Lectures on Machine Learning (Fall 2017)

Hyeong In Choi Seoul National University

Lecture 9: Classification and Regression Tree (CART) (Draft: version 0.8.1)

Topics to be covered:

- Basic ideas of CART
- Classification tree
- Regression tree
- Impurity: entropy & Gini
- Node splitting
- Pruning
- Tree model selection (validation)

1 Introduction: basic ideas of CART

The Classification And Regression Tree (in short, CART) is a very popular tree-based method that is in wide use. Its origin is in the so-called decision tree, but the version we present here adheres to a particular way of constructing trees. For example, CART always creates a binary tree, meaning each non-terminal node has two child nodes. This is in contrast with

general tree-based methods that may allow multiple child nodes. One big appeal the tree-based method, and in particular CART, has is that the decision process is very much akin to how we humans make decisions. Therefore it is easy to understand and accept the results coming from the tree-style decision process. This intuitive explanatory power is one big reason why the tree method will never go away. Another very appealing aspect of CART is that, unlike many linear combination methods like logistic regression or support vector machine, it allows diverse types of input data. So the input data can mix numerical variables like price or area and categorical variables like house type or location. This flexibility makes CART a favored tool of choice in a variety of applications. In this lecture we cover both the CART classification tree and the CART regression tree. The book by Breiman et al. [1] remains the best reference for CART.

1.1 Classification Tree

Let us first look at the classification problem. Figure 1 shows points in the plane. These points are marked with either a cross or circle. The goal of a classifier is to partition the plane into smaller regions and assign a classification mark–a cross or circle–to each region. It has to do it in such way that the classification error for other similar data would be reasonably small. Let us illustrate how CART goes about constructing such a classifier.

To begin, the whole is plane marked as t_1 and it corresponds to the single node in the tree consisting of one node as in Figure 2.

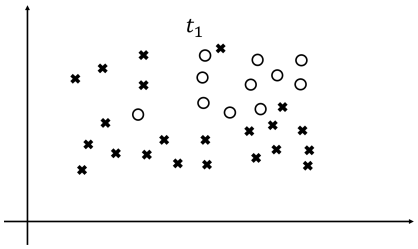

Figure 1: The data on $\mathfrak{X}$

Now take a vertical line as in Figure 3. This line splits the plane into two regions: the left region and the right region. The left region is marked as t_2 and the right as t_3. The splitting of t_1 into two subregions t_2 and t_3 is

Figure 2: The classification tree with one node t_1

encoded in the tree-style data structure as depicted in Figure 4, where the root node t_1 has two child nodes t_2 and t_3.

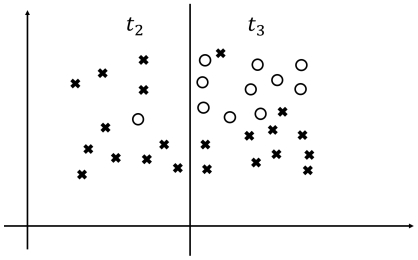

Figure 3: Splitting t_1

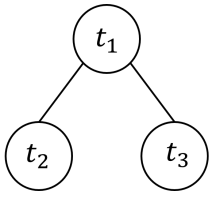

Figure 4: The classification tree after the split of t_1

Now the region t_3 is further split into two subregions t_4 and t_5. This situation is encoded in the tree in Figure 6.

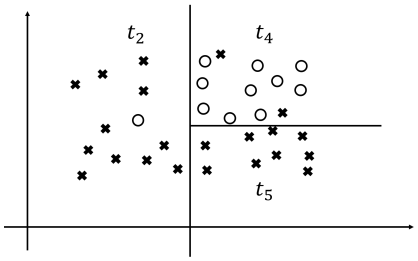

Figure 5: Splitting t_3

Suppose we stop splitting here. Then the three regions corresponding to the terminal nodes t_2, t_4 and t_5 constitute the partition of the whole plane. Our job now is to assign the class label to each of the three partitioned regions. Now upon inspection, the region t_2 has more crosses than circles. So by the *majority vote* the whole region of t_2 is assigned the class label 'cross'. Similarly the region t_5 is assigned the class label 'cross'. On the other hand, the region t_4 has more circles than crosses. Thus it is assigned the class label 'circle'. This way every point in the plane is assigned a class label.

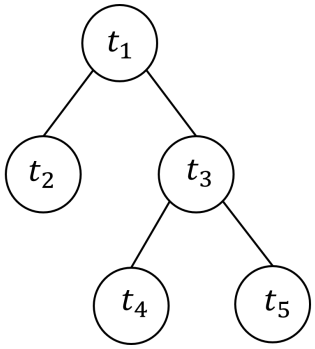

Figure 6: The classification tree after the split of t_3

So in the future, given any point p, which is not necessarily one of the data points we have been dealing with, its response (class label) has to be the one gotten by the majority vote. Say, if p is in region t_4, it has to be 'circle', which is the class label of the region t_4; similarly, if p is in t_2, its response has to be 'cross'. This way, our tree is regarded as a classifier.

1.2 Regression tree

Now let us look at the regression tree. Figure 7 shows points in the xy-plane. The regression problem is to find a "good" function $y = f(x)$ whose graph lies close to the given data points in Figure 7. The method of the CART regression tree is similar to that of the CART classification tree in that the whole region–the x axis–is partitioned into subregions and the partitioning pattern is also encoded in a tree data structure, called the regression tree. The only difference here is that instead of taking the majority vote as was done for the classification tree, the average y-value of the data is used as the (constant) value of the regression function on each partitioning subregion.

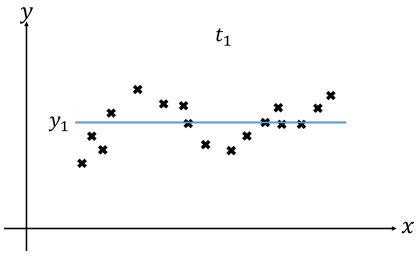

Figure 7: Given data with $\mathfrak{X}$ on the x axis and $\mathfrak{Y}$ on the y axis

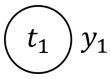

Figure 8: The regression tree with one node t_1

To start, Figure 7 has only one region: the entire x-axis. On this region, the regression function must be a constant function. So the most reasonable representative has to be the average. This average y-value y_1 is recorded in the one-node tree in Figure 8.

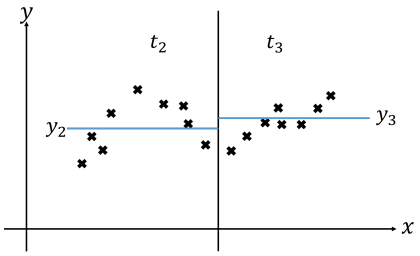

Figure 9: Splitting t_1

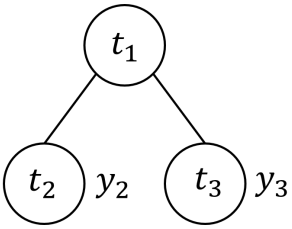

Figure 10: The regression tree after the split of t_1

Now the region t_1 is split into two subregions t_2 and t_3. On each subregion the average y-value of the data is y_2 and y_3, respectively, which is recorded in Figure 10. The subregions t_2 and t_3 are further split and the average y-value on each smaller subregion is computed and recorded in Figure 12.

Suppose now we stop splitting here. Then the region is partitioned into four subregions: t_4, t_5, t_6 and t_7. On each subregion the average t-value, y_4, y_5, y_6 and y_7 is computed, respectively. This way, the final regression function $y = f(x)$ is a piecewise constant function with values y_i on t_i for $i = 4, 5, 6$ and 7.

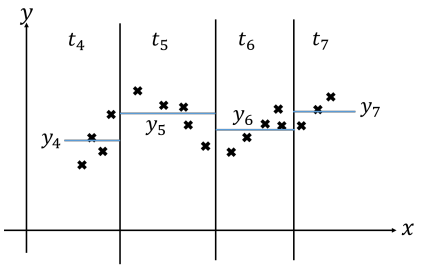

Figure 11: splitting of t_2 and t_3

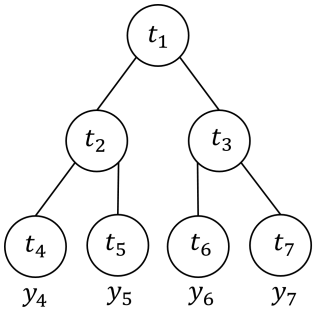

Figure 12: The regression tree after the split of t_2 and t_3

We have seen two simple examples of how the CART classification tree and regression tree are created and how the classifier and the regression function are determined. In either case, the tree is nothing but a bookkeeping device to keep track of how the region is partitioned and how to organize the final classifier or regression function. But many questions still remain unanswered. Which variable (co-ordinate axis) does one choose and where does one put the splitting boundary? When does one stop splitting? Among

many possible candidates for the CART trees, which is the best? What is the training error and what is the reasonable expectation of the generalization error? These are all immediate questions to be answered and there are many other issues to be clarified, which are the focus of the rest of this lecture.

2 Details of classification tree

Let us set up the basic notations we will be using in our subsequent discussion on classification tree. The data set is $\mathfrak{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $x^{(i)} \in \mathfrak{X}$ and $y^{(i)} \in \mathfrak{Y}$, where $\mathfrak{Y} = \{1, \cdots, K\}$ is the set of K class labels. We define the following *empirical* quantities:

$N = |\mathfrak{D}|$: the total number of data points

$N_j = |\{(x^{(i)}, y^{(i)}) \in \mathfrak{D} : y^{(i)} = j\}|$: the number of data points with class label $j \in \mathfrak{Y}$

Let t be a node, which is also identified with a subregion of $\mathfrak{X}$ it represents. Then at t, we define

$\mathfrak{D}(t) = \{(x^{(i)}, y^{(i)}) \in \mathfrak{D} : (x^{(i)}, y^{(i)}) \in t\}$: the set of data points in the node t

$N(t) = |\mathfrak{D}(t)|$: the number of data points in the node t

$N_j(t) = |\{(x^{(i)}, y^{(i)}) \in \mathfrak{D}(t) : y^{(i)} = j\}|$: the number of data points in the node t with class label $j \in \mathfrak{Y}$

If one randomly chooses a data point from $\mathfrak{D}$, the probability of picking a data point with class label j should normally be N_j/N, if every point has equal chance. In some case, however, this assumption of "equal chance" may not be desirable. So in general we let $\pi = (\pi(1), \cdots, \pi(K))$ be the prior probability distribution of $\mathfrak{Y}$ that represents the probability of picking the class label j. Now define

$$p(j, t) = \pi(j) N_j(t)/N_j.$$

This is the probability of picking up a data point that is in t with the class label j. (Note that if $\pi(j) = N_j/N$, $p(j, t) = N_j(t)/N$.) Define

$$p(t) = \sum_j p(j, t), \tag{1}$$

which is the probability of picking a data point in the node t. We also define

$$p(j|t) = p(j,t)/p(t), \tag{2}$$

which is the conditional probability of picking a data point with the class label j when the node t is given. Note that if $\pi(j) = N_j/N$, then $p(t) = \sum_j N_j(t)/N = N(t)/N$ and hence $p(j|t) = N_j(t)/N(t)$. As we said above, CART has a particular way of creating a tree. First, it always splits along a *single* feature (variable) of the input. Recall that the input vector x is of the form $x = (x_1, \cdots, x_d)$ where each x_i can be a numeric, categorical (nominal), or discrete ordinal variable. (We also call it a co-ordinate or co-ordinate variable in this lecture.) At each node, CART always chooses one of those variables and splits the node based on the values of that variable only. In other words, CART does *not combine* variables like logistic regression does. Some tried to extend CART by combining variables, but the benefit weighed against the added complexity does not seem to justify the effort. Moreover, CART is not our ultimate tree-based method. In view of *random forests* we will be covering in our next lecture, we prefer the simpler and faster approach of CART. Another particular way CART does splitting is that it always creates two child nodes–hence the binary splitting–which results in a binary tree. Of course, general tree-based methods can split nodes into multiple (more than two) child nodes. But for CART, the split is always binary. The reason is again simplicity as a single multiple split can be equivalently made by several successive binary splits .

2.1 Impurity and splitting

In order to do the splitting, we need the following definitions.

Definition 1. *An* **impurity function** ϕ *is a function* $\phi(p_1, \cdots, p_K)$ *defined for* $p_1, \cdots, p_K$ *with* $p_i \geq 0$ *for all* i *and* $p_1 + \cdots + p_K = 1$ *such that*

(i) $\phi(p_1, \cdots, p_K) \geq 0$

(ii) $\phi(1/K, \cdots, 1/K)$ *is the maximum value of* ϕ

(iii) $\phi(p_1, \cdots, p_K)$ *is symmetric with regard to* $p_1, \cdots, p_K$

(iv) $\phi(1, 0, \cdots, 0) = \phi(0, 1, \cdots, 0) = \phi(0, \cdots, 0, 1) = 0$.

The following are examples of impurity functions:

Example 1. *(i) Entropy impurity*

$$\phi(p_1, \cdots, p_K) = -\sum_i p_i \log p_i,$$

where we use the convention $0 \log 0 = 0$.

(ii) Gini impurity

$$\phi(p_1, \cdots, p_K) = \frac{1}{2}\sum_i p_i(1-p_i) = \sum_{i<j} p_i p_j$$

Definition 2. *For node t, its impurity (measure) $i(t)$ is defined as*

$$i(t) = \phi(\, p(1|t), \cdots, p(K|t)\,).$$

Splitting criterion

Given a node t, there are many possible ways of doing binary split. A general principle is that it is better to choose the split that decreases the impurity the most. Suppose t is split into t_L and t_R as in Figure 13. We define

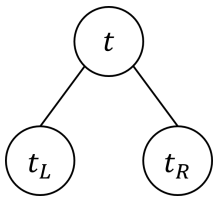

Figure 13: t_L and t_R

$p_L = p(t_L)/p(t)$ and $p_R = p(t_R)/p(t)$ so that $p_L + p_R = 1$. Let us now see an example on how impurity function is used to facilitate the "best" split. The left of Figure 14 denotes a node with four elements–two labeled with crosses and two with circles. The middle and the right figures represent two possible ways of splitting it.

The middle figure shows two child nodes t_1 and t_2. The entropy impurity of each child node is computed as:

$$\begin{aligned} t_1 \;:\; & p(\circ|t_1) = 0, \quad p(\times|t_1) = 1 \\ & i(t_1) = -\{0 \cdot \log 0 + 1 \cdot \log 1\} = 0 \\ t_2 \;:\; & p(\circ|t_2) = 1 \quad p(\times|t_2) = 0 \\ & i(t_2) = -\{1 \cdot \log 1 + 0 \cdot \log 0\} = 0. \end{aligned}$$

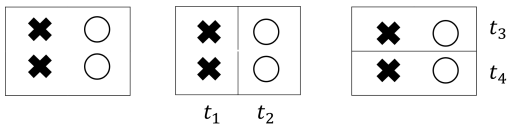

Figure 14: Two possible ways of splitting

The right figure shows two different child nodes t_3 and t_4. The entropy impurity is again computed as:

$$\begin{aligned} t_3 \;:\; & p(\circ|t_3) = 1/2, \quad p(\times|t_3) = 1/2 \\ & i(t_3) = -\Big\{\frac{1}{2}\cdot\log\frac{1}{2} + \frac{1}{2}\cdot\log\frac{1}{2}\Big\} = \log 2 \\ t_3 \;:\; & p(\circ|t_4) = 1/2 \quad p(\times|t_4) = 1/2 \\ & i(t_3) = -\Big\{\frac{1}{2}\cdot\log\frac{1}{2} + \frac{1}{2}\cdot\log\frac{1}{2}\Big\} = \log 2. \end{aligned}$$

Therefore the split done as in the middle figure is better in the sense that the child nodes have lower impurity. One can check the same thing with the Gini impurity. In order to formalize this process, we need the following definitions.

Definition 3. *Let t be a node and let $\mathfrak{s}$ be split of t into two child nodes t_L and t_R. Define the* decrease in impurity *as*

$$\Delta i(\mathfrak{s}, t) = i(t) - p_L i(t_L) - p_R i(t_R).$$

Definition 4. *Let T be a tree. We denote by $\tilde{T}$ the set of the terminal (leaf) nodes of T.*

Definition 5. *Define the impurity of the node t by*

$$I(t) = i(t)p(t),$$

and the impurity of the tree T by

$$I(T) = \sum_{t\in\tilde{T}} I(t) = \sum_{t\in\tilde{T}} i(t)p(t).$$

Let us now see how split affects the tree impurity. The left in Figure 15 is a tree T in which the terminal nodes are the ones marked as t or t'. The

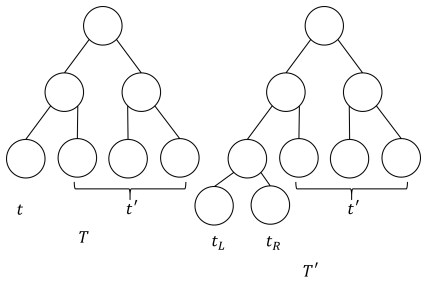

Figure 15: T and T'

right shows a new tree T' which is gotten by splitting the terminal node t into two child nodes t_L and t_R, while the rest of the terminal nodes remain the same.

Then it is easy to see that

$$\begin{aligned} I(T) &= I(t) + \sum_{t' \in \tilde{T} - \{t\}} I(t') \\ I(T') &= I(t_L) + I(t_R) + \sum_{t' \in \tilde{T} - \{t\}} I(t'). \end{aligned}$$

Therefore the change in the total impurity from T to T' is

$$I(T) - I(T') = I(t) - I(t_L) - I(t_R),$$

which inspires the following defition.

Definition 6. *Let t be a node and let $\mathfrak{s}$ be split of t into two child nodes t_L and t_R. Then the total impurity change due to $\mathfrak{s}$ is defined to be*

$$\Delta I(\mathfrak{s}, t) = I(t) - I(t_L) - I(t_R).$$

Note that

$$\begin{aligned} \Delta I(\mathfrak{s}, t) &= i(t)p(t) - i(t_L)p(t_L) - i(t_R)p(t_R) \\ &= \{i(t) - p_L i(t_L) - p_R i(t_R)\}p(t) \\ &= \Delta i(\mathfrak{s}, t)p(t) \end{aligned}$$

Therefore the split $\mathfrak{s}$ that maximizes $\Delta I(\mathfrak{s}, t)$ also maximizes $\Delta i(\mathfrak{s}, t)$. So we can use either node impurity measure to the same effect.

2.2 Determining node class label

Once a tree is fixed, a classifier can be defined from that tree. First, recall that every node of a tree corresponds to a subregion of $\mathfrak{X}$, and thus the totality of the terminal nodes constitutes a partition of $\mathfrak{X}$ into mutually disjoint subregions of $\mathfrak{X}$. The basic principle of classification using the tree is that the class label of such individual subregion should be determined by *majority vote*. But the way we apply the majority vote is slightly more general. Namely, instead of treating all misclassification cases with equal weights, we give different weight to each individual misclassification case by introducing the following cost function.

Definition 7. *Let $i, j \in \mathfrak{Y}$ be labels. A cost $C(i|j)$ of misclassifying class j as class i is defined as a nonnegative number satisfying $C(j|j) = 0$ for all $j \in \mathfrak{Y}$.*

The simplest example of cost is $C(i|j) = 1-\delta_{ij}$, which simply says that the misclassification costs are the same regardless of the labels involved. Assume our classifier declares the label of a node t to be i, while its "true" label is unknown. Then the expected cost of declaring the (response) label as i has to be the weighted (in accordance with the probability $p(j|t)$) sum of cost, i.e.,

$$\sum_j C(i|j)p(j|t). \tag{3}$$

In particular, in case $C(i|j) = 1 - \delta_{ij}$, the expected cost of misclassification is easily seen to be

$$\sum_j (1 - \delta_{ij})p(j|t) = 1 - p(i|t). \tag{4}$$

Definition 8. *Let t be a node. The class label $j^*(t)$ of t is defined to be*

$$j^*(t) = \operatorname*{argmin}_i \sum_j C(i|j)p(j|t). \tag{5}$$

In case there is a tie, tie breaking is done by some arbitrary rule.

In case $C(i|j) = 1 - \delta_{ij}$, it is easily seen by (4) that

$$\operatorname*{argmin}_i \sum_j C(i|j)p(i|t) = \operatorname*{argmax}_i p(i|t) \tag{6}$$

Assume further that $\pi(i) = N_i/N$. Then we have $p(i|t) = N_i(t)/N(t)$. Thus $j^*(t)$ is reduced to

$$j^*(t) = \underset{i}{\text{argmax}} N_i(t).$$

Namely in this case, the class label $j^*(t)$ of t is determined by the simple majority vote rule.

2.3 Misclassification cost

A goal of the classification tree is to minimize misclassification. A simplistic way of looking at this is to count the number of misclassified cases and calculate the ratio. But a more general way is to utilize the misclassification cost defined above. For that, we need the following definitions.

Definition 9. *Define the misclassification cost $r(t)$ of node t by*

$$r(t) = \min_i \sum_j C(i|j)p(j|t). \tag{7}$$

Let $R(t) = r(t)p(t)$. Using this, define the misclassification cost $R(T)$ of tree T by

$$R(T) = \sum_{t \in \tilde{T}} R(t) = \sum_{t \in \tilde{T}} r(t)p(t).$$

The following proposition shows that *any* split decreases the misclassification cost.

Proposition 1. *Let a node t is split into t_L and t_R. Then*

$$R(t) \geq R(t_L) + R(t_R)$$

and the equality holds if $j^*(t) = j^*(t_L) = j^*(t_R)$.

Proof. By (5),

$$r(t) = \sum_j C(j^*(t)|j)p(j|t). \tag{8}$$

Thus

$$\begin{aligned} R(t) &= r(t)p(t) = \sum_j C(j^*(t)|j)p(j|t)p(t) \\ &= \sum_j C(j^*(t)|j)p(j,t) \\ &= \sum_j C(j^*(t)|j)\{p(j,t_L) + p(j,t_R)\}, \end{aligned} \tag{9}$$

where the last equality is due to the fact that $N_j(t) = N_j(t_L) + N_j(t_R)$. Note that for any node t, by (2) and (7), we have

$$R(t) = r(t)p(t) = \min_i \sum_j C(i|j)p(j,t).$$

Therefore, using this and (9), we have

$$\begin{aligned} R(t) - R(t_L) - R(t_R) = & \sum_j C(j^*(t)|j)p(j,t_L) - \min_i \sum_j C(i|j)p(j,t_L) \\ + & \sum_j C(j^*(t)|j)p(j,t_R) - \min_i \sum_j C(i|j)p(j,t_R) \geq 0. \end{aligned}$$

It is trivial to see that if $j^*(t) = j^*(t_L) = j^*(t_R)$, then the equality holds. □

2.4 Weakest-link cutting & minimal cost-complexity pruning

Proposition 1 above says that it is always better (not worse off) to keep splitting all the way down. But then we may eventually get down to the pure terminal nodes that contain only one class. Such terminal node may even contain only one point. But this must be an extreme case of overfitting. So the question arises as to when and how to stop splitting and on how to prune back, if we have split too much. One quick rule of thumb is that one should stop splitting the node if the number of elements of the node is less than the preset number. But still, the question remains:"What is the right-sized tree and how do we find such a tree that has a *good* generalization behavior?" This is the key issue we are addressing here. One way of preventing such problem is to penalize the tree if it has too many terminal nodes. It is akin to regularization we discussed in earlier lectures. For that, we need the following definition.

Definition 10. *Let $\alpha \geq 0$. For any node t, not necessarily a terminal node, define*

$$R_\alpha(t) = R(t) + \alpha.$$

and for a tree T, define

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|,$$

where $|\tilde{T}|$ is the number of terminal elements.

Now if we use $R_\alpha(t)$ as a new measure for (augmented) misclassification cost, a split does not not necessarily decrease $R_\alpha(t)$, because it increases $\alpha|\tilde{T}|$. Thus this term acts like a regularization term and hopefully we may come to a happy compromise.

Let us set up more notations. Let T be a given tree as in Figure 16. Let t_2 be a node of T and let T_{t_2} be a new tree having t_2 as the root node together with all of its descendants in T. T_{t_2} is a subtree of T and is called a branch of

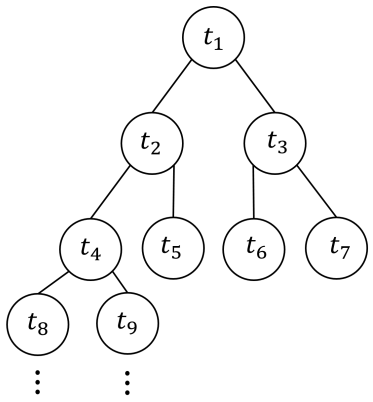

Figure 16: Given tree T

T at t_2. This T_{t_2} is depicted in Figure 17. Suppose we cut off T_{t_2} from T. It

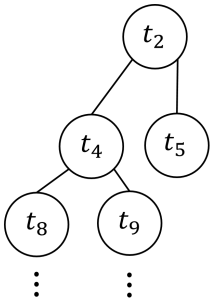

Figure 17: Subtree T_{t_2}

means that we remove from T all the nodes of T_{t_2} except t_2. The remaining tree, denoted by $T - T_{t_2}$, is depicted in Figure 18. Note that the node t_2 still remains intact in $T - T_{t_2}$. If we were to remove t_2 from $T - T_{t_2}$, the subregion represented by t_2 or its descendants disappear from the partition of $\mathfrak{X}$ so that

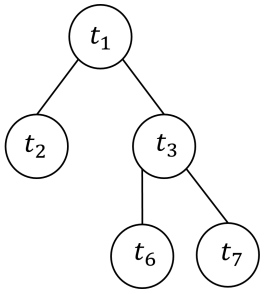

Figure 18: Pruned subtree $T - T_{t_2}$

the terminal nodes of the remaining tree would only be t_6 and t_7. But then the regions corresponding to t_6 and t_7 would not form the full set of $\mathfrak{X}$. If, on the other hand, t_2 is still intact in $T - T_{t_2}$ as in Figure 18, the totality of the subregions corresponding to the terminal nodes of $T - T_{t_2}$ would still form a partition of $\mathfrak{X}$. That is the reason why we keep t_2 in $T - T_{t_2}$. To formalize this process, let us give the following definition.

Definition 11. *A pruned subtree T_1 of a tree T is a subtree obtained by the following process:*

(1) Take a node t of T

(2) Remove all descendants of t from T, but not t itself.

If T_1 is a pruned subtree of T, we write $T_1 \preceq T$. Note in particular that T_1 always contains the root node of T. One should note the subtlety of definition: pruned subtree also has the same root note of the original tree, while subtree can be any part of the original tree so that it may not necessarily contain the root node of the original tree.

Suppose now we have done all the splitting and have grown a big tree which is denoted by $T_{\max}$. We want to find a subtree of $T_{\max}$ that minimizes R_α. But there may be more than one such subtree. Thus to clear up the confusion, we need the following definition.

Definition 12. *Given a tree T and $\alpha \geq 0$, the optimally pruned subtree $T(\alpha)$ of T is defined by the following conditions:*

(i) $T(\alpha) \leq T$

(ii) $R_\alpha(T(\alpha)) \le R_\alpha(T')$ *for any pruned subtree* $T' \preceq T$.

(iii) If a pruned subtree $T' \preceq T$ *satisfies the condition* $R_\alpha(T(\alpha)) = R_\alpha(T')$, *then* $T(\alpha) \preceq T'$

Proposition 2. $T(\alpha)$ *exists and is unique.*

Proof. Let T be a given tree with its root node t_1. We use the induction on $n = |\tilde{T}|$. If $n = 1$, then $T = \{t_1\}$, in which case there is nothing to prove. So we assume $n > 1$. Then there must be child nodes t_L and t_R of t_1. Let T_L be the subtree of T having t_L as its root node together with all of its descendants. Similarly define T_R. Thus $T_L = T_{t_L}$ and $T_R = T_{t_R}$. These T_L and T_R, called the *primary branches* of T, are *not* pruned subtrees of T, because they do not contain the root node of the original tree.

Let T' be a nontrivial (having more than one node) pruned subtree of T. Thus T' must contain the nodes t_1, t_L and t_R of T. Let $T'_L = T'_{t_L}$ and $T'_R = T'_{t_R}$ be the primary branches of T'. Then clearly T'_L is a pruned subtree of T_L having the same root node t_L. Similarly, T'_R is a pruned subtree of T_R having the same root node t_R. (Note T'_L and T'_R are *not* pruned subtree of T.) By the induction hypothesis, there must be optimally pruned subtrees $T_L(\alpha)$ and $T_R(\alpha)$ of T_L and T_R, respectively. There are two cases to consider.

The first to consider is the case in which

$$R_\alpha(t_1) \le R_\alpha(T_L(\alpha)) + R_\alpha(T_R(\alpha)). \tag{10}$$

Then by the definition of $T_L(\alpha)$, we must have $R_\alpha(T_L(\alpha)) \le R_\alpha(T'_L)$, and similarly, $R_\alpha(T_R(\alpha)) \le R_\alpha(T'_R)$. Since $R_\alpha(T') = R_\alpha(T'_L) + R_\alpha(T'_R)$, combining this with (10), we then have

$$R_\alpha(t_1) \le R_\alpha(T').$$

Since this is true for any pruned subtree T' of T, we can conclude that $T(\alpha) = \{t_1\}$.

The remaining case is the one in which

$$R_\alpha(t_1) > R_\alpha(T_L(\alpha)) + R_\alpha(T_R(\alpha)). \tag{11}$$

Now create a new tree T_1 by joining $T_L(\alpha)$ and $T_R(\alpha)$ to t_1. Then clearly T_1 is a pruned subtree of T and $T_L(\alpha)$ and $T_R(\alpha)$ are the primary branches of T_1. Clearly

$$R_\alpha(T_1) = R_\alpha(T_L(\alpha)) + R_\alpha(T_R(\alpha)). \tag{12}$$

Now by (11), $T(\alpha)$, if it ever exists, cannot be $\{t_1\}$. So now let T' be any nontrivial pruned subtree of T. Then by the definition of $T_L(\alpha)$, we have $R_\alpha(T_L(\alpha)) \leq R_\alpha(T'_L)$, and similarly, $R_\alpha(T_R(\alpha)) \leq R_\alpha(T'_R)$. Since $R_\alpha(T') = R_\alpha(T'_L) + R_\alpha(T'_R)$, combining this with (12), we then must have

$$R_\alpha(T_1) \leq R_\alpha(T').$$

Therefore T_1 is the pruned subtree of T that has the smallest R_α-value among all pruned subtrees of T.

It remains to show that any strictly smaller pruned subtree of T_1 has larger R_α-value. So let T' be a strictly smaller pruned subtree of T_1. We may assume without loss of generality $T'_L < T_L(\alpha)$ and $T'_R \leq T_R(\alpha)$. Then again, by the definition of $T_L(\alpha)$, we have $R_\alpha(T_L(\alpha)) < R_\alpha(T'_L)$, and similarly $R_\alpha(T_R(\alpha)) \leq R_\alpha(T'_R)$. Then using the same argument as above, we have

$$R_\alpha(T_1) < R_\alpha(T').$$

Therefore we can conclude that

$$T_1 = T(\alpha).$$

□

Let us now embark on the pruning process. The first thing to do is to come up with the smallest subtree $T(0)$ with the cost $R_0 = R$. Let us see how we do it. Start with $T_{\max}$. Suppose t_L and t_R are terminal nodes with the same parent t. By Proposition 1, $R(t) \geq R(t_L) + R(t_R)$. If, furthermore, $R(t) = R(t_L) + R(t_R)$, then prune at t. And repeat this process to get a smallest $T_1 \leq T_{\max}$ with $R(T_1) = R(T_{\max})$. Note that T_1 is easily seen to be equal to $T(0)$. Also ,the following proposition easily follows from the way T_1 is constructed.

Proposition 3. *For any non-terminal node $t \in T_1 = T(0)$,*

$$R(T_t) < R(t).$$

The next step is to find $T(\alpha)$. But the trouble is that there are so many possibilities for α. Let us see how to come up with such "good" α.

Proposition 3, by the continuity argument, easily implies that for α is sufficiently small,

$$R_\alpha(T_t) < R_\alpha(t), \tag{13}$$

at any node $t \in T(0)$. In that case, we'd be better off not pruning. Note that

$$\begin{aligned} R_\alpha(t) &= R(t) + \alpha \\ R_\alpha(T_t) &= R(T_t) + \alpha|\tilde{T}_t|. \end{aligned}$$

Therefore (13) is equivalent to saying that

$$\alpha < \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}. \tag{14}$$

But as α gets larger, there appears a node (or nodes) at which (14) is no longer valid. To find such node(s), let us define a function $g_1(t)$ of node t by

$$g_1(t) = \begin{cases} \dfrac{R(t) - R(T_t)}{|\tilde{T}_t| - 1} & \text{if } t \notin \tilde{T}_1 \\ \infty \quad \text{else.} \end{cases}$$

Define

$$\alpha_2 = \min_{t \in T_1} g_1(t).$$

So for $\alpha < \alpha_2$, there is no node at which (14) is violated. But for $\alpha = \alpha_2$, there is at least one node at which the inequality (14) becomes the equality. Such nodes are called *weakest-link nodes*. Suppose t_1' is such node. Then

$$g_1(t_1') = \alpha_2 = \min_{t \in T_1} g_1(t).$$

We then prune T_1 at t_1' to come up with a pruned subtree $T_1 - T_{t'}$. Note that $R_{\alpha_2}(T_{t'}) = R_{\alpha_2}(t')$. Therefore

$$R_{\alpha_2}(T_1) = R_{\alpha_2}(T_1 - T_{t'}).$$

There may be more than one such weakest-link nodes. Suppose there are two such nodes. If one of them is an ancestor of the other, prune at the ancestor (then its descendants are all gone). If neither one is an ancestor of the other, prune at both. Keep doing this for all such weakest-link nodes. This way, we get a new subtree T_2 and it is not hard to see that

$$T_2 = T(\alpha_2).$$

Using T_2 in lieu of T_1, we can repeat the same process to come up with α_3 and $T_3 = T(\alpha_3)$. Repeat this process until we arrive at the subtree with the root node only. Namely, we get the following sequence of pruned subtrees.

$$T_1 \geq T_2 \geq \cdots \geq \{t_1\}, \tag{15}$$

where $T_k = T(\alpha_k)$ and

$$0 = \alpha_1 < \alpha_2 < \cdots .$$

We need the following proposition for cross validation.

Proposition 4. *(i) If $\alpha_1 \leq \alpha_2$, then $T(\alpha_2) \leq T(\alpha_1)$*

(ii) For α with $\alpha_k \leq \alpha < \alpha_{k+1}$, $T(\alpha_k) = T(\alpha)$

Proof. Proof of (i): The proof is based on induction on $|\tilde{T}|$. Recall that the proof of Proposition 2 shows that $T(\alpha)$ is either $\{t_1\}$ or the join of $T_L(\alpha)$ and $T_R(\alpha)$ at t_1. Suppose $T(\alpha_1) = \{t_1\}$, i.e.,

$$R_\alpha(t_1) \leq R_\alpha(T'), \tag{16}$$

for any pruned subtree T' of T. Then

$$\begin{aligned} R_{\alpha_2}(t_1) &= R_{\alpha_1}(t_1) + \alpha_2 - \alpha_1 \\ &\leq R_{\alpha_1}(T') + \alpha_2 - \alpha_1 \\ &\leq R_{\alpha_1}(T') + (\alpha_2 - \alpha_1)|\tilde{T}'| \\ &= R_{\alpha_2}(T'), \end{aligned}$$

where the first inequality above is due to (16). Therefore we can easily conclude that $T(\alpha_2) = \{t_1\}$.

Assume now $T(\alpha_1)$ is nontrivial and is the join of $T_L(\alpha_1)$ and $T_R(\alpha_1)$ at t_1. By the induction hypothesis, we may assume that $T_L(\alpha_2) \leq T_L(\alpha_1)$ and $T_R(\alpha_2) \leq T_R(\alpha_1)$. Thus if $T(\alpha_2)$ is the join of $T_L(\alpha_2)$ and $T_R(\alpha_2)$ at t_1, the proof of (i) is complete. On the other hand, if $T(\alpha_2) = \{t_1\}$, then again it is obvious that $\{t_1\}$ is a pruned subtree of anything.

Proof of (ii): Suppose the assertion of (ii) is false. Then (i) says that $T(\alpha) < T(\alpha_k)$. Therefore the pruning of $T(\alpha_k)$ must have occurred at some node, say, at $t' \in T(\alpha_k)$. Let $T(\alpha_k)_{t'}$ be the branch of $T(\alpha_k)$ at t'. Namely, $T(\alpha_k)_{t'}$ is the subtree of $T(\alpha_k)$ with t' as its root node together with all of its descendant nodes in $T(\alpha_k)$. Since the pruning has occurred at t', we must have $R_\alpha(t') \leq R_\alpha(T(\alpha_k)_{t'})$. It is then not hard to see that $\alpha_{k+1} \leq \alpha$. This is a contradiction to the assumption of (ii). □

2.5 Model selection (validation)

Once the subtrees are found as in (15), our next job is to determine which one is the "best" subtree. It is really a model selection problem. As usual, there are two ways to do it: testing and cross validation.

Before we embark on the model selection task, let us review the underlying probability model. Recall the nature of data set $\mathfrak{D} = \{(x_i, y_i)\}_{i=1}^n$. The data point (x_i, y_i) is gotten as a random IID sample of the random variable (X, Y) drawn from the distribution $P_{X,Y}$. It is denoted in our usual notation by $(X, Y) \sim P_{X,Y}$. Let us now fix a tree T. Then T gives rise to a classifier f as explained in Definition 8 and the discussion following it.

With f and the probability $P = P_{X,Y}$, we can define the following quantities. Note that they depend only on P not on any particular data set $\mathfrak{D}$. In this sense, we call them *theoretical* quantities associated with f or equivalently with T.

(i) $Q^*(i|j) = P(f(x) = i|Y = j)$: the theoretical probability of classifying class j as being class i

(ii) $R^*(j) = \sum_i C(i|j)Q^*(i|j)$: the theoretical expected misclassification cost of the class j

(iii) $R^*(f) = R^*(T) = \sum_j R^*(j)\pi(j)$: the overall theoretical expected cost of the classifier f

Note that in case $C(i|j) = 1 - \delta_{ij}$, $R^*(f)$ is nothing but the misclassification

probability. Namely,

$$\begin{aligned}
R^*(f) &= \sum_j R^*(j)\pi(j) \\
&= \sum_j \sum_i C(i|j)Q^*(i|j)\pi(j) \\
&= \sum_j \sum_i (1-\delta_{ij})Q^*(i|j)\pi(j) \\
&= \sum_j \sum_i Q^*(i|j)\pi(j) - \sum_i Q^*(i|i)\pi(i) \\
&= 1 - \sum_i Q^*(i|i)\pi(i) \\
&= \sum_i \{1 - Q^*(i|i)\}\pi(i) \\
&= \sum_i P(f(X) \neq Y|Y=i)\pi(i) \\
&= P(f(X) \neq Y).
\end{aligned}$$

Method A: Using separate training and testing sets
Divide $\mathfrak{D}$ into disjoint training set $\mathfrak{D}^{(1)}$ and testing set $\mathfrak{D}^{(2)}$. So

$$\mathfrak{D} = \mathfrak{D}^{(1)} \cup \mathfrak{D}^{(2)}.$$

Use $\mathfrak{D}^{(1)}$ to construct

$$T_1 \geq T_2 \geq \cdots \geq \{t_1\}.$$

as in (15). Then use $\mathfrak{D}^{(2)}$ for testing and selecting the best model (T_k).

Now each T_k defines the classifier f_k. With it, define the following quantities: Let $N_j^{(2)}$ be the number of data points in $D^{(2)}$ with class label j and $N_{ij}^{(2)}(k)$ the number of data points in $D^{(2)}$ with class label j that is classified by f_k as class i. Define the estimator $Q_k^{ts}(i|j)$ for $Q^*(i|j)$ by

$$Q_k^{ts}(i|j) = N_{ij}^{(2)}(k)/N_j^{(2)}.$$

Here, 'ts' stands for "test sample." If $N_j^{(2)} = 0$, we set $Q_k^{ts}(i|j) = 0$. Similarly define the estimators $R_k^{ts}(j)$ and $R^{ts}(T_k)$ for $R^*(j)$ and $R^*(T_k)$, respectively

by

$$\begin{aligned} R_k^{ts}(j) &= \sum_i C(i|j) Q_k^{ts}(i|j) \\ R^{ts}(T_k) &= \sum_j R_k^{ts}(j)\pi(j). \end{aligned}$$

This way, the overall expected cost $R^{ts}(T_k)$ of the classifier f_k is obtained for each k. We then choose $T_{k'}$ with the smallest overall expected cost. Namely,

$$R^{ts}(T_{k'}) = \min_k R^{ts}(T_k).$$

Method B: Cross validation
As is the standard practice in the V-fold cross validation, divide $\mathfrak{D}$ into V equal (as nearly as possible) disjoint sets $\mathfrak{D}_1, \cdots, \mathfrak{D}_V$ and Let $\mathfrak{D}^{(v)} = \mathfrak{D} - \mathfrak{D}_v$, for $v = 1, \cdots, V$. We first grow $T_{\max}$ using $\mathfrak{D}$ and get $0 = \alpha_1 < \alpha_2 < \cdots$ and

$$T_1 \geq T_2 \geq \cdots \geq \{t_1\},$$

where $T_k = T(\alpha_k)$ Now these trees $T_1, T_2, \cdots$ cannot be used for cross validation because the data in the cross validation sets $\mathfrak{D}_v$ were already used to construct them, hence tainted. Now define

$$\alpha'_k = \sqrt{\alpha_k \alpha_{k+1}}.$$

For each $v \in \{1, 2, \cdots, V\}$ and $\alpha \in \{\alpha'_1, \alpha'_2, \cdots\}$, use $\mathfrak{D}^{(v)}$ to grow $T_{\max}^{(v)}$ and then find $T^{(v)}(\alpha)$ by applying the usual pruning process using R_α. Since $\mathfrak{D}_v$ is not used in the construction of $T^{(v)}(\alpha)$, it can be used as a cross validation set. So put down $\mathfrak{D}_v$ through $T^{(v)}(\alpha)$ to compute the following quantities

- $N_{ij}^{(v)}$: the number of class j in $\mathfrak{D}_v$ classified by $T^{(v)}(\alpha)$ as i
- $N_{ij} = \sum_v N_{ij}^{(v)}$: the total number of class j that are classified as i in the cross validation process

The idea is that for large V, $T^{(v)}(\alpha)$ and $T(\alpha)$ have very similar accuracy. Thus we can define the following estimates. In here, the symbol $\sim$ denotes

that the left of it is an estimator of the quantity in the right.

$$\begin{aligned} Q^{cv}(i|j) = N_{ij}/N &\sim Q^*(i|j) \\ R^{cv}(j) = \sum_i C(i|j) Q^{cv}(i|j) &\sim R^*(j) \\ R^{cv}(T(\alpha)) = \sum_j R^{cv}(j)\pi(j) &\sim R^*(T(\alpha)). \end{aligned}$$

Now by Proposition 4, $T_{\alpha_{k'}} = T_{\alpha_k} = T_k$. Therefore we get the estimates for $R^{cv}(T_k)$ by

$$R^{cv}(T_k) = R^{cv}(T_{\alpha_k}).$$

Finally, choose $T_{k'}$ among $T_1, T_2, \cdots, \{t_1\}$ such that

$$R^{cv}(T_{k'}) = \min_k R^{cv}(T_k).$$

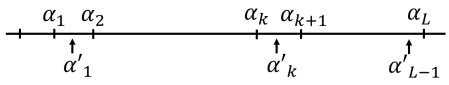

Figure 19: α'_k

3 Details of regression tree

We now look at the regression tree. The data is $\mathfrak{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, where $x^{(i)} \in X$ and $y^{(i)} \in \mathfrak{Y}$. Typically $\mathfrak{X} = \mathbb{R}^d$ and $\mathfrak{Y} = \mathbb{R}^K$. The goal of regression tree algorithm is to construct a function $f : \mathfrak{X} \to \mathfrak{Y}$ such that the error

$$\sum_i |f(x^{(i)}) - y^{(i)}|^2$$

is small. The way to do it is to construct a tree and define a constant value on each subregion corresponding to the terminal node of the tree. Thus f constructed this way is a piecewise constant function.

Recall the way we construct a tree. In particular, any node $t \in T$ corresponds to a subset of $\mathfrak{X}$. On each node t, define the average y-value $\bar{y}(t)$ of the data on the node t by

$$\bar{y}(t) = \frac{1}{N(t)} \sum_{x^{(i)} \in t} y^{(i)},$$

which is an estimator of $E[Y|X \in t]$. We also define the (squared) error rate $r(t)$ of node t by

$$r(t) = \frac{1}{N(t)} \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t))^2.$$

It is nothing but the variance of the node t, which is also an estimator of

$$\mathrm{Var}(Y|X \in t) = \sigma^2(Y|X \in t).$$

We define the cost $R(t)$ of the node t by

$$R(t) = r(t)p(t).$$

Recall that $p(t) = \dfrac{N(t)}{N}$. Therefore

$$R(t) = \frac{1}{N} \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t))^2.$$

We need the following proposition.

Proposition 5. *Suppose a node t is split into t_L and t_R. Then*

$$R(t) \geq R(t_L) + R(t_R).$$

The equality holds if and only if $\bar{y}(t) = \bar{y}(t_L) = \bar{y}(t_R)$.

Proof.

$$\begin{aligned} R(t) &= \frac{1}{N} \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t))^2 \\ &= \frac{1}{N} \sum_{x^{(i)} \in t_L} (y^{(i)} - \bar{y}(t))^2 + \frac{1}{N} \sum_{x^{(i)} \in t_R} (y^{(i)} - \bar{y}(t))^2 \\ &\geq \frac{1}{N} \sum_{x^{(i)} \in t_L} (y^{(i)} - \bar{y}(t_L))^2 + \frac{1}{N} \sum_{x^{(i)} \in t_R} (y^{(i)} - \bar{y}(t_R))^2, \end{aligned}$$

where the last inequality is due to following Lemma 1. The statement on the equality also follows from Lemma 1. □

Lemma 1.

$$\bar{y}(t) = \operatorname*{argmin}_a \sum_{x^{(i)} \in t} (y^{(i)} - a)^2.$$

i.e.,

$$\sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t))^2 \leq \sum_{x^{(i)} \in t} (y^{(i)} - a)^2.$$

for all $a \in \mathbb{R}$ and "=" holds if and only if $a = \bar{y}(t)$.

Proof.

$$\begin{aligned}
\sum_{x^{(i)} \in t} (y^{(i)} - a)^2 &= \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t) + \bar{y}(t) - a)^2 \\
&= \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t))^2 - 2(\bar{y}(t) - a) \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t)) + N(t)(\bar{y}(t) - a)^2 \\
&= \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t))^2 + N(t)(\bar{y}(t) - a)^2 \\
&\geq \sum_{x^{(i)} \in t} (y^{(i)} - \bar{y}(t))^2
\end{aligned}$$

Clearly the equality holds if and only if $\bar{y}(t) - a = 0$. □

Let $\mathfrak{s}$ be a split of a node t. Define the decrease $\Delta R(\mathfrak{s}, t)$ of the cost by $\mathfrak{s}$ as

$$\Delta R(\mathfrak{s}, t) = R(t) - R(t_L) - R(t_R).$$

The splitting rule at t is s^* such that we take the split $\mathfrak{s}^*$ among all possible candidate splits that decreases the cost most. Namely,

$$\Delta R(\mathfrak{s}^*, t) = \max_{\mathfrak{s}} \Delta R(\mathfrak{s}, t).$$

Remark. *One may use this splitting rule for the split of classification tree.*

This way, we can grow the regression tree to $T_{\max}$. As before, one quick rule of thumb is that one stops splitting the node if the number of elements of the node is less than the preset number. Once $T_{\max}$ is found, we can prune back. The pruning method for the regression tree is exactly the same for the classification tree except that we define $R^{ts}(T)$ and $R^{cv}(T)$ differently.

For the test sample method, divide $\mathfrak{D} = \mathfrak{D}^{(1)} \cup \mathfrak{D}^{(2)}$, and use $\mathfrak{D}^{(1)}$ for constructing the tree and $\mathfrak{D}^{(2)}$ for testing. Define

$$R^{ts}(T) = \frac{1}{|\mathfrak{D}^{(2)}|} \sum_{(x^{(i)},y^{(i)})\in\mathfrak{D}^{(2)}} (y^{(i)} - \bar{y}(\tau(x^{(i)})))^2,$$

where τ is the map $\tau : \mathfrak{X} \to \tilde{T}$ such that $\tau(x)$ is the terminal node that contains x. As was done in the case of classification tree, this $R^{ts}(T)$ is used for choosing the best tree.

For the cross-validation method, use the division $\mathfrak{D} = \mathfrak{D}_1 \cup \cdots \cup \mathfrak{D}_V$ and for each v use $\mathfrak{D}^{(v)} = \mathfrak{D} - \mathfrak{D}_v$ for constructing the tree $T^{(v)}$, and $\mathfrak{D}_v$ for cross validation. Define

$$R^{cv}(T) = \frac{1}{V} \sum_{v=1}^{V} \sum_{(x^{(i)},y^{(i)})\in\mathfrak{D}_v} (y^{(i)} - \bar{y}^{(v)}(\tau^{(v)}(x^{(i)}))),$$

where $\tau^{(v)} : \mathfrak{X} \to \tilde{T}^{(v)}$ is the map such that $\tau^{(v)}(x)$ is the terminal node of $T^{(v)}$ containing x, and $\bar{y}^{(v)}$ is gotten using $\mathfrak{D} - \mathfrak{D}_v$ as the training set. The rest of the cross validation is routine.

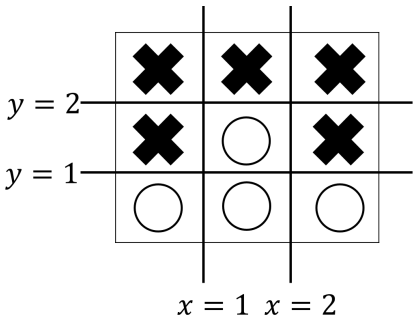

Figure 20: The training set

Homework. *Suppose the data is given as in Figure 20.*

(1) *Construct the classification tree $T_{\max}$ all of whose terminal nodes are pure (having all ◯ or ×). You may use only the lines $x = 1, x = 2, y = 1, y = 2$ as possible split candidates. When split, use the Gini impurity.*

(2) *Prune* $T_{\max}$ *using the weakest-link cut to come up with a sequence of subtrees:*

$$T_1 \geq T_2 \geq \cdots \geq \{root\}.$$

(3) *Suppose the test set is given as in Figure 21. Use this test set to choose the "best"* T_k.

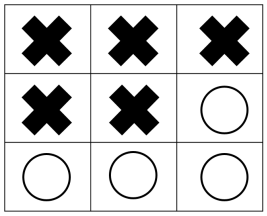

Figure 21: The test set

References

[1] Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J., *Classification And Regression Trees*, Chapman & Hall/CRC (1984)