國立臺灣大學工學院工業工程學研究所

碩士論文

Graduate Institute of Industrial Engineering

College of Engineering

National Taiwan University

Master Thesis

隨機森林與梯度提升決策樹在大數據下之探討

A Study of Random Forests and Gradient Boosting Decision Trees for Large-Scale Data

陳聖惟

Sheng-Wei Chen

指導教授：林智仁博士

Advisor: Chih-Jen Lin, Ph.D.

中華民國 106 年 7 月

July, 2017

doi:10.6342/NTU201702065

Abstract

In the past, we know that the tree-based methods may not handle the large-scale data sets. Therefore, the solver of the gradient boosting decision trees performs excellent in the large-scale data competitions. To know the details, we analyze the models of these tree-based methods. Furthermore, we compare their test accuracy and training time, we also consider the linear model and kernel method.

KEYWORDS: gradient boosting decision trees, random forests, support vector machine, classification and regression tree

doi:10.6342/NTU201702065

Contents

doi:10.6342/NTU201702065

doi:10.6342/NTU201702065

List of Figures

doi:10.6342/NTU201702065

List of Tables

doi:10.6342/NTU201702065

Chapter 1

Introduction

Nowadays, many methods are available to solve the classification problems. In Fernández-Delgado et al. (2014), they show that for many benchmark sets SVM and random forests are the two most effective methods in terms of the test accuracy. Recently, gradient boosting decision trees (GBDT) type solvers (Friedman, 2001) such as XGBoost (Chen and Guestrin, 2016) and LightGBM (Meng et al., 2016) have been shown to give excellent performance in many competitions. The goal of this thesis is to compare these state-of-the-art methods.

In Chapter 2, we introduce the models in detail. These methods include SVM, decision tree, random forests, and GBDT. After the introduction of the models, we explore the implementations of these models in Chapter 3. In particular, we give complexity analysis. The experimental result is given in Chapter 4. Our goal is to compare SVM and tree-based classification machines in different situations. We investigate that for large-scale data with many features, whether tree-based methods are better than SVM in terms of test accuracy and training time.

doi:10.6342/NTU201702065

Chapter 2

The Models

2.1 Support Vector Machine

The framework of SVM (Boser et al., 1992), is a very successful classification model that covers different convex loss functions. We can obtain the SVM model easily because it involves a convex minimization problem.

We introduce the linear form first. The data set is denoted as

$$D = \{(y_i, \boldsymbol{x}_i) \mid \boldsymbol{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}, i = 1, \ldots, l\},$$

for the binary classification problem, where y_i is the label of the ith instance, $\boldsymbol{x}_i$ is the feature vector of the ith instance, l is the number of the instances, and n is the total number of the features. We define the linear SVM model as

$$f^{\text{SVM}}(\boldsymbol{w}; \phi^{\text{linear}}(\boldsymbol{x})) = \boldsymbol{w}^T \phi^{\text{linear}}(\boldsymbol{x}),$$

where

$$\phi^{\text{linear}}(\boldsymbol{x}) = \boldsymbol{x}$$

and

$$\boldsymbol{w} \in \mathbb{R}^n.$$

doi:10.6342/NTU201702065

If we set

$$f^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}))=0,$$

then we get

$$\boldsymbol{w}^T\phi^{\text{linear}}(\boldsymbol{x})=w_1x_1+\cdots+w_nx_n=0,$$

which is a hyperplane to separate the two classes of data. That is, we predict

$$\begin{cases}1 & \text{if } f^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}))>0,\\ -1 & \text{otherwise.}\end{cases}$$

We train f^{SVM} by solving

$$\min_{\boldsymbol{w}}\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}+C\sum_{i=1}^{l}\xi(f^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i)),y_i), \tag{2.1}$$

where C is the cost parameter, and ξ is a loss function convex with respect to the first argument. The following formulas are the common convex loss functions.

$$\begin{aligned}\text{L1-loss}:\quad &\xi_{\text{L1}}(f^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i)),y_i)=\max\{0,1-y_if^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i))\}\\ \text{L2-loss}:\quad &\xi_{\text{L2}}(f^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i)),y_i)=\max\{0,1-y_if^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i))\}^2\\ \text{LR-loss}:\quad &\xi_{\text{LR}}(f^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i)),y_i)=\log(1+\exp(-y_if^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i))))\end{aligned}$$

The loss function gives a value of how we classify the data $\boldsymbol{x}_i$ incorrectly. If $y_if^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_i))$ is negative, then the function f^{SVM} does not predict $\phi^{\text{linear}}(\boldsymbol{x}_i)$ correctly. By using a convex loss function, (2.1) has a unique optimal function value and can be easily solved.

Next, we explain the kernel method. Because a linear model may not be good enough in some data sets, the kernel method extends the linear model into a nonlinear model. The idea of the kernel is to map the data $\boldsymbol{x}$ into a higher dimensional space $\mathbb{R}^{\hat{n}}$, where $\hat{n}>n$. That is,

$$\phi^{\text{kernel}}(\boldsymbol{x})\in\mathbb{R}^{\hat{n}}.$$

doi:10.6342/NTU201702065

Then we find a hyperplane

$$f^{\text{SVM}}(\boldsymbol{w}; \phi^{\text{kernel}}(\boldsymbol{x})) = \boldsymbol{w}^T \phi^{\text{kernel}}(\boldsymbol{x})$$

in $\mathbb{R}^{\hat{n}}$ by minimizing (2.1). When we use a hyperplane to separate the mapped data in the $\mathbb{R}^{\hat{n}}$ space, it is like to find a nonlinear function to seperate the original data in the $\mathbb{R}^n$ space. However, the $\phi^{\text{kernel}}(\boldsymbol{x})$ vector may be in a very high dimension space. To solve this problem, we consider special ϕ^{kernel} so that the following kernel function can be easily calculated.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi^{\text{kernel}}(\boldsymbol{x}_i)^T \phi^{\text{kernel}}(\boldsymbol{x}_j).$$

That is, we use another metric to meature the distance from $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$. Two common kernel functions are

$$\begin{aligned} K^{\text{polynomial}}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= (\gamma \boldsymbol{x}_i^T \boldsymbol{x}_j + r)^{\text{deg}}, \\ K^{\text{Gaussion}}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2), \end{aligned} \tag{2.2}$$

where

$$\gamma, r > 0,$$

and

$$\text{deg} \geq 1$$

are kernel parameters to be decided by the users. We give an example of the relationship between $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\phi^{\text{kernel}}(\boldsymbol{x})$. Suppose we use the polynomial kernel with

$$\gamma = 1, r = 1, \text{deg} = 2.$$

We can calculate

$$\begin{aligned} K^{\text{polynomial}}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= (\boldsymbol{x}_i^T \boldsymbol{x}_j + 1)^2 \\ &= (\boldsymbol{x}_i^T \boldsymbol{x}_j)^2 + 2\boldsymbol{x}_i^T \boldsymbol{x}_j + 1. \end{aligned} \tag{2.3}$$

doi:10.6342/NTU201702065

If we take

$$\phi^{\text{kernel}}(\boldsymbol{x}) = [x_1^2, \ldots, x_n^2, x_1x_2, \ldots, x_{n-1}x_n, \sqrt{2}x_1, \ldots, \sqrt{2}x_n, 1]^T,$$

then $\phi^{\text{kernel}}(\boldsymbol{x}_i)^T\phi^{\text{kernel}}(\boldsymbol{x}_j)$ is equal to (2.3). In using the Gaussion kernel, the primal model

$$\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \xi(f^{\text{SVM}}(\boldsymbol{w}; \phi^{\text{kernel}}(\boldsymbol{x}_i)), y_i) \tag{2.4}$$

may be difficult to solved, so we solve (2.4) by the dual problem. We give an example by using the L1-loss function. The model (2.4) is equivalent to

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\zeta}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\zeta_i \\ &\text{s.t. } y_i\boldsymbol{w}^T\phi^{\text{kernel}}(\boldsymbol{x}_i) \geq 1 - \zeta_i, i = 1, \ldots, l, \\ &\quad\ \zeta_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{2.5}$$

The dual problem of (2.5) is

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ &\text{s.t. } 0 \leq \alpha_i \leq C, \text{ for } i = 1, \ldots, l, \end{aligned} \tag{2.6}$$

where $Q_{ij} = y_iy_jK(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The derivation can be seen in, for example, Section A.2 in Cortes and Vapnik (1995). Note that we do not have the bias term b considered in other SVM works. It is important that we only need $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ when solving the problem (2.6). After we have the dual optimal solution $\boldsymbol{\alpha}^*$, we can use the primal-dual relationship

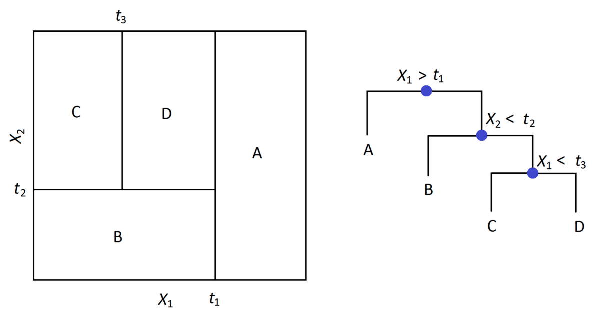
$$\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \phi^{\text{kernel}}(\boldsymbol{x}_i)$$

to find the primal optimal solution $\boldsymbol{w}^*$. The reason we can solve (2.5) by solving (2.6) is that the strong duality holds for this problem. Thus the primal objective value is equal to the dual objective value.

It is known that in general kernel SVM gives a model at least as good as linear SVM

doi:10.6342/NTU201702065

Figure 2.1: The example of the decision tree



(Keerthi and Lin, 2003). However, for large and sparse data, a linear model can often be competitive with a kernel one, but training and prediction are much faster, (Chu et al., 2015).

2.2 Classification and Regression Tree

Classification and regression tree (CART) by Breiman et al. (1984) is a useful classification model, and it is a binary decision tree model. The binary decision tree is like the one on the right of Figure 2.1. It has a logical statement at each node, and predicts the label of the data at the leaves. CART can handle both classification and regression. For classification, it uses the Gini impurity to meature the value of the tree. If we are doing multi-class classification with J classes, then the Gini impurity is

$$f^{\text{Gini}}(t^{\text{CART}}) = \sum_{m=1}^{M} \sum_{j \in J} p_j^m (1 - p_j^m),$$

where

$$p_j^m = \frac{|\{i | y_i = j, i = 1, \ldots, l\}|}{|I^m|},$$

t^{CART} is the CART function, and m, M, I^m will be introduced later in this section. For regression , CART considers the square loss defined as

$$\xi_{\text{Square}}(t^{\text{CART}}(\{s^m, R^m\}_{m=1}^{M}; \boldsymbol{x}), y) = (y - t^{\text{CART}}(\{s^m, R^m\}_{m=1}^{M}; \boldsymbol{x}))^2,$$

doi:10.6342/NTU201702065

where the CART function $t^{\text{CART}}(\{s^m, R^m\}_{m=1}^M; \boldsymbol{x})$ will be introduced later in this section. To make the problem be simpler, we discuss the square loss in this thesis.

Now, we introduce some properties of CART. The CART structure separates the feature space $\mathbb{R}^n$ into many regions. At each node we split data according to one feature value. Because each leaf corresponds to one region, we denote all regions as

$$\{R^m\}_{m=1}^M,$$

where M is the number of the leaves. That is,

$$\bigcup_{m=1}^M R^m = \mathbb{R}^n,$$

and

$$R^{\hat{m}} \bigcap R^{\dot{m}} = \emptyset, \text{ if } \hat{m} \neq \dot{m}.$$

In addition, each region R^m is a hyper rectangle in the $\mathbb{R}^n$ space. We denote

$$t^{\text{CART}}(\{s^m, R^m\}_{m=1}^M; \boldsymbol{x}) = \begin{cases} s^1 & \text{if } \boldsymbol{x} \in R^1, \\ & \vdots \\ s^M & \text{if } \boldsymbol{x} \in R^M, \end{cases}$$

as the CART function. If the data is in the leaf region R^m, t^{CART} will predict the scalar s^m. Now, we introduce how to calculate the scalar s^m. Suppose we have the following training instances

$$\{(z_i, \boldsymbol{x}_i) \mid \boldsymbol{x}_i \in R^m\}$$

in the leaf region R^m. Then we want to minimize the loss function

$$\min_s \sum_{i \in R^m} (z_i - s)^2.$$

doi:10.6342/NTU201702065

Because this is a convex problem, we can find the optimal solution s^m by

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}s}\sum_{i\in R^m}(z_i - s)^2 = 0,\\
\Rightarrow &\sum_{i\in R^m} s = \sum_{i\in R^m} z_i,\\
\Rightarrow &s = \frac{1}{|I^m|}\sum_{i\in R^m} z_i.
\end{aligned}$$

It implies that

$$s^m = \frac{1}{|I^m|}\sum_{i\in I^m} z_i, \text{ where } I^m = \{i \mid \boldsymbol{x}_i \in R^m\}. \tag{2.7}$$

Next, we introduce how to build a CART. With the square loss, define the following measure function

$$\pi(T) = \sum_{m=1}^{M}\sum_{i\in R^m} \xi_{\text{Square}}(t^{\text{CART}}(\{s^m, R^m\}_{m=1}^{M}; \boldsymbol{x}_i), y_i), \tag{2.8}$$

where T is the CART. If $\pi(\tilde{T})$ is smaller than $\pi(T)$, then we know the regression tree $\tilde{T}$ is better than T to fit the data. This property is used on growing the tree. Suppose we have the CART T. From a leaf node $\mathfrak{N}$ we want to grow the tree, and we denote $\tilde{T}$ as the new tree. Then for every feature we find the best point to split data in $\mathfrak{N}$. In the end we find $\tilde{T}$ that maximizes $\pi(T) - \pi(\tilde{T})$. We give the pseudo code in algorithm 1 and leave the complexity analysis in Chapter 3. With the algorithm 1, and the root node $\{\bar{y}, \mathbb{R}^n\}$, where

$$\bar{y} = \frac{1}{l}\sum_{i=1}^{l} y_i,$$

we can run the procedure recursively. In the end we will get the tree structure of CART.

2.3 Random Forests

Before we introduce random forests by Breiman (2001), we explain the Bootstrap aggregating model (i.e., the Bagging model) by Breiman (1996). We will give an example after we introduce the model. Now, we give some notations for the Bagging model. If

doi:10.6342/NTU201702065

Algorithm 1 Grow one leaf of the CART

```
**Input:** $T$, the CART we want to grow
**Input:** $\mathfrak{N}$, the target leaf, and its region is $R^m$
$D_{\mathfrak{N}} = \{(y_i, \boldsymbol{x}_i) \mid \boldsymbol{x}_i \in R^m\}$
score = $-\infty$, Best_splitting = $\emptyset$.
**for** $i = 1$ **to** $n$ **do**
    sort $D_{\mathfrak{N}}$ with the feature $i$
    $\mathfrak{N}_{\mathrm{R}} \leftarrow D_{\mathfrak{N}}$
    **for** $d$ in $D_{\mathfrak{N}}$ **do**
        $\mathfrak{N}_{\mathrm{L}} \leftarrow \mathfrak{N}_{\mathrm{L}} \cup \{d\}$, $\mathfrak{N}_{\mathrm{R}} \leftarrow \mathfrak{N}_{\mathrm{R}} \setminus \{d\}$
        $\tilde{T}$ = the CART $T$ after growing $\mathfrak{N}_{\mathrm{L}}$ and $\mathfrak{N}_{\mathrm{R}}$ at $\mathfrak{N}$
        **if** score $< \pi(T) - \pi(\tilde{T})$ **then**
            Best_splitting = ($\mathfrak{N}_{\mathrm{L}}$, $\mathfrak{N}_{\mathrm{R}}$, $i$, $d$)
            score $\leftarrow \pi(T) - \pi(\tilde{T})$
        **end if**
    **end for**
**end for**
grow the leaf $\mathfrak{N}$ with Best_splitting as $\tilde{T}$
**return** $\tilde{T}$
```

the number of the submodels which we want is N, we have the submodel t_k^{Bagging} with parameter $\boldsymbol{a}_k$, for $k = 1, \ldots, N$. We can subsample the instances of the data set. That is,

$$D_{J_k} = \{(y_j, \boldsymbol{x}_j) \mid j \in J_k\},$$

where J_k is the index set subsampled in the kth round. By training the set D_{J_k} with parameter $\boldsymbol{a}_k$, we obtain a submodel

$$t_k^{\text{Bagging}}(\boldsymbol{a}_k; \boldsymbol{x}).$$

The whole Bagging model is

$$f_N^{\text{Bagging}}(\boldsymbol{x}) = \frac{1}{N} \sum_{k=1}^{N} t_k^{\text{Bagging}}(\boldsymbol{a}_k; \boldsymbol{x}).$$

By the law of large numbers, Theorem 1.2 and Appendix I in Breiman (2001) show that

$$\frac{1}{N} \sum_{k=1}^{N} t_k^{\text{Bagging}}(\boldsymbol{a}_k; \boldsymbol{x}) \rightarrow \mathbb{E}_{\boldsymbol{x}}[t^{\text{Bagging}}(\boldsymbol{a}; \boldsymbol{x})], \text{ as } N \rightarrow \infty,$$

doi:10.6342/NTU201702065

where $t^{\text{Bagging}}(\boldsymbol{a};\boldsymbol{x})$ is trained by the whole data set without subsampling. We give a simple example here. Suppose our submodel t_k^{Bagging} is linear SVM with the L2-loss function. The parameter $\boldsymbol{a}_k$ in the Bagging model is the hyperplane weight $\boldsymbol{w}_k$ in the SVM model. We can train t_k^{Bagging} by solving

$$\boldsymbol{w}_k = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{j\in J_k} \xi_{\text{L2}}(f^{\text{SVM}}(\boldsymbol{w};\phi^{\text{linear}}(\boldsymbol{x}_j)), y_j).$$

Then we have

$$t_k^{\text{Bagging}}(\boldsymbol{a}_k;\boldsymbol{x}) = f^{\text{SVM}}(\boldsymbol{w}_k;\phi^{\text{linear}}(\boldsymbol{x})).$$

Finally, we have

$$f_N^{\text{Bagging}}(\boldsymbol{x}) = \frac{1}{N}\sum_{k=1}^{N} t_k^{\text{Bagging}}(\boldsymbol{a}_k;\boldsymbol{x}) = \frac{1}{N}\sum_{k=1}^{N} f^{\text{SVM}}(\boldsymbol{w}_k;\phi^{\text{linear}}(\boldsymbol{x}))$$

as the predicted value. Simiarly, we can do the subsampling on the features. That is,

$$D_{\tilde{J}_k} = \{(y_i, \psi_k(\boldsymbol{x}_i)) \mid i = 1,\ldots,l\},\ \text{for } k = 1,\ldots,N,$$

where $\tilde{J}_k$ is the feature subset selected in the kth round, and we suppose

$$\tilde{J}_k = \{\tilde{j}_1 \cdots \tilde{j}_{\tilde{n}}\}$$

is an ordered set, and

$$\psi_k(\boldsymbol{x}) = [x_{\tilde{j}_1} x_{\tilde{j}_2} \cdots x_{\tilde{j}_{\tilde{n}}}]^T.$$

We train t_k^{Bagging} with $\boldsymbol{a}_k$ and $D_{\tilde{J}_k}$ to get

$$t_k^{\text{Bagging}}(\boldsymbol{a}_k;\psi_k(\boldsymbol{x})),\ \text{for } k = 1,\ldots,N,$$

and the Bagging model is

$$f_N^{\text{Bagging}}(\boldsymbol{x}) = \frac{1}{N}\sum_{k=1}^{N} t_k^{\text{Bagging}}(\boldsymbol{a}_k;\psi_k(\boldsymbol{x})).$$

doi:10.6342/NTU201702065

In the Bagging model, if we set a CART model t_k^{CART} as the submodel t_k^{Bagging}, and we do the subsampling on the features, then we get the random forests model.

$$f_N^{\text{RF}}(\boldsymbol{x}) = \frac{1}{N}\sum_{k=1}^{N} t_k^{\text{CART}}(\{s_k^m, R_k^m\}_{m=1}^{M_k}; \psi_k(\boldsymbol{x})).$$

2.4 Gradient Boosting Decision Trees

In recent years, GBDT (Friedman, 2001), is a very popular model. Successful implementations such as XGBoost (Chen and Guestrin, 2016) and LightGBM (Meng et al., 2016) have been published. GBDT is related to random forests, but we will show the difference between these two models later. To begin, we discuss a boosting model.

2.4.1 Boosting Model

A boosting model consists of many submodels, which will be introduced later, and is constructed step by step. The following is a boosting model:

$$f_N^{\text{Boosting}}(\boldsymbol{x}) = \sum_{k=1}^{N} \beta_k t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}),$$

where N is the number of boosting rounds, t_k^{Boosting} is the submodel, $\boldsymbol{a}_k$ is its parameters and $\beta_k \in \mathbb{R}$ for $k = 1, \ldots, N$. That is, we have N submodels t_k^{Boosting} with different parameters $\boldsymbol{a}_k$ to construct the whole model f_N^{Boosting}, for $k = 1, \ldots, N$.

We explain the submodels by using an example, where $t_k^{\text{Boosting}}(\boldsymbol{w}_k; \boldsymbol{x})$ is a linear model with parameter $\boldsymbol{w}_k$. After a training procedure specified below, we have

$$t_k^{\text{Boosting}}(\boldsymbol{w}_k; \boldsymbol{x}) = \boldsymbol{w}_k^T \phi^{\text{linear}}(\boldsymbol{x})$$

as our predict function, for $k = 1, \ldots, N$. Then we have

$$f_N^{\text{Boosting}}(\boldsymbol{x}) = \sum_{k=1}^{N} \beta_k(\boldsymbol{w}_k^T \phi^{\text{linear}}(\boldsymbol{x})).$$

doi:10.6342/NTU201702065

Next, we discuss how to construct f_k^{Boosting} in the kth round. Given a loss function ξ, we generate $(\beta_k, \boldsymbol{a}_k)$ by

$$\arg\min_{\beta,\boldsymbol{a}} \sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \beta t_k^{\text{Boosting}}(\boldsymbol{a}; \boldsymbol{x}_i), y_i) \tag{2.9}$$

where $f_{k-1}^{\text{Boosting}}$ is given after we finished round $k-1$. In (2.9), we try to imporve the performce of the prediction function $f_{k-1}^{\text{Boosting}}$ by using the submodel t_k^{Boosting}.

2.4.2 Gradient Boosting

We have defined the boosting model f_N^{Boosting}, but still have problems in solving (2.9). Gradient boosting is one of the methods to find a proximal minimum of (2.9). We calculate the first derivative g_{ik} of the loss function ξ for $i = 1, \ldots, l$.

$$g_{ik} = \left.\frac{\partial \xi(z, y_i)}{\partial z}\right|_{z=f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i)}$$

The reason we calculate g_{ik} for $i = 1, \ldots, l$ is that

$$\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) - \delta g_{ik}, y_i)$$

can be reduced if the positive constant δ is small enough. We can give the detail by the Taylor expansion

$$\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) - \delta g_{ik}, y_i) = \sum_{i=1}^{l} [\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i) - \delta g_{ik}^2 + \varepsilon_{ik})], \tag{2.10}$$

where $\varepsilon_{ik}/\delta \to 0$ as $\delta \to 0$ for $i = 1, \ldots, l$. From (2.10),

$$\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) - \delta g_{ik}, y_i) - \sum_{i=1}^{l} (\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i)) = \sum_{i=1}^{l} (-\delta g_{ik}^2 + \varepsilon_{ik})),$$

and we focus on

$$-\delta g_{ik}^2 + \varepsilon_{ik}.$$

doi:10.6342/NTU201702065

Because

$$\lim_{\delta \to 0} \frac{\varepsilon_{ik}}{\delta} = 0, \text{for } i = 1, \ldots, l,$$

we know that ε_{ik} vanishes faster than δg_{ik}^2 when δ goes to zero. That is,

$$-\delta g_{ik}^2 + \varepsilon_{ik} \le 0,$$

so we have

$$\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) - \delta g_{ik}, y_i) - \sum_{i=1}^{l} (\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i)) \le 0,$$

if δ is small enough. We use the Taylor expansion again with the submodel t_k^{Boosting} to have

$$\begin{aligned} &\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \hat{\delta} t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i), y_i) \\ =&\sum_{i=1}^{l} [\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i) + \hat{\delta} g_{ik} t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i) + \hat{\varepsilon}_{ik})], \end{aligned}$$

where $\hat{\varepsilon}_{ik}/\hat{\delta} \to 0$ as $\hat{\delta} \to 0$ for $i = 1, \ldots, l$. If

$$\sum_{i=1}^{l} t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i) g_{ik} < 0, \tag{2.11}$$

then we can imply

$$\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \hat{\delta} t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i), y_i) \le \sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i) \tag{2.12}$$

when $\hat{\delta}$ is small enough. We see that (2.12) can help to find a proximal solution of (2.9). To make (2.11) happen, we can solve

$$\min_{\boldsymbol{a}} \sum_{i=1}^{l} (t_k^{\text{Boosting}}(\boldsymbol{a}; \boldsymbol{x}_i) - (-g_{ik}))^2 \tag{2.13}$$

doi:10.6342/NTU201702065

as a way to find $\boldsymbol{a}_k$. The regression problem (2.13) aims to make $t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x})$ close to $-g_{ik}$ so that (2.11) can hold.

Unfortunately, solving (2.13) doese not guarantee to obtain a model $\boldsymbol{w}_k$ such that (2.11) holds. If the property (2.11) does not hold, then

$$[t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_1) \cdots t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_l)]^T, \tag{2.14}$$

is an ascent direction. However, to avoid checking (2.11), we will show in Section 2.4.4 that by using CART to fit the regression problem (2.13), (2.11) can always hold.

Lastly, suppose (2.14) is a descent direction and we hope to find

$$\beta_k = \arg\min_{\beta} \sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \beta t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i), y_i). \tag{2.15}$$

Therefore, we do the line search to find β_k. We can see that the minimization of (2.15) involves only one variable. It can be approximately solved by backtracking line search. We give an ordered sequence $S = \{2^{-h}\}_{h=0}^{\infty}$, and denote s_h as the hth component in sequence S. We check the function value with $h = 0, 1, \ldots$, until

$$\begin{aligned} &\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + s_h t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i), y_i) \\ <&\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i) + s_h \mu \sum_{i=1}^{l} t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i) g_{ik}, \end{aligned}$$

where $\mu \in (0, 1)$ is a given constant. After the line search procedure terminates, we set s_h as β_k. With (2.11) and (2.13), we have

$$\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \beta_k t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i), y_i) < \sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i),$$

and $(\boldsymbol{a}_k, \beta_k)$ is a proximal solution of (2.9)

doi:10.6342/NTU201702065

2.4.3 Newton Boosting

In Section 2.4.2 we have used first-order approximation to find a proximal solution of (2.9). Now we use second-order approximation to find a proximal solution. The use of second-order approximation was developed in LogitBoost (Friedman et al., 2000), in which the LR-loss is used. Here we consider general losses. Define the Newton step from the loss function of the ith instance as

$$\frac{-g_{ik}}{h_{ik}}, \text{ where } h_{ik} = \left.\frac{\partial^2 \xi(z, y_i)}{\partial z^2}\right|_{z=f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i)}.$$

We can explain that it is a descent direction by Taylor expansion

$$\begin{aligned}&\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \delta(\frac{-g_{ik}}{h_{ik}}), y_i)\\ =&\sum_{i=1}^{l} [\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i) + \delta g_{ik}(\frac{-g_{ik}}{h_{ik}}) + \frac{1}{2}\delta^2 h_{ik}(\frac{-g_{ik}}{h_{ik}})^2 + \varepsilon_{ik})],\end{aligned}$$

where $\varepsilon_{ik}/\delta^2 \to 0$ as $\delta \to 0$. We focus on

$$\delta g_{ik}(\frac{-g_{ik}}{h_{ik}}) + \frac{1}{2}\delta^2 h_{ik}(\frac{-g_{ik}}{h_{ik}})^2.$$

It is equivalent to

$$\frac{g_{ik}^2}{h_{ik}}(-\delta + \frac{1}{2}\delta^2).$$

If ξ is a strongly convex function, we can imply $h_{ik} > 0$. We have

$$-\delta + \frac{1}{2}\delta^2 < 0,$$

when δ is small enough, and therefore

$$\sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \delta(\frac{-g_{ik}}{h_{ik}}), y_i) \le \sum_{i=1}^{l} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i).$$

doi:10.6342/NTU201702065

We use Taylor expansion again with the submodel t_k^{Boosting} to have

$$\begin{aligned}&\sum_{i=1}^{l}\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i)+\hat{\delta}t_k^{\text{Boosting}}(\boldsymbol{a}_k;\boldsymbol{x}_i),y_i)\\=&\sum_{i=1}^{l}[\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i),y_i)+\hat{\delta}g_{ik}t_k^{\text{Boosting}}(\boldsymbol{a}_k;\boldsymbol{x}_i)+\frac{1}{2}\hat{\delta}^2h_{ik}t_k^{\text{Boosting}}(\boldsymbol{a}_k;\boldsymbol{x}_i)^2+\hat{\varepsilon}_{ik}],\end{aligned}$$

where $\hat{\varepsilon}_{ik}/\hat{\delta}^2\to 0$ as $\hat{\delta}\to 0$. If

$$\sum_{i=1}^{l}[\hat{\delta}g_{ik}t_k^{\text{Boosting}}(\boldsymbol{a}_k;\boldsymbol{x}_i)+\frac{1}{2}\hat{\delta}^2h_{ik}t_k^{\text{Boosting}}(\boldsymbol{a}_k;\boldsymbol{x}_i)^2]<0, \tag{2.16}$$

then

$$\sum_{i=1}^{l}\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i)+\hat{\delta}t_k^{\text{Boosting}}(\boldsymbol{a}_k;\boldsymbol{x}_i),y_i)\le\sum_{i=1}^{l}\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i),y_i),$$

when $\hat{\delta}$ is small enough. To make (2.16) happen, we do

$$\arg\min_{\boldsymbol{a}}\sum_{i=1}^{l}(t_k^{\text{Boosting}}(\boldsymbol{a};\boldsymbol{x}_i)-(\frac{-g_{ik}}{h_{ik}}))^2 \tag{2.17}$$

as a way to find $\boldsymbol{a}_k$. Similar to the situation of solving (2.13), this setting does not guarantee that (2.16) holds. We can still multiply -1 on the direction, but we will show as well in Section 2.4.4 that by using CART to fit the regression problem in (2.17), (2.16) always holds.

Lastly, suppose (2.14) is the descent direction. We do the backtracking line search to find the proximal solution β_k by

$$\arg\min_{\beta}\sum_{i=1}^{l}\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i)+\beta t_k^{\text{Boosting}}(\boldsymbol{a}_k;\boldsymbol{x}_i),y_i)$$

and $(\boldsymbol{a}_k,\beta_k)$ is another proximal solution of (2.9).

To compare first-order and second-order boosting, we give two examples both with the LR-loss function. The first-order one is multiple additive regression trees (MART) by Friedman and Meulman (2003). The second-order one is additive logistic regression (LogitBoost) by Friedman et al. (2000). In Li (2012), we can see the comparison result in

doi:10.6342/NTU201702065

Appendex A. One thing to note is that LogitBoost uses the weighted regression tree, so it is different from (2.17). Another difference between Robust LogitBoost and LogitBoost is the definition of the π function in CART, where details are in Section 2.1 in Li (2012). The other thing we may notice is that neither MART nor LogitBoost uses the the line search to find the step size. The reason can be found in (8) of Li (2012), and we give more details in Section 2.4.4.

2.4.4 Some Technical Details

After we explained the boosting model, we know that (2.13) and (2.17) are not sufficient to lead to the conditions (2.11) and (2.16), respectively. If we use CART to generate a submodel t_k^{CART} as our t_k^{Boosting}, the conditions (2.11) and (2.16) are guaranteed to hold.

We focus on the gradient boosting tree first by using the CART to fit the problem (2.13). If M_k is large enough, we may construct the region $\{R_k^m\}_{m=1}^{M_k}$ to satisfy the following property. For any $\varepsilon_k^m > 0$, there exists a region R_k^m, such that for all $\boldsymbol{x}_i$ in R_k^m, we have

$$\sum_{i \in I_k^m} [s_k^m - (-g_{ik})]^2 < \varepsilon_k^m, \tag{2.18}$$

where s_k^m is the average of

$$\{-g_{ik} \mid i \in I_k^m\}, \text{ and } I_k^m = \{i \mid \boldsymbol{x}_i \in R_k^m\}. \tag{2.19}$$

Note that s_k^m here is defined differently from that in (2.7). The reason of (2.18) is that l is finite. If we let M_k equal l, and for each region we have only one instance, then (2.18) holds. There are two cases we need to discuss.

Case 1: $g_{ik} \neq 0,$ for some $i \in R_k^{\hat{m}},$ for all $\hat{m}.$

Case 2: $g_{ik} = 0,$ for all $i \in R_k^{\hat{m}},$ for some $\hat{m}.$

In Case 1, we rewrite the statement (2.18) to

$$\sum_{i \in I_k^m} [(s_k^m g_{ik}] < \frac{1}{2}[\varepsilon_k^m - \sum_{i \in I_k^m} ((s_k^m)^2 + g_{ik}^2)].$$

doi:10.6342/NTU201702065

If

$$\varepsilon_k^m - \sum_{i \in I_k^m} ((s_k^m)^2 + g_{ik}^2) < 0,$$

for all m, then

$$\begin{aligned}\sum_{i=1}^{l} t_k^{\text{CART}}(\{s_k^m, R_k^m\}_{m=1}^{M_k}; \boldsymbol{x}_i) g_{ik} &= \sum_{m=1}^{M_k} \sum_{i \in I_k^m} [s_k^m g_{ik}] \\ &< \sum_{m=1}^{M_k} [\varepsilon_k^m - \sum_{i \in I_k^m} ((s_k^m)^2 + g_{ik}^2)] \\ &< 0.\end{aligned}$$

In Case 2, suppose we have a region $R^{\hat{m}}$ such that

$$g_{ik} = 0, \forall i \in I_k^{\hat{m}}.$$

Then we have

$$\sum_{i \in I_k^m} s_k^{\hat{m}} g_{ik} = 0,$$

so

$$\begin{aligned}\sum_{i=1}^{l} t_k^{\text{CART}}(\{s_k^m, R_k^m\}_{m=1}^{M_k}; \boldsymbol{x}_i) g_{ik} &= \sum_{m=1}^{M_k} \sum_{i \in I_k^m} s_k^m g_{ik} \\ &= \sum_{m \neq \hat{m}} \sum_{i \in I_k^m} s_k^m g_{ik}.\end{aligned}$$

Thus, we can reduce the number of updated regions. If

$$g_{ik} = 0, \forall i = 1, \ldots, l,$$

and

$$k < N,$$

then we early stop the boosting.

Next, we move on to the Newton boosting tree. Similarly, we have two cases to dis-

doi:10.6342/NTU201702065

cuss, if M is large enough, for any $\tilde{\varepsilon}_k^m > 0$, there exists the region R_k^m, such that for all $\boldsymbol{x}_i$ in R_k^m, we have

$$[s_k^m - (\frac{-g_{ik}}{h_{ik}})]^2 < \tilde{\varepsilon}_k^m, \text{ for all } i \in I_k^m,$$

where s_k^m is the average of

$$\{\frac{-g_{ik}}{h_{ik}} \mid i \in I_k^m\},$$

and I_k^m is the same as that in (2.19). In Case 1, assume

$$\frac{g_{ik}}{h_{ik}} \neq 0 \text{ for some } i \in R_k^{\hat{m}}, \text{ for all } \hat{m}.$$

Then we imply

$$g_{ik}s_k^m < \frac{h_{ik}}{2}[\tilde{\varepsilon}_k^m - (s_k^m)^2 - (\frac{g_{ik}}{h_{ik}})^2].$$

Suppose we have a quadratic polynomial

$$\delta B_i + \frac{1}{2}\delta^2 A_i,$$

where $B_i < 0$ and $A_i > 0$. There exists $\hat{\delta}_i > 0$, such that

$$\hat{\delta}_i B_i + \frac{1}{2}\hat{\delta}_i^2 A_i < 0.$$

We set $B_i = g_{ik}s_k^m$, $A_i = h_{ik}(s_k^m)^2$, and take

$$\hat{\delta} = \min\{\hat{\delta}_1 \ldots \hat{\delta}_l\}.$$

Then we imply

$$\hat{\delta} B_i + \frac{1}{2}\hat{\delta}^2 A_i < 0, \text{ for all } i = 1, \ldots, l.$$

Thus, (2.16) holds. In Case 2, assume

$$\frac{g_{ik}}{h_{ik}} = 0, \text{ for all } i \in R_k^{\hat{m}}, \text{ for some } \hat{m}.$$

doi:10.6342/NTU201702065

Then

$$s_k^{\hat{m}} = 0,$$

and

$$\begin{aligned}
&\sum_{i=1}^{l} [\hat{\delta} g_{ik} t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i) + \frac{1}{2}\hat{\delta}^2 h_{ik} t_k^{\text{Boosting}}(\boldsymbol{a}_k; \boldsymbol{x}_i)^2] \\
=&\sum_{m=1}^{M_k} \sum_{i \in I_k^m} [\hat{\delta} g_{ik} s_k^m + \frac{1}{2}\hat{\delta}^2 h_{ik} s_k^{m2}] \\
=&\sum_{m \neq \hat{m}} \sum_{i \in I_k^m} [\hat{\delta} g_{ik} s_k^m + \frac{1}{2}\hat{\delta}^2 h_{ik} s_k^{m2}].
\end{aligned}$$

Similar to the first-order form, we can reduce the updated regions. If

$$\frac{g_{ik}}{h_{ik}} = 0, \forall i = 1, \ldots, l,$$

and

$$k < N,$$

then we early stop the boosting. Finally, we have proved that (2.11) and (2.16) can hold if we use CART to fit the data with a large enough M_k.

Furtheremore, if CART is used, the feature space can be splitted into $\{R_k^m\}_{m=1}^{M_k}$. Every leaf region is adjoint, so we can modify the function value s_k^m to find the best descent step. Given a region R_k^m, we can take

$$\beta_k^m = \arg\min_{\beta} \sum_{i \in I_k^m} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \beta s_k^m, y_i), \tag{2.20}$$

such that we have

$$t_k^{\text{CART}}(\{\beta_k^m s_k^m, R_k^m\}_{m=1}^{M_k}; \boldsymbol{x}) = \begin{cases} \beta_k^1 s_k^1 & \text{if } \boldsymbol{x} \in R_k^1, \\ & \vdots \\ \beta_k^{M_k} s_k^{M_k} & \text{if } \boldsymbol{x} \in R_k^{M_k}. \end{cases}$$

The setting is similar to the line search in each region. Futhermore, we use the second-

doi:10.6342/NTU201702065

order Taylor expansion:

$$\sum_{i \in I_k^m} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \beta s_k^m, y_i) \approx \sum_{i \in I_k^m} [\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i) + g_{ik} \beta s_k^m + \frac{1}{2} h_{ik} \beta^2 (s_k^m)^2].$$

Then we can imply

$$\begin{aligned} \beta_k^m &= \arg\min_{\beta} \sum_{i \in I_k^m} \xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i) + \beta s_k^m, y_i) \\ &\approx \arg\min_{\beta} \sum_{i \in I_k^m} [\xi(f_{k-1}^{\text{Boosting}}(\boldsymbol{x}_i), y_i) + g_{ik} \beta s_k^m + \frac{1}{2} h_{ik} \beta^2 (s_k^m)^2] \\ &= \frac{1}{s_k^m} \frac{-\sum_{i \in I_k^m} g_{ik}}{\sum_{i \in I_k^m} h_{ik}}. \end{aligned} \tag{2.21}$$

After (2.21), we have

$$\beta_k^m s_k^m = \frac{-\sum_{i \in I_k^m} g_{ik}}{\sum_{i \in I_k^m} h_{ik}},$$

so the model is

$$t_k^{\text{CART}}(\{\beta_k^m s_k^m, R_k^m\}_{m=1}^{M_k}; \boldsymbol{x}) = \begin{cases} \frac{-\sum_{i \in I_k^1} g_{ik}}{\sum_{i \in I_k^1} h_{ik}} & \text{if } \boldsymbol{x} \in R_k^1, \\ \vdots & \\ \frac{-\sum_{i \in I_k^{M_k}} g_{ik}}{\sum_{i \in I_k^{M_k}} h_{ik}} & \text{if } \boldsymbol{x} \in R_k^{M_k}. \end{cases}$$

Note that (2.21) is an approximation. In other words, f_k^{Boosting} may be larger than $f_{k-1}^{\text{Boosting}}$, but we do not have the cost of the line search. Last, in Section 5 of Friedman (2001), a learning rate η is used to avoid overfitting. Thus, the second-order GBDT model will be

$$f_N^{\text{GBDT}}(\boldsymbol{x}) = \eta \sum_{k=1}^{N} t_k^{\text{CART}}(\{\beta_k^m s_k^m, R_k^m\}; \boldsymbol{x}).$$

doi:10.6342/NTU201702065

Chapter 3

The Implementations

In this Chapter, we will discuss how to implement the models in Chapter 2 and calculate their computational complexities.

3.1 The Complexity of CART

Before we explain random forests and GBDT, we need to understand the complexity of the CART. In Section 6.1 of Witten et al. (2011), they have derived the following complexity for building a balanced tree.

$$\begin{aligned}&O(n \times l \times \log(l)), \text{ for calculating the square loss,}\\&O(n \times l \times \log(l)), \text{ for sorting the data.}\end{aligned}$$

They assume that a complete tree is established, but in practice, the tree is often constructed up to a give depth d. We extend their analysis to cover such situations. Without loss of generality, we let T be the tree shown in Figure 3.1, and we would like to grow a tree from the black node $\mathfrak{N}$. Assume we select a subset of p features to construct the tree. Let

$$I_{\mathfrak{N}} = \{i \mid \boldsymbol{x}_i \in \text{Region}(\mathfrak{N})\}$$

and for each of the p selected features, we find the best splitting point. If the jth feature is given, we conduct the following steps:

doi:10.6342/NTU201702065

Figure 3.1: Growing leaf $\mathfrak{N}$

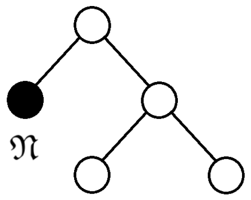
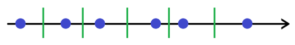


Figure 3.2: Splitting candidates



Step 1: Sort $(z_i, \boldsymbol{x}_i), \forall i \in I_{\mathfrak{N}}$ by the jth conponent of $\boldsymbol{x}_i$, $i \in I_{\mathfrak{N}}$.

Step 2: Check the Gain$(\mathfrak{N}_L, \mathfrak{N}_R)$ of all splitting candidates; see an example in Figure 3.2.

Here

$$\text{Gain}(\mathfrak{N}_L, \mathfrak{N}_R) = \pi(\tilde{T}) - \pi(T),$$

where T is the CART before we grow $\mathfrak{N}_L$ and $\mathfrak{N}_R$, $\tilde{T}$ is the one after growing, and π is defined at (2.8). We give an example in Figure 3.3, where we predict $T(\boldsymbol{x})$ as

$$s_{\mathfrak{N}}, \text{ if } \boldsymbol{x} \in \text{Region}(\mathfrak{N}),$$

and predict $\tilde{T}(\boldsymbol{x})$ as

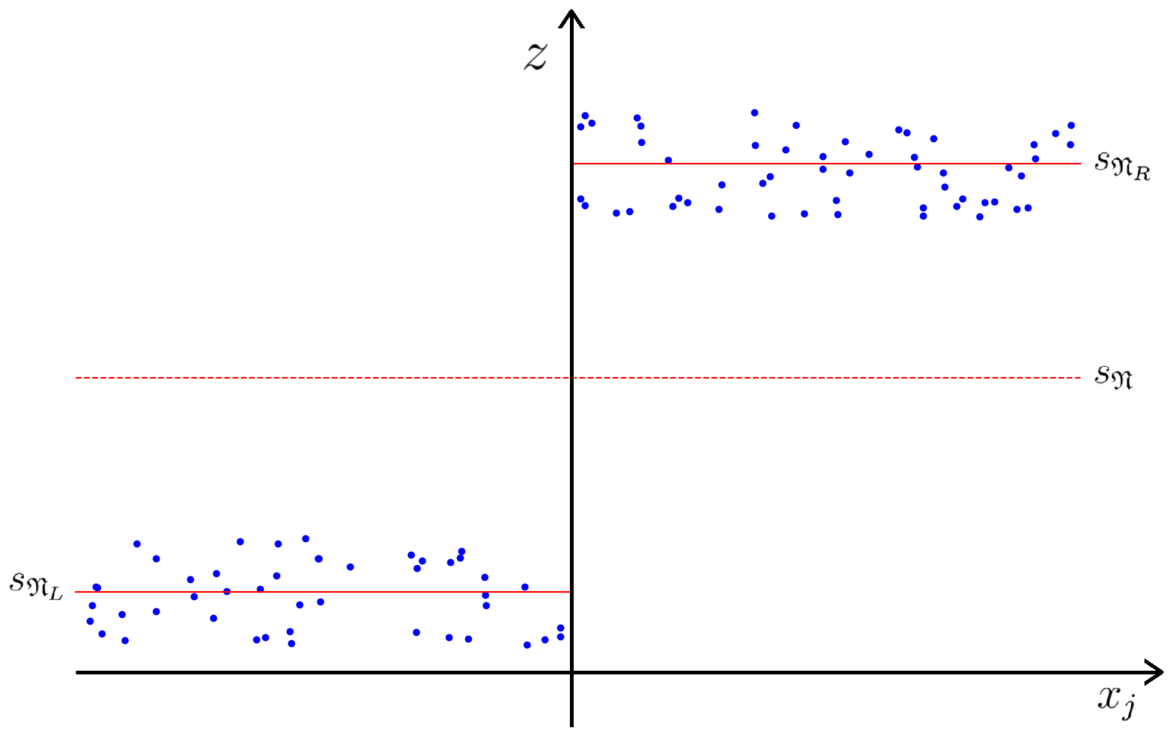
$$s_{\mathfrak{N}_L}, \text{ if } \boldsymbol{x} \in \text{Region}(\mathfrak{N}_L),$$
$$s_{\mathfrak{N}_R}, \text{ if } \boldsymbol{x} \in \text{Region}(\mathfrak{N}_R).$$

Then we can calculate the square loss in the π function as

$$\xi_{\text{Square}}(\mathfrak{N}) = \sum_{i \in I_{\mathfrak{N}}} (z_i - s_{\mathfrak{N}})^2,$$
$$\xi_{\text{Square}}(\mathfrak{N}_L) = \sum_{i \in I_{\mathfrak{N}_L}} (z_i - s_{\mathfrak{N}_L})^2,$$
$$\xi_{\text{Square}}(\mathfrak{N}_R) = \sum_{i \in I_{\mathfrak{N}_R}} (z_i - s_{\mathfrak{N}_R})^2,$$

doi:10.6342/NTU201702065

Figure 3.3: The prediction before and after growing children

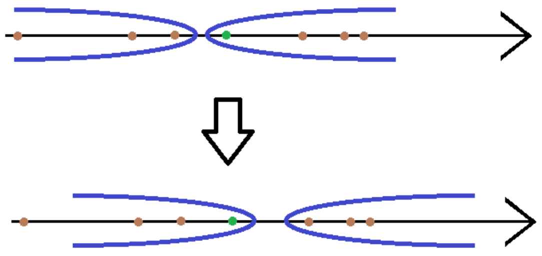


where $I_{\mathfrak{N}}$, $I_{\mathfrak{N}_L}$, $I_{\mathfrak{N}_R}$ are index sets of instances in $\mathfrak{N}$, $\mathfrak{N}_L$, $\mathfrak{N}_R$, respectively. To find the splitting point, we consider the middle point in the interval between any two adjacent points; see Figure 3.4. There exists a quick way to know the loss value of these two sets. Suppose we have $\mathfrak{N}_L$ and $\mathfrak{N}_R$. After we move one element $(z_{\hat{i}}, \boldsymbol{x}_{\hat{i}})$ from $\mathfrak{N}_R$ to $\mathfrak{N}_L$, we get $\tilde{\mathfrak{N}}_L$ and $\tilde{\mathfrak{N}}_R$ with

$$\begin{aligned}\tilde{\mathfrak{N}}_L =& \mathfrak{N}_L \cup \{(z_{\hat{i}}, \boldsymbol{x}_{\hat{i}})\},\\ \tilde{\mathfrak{N}}_R =& \mathfrak{N}_R - \{(z_{\hat{i}}, \boldsymbol{x}_{\hat{i}})\}.\end{aligned}$$

doi:10.6342/NTU201702065

Figure 3.4: Searching the splitting point by considering all middle points between any two adjacent feature values.



The calculation is as follows.

$$
\begin{aligned}
\xi_{\text{Square}}(\tilde{\mathfrak{N}}_L) &= \sum_{i \in I_{\tilde{\mathfrak{N}}_L}} (z_i - s_{\tilde{\mathfrak{N}}_L})^2 \\
&= \sum_{i \in I_{\tilde{\mathfrak{N}}_L}} z_i^2 - \frac{(\sum_{i \in I_{\tilde{\mathfrak{N}}_L}} z_i)^2}{|I_{\tilde{\mathfrak{N}}_L}|} \\
&= \sum_{i \in I_{\mathfrak{N}_L}} z_i^2 + z_{\hat{i}}^2 - \frac{(\sum_{i \in I_{\mathfrak{N}_L}} z_i + z_{\hat{i}})^2}{|I_{\mathfrak{N}_L}| + 1} \\
=&\xi_{\text{Square}}(\mathfrak{N}_L) + \frac{(\sum_{i \in I_{\mathfrak{N}_L}} z_i)^2}{|I_{\mathfrak{N}_L}|} + z_{\hat{i}}^2 - \frac{(\sum_{i \in I_{\mathfrak{N}_L}} z_i + z_{\hat{i}})^2}{|I_{\mathfrak{N}_L}| + 1} \\
=&\xi_{\text{Square}}(\mathfrak{N}_L) + \frac{1}{|I_{\mathfrak{N}_L}| + 1}[\frac{|I_{\mathfrak{N}_L}| + 1}{|I_{\mathfrak{N}_L}|}(\sum_{i \in I_{\mathfrak{N}_L}} z_i)^2 \\
&+ (|I_{\mathfrak{N}_L}| + 1)z_{\hat{i}}^2 - (\sum_{i \in I_{\mathfrak{N}_L}} z_i + z_{\hat{i}})^2] \\
=&\xi_{\text{Square}}(\mathfrak{N}_L) + \frac{1}{|I_{\mathfrak{N}_L}| + 1}[\frac{1}{|I_{\mathfrak{N}_L}|}(\sum_{i \in I_{\mathfrak{N}_L}} z_i)^2 \\
&+ (|I_{\mathfrak{N}_L}|)z_{\hat{i}}^2 - 2(\sum_{i \in I_{\mathfrak{N}_L}} z_i)z_{\hat{i}}] \\
=&\xi_{\text{Square}}(\mathfrak{N}_L) + \frac{|I_{\mathfrak{N}_L}|}{|I_{\mathfrak{N}_L}| + 1}[\frac{\sum_{i \in I_{\mathfrak{N}_L}} z_i}{|I_{\mathfrak{N}_L}|} - z_{\hat{i}}]^2. \\
=&\xi_{\text{Square}}(\mathfrak{N}_L) + \frac{|I_{\mathfrak{N}_L}|}{|I_{\mathfrak{N}_L}| + 1}[s_{\mathfrak{N}_L} - z_{\hat{i}}]^2.
\end{aligned}
$$

Similarly,

$$\xi_{\text{Square}}(\tilde{\mathfrak{N}}_R) = \xi_{\text{Square}}(\mathfrak{N}_R) - \frac{|I_{\mathfrak{N}_R}|}{|I_{\mathfrak{N}_R}| - 1}[s_{\mathfrak{N}_R} - z_{\hat{i}}]^2.$$

doi:10.6342/NTU201702065

Base on this updating rule, we know that the complexity of searching for the best splitting node with the jth feature is

$$O(|I_{\mathfrak{N}}|).$$

Remember that we have p features to search, so we have

$$O(p \times |I_{\mathfrak{N}}|)$$

in growing node $\mathfrak{N}$.

Next, we discuss the complexity of calculating the square loss in two cases.

Case 1: Balanced tree.

Case 2: Unbalanced tree.

In case 1, suppose the tree is balanced. We use

$$1, \ldots, 2^{d+1} - 1$$

as indices of nodes and let their corresponding regions be

$$\mathfrak{N}_1, \ldots, \mathfrak{N}_{2^{d+1}-1}.$$

See an illustration in Figure 3.5. Therefore, numbers of instances in regions which are at the same layer are about equal. That is,

$$|I_{\mathfrak{N}_{2^{\tilde{d}}}}| \approx |I_{\mathfrak{N}_{2^{\tilde{d}}+1}}| \approx \cdots \approx |I_{\mathfrak{N}_{2^{\tilde{d}}+2^{\tilde{d}}-1}}| \text{ for } \tilde{d} = 1, \ldots, d.$$

In node $\mathfrak{N}_i$, we need

$$O(p \times |I_{\mathfrak{N}_i}|).$$

doi:10.6342/NTU201702065

Figure 3.5: A balanced tree with depth d

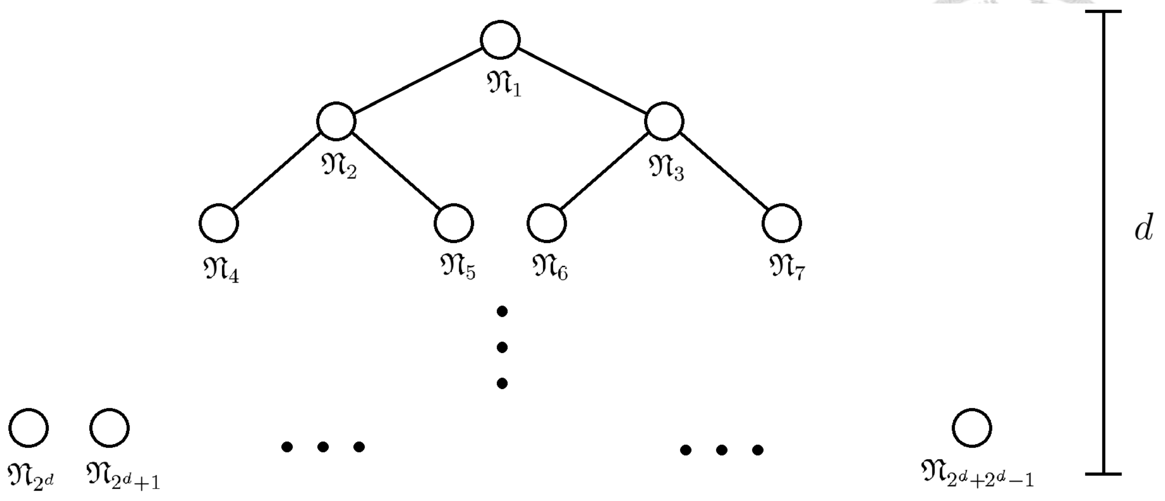
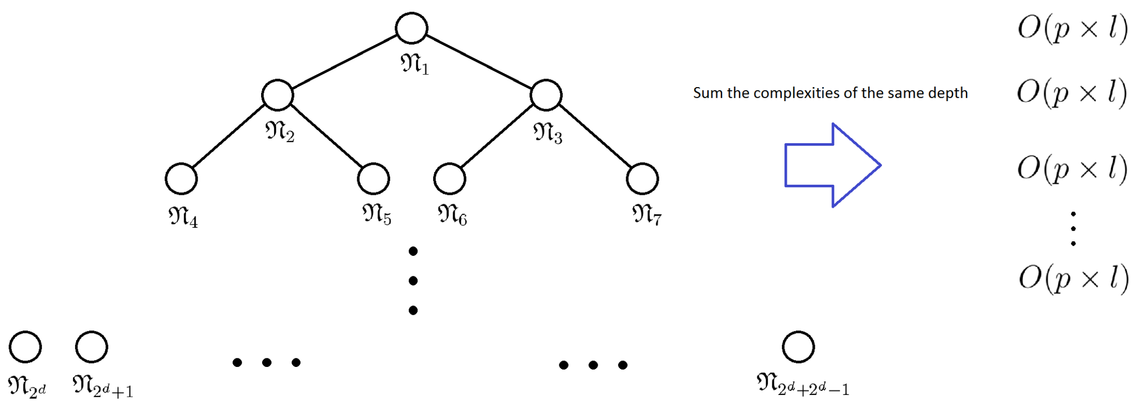


Figure 3.6: The complexity of training a balanced CART.



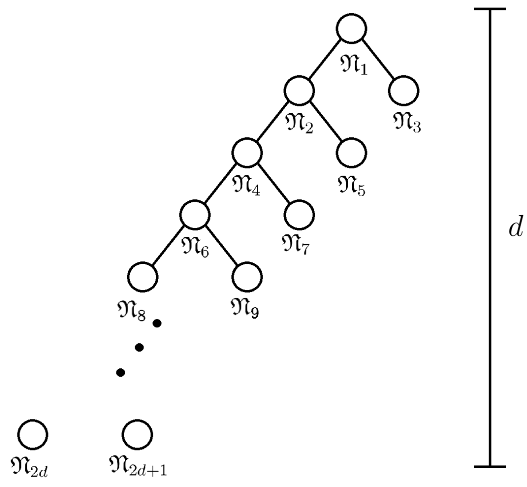
Given a tree of $\hat{d}$ layers, the complexity of growing it to $\hat{d}+1$ layers is

$$
\begin{aligned}
&O(p \times |I_{\mathfrak{N}_{2^{\hat{d}-1}}}|) + \cdots + O(p \times |I_{\mathfrak{N}_{2^{\hat{d}}-1}}|)\\
=&2^{\hat{d}} O(p \times |I_{\mathfrak{N}_{2^{\hat{d}-1}}}|)\\
=&2^{\hat{d}} O(p \times \frac{l}{2^{\hat{d}}})\\
=&O(p \times l).
\end{aligned}
$$

The complexity of nodes at different layers is shown in Figure 3.6. From Figure 3.6, we

doi:10.6342/NTU201702065

Figure 3.7: An unbalanced tree with depth d



can easily know the total complexity of calculating the square loss is

$$O(p \times l) + \overset{d}{\cdots} + O(p \times l) = O(p \times d \times l).$$

In Case 2, we assume T is an unbalanced tree, and use

$$1, \ldots, 2d + 1$$

as indices of nodes and let their corresponding regions be

$$\mathfrak{N}_1, \ldots, \mathfrak{N}_{2d+1}.$$

See an illustration in Figure 3.7. To discuss the worst case, we assume that

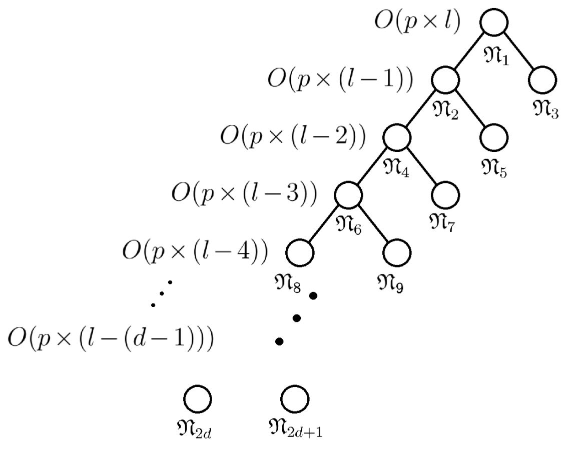
$$|I_{\mathfrak{N}_{2\tilde{d}+1}}| = 1, \text{ for } \tilde{d} = 1, \ldots, d.$$

That is, we have only one instance in each of these leaves. At a layer $\hat{d}$, to expand the node $\mathfrak{N}_{2\hat{d}}$, the cost is

$$O(p \times |I_{\mathfrak{N}_{2\hat{d}}}|) = O(p \times (l - \hat{d})).$$

doi:10.6342/NTU201702065

Figure 3.8: The complexity of training an unbalanced CART.



The complexity of nodes at different layers are shown in Figure 3.8. We can know the the total complexity of calculating the square loss is

$$\begin{aligned} O(p\times l)+O(p\times(l-1))+\ldots+O(p\times(l-d+1)) &= O(p\times\frac{d(2l-d+1)}{2}) \\ &\approx O(p\times d\times l). \end{aligned}$$

After the discussion, we know that whether the tree is balanced or not, we have the complexity of calculating the square loss to be $O(p\times d\times l)$.

Next, we introduce the sorting complexity in Step 1. From Section 6.1 of Witten et al. (2011), we may use the appropriate algorithm. For example, suppose we have

value	10	30	3	16	27
rank	2	5	1	3	4

.

If we have the following subset

$$10\quad 30\quad 27,$$

and want to sort them, then we check the indices of the rank, which can be obtained in the beginning. They are

$$2\quad 5\quad 4,$$

doi:10.6342/NTU201702065

so we can sort them. Because we select a fixed subset of p features in growing the tree, the sorting operation is needed only once in the beginning. The cost is

$$O(p \times \frac{\Upsilon}{n} \times \log(\frac{\Upsilon}{n})),$$

where Υ is the number of the non-zeros in the data set. Note that for sparse data sets, for each feature we only need to sort non-zero values, and on average the number of non-zero elements is

$$\frac{\Upsilon}{n}.$$

In conclusion, the complexity of constructing the CART is

$$O(p \times d \times l), \text{ for calculating the square loss,}$$
$$O(p \times \frac{\Upsilon}{n} \times \log(\frac{\Upsilon}{n})), \text{ for sorting the data.}$$

3.2 Complexity of Random Forests

In a random forests model, we train N CARTs and pre-sort all the data once, so the complexity is

$$O(N \times d \times p \times l), \text{ for calculating the square loss,}$$
$$O(N \times \Upsilon \times \log(\frac{\Upsilon}{n})), \text{ for sorting the data.}$$

3.3 Complexity of GBDT

In GBDT, we train N CARTs. Before we fit the kth CART, we must calculate g_{ik} and h_{ik}, for $i = 1, \ldots, l$. The complexity is

$$O(N \times l), \text{ for calculating } g_{ik} \text{ and } h_{ik}, \forall i, \forall k\ .$$
$$O(N \times d \times p \times l), \text{ for calculating the square loss,}$$
$$O(N \times \Upsilon \times \log(\frac{\Upsilon}{n})), \text{ for sorting the data.}$$

doi:10.6342/NTU201702065

Recently, to improve the sorting time and the memory usage, LightGBM uses the histogram technique (Jin and Agrawal, 2003). This technique uses the ordered integer indices to replace the continues values. Then we can easily do the sort by these index tags. When the number of the bins is given, we need to scan the values of the data set and replace them with bin indices. For the jth feature, the cost is

$$\frac{\Upsilon}{n} + \frac{\Upsilon}{n}.$$

The first part of the cost is for finding the minimum and the maximum values of the jth feature, denoted as max_val$_j$ and min_val$_j$, respectively. We then calculate

$$\frac{\text{max\_val}_j - \text{min\_val}_j}{\#(\text{bins} - 1)}$$

for the interval of the bins. The second part of the cost is for replacing the values with the indices of the bins. The total cost for all features is

$$2 \times \frac{\Upsilon}{n} \times n = 2 \times \Upsilon.$$

Note that, we replace the values in all of the data set, not only for the p features used in a tree. The benefits of using histogram are the saving of the memory usage and the saving of the sorting time, but we transform the data to some discrete points. Now we check how using a histogram saves the memory usage. For a double floating-point value, 64 bits are need to store it. On the other hand, if the number of the bins is 256, we only need 8 bits to store the bin index. Back to the sorting, with the indices of the bins, we only need to scan data once. The complexity of sorting in CART is

$$O(p \times \frac{\Upsilon}{n}).$$

doi:10.6342/NTU201702065

The total complexity with using histogram are

$$\begin{aligned}
O(2 \times \Upsilon) &\text{ for replacing continues values to bins indices,} \\
O(N \times l) &\text{ for calculating } g_{ik} \text{ and } h_{ik}, \forall i, \forall k, \\
O(N \times d \times p \times l) &\text{ for calculating the square loss,} \\
O(N \times d \times p \times \frac{\Upsilon}{n}) &\text{ for sorting the data.}
\end{aligned}$$

Both of the libraries XGBoost and LightGBM support the histogram method nowadays.

doi:10.6342/NTU201702065

Chapter 4

Experiments

After introducing various classification methods, our purpose in this chapter is to compare them in detail. We select some large data sets in LIBSVM data sets. [1] Note that astro-physic and yahoo-japan are not publicly available. While all these sets have a large number of instances, their number of features significantly varies. Most data sets come with available training and test sets, but for astro-physic, covtype, real-sim, url and webspam, only training instances are available. Therefore, we randomly split the original training set as the training set and test set with the ratio $4 : 1$. Statistics of data sets are presented in Table 4.1. Testing accuracy is the ratio between the number of correct predictions and the testing size.

4.1 Parameter Selection

For each of these classifiers, some suitable parameters must be chosen. We conduct cross validation to select parameters. We respectively discuss parameters of each method in a subsection. We also discuss the package used for each method.

4.1.1 SVM

To train SVM, we consider LIBSVM (Chang and Lin, 2011) for kernel SVM with using l1 loss and LIBLINEAR (Fan et al., 2008) for linear SVM with using LR loss. For

[1] https://www.csie.ntu.edu.tw/c̃jlin/libsvmtools/datasets/

doi:10.6342/NTU201702065

Data set	Training size	Test size	#features (n)	#nnz / n
astro-physic	49,896	12,473	99,757	38
cod-RNA	59,535	271,617	8	59,535
covtype	464,810	116,202	54	102,824
ijcnn1	49,990	91,701	22	29,540
kdd10b-raw	19,264,097	748,401	1,163,024	149
kdd10b	19,264,097	748,401	29,890,095	15
kdd12	119,711,284	29,927,821	54,686,452	24
MNIST38	11,982	1,984	752	2,680
news20	15,997	3,999	1,355,191	5
rcv1	20,242	677,399	47,236	32
real-sim	57,848	14,461	20,958	142
url	1,916,904	479,226	3,231,961	69
webspam	280,000	70,000	254	93,827
yahoo-japan	140,963	35,240	832,026	23

Table 4.1: Statistics of data sets.

kernel SVM, the Gaussion (RBF) kernel (2.2) is considered. Therefore, parameters to be selected are (C, γ) for kernel SVM, and C for for linear SVM. For linear SVM, LIBLINEAR implements an approach in Chu et al. (2015) to quickly find a suitable C value. For kernel SVM, we conduct cross validation (CV) on $C \in \{2^{-10}, 2^{-8}, 2^{-6}, \dots, 2^{10}\}$ and $\gamma \in \{2^{-10}, 2^{-8}, 2^{-6}, \dots, 2^{10}\}$ to select the one with the highest CV accuracy.

4.1.2 Random Forests

We consider the random forests implementation in Scikit-learn (Pedregosa et al., 2011), which implements the code by themselves. In Scikit-learn, they have the following options on the random forests models:

1. n_estimators: The number of trees in the random forests.
2. criterion: The function to measure the quality of a split.
3. max_features: The number of features to consider when looking for the best split.
4. max_depth: The maximum depth of the tree.
5. min_samples_split: The minimum number of samples required to split an internal node.

doi:10.6342/NTU201702065

6. min_samples_leaf: The minimum number of samples required to be at a leaf node.

Because of the many parameters, it is not practical to check all combinations of parameter values. To save the running time, we consider only two or three candidates for each value. They are listed below.

$$\begin{aligned}
\text{n\_estimators} &= \{100\},\\
\text{criterion} &= \{\text{Gini}, \text{Entropy}\},\\
\text{max\_feature} &= \{\sqrt{n}, 0.001n, 0.002n, 0.005n, 0.01n, 0.02n, 0.05n, 0.1n, 0.2n, 0.5n\},\\
\text{max\_depth} &= \{\text{unlimited}, 200, 100\},\\
\text{min\_sample\_split} &= \{2, 4\},\\
\text{min\_sample\_leaf} &= \{1, 2\}.
\end{aligned}$$

Note that parameters min_sample_split, min_sample_leaf and max_depth are related. Recalled that the training time is related to the complexity $O(N \times d \times p \times l)$. When we have more instances, a smaller number of max_feature (p) should be chosen. Otherwise, the training time may be too long. Therefore, for the data sets url, kdd10b and kdd10b-raw, we have to limit the max_depth to $\{10, 20, 50\}$, reduce the n_estimators and the max_feature to $\{20\}$ and $\{0.0001n, 0.0002n, 0.0005n, 0.001n\}$, respectively. For url, we reduce the max_feature to $\{0.0001n, \ldots, 0.01n\}$. For the data set kdd12, we only run the parameters

$$\{\text{ n\_estimators, criterion, max\_feature, max\_depth }\} = \{20, \text{gini}, 0.0001n, 10\}$$

in a reasonable time. Note that, we also cancel the cross validation on these large data sets when we train the random forests model, and change to use holdout validation with a split ratio $4 : 1$ (training : validation). In addition, we change min_sample_leaf and min_sample_split to $\{1\}$ and $\{2\}$, respectively, because we observe that they do not impact the test accuracy very much.

doi:10.6342/NTU201702065

4.1.3 GBDT

We consider the GBDT implementation in XGBoost (Chen and Guestrin, 2016) and LightGBM (Meng et al., 2016). In XGBoost, they have following options on the GBDT models:

1. rounds: The number of trees in the GBDT.
2. eta: The learning rate.
3. max_depth: The maximum depth of the tree.
4. min_child_weight: Minimum sum of instance hessian (h_{ik}) needed in a child.
5. gamma: The regularization of the number of the leaves.
6. lambda: The regularization of the value in the leaves (s_k^m).

In LightGBM, they have the following options:

1. num_trees: Same with rounds in XGBoost.
2. learning_rate: Same with eta in XGBoost.
3. max_depth: Same in XGBoost.
4. min_sum_hessian_in_leaf: Same with min_child_weight in XGBoost.
5. min_gain_to_split: Same with gamma in XGBoost.
6. lambda_l2: Same with lambda in XGBoost.
7. num_leaves: The maximum number of the leaves.
8. min_data_in_leaf: The minimum number of samples required to be at a leaf.
9. max_bins: The maximum number of the bins.

Both of them also support the subsampling on the features to decide p and the subsampling on the instances to reduce the instance size l. In our experiments, we do not use

doi:10.6342/NTU201702065

the subsample options on features and instances. Note that the regularization parameter lambda and gamma of the XGBoost which we do not claim in this thesis, but the details can be found in Chen and Guestrin (2016).

Similar to the random forests, we consider two or three candidates for each parameter value. For XGBoost,

$$\begin{aligned}
\text{rounds} &= \{100\},\\
\text{eta} &= \{0.1, 0.2, 0.3, 0.4, 0.5\},\\
\text{max\_depth} &= \{4, 5, 6, 7, 8, 9\},\\
\text{min\_child\_weight} &= \{0, 1, 2\},\\
\text{gamma} &= \{0, 0.1, 1\},\\
\text{lambda} &= \{0, 1, 10\}.
\end{aligned}$$

For LightGBM,

$$\begin{aligned}
\text{num\_trees} &= \{100\},\\
\text{learning\_rate} &= \{0.1, 0.2, 0.3, 0.4, 0.5\},\\
\text{max\_depth} &= \{\text{unlimited}\},\\
\text{min\_sum\_hessian\_in\_leaf} &= \{0, 1, 2\},\\
\text{min\_gain\_to\_split} &= \{0\},\\
\text{lambda\_l2} &= \{0, 1, 10\},\\
\text{num\_leaves} &= \{15, 31, 63, 127, 255, 511\},\\
\text{min\_sample\_leaf} &= \{10, 50, 100\},\\
\text{max\_bins} &= \{255\}.
\end{aligned}$$

Because the number of the parameters in LightGBM is larger than that of XGBoost, we set max_bins, min_gain_to_split and max_depth to the default setting. Note that we do not use the histogram technique and unblanced tree in XGBoost, but we use them in LightGBM.

doi:10.6342/NTU201702065

Futhermore, we consider not only the GBDT model, but also the linear boosting model in XGBoost. The parameters are shown at the following.

1. rounds: The number of trees in the GBDT.
2. eta: The learning rate.
3. lambda: The regularization of the linear regression weight w.

For a linear model, we may consider more candidates for each parameter,

$$\begin{aligned} \text{rounds} &= \{100\}, \\ \text{eta} &= \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}, \\ \text{lambda} &= \{2^{-10}, 2^{-9}, \ldots, 2^{10}\}. \end{aligned}$$

Note that, when we use XGBoost to train the data set kdd10b-raw and url, we use the holdout validation with the ratio $4 : 1$. We also reduce the search region of the data set kdd10b-raw which we let gamma, lambda and min_child_weight be the default value.

4.2 Result

We run the experiments on a machine with a 10-core CPU i7-6950X and 128GB RAM. For LIBLINEAR and LIBSVM, we run them by using a single thread. For the others, we use ten threads. We compare the following methods.

SVM_LR	:	SVM with linear kernel and LR loss.
SVM_RBF	:	SVM with Gaussion kernel.
RF	:	Random forests.
XGB_linear	:	XGBoost with using linear regression.
XGB	:	XGBoost with using regression tree.
LightGBM	:	LightGBM.

doi:10.6342/NTU201702065

Data sets	SVM_LR	SVM_RBF	RF	XGB_linear	XGB	LightGBM
astro-physic	96.81	97.31	96.67	96.99	96.30	96.38
cod-RNA	95.11	96.67	96.68	94.48	96.79	96.80
covtype	75.77	96.11	97.65	75.32	95.73	97.39
ijcnn1	91.96	98.69	98.21	92.01	98.07	98.32
kdd10b-raw	89.05	-	88.77	88.07	89.05	89.43
kdd10b	88.83	-	86.09	-	-	87.96
kdd12	95.56	-	95.55	-	-	96.58
MNIST38	96.82	99.70	99.09	96.93	99.29	99.40
news20	97.10	96.90	89.67	97.50	93.87	95.12
rcv1	97.57	-	97.67	97.67	97.58	98.11
real-sim	97.60	97.82	96.17	97.87	98.43	95.43
url	97.80	-	98.97	99.47	99.35	99.64
webspam	92.35	99.26	99.03	92.58	99.23	99.27
yahoojp	92.97	93.31	92.24	93.08	93.40	92.22

Table 4.2: Test accuracy

The results presented in the Table 4.2 are test accuracy in percentage. Because some data sets are too large to be solved by some methods, we use the notation '-' to denote that the results are not available. (Training time is greater than 30,000 seconds, or out of memory) Table 4.3 presents the training time (in seconds) with the best parameters after the grid search.

In Table 4.2, we give the result on the following cases:

1. linear $\approx$ kernel $\approx$ RF $\approx$ GBDT: astro-physic, real-sim and yahoojp.

2. kernel $\approx$ RF $\approx$ GBDT $>$ linear: cod-RNA, ijcnn1, MNIST38 and webspam.

3. linear $>$ kernel $\approx$ RF $\approx$ GBDT: kdd10b.

4. linear $\approx$ RF $\approx$ GBDT $>$ kernel: kdd10b-raw and rcv1.

5. RF $\approx$ GBDT $>$ kernel $>$ linear: covtype.

6. linear $\approx$ kernel $>$ GBDT $>$ RF: news20.

7. GBDT $>$ RF $\approx$ linear $>$ kernel: kdd12.

8. GBDT $>$ RF $>$ linear $>$ kernel: url.

Where linear including SVM_LR and XGB_linear, GBDT including XGB and lightGBM. On the other hand, in Table 4.3, we give the result as the followed:

doi:10.6342/NTU201702065

Data sets	SVM_LR	SVM_RBF	RF	XGB_linear	XGB	LightGBM
astro-physic	2.20	2076.63	38.74	3.50	12.22	5.57
cod-RNA	0.30	411.17	6.33	0.76	1.56	1.61
covtype	2.28	26796.28	131.87	2.80	13.98	3.99
ijcnn1	0.29	45.52	7.65	0.52	1.58	0.92
kdd10b-raw	102.63	-	142.08	306.73	19498.48	467.48
kdd10b	278.19	-	16254.93	-	-	1306.15
kdd12	1110.45	-	4054.71	-	-	2081.70
MNIST38	0.66	13.95	1.80	0.98	1.18	0.81
news20	4.72	935.79	31.59	3.86	45.15	28.90
rcv1	32.27	-	1497.72	10.75	141.55	185.86
real-sim	1.54	1370.17	12.11	1.95	8.94	4.01
url	81.48	-	2421.92	209.98	564.11	353.44
webspam	9.40	6519.51	390.64	10.35	75.05	7.92
yahoojp	15.39	10026.14	1911.26	13.24	48.15	20.91

Table 4.3: Training time

1. kernel $\gg$ RF $>$ GBDT $>$ linear: astro-physic, cod-RNA, covtype, ijcnn1 and rcv1.

2. kernel $\gg$ RF $\approx$ GBDT $>$ linear: news20 and real-sim.

3. kernel $\gg$ RF $>$ GBDT $>$ XGB_linear $>$ SVM_LR: url.

4. kernel $\gg$ RF $\approx$ GBDT $\approx$ linear: MNIST38.

5. kernel $\gg$ XGB $\gg$ lightGBM $>$ XGB_linear $>$ RF $\approx$ SVM_LR: kdd10b-raw.

6. kernel $\approx$ XGB $\approx$ XGB_linear $\gg$ RF $>$ lightGBM $>$ SVM_LR: kdd10b and kdd12.

7. kernel $\gg$ RF $>$ XGB $>$ lightGBM $\approx$ linear: webspam and yahoojp.

We may observe that kernel method (RBF kernel) cost too much training time, it is not suitable in large-scale data sets. For the tree-based methods, lightGBM gets a beautiful training time, and histogram is very useful in GBDT. The random forests is disadvantaged on the implementation, we do not know the situation that the histogram technique is used with the random forests. For the linear models, the speed of training time is very fast, it is important for analyzing the large-scale data sets, but the performence may not exceed tree-based models and kernel method.

doi:10.6342/NTU201702065

Chapter 5

Conclusion

After analyzing the model of random forests and GBDT, we know that the essence of these two models is totally different. Random forests uses the law of large number to make the prediction of the model be the expectation of it. On the other hand, GBDT adds the submodels as the descent direction of its loss function. This difference may be a reason that GBDT performs better than random forests in some sparse data sets. For the dense data, their performences are similar. Furthermore, according to our experiments, GBDT with the histogram technique is fast enough for the large-scale data sets, and the test accuracy is performed well. There is a disadvantage part of parameter selection in tree-based methods, this disadvantage makes the total training time be longer.

doi:10.6342/NTU201702065

Bibliography

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

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

L. Breiman, J. Friedman, C. J. Stone, and R. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC; 1 edition, 1984.

C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/ cjlin/libsvm`.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD '16: Proceedings of the 22th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016. URL `http://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf`.

B.-Y. Chu, C.-H. Ho, C.-H. Tsai, C.-Y. Lin, and C.-J. Lin. Warm start for parameter selection of linear classifiers. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015. URL `http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/warm-start/warm-start.pdf`.

C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

doi:10.6342/NTU201702065

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/ cjlin/papers/liblinear.pdf`.

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014. URL `http://jmlr.org/papers/v15/delgado14a.html`.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann. Statist.*, 28(2):337–407, 04 2000. doi: 10.1214/aos/1016218223. URL `http://dx.doi.org/10.1214/aos/1016218223`.

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

J. H. Friedman and J. J. Meulman. Multiple additive regression trees with application in epidemiology. *Statistics in Medicine*, 22(9):1365–1381, 2003. ISSN 1097-0258. doi: 10.1002/sim.1501. URL `http://dx.doi.org/10.1002/sim.1501`.

R. Jin and G. Agrawal. Communication and memory efficient parallel decision tree construction. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 119–129. SIAM, 2003.

S. S. Keerthi and C.-J. Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation*, 15(7): 1667–1689, 2003. doi: 10.1162/089976603321891855. URL `http://dx.doi.org/10.1162/089976603321891855`.

P. Li. Robust logitboost and adaptive base class (ABC) logitboost. *CoRR*, abs/1203.3491, 2012. URL `http://arxiv.org/abs/1203.3491`.

Q. Meng, G. Ke, T. Wang, W. Chen, Q. Ye, Z.-M. Ma, and T. Liu. A communication-efficient parallel algorithm for decision tree. In D. D. Lee, M. Sugiyama, U. V.

doi:10.6342/NTU201702065

Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1279–1287. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6381-a-communication-efficient-parallel-algorithm-for

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, third edition, 2011.

doi:10.6342/NTU201702065