

FBP: A Frontier-Based Tree Pruning Algorithm

Xiaoming Huo, Seoung Bum Kim, Kwok L. Tsui, Shuchun Wang
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia
30332-0205, USA, {xiaoming, sbkim, tsui, swang1@isye.gatech.edu}

A frontier-based tree-pruning algorithm (FBP) is proposed. The new method has comparable order of computational complexity, comparing with Cost-Complexity Pruning (CCP). Regarding tree pruning, it provides a full spectrum of information; specifically, (1) given the value of the penalization parameter λ , it gives the minimum size of a decision tree; (2) given the size of a decision tree, it provides the range of the penalization parameter λ , within which the complexity-penalization approach will render such a tree size; (3) it finds the sizes of trees that are *inadmissible* — no matter what the value of the penalty parameter is, the resulting tree based on a complexity-penalization framework will never have these sizes.

Simulations on real datasets reveals a ‘surprise’: in the complexity-penalization approach, most of the tree sizes are inadmissible. FBP facilitates a more faithful implementation of the principle: Cross-Validation (CV). Simulations seem to favor such an approach. Utilizing FBP, a stability analysis for CV is proposed.

Key words: decision trees; pruning methods; data mining; classification

1. Introduction

Due to their flexibility and interpretability, tree-based methods have gained enormous popularity in statistical modelling and data mining. A recent paper by Li and his coauthors (Li et al. 2001) gives an excellent survey on tree-building (including tree-growing and tree-pruning) techniques. An important technical question in building a statistically optimal tree is to find an optimal strategy to *prune* a huge tree. To avoid repetition, we will simply refer to Li et al. (2001) for relevant references.

This paper is focused on the methodology of tree pruning. The concept of “complexity-penalization” was well adopted in this topic. The essence of this approach is to solve the following optimization problem:

$$\min_T L(T) + \lambda|T|, \tag{1.1}$$

where

- T denotes an ‘admissible’ subtree, which can be corresponded to a partition,
- function $L(T)$ is the error rate of the corresponding decision tree,
- λ is the penalization parameter, which was mentioned earlier, and
- $|T|$ denotes the size of the tree, which normally is equal to the number of ‘leaves’ (i.e., terminal nodes) in tree T .

Such an approach was essentially the objective that was embedded in CCP (Breiman et al., 1984). In this tree pruning framework, there are two interwoven quantities, which are

1. the value of the penalization parameter (i.e., λ), and
2. the size of the tree.

While considering the performance of a tree model on testing data, one needs to consider *generalization error*. We intentionally leave it out in the above list, because it is not essential in designing the algorithm. It eventually will be used in evaluating the obtained decision tree.

In pruning a large tree, the following four questions are of interests.

1. Given the value of the penalization parameter, applying the criterion of “minimizing the complexity-penalized loss function” (which will be elaborated later), what will be the size of the pruned tree?
2. Given the target tree size, what is the range of the penalization parameter, λ , so that when the principle of “minimum complexity-penalized loss function” is applied and the parameter λ is in this range, the size of the pruned tree is equal to this target size?
3. Are there some sizes of trees, for which no value of the penalization parameter will render pruned trees that are of these sizes? We will call these occasions the cases of *inadmissibility*.
4. Given the unseen data, does the pruning method help improve classification accuracy? Or in other words, can the generalized error be reduced? This will be addressed by combining with CV.

The first three questions are related to an algorithmic parameter. The last one is on the generalization error. Answering these questions evidently gives users insights on which tree model should be chosen.

The key contribution of this paper is to propose a different algorithm in tree pruning. Apart from the local greed approach that was adopted in CCP (Breiman et al., 1984), the proposed method — FBP — keeps *all* the useful information, and propagate them in a bottom-up fashion to the top layer (i.e., the root node). Specific algorithm is designed, so that the above can be achieved efficiently — having nearly the same order of complexity as running CCP.

The proposed algorithm can automatically and simultaneously answer the first three of the four questions raised previously. Identification of inadmissible tree sizes has not been considered in any other tree pruning methods; although it seems to be observed in several occasions — see the justification of a dynamic programming approach in the paper (Li et al. 2001). To our knowledge, it is the first time to address it explicitly in a *quantitative* manner.

The proposed method has a very nice graphic interpretation, which has a connection with the Pareto-Optimality, and is highly intuitive.

To prune a single tree, both CCP and FBP solve the same problem, which is to minimize the objective in (1.1). However, because FBP provides an entire spectrum of information on pruning, it facilitates a more faithful realization of the cross validation principle. In simulations, it is shown that such a difference can lead to improvement in the testing errors.

Also due to the design of FBP, one can study whether a CV is stable. More specifically, one may ask if the partitioning that is called by CV can significantly affect the output of CV? We study this problem through simulations. Because of the availability of FBP, nice illustration can be generated.

The rest of the paper is organized as follows. We preview the key motivation of the algorithmic design in Section 2. To make necessary preparation, Section 3 reviews the complexity-penalized tree-pruning algorithm that was mentioned at the beginning of this paper. Section 4 describes the proposed algorithm: FBP. Section 5 describes how the frontier-based tree-pruning method can be integrated with cross validation, and how the CV in FBP is different from the CV in CCP. A study of the stability of CV will be described too. Section 6 presents simulation studies. Section 7 describes the architecture of a software, which was developed to carry out all of the simulations. (The software is available under request.) Some concluding marks are made in Section 8.

2. The main idea of the algorithm

As an illustration of the main idea of the proposed approach, let us study the graph in Figure 1, in which the x-axis is the value of the penalizing parameter, λ . (The value of λ was considered the cost of an additional node in a tree.) For a given λ , when the target size of the tree is m , the minimum value of the Complexity-Penalized Loss Function (CPLF) is $c_m + m\lambda$, where c_m is a constant (i.e., the intercept). In the same graph, let the y-axis be the value of the CPLF. A function “ $c_m + m\lambda$ ” represents a straight line in the graph. Given λ , the slope, m , of function “ $c_m + m\lambda$ ” is the size of the subtree. Let us define a function

$$f(\lambda) = \min_{m=1,2,\dots} \{c_m + m\lambda\}.$$

For a given value λ_0 , the slope of function $f(\lambda_0)$ is the size of the optimal subtree. It is not hard to see that function $f(\cdot)$ is piecewise linear. Given a fixed integer (denoted by m_0), there is an interval of λ , within which the slope of the function $f(\cdot)$ is equal to m_0 . Or in other words, if λ takes value inside this interval, the size of the optimal subtree is equal to m_0 . To check if a specific tree size (e.g., m_0) is optimal, one can check if the line “ $c_m + m_0\lambda$ ” composes the part of the curve that is associated with the function $f(\cdot)$.

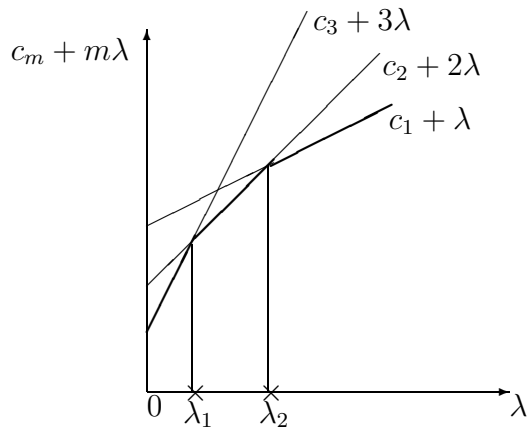


Figure 1: An illustration for the *frontier-based tree-pruning* algorithm.

A lower bound in a bundle of lines in a graph is associated with the size of optimal subtree (within the domain of the parameter λ); this is analogous to an efficient frontier in Investment Theory (Luenberger, 1998), in which an efficient frontier is a curve representing a set of efficient portfolios that maximize expected returns at each level of portfolio risks. For more details, we refer to the paper by Harry Markowitz (1952).

This interpretation illustrates why our method is named *frontier-based tree-pruning* (FBP) algorithm.

Readers can imagine that in tree-pruning, the key problem is that given two sets of lines, how to find their common lower bound? More challengingly, how to achieve this with the lowest possible computational cost? Readers can investigate our algorithms (Section 4) for answers.

3. Tree Pruning

Tree methods in data analysis and modelling were ‘boosted’ by the book (Breiman et al. 1984). Since then, tremendous literature has been developed. A survey in Li et al. (2001) indicates that in building a tree, the relatively important part is to prune a redundant tree. In this paper, we adopted this conclusion without further justification. Tree methods have been extremely successful in data mining. For example, nowadays the most popular data mining tools—CART and MARS (CART&MARS)—are based on tree methods. Researchers have studied various issues in building an “optimal” tree, see Buja and Lee (2001), etc. In this paper, we assume that a big and redundant tree has been built.

The complexity-penalized tree-pruning approach has been described in Breiman et al. (1984), and it has been well known in statistics. The idea is originally rooted in statistical model selection. A significant advantage of adopting this principle is that people can prove various optimality results, e.g., Donoho (1997), Donoho (1999), etc. In the current paper, we will not prod further in that direction, just emphasizing the importance of this approach.

In the rest of this section, we first review the general principle of complexity-penalized tree-pruning (Section 3.1). A bottom-up tree pruning algorithm is described (Section 3.2).

3.1 The principle of Minimizing Complexity-Penalized Loss Function

We give more details on a previously mentioned idea. The objective of a complexity-penalized tree-pruning approach is (from a big redundant tree) to find a subtree that minimizes the Complexity-Penalized Loss Function (CPLF). The principle of minimizing CPLF will be described in the following. Let T denote a big un-pruned tree. In tree modelling, each tree is associated with a *recursive dyadic partitioning* (RDP) of a state space. Without loss of generality, we focus our attention on binary trees. Due to their simplicity and interpretability,

binary trees are often used in tree-based algorithms. The size of a tree (or a subtree) is the number of *terminal* nodes, which is also equal to the number of regions in an RDP of the state space. If two regions in an RDP are merged, the size of the tree is reduced by one. In tree pruning, the region merging is equivalent to pruning a two-leave branch from a binary tree. A subtree is a tree that can be obtained by repeating the above procedure. A subtree is denoted by T_b , where b is an index for subtrees. Since each subtree is associated with a partition of the state space, one can choose the tree that minimizes a criterion function. In this case, this criterion function is the *complexity-penalized loss function*, which is defined in the following. Let $L(T_b)$ denote the loss function associated with tree T_b . Let $|T_b|$ denote the size of the tree T_b . The *complexity-penalized loss function* is equal to

$$L(T_b) + \lambda|T_b|,$$

where λ is a penalizing parameter. The principle of minimum CPLF is to choose a subtree T_b that minimizes the CPLF. In other words, it is to solve

$$T_{b_0} = \operatorname{argmin}_{T_b} \{L(T_b) + \lambda|T_b|\}. \quad (3.2)$$

3.2 Bottom-up Tree-Pruning Algorithm

There is a well-known algorithm that can efficiently solve the above problem. The algorithm is called a *bottom-up tree-pruning algorithm*. This algorithm uses the same idea that has been used in other algorithms such as *Best Basis* (Coifman and Wickerhauser, 1992), which is originated in the wavelet literature. This algorithm is also described in Breiman et al. (1984). To illustrate the idea, a graph is provided in Figure 2.

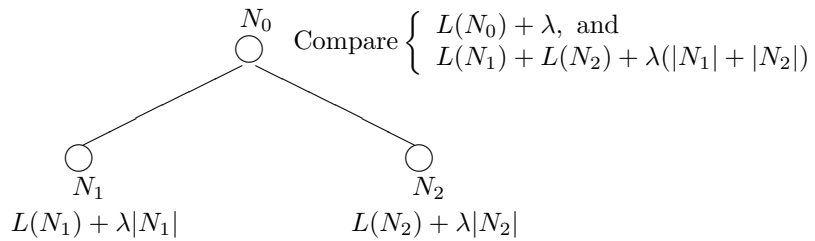


Figure 2: An illustration on bottom-up tree pruning here.

We consider two terminal nodes N_1 and N_2 in a binary tree. Their common parent is N_0 . Let $L(N_0)$, $L(N_1)$, and $L(N_2)$ denote the values of the loss function for the nodes N_0 , N_1 , and N_2 respectively. The CPLF that is associated with N_1 and N_2 are $L(N_1) + \lambda$ and

$L(N_2) + \lambda$ respectively. Recall that in a binary tree, each node is associated with a region in an RDP of the state space. When one goes up to the node N_0 , there are two possibilities: splitting the region into two regions N_1 and N_2 , or no splitting. If the region N_0 is split, then value of the CPLF should be

$$L(N_1) + L(N_2) + 2\lambda,$$

otherwise, value of the CPLF should be

$$L(N_0) + \lambda.$$

Which one should be chosen depends on which value of the CPLFs is smaller. In a bottom-up tree-pruning algorithm, one starts at the bottom of a tree (or terminal nodes), and then repeatedly apply the above procedure until the top (root) node is reached. Note that when the nodes N_1 and N_2 are not terminal nodes, the expressions of their penalization functions are different. They should be $\lambda|N_1|$ and $\lambda|N_2|$. In the parent node (N_0), the comparison is between

$$L(N_1) + L(N_2) + \lambda(|N_1| + |N_2|), \quad (\text{splitting,})$$

and

$$L(N_0) + \lambda, \quad (\text{no splitting.})$$

which are also indicated in Figure 2.

It is proven that the bottom-up tree-pruning algorithm finds the subtree that minimizes the CPLF, see Breiman et al. (1984).

One advantage of the bottom-up tree-pruning algorithm is that it is computationally efficient. If the un-pruned tree has size $|T|$, it takes no more than $O(|T|)$ operations to find the minimizer of the problem (3.2).

The bottom-up tree-pruning approach has some intrinsic links with other approaches. For example, in Breiman et al. (1984) and Li et al. (2001), a method based on analyzing the reduction of the total loss function is discussed. This method is called “cost complexity pruning”. Apparently, in the bottom-up tree-pruning algorithm, the choice of every step depends on whether the reduction of the loss function (following splitting the parent node, which is equal to $L(N_0) - L(N_1) - L(N_2)$) is larger than the value of the parameter λ (or the reduction of the penalization function $\lambda(|N_1| + |N_2| - 1)$ when the nodes N_1 and N_2 are nonterminal). This will lead to the discussion of a pruning algorithm in the original CART book (Breiman et al. 1984).

4. Frontier-based Tree-pruning Algorithm (FBP)

There are five subsections. Section 4.1 describes the frontier-based tree-pruning algorithm. Section 4.2 interprets the occurrences of inadmissible tree sizes. Section 4.3 provides a fast algorithm in numerically realizing the idea in Section 4.1. Section 4.4 gives the computational complexity of our approach. Finally, Section 4.5 explains the connection between the frontier-based method and the dynamic-programming based method.

4.1 Algorithm

Now we start describing the frontier-based pruning approach. The key point is to create a list of linear functions that have the form “ $c + m\lambda$ ” at each node, in which c is the value of a loss function, and m is the size of a subtree. At the root node, the information is summarized.

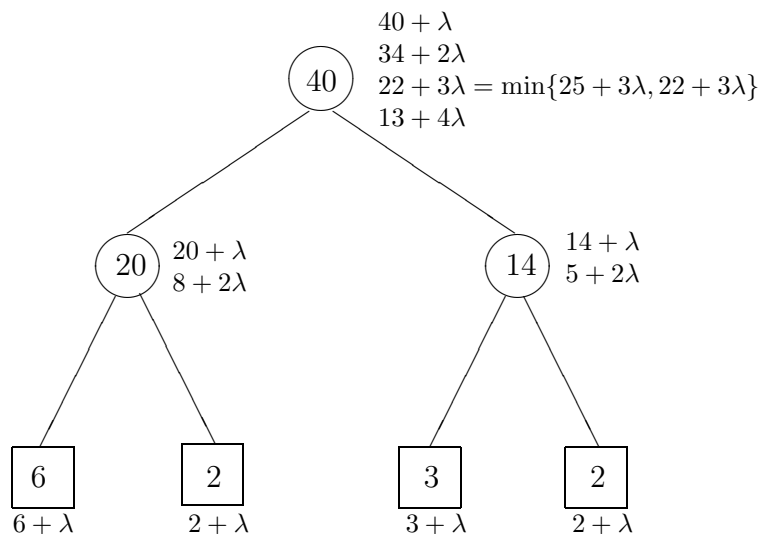


Figure 3: An example of the frontier-based tree-pruning approach.

To explain the frontier-based tree-pruning algorithm, we use an example that is illustrated in Figure 3, in which a circle indicates an intermediate node and a square indicates a terminal node. The values in the circles and squares are the values of the loss function. Let λ denote the penalizing parameter. At each node, all the possible expressions of the CPLF are listed as the linear functions of λ . For example, at all terminal nodes, the CPLFs have form $c + \lambda$, where c is the value of the loss function. When one goes up in the tree, at an intermediate node, the expressions for the CPLFs have forms $c_m + m\lambda$, $m = 1, 2, \dots$, where c_m is the value of the loss function when the size of the tree is equal to m . Each node will have a list of

linear functions. At each node, we only need to determine the sequence of c_m 's. The list at the parent node can be derived from the list at the two siblings. For example, for the node where a value 20 is inside a circle, the value of c_1 should be 20, and the value of c_2 should be $6 + 2 = 8$. There should be two linear functions at this node. Sometimes, the value of intercept c_m is not uniquely defined. For example, at the root node of the above example, when the size of the tree is equal to 3, the intercept should take the minimum value between $25 = 20 + 5$ and $22 = 8 + 14$. We start from the bottom of the tree, the lists of linear functions are built (following a bottom-up strategy) for all nodes.

The list building, which is described above, is the first step of the frontier-based pruning algorithm. Consequently, the list at the root node is processed in the following way: in a Cartesian plane, the x-axis is taken to be the value of λ , and the y-axis is the value of the linear functions ($c_m + m\lambda$). All the linear functions are plotted on this plane. An illustration of these linear functions is in Figure 1. The lower bound of these functions is considered. This function is defined previously:

$$f(\lambda) = \min_{m=1,2,\dots,T} \{c_m + m\lambda\},$$

where T is the number of terminal nodes, and $c_m + m\lambda$ are linear functions at the root node. The following observations indicate how to use f :

1. For fixed λ , the value $f(\lambda)$ gives the minimum CPLF.
2. For a fixed size of the tree m_0 , let (a_0, b_0) denote the interval of the parameter λ such that when the λ is taking value inside this interval, the slope of the function $f(\lambda)$ is equal to m_0 . On the other hand, to get a subtree that has m_0 terminal nodes, the value of λ should be chosen within the interval (a_0, b_0) .
3. It is possible that for an integer m_0 , there is no part of curve $f(\lambda)$ such that its slope is equal to m_0 . In this case, the tree size m_0 is inadmissible — no matter what the value of the λ is, in the minimum CPLF approach, the final tree size will never be m_0 . An illustration of this case can be found in Figure 4.

4.2 Inadmissibility

It is possible that for a certain tree size, no matter what the value of the parameter λ is, the minimum CPLF approach will not render a tree that is of this size. In such a case, this

tree size is called *inadmissible*, which is analogous to inadmissible estimators in statistics. Apparently the definition of *inadmissibility* relies on the minimum CPLF procedure. Even if a tree size is inadmissible, it does not necessarily imply that the tree of this size is not optimal with respect to other criteria. We would like to emphasize the dependence of this definition on the CPLF, and warn the possible confusion between *admissibility* and *optimality*.

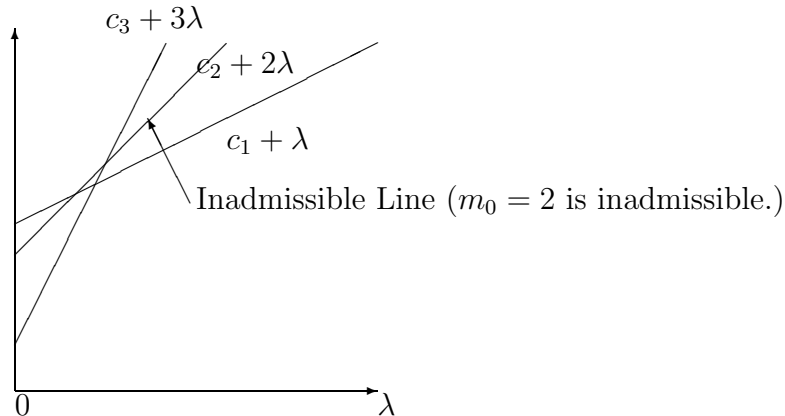


Figure 4: An illustration of an inadmissible case.

An illustration of the inadmissible case is given in Figure 4. In the frontier-based tree-pruning algorithm, the inadmissible cases can be quickly identified. A fast numerical algorithm, which identifies admissible tree sizes as well as associated intervals, is presented in Section 4.3.

For the example in Figure 3, the tree sizes 2 and 3 are inadmissible, because

$$34 + 2\lambda \geq \min(40 + \lambda, 13 + 4\lambda), \quad \text{and}$$

$$22 + 3\lambda \geq \min(40 + \lambda, 13 + 4\lambda).$$

So the linear functions $34 + 2\lambda$ and $22 + 3\lambda$ are dominated by linear functions $40 + \lambda$ and $13 + 4\lambda$.

4.3 Algorithm to Find a Lower Bound in a Bundle of $a\lambda + b$ Lines

An algorithm that finds the lower bound of a bundle of lines having form “ $k\lambda + c_k$ ” is described here. Here slope k is a positive integer, λ is a variable, and c_k ’s are the intercepts. When the number of lines is N , this algorithm finds the lower bound with at most $O(N)$ operations.

Formation. Suppose the lines are $\lambda + c_1, 2\lambda + c_2, 3\lambda + c_3, \dots, N\lambda + c_N$. The lower bound of them is the function

$$f(x) = \min_{1 \leq k \leq N} \{k\lambda + c_k\}.$$

Apparently, $f(x)$ is a piecewise linear function, which is determined by (1) the positions where the slope changes and (2) the constant slope within each interval. Recall that in tree pruning, we have

$$c_1 > c_2 > c_3 > \dots > c_N.$$

Algorithm. We start with the following table.

slope: 1
positions: 0

This table indicates that there is one interval: $(0, +\infty)$. Within this interval, the slope is 1.

Now we take into account the line $2\lambda + c_2$. A new table is established,

slope: 2	1
positions: 0	$x_{1,2}$

where $x_{1,2}$ is the intersecting position of lines $\lambda + c_1$ and $2\lambda + c_2$. This is done by inserting a slope 2 at the beginning of the *slope* row and an intersecting position $x_{1,2}$ at the end of the *position* row.

To illustrate the idea, now consider adding the line $3\lambda + c_3$. Recall that if

$$3x_{1,2} + c_3 < 2x_{1,2} + c_2, \tag{4.3}$$

then the line $3\lambda + c_3$ is lower than the line $2\lambda + c_2$ at the position $x_{1,2}$. Hence the line $2\lambda + c_2$ is always above the minimum of the line $\lambda + c_1$ and the line $3\lambda + c_3$. In other words, line $2\lambda + c_2$ is dominated; we do not need to consider the line $2\lambda + c_2$. Based on the above, a new table should be

slope: 3	1
positions: 0	$x_{1,3}$

where $x_{1,3}$ is the position where the line $\lambda + c_1$ and $3\lambda + c_3$ intersect. If inequality (4.3) is untrue, the line $2\lambda + c_2$ is not dominated. The new table should be

slope: 3	2	1
positions: 0	$x_{2,3}$	$x_{1,2}$

no more than $N + T(T - 1)$. Moreover, in a binary tree, there is a relationship between N and T .

$$N = 2T - 1.$$

Therefore, the order of complexity is $N + \frac{1}{4}(N^2 - 1)$ for the minimum CPLF algorithm with a fixed λ . But this is the price to pay for more information.

Theorem 4.1 *Let N and T denote the number of nodes and terminal nodes in a binary tree. If we use N to express the complexity, it takes no more than $N + \frac{1}{4}(N^2 - 1)$ operations to generate the lists of linear functions at all nodes.*

Proof. There are two stages in our proof. Firstly, at all nodes, for terms like ‘ $c + \lambda$ ’, it takes N operations to create them. Secondly, for terms like ‘ $c + m\lambda$ ’, where $m \geq 2$, it can be shown that it takes no more than $T(T - 1)/2$ operations to calculate all the related linear functions, and it takes no more than $T(T - 1)$ operations to generate the lists. The second statement can be proved by induction. Here we explain the idea:

Assume that the second statement above is true for any tree with number of terminal nodes smaller than T . For a binary tree with T terminal nodes, let m_1 (resp., m_2) denote the number of terminal nodes that have the left (resp., right) child of the root node as an ancestor. Note here we follow the common terminology of a classification and regression tree. We must have

$$T = m_1 + m_2.$$

Based on the assumption, for each binary tree whose *root node* is one of the immediate children of the root node in the original tree, it takes $m_1(m_1 - 1)/2$ and $m_2(m_2 - 1)/2$ operations to compute all the linear functions. For the root node, it takes m_1m_2 operations to compute all the necessary linear functions. Altogether, the number of operations to compute linear functions is

$$m_1(m_1 - 1)/2 + m_2(m_2 - 1)/2 + m_1m_2 = (m_1 + m_2)(m_1 + m_2 - 1)/2 = T(T - 1)/2.$$

Note that to find the minimum values, in some cases, it takes no more than the steps needed to scan through all the sums. Hence it requires no more than $T(T - 1)/2$ operations. From all the above, the number of operations to create the lists of linear functions *at all nodes* is no more than $T(T - 1)$, which is equivalent to $\frac{1}{4}(N^2 - 1)$.

The summary of the above two facts leads to the proof of the theorem. □

4.5 Connection with the Dynamic-Programming Based Approach

In Li et al. (2001), a Dynamic-Programming Based Pruning (DPP) is proposed. The advantage of their method is that the DPP can find optimal trees with any sizes. In our Frontier-Based Pruning (FBP) algorithm, by tracing back the route of generating linear function $c_m + m\lambda$, one can extract the optimal size- m subtree. It can be shown that if the optimal subtree is unique and admissible, then both approaches will find the optimal one.

Based on the previous description on the FBP algorithm, the tracing back is straightforward. As an example, we study the case in Figure 3. Suppose one wants to find the optimal subtree that has 3 terminal nodes. The corresponding linear function at the root node is

$$22 + 3\lambda,$$

which is induced by

$$(8 + 2\lambda) + (14 + \lambda).$$

Moreover, the linear function $8 + 2\lambda$ is made by

$$(6 + \lambda) + (2 + \lambda).$$

Every time when a linear function having form $c + \lambda$ is reached, a terminal node is reached. Hence in this case, the optimal size-3 subtree is made by the first two terminal nodes (from the left) and the intermediate node with misclassification rate being equal to 14. The above procedure determines a subtree with size 3. As mentioned in Li et al. (2001), it is possible that the optimal subtree is not unique. In that case, one can design some ad-hoc rules to specify it.

We now describe the consistency between the two approaches.

Theorem 4.2 *Among the subtrees that have size m , the one with the smallest misclassification rate can be found by both DPP and FBP.*

Proof. We only prove the case when the optimal subtree is unique and the goodness of fit is measured by the number of misclassifications. In DPP, it is known that the algorithm finds the subtree with the smallest number of misclassifications. This can also be proved by showing that if there exists a subtree with the same size but smaller number of misclassifications, then the dynamic programming approach should end up with the subtree with smaller

amount of misclassifications. Because the DPP enumerates all the possibilities to prune the tree with minimum increment of number of misclassifications.

In the FBP, the algorithm also finds the subtree with the smallest amount of misclassifications. This can be proved by showing that if there is a subtree with smaller number of misclassifications, then at the root node, this subtree will generate a linear function with smaller intercept. This is contradictory to the definition in the FBP algorithm.

From all the above, the DPP and FBP should generate the same subtree. □

5. Integration with Cross Validation

The FBP algorithm can be used in Cross Validation (CV). We begin with the principle of CV, then describe how the FBP can be used in implementing the CV.

Suppose the observations are

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_M, y_M)\},$$

where M is the number of observations, x_i 's are predictor variables and y_i 's are responses. Note that x_i 's can be multivariate. Suppose the above set is (equally) partitioned into K subsets:

$$S_1 \cup S_2 \cup \dots \cup S_K.$$

At each time, we leave out one subset (say S_i) and use the remaining subsets to grow a tree and then prune a tree, the lower bound function will be denoted as $f_{-i}(\cdot)$. For each value of the parameter λ , the size of the optimal subtree and the subtree itself can be extracted. The optimal subtree determines a model. This model is then applied to the leave out subset (which is S_i). This is called *testing*. The error rate in testing can be computed, and is denoted by $e_{-i}(\cdot)$. Note that functions $f_{-i}(\cdot)$ and $e_{-i}(\cdot)$ are of the same variable. Since function $f_{-i}(\cdot)$ is a piecewise linear function, it is not hard to prove that function $e_{-i}(\cdot)$ is a piecewise constant function (or step function). The principle of CV is to find the value of the parameter λ such that the value of the average of e_{-i} 's,

$$\frac{1}{K} \sum_{i=1}^K e_{-i},$$

is minimized. Throughout this paper, the above quantity will be called *cross validation error* (CVE, or CV error).

This principle can be easily implemented by using FBP. The generation of functions $f_{-i}(\cdot)$ is already in the FBP. The generation of functions $e_{-i}(\cdot)$ can be easily implemented. A simulation example for the Cleveland Heart Disease data is presented in Figure 5. Based on that, we can conclude that the model made by using λ between 0.55 and 0.58 gives the minimum cross validation error rate.

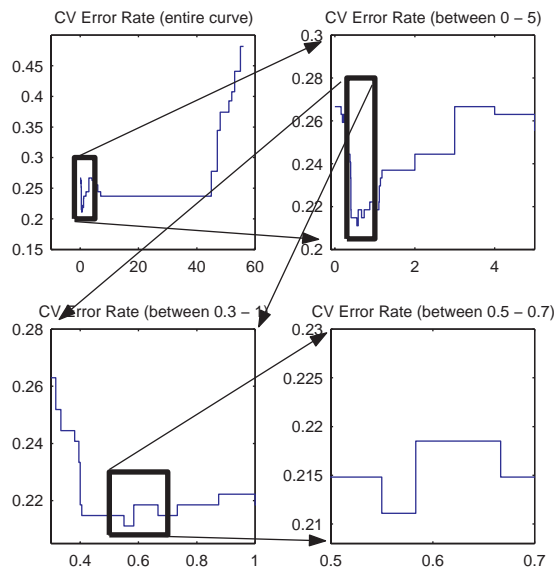


Figure 5: Using frontier-based pruning in cross validation. The minimum of the cross validation error was found by zooming in.

5.1 Numerical Analysis of the Stability of the Cross Validation Method

As mentioned earlier, we can use FBP to study (and graphically illustrate) the stability of the CV. Here we study a ten-fold CV. In each experiment, the dataset is randomly and equally partitioned into ten subsets. The CV is to leave out one subset at a time, and uses the rest nine subsets to train a model, then the left-out one is used to test (compute the testing error rate). This process is repeated for each subset, and the overall error rate is the average of the ten that are generated. Apparently in a ten-fold CV, the partition of the dataset will change the result of CV. If CV is stable, the variation that is introduced by different partition should be small. Is this always the case? By using FBP, people can develop graphical tools to examine this condition.

We choose to study the stability of the CV approach in the Wisconsin Breast Cancer data. In Figure 6, five CV curves are plotted. It shows that the optimal intervals for the penalization parameter λ are roughly in the same neighborhood.

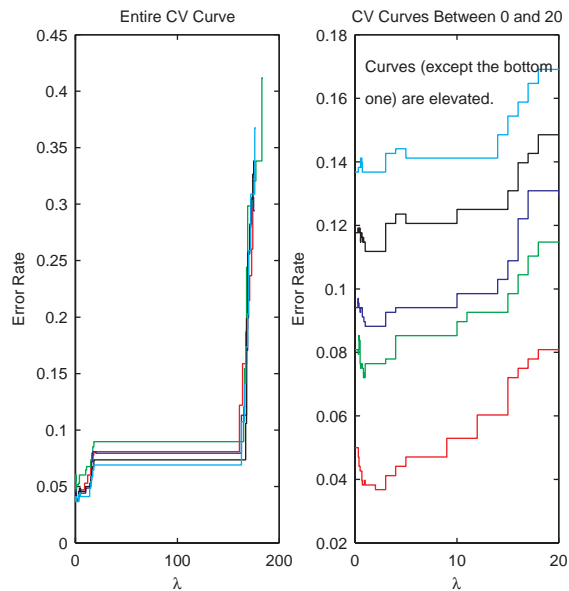


Figure 6: Five experiments of cross validation with the Wisconsin Breast Cancer data. The left panel shows the entire CV curves. The right one focuses on the region between $(0, 20)$, which is the interval that includes the minima in all five experiments.

In Figure 7, one hundred optimal intervals derived based on the CV principle are shown. Each row gives the location of the interval, inside which when the parameter λ takes value, the cross validation error is minimized. We will call these intervals the *optimal intervals*. The figure shows the locations of these intervals out of one hundred simulations. It demonstrates that even though in many cases, the optimal intervals overlap significantly, it is possible that in some cases, an optimal interval is dramatically different from most of others. This study demonstrates one application of FBP. Note that in Figure 7, the horizontal axis is in logarithmic scale.

More quantitative analysis on the above phenomenon will be an interesting future research project.

5.2 Difference between the CV in CCP and the CV in FBP

Cost-Complexity-Pruning (CCP) was advocated by Breiman et al (1984). Frontier-Based-Pruning (FBP) is the proposed method in this paper. A careful comparison between the CV

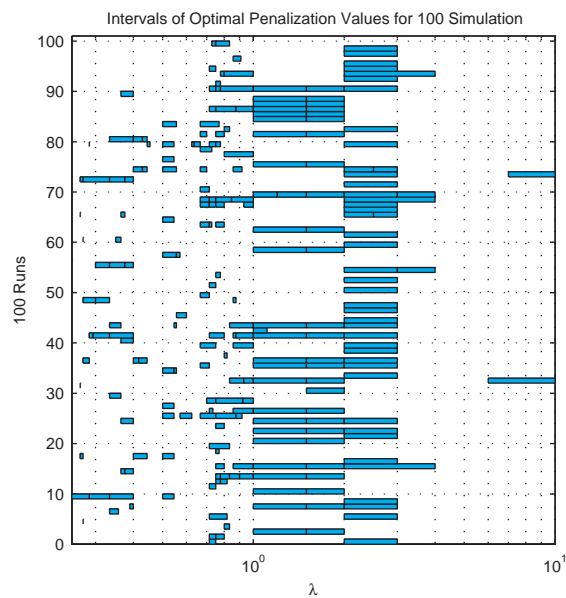


Figure 7: The locations of optimal intervals. Each row of horizontal bars indicates the locations of optimal intervals. Sometimes the cross validation principle renders several discontinuous intervals, which reflect in our figure as a chain of disconnected bars. This simulation shows that in most of the time, the cross validation principle will render similar optimal range of the penalization parameter λ . But it is possible that the partitioning of the dataset can dramatically influence the result of cross validation.

that is described in Breiman et al (1984) section 11.5 and the method that is described at the beginning of this section reveals a significant difference between the two. We argue that the proposed method (i.e., CV in FBP) more faithfully realizes the principle of CV, while the CV in CCP only examines a subset of potential cases. An illustration of a case when the difference occurs will be provided in the end of this section.

To explain the difference, let us recall the CV procedure in CCP. More details are available in the book (Breiman et al., 1984), hence they are omitted. In CCP, the real axis for the penalization parameter (λ) is partitioned into intervals with boundaries:

$$\alpha_0 = 0 < \alpha_1 < \alpha_2 < \alpha_3 < \dots < \alpha_K < +\infty = \alpha_{K+1},$$

where constant K is equal to the size of an unpruned tree, and the above boundaries are computed based on pruning a tree with the entire data — equivalent to running FBP on the entire dataset. In CCP plus CV, a special set of quantities are chosen:

$$\{\alpha_j^* : \alpha_j^* = \sqrt{\alpha_j \alpha_{j+1}}, j = 1, 2, \dots, K - 1\}.$$

For a given λ , let $\text{CVE}(\lambda)$ denote the *cross validation error* that was defined at the beginning of this section. The CV in CCP basically solves the following

$$\min_j \text{CVE}(\alpha_j^*),$$

where $j \in \{1, 2, \dots, K - 1\}$ plus two cases corresponding to the two boundary intervals.

Apparently, the CV in CCP is computed based on a finite set of possible values of parameter λ . This is not the CV principle that was described at the beginning of this section. FBP allows us to examine $\text{CVE}(\lambda)$ for all possible value of λ 's. Based on this, we think that CV in FBP is more ‘faithful.’ Simulations will indicate that such a faithfulness can be linked to better performance in applications. Figure 8 illustrates the difference between the two CV procedures, and when different optimal results will occur.

To be more specific about our CV implementation with FBP. For each leave-out set, we can compute the error rate (as e_{-i} in previous description) as a piecewise constant function *on the entire real line* of λ . In ten-fold CV, ten functions like this will be generated. The (point-wise) average of them is our $\text{CVE}(\lambda)$, which provably is another piecewise constant function. The optimal value of λ is the minimizer of this function.

To illustrate the difference between CV-FBP and CV-CCP, we apply them to the Iris data. Ten-fold CV is considered. Each iteration provides a set of λ 's and corresponding

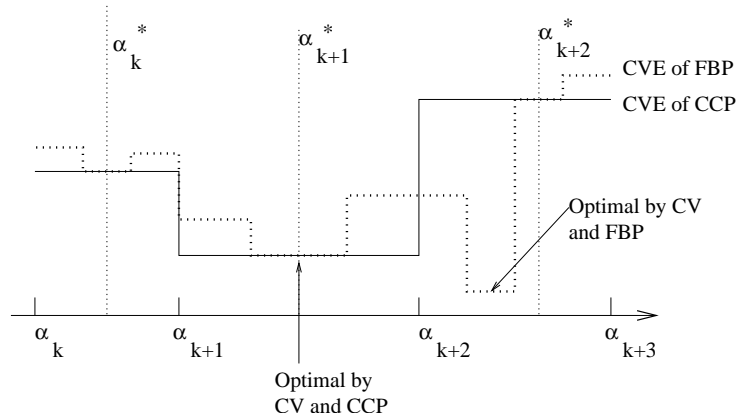


Figure 8: Difference between the two CV procedures (in CCP and FBP). The vertical dotted lines indicate where the CV-CCP tries to find the optimal values; the solid piecewise constant curve reveals the function that was minimized. The dotted curve indicates the function that was minimized by the CV-FBP approach. One can see that CV-CCP only consider an incomprehensive subset.

Table 1: Iris example: λ 's and error rates from 10-fold CV

λ	error rate
0	0.060
0.1667	0.0667
0.3333	0.0600
0.3571	0.0667
0.4000	0.0533
0.4167	0.0467
0.4545	0.0533
1.0000	0.0733
2.0000	0.0867
6.3333	0.1200
36.0000	0.1733
37.0000	0.2133
38.0000	0.2933
39.0000	0.3200
40.0000	0.3933
41.0000	0.4133
43.0000	0.5067
44.0000	0.6267
45.0000	0.6600
46.0000	0.7667

error rates. Note that the error rate is a step function. Ten step functions are then averaged.

Table 1 provides the average error rates for each interval of λ .

Now we explain the difference between CV-CCP and CV-FBP.

- CV-FBP considers all range of λ s (listed in Table 1) and find the optimal one that has the minimum CV error. In our example, the minimum CV error is 0.0467 and the optimal λ is between 0.4167 and 0.4545.
- CV-CCP first uses the entire training data to find the partition of λ -axis. Then the geometric means of pairs of adjacent λ 's are computed. Table 2 shows the range of λ 's and their geometric means. CV-CCP then chooses the optimal λ among α^* 's. In this case, the minimum CV error in CCP is 0.0533 and the optimal λ is 0.4802. One can observe that FBP produces smaller CV error than CCP (0.0467 vs. 0.0533). In general, CV-FBP should always generate a smaller CV error than CV-CCP.

Table 2: Iris example: the range of λ s from the entire training data and geometric mean values.

Tree size	13	7	5	3	2	1
α (range of λ)	0	0.3333	0.5000	1.5000	44.0000	50.0000
α^* (geometric means)	0	0.4802	0.8660	8.1240	46.9042	50.0000

One thing we would like to point out is that in this example, both CV-FBP and CV-CCP lead to the same trained model. Because the optimal λ from both methods are in the same interval $(0.3333, 0.5000]$, which renders an identical tree. This leads to the same testing error rates for both methods. However, in many of our simulations, CV-FBP and CV-CCP lead to different testing errors. This can be seen in Table 5 and Table 6 in the next section.

We have not addressed the problem — does smaller CV error lead to smaller testing error? We leave it for future research.

6. Simulations

This section contains the following contents: (organized in subsections)

- Section 6.1 gives the simulation results that are related to the *inadmissible* tree sizes.
- Section 6.2 compares the cross validation errors (a sanitary check).
- Section 6.3 shows the comparison results between CCP and FBP for testing errors.
- Section 6.4 provides the comparison of tree sizes.

- Section 6.5 illustrates the overall comparison.

CCP algorithm is available in CART software from Salford-Systems and has also been implemented by authors in MATLAB. Following experimental setups are used throughout.

1. Gini index is used as the impurity measure for tree-growing.
2. Twelve data sets available on the UCI database are used. A detailed description on these data sets can be found in the appendix of Li et al. (2001).
3. We perform 10-fold cross validation. Each data set is split into two parts – training set and testing set – for ten times. At each time, the 10-fold CVs are applied to the training set; the cross validation error, testing error, and tree size of the trained model are recorded. The reported values in the following tables are the average of these ten simulations.

6.1 Inadmissibility

An interesting discovery is that in all of these cases, the numbers of admissible tree sizes are significantly smaller than the total number of possible tree sizes. Here the tree sizes are measured by the number of terminal nodes; or in other words, in an RDP, they are the numbers of regions. The simulation result is presented in Table 3. An effective tree size is a tree size that is admissible. The column titled *effective* gives the number of admissible tree sizes. The last column contains the number of terminal nodes in the un-pruned trees. In general, the number of admissible tree sizes is roughly 10% of all the possible sizes of subtrees.

As mentioned in Introduction, we have seen researchers mentioning this phenomenon loosely. We have not seen a quantitative illustration of this fact.

6.2 Cross Validation Errors

Based on the explanation in Section 5.2, between CCP and FBP, one would expect that FBP *always* gives smaller CV errors. The simulations verify this, seeing Table 4. Statistical analysis was implemented as well. Comparing CCP with FBP, the p-value of the paired t-test is roughly 0.001. The 95% confidence interval for the mean difference of the CV error between CCP and FBP is (0.0770, 0.2196). As mentioned earlier, we treat this as a ‘sanitary check’.

Table 3: Comparison of the *effective* tree sizes with the sizes of the largest possible trees.

Name of Dataset	Effective	The Largest Tree Size
Australian Credit Approval	21	481
Cleveland Heart Disease	11	100
Congressional Voting Records	8	44
Wisconsin Breast Cancer	11	45
Iris Plants	6	13
BUPA Liver Disorder	13	132
PIMA Indian Diabetes	20	260
Image Segmentation	30	242
German Credit	26	344
Vehicle Silhouette	30	269
Waveform	20	367
Satellite Image	46	745

Table 4: Comparison of the *CV error rates* between CCP and FBP

Name of Dataset	CCP	FBP	Winner
Australian Credit Approval	14.13	14.01	FBP
Cleveland Heart Disease	21.15	20.89	FBP
Congressional Voting Records	4.16	4.12	FBP
Wisconsin Breast Cancer	4.56	4.47	FBP
Iris Plants	5.20	5.07	FBP
BUPA Liver Disorder	31.27	31.03	FBP
PIMA Indian Diabetes	24.37	23.97	FBP
Image Segmentation	3.84	3.83	FBP
German Credit	24.61	24.48	FBP
Vehicle Silhouette	28.02	27.90	FBP
Waveform	22.86	22.83	FBP
Satellite Image	12.67	12.65	FBP

6.3 Comparison of the Testing Errors

In simulation studies, it is well-received that testing errors are the most important quantities. Table 5 gives the *average* testing errors for 10 simulations between CCP and FBP. (See more explanations at the beginning of this section.) It is shown that FBP tends to give smaller testing errors: 6 cases of ‘smaller than CCP’ and 2 ties out of 12 simulations. However, the difference is not dramatic. The paired t-test reveals that the p-value is 0.152. The 95% confidence interval of the mean difference of the testing errors (between CCP and FBP) is $(-0.0496, 0.2813)$. Note that ‘0’ is inside this interval.

Table 5: Comparison of the *testing error* between CCP and FBP

Name of Dataset	CCP	FBP	Winner
Australian Credit Approval	14.84	14.58	FBP
Cleveland Heart Disease	26.82	27.19	CCP
Congressional Voting Records	5.50	5.60	CCP
Wisconsin Breast Cancer	5.09	4.94	FBP
Iris Plants	7.73	7.33	FBP
BUPA Liver Disorder	35.20	34.74	FBP
PIMA Indian Diabetes	25.34	25.26	FBP
Image Segmentation	5.80	5.81	CCP
German Credit	27.60	27.06	FBP
Vehicle Silhouette	30.25	30.27	CCP
Waveform	27.47	27.47	FBP, CCP
Satellite Image	12.85	12.85	FBP, CCP

6.4 Tree Sizes

In tree models, the size of a tree indicates the complexity of the model. In Table 6, the average tree sizes are given for 12 datasets; each have ten repetitions as explained earlier. We observe that FBP gives the smaller tree sizes than CCP does. Statistical analysis for the difference between CCP and FBP is performed. The p-value for the paired t-test is 0.006. The 95% confidence interval for the mean difference of tree size between CCP and FBP is $(0.617, 2.950)$. Based on this, one may tend to agree that a combination of CV and FBP generates smaller trees. We would like to emphasize that this is just an empirical result. More theoretical understanding is needed.

Table 6: Comparison of the *tree sizes* (*number of all nodes*) between CCP and FBP

Name of Dataset	CCP	FBP	Winner
Australian Credit Approval	10.60	8.00	FBP
Cleveland Heart Disease	8.20	6.80	FBP
Congressional Voting Records	7.00	5.80	FBP
Wisconsin Breast Cancer	14.20	10.60	FBP
Iris Plants	7.20	6.60	FBP
BUPA Liver Disorder	16.60	13.80	FBP
PIMA Indian Diabetes	15.20	9.00	FBP
Image Segmentation	89.40	87.20	FBP
German Credit	29.20	29.20	FBP, CCP
Vehicle Silhouette	63.40	62.60	FBP
Waveform	69.00	69.00	FBP, CCP
Satellite Image	249.00	249.0	FBP, CCP

6.5 Overall Comparison

The Figure 9 provides a graphical way to compare three methods (C4.5, CCP and FBP), based on the testing errors and tree sizes. Here we include C4.5, which is another major tree building method. Note that it is not appropriate to compare C4.5 directly with CCP and FBP. Because C4.5 uses different tree-growing algorithm (e.g., ID3 algorithm) and generates multi-way splits; it allows the nodes to have more than two child nodes, while CCP and FBP are binary trees. However it is still interesting to see the difference of their performance with some universal measurements: ‘Smaller testing errors’ and ‘smaller tree sizes’, which are ideal for a classifier. Based on this, being lower and left in Figure 9 is desired. Comparing the tree methods, we can see that FBP is relatively better than the other two methods. Only the first seven datasets are plotted in the figure. Including of more cases will make the figure look confusing. The same trends are observed for all datasets though.

7. Reproducible Research and Software

The principle of *Reproducible Research* was advocated by scientists such as Professor J. Claerbout (Claerbout) and Professor D. L. Donoho (WaveLab). Their main idea is: “An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.”

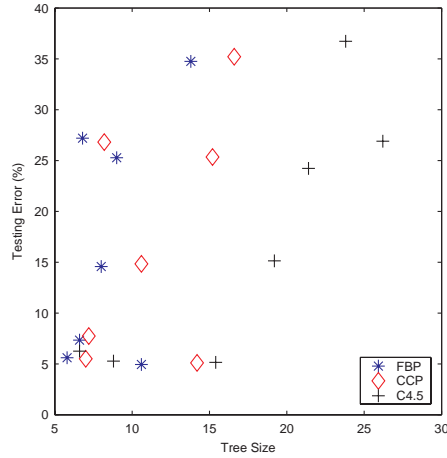


Figure 9: Testing errors versus tree sizes: being lower-left in this figure is optimal.

Following this principle, we are going to release the software that have been used in our simulation studies. At the current stage, the software is available by email from the first author. In the future, we may create a web page for the release.

The algorithms are implemented in MATLAB. Cares are given to ensure the efficiency of each implemented algorithm. There are seven basic components:

1. *Tree Growing.* Grow a big and potentially redundant tree from data.
2. *Tree Pruning.* Use FBP to generate lists of linear functions at each node.
3. *Find Lower Bound.* Based on the list of linear functions at the root node, find the lower bound function $f(\cdot)$.
4. *Identify the Best Subtree.* Given a value of the parameter λ , identify the size of the best subtree, together with the subtree itself.
5. *Testing.* Apply a tree model that is generated above to a testing data, and report the testing result.
6. *Application in Cross Validation.* Use the frontier-based tree-pruning algorithm to realize cross validation.

Figure 10 provides an overview of the software architecture.

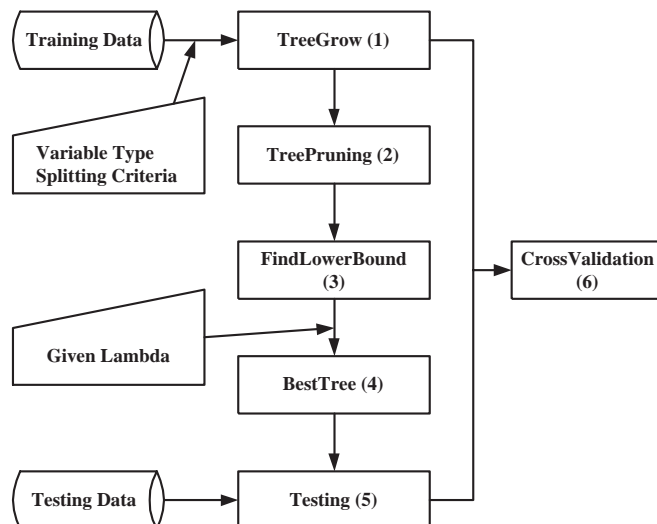


Figure 10: Relation of the functions. The numbers in the parentheses indicate the steps.

8. Conclusion

A frontier-based tree-pruning algorithm (which is named FBP) was proposed. This algorithm provides a graphical way to implement the task of minimizing CPLF. FBP has the same objective with CCP; however it is more advantageous, because it provides a full spectrum of information in tree pruning. It can be used to ‘more faithfully’ realize the principle of cross validation. A combination of FBP and cross validation render ‘better’ classifiers in simulations, comparing with other existing methods. Simulation results on real data sets also render several other interesting findings, for example, the number of admissible tree sizes is always a small proportion (roughly 10%) of the number of all possible tree sizes. Computational complexity analysis is provided. This method is appealing in implementations and has potentials to be used in other tree related study, e.g. the stability of applying cross validation in building tree models.

Acknowledgement

The authors would like to thank the editor, associate editor, and two anonymous referees, whose comments help greatly in improving the presentation of this paper.

The presented material is based upon work partially supported by National Science Foundation under grants No. 0140698, No. 0140587, and No. 0346307.

Appendix 1. Datasets

The 12 datasets are from the UCI repository. Table 7 provides a summary of these datasets.

Table 7: Summary of Datasets.

Dataset	Classes	Attributes	Sample Size Training (Testing)
Wisconsin Breast Cancer	2	9 numeric	683
Cleveland Heart Disease	2	7 numeric, 6 categorical	270
PIMA Indian diabetes	2	7 numeric	768
BUPA Liver Disorder	2	6 numeric	345
Congressional Voting Records	2	16 categorical	435
Australian Credit Approval	2	6 numeric, 8 categorical	690
German Credit	2	7 numeric, 13 categorical	1000
Iris Plants	3	4 numeric	150
Satellite Image	6	36 numeric	4435 (2000)
Image Segmentation	7	19 numeric	2310
Vehicle Silhouette	4	18 numeric	846
Waveform	3	21 numeric	600 (3000)

References

- Buja, A. and Y.-S. Lee. 2001. Data mining criteria for tree-based regression and classification. *Proc. Knowledge Discovery and Data Mining-2001* 27-36.
- Breiman, L., J. Friedman, R. Olshen, and C. Stone. 1984. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA.
- Coifman, R. R. and M. V. Wickerhauser. 1992. Entropy-based algorithms for best basis selection. *IEEE Trans. Inform. Theory* **38** 713-718.
- Donoho, D. L. 1997. CART and best-ortho-basis: a connection. *Ann. Statistics* **25** 1870-1911.
- Donoho, D. L. 1999. Wedgelets: nearly minimax estimation of edges. *Ann. Statistics*, 27 (3): 859-897.
- Li, X.-B., J. Sweigart, J. Teng, J. Donohue, and L. Thombs. 2001. A dynamic programming based pruning method for decision trees. *INFORMS J. on Computing* **13** 332-344.

Luenberger D. G., 1998. *Investment Theory*. Oxford University Press. New York.

Markowitz, H. 1952. Portfolio Selection. *J. Finance* **7** 77-91.

CART&MARS. <http://www.salford-systems.com/>.

C4.5. <http://www2.cs.uregina.ca/~hamilton/courses/831/notes/ml/dtrees/c4.5/tutorial.html>.

Claerbout, J. <http://sep.stanford.edu/>.

WaveLab. <http://www-stat.stanford.edu/~wavelab>.