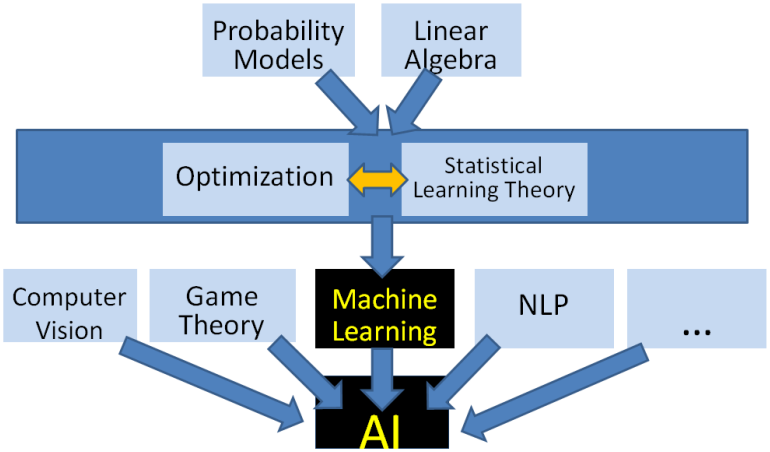
MSDS 422: Practical Machine Learning

Tree-Based Methods



NORTHWESTERN UNIVERSITY

Dr. Nathan Bastian

References

- *An Introduction to Statistical Learning, with Applications in R* (2013), by G. James, D. Witten, T. Hastie, and R. Tibshirani.

- *Classification and Regression Trees* (1984), by L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone.

- *The Elements of Statistical Learning* (2009), by T. Hastie, R. Tibshirani, and J. Friedman.

- *Machine Learning: A Probabilistic Perspective* (2012), by K. Murphy

Tree-Based Methods

- *Tree-based* methods for regression and classification involve stratifying or segmenting the predictor space into a number of simple regions.

- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of machine learning approaches are known as *decision tree* methods.

- The basic idea of these methods is to partition the space and identify some representative centroids.

Regression Trees

- One way to make predictions in a regression problem is to divide the predictor space (i.e. all the possible values for $X_1, X_2, \ldots, X_p$) into distinct regions, say $R_1, R_2, \ldots, R_k$ (*terminal nodes* or *leaves*).

- Then, for every X that falls into a particular region (say R_j) we make the same prediction.

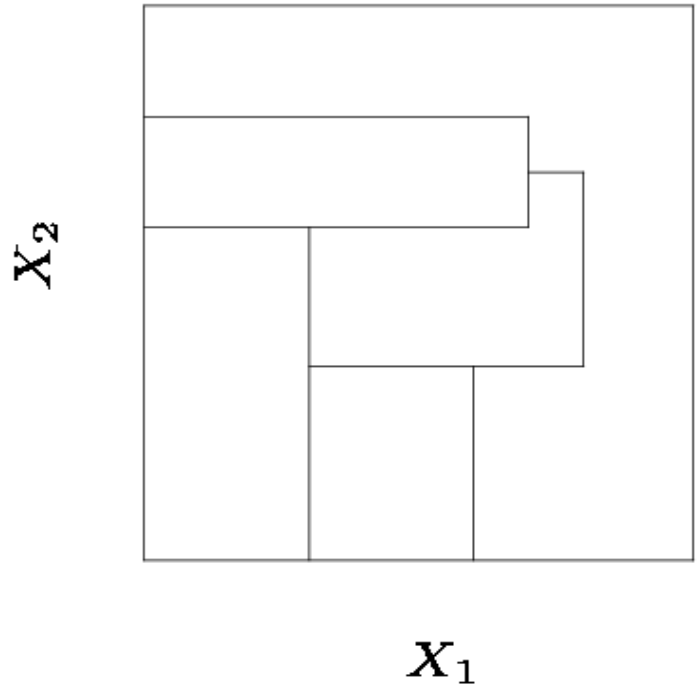
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.

- The points along the tree where the predictor space is split are referred to as *internal nodes*. The segments of the trees that connect the nodes are *branches*.

Regression Trees (cont.)

- Suppose we have two regions R_1 and R_2 with $\hat{Y}_1 = 10$ and $\hat{Y}_2 = 20$, respectively.

- For any value of X such that $X \in R_1$, we would predict 10. For any value of X such that $X \in R_2$, we would predict 20.

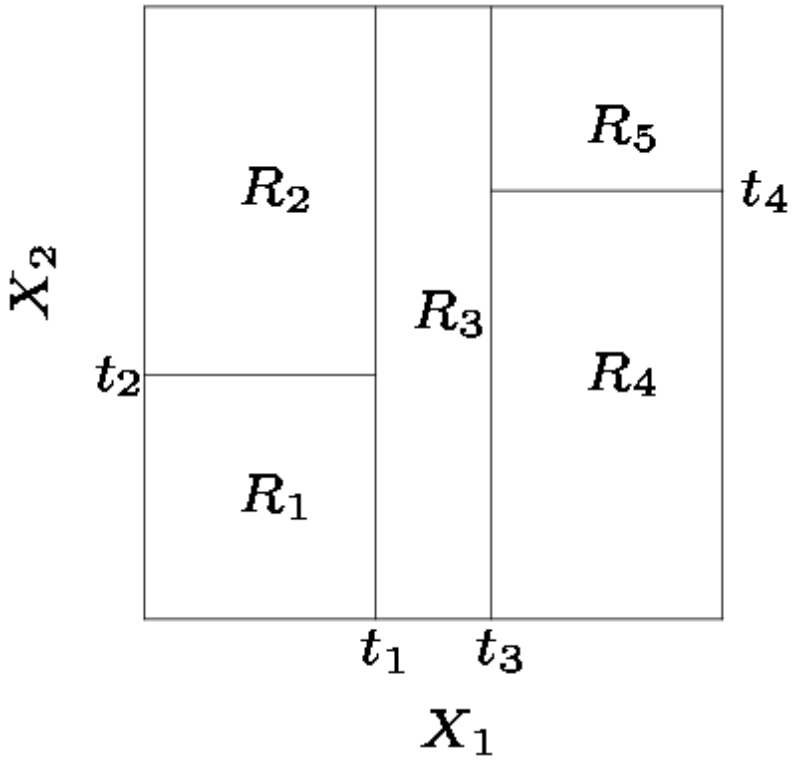


- In this illustrative example, we have two predictors and five distinct regions. Depending on which region the new X comes from, we would make one of five possible predictions for Y.

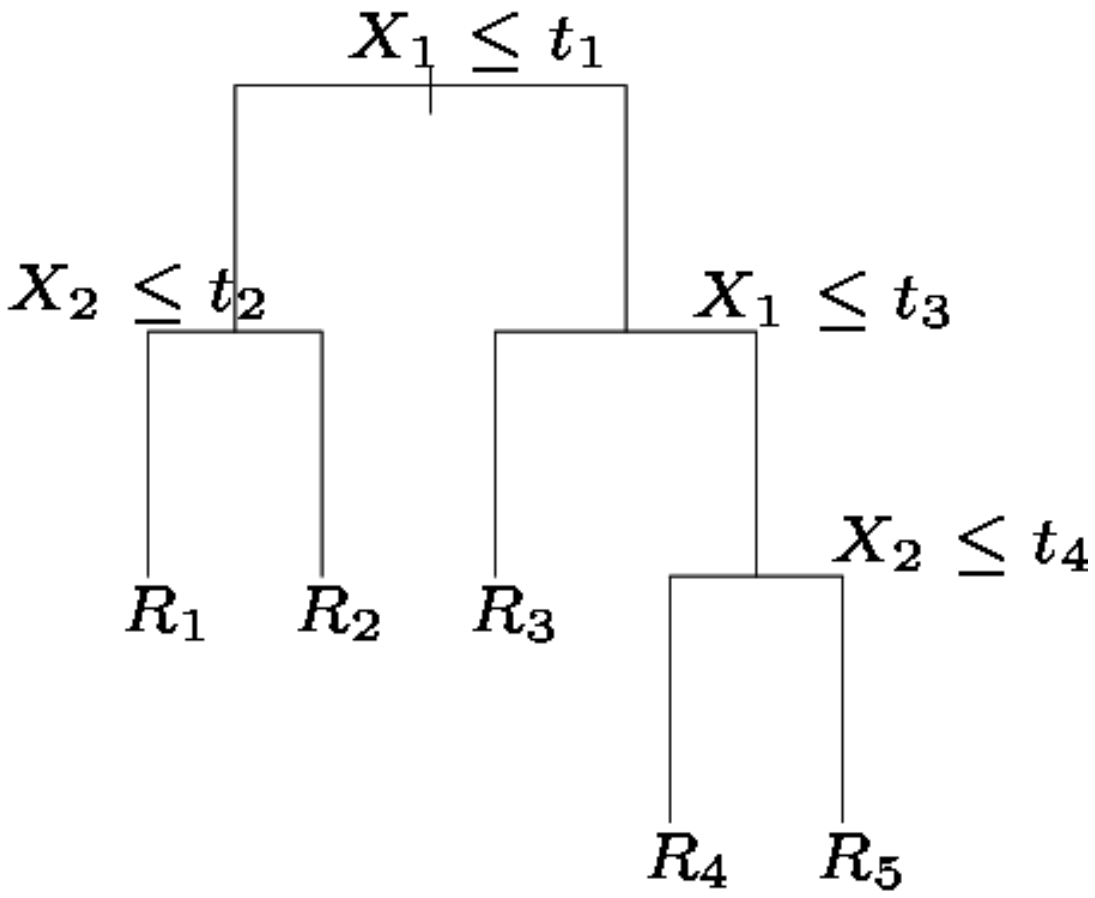
Regression Trees (cont.)

- Generally, we create the partitions by iteratively splitting one of the X variables into two regions.

- First split on $X_1 = t_1$
- If $X_1 < t_1$, split on $X_2 = t_2$
- If $X_1 > t_1$, split on $X_1 = t_3$
- If $X_1 > t_3$, split on $X_2 = t_4$



Regression Trees (cont.)



- When we create partitions this way, we can always represent them using a tree structure.
- As a result, this provides a very simple way to explain the model to a non-expert.

Classification Trees

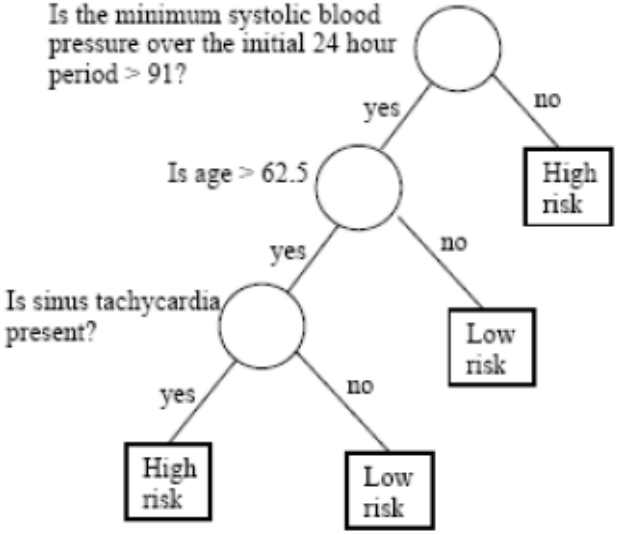
- Classification trees are a hierarchical way of partitioning the space.
- We start with the entire space and recursively divide it into smaller regions.
- At the end, every region is assigned with a class label.
- We start with a medical example to get a rough idea about classification trees.

Classification Trees (cont.)

- One big advantage for decision trees is that the classifier generated is highly interpretable. For physicians, this is an especially desirable feature.
- In this example, patients are classified into one of two classes: high risk versus low risk.
- It is predicted that the high risk patients would not survive at least 30 days based on the initial 24-hour data.
- There are 19 measurements taken from each patient during the first 24 hours. These include blood pressure, age, etc.

Classification Trees (cont.)

- Here, we generate a tree-structured classification rule, which can be interpreted as follows:



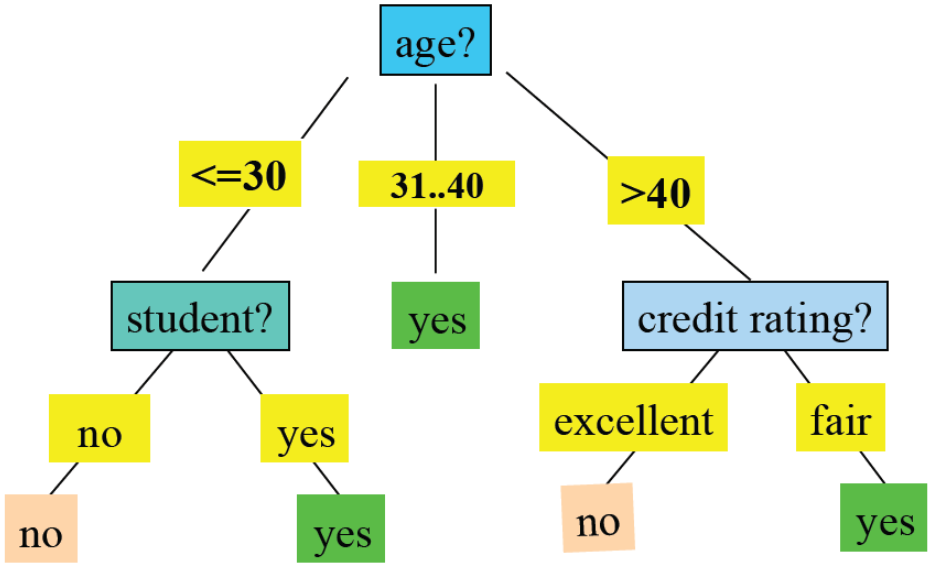
- Only three measurements are looked at by this classifier. For some patients, only one measurement determines the final result.
- Classification trees operate similarly to a doctor's examination.

Classification Trees (cont.)

- First we look at the minimum systolic blood pressure within the initial 24 hours and determine whether it is above 91.
- If the answer is no, the patient is classified as high-risk. We don't need to look at the other measurements for this patient.
- If the answer is yes, then we can't make a decision yet. The classifier will then look at whether the patient's age is greater than 62.5 years old.
- If the answer is no, the patient is classified as low risk. However, if the patient is over 62.5 years old, we still cannot make a decision and then look at the third measurement, specifically, whether sinus tachycardia is present.
- If the answer is no, the patient is classified as low risk. If the answer is yes, the patient is classified as high risk.

Classification Trees (cont.)

- **Business marketing:** predict whether a person will buy a computer?



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Classification Trees (cont.)

Notation:

- We will denote the feature space by **X**. Normally **X** is a multidimensional Euclidean space.

- However, sometimes some variables may be categorical such as gender (male or female).

- Classification and Regression Trees (CART) have the advantage of treating real variables and categorical variables in a unified manner.

- This is not so for many other classification methods, such as LDA.

Classification Trees (cont.)

Notation (cont'd):

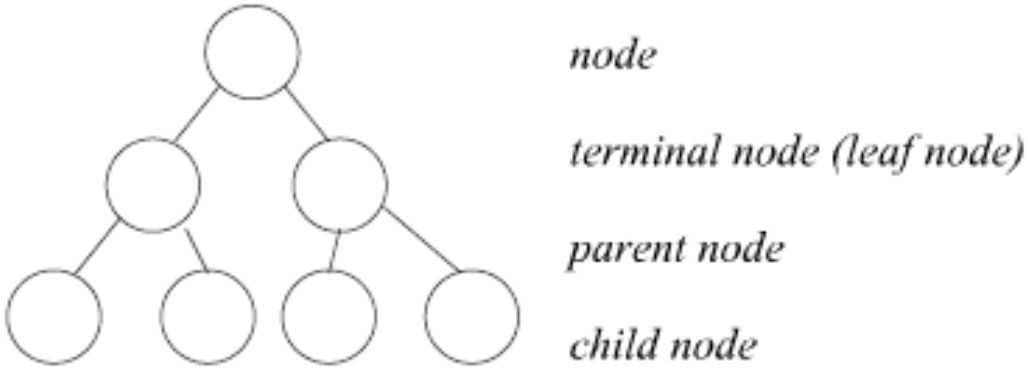
- The input vector is indicated by $X \in \boldsymbol{X}$ contains p features $X_1, \ldots, X_p$
- Tree structured classifiers are constructed by repeated splits of the space X into smaller and smaller subsets, beginning with **X** itself.
- *Node*: Any of the nodes in the classification tree, where each corresponds to a region in the original space.
- *Terminal Node*: The final node resulting from successive splits, where each is assigned a unique class.

Classification Trees (cont.)

Notation (cont'd):

- *Parent Node*: those nodes that are split into two child nodes.

- *Child Node*: these result from the splitting of a parent node. Two child nodes are two different regions. Together they occupy the same region of the parent node.



Classification Trees (cont.)

Notation (cont'd):

- One thing that we need to keep in mind is that the tree represents recursive splitting of the space.
- Therefore, every node of interest corresponds to one region in the original space.
- Two child nodes will occupy two different regions and if we put the two together, we get the same region as that of the parent node.
- In the end, every leaf node is assigned with a class and a test point is assigned with the class of the leaf node it lands in.

Classification Trees (cont.)

Notation (cont'd):

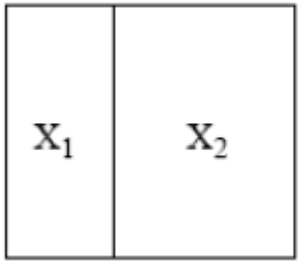
- A node is denoted by t. We will also denote the *left* child node by t_L and the *right* one by t_R.

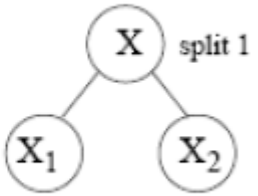
- We denote the collection of all the nodes in the tree by T and the collection of all the leaf nodes by $\tilde{T}$.

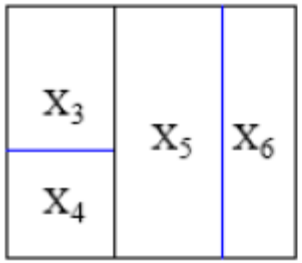
- A split will be denoted by s, and the set of splits is denoted by S.

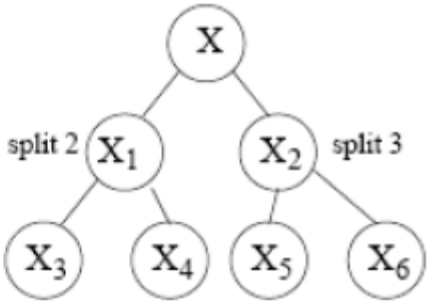
- Let's next take a look at how these splits can take place, where the whole space is represented by X.

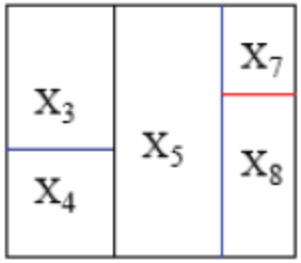
Classification Trees (cont.)

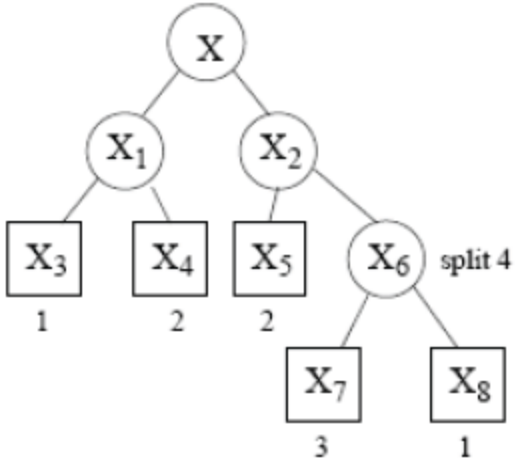




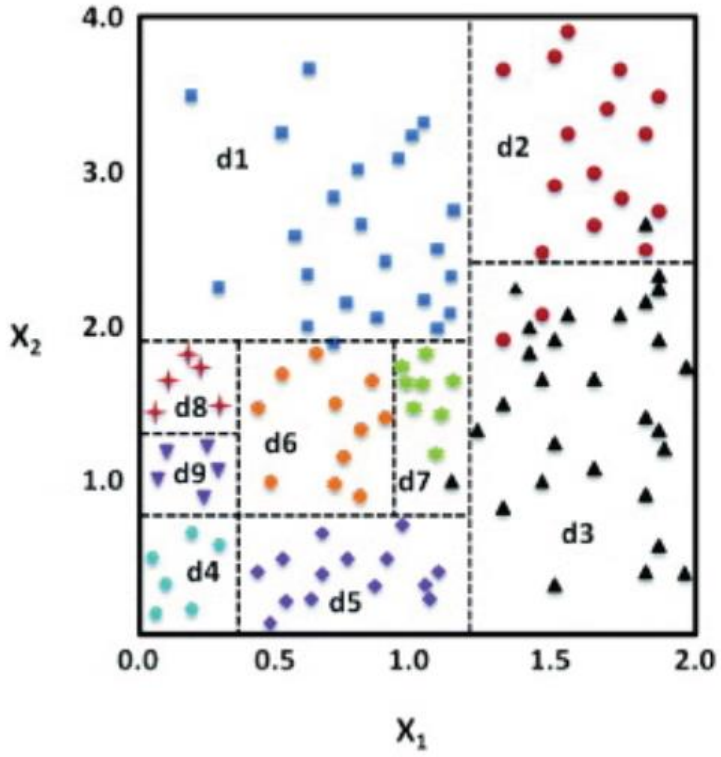




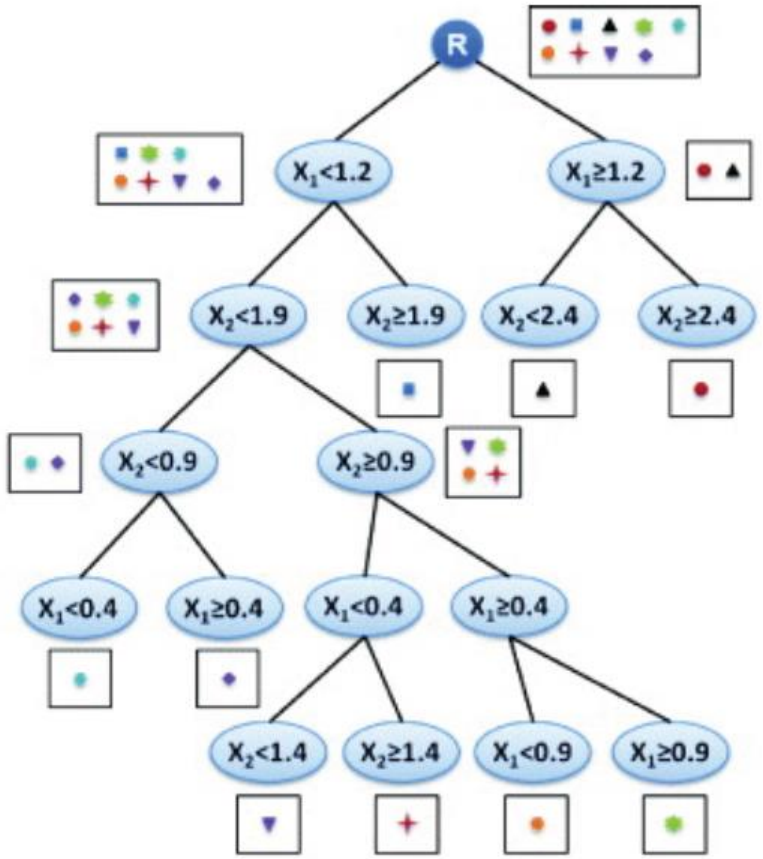




Classification Trees (cont.)



(a)



(b)

Classification Trees (cont.)

- The construction of a tree involves the following three elements:
  - The selection of the splits, i.e., how do we decide which node (region) to split and how to split it?
  - If we know how to make splits or 'grow' the tree, how do we decide when to declare a node terminal and stop splitting?
  - We have to assign each terminal node to a class. How do we assign these class labels?

- In particular, we need to decide upon the following:
  - The pool of candidate splits that we might select from involves a set Q of binary questions of the form $\{\text{Is } \mathbf{x} \in A?\}$, $A \subseteq \mathbf{X}$. Basically, we ask whether our input $\mathbf{x}$ belongs to a certain region, A. We need to pick one A from the pool.

Classification Trees (cont.)

- In particular, we need to decide upon the following (cont'd):
  - The candidate split is evaluated using a *goodness of split* criterion $\Phi(s, t)$ that can be evaluated for any split s of any node t.
  - A stop-splitting rule, i.e., we have to know when it is appropriate to stop splitting. One can 'grow' the tree very big. In an extreme case, one could 'grow' the tree to the extent that in every leaf node there is only a single data point. Then it makes no sense to split any farther.
  - Finally, we need a rule for assigning every terminal node to a class.
- Next, we get into the details for each of these four decisions that we have to make.

Classification Trees (cont.)

- **Standard Set of Questions for Suggesting Possible Splits**
  - Let's say that the input vector $X = (X_1, \ldots, X_p)$ contains features of both categorical and ordered types.
  - CART makes things simple because every split depends on the value of only a single, *unique* variable.
  - If we have an ordered variable (e.g. X_j), the question inducing the split is whether X_j is smaller or equal to some threshold? Thus, for each ordered variable X_j , Q includes all questions of the form: $\{\text{Is } X_j \leq c?\}$ for all real-valued c.
  - There are other ways to partition the space. For instance, you might ask whether $X_1 + X_2$ is smaller than some threshold.

Classification Trees (cont.)

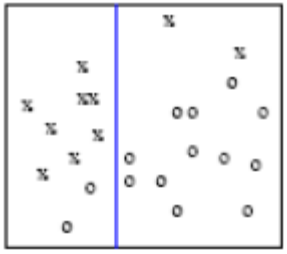
- **Standard Set of Questions for Suggesting Possible Splits (cont'd)**
  - In this case, the split line is not parallel to the coordinates. However, here we restrict our interest to the questions of the above format. Every question involves one of $X_1, \ldots, X_p$, and a threshold.
  - Since the training data set is finite, there are only *finitely* many thresholds c that result in distinct division of the data points.
  - If X_j is categorical, say taking values from $\{1, 2, \ldots, M\}$, the questions Q are of the form: $\{\text{Is } X_j \in A?\}$, where A is any subset of $\{1, 2, \ldots, M\}$.
  - The splits or questions for all p variables form the pool of candidate splits. This first step identifies all the candidate questions. Which one to use at any node when constructing the tree is the next question.

Classification Trees (cont.)

- **Determining Goodness of Split**
  - The way we choose the question, i.e., split, is to measure every split by a *goodness of split* measure, which depends on the split question as well as the node to split.
  - The *goodness of split* in turn is measured by an *impurity function*.
  - Intuitively, when we split the points we want the region corresponding to each leaf node to be "pure", that is, most points in this region come from the same class, that is, one class dominates.
  - Let’s look at the following example…

Classification Trees (cont.)

- **Determining Goodness of Split (cont'd)**
  - We have two classes shown in the plot by **x**'s and **o**'s. We could split first by checking whether the horizontal variable is above or below a threshold:

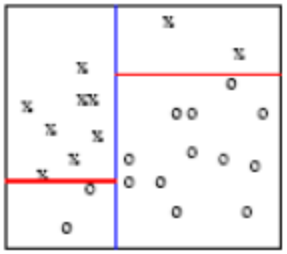


  - The split is indicated by the blue line. Remember by the nature of the candidate splits, the regions are always split by lines parallel to either coordinate.

  - For the example split above, we might consider it a good split because the left-hand side is nearly pure in that most of the points belong to the **x** class. Only two points belong to the **o** class. The same is true of the right-hand side.

Classification Trees (cont.)

- **Determining Goodness of Split (cont'd)**
  - Generating *pure* nodes is the intuition for choosing a split at every node. If we go one level deeper down the tree, we have created two more splits.



  - Now you see that the upper left region or leaf node contains only the **x** class. Therefore, it is 100% *pure*, no class blending in this region. The same is true for the lower right region. It only has **o**'s.

  - Once we have reached this level, it is unnecessary to further split because all the leaf regions are 100% pure.

Classification Trees (cont.)

The Impurity Function

- The *impurity function* measures the extent of purity for a region containing data points from possibly different classes.

- Suppose the number of classes is K. Then the impurity function is a function of $p_1, \ldots, p_K$,the probabilities for any data point in the region belonging to class 1, 2,..., K.

- During training, we do not know the real probabilities. What we would use is the percentage of points in class 1, class 2, class 3, and so on, according to the training data set.

Classification Trees (cont.)

- The impurity function can be defined in different ways, but the bottom line is that is satisfies three properties.

- **Definition:** An **impurity function** is a function Φ defined on the set of all K-tuples of numbers $(p_1, \ldots, p_K)$ satisfying $p_j \geq 0, j = 1, \ldots, K$, $\Sigma_j p_j = 1$ with the properties:
  - Φ achieves maximum only for the uniform distribution, that is all the p_j are equal.
  - Φ achieves minimum only at the points $(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0), \ldots, (0, 0, \ldots, 0, 1)$, i.e., when the probability of being in a certain class is 1 and 0 for all the other classes.
  - Φ is a symmetric function of $p_1, \ldots, p_K$, i.e., if we permute p_j, Φ remains constant.

Classification Trees (cont.)

- **Definition:** Given an impurity function Φ, define the impurity measure, denoted as $i(t)$, of a node t as follows:

$$i(t) = \phi(p(1|t), p(2|t), \ldots, p(K|t))$$

where $p(j \mid t)$ is the estimated posterior probability of class j given a point is in node t.

- This is called the *impurity function* or the *impurity measure* for node t.

Classification Trees (cont.)

- Once we have $i(t)$, we define the goodness of split s for node t, denoted by $\Phi(s, t)$:

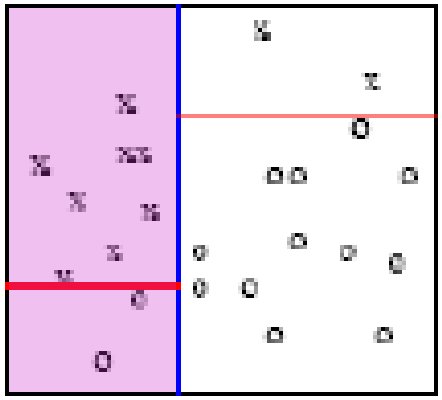
$$\Phi(s,t) = \Delta i(s,t) = i(t) - p_R i(t_R) - p_L i(t_L)$$

- $\Delta i(s, t)$ is the difference between the impurity measure for node t and the weighted sum of the impurity measures for the right child and the left child nodes.

- The weights, p_R and p_L , are the proportions of the samples in node t that go to the right node t_R and the left node t_L respectively.

Classification Trees (cont.)

- Look at the graphic again. Suppose the region to the left (shaded purple) is the node being split, the upper portion is the left child node the lower portion is the right child node.



- You can see that the proportion of points that are sent to the left child node is $p_L = 8/10$. The proportion sent to the right child is $p_R = 2/10$.

Classification Trees (cont.)

- The classification tree algorithm goes through all the candidate splits to select the best one with maximum $\Delta i(s, t)$.

- Next, we will define $I(t) = i(t)\, p(t)$, that is, the impurity function of node t weighted by the estimated proportion of data that go to node t. $p(t)$ is the probability of data falling into the region corresponding to node t.

- A simple way to estimate it is to count the number of points that are in node t and divide it by the total number of points in the whole data set.

Classification Trees (cont.)

- The aggregated impurity measure of tree T, $I(T)$ is defined by

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

- This is the sum over all leaf nodes (not all nodes in the tree). Note for any node t the following equations hold:

$$\begin{aligned} p(t_L) + p(t_R) &= p(t) \\ p_L &= p(t_L)/p(t) \\ p_R &= p(t_R)/p(t) \\ p_L + p_R &= 1 \end{aligned}$$

Classification Trees (cont.)

- The region covered by the left child node, t_L, and the right child node, t_R, are *disjoint* and if combined, form the bigger region of their parent node t.

- The sum of the probabilities over two disjoined sets is equal to the probability of the union. p_L then becomes the relative proportion of the left child node with respect to the parent node.

- Next, we define the difference between the weighted impurity measure of the parent node and the two child nodes.

Classification Trees (cont.)

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

- It should be understood that no matter what impurity function we use, the way you use it in a classification tree is the same.

- The only difference is what specific impurity function to plug in. Once you use this, what follows is the same.

Classification Trees (cont.)

- Possible impurity functions:
  - Entropy function: $\sum_{j=1}^{K} p_j \log \frac{1}{p_j}$. If $p_j = 0$, use the limit $\lim p_j \to \log p_j = 0$.
  - Misclassification rate: $1 - \max_j p_j$.
  - Gini index: $\sum_{j=1}^{K} p_j \left(1 - p_j\right) = 1 - \sum_{j=1}^{K} {p_j}^2$
- Remember, impurity functions have to 1) achieve a maximum at the uniform distribution, 2) achieve a minimum when $p_j = 1$, and 3) be symmetric with regard to their permutations.

Classification Trees (cont.)

- Another splitting method is the **Twoing Rule**. This approach does not have anything to do with the impurity function.

- The intuition here is that the class distributions in the two child nodes should be as different as possible and the proportion of data falling into either of the child nodes should be balanced.

- The twoing rule: At node *t*, choose the split *s* that maximizes:

$$\frac{p_L p_R}{4}\left[\sum_j |p(j|t_L) - p(j|t_R)|\right]^2$$

Classification Trees (cont.)

- When we break one node to two child nodes, we want the posterior probabilities of the classes to be *as different as possible*. If they differ a lot, each tends to be *pure*.

- If instead the proportions of classes in the two child nodes are roughly the same as the parent node, this indicates the splitting does not make the two child nodes much purer than the parent node and hence not a successful split.

- In summary, one can use either the *goodness of split* defined using the impurity function or the *twoing rule*. At each node, try all possible splits exhaustively and select the best from them.

Classification Trees (cont.)

- The impurity function is a function of the posterior probabilities of k classes.
- We answer the question: *How do we estimate these probabilities?*
- Let's begin by introducing the notation N, the total number of samples. The number of samples in class j, $1 \leq j \leq K$, is N_j .
- If we add up all the N_j data points, we get the total number of data points N.

Classification Trees (cont.)

- We also denote the number of samples going to node t by $N(t)$, and, the number of samples of class j going to node t by $N_j(t)$.

- Then for every node t, if we add up over different classes we should get the total number of points back:

$$\sum_{j=1}^{K} N_j(t) = N(t)$$

- And, if we add the points going to the left and the points going the right child node, we should also get the number of points in the parent node: $N_j(t_L) + N_j(t_R) = N_j(t)$

Classification Trees (cont.)

- For a full tree (balanced), the sum of $N(t)$ over all the node t's at the same level is N.

- Next, we will denote the prior probability of class j by π_j . The prior probabilities very often are estimated from the data by calculating the proportion of data in every class.

- For instance, if we want the prior probability for class 1, we simply compute the ratio between the number of points in class one and the total number of points, N_j / N. These are the so-called *empirical frequencies* for the classes.

Classification Trees (cont.)

- This is one way of getting priors. Sometimes the priors may be pre-given. For instance, in medical studies, researchers collect a large amount of data from patients who have a disease.

- The percentage of cases with the disease in the collected data may be much higher than that in the population.

- In this case, it is inappropriate to use the empirical frequencies based on the data.

- If the data is a random sample from the population, then it may be reasonable to use empirical frequency.

Classification Trees (cont.)

- The estimated probability of a sample in class j going to node t is $p(t \mid j) = N_j(t) / N_j$. Obviously, $p(t_L \mid j) + p(t_R \mid j) = p(t \mid j)$.

- Next, we can assume that we know how to compute $p(t \mid j)$ and then we will find the joint probability of a sample point in class j and in node t.

- The joint probability of a sample being in class j and going to node t is as follows: $p(j, t) = \pi_j \, p(t \mid j) = \pi_j \, N_j(t) / N_j$.

- Because the prior probability is assumed known (or calculated) and $p(t \mid j)$ is computed, the joint probability can be computed.

Classification Trees (cont.)

- The probability of any sample going to node t regardless of its class:

$$p(t) = \sum_{j=1}^{K} p(j,t) = \sum_{j=1}^{K} \pi_j N_j(t)/N_j$$

- Now, what we really need is $p(j \mid t)$. That is, if I know a point goes to node t, what is the probability this point is in class j.

- The probability of a sample being in class j given that it goes to node t is: $p(j \mid t) = p(j, t) / p(t)$.

- Probabilities on the right hand side are both solved from the previous formulas.

Classification Trees (cont.)

- For any t, $\sum_{j=1}^{K} p(j|t) = 1$.

- There is a shortcut if the prior is not pre-given, but estimated by the empirical frequency of class j in the dataset!

- When $\pi_j = N_j / N$, the simplification is as follows:
  - Calculate $p(j \mid t) = N_j(t) / N(t)$ - for all the points that land in node t
  - $p(t) = N(t) / N$
  - $p(j, t) = N_j(t) / N$

- This is the shortcut equivalent to the previous approach.

Classification Trees (cont.)

Determining Stopping Criteria

- When we grow a tree, there are two basic types of calculations needed.

- First for every node, we compute the posterior probabilities for the classes, that is, $p(j \mid t)$ for all j and t.

- Then we have to go through all the possible splits and exhaustively search for the one with the maximum goodness.

Classification Trees (cont.)

Determining Stopping Criteria (cont'd)

- Suppose we have identified 100 candidate splits (i.e., splitting questions), to split each node, 100 class posterior distributions for the left and right child nodes each are computed, and 100 goodness measures are calculated.

- At the end, one split is chosen and only for this chosen split, the class posterior probabilities in the right and left child nodes are stored.

- For the moment, let's assume we will leave off our discussion of pruning for later and that we will grow the tree until some sort of stopping criteria is met.

Classification Trees (cont.)

Determining Stopping Criteria (cont'd)

1. The goodness of split is lower than a threshold.
2. The samples in a node have the same labels.
3. The number of samples in every leaf is below a threshold.
4. The number of leaf nodes reaches a limit
5. Tree has exceeded the maximum desired depth.

Classification Trees (cont.)

Determining Stopping Criteria (cont'd)

- A simple criterion is as follows. We will stop splitting a node t when:

$$\max_{s \in S} \Delta I(s,t) < \beta$$

where ΔI (defined before) is the decrease in the impurity measure weighted by the percentage of points going to node t, s is the optimal split and β is a pre-determined threshold.

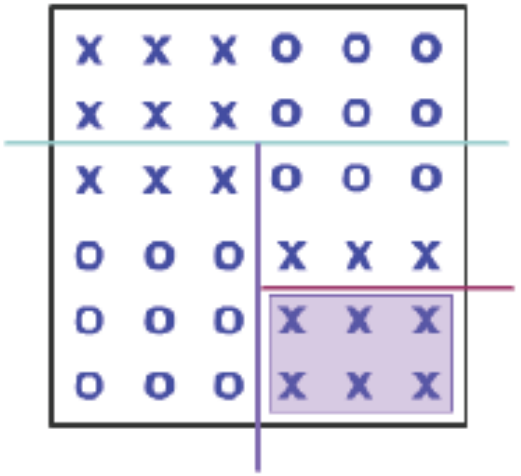
- We must note, however, the above stopping criterion for deciding the size of the tree is *not a satisfactory* strategy. The reason is that the tree growing method is greedy.

Classification Trees (cont.)

Determining Stopping Criteria (cont'd)

- The split at every node is 'nearsighted' in the sense that we can only look one step forward.

- A bad split in one step may lead to very good splits in the future. The tree growing method does not consider such cases.



- This process of growing the tree is *greedy* because it looks only one step ahead as it goes.

Classification Trees (cont.)

Determining Stopping Criteria (cont'd)

- Going one step forward and making a bad decision doesn't mean that it is always going to end up bad.
- If you go a few steps more you might actually gain something. You might even perfectly separate the classes.
- A response to this might be that what if we looked two steps ahead?
- How about three steps? You can do this but it begs the question, "How many steps forward should we look?"

Classification Trees (cont.)

Determining Stopping Criteria (cont'd)

- No matter how many steps we look forward, this process will always be greedy.

- Looking ahead multiple steps will not fundamentally solve this problem.

- This is why we need *pruning*.

Classification Trees (cont.)

Determining Class Assignment Rules

- Finally, how do we decide which class to assign to each leaf node?
- The decision tree classifies new data points as follows. We let a data point pass down the tree and see which leaf node it lands in.
- The class of the leaf node is assigned to the new data point.
- Basically, all the points that land in the same leaf node will be given the same class.

Classification Trees (cont.)

Determining Class Assignment Rules (cont'd)

- A class assignment rule assigns a class $j = \{1, \ldots, K\}$ to every terminal (leaf) node $t \in \tilde{T}$.

- The class is assigned to node t . $\tilde{T}$ is denoted by $\kappa(t)$, e.g., if $\kappa(t) = 2$, all the points in node t would be assigned to class 2.

- If we use 0-1 loss, the class assignment rule is as follows:

$$\kappa(t) = \arg \max_j p(j|t)$$

Classification Trees (cont.)

Determining Class Assignment Rules (cont'd)

- Let's assume for a moment that I have a tree and have the classes assigned for the leaf nodes.

- Now, I want to estimate the classification error rate for this tree.

- In this case, we need to introduce the *resubstitution estimate* $r(t)$ for the probability of misclassification, given that a case falls into node t:

$$r(t) = 1 - \max_j p(j|t) = 1 - p(\kappa(t)|t)$$

Classification Trees (cont.)

Determining Class Assignment Rules (cont'd)

- Denote $R(t) = r(t)\, p(t)$. The resubstitution estimation for the overall misclassification rate $R(T)$ of the tree classifier T is:

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

- One thing that we should spend some time proving is that if we split a node t into child nodes, the misclassification rate is ensured to improve.

- In other words, if we estimate the error rate using the resubstitution estimate, the more splits, the better.

Classification Trees (cont.)

Determining Class Assignment Rules (cont'd)

- This also indicates an issue with estimating the error rate using the re-substitution error rate because it is always biased towards a bigger tree. Let's briefly go through the proof.

- **Proposition:** For any split of a node t into t_L and t_R,

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

- **Proof:** Denote $j^* = \kappa(t)$. Let's take a look at being in class j^* given that you are in node t.

Classification Trees (cont.)

Determining Class Assignment Rules (cont'd)

$$
\begin{aligned}
p(j^*|t) &= p(j^*, t_L|t) + p(j^*, t_R|t) \\
&= p(j^*|t_L)p(t_L|t) + p(j^*|t_R)p(t_R|t) \\
&= p_L p(j^*|t_L) + p_R p(j^*|t_R) \\
&\leq p_L \max_j p(j|t_L) + p_R \max_j p(j|t_R)
\end{aligned}
$$

$$
\begin{aligned}
r(|t) &= 1 - p(j^*|t) \\
&\geq 1 - \left[p_L \max_j p(j|t_L) + p_R \max_j p(j|t_R) \right] \\
&= p_L(1 - \max_j p(j|t_L)) + p_R(1 - \max_j p(j|t_R)) \\
&= p_L r(t_L) + p_R r(t_R)
\end{aligned}
$$

$$
\begin{aligned}
R(t) &= p(t)r(t) \\
&\geq p(t)p_L r(t_L) + p(t)p_R r(t_R) \\
&= p(t_L)r(t_L) + p(t_R)r(t_R) \\
&= R(t_L) + R(t_R)
\end{aligned}
$$

Classification Trees (cont.)

Variable Combinations

- So far, we have assumed that the classification tree only partitions the space by hyperplanes parallel to the coordinate planes.

- In the two-dimensional case, we only divide the space either by horizontal or vertical lines.

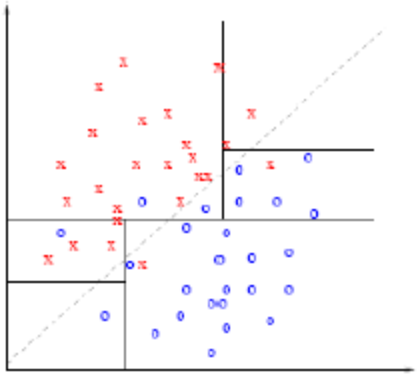
- How much do we suffer by such restrictive partitions?

- Let's look at an example…

Classification Trees (cont.)

Variable Combinations (cont'd)

- In this example, we might want to make a split using the dotted diagonal line which separates the two classes well.



- Splits parallel to the coordinate axes seem inefficient for this data set.

- Many steps of splits are needed to approximate the result generated by one split using a sloped line.

Classification Trees (cont.)

Variable Combinations (cont'd)

- There are classification tree extensions which, instead of thresholding individual variables, perform LDA for every node.

- Or we could use more complicated questions that use linear combinations of variables.

- This would increase the amount of computation significantly.

Classification Trees (cont.)

Variable Combinations (cont'd)

- Research seems to suggest that using more flexible questions often does not lead to obviously better classification result, if not worse.

- Over fitting is more likely to occur with more flexible splitting questions.

- It seems that using the right sized tree is more important than performing good splits at individual nodes.

Classification Trees (cont.)

Missing Values

- We may have missing values for some variables in some training sample points.

- For instance, gene expression microarray data often have missing gene measurements.

- Suppose each variable has 5% chance of being missing independently. Then for a training data point with 50 variables, the probability of missing some variables is as high as 92.3%!

Classification Trees (cont.)

Missing Values (cont'd)

- This means that at least 90% of the data will have at least one missing value!
- Thus, we cannot simply throw away data points whenever missing values occur.
- A test point to be classified may also having missing variables.
- Classification trees have a nice way of handling missing values by *surrogate splits*.

Classification Trees (cont.)

Missing Values (cont'd)

- Suppose the best split for node t is s which involves a question on X_m.
- Then think about what to do if this variable is not there.
- Classification trees tackle the issue by finding a replacement split.
- To find another split based on another variable, classification trees look at all the splits using all the other variables and search for the one yielding a division of training data points most similar to the optimal split.

Classification Trees (cont.)

Missing Values (cont'd)

- Along the same line of thought, the second best surrogate split could be found in case both the best variable and its top surrogate variable are missing, so on so forth.

- One thing to notice is that to find the surrogate split, classification trees do not try to find the second best split in terms of goodness measure.

- Instead, they try to approximate the result of the best split.

Classification Trees (cont.)

Missing Values (cont'd)

- Here, the goal is to divide data as similarly as possible to the best split so that it is meaningful to carry out the future decisions down the tree, which descend from the best split.

- There is no guarantee the second best split divides data similarly as the best split, although their goodness measurements are close.

Classification Trees (cont.)

Right Sized Tree via Pruning

- To prevent overfitting, we can stop growing the tree, but the stopping criteria tends to be myopic.
- The standard approach is therefore to grow a "full" tree and then perform pruning.
- Let the expect misclassification rate of a tree T be $R^*(T)$; recall that we used the resubstitution estimate: $$R(T) = \sum_{t \in \tilde{T}} r(t)p(t) = \sum_{t \in \tilde{T}} R(t)$$
- Remember also that $r(t)$ is the probability of making a wrong classification for points in node t.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- For a point in a given leaf node t, the estimated probability of misclassification is 1 minus the probability of the majority class in node t based on the training data.

- To get the probability of misclassification for the whole tree, a weighted sum of the *within leaf node error rate* is computed according to the total probability formula.

- We also previously mentioned that the resubstitution error rate $R(T)$ is biased downward.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Specifically, we proved the weighted misclassification rate for the parent node is guaranteed to be greater or equal to the sum of the weighted misclassification rates of the left and right child nodes.

- This means that if we simply minimize the resubstitution error rate, we would always prefer a bigger tree. There is no defense against overfitting.

- First we would grow the tree to a large size. Denote this maximum size by T_{max}.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Stopping criterion is not important here because as long as the tree is fairly big, it doesn't really matter when to stop.

- The over grown tree will be pruned back eventually. There are a few ways of deciding when to stop:
  - *Keep going until all terminal nodes are pure (contain only one class).*
  - *Keep going until the number of data in each terminal node is no greater than a certain threshold, say 5, or even 1.*
  - *As long as the tree is sufficiently large, the size of the initial tree is not critical.*

- The key here is to make the initial tree sufficiently big before pruning back!

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Now we need to introduce some notation…

- **Descendant:** a node *t'* is a descendant node *t* if there is a connected path down the tree leading from *t* to *t'*.

- **Ancestor:** *t* is an ancestor of *t'* if *t'* is its descendant.

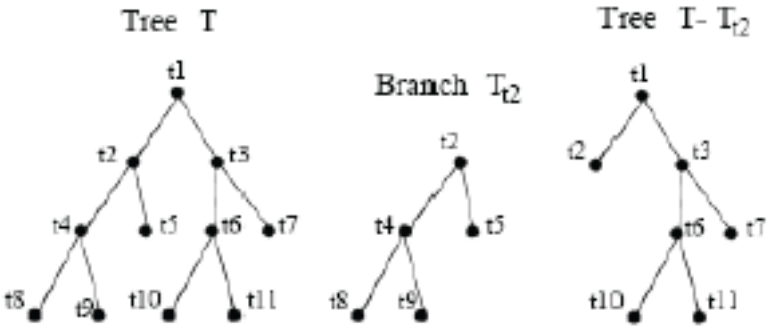
- A **branch** T_t of T with root node $t \in T$ consists of the node *t* and all descendants of *t* in *T*.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- **Pruning** a branch T_t from a tree T consists of deleting from T all descendants of t, that is, cutting off all of T_t except its root node. The tree pruned this way will be denoted by $T - T_t$.

- If T' is gotten from T by successively pruning off branches, then T' is called a **pruned subtree** of T and denoted by $T' < T$.



Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Even for a moderate sized T_{max}, there is an enormously large number of subtrees and an even larger number ways to prune the initial tree to get any.

- We therefore cannot exhaustively go through all the subtrees to find out which one is the best in some sense.

- Moreover, we typically do not have a separate test data set to serve as a basis for selection.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- A smarter method is necessary. A feasible method of pruning should ensure the following:
  - The subtree is optimal in a certain sense.
  - The search of the optimal subtree should be computationally tractable.

- As we just discussed, $R(T)$, is not a good measure for selecting a subtree because it always favors bigger trees.

- We need to add a complexity penalty to this resubstitution error rate. The penalty term favors smaller trees, and hence balances with $R(T)$.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- For any subtree $T < T_{max}$, we will define its complexity as $|\tilde{T}|$, the number of terminal or leaf nodes in T.

- Let $\alpha \geq 0$ be a real number called the **complexity parameter** and define the cost-complexity measure $R_\alpha(T)$ as:

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

- The more leaf nodes that the tree contains the higher complexity of the tree because we have more flexibility in partitioning the space into smaller pieces, and therefore more possibilities for fitting the training data.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- There's also the issue of how much importance to put on the size of the tree.
- The complexity parameter α adjusts that.
- At the end, the cost complexity measure comes as a penalized version of the resubstitution error rate.
- This is the function to be minimized when pruning the tree.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Which subtree is selected eventually depends on α. If α = 0, then the biggest tree will be chosen because the complexity penalty term is essentially dropped.

- As α approaches infinity, the tree of size 1, i.e., a single root node, will be selected.

- Given a pre-selected α, find the subtree $T(\alpha)$ that minimizes $R_\alpha(T)$:

$$R_\alpha(T(\alpha)) = \min_{T \preceq T_{max}} R_\alpha(T)$$

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- The minimizing subtree for any α always exists since there are only finitely many subtrees.

- Since there are at most a finite number of subtrees of T_{max}, $R_\alpha(T(\alpha))$ yields different values for only finitely many α's.

- $T(\alpha)$ continues to be the minimizing tree when α increases until a jump point is reached.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Two questions:
  - Is there a unique subtree $T < T_{max}$ which minimizes $R_\alpha(T)$?
  - In the minimizing sequence of trees T_1, T_2, ... is each subtree obtained by pruning upward from the previous subtree?

- If the optimal subtrees are nested, the computation will be a lot easier.

- We can first find T_1, and then to find T_2, we don't need to start again from the maximum tree, but from T_1, (because T_2 is guaranteed to be a subtree of T_1).

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- In this way when α increases, we prune based on a smaller and smaller subtree.

- The smallest minimizing subtree $T(\alpha)$ for complexity parameter α is defined by the conditions:
  - $R_\alpha(T(\alpha)) = \min_{T \leq T_{max}} R_\alpha(T)$
  - If $R_\alpha(T) = R_\alpha(T(\alpha))$, then $T(\alpha) \leq$ T. If another tree achieves the minimum at the same α, then the other tree is guaranteed to be bigger than the smallest minimized tree $T(\alpha)$.

- By definition, if the smallest minimizing subtree $T(\alpha)$ exists, it must be unique.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Earlier we argued that a minimizing subtree always exists because there are only a finite number of subtrees.

- Here we go one step more. We can prove that the smallest minimizing subtree always exists.

- This is not trivial to show because one tree smaller than another means the former is embedded in the latter.

- Tree ordering is a partial ordering.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- The starting point for the pruning is not T_{max}, but rather $T_1 = T(0)$, which is the smallest subtree of T_{max} satisfying: $R(T_1) = R(T_{max})$.

- The way that you get T_1 is as follows:

- First, look at the biggest tree, T_{max}, and for any two terminal nodes descended from the same parent, for instance t_L and t_R, if they yield the same re-substitution error rate as the parent node t, prune off these two terminal nodes.
  - That is, if $R(t) = R(t_L) + R(t_R)$, prune off t_L and t_R.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- This process is applied *recursively*. After we have pruned one pair of terminal nodes, the tree shrinks a little bit.

- Then based on the smaller tree, we do the same thing until we cannot find any pair of terminal nodes satisfying this inequality. The resulting tree at this point is T_1.

- We will use T_t to denote a branch rooted at t. Then, for T_t, we define $R(T_t)$, by:

$$R(T_t) = \sum_{t' \in \tilde{T}_t} R(t')$$

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- What we can prove is that, if t is any non-terminal or internal node of T_1, then it is guaranteed to have a smaller re-substitution error rate than the re-substitution error rate of the branch.

- If we prune off the branch at t, the resubstitution error rate will strictly increase.

- The **weakest link cutting method** not only finds the next α which results in a different optimal subtree, but find that optimal subtree.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Remember, we previously defined R_α for the entire tree. Here, we extend the definition to a node and then for a single branch coming out of a node.

- For any node $t \in T_1$, we can set $R_\alpha(\{t\}) = R(t) + \alpha$.

- Also, for any branch T_t, we can define $R_\alpha(T_t) = R(T_t) + \alpha|\tilde{T}|$.

- We know that when $\alpha = 0$, $R_0(T_t) < R_0(\{t\})$. The inequality holds for sufficiently small α.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- If we gradually increase α, because $R_\alpha(T_t)$ increases faster with α (the coefficient in front of α is larger than that in $R\alpha(\{t\})$), at a certain α (α_1 below), we will have $R_\alpha(T_t) = R_\alpha(\{t\})$.

- If α further increases, the inequality sign will be reversed, and we have $R_\alpha(T_t) > R_\alpha(\{t\})$. Some node t may reach the equality earlier than some other.

- The node that achieves the equality at the smallest α is called the **weakest link**.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- It is possible that several nodes achieve the equality at the same time, and hence there are several weakest link nodes.

- Solve the inequality $R_\alpha(T_t) < R_\alpha(\{t\})$ and get: $\alpha < \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}$

- The right hand side is the ratio between the difference in resubstitution error rates and the difference in complexity, which is positive because both the numerator and the denominator are positive.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Define a function $g_1(t)$, $t \in T_1$ by:

$$g_1(t)\begin{cases} \dfrac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}, & t \notin \tilde{T}_1 \\ +\infty, & t \in \tilde{T}_1 \end{cases}$$

- The weakest link $\bar{t}_1$ in T_1 achieves the minimum of $g_1(t)$:

$$g_1(\bar{t}_1) = \min_{t \in T_1} g_1(t)$$

- and put $\alpha_2 = g_1(\bar{t}_1)$. To get the optimal subtree corresponding to α_2, simply remove the branch growing out of $\bar{t}_1$.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- When α increases, $\bar{t}_1$ is the first node that becomes more preferable than the branch $T_{\bar{t}_1}$ descended from it.

- If there are several nodes that simultaneously achieve the minimum $g_1(t)$, we remove the branch grown out of each of these nodes.

- α_2 is the first value after $\alpha_1 = 0$ that yields an optimal subtree strictly smaller than T_1.

- For all $\alpha_1 \leq \alpha < \alpha_2$, the smallest minimizing subtree is the same as T_1.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Let $T_2 = T_1 - T_{\bar{t}_1}$. Repeat the previous steps.

- Use T_2 instead of T_1 as the starting tree, find the weakest link in T_2 and prune off at all the weakest link nodes to get the next optimal subtree.

$$g_2(t) = \begin{cases} \dfrac{R(t) - R(T_{2t})}{|\tilde{T}_{2t}| - 1}, & t \in T_2, t \notin \tilde{T}_2 \\ +\infty, & t \in \tilde{T}_2 \end{cases}$$

$$g_2(\bar{t}_2) = \min_{t \in T_2} g_2(t)$$

$$\alpha_3 = g_2(\bar{t}_2)$$

$$T_3 = T_2 - T_{\bar{t}_2}$$

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- In terms of computation, we need to store a few values at each node:
  - $R(t)$, the resubstitution error rate for node t. This only need to be computed once.
  - $R(T_t)$, the resubstitution error rate for the branch coming out of node t. This may need to be updated after pruning because T_t may change after pruning.
  - $|T_t|$, the number of leaf nodes on the branch coming out of node t. This may need to be updated after pruning.
- In order to compute the resubstitution error rate $R(t)$, we need the proportion of data points in each class that land in node t.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Let's suppose we compute the class priors by the proportion of points in each class.
- As we grow the tree, we store the number of points that land in node t, as well as the number of points in each class that land in node t.
- Given those numbers, we can easily estimate the probability of node t and the class posterior given a data point is in node t.
- $R(t)$ can then be calculated.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- As far as calculating the next two numbers, a) the resubstitution error rate for the branch coming out of node *t*, and b) the number of leaf nodes that are on the branch coming out of node *t*, these two numbers change after pruning.

- After pruning, we to need to update these values because the number of leaf nodes will have been reduced. To be specific, we would need to update the values for all of the ancestor nodes of the branch.

- A recursive procedure can be used to compute $R(T_t)$ and $|T_t|$.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- One approach for choosing the best pruned subtree is using a *test sample set*.
- If we have a large test data set, we can compute the error rate using the test data set for all the subtrees and see which one achieves the minimum error rate.
- However, in practice, we rarely have a large test data set. Even if we have a large test data set, instead of using the data for testing, we might rather use this data for training in order to train a better tree.
- When data is scarce, we may not want to use too much for testing.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- Another approach for choosing the best pruned subtree is using *cross-validation*.

- How do we conduct cross-validation for trees when trees are unstable?

- If the training data vary a little bit, the resulting tree may be very different.

- Therefore, we would have difficulty to match the trees obtained in each fold with the tree obtained using the entire data set.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

- However, although we said that the trees themselves can be unstable, this does not mean that the classifier resulting from the tree is unstable.

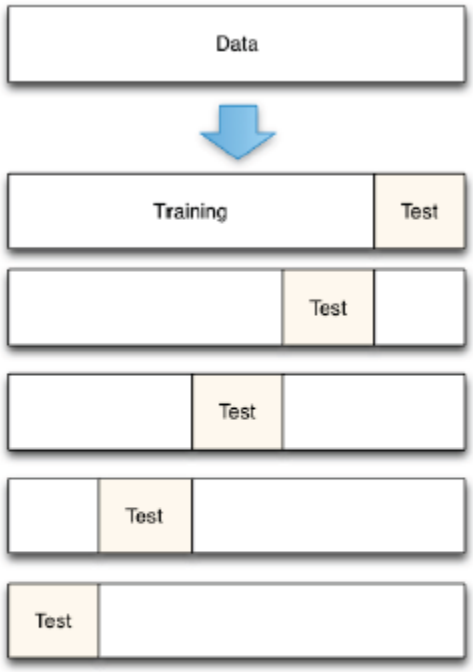
- We may end up with two trees that look very different, but make similar decisions for classification. The key strategy in a classification tree is to focus on choosing the right complexity parameter α.

- Instead of trying to say which tree is best, a classification tree tries to find the best complexity parameter α.

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

1. Use k-fold cross-validation on the *training data*.
2. In each validation, $(k-1)/k$ of the training data is used as training, $1/k$ of training data is used as test.
   1. Tree starts with one (terminal) node ($m = 1$)
   2. Train a tree with m terminal nodes
   3. Test it and get the error rate
   4. Grow the tree with one more split ($m \leftarrow m + 1$)
   5. If the tree is not full yet, go to 2
3. Pick m^* terminal nodes which gives us the lowest error.

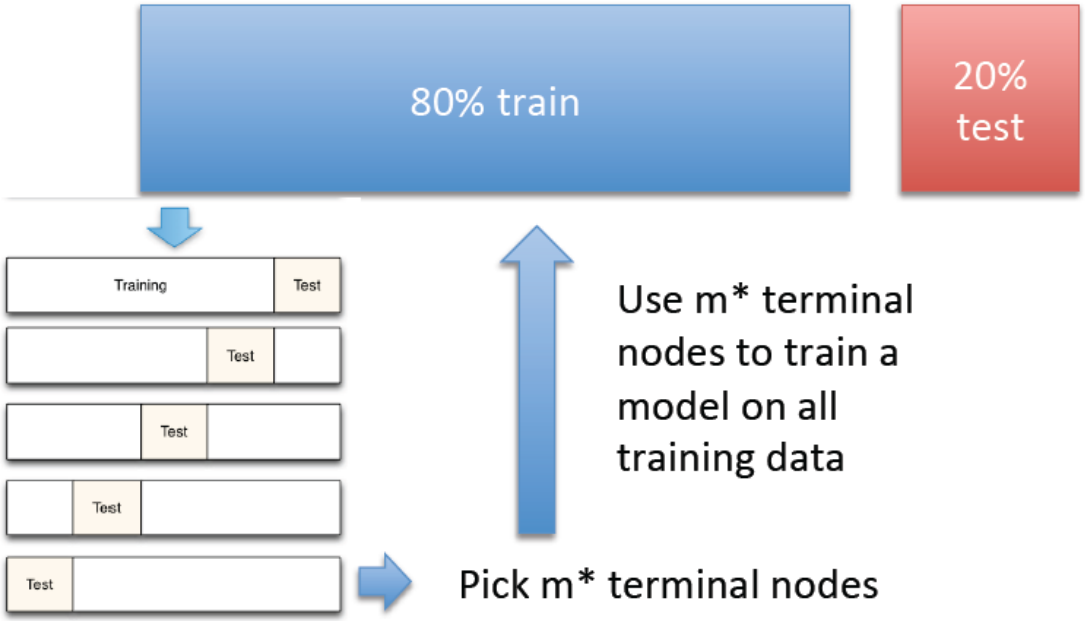


5-fold cross validation

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)

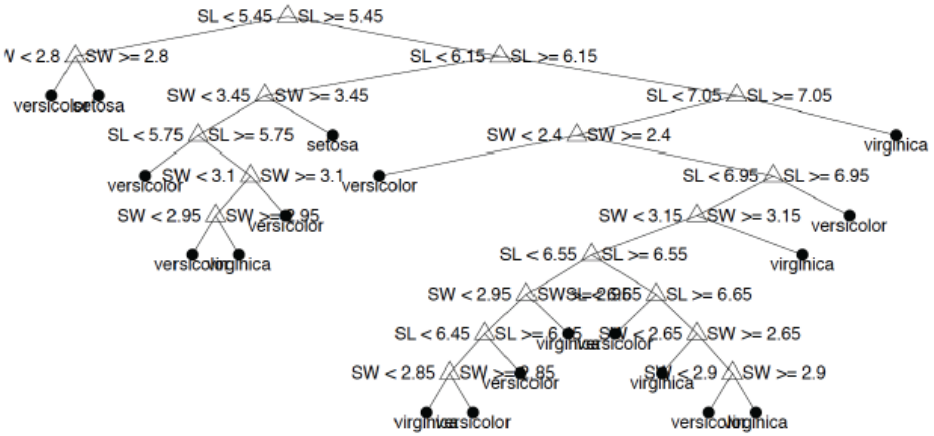
Randomly pick 80% as train, 20% as test
(if using 5-fold cross validation, need to do this process 5 times)

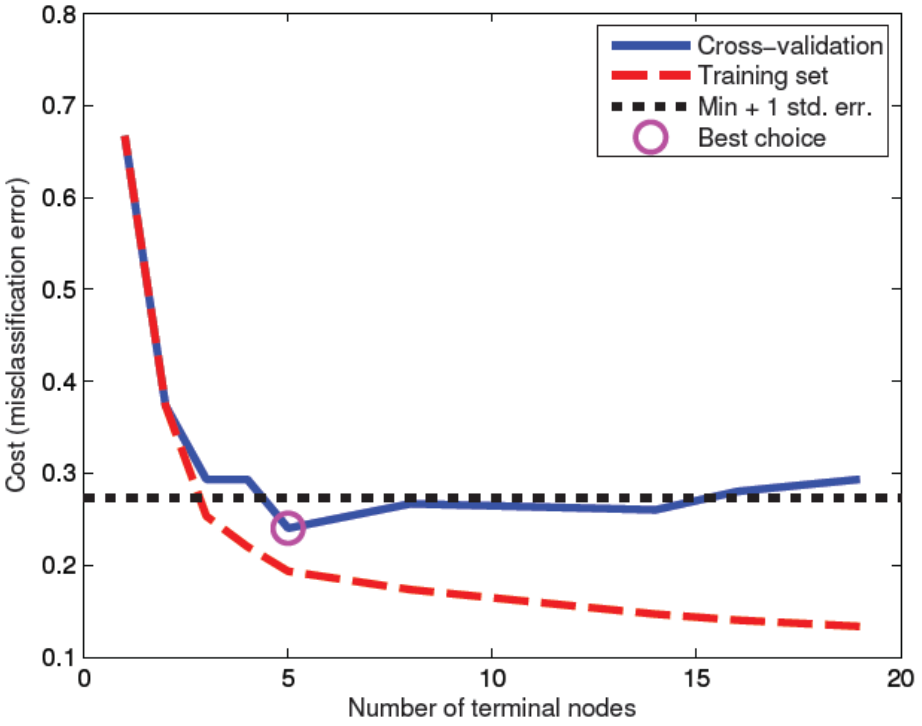


Note: the cross-validation to prune the tree should only use the training data

Classification Trees (cont.)

Right Sized Tree via Pruning (cont'd)





Classification Trees (cont.)

Advantages of the Tree-Structured Approach

- As previously mentioned, the tree structured approach handles both categorical and ordered variables in a simple and natural way.
- Classification trees sometimes do an automatic step-wise variable selection and complexity reduction.
- They provide an estimate of the misclassification rate for a test point.
- For every data point, we know which leaf node it lands in and we have an estimation for the posterior probabilities of classes for every leaf node.

Classification Trees (cont.)

Advantages of the Tree-Structured Approach (cont'd)

- The misclassification rate can be estimated using the estimated class posterior.
- Classification trees are invariant under all monotone transformations of individual ordered variables.
- The reason is that classification trees split nodes by thresholding.
- Monotone transformations cannot change the possible ways of dividing data points by thresholding.

Classification Trees (cont.)

Advantages of the Tree-Structured Approach (cont'd)

- Classification trees are also relatively robust to outliers and misclassified points in the training set.

- They do not calculate any average or anything else from the data points themselves.

- Classification trees are easy to interpret, which is appealing especially in medical applications.

Classification Trees (cont.)

Disadvantages of the Tree-Structured Approach (cont'd)

- Do not predict very accurately compared to other kinds of machine learning models, which is due to the greedy nature of the tree construction algorithm.

- Trees are unstable
  - Small changes to the input data can have large effects on the structure of the tree due to the hierarchical nature of the tree-growing process.
  - Solution: bagging and boosting.

Model Averaging

- Previously, we focused on machine learning procedures that produce a single set of results:
  - A regression equation, with one set of regression coefficients or smoothing parameters.
  - A classification regression tree with one set of leaf nodes.

- Model selection is often required: a measure of fit associated with each candidate model.

- Here, the discussion shifts to machine learning building on many sets of outputs that are **aggregated** to produce results.

Model Averaging (cont'd)

- The aggregating procedure makes a number of passes over the data.
- On each pass, inputs X are linked with outputs Y just as before.
- However, of interest now is the collection of all the results from all passes over the data.
- Aggregated results have several important benefits.

Model Averaging (cont'd)

- Averaging over a collection of fitted values can help to avoid overfitting.
- It tends to cancel out the uncommon features of the data captured by a specific model. Therefore, the aggregated results are more stable.
- A large number of fitting attempts can produce very flexible fitting functions.
- Putting the averaging and the flexible fitting functions together has the potential to break the bias-variance tradeoff.

Model Averaging (cont'd)

- Any attempt to summarize patterns in a dataset risks overfitting.
- All fitting procedures adapt to the data on hand, so that even if the results are applied to a new sample from the same population, fit quality will likely decline.
- Hence, generalization can be somewhat risky.
- Optimism increases linearly with the number of inputs or basis functions, but decreases as the training sample size increases.

Model Averaging (cont'd)

- Consider decision trees as a key illustration.
- The overfitting often increases with:
  - the number of possible splits for a given predictor
  - the number of candidate predictors
  - the number of stages which is typically represented by the number of leaf nodes
- When overfitting occurs in a classification tree, the classification error is underestimated; the model may have a structure that will not generalize well.
- For example, one or more predictors may be included in a tree that really do not belong.

Model Averaging (cont'd)

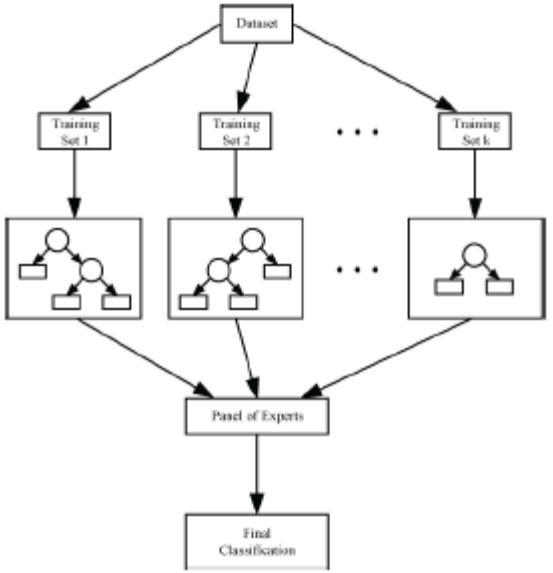
- Ideally, one would have two random samples from the same population: a training dataset and a test dataset. The fit measure from the test data would be a better indicator of how accurate the classification is.

- Often there is only a single dataset. The data are split up into several randomly chosen, non-overlapping, partitions of about the same size. With ten partitions, each would be a part of the training data in nine analyses, and serve as the test data in one analysis.

- The model selection is based on choosing the one with smallest cross-validation prediction error.

Bagging

- There is a very powerful idea in the use of subsamples of the data and in averaging over subsamples through bootstrapping.
- Bagging exploits that idea to address the overfitting issue in a more fundamental manner.
- It was invented by Leo Breiman, who called it "**b**ootstrap **agg**regat**ing**" or simply "bagging.“
- In a classification tree, bagging takes a majority vote from classifiers trained on bootstrap samples of the training data.



Bagging (cont'd)

- **Algorithm:** Consider the following steps in a fitting algorithm with a dataset having *N* observations and a binary response variable:
  1. Take a random sample of size *N with replacement* from the data (a bootstrap sample).
  2. Construct a "full" classification tree as usual but do not prune.
  3. Assign a class to each terminal node, and store the class attached to each case coupled with the predictor values for each observation.
  4. Repeat Steps 1-3 a large number of times.
  5. For each observation in the dataset, count the number of trees that it is classified in one category over the number of trees.
  6. Assign each observation to a final category by a majority vote over the set of trees. Thus, if 51% of the time over a large number of trees a given observation is classified as a "1", that becomes its classification.

Bagging (cont'd)

- Although there remain some important variations and details to consider, these are the key steps to producing "bagged" classification trees.

- The idea of classifying by averaging over the results from a large number of bootstrap samples generalizes easily to a wide variety of classifiers beyond classification trees.

- Bagging introduces a new concept, "margins." Operationally, the "margin" is the difference between the proportion of times a case is correctly classified and the proportion of times it is incorrectly classified.

Bagging (cont'd)

- *Large margins are desirable* because a more stable classification is implied.
- Ideally, there should be large margins for all of the observations. This bodes well for generalization to new data.
- For each tree, observations not included in the bootstrap sample are called "out-of-bag" observations.
- These "out-of-bag" observations can be treated as a test dataset, and dropped down the tree.

Bagging (cont'd)

- To get a better evaluation of the model, the prediction error is estimated only based on the "out-of-bag" observations.
- In other words, the averaging for a given observation is done only using the trees for which that observation was not used in the fitting process.
- The core of bagging's potential is found in the averaging over results from a substantial number of bootstrap samples.
- As a first approximation, the averaging helps to cancel out the impact of random variation.

Bagging (cont'd)

- Bagging can help with the variance.
- The conceptual advantage of bagging is to aggregate fitted values from a large number of bootstrap samples.
- Ideally, many sets of fitted values, each with low bias but high variance, may be averaged in a manner than can effectively reduce the bite of the bias-variance tradeoff.
- The ways in which bagging aggregates the fitted values is the basis for many other machine learning developments.

Bagging (cont'd)

- Recall that a regression tree maximizes the reduction in the error sum of squares at each split.
- All of the concerns about overfitting apply, especially given the potential impact that outliers can have on the fitting process when the response variable is quantitative.
- Bagging works by the same general principles when the response variable is numerical:
  - For each tree, each observation is placed in a terminal node and assigned the mean of that terminal node.
  - Then, the average of these assigned means over trees is computed for each observation.
  - This average value for each observation is the bagged fitted value.

Random Forests

- In bagging, simply re-running the same machine learning algorithm on different subsets of the data can result in highly correlated predictors, which makes the model biased.

- On the other hand, random forests try to decorrelate the base learners by learning trees based on a randomly chosen subset of input variables, as well as a randomly chosen subset of data samples.

- Such models often have *very good predictive accuracy* and have been *widely used* in many applications.

Random Forests (cont.)

- Bagging constructs a large number of trees with bootstrap samples from a dataset.
- But now, as each tree is constructed, take a random sample of predictors before each node is split.
- For example, if there are twenty predictors, choose a random five as candidates for constructing the best split.
- Repeat this process for each node until the tree is large enough. And as in bagging, do not prune.

Random Forests (cont'd)

- **Algorithm:** The random forests algorithm is very much like the bagging algorithm. Let *N* be the number of observations and assume for now that the response variable is binary:
  1. Take a random sample of size *N* with replacement from the data (bootstrap sample).
  2. Take a random sample without replacement of the predictors.
  3. Construct a split by using predictors selected in Step 2.
  4. Repeat Steps 2 and 3 for each subsequent split until the tree is as large as desired. Do not prune. Each tree is produced from a random sample of cases, and at each split a random sample of predictors.
  5. Drop the out-of-bag data down the tree. Store the class assigned to each observation along with each observation's predictor values.
  6. Repeat Steps 1-5 a large number of times (e.g., 500).
  7. For each observation in the dataset, count the number of trees that it is classified in one category over the number of trees.
  8. Assign each observation to a final category by a majority vote over the set of trees. Thus, if 51% of the time over a large number of trees a given observation is classified as a "1", that becomes its classification.

Random Forests (cont'd)

Why they work?

- Variance reduction: the trees are more independent because of the combination of bootstrap samples and random draws of predictors.

- It is apparent that random forests are a form of bagging, and the averaging over trees can substantially reduce instability that might otherwise result. Moreover, by working with a random sample of predictors at each possible split, the fitted values across trees are more independent.

- Consequently, the gains from averaging over a large number of trees (variance reduction) can be more dramatic.

Random Forests (cont'd)

Why they work?

- Bias reduction: a very large number of predictors can be considered, and local feature predictors can play a role in the tree construction.
- Random forests are able to work with a very large number of predictors, even more predictors than there are observations.
- An obvious gain with random forests is that more information may be brought to reduce bias of fitted values and estimated splits.
- There are often a few predictors that dominate the decision tree fitting process because on the average they consistently perform just a bit better than their competitors.

Random Forests (cont'd)

Why they work?

- Consequently, many other predictors, which could be useful for very local features of the data, are rarely selected as splitting variables.
- With random forests computed for a large enough number of trees, each predictor will have at least several opportunities to be the predictor defining a split.
- In those opportunities, it will have very few competitors. Much of the time a dominant predictor will not be included.
- Therefore, local feature predictors will have the opportunity to define a split.

Random Forests (cont'd)

- Random forests are among the very best classifiers invented to date!

- Random forests include three main tuning parameters.
  - **Node Size:** unlike in decision trees, the number of observations in the terminal nodes of each tree of the forest can be very small. The goal is to grow trees with as little bias as possible.

  - **Number of Trees:** in practice, 500 trees is often a good choice.

  - **Number of Predictors Sampled:** the number of predictors sampled at each split would seem to be a key tuning parameter that should affect how well random forests perform. Sampling 2-5 each time is often adequate.

Random Forests (cont'd)

- With forecasting accuracy as a criterion, bagging is in principle an improvement over decision trees.
- It constructs a large number of trees with bootstrap samples from a dataset.
- Random forests is in principle an improvement over bagging.
- It draws a random sample of predictors to define each split.

Random Forests (cont'd)

- **Weighted Classification Votes:** After all of the trees are built, one can differentially weight the classification votes over trees.
  - For example, one vote for classification in the rare category might count the same as two votes for classification in the common category.

- **Stratified Bootstrap Sampling:** When each bootstrap sample is drawn before a tree is built, one can oversample one class of cases for which forecasting errors are relatively more costly.
  - The procedure is much in the same spirit as disproportional stratified sampling used for data collection.

Boosting

- Boosting, like bagging, is another general approach for improving prediction results for various machine learning methods.
- It is also particularly well suited to decision trees.
- Rather than resampling the data, however, boosting weights on some examples during learning.
- Specifically, in boosting the trees are grown *sequentially*: each tree is grown using information from previously grown trees.

Boosting (cont.)

General Boosting Algorithm for Regression Trees:

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   - 2.1 Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   - 2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   - 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Boosting (cont.)

General Boosting Algorithm for Regression Trees (cont'd):

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.

- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.

- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.

- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting (cont.)

Boosting for Classification

- Boosting iteratively learns weak classifiers; a weak classifier is one whose error rate is only slightly better than random guessing.

- The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers.

- The predictions from all of them are then combined through a weighted majority vote to produce the final prediction.

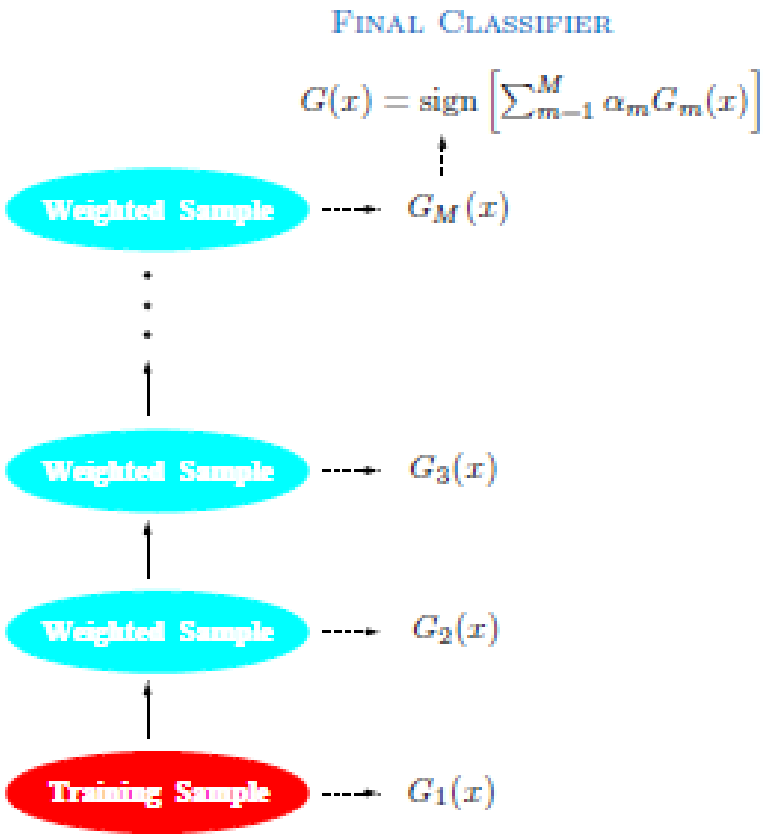
Boosting (cont.)

Boosting for Classification (cont'd)

- Thus, the final result is the weighted sum of the results of weak classifiers.

- The alphas in the final classifier are computed by the boosting algorithm, which weight the contribution to give higher influence to more accurate classifiers in the sequence.

- Weights are modified at each boosting step!



Boosting (cont.)

- **Up-weight data** that are difficult; incorrectly classified in the previous round. **Down-weight data** that are easy; correctly classified in the previous round.

- Here are some commonly used loss functions, gradients, population minimizers, and algorithms to minimize the loss:

Name	Loss	Derivative	f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$	$\mathbb{E}[y\|\mathbf{x}_i]$	L2Boosting
Absolute error	$\|y_i - f(\mathbf{x}_i)\|$	$\text{sgn}(y_i - f(\mathbf{x}_i))$	$\text{median}(y\|\mathbf{x}_i)$	Gradient boosting
Exponential loss	$\exp(-\tilde{y}_i f(\mathbf{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBoost

Boosting (cont.)

- There are many different boosting algorithms for classification:
  - AdaBoost, LPBoost, BrownBoost, LogitBoost, Gradient Boosting, etc.

- Here is a general boosting algorithm:

  - given training set $(x_1, y_1), \ldots, (x_m, y_m)$
  - $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
  - for $t = 1, \ldots, T$:
    - construct distribution D_t on $\{1, \ldots, m\}$
    - find weak classifier ("rule of thumb")

      $$h_t : X \to \{-1, +1\}$$

      with error ϵ_t on D_t:

      $$\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$
  - output final/combined classifier H_{final}

Boosting (cont.)

- **Example of Boosting Algorithm (AdaBoost):**

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Boosting (cont.)

- Boosting is remarkably resistant to overfitting, and it is fast and simple.
- In fact, it can often continue to improve even when the training error has gone to zero.
- It improves the performance of many kinds of machine learning algorithms.
- Boosting does not work when:
  - Not enough data, base learner is too weak or too strong, and/or susceptible to noisy data.

Summary

- Basic idea of decision trees.
- The three elements in the construction of a classification tree.
- The definition of the impurity function and several example functions.
- Estimating the posterior probabilities of classes in each tree node.
- Advantages of tree-structured classification methods.
- Resubstitution error rate and the cost-complexity measure, their differences, and why the cost-complexity measure is introduced.
- Weakest-link pruning.
- Purpose of model averaging.
- Bagging procedure.
- Random forest procedure.
- Boosting approach.