Bootstrap Aggregating and Random Forest

Tae-Hwy Lee, Aman Ullah and Ran Wang

Abstract Bootstrap Aggregating (Bagging) is an ensemble technique for improving the robustness of forecasts. Random Forest is a successful method based on Bagging and Decision Trees. In this chapter, 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.

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. First, we introduce Bagging and some variants. Second, we discuss Decision Trees in details. Then, we move to Random Forest which is one of the most attractive machine learning algorithms combining Decision Trees and Bagging. Finally, 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.

Tae-Hwy Lee
Department of Economics, University of California, Riverside, e-mail: `tae.lee@ucr.edu`

Aman Ullah
Department of Economics, University of California, Riverside, e-mail: `aman.ullah@ucr.edu`

Ran Wang
Department of Economics, University of California, Riverside, e-mail: `ran.wang@email.ucr.edu`

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.

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 $y_i \in \mathbb{R}$ is the dependent variable and $x_i \in \mathbb{R}^p$ are 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

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

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)

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

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. At the same time, 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.

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 computations 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 (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.

2.3 *Bootstrap robust aggregating (Bragging)*

In Sections 2.1 and 2.2, we have 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.

2.4 *Out-of-Bag Error for Bagging*

In Sections 2.1 to 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. This is discussed in Breiman (1996) in detail. The bth Out-of-Bag error is calculated by

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

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

- Based on the sample, we generate B different bootstrap samples $\{(y^b_1,x^b_1),\ldots,(y^b_N,x^b_N)\}$ via randomly drawing with replacement.
- To each bootstrap sample, estimate $\hat{f}_b(x)$ via minimizing the Loss function

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

- Compare the bth bootstrap sample to the original sample to get the the bth Out-of-Bag sample $\{(y^b_{1,OOB},x^b_{1,OOB}),\ldots,(y^b_{N_b,OOB},x^b_{N_b,OOB})\}$, 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

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

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, we discuss the Random Forest in detail. Several effective variants of Random Forest are discussed in detail in Section 5.

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 13.1 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.

First of all, let us explore the structure of the decision tree and clarify the names of components in the decision tree. Figure 13.2 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

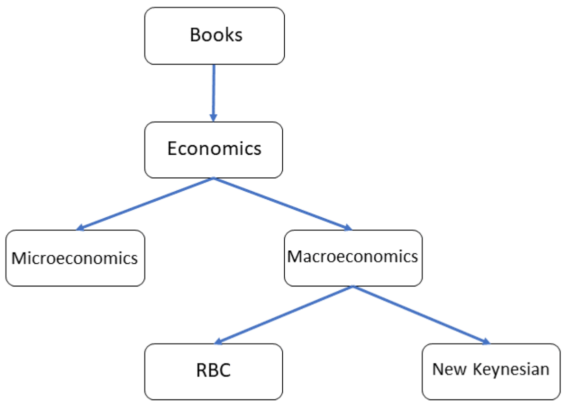


Fig. 1 A Tree of Structured Data about Economic Books

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 calculate 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

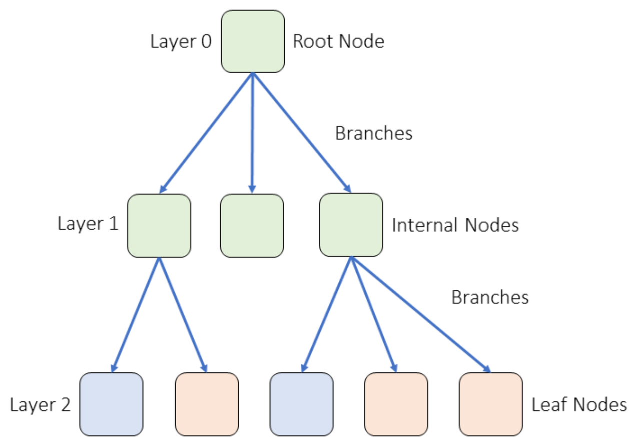


Fig. 2 The Components in a Decision Tree

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

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:

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

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.

In Figure 13.3, 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

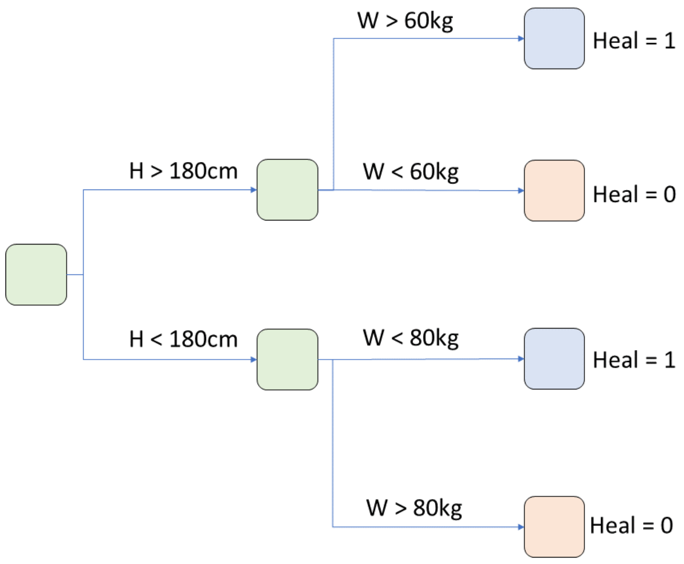


Fig. 3 A Tree of People's Health

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:

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

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.

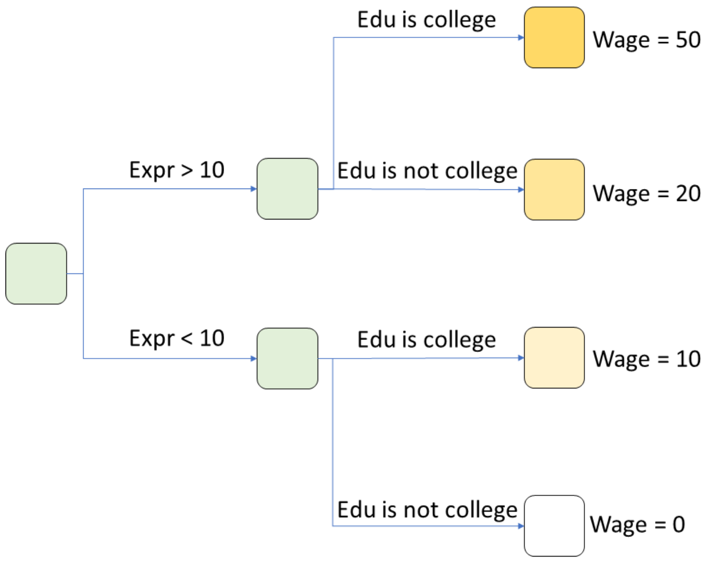


Fig. 4 A Tree of Woman's Wage

Figure 13.4 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 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.

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

In Section 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 subsections 3.3 and 3.4, we will introduce the Classification and Regression Tree (CART) method for the classification problem and the regression problem, respectively.

Let us go back to the weight, height and health example. 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 $Heal = 1$.

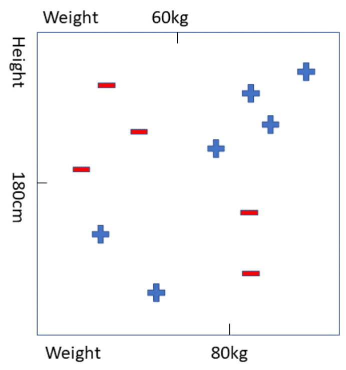
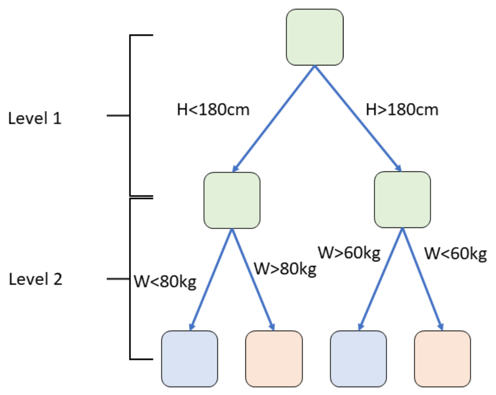


Fig. 5 Health Data in 2D Plot

Figure 13.6 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.



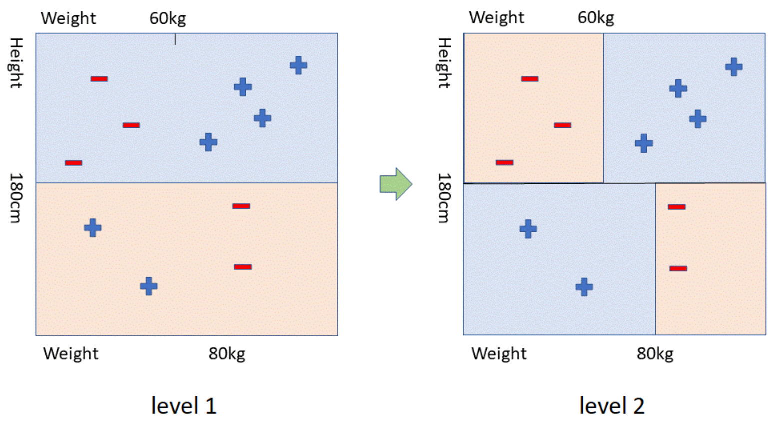


Fig. 6 Grow a Tree for Health Data

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 calculates 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.

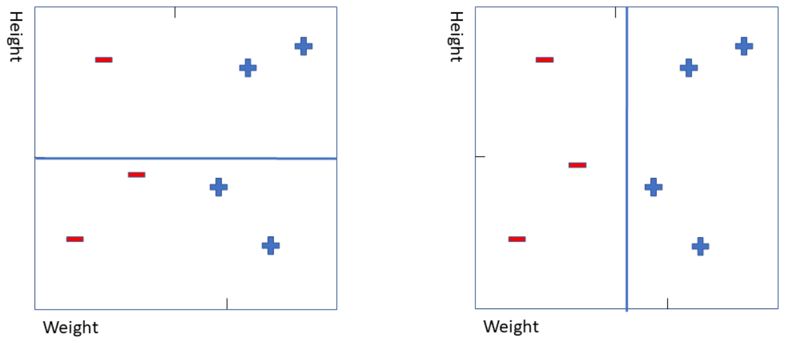


Fig. 7 Sub-spaces Generated by Decision Stumps

Figure 13.7 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

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

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

$$p_c = \frac{1}{N_S} \sum_{x_i \in S} I(y_i = c), \tag{2}$$

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

$$Info_D(S) = \sum_{j=1}^{v} \frac{N_{S_j}}{N_S} \times Info(S_j), \tag{3}$$

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

$$Gain(D) = Info(S) - Info_D(S). \tag{4}$$

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 13.8, 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 (13.1), 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

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

Based on Equation (13.3), the average entropy of S after splitting is

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

After splitting, the information decreases from 1 to 0.69. According to Equation (13.3), the information gain of D is

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

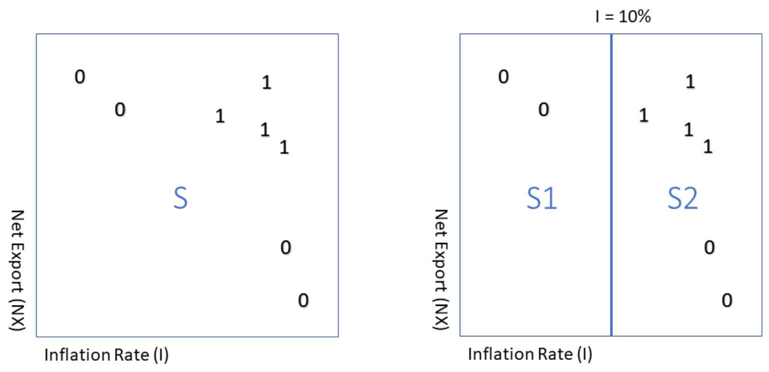


Fig. 8 Plots for Economic Growth Data

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 $x_i \in \mathbb{R}^p$. 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**

$$Split\ Info_D(S) = -\sum_{j=1}^{v} \frac{N_{S_j}}{N_S} \times \log_2 \frac{N_{S_j}}{N_S}, \tag{5}$$

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.

To the left case, the splitting information is calculated based on Equation (13.5) 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

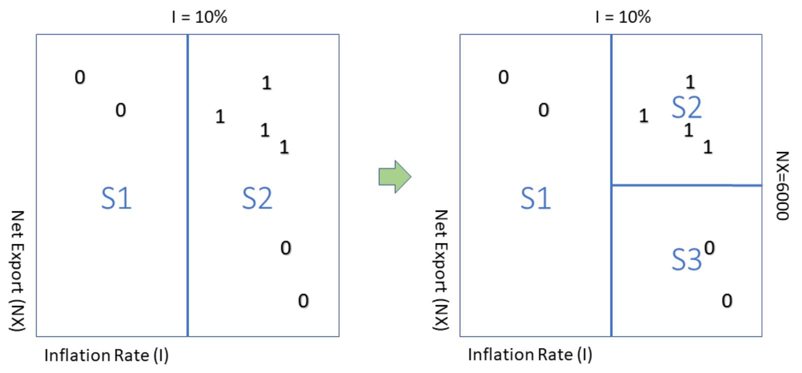


Fig. 9 Grow a Tree for Economic Growth Prediction

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

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)**

$$
Gain\ Ratio(D) = \frac{Gain(D)}{Split\ Info(D)}. \tag{6}
$$

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 $x_i \in \mathbb{R}^p$. 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

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

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

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

- 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.

3.3 Growing a decision tree for classification: CART

In Section 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.

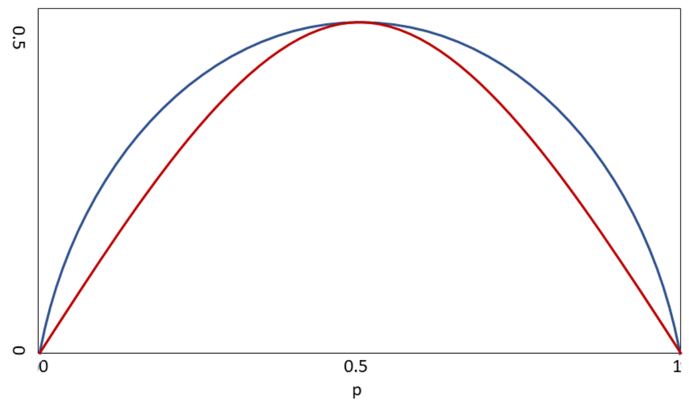


Fig. 10 Entropy (blue) and Gini Impurity (red)

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

$$Gini(S) = \sum_{j=1}^{M} p_j(1-p_j) = 1 - \sum_{j=1}^{M} p_j^2, \tag{7}$$

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 13.10 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

$$Gini_D(S) = \frac{N_{S_1}}{N_S} Gini(S_1) + \frac{N_{S_2}}{N_S} Gini(S_2). \tag{8}$$

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

$$\Delta Gini_D(S) = Gini(S) - Gini_D(S). \tag{9}$$

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

$$Total\ Cost = Measure\ of\ Fit + Measure\ of\ Complexity. \tag{10}$$

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 $x_i \in \mathbb{R}^p$. 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.

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

$$Variation(S) = \sum_{i=1}^{N} (y_i - \bar{y})^2, \tag{11}$$

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

$$Variation_D(S) = \frac{1}{v}\sum_{j=1}^{v} Variation_j(S), \tag{12}$$

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

$$Gain(D) = Variation(S) - Variation_D(S). \tag{13}$$

Based on the total cost in Equation (13.10), 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 $y_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^p$. 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, calculate 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.

3.5 Variable importance in a decision tree

In Sections 3.2 to 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, Breiman et al (1984) defined a **relative importance** as

$$I_j^2 = \sum_{t=1}^{T-1} e_t^2 I(v(t) = j), \tag{14}$$

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 calculate 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 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.

4.1 Constructing a random forest

As we discussed in Section 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

$$Var(g) = Var\left(\frac{1}{B}\sum_{b=1}^{B} f_b\right) = \frac{1}{B}\sigma^2. \tag{15}$$

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

$$\begin{aligned}
Var(g) &= \frac{1}{B^2} Var\left(\sum_{b=1}^{B} f_b\right) \\
&= \frac{1}{B^2}\left(\sum_{b=1}^{B} Var(f_b) + 2\sum_{b\neq c} cov(f_b, f_c)\right) \\
&= \frac{1}{B^2}(B\sigma^2 + (B^2 - B)\rho\sigma^2) \\
&= \rho\sigma^2 + \frac{(1-\rho)}{B}\sigma^2,
\end{aligned} \tag{16}$$

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.

4.2 Variable importance in a random forest

In Section 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

$$I_j^2 = \frac{1}{B} \sum_{b=1}^{B} I_j^2(b), \tag{17}$$

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

$$I_j^2(b) = \sum_{t=1}^{T_b - 1} e_t^2 I(v(t)_b = j). \tag{18}$$

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 permuting**. That is, for one variable, we perturb the samples by random permuting. 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 permuting. Intuitively, if one variable is not important, comparing the test error on the original test sample, the test error on permuting 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 permuting is

$$
\begin{aligned}
VI_j &= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_t}\sum_{i=1}^{N_t} Loss(y_i, tree_b(x_{1,i},....,x^*_{j,i},...)) - Loss(y_i, tree_b(x_{1,i},....,x_{j,i},...)) \\
&= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_t}\sum_{i=1}^{N_t} \Delta Loss(y_i, tree_b(x_{1,i},....,x^*_{j,i},...)).
\end{aligned} \tag{19}
$$

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 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

$$
\begin{aligned}
VI_j^{OOB} &= \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N_b}\sum_{i=1}^{N_b} (Loss(y_{i,OOB}, tree_b(x_{1,i,OOB},....,x^*_{j,i,OOB},...)) \\
&\qquad - Loss(y_{i,OOB}, tree_b(x_{1,i,OOB},....,x_{j,i,OOB},...;tree_b))) \\
&= \frac{1}{B}\sum_{b=1}^{B}\left(\widehat{err}^*_{OOB,b} - \widehat{err}_{OOB,b}\right) \\
&= \frac{1}{B}\sum_{b=1}^{B}\Delta\widehat{err}_{OOB,b}
\end{aligned} \tag{20}
$$

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.
- Calculate OOB error for the bth decision tree based on the OOB sample with and without random permuting.
- 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. Recently, Strobl et al (2008) and Janitza et al (2016) considered the hypothesis testing to select variables in Random Forest.

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

$$\int_R P(+1|z)P(dz) \geq \int_R P(-1|z)P(dz), \tag{21}$$

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 (13.21), we have the output $+1$ from a decision tree given an input x when Equation (13.22) holds

$$\int_{R_x(\theta)} P(1|z)P(dz) \geq \int_{R_x(\theta)} P(-1|z)P(dz), \tag{22}$$

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 (13.22) can be written as

$$\int_{R_x(\theta)} D(z)P(dz) \geq 0. \tag{23}$$

According to Equation (13.23), 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

$$K_{RF}(x,z) = \frac{1}{B} \sum_{b=1}^{B} \sum_{t=1}^{T_b} I(x, z \in R^t(\theta_b)). \tag{24}$$

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

$$\begin{aligned} K_{RF}(x,z) &= \frac{1}{B} \sum_{b=1}^{B} \sum_{t=1}^{T_b} I(x, z \in R^t(\theta_b)) \\ &\to P_\theta(x, z \in A(\theta)), \end{aligned} \tag{25}$$

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 13.11 illustrates the difference between Decision Tree and KNN.

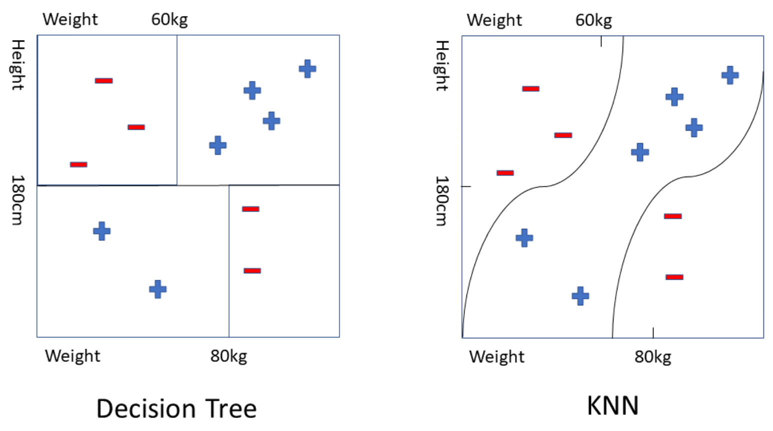


Fig. 11 2D Plot of Decision Tree and KNN

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.

5.1 Extremely randomized trees

For Bagging method, we discussed its effectiveness related to the variance of ensemble model. According to Equation (13.16)

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

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 et al (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.
- 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 et al (2006) and Echo State Networks designed by Jaeger (2001)

We summarize Bagging Decision Trees, Random Forest and Extremely Randomized Trees in Table 1.

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

Table 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}

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

$$f(Expr) = \mu_1 \times I(g(Expr) > 0) + \mu_2 \times (1 - I(g(Expr) > 0)). \tag{26}$$

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

$$F_m(x) = F_m^L(x) \times I(g_m(x) > 0) + F_m^R(x) \times (1 - I(g_m(x) > 0)). \tag{27}$$

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 (13.27), 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

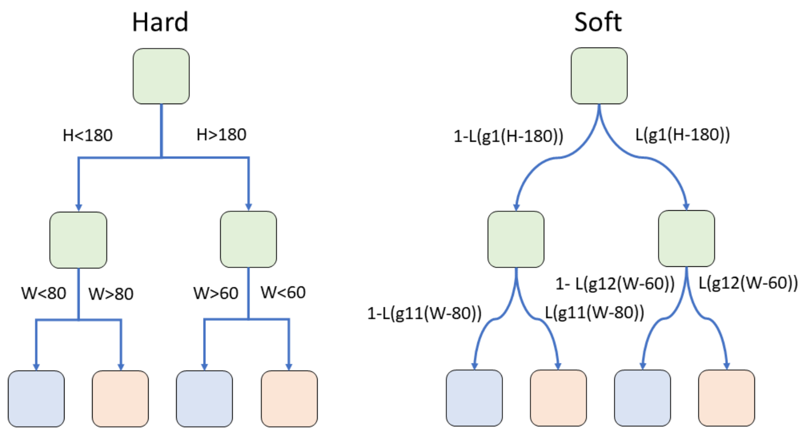


Fig. 12 Hard Decision Tree and Soft Decision Tree

$$F_m(x) = F_m^L(x) \times L(g_m(x)) + F_m^R(x) \times (1 - L(g_m(x))), \tag{28}$$

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 13.12, 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.

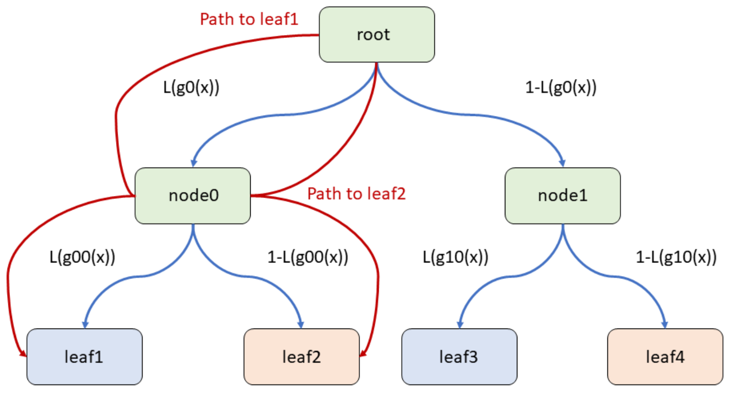


Fig. 13 Hierarchical Mixtures of Experts

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 (13.28), we have

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

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 13.13 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

$$\begin{aligned} G_1(x) &= I(1 = left) \times L(g_{00}(x)) + (1 - I(1 = left)) \times (1 - L(g_{00}(x))) \\ &= L(g_{00}(x)). \end{aligned}$$

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

$$\begin{aligned} P_1(x) &= \prod_{p \to 1} G_p(x) = G_0(x) \times G_1(x) \\ &= L(g_0(x)) \times L(g_{00}(x)). \end{aligned} \tag{29}$$

Similarly, to the path to leaf 2, we have

$$\begin{aligned} P_2(x) &= \prod_{p \to 2} G_p(x) = G_0(x) \times G_1(x) \\ &= L(g_0(x)) \times (1 - L(g_{00}(x))). \end{aligned} \tag{30}$$

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, Jordan and Jacob (1994) 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 calculate 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 predetermined 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 et al (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 et al (2016) constructed an ensemble of soft decision trees via Bagging to explore the ensemble of soft decision trees.

Basically, the soft decision tree method is not so popular as the decision tree since the training process is slower than growing a decision tree. But in many recent research papers, the soft decision tree shows the power for learning hierarchical features adaptively. Kontschieder et al (2015) proposed deep neural decision forest that combines the Convolutional Neural Network (CNN) feature extractor and tree structure into one differential hierarchical CNN and implements it into the tasks of computer vision. Frosst and Hinton (2017) explored the soft decision tree in distilling the knowledge or features extracted by a neural network based on its hierarchical structure. They found that the soft decision tree method can definitely learn hierarchical features via training.

6 Applications of Bagging and Random Forest in Economics

6.1 Bagging in economics

Recently, Bootstrap Aggregating is widely used in macroeconomic analysis and forecasting. Panagiotelis et al (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 et al (2014), Lee et al (2015) and Hillebrand et al (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 et al (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 average 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.

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, Strobl et al (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 (2018) 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 (Biau and D'Elia (2011)) and financial volatility forecasting (Luong and Dokuchaev (2018)). Finally, Fischer et al (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.

7 Summary

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

References

Audrino F, Medeiros MC (2011) Modeling and forecasting short-term interest rates: the benefits of smooth regimes, macroeconomic variables, and Bagging. Journal of Applied Econometrics 26(6):999–1022

Biau O, D'Elia A (2011) Euro area GDP forecasting using large survey datasets. A Random Forest approach

Breiman L (1996) Bagging predictors. Machine Learning 26(2):123–140

Breiman L (2000) Some infinity theory for predictor ensembles. Tech. rep.

Breiman L (2001) Random Forests. In: Machine Learning, pp 5–32

Breiman L, Friedman J, Stone C, Olshen R (1984) Classification and regression trees. The Wadsworth and Brooks-Cole statistics-probability series, Taylor & Francis

Bühlmann P (2004) Bagging, Boosting and ensemble methods, Springer, pp 877–907. Handbook of Computational Statistics: Concepts and Methods

Bühlmann P, Yu B (2002) Analyzing Bagging. Annals of Statistics 30(4):927–961

Buja A, Stuetzle W (2000a) Bagging does not always decrease mean squared error definitions. Tech. rep.

Buja A, Stuetzle W (2000b) Smoothing effects of Bagging. Preprint AT&T Labs-Research

Fischer T, Krauss C, Treichel A (2018) Machine learning for time series forecasting - a simulation study

Friedman JH, Hall P (2007) On Bagging and nonlinear estimation. Journal of Statistical Planning and Inference 137(3):669–683

Frosst N, Hinton G (2017) Distilling a Neural Network into a Soft Decision Tree. In: CEUR Workshop Proceedings

Geurts P, Ernst D, Wehenkel L (2006) Extremely Randomized Trees. Machine Learning 63(1):3–42

Hillebrand E, Lee TH, Medeiros M (2014) Bagging constrained equity premium predictors, Oxford University Press, chap 14, pp 330–356. Essays in Nonlinear Time Series Econometrics, Festschrift in Honor of Timo Tersvirta

Hirano K, Wright JH (2017) Forecasting with model uncertainty: representations and risk reduction. Econometrica 85(2):617–643

Hothorn T, Zeileis A (2017) Transformation Forests.

Huang GB, Zhu QY, Siew CK (2006) Extreme Learning Machine: algorithm, theory and applications. Neurocomputing 70:489–501

Inoue A, Kilian L (2008) How useful is Bagging in forecasting economic time. Journal of the American Statistical Association 103(482):511–522

Irsoy O, Yldz OT, Alpaydn E (2012) A Soft Decision Tree. In: 21st International Conference on Pattern Recognition (ICPR 2012)

Jaeger H (2001) The echo state approach to analysing and training Recurrent Neural Networks-with an erratum note. Tech. rep.

Janitza S, Celik E, Boulesteix AL (2016) A computationally fast variable importance test for Random Forests for high-dimensional data. Advances in Data Analysis and Classification (185):1–31

Jin S, Su L, Ullah A (2014) Robustify financial time series forecasting with Bagging. Econometric Reviews 33(5-6):575–605

Jordan M, Jacob R (1994) Hierarchical Mixtures of Experts and the EM algorithm. Neural Computation 6:181–214

Kontschieder P, Fiterau M, Criminisi A, Bul SR, Kessler FB, Bulo' SR (2015) Deep Neural Decision Forests. In: The IEEE International Conference on Computer Vision (ICCV), pp 1467–1475

Lee TH, Yang Y (2006) Bagging binary and quantile predictors for time series. Journal of Econometrics 135(1):465–497

Lee TH, Tu Y, Ullah A (2014) Nonparametric and semiparametric regressions subject to monotonicity constraints: estimation and forecasting. Journal of Econometrics 182(1):196–210

Lee TH, Tu Y, Ullah A (2015) Forecasting equity premium: global historical average versus local historical average and constraints. Journal of Business and Economic Statistics 33(3):393–402

Lin Y, Jeon Y (2006) Random Forests and Adaptive Nearest Neighbors. Journal of the American Statistical Association 101(474):578–590

Luong C, Dokuchaev N (2018) Forecasting of realised volatility with the Random Forests algorithm. Journal of Risk and Financial Management 11(4):61

Nyman R, Ormerod P (2016) Predicting economic recessions using machine learning

Panagiotelis A, Athanasopoulos G, Hyndman RJ, Jiang B, Vahid F (2019) Macroeconomic forecasting for Australia using a large number of predictors. Interantional Journal of Forecasting 35(2):616–633

Quinlan J (1986) Induction of Decision Trees. Machine Learning 1:81–106

Quinlan JR (1994) C4.5: programs for machine learning. Machine Learning 16(3):235–240

Strobl C, Boulesteix AL, Kneib T, Augustin T, Zeileis A (2008) Conditional variable importance for Random Forests. BMC Bioinformatics 9:1–11

Wager S, Athey S (2018) Estimation and inference of heterogeneous treatment effects using Random Forests. Journal of the American Statistical Association 113(523):1228–1242

Welch I, Goyal A (2008) A comprehensive look at the empirical performance of equity premium prediction. Review of Financial Studies 21-4:1455–1508

Yıldız OT, İrsoy O, Alpaydın E (2016) Bagging Soft Decision Trees. In: Machine Learning for Health Informatics, vol 9605, pp 25–36