateju, Marek Rusnak, Katerina Smidkova, and Borek Vasicek. Banking, debt, and currency crises: early warning indicators for developed countries. *ECB Working Paper 1485*, 2012.

[14] Jushan Bai and Serena Ng. Boosting diffusion indices. *Journal of Applied Econometrics*, 24(4):607–629, jun 2009.

[15] P L Bartlett and M Traskin. AdaBoost is consistent. *Journal of Machine Learning Research*, 8:2347–2368, 2007.

[16] Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

[17] Suman S Basu, Marcos d Chamon, and Christopher W. Crowe. A model to assess the probabilities of growth, fiscal, and financial crises. *IMF Working Paper WP/17/282*, 2017.

[18] Alexandre Belloni, Victor Chernozhukov, Denis Chetverikov, and Ying Wei. Uniformly Valid Post-Regularization Confidence Regions for Many Functional Parameters in Z-estimation Framework. *Annals of Statistics*, 46(6B):3643–3675, 2018.

[19] Alexandre Belloni, Victor Chernozhukov, and Christian Hansen. Inference on Treatment Effects after Selection among High-Dimensional Controls. *Review of Economic Studies*, 81(2):608–650, 2013.

[20] Alexandre Belloni, Victor Chernozhukov, and Christian Hansen. High-Dimensional Methods and Inference on Structural and Treatment Effects. *Journal of Economic Perspectives*, 28(2):29–50, 2014.

[21] Ben Bernanke. The Great Moderation. *Remarks at the Eastern Economic Association*, 2004.

[22] Johannes Beutel, Sophia List, and Gregor von Schweinitz. An evaluation of early warning models for systemic banking crises: does machine learning improve predictions? *Discussion Paper No. 48/2018*, 2018.

[23] Gerard Biau, Luc Devroye, and Lugosi Gabor. Consistency of Random Forests and Other Averaging Classifiers. *Journal of Machine Learning Research*, 8:2015–2033, 2008.

[24] Olivier Biau and Angela D'Elia. Euro area GDP forecasting using large survey datasets. A Random Forest approach. 2011.

[25] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, apr 1986.

[26] Richard Breen, Seongsoo Choi, and Anders Holm. Heterogeneous Causal effects and Sample Selection Bias. *Sociological Science*, 2:351–369, 2015.

[27] Leo Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

[28] Leo Breiman. Some infinity theory for predictor ensembles. Technical report, 2000.

[29] Leo Breiman. Random Forests. In *Machine Learning*, pages 5–32. 2001.

[30] Leo Breiman. Population theory for boosting ensembles. *Annals of Statistics*, 32(1):1–11, feb 2004.

[31] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and regression trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[32] Peter Bühlmann. Bagging, Subagging and Bragging for Improving Some Prediction Algorithms. Technical report, 2003.

[33] Peter Bühlmann. *Bagging, Boosting and ensemble methods*, pages 877–907. Handbook of Computational Statistics: Concepts and Methods. Springer, 2004.

[34] Peter Bühlmann. Boosting for high-dimensional linear models. *Annals of Statistics*, 34(2):559–583, apr 2006.

[35] Peter Bühlmann and Bin Yu. Analyzing Bagging. *Annals of Statistics*, 30(4):927–961, 2002.

[36] Peter Bühlmann and Bin Yu. Boosting with the L2 loss: Regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, jun 2003.

[37] Andreas Buja and Werner Stuetzle. Bagging does not always decrease mean squared error definitions. Technical report, 2000.

[38] Andreas Buja and Werner Stuetzle. Smoothing effects of Bagging. *Preprint. AT&T Labs-Research.*, 2000.

[39] Valerie Cerra and Sweta Chaman Saxena. Growth dynamics: The myth of economic recovery. *American Economic Review*, 98(1):439–57, 2008.

[40] Marcos Chamon and C. Crowe. *Predictive Indicators of Financial Crises*, pages 499–505. 2013.

[41] K. S. Chan and H. Tong. on estimating thresholds in autoregressive models. *Journal of Time Series Analysis*, 7(3):179–190, 1986.

[42] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321357, 2002.

[43] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *SIGKDD Explor. Newsl.*, 6(1):16, 2004.

[44] Jiahua Chen. Consistency of the MLE under Mixture Models. *Statistical Science*, 32(1):47–63, 2017.

[45] Rong Chen and Ruey S. Tsay. Nonlinear additive ARX models. *Journal of the American Statistical Association*, 88(423):955–967, sep 1993.

[46] Xiaohong Chen. Large Sample Sieve Estimation of Semi-Nonparametric Models. *Handbook of Econometrics*, 6(SUPPL. PART B):5549–5632, 2007.

[47] Xiaohong Chen and Halbert White. Improved Rates and Asymptotic Normality for Nonparametric Neural Network Estimators. *IEEE Transactions on Information Theory*, 45(2):682–691, 1999.

[48] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, and Whitney Newey. Double/debiased/Neyman Machine Learning of Treatment Effects. *American Economic Review*, 107(5):261–265, 2017.

[49] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/Debiased Machine Learning for Treatment and Causal Parameters. *The Econometrics Journal*, 21:C1–C68, 2018.

[50] Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010.

[51] Lawrence J. Christiano, Martin S. Eichenbaum, and Mathias Trabandt. On dsge models. *Journal of Economic Perspectives*, 32(3):113–40, August 2018.

[52] Charis Christofides, Theo Eicher, and Chris Papageorgiou. Did established early warning signals predict the 2008 crises? *European Economic Review*, 81(C):103–114, 2016.

[53] Jianghao Chu, Tae-Hwy Lee, and Aman Ullah. Component-wise AdaBoost algorithms for high-dimensional binary classification and class probability prediction. In *Handbook of Statistics*. Elsevier, dec 2018.

[54] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations*, 2016.

[55] Juan De Oña and Concepción Garrido. Extracting the contribution of independent variables in neural network models: A new approach to handle instability. *Neural Comput. Appl.*, 25:3 – 4, 2014.

[56] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[57] Donald B., Rubin. Estimating Causal Effects of Treatment in Randomized and Nonrandomized Studies. *Journal of Educational Psychology*, 66(5):688–701, 1974.

[58] Brandley Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[59] Jianqing Fan and Runze Li. Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

[60] Max H. Farrell, Tengyuan Liang, and Sanjog Misra. Deep Neural Networks for Estimation and Inference. 2018.

[61] Thomas Fischer, Christopher Krauss, and Alex Treichel. Machine learning for time series forecasting - a simulation study. 2018.

[62] Jeffrey A. Frankel and Andrew K. Rose. Currency crashes in emerging markets: An empirical treatment. *Journal of International Economics*, 41(3):351 – 366, 1996.

[63] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[64] Rina Friedberg, Julie Tibshirani, Susan Athey, and Stefan Wager. Local Linear Forests. pages 1–36, 2018.

[65] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28(2):337–407, apr 2000.

[66] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[67] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, feb 2002.

[68] Jerome H Friedman and Peter Hall. On Bagging and nonlinear estimation. *Journal of Statistical Planning and Inference*, 137(3):669–683, 2007.

[69] Nicholas Frosst and Geoffrey Hinton. Distilling a Neural Network into a Soft Decision Tree. In *CEX Workshop Proceedings*, 2017.

[70] Sylvia Fruhwirth-Schnatter. Dealing with label switching. *Mixtures: Estimation and Applications*, 2011.

[71] Christopher R. Genovese and Larry Wasserman. Rates of convergence for the gaussian mixture sieve. *The Annals of Statistics*, 28(4):1105–1127, 2000.

[72] Christopher R. Genovese and Larry Wasserman. Rates of convergence for the gaussian mixture sieve. *The Annals of Statistics*, 28(4):1105–1127, 2000.

[73] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely Randomized Trees. *Machine Learning*, 63(1):3–42, 2006.

[74] Jason Hartford, Greg Lewis, Kevin Leyton-Brown, and Matt Taddy. Deep IV: A Flexible Approach for Counterfactual Prediction. In *The 34th International Conference on Machine Learning, Sydney, Australia*, 2016.

[75] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. 2009.

[76] Trevor Hastie, Robert Tibshirani, and Martin Wainwright Hastie. Statistical Learning with Sparsity Monographs on Statistics and Applied Probability. *Crc*, page 362, 2015.

[77] Eric Hillebrand, Tae-Hwy Lee, and Marcelo Medeiros. *Bagging constrained equity premium predictors*, chapter 14, pages 330–356. Essays in Nonlinear Time Series Econometrics, Festschrift in Honor of Timo Tersvirta. Oxford University Press, 2014.

[78] Keisuke Hirano and Jonathan H. Wright. Forecasting with model uncertainty: representations and risk reduction. *Econometrica*, 85(2):617–643, 2017.

[79] Markus Holopainen and Peter Sarlin. Toward robust early-warning models: a horse race, ensembles and model uncertainty. *Quantitative Finance*, 17(12):1933–1963, 2017.

[80] Torsten Hothorn and Achim Zeileis. Transformation Forests. 2017.

[81] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme Learning Machine: algorithm, theory and applications. *Neurocomputing*, 70:489–501, 2006.

[82] Imbens, Guido W. and Rubin, Donald B. *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction*. Cambridge University Press, 2017.

[83] Atsushi Inoue and Lutz Kilian. How useful is Bagging in forecasting economic time. *Journal of the American Statistical Association*, 103(482):511–522, 2008.

[84] Ozan Irsoy, Olcay Taner Yildiz, and Ethem Alpaydin. Soft decision trees. In *International Conference on Pattern Recognition*, 2012.

[85] Ozan Irsoy, Olcay Taner Yldz, and Ethem Alpaydn. A Soft Decision Tree. In *21st International Conference on Pattern Recognition (ICPR 2012)*, 2012.

[86] Herbert Jaeger. The echo state approach to analysing and training Recurrent Neural Networks-with an erratum note. Technical report, 2001.

[87] Silke Janitza, Ender Celik, and Anne Laure Boulesteix. A computationally fast variable importance test for Random Forests for high-dimensional data. *Advances in Data Analysis and Classification*, (185):1–31, 2016.

[88] Heinrich Jiang and Ofir Nachum. Identifying and correcting label bias in machine learning. *ArXiv*, abs/1901.04966, 2019.

[89] Sainan Jin, Liangjun Su, and Aman Ullah. Robustify financial time series forecasting with Bagging. *Econometric Reviews*, 33(5-6):575–605, 2014.

[90] Michael Jordan and Robert Jacob. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6:181–214, 1994.

[91] Michael I. Jordan and Lei Xu. Convergence Results for the EM Approach to Mixtures of Experts Architectures. *Neural Networks*, 8(9):1409–1431, 1995.

[92] Graciela Laura Kaminsky. Leading indicators of currency crises. *IMF Staff Papers*, 45(1):1–48, 1998.

[93] J. Kiefer and J. Wolfowitz. Consistency of the maximum likelihood estimator in the presence of infinitely many incidental parameters. *Ann. Math. Statist.*, 27(4):887–906, 12 1956.

[94] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, 2017-December:972–981, 2017.

[95] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, Samuel Rota Bul, Fondazione Bruno Kessler, and S Rota Bulo'. Deep Neural Decision Forests. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475, 2015.

[96] Sotiris Kotsiantis, D. Kanellopoulos, and P. Pintelas. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30:25–36, 2005.

[97] G. Kishor Kumar, P. Viswanath, and A. Ananda Rao. Ensemble of Randomized Soft Decision Trees for Robust Classification. *Sadhana*, 41(3):273–282, 2016.

[98] Jan Hannes Lang, Tuomas A. Peltonen, and Peter Sarlin. A framework for early-warning modeling with an application to banks. *ECB Working Paper*, (2182), 2018.

[99] Tae-Hwy Lee, Yundong Tu, and Aman Ullah. Nonparametric and semiparametric regressions subject to monotonicity constraints: estimation and forecasting. *Journal of Econometrics*, 182(1):196–210, 2014.

[100] Tae-Hwy Lee, Yundong Tu, and Aman Ullah. Forecasting equity premium: global historical average versus local historical average and constraints. *Journal of Business and Economic Statistics*, 33(3):393–402, 2015.

[101] Tae-Hwy Lee and Yang Yang. Bagging Binary and Quantile Predictors for Time Series. *Journal of Econometrics*, 2(1):465–497, 2006.

[102] Tae-Hwy Lee and Yang Yang. Bagging binary and quantile predictors for time series. *Journal of Econometrics*, 135(1):465–497, 2006.

[103] Greg Lewis and Vasilis Syrgkanis. Adversarial Generalized Method of Moments. *arXiv*, 2018.

[104] Yi Lin and Hao Helen Zhang. Component selection and smoothing in multivariate nonparametric regression. *The Annals of Statistics*, 34(5):2272–2297, 2006.

[105] Jared K. Lunceford and Marie Davidian. Stratification and Weighting via the Propensity Score in Estimation of Causal Treatment Effects: A Comparative Study. *Statistics in Medicine*, 23(19):2937–2960, 2004.

[106] Ye Luo and Martin Spindler. Estimation and Inference of Treatment Effects with L_2-Boosting in High-Dimensional Settings. 2017.

[107] Chuong Luong and Nikolai Dokuchaev. Forecasting of realised volatility with the Random Forests algorithm. *Journal of Risk and Financial Management*, 11(4):61, 2018.

[108] Lester Mackey, Vasilis Syrgkanis, and Dias Zadik. Orthogonal Machine learning: Power and Limitations. *International Conference on Machine Learning*, 13:9112–9124, 2018.

[109] Ashok Vardhan Makkuva, Sewoong Oh, Sreeram Kannan, and Pramod Viswanath. Breaking the Gridlock in Mixture-of-Experts: Consistent and Efficient Algorithms. pages 1–47, 2018.

[110] Robert McCulloch. BART: Bayesian additive regression trees presentation. 4(1):266–298, 2010.

[111] David Mease, Abraham J. Wyner, and Andreas Buja. Cost-weighted boosting with jittering and over/under-sampling: Jous-boost. *Journal of Machine Learning Research*, 8:409–439, 2007.

[112] Paulo Medas, Tigran Poghosyan, Yizhi Xu, Juan Farah-Yacoub, and Kerstin Gerling. Fiscal crises. *Journal of International Money and Finance*, 88:191 – 207, 2018.

[113] Indraneel Mukherjee, Kevin Canini, Rafael Frongillo, and Yoram Singer. Parallel boosting with momentum. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železny, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 17–32. Springer, Berlin, Heidelberg, 2013.

[114] Serena Ng. Viewpoint: Boosting recessions. *Canadian Journal of Economics*, 47(1):1–34, feb 2014.

[115] Rickard Nyman and Paul Ormerod. Predicting economic recessions using machine learning. 2016.

[116] Anastasios Panagiotelis, George Athanasopoulos, Rob J Hyndman, Bin Jiang, and Farshid Vahid. Macroeconomic forecasting for Australia using a large number of predictors. *Interantional Journal of Forecasting*, 35(2):616–633, 2019.

[117] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *International Conference on Learning Representations*, 2014.

[118] Catherine A Pattillo and Andrew Berg. Are currency crisis predictable? A test. *IMF Staff Papers*, pages 107–138, 1999.

[119] Pearl, Judea. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2009.

[120] Pearl, Judea and Mackenzie, Dana. *The Book of Why: The New Science of Cause and Effect*. Basic Books, 2018.

[121] J. Pfanzagl. Regression shrinkage and selection via the lasso. *Journal of Statistical Planning and Inference*, 19:137–158, 1988.

[122] Dimitris N Politis, Joseph P Romano, and Mechael Wolf. *Subsampling*. 1999.

[123] J. Ross Quinlan. C4.5: programs for machine learning. *Machine Learning*, 16(3):235–240, 1994.

[124] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[125] Trivellore Raghunathan, James Lepkowski, John Hoewyk, and Peter Solenberger. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey Methodology*, 27(1):85–95, 2000.

[126] Richard Redner. Note on the Consistency of the Maximum Likelihood Estimate for Nonidentifiable Distributions. *Annals of Statistics*, 9(1):225–228, 1981.

[127] Richard A. Redner and Homer F. Walker. Mixture Densities , Maximum Likelihood and the EM Algorithm. *SIAM Review*, 26(2):195–239, 2008.

[128] Carmen M. Reinhart and Kenneth S. Rogoff. Recovery from financial crises: Evidence from 100 episodes. *American Economic Review*, 104(5):50–55, 2014.

[129] Adam Richardson, Thomas Mulder, and Tugru l Vehbi. Nowcasting New Zealand GDP using Machine learning Algorithms. *SSRN Electronic Journal*, 2018.

[130] Greg Ridgeway. Generalized boosted models: A guide to the gbm package. Technical Report 4, 2007.

[131] P. M. Robinson. Root-n-consistent semiparametric regression. *Econometrica*, 56(4):931–954, 1988.

[132] Nikolay Robinzonov, Gerhard Tutz, and Torsten Hothorn. Boosting Techniques for Nonlinear Time Series Models Boosting Techniques for Nonlinear Time Series Models. Technical Report 075, 2012.

[133] L. Rosasco, M. Santoro, S. Mosci, A. Verri, and S. Villa. A Regularization Approach to Nonlinear Variable Selection. *Journal of Machine Learning Research*, 9:653–660, 2010.

[134] Paul R. Rosenbaum and Donald B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

[135] Alberto G. Rossi and Allan Timmermann. Modeling covariance risk in Merton's ICAPM. *Review of Financial Studies*, 28(5):1428–1461, may 2015.

[136] Qifan Song and Faming Liang. High-Dimensional Variable Selection With Reciprocal L1-Regularization. *Journal of the American Statistical Association*, 110(512):1607–1620, 2015.

[137] Joseph E Stiglitz. Where modern macroeconomics went wrong. *Oxford Review of Economic Policy*, 34(1-2):70–106, 2018.

[138] Carolin Strobl, Anne Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. Conditional variable importance for Random Forests. *BMC Bioinformatics*, 9, 2008.

[139] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8, 2007.

[140] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[141] H Tong and K S Lim. Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 42(3):245–292, 1980.

[142] Fabian Valencia and Luc Laeven. Systemic banking crises: a new database. *IMF Economic Review*, 61(2):225–270, 2013.

[143] Stef van Buuren. Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research*, 16(3):219–242, 2007.

[144] Mark J. van der Laan and Daniel Rubin. Targeted maximum likelihood learning. *The International Journal of Biostatistics*, 2, 2006.

[145] Laurens van der Maaten and Geoffrey Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

[146] Mathijs Van Dijk. The social costs of financial crises. *Mimeo*, 2013.

[147] Hal R. Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, 2014.

[148] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, page 10961103. Association for Computing Machinery, 2008.

[149] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:33713408, 2010.

[150] Stefan Wager and Susan Athey. Estimation and Inference of Heterogeneous Treatment Effects using Random Forests. *Journal of the American Statistical Association*, 113(523):1228–1242, 2018.

[151] Ivo Welch and Amit Goyal. A comprehensive look at the empirical performance of equity premium prediction. *Review of Financial Studies*, 21-4:1455–1508, 2008.

[152] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, page 478487. JMLR.org, 2016.

[153] Lei Xu, Michael I. Jordan, and Geoffrey E. Hinton. An alternative model for mixtures of experts. *Conference on Neural Information Processing Systems*, 1994.

[154] Olcay Taner Yıldız, Ozan Írsoy, and Ethem Alpaydın. Bagging Soft Decision Trees. In *Machine Learning for Health Informatics*, volume 9605, pages 25–36. 2016.

[155] Ming Yuan and Yi Lin. On the non-negative garrotte estimator. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 69(2):143–161, 2007.

[156] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. Twenty Years of Mixture of Experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1177–1193, 2012.

[157] Zhi-hua Zhou and Ji Feng. Deep Forest: Towards an Alternative to Deep Neural Networks. In *Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[158] Hui Zou. The Adaptive Lasso and Its Oracle Properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006.

Appendix A

Appendix for Chapter 2

A.1 Proof for Theorem 7

Since we show that the soft decision tree $\hat{\mu}_{tree}(x)$ is equalvalent to a mixture model. Now assume we have a estimated mixture model with S components (kernels):

$$\hat{f}(x) = \frac{\sum_{s=1}^{S} K(x; \hat{\beta}_s, \hat{c}_s)\hat{\mu}_s}{\sum_{s=1}^{S} K(x; \hat{\beta}_s, \hat{c}_s)}$$

.

And the expectation is

$$E(\hat{f}(x)) = f(x) = \frac{\sum_{s=1}^{S} K(x; \beta_s, c_s)\mu_s}{\sum_{s=1}^{S} K(x; \beta_s, c_s)}$$

.

First, consider a simple case where $\beta_s = \beta$:

$$
\begin{aligned}
f(x) &= \frac{\sum_{s=1}^{S} K(x;\beta_s,c_s)\mu_s}{\sum_{s=1}^{S} K(x;\beta_s,c_s)} \\
&= \frac{\sum_{s=1}^{S} exp(-lg_2^S\beta(x-c_s)^2)\mu_s}{\sum_{s=1}^{S} exp(-lg_2^S\beta(x-c_s)^2)} \\
&= \frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)\mu_s}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)}
\end{aligned}
$$

We can see that the kernel is similar to a Gaussian kernel with bandwidth $h = 1/\sqrt{lg_2^S\beta}$.

Now, consider the bound of the bias:

$$
\begin{aligned}
&|f_0(x) - E(\hat{f}(x))| \\
&= |f_0(x) - f(x)| \\
&= \left| f_0(x) - \frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)\mu_s}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)} \right| \\
&= \frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)\mu_s}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)} |f_0(x) - \mu_s|
\end{aligned}
$$

In the original soft decision tree, μ_s is treated as a parameter to estimate. In our paper, we introduce the definition **honesty** from Wager and Athey (2018). We first randomly seperate the sample N into two equal sub samples $N_1 = N_2 = \frac{N}{2}$. Then, after using the first sub sample to estimate β_s and c_s, the second sub sample can be used for estimate the μ_s such that $\mu_s = E(\hat{\mu}(c_s)) = f_0(c_s)$.

Specifically, we estimate μ_s via a kernel regression. For example, using the exponential kernel function estimated in the soft decision tree:

$$\hat{\mu}_s(c_s) = \frac{\sum_{i=1}^{N/2} K(x_i; \beta_s, c_s) y_i}{\sum_{i=1}^{N/2} K(x_i; \beta_s, c_s)} = f_0(c_s) + \varepsilon.$$

where $K(x_i; \beta_s, c_s) = exp\left(-\left(\frac{x_i - c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)$.

Similar to the nonparameteric kernel method, it is easily to show that it converge to the true known function $f_0(x)$. The reason we choose this kernel function is related to the MLE method. We will show that in the Appendix A.2.

Plugging it into the previous equation, we have:

$$\begin{aligned}
&= \frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x - c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x - c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |f_0(x) - \hat{\mu}_s| \\
&= \frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x - c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x - c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |f_0(x) - f_0(c_s) - \varepsilon| \\
&= \frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x - c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x - c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |f_0(x) - f_0(c_s)| + \varepsilon
\end{aligned}$$

The next assumption we need is the so-called **Lipschitz continuity**. That is, we assume the target function $f_0(x)$ is changing slowly given finite support of x. That is, a Lipschitz continuous function is limited in how fast it can change:

$$|f_0(x) - f_0(x')| \leq D|x - x'|$$

Thus, our previous equation can be rewritten as follows:

$$\begin{aligned}
&\frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |f_0(x) - f_0(c_s)| + \varepsilon \\
&\leq \frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |x - c_s| + \varepsilon \\
&= (a) + \varepsilon
\end{aligned}$$

Let us focus on part (a):

$$\begin{aligned}
(a) &= D\frac{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |x - c_s| \\
&= D\frac{exp\left(-\left(\frac{x-c_1}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |x - c_1| + ... + D\frac{exp\left(-\left(\frac{x-c_S}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{x-c_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} |x - c_S|
\end{aligned}$$

Let $d_s = |x - c_s|$, we have

$$(a) = D\frac{exp\left(-\left(\frac{d_1}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{d_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} d_1 + ... + D\frac{exp\left(-\left(\frac{d_S}{1/\sqrt{lg_2^S \beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{d_s}{1/\sqrt{lg_2^S \beta}}\right)^2\right)} d_S$$

For the first component $\frac{exp\left(-\left(\frac{d_1}{1/\sqrt{lg_2^S\beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{d_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)}d_1$, let $d_{1*} = d_1/(1/\sqrt{lg_2^S\beta})$ we have:

$$= \frac{1}{\sqrt{lg_2^S\beta}} \frac{exp\left(-\left(d_{1*}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(d_{s*}\right)^2\right)} d_{1*} \tag{A.1}$$

To $\frac{1}{\sum_{s=1}^{S} exp\left(-(d_{s*})^2\right)}$, we have $\sum_{s=1}^{S} e\left(-\left(d_{s*}\right)^2\right) \to S\int exp^{-(x)^2}dx = \sqrt{2\pi}S$ when $S \to \infty$. Hense,

$$\begin{aligned} &= \frac{1}{\sqrt{lg_2^S\beta}} \frac{exp\left(-\left(d_{1*}\right)^2\right)}{\sqrt{2\pi}S} d_{1*} \\ &\leq \frac{C}{S\sqrt{lg_2^S\beta}} \end{aligned}$$

where $C = \frac{1}{\sqrt{2\pi}}$ is a constant. To part (a), we have

$$\begin{aligned} (a) &= D\frac{exp\left(-\left(\frac{d_1}{1/\sqrt{lg_2^S\beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{d_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)}d_1 + ... + D\frac{exp\left(-\left(\frac{d_S}{1/\sqrt{lg_2^S\beta}}\right)^2\right)}{\sum_{s=1}^{S} exp\left(-\left(\frac{d_s}{1/\sqrt{lg_2^S\beta}}\right)^2\right)}d_S \\ &\leq S \times \frac{C}{S\sqrt{lg_2^S\beta}} \\ &= \frac{C}{\sqrt{lg_2^S\beta}} \end{aligned}$$

Since $S \to \infty$, it is guaranteed that:

$$(a) \leq \frac{C}{\sqrt{lg_2^S\beta}} \asymp lg_2^S\beta^{-1/2} \to 0$$

Next, the second bias term ε is equivalent to the bias term in a nonparametric kernel regression with bandwidth $h = \sqrt{lg_2^S \beta}$. Thus, the second term has a convergence rate same as part (a).

Finally, we have the following conclusion. When $S \to \infty$:

$$|f_0(x) - E(\hat{f}(x))| \leq \frac{C}{\sqrt{lg_2^S \beta}} \asymp lg_2^S \beta^{-1/2} \to 0$$

Specifically, let $\beta = \frac{S}{lg_s^S}$, we have:

$$|f_0(x) - E(\hat{f}(x))| \leq \frac{C}{\sqrt{lg_2^S \beta}} = \frac{C}{\sqrt{lg_2^S \frac{S}{lg_s^S}}} \asymp S^{-1/2} \to 0$$

For the multivariable case, assume x is a p dimensional input vector. In the soft decision tree, we can choose a different kernel function:

$$e^{(-lg_2^S (x-c_s)^T \Sigma_\beta (x-c_s)))}$$

where Σ_β is a symmetric weighted matrix. Assume that each dimension of x is independent, we can simplify the kernel function as follows:

$$e^{(-lg_2^S (x-c_s)^T \Sigma_\beta lg_2^S (x-c_s)))}$$

$$= e^{(-lg_2^S (x-c_s)^T V \Lambda_\beta V^T lg_2^S (x-c_s)))},$$

where $\Lambda_\beta = \begin{pmatrix} \beta_1 & & & \\ & \beta_2 & & \\ & & \ddots & \\ & & & \beta_p \end{pmatrix}$ containing p eigenvalues and V is an orthonomal matrix. Thus, let $V^T(x - c_s) = (x - c_s)_V$, we have:

$$e^{(-lg_2^S(x-c_s)^T V \Lambda_\beta V^T (x-c_s)))}$$

$$= e^{(-lg_2^S \sum_{k=1}^p \beta_k (x_k - c_{ks})_V^2)}$$

The bias term should be as follows:

$$\begin{aligned} |f_0(x) - E(\hat{f}(x))| &= \frac{\sum_{s=1}^S e^{(-lg_2^S \sum_{k=1}^p \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^S e^{(-lg_2^S \sum_{k=1}^p \beta_k (x_k - c_{ks})_V^2))}} E\left|f_0(x) - \hat{\mu}_s\right| \\ &= \frac{\sum_{s=1}^S e^{(-lg_2^S \sum_{k=1}^p \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^S e^{(-lg_2^S \sum_{k=1}^p \beta_k (x_k - c_{ks})_V^2))}} E\left|f_0(x) - f_0(c_s)\right| + \varepsilon \end{aligned}$$

Considering the first term, according to the multivariate Lipchitz continuity, we choose the L_2 norm as the metric. The bias becomes:

$$
\begin{aligned}
|f_0(x) - E(\hat{f}(x))| &= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}} E\,|f_0(x) - f_0(c_s)| \\
&= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}} d_{L_2}(x, c_s) \\
&= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}} \sqrt{\sum_{k=1}^{p} (x_k - c_{ks})^2} \\
&= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}} \sqrt{(x - c_k)^T (x - c_s)} \\
&= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}} \sqrt{(x - c_k)^T V V^T (x - c_s)} \\
&= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}} \sqrt{(V^T (x - c_k))^T V^T (x - c_s)} \\
&= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (x_k - c_{ks})_V^2))}} \sqrt{\sum_{k=1}^{p} (x_k - c_{ks})_V^2} \\
&= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (d_{ks})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} \beta_k (d_{ks})_V^2))}} \sqrt{\sum_{k=1}^{p} (d_{ks})_V^2}
\end{aligned}
$$

Let $d_{ks*V} = \sqrt{\beta_k} d_{ksV}$, we have:

$$
\begin{aligned}
|f_0(x) - E(\hat{f}(x))| &= \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}} \sqrt{\sum_{k=1}^{p} \frac{1}{\beta_k} (d_{ks*})_V^2} \\
&\leq \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}} \sqrt{\frac{1}{\beta_{max}} \sum_{k=1}^{p} (d_{ks*})_V^2} \\
&= \frac{1}{\beta_{max}} \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}} \sqrt{\sum_{k=1}^{p} (d_{ks*})_V^2} \\
&= \frac{1}{\sqrt{lg_2^S \beta_{max}}} \frac{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}}{\sum_{s=1}^{S} e^{(-lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2))}} \sqrt{lg_2^S \sum_{k=1}^{p} (d_{ks*})_V^2}
\end{aligned}
\tag{A.2}
$$

Comparing the final result in Equation (A.1) with Equation (A.2), we find that the two results are the same. Thus, we can obtain a consistent result for the multiple case:

$$|f_0(x) - E(\hat{f}(x))| \leq \frac{C^*}{\sqrt{lg_2^S \beta_{max}} S^{(1/2p-1/2)}} \asymp \frac{1}{\sqrt{lg_2^S \beta_{max}} S^{(1/2p-1/2)}} \to 0,$$

where β_{max} is the max eigenvalue of Σ_β and $C^* = \frac{D}{\sqrt{2\pi}}$. Let $\beta_{max} = \frac{S}{lg_2^S}$, we have

$$|f_0(x) - E(\hat{f}(x))| \leq \frac{C^*}{\sqrt{lg_2^S \frac{S}{lg_2^S}} S^{(1/2p-1/2)}} \asymp S^{-1/2p} \to 0,$$

A.2 Proof for Theorem 9

Given the normal soft decision tree $\mu_{tree}(x;\theta)$. If the θ is estimated via QMLE and has the asymptotic normal distribution $\sqrt{n}(\hat{\theta} - \theta^*) \sim N(0, \Sigma_{\theta^*})$, the soft decision tree estimator is asymptotically distributed as:

$$(\hat{\mu}_{tree}(x;\theta) - E(\hat{\mu}_{tree}(x;\theta))) \to N(0, \sigma^2_{tree}(x)),$$

where $\sigma^2_{tree}(x) = \frac{1}{N} J(x)' \Sigma_{\theta^*} J(x) = \frac{1}{N} \begin{pmatrix} \frac{\partial \mu(x)}{\partial \theta_1} & \dots & \frac{\partial \mu(x)}{\partial \theta_m} \end{pmatrix} \begin{pmatrix} \sigma_1^2 & \cdots & \sigma_{1m} \\ \vdots & \ddots & \vdots \\ \sigma_{m1} & \cdots & \sigma_m^2 \end{pmatrix}$

However, to sample-splitting soft decision tree, the parameters are estimated on two seperate samples. Thus, the corresponding asymptotic distribution is:

$$(\hat{\mu}_{debiased\ tree}(x) - E(\mu(x))) \to N(0, \sigma^2_{debiased\ tree}(x)),$$

where $\sigma^2_{debiased\ tree}(x) = J(x)' \Sigma_{\theta^*} J(x)$

$$= \begin{pmatrix} \frac{\partial \mu(x)}{\partial \theta_1} & \cdots & \frac{\partial \mu(x)}{\partial \theta_m} \end{pmatrix} \begin{pmatrix} \frac{1}{wN}\sigma_1^2 & \cdots & \frac{1}{wN}\sigma_{1,m_1} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{wN}\sigma_{m_1,1} & \cdots & \frac{1}{wN}\sigma_{m_1}^2 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \frac{1}{(1-w)N}\sigma_{m_1+1}^2 & \cdots & \frac{1}{(1-w)N}\sigma_{m_1+1,m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \frac{1}{(1-w)N}\sigma_{m,m_1+1} & \cdots & \frac{1}{(1-w)N}\sigma_m^2 \end{pmatrix}$$

$$= \frac{1}{wN} \begin{pmatrix} \frac{\partial \mu(x)}{\partial \theta_1} & \cdots & \frac{\partial \mu(x)}{\partial \theta_{m_1}} \end{pmatrix} \begin{pmatrix} \sigma_1^2 & \cdots & \sigma_{1,m_1} \\ \vdots & \ddots & \vdots \\ \sigma_{m_1,1} & \cdots & \sigma_{m_1,m_1}^2 \end{pmatrix} \begin{pmatrix} \frac{\partial \mu(x)}{\partial \theta_1} \\ \vdots \\ \frac{\partial \mu(x)}{\partial \theta_{m_1}} \end{pmatrix}$$

$$+ \frac{1}{(1-w)N} \begin{pmatrix} \frac{\partial \mu(x)}{\partial \theta_{m_1+1}} & \cdots & \frac{\partial \mu(x)}{\partial \theta_m} \end{pmatrix} \begin{pmatrix} \sigma_{m_1+1}^2 & \cdots & \sigma_{m_1+1,m} \\ \vdots & \ddots & \vdots \\ \sigma_{m,m_1+1} & \cdots & \sigma_m^2 \end{pmatrix} \begin{pmatrix} \frac{\partial \mu(x)}{\partial \theta_{m_1+1}} \\ \vdots \\ \frac{\partial \mu(x)}{\partial \theta_m} \end{pmatrix}$$

$$= \frac{1}{N} \left(\frac{\sigma_1^2(x)}{w} + \frac{\sigma_2^2(x)}{1-w} \right)$$

Thus, the optimal weight w should depend on the two variances. To be simple, let $w = 1/2$, the variance of the debiased soft decision tree is $\sigma^2_{debiased\ tree}(x) = \frac{2}{N}\left(\sigma_1^2(x) + \sigma_2^2(x)\right)$. Precisely, based on the $\mu_{tree} = \frac{\sum_{s=1}^{S} e^{(-log_2^S(x-c_s)^T \Sigma_\beta (x-c_s))} \mu_s}{\sum_{s=1}^{S} e^{(-log_2^S(x-c_s)^T \Sigma_\beta (x-c_s))}}$, the upper bounds of the two variances are $\sigma_1^2(x) < O(lg_2^S \beta_{max} pS)$ and $\sigma_2^2(x) < O(lg_2^S \beta_{max} S)$. Hence,

$$\sigma^2_{debiased\ tree}(x) \sim \frac{2}{N} \times O(lg_2^S \beta_{max} pS) = O\left(\frac{lg_2^S \beta_{max} pS}{N}\right).$$

where S is the number of leaf of the soft decision tree. Let $\beta_{max} = \frac{S}{lg_2^S}$, we have

$$\sigma^2_{debiased\ tree}(x) \sim \frac{2}{N} \times O(pS^2) = O\left(\frac{pS^2}{N}\right).$$

A.3 Proof for Theorem 10

Based on the bounded bias from Theorem 7 and asymptotic distribution in Theorem 9, we can derive the consistency of soft decision tree via Chebyshev's inequality:

$$\begin{aligned} P(|\hat{\mu}_{tree}(x) - f(x)| \geq \epsilon) &\leq \frac{E(\hat{\mu}_{tree}(x) - f(x))^2}{\epsilon^2} \\ &= \frac{1}{\epsilon^2}((E(\hat{\mu}_{tree}(x)) - f(x))^2 + Var(\hat{\mu}_{tree}(x))) \\ &\sim O(S^{-1/p}) + O\left(\frac{pS^2}{N}\right) \to 0 \end{aligned} \tag{A.3}$$

Let $S = N^\alpha$, we have

$$\begin{aligned} P(|\hat{\mu}_{tree}(x) - f(x)| \geq \epsilon) &\sim O(N^{-\alpha/p}) + O(\frac{pN^{2\alpha}}{N}) \\ &\sim O(N^{-\alpha/p}) + O(pN^{2\alpha-1}) \end{aligned} \tag{A.4}$$

The convergence rate is optimal when two terms have same convergence rate. Thus, when $p = p_0$, the condition of optimal convergence is:

$$-\alpha/p_0 = 2\alpha - 1$$

$$\alpha/p_0 + 2\alpha = 1$$

$$\alpha^* = \frac{1}{(1/p_0 + 2)}$$

Thus, the optimal convergence rate is $O(N^{-\frac{1}{1+2p_0}})$. Next, we can get

$$\begin{aligned}
\frac{\hat{\mu}_{debiased\ tree}(x) - f(x)}{\sqrt{Var(\hat{\mu}_{debiased\ tree}(x))}} &= \frac{\hat{\mu}_{debiased\ tree}(x) - E(\hat{\mu}_{debiased\ tree}(x))}{\sqrt{Var(\hat{\mu}_{debiased\ tree}(x))}} + \frac{E(\hat{\mu}_{debiased\ tree}(x)) - f(x)}{\sqrt{Var(\hat{\mu}_{debiased\ tree}(x))}} \\
\rightarrow \frac{\hat{\mu}_{tree}(x) - f(x)}{\sigma_{debiased\ tree}(x)} &= \frac{\hat{\mu}_{debiased\ tree}(x) - E(\hat{\mu}_{debiased\ tree}(x))}{\sigma_{debiased\ tree}(x)} + \frac{E(\hat{\mu}_{debiased\ tree}(x)) - f(x)}{\sigma_{debiased\ tree}(x)} \\
&\sim N(0,1) + \frac{O(N^{-\alpha/2p_0})}{O(\sqrt{p_0}N^{(2\alpha-1)/2})} \\
&= N(0,1) + O(N^{-\alpha/2p_0-(2\alpha-1)/2}) \\
&= N(0,1) + O(N^{-(\alpha/2p_0+\alpha+1/2)}) \\
&= N(0,1) + O(N^{-(\alpha(1/2p_0+1)+1/2)})
\end{aligned}$$

Thus, given $\frac{1}{1/p_0+2} < \alpha < 1$, the asymptotic bias term $\frac{E(\hat{\mu}_{debiased\ tree}(x))-f(x)}{\sigma_{debiased\ tree}(x)} = O(N^{-(\alpha(1/2p_0+1)+1/2)}) \rightarrow 0$ and then the asymptotic normality of $\hat{\mu}_{tree}(x)$ is:

$$\frac{\hat{\mu}_{debiased\ tree}(x) - f(x)}{\sigma_{debiased\ tree}(x)} \rightarrow N(0,1) \tag{A.5}$$

A.4 Proof for Theorem 11

Based on our previous discussions, we have the following conclusions:

Based on our previous discussion, we have the following conclusions:

1 the bias of $\hat{\mu}_{tree}(x)$ is

$$|E(\hat{\mu}_{tree}(x)) - f(x)| < O(S^{-1/2p})$$

2 the asymptotic distribution of $\hat{\mu}_{tree}(x)$is:

$$\frac{\hat{\mu}_{tree}(x) - E(\hat{\mu}_{tree}(x))}{\sqrt{Var(\hat{\mu}_{tree}(x))}} \to N(0,1)$$

where $Var(\hat{\mu}_{tree}(x)) \to O(\frac{pS^2}{N})$

Take Taylor expension on the bias:

$$\begin{aligned} |E(\hat{\mu}_{tree}(x) - f(x)| &= |\mu^*_{tree}(x) - f(x)| \\ &= |\mu^*_{tree}(x;\theta_0) + (\theta^* - \theta_0)^T \frac{\mu^*_{tree}}{\theta} - f(x)| \\ &= |(\theta^* - \theta_0)^T \frac{\partial \mu^*_{tree}}{\partial \theta}|_{\tilde{\theta}_0}| < O(S^{-1/2p}). \end{aligned}$$

where we assume that there exists a θ_0 such that $\mu^*_{tree}(x;\theta_0) = f(x)$ and $\tilde{\theta}_0 \in (\theta^*, \theta_0)$. Thus, the norm of the vector of bias $\beta = \theta^* - \theta_0 = E(\hat{\theta}) - \theta_0$ should also be bounded by $O(S^{-1/p})$.

Thus, to the parameter estimator $\hat{\theta}$, we have:

1 the bias $\beta = \theta^* - \theta_0$ satisfies:

$$|(\theta^* - \theta_0)^T \frac{\partial \mu^*_{tree}}{\partial \theta}|_{\theta_0}| = \beta^T \frac{\partial \mu^*_{tree}}{\partial \theta}|_{\tilde{\theta}_0} < O(S^{-1/p})$$

2 the asymptotic distribution of $\hat{\theta}$is:

$$\sqrt{N}(\hat{\theta} - E(\hat{\theta})) \sim N(0, \Sigma_\theta)$$

First, let us consider the estimator of the L_2 norm of the first derivative $||\frac{\partial \hat{f}}{\partial x_p}||_2^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial \hat{f}}{\partial x_p}|_{x=x_i}\right)^2$. Based on the asymptotic normality of the estimator of SDT's parameters $\sqrt{N}(\hat{\theta} - E(\hat{\theta})) \sim N(0, \Sigma_\theta)$, we can obtain:

1 Asymptotic distribution:

$$\frac{\left(\widehat{f'_p|_i} - E(\widehat{f'_p|_i})\right)}{\sqrt{Var(\widehat{f'_p|_i})}} \sim N(0,1)$$

2 Bias β:

$$\left| f'_p|_i - E(\widehat{f'_p|_i}) \right| = \beta^T J(f'_{p,\theta|i}) < O(S^{-1/2p}).$$

where $f'_p|_i = (\frac{\partial f}{\partial x_p}|_{x=x_i})$, $Var(\widehat{f'_p|_i}) = \frac{1}{N} J(f'_{p|i,\theta})^T \Sigma_\theta J(f'_{p|i,\theta}) \to O(\frac{pS^2}{N})$ and $J(f'_{p,\theta|i}) = \frac{\partial f'_p|_i)}{\partial \theta}$.

To $||\frac{\partial \hat{f}}{\partial x_p}^2||_2^2$, we have:

1 Asymptotic distribution:

$$\widehat{f'_p|_i} \sim N\left(E(\widehat{f'_p|_i}), Var(\widehat{f'_p|_i})\right)$$

$$\sum_{i=1}^{N} \frac{\widehat{f'_p|_i}^2}{Var(\widehat{f'_p|_i})} \sim non\chi^2$$

where $non\chi^2$ is an non central Chi square distribution. Thus we have:

$$E\left(\sum_{i=1}^{N} \frac{\widehat{f'_p|_i}^2}{Var(\widehat{f'_p|_i})}\right) = N + \sum_{i=1}^{N} \frac{E^2(\widehat{f'_p|_i})}{Var(\widehat{f'_p|_i})}.$$

$$E\left(\frac{1}{N}\sum_{i=1}^{N} \widehat{f'_p|_i}^2\right) = \bar{Var}(\widehat{f'_p}) + \frac{1}{N}\sum_{i=1}^{N} E^2(\widehat{f'_p|_i}).$$

$$Var\left(\sum_{i=1}^{N} \frac{\widehat{f'_p|_i}^2}{Var(\widehat{f'_p|_i})}\right) = 2N + 4\sum_{i=1}^{N} \frac{E^2(\widehat{f'_p|_i})}{Var(\widehat{f'_p|_i})}.$$

$$Var\left(\frac{1}{N}\sum_{i=1}^{N}\widehat{f'_p|_i}^2\right) = 2\bar{Var}^2(\widehat{f'_p}) + \frac{4\bar{Var}(\widehat{f'_p})}{N}\sum_{i=1}^{N}E^2(\widehat{f'_p|_i}).$$

To sum up, we know that $\frac{1}{N}\sum_{i=1}^{N}\widehat{f'_p|_i}^2 - E\left(\frac{1}{N}\sum_{i=1}^{N}\widehat{f'_p|_i}^2\right) \sim O_P(\frac{pS^2}{N})$.

2 Bias $\beta(f'_p)$:

$$\left|f'^2_p|_i - E(\widehat{f'_p|_i}^2)\right| = \beta^T J(f''^2_\theta)$$

$$\left|\sum_i f'^2_p|_i - \sum_i E(\widehat{f'_p|_i}^2)\right| = \sum_i \beta^T J(f''^2_{\theta|i})$$

$$\left|\frac{1}{N}\sum_i f'^2_p|_i - E\left(\frac{1}{N}\sum_{i=1}^{N}\widehat{f'_p|_i}^2\right)\right| = \beta^T \bar{J}(f''^2_\theta)$$

which means $\left|\frac{1}{N}\sum_i f'^2_p|_i - E\left(\frac{1}{N}\sum_{i=1}^{N}\widehat{f'_p|_i}^2\right)\right| < O(S^{-1/2p})$

Now, we consider the penalized most likelihood function:

$$L_R = L(y, x; \theta) - \lambda R_\lambda(f') \tag{A.6}$$

where $L(y, x; \theta) = \sum_{i=1}^{N} Log \sum_{s=1}^{2} \alpha_s(x_i; \theta) P_s(y_i|x_i; \theta)$ and

$$R_\lambda(f') = \sum_{p=1}^{P} \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial f}{\partial x_p}|_{x=x_i}\right)^2}}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial \hat{f}}{\partial x_p}|_{x=x_i}\right)^{2^\gamma}}} = \sum_{p=1}^{P} R^P_\lambda(f')$$

By Taylor's expansion, we have:

$$
\begin{aligned}
L_R(y, x; \theta) &= L(y, x; \theta) - \lambda R_\lambda(f') \\
&= L(y, x; \theta_0) + (\theta - \theta_0)^T \frac{\partial L}{\partial \theta}|_{\theta_0} + 1/2(\theta - \theta_0)^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(\theta - \theta_0) - \lambda R_\lambda(f')
\end{aligned} \tag{A.7}
$$

Additionally, we set $\theta = \theta_0 + \frac{u}{\sqrt{N}} + \frac{\beta}{\sqrt{N}}$, where β is a bias vector. Then, plug θ in:

$$
\begin{aligned}
-L_R(y, x; \theta) + L_R(y, x; \theta_0) &= -(\theta - \theta_0)^T \frac{\partial L}{\partial \theta}|_{\theta_0} - 1/2(\theta - \theta_0)^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(\theta - \theta_0) \\
&= Part(1) + Part(2)
\end{aligned} \tag{A.8}
$$

For $Part(1)$, we have:

$$
\begin{aligned}
Part(1) &= -(\theta - \theta_0)^T \frac{\partial L}{\partial \theta}|_{\theta_0} - 1/2(\theta - \theta_0)^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(\theta - \theta_0) \\
&= -(u/\sqrt{N} + \beta/\sqrt{N})^T \frac{\partial L}{\partial \theta}|_{\theta_0} - 1/2(u/\sqrt{N} + \beta/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(u/\sqrt{N} + \beta/\sqrt{N}) \\
&= -(u/\sqrt{N})^T \frac{\partial L}{\partial \theta}|_{\theta_0} - (\beta/\sqrt{N})^T \frac{\partial L}{\partial \theta}|_{\theta_0} \\
&- 1/2(u/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(u/\sqrt{N}) + 1/2(u/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(\beta/\sqrt{N}) \\
&+ 1/2(\beta/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(u/\sqrt{N}) - 1/2(\beta/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(\beta/\sqrt{N})
\end{aligned} \tag{A.9}
$$

We found that $\frac{\partial L}{\partial \theta}|_{\theta_0}$ is the score function of the log-likelihood function and that $\frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}$ is the negative fisher information matrix. Thus, since $\frac{1}{\sqrt{n}} \frac{\partial L}{\partial \theta}|_{\theta_0} \to N(0, I(\theta))$ and $\frac{1}{n} \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0} \to I(\theta)$, we have:

$$
\begin{aligned}
Part(1) &= -(u/\sqrt{N})^T \frac{\partial L}{\partial \theta}|_{\theta_0} - (\beta/\sqrt{N})^T \frac{\partial L}{\partial \theta}|_{\theta_0} \\
&\quad - 1/2(u/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(u/\sqrt{N}) + 1/2(u/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(\beta/\sqrt{N}) \\
&\quad + 1/2(\beta/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(u/\sqrt{N}) - 1/2(\beta/\sqrt{N})^T \frac{\partial L^2}{\partial \theta \partial \theta^T}|_{\theta_0}(\beta/\sqrt{N}) \\
&\quad \to -u^T W - \beta^T W + 1/2 u^T I(\theta) u - 1/2 u^T I(\theta)\beta - 1/2\beta^T I(\theta) u + 1/2\beta^T I(\theta)\beta
\end{aligned}
\tag{A.10}
$$

where $W \sim N(0, I(\theta))$. Then, $part(1)$ takes the maximum value when $u = I(\theta)^{-1}W + \beta \to N(0, I(\theta)^{-1})$ when $\beta \to 0$.

For $Part(2)$, let $\frac{1}{N}\sum_i \hat{f}_p'^2|_i = \frac{1}{N}\sum_i f_p'^2|_i + \frac{U_f}{\frac{pS^2}{N}} + \frac{\beta_f}{\frac{pS^2}{N}}$ since $\sqrt{Var(\frac{1}{N}\sum_i \hat{f}_p'^2|_i)} \sim O\left(\frac{pS^2}{N}\right)$ we have:

$$
\begin{aligned}
&\lambda\left(R_\lambda^P(f') - R_\lambda^P(f')_0\right) \\
&= \lambda\left(\frac{\sqrt{\frac{1}{N}\sum_i \hat{f}_p'^2|_i}}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f}'^2_p|_i}\right)^\gamma} - \frac{\sqrt{\frac{1}{N}\sum_i f_p'^2|_i}}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f}'^2_p|_i}\right)^\gamma}\right) \\
&= \lambda\left(\frac{\sqrt{\frac{1}{N}\sum_i f_p'^2|_i + \frac{U_f}{\frac{4pS^2}{N}} + \frac{\beta_f}{\frac{4pS^2}{N}}}}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f}'^2_p|_i}\right)^\gamma} - \frac{\sqrt{\frac{1}{N}\sum_i f_p'^2|_i}}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f}'^2_p|_i}\right)^\gamma}\right) \\
&= \frac{\lambda}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f}'^2_p|_i}\right)^\gamma}\left(\sqrt{\frac{1}{N}\sum_i f_p'^2|_i + \frac{U_f}{\frac{4pS^2}{N}} + \frac{\beta_f}{\frac{4pS^2}{N}}} - \sqrt{\frac{1}{N}\sum_i f_p'^2|_i}\right)
\end{aligned}
\tag{A.11}
$$

Considering different cases, we have:

$$= \begin{cases} \frac{\lambda}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f'}_p^2|_i}\right)^\gamma}\sqrt{\frac{U_f}{\sigma(f)}+\frac{\beta_f}{\sigma(f)}} \to \frac{\lambda\left(\frac{pS^2}{N}\right)^{\gamma/2}}{\sqrt{\frac{pS^2}{N}}} \to \infty & if: \sqrt{\frac{1}{n}\sum_i f_p'^2|_i} = 0 \\ 0 & if: \sqrt{\frac{1}{n}\sum_i f_p'^2|_i} = 0 \\ \frac{\lambda}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f'}_p^2|_i}\right)^\gamma}\frac{\left(\frac{U_f}{\sigma(f)}+\frac{\beta_f}{\sigma(f)}\right)}{2\sqrt{\tilde{f}_p}} \to \lambda\sqrt{\frac{pS^2}{N}} \to 0 & if: \sqrt{\frac{1}{n}\sum_i f_p'^2|_i} \neq 0 \end{cases} \tag{A.12}$$

where $\tilde{f}_p \in (\frac{1}{N}\sum_i f_p'^2|_i + \frac{U_f}{\sigma(f)} + \frac{\beta_f}{\sigma(f)}, \frac{1}{N}\sum_i f_p'^2|_i)$.

Now, considering $L_R(y,x;\theta) - L_R(y,x;\theta_0)$, we have:

$$\begin{aligned} -L_R(y,x;\theta) + L_R(y,x;\theta_0) &= -(\theta-\theta_0)^T\frac{\partial L}{\partial\theta}|_{\theta_0} - 1/2(\theta-\theta_0)^T\frac{\partial L^2}{\partial\theta\partial\theta^T}|_{\theta_0}(\theta-\theta_0) \\ &= -u^TW - \beta^TW + 1/2u^TI(\theta)u - 1/2u^TI(\theta)\beta \\ &\quad - 1/2\beta^TI(\theta)u + 1/2\beta^TI(\theta)\beta + Part(2) \end{aligned} \tag{A.13}$$

Let $V_n(u) = -L_R(y,x;\theta) + L_R(y,x;\theta_0)$, and we have:

$$V(u) = \begin{cases} -u^TW - \beta^TW + u^TI(\theta)u/2 - u^TI(\theta)\beta/2 - \beta^TI(\theta)u/2 + \beta^TI(\theta)\beta/2 & if :\neq 0 \\ \infty & otherwise \end{cases} \tag{A.14}$$

By Slutskys theorem, we have $V_n(u) \to V(u)$. Thus, we have:

$$\hat{u} \to I(\theta)^{-1}W + \beta$$

Intuitively, when $f_p' = 0$ is satisfied, the variable should not belong to the true variable set A. For the true function $f(x_1, ..., x_{p-1})$, not only the first derivative f_p' should be zero, but the derivative of f_p' for parameter θ: $f_{p,\theta}''$ should also be zero, which means $f(x)$ does not contain anything about x_p and any parameters θ are not related to f_p'.

In summary, we have approved the part of asymptotic normality:

$$\frac{\hat{f}(x) - f(x)}{\sqrt{Var(\hat{f}(x))}} \sim N(0,1)$$

where $Var(\hat{f}(x)) = J_\theta(x)^T \Sigma_{\theta*} J_\theta(x) \sim O\left(\frac{pS^2}{N}\right)$.

Next, we consider the consistency of variable selection.

From the asymptotic result, we can obtain:

$$P(\hat{f}'_p \neq 0 | p \in A) \to 1$$

Thus, these variables should satisfy the following:

$$\frac{\partial L}{\partial \theta_s} = \lambda \sum_{p=1}^{P} \frac{1}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f'}_p|_i^2}\right)^\gamma} \frac{\frac{1}{N}\sum_{i=1}^N \hat{f}'_p|_i \hat{f}''_{p,\theta_s}|_i}{\sqrt{\frac{1}{N}\sum_{i=1}^N \hat{f}'_p|_i^2}}$$

where $\hat{f}''_{p,\theta_s}|_i = \frac{\hat{\partial} f'_p|_i}{\partial \theta_s}$.

Previously, we showed that

$$\frac{\left(\widehat{f'_p|_i} - f'_p|_i)\right)}{\sqrt{Var(\widehat{f'_p|_i})}} \to N(0,1)$$

Similarly, it is obvious to have the asymptotic result for $\hat{f}''_{p,\theta_s}|_i$:

$$\frac{\left(\widehat{f''_{p,\theta_s}|_i} - f''_{p,\theta_s}|_i)\right)}{\sqrt{Var(\widehat{f''_{p,\theta_s}|_i})}} \to N(0,1)$$

In terms of $\sqrt{Var(\widehat{f''_{p,\theta_s}|_i})} \sim O\left(\frac{pS^2}{N}\right)$ and $\sqrt{Var(\widehat{f''_{p,\theta_s}|_i})} \sim O\left(\frac{pS^2}{N}\right)$, it is simply to show that $\widehat{f'_p|_i} \sim O_p\left(\sqrt{\frac{pS^2}{N}}\right)$ and $\widehat{f''_{p,\theta_s}|_i} \sim O_p\left(\sqrt{\frac{pS^2}{N}}\right)$. Thus, we can have:

$$\frac{1}{N}\sum_{i=1}^{N}\hat{f}'_p|_i\hat{f}''_{p,\theta_s}|_i \sim O_p\left(\frac{pS^2}{N}\right).$$

Combined with $\frac{\lambda\left(\frac{pS^2}{N}\right)^{\gamma/2}}{\sqrt{\left(\frac{pS^2}{N}\right)}} \to \infty$, since $\hat{f}'_p = 0$, we have:

$$\sum_{p=1}^{P}\frac{\lambda}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f}'_p|_i^2}\right)^{\gamma}}\frac{\frac{1}{N}\sum_{i=1}^{N}\hat{f}'_p|_i\hat{f}''_{p,\theta_s}|_i}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\hat{f}'_p|_i^2}} \to \frac{\lambda\left(\frac{pS^2}{N}\right)^{\gamma/2}}{\sqrt{\left(\frac{pS^2}{N}\right)}} \to \infty.$$

Since $\frac{\partial L}{\partial \theta_s} \to 0$, we can see that:

$$P(\hat{f}'_p = 0|p \in A) = P\left(-\frac{\partial L}{\partial \theta_s} = \sum_{p=1}^{P}\frac{\lambda}{\left(\sqrt{\frac{1}{N}\sum_i \hat{f}'_p|_i^2}\right)^{\gamma}}\frac{\frac{1}{N}\sum_{i=1}^{N}\hat{f}'_p|_i\hat{f}''_{p,\theta_s}|_i}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\hat{f}'_p|_i^2}}\right) \to 0$$

The oracle properties are proved.