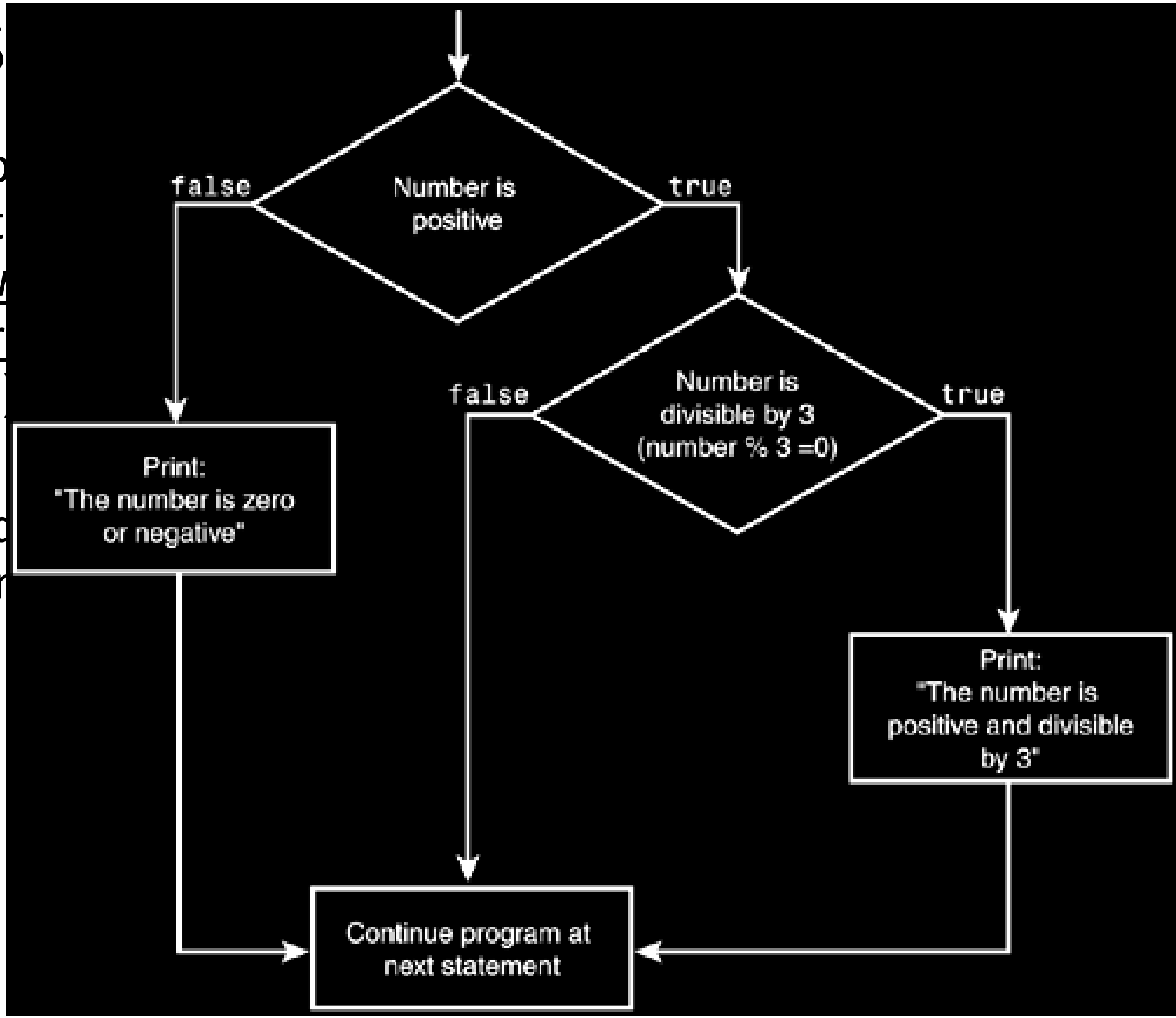
CLASSIFICATION METHODS:
- **CART**

CS5691- PRML

CLAS

- The p
  input
  maki
  binar
  node
- Here
  called
  (Brein



CLASSIFICATION AND REGRESSION TREES

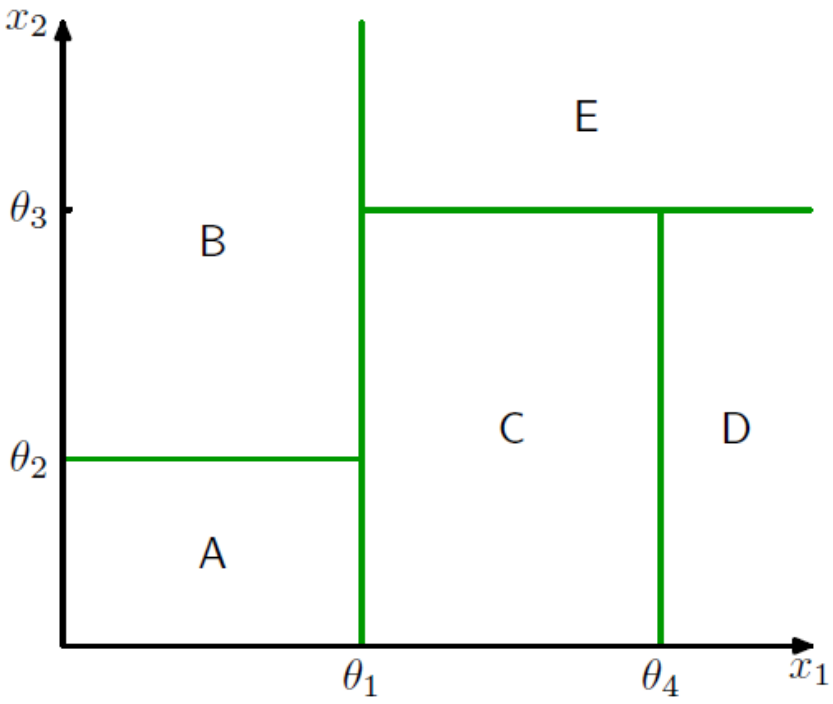
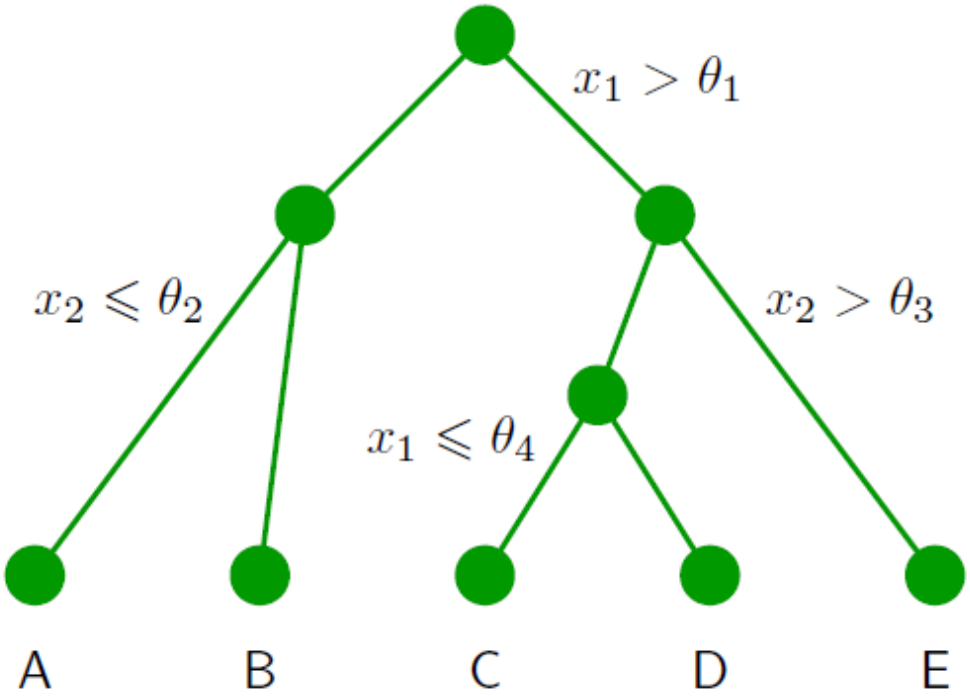
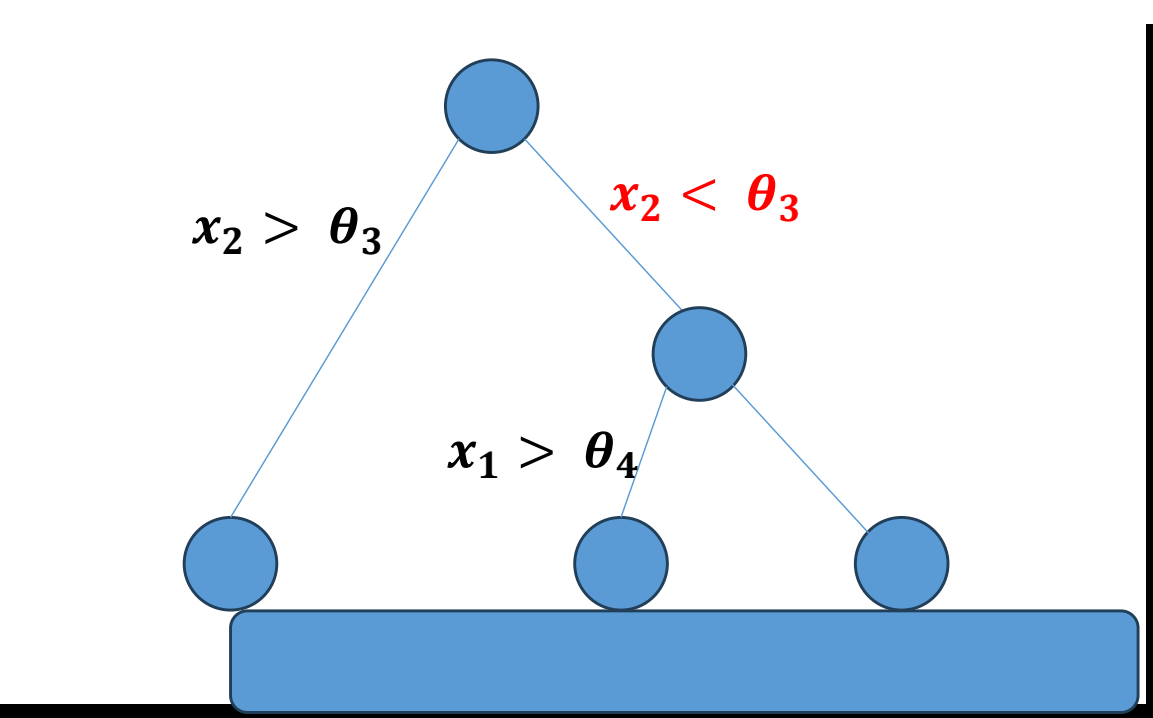
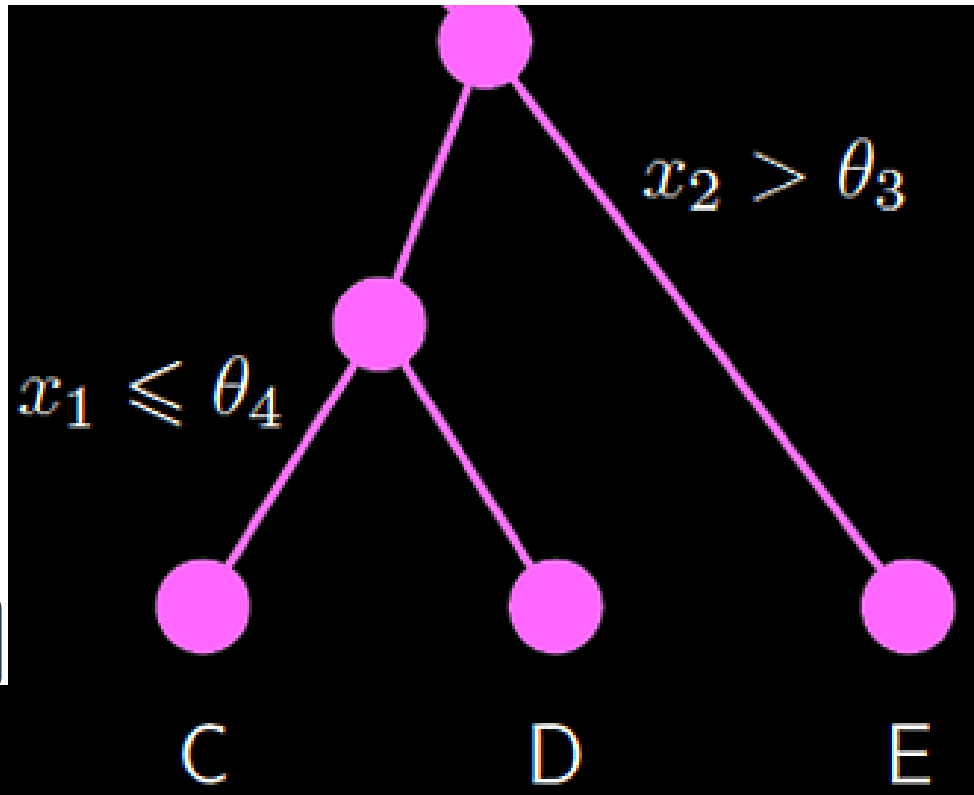


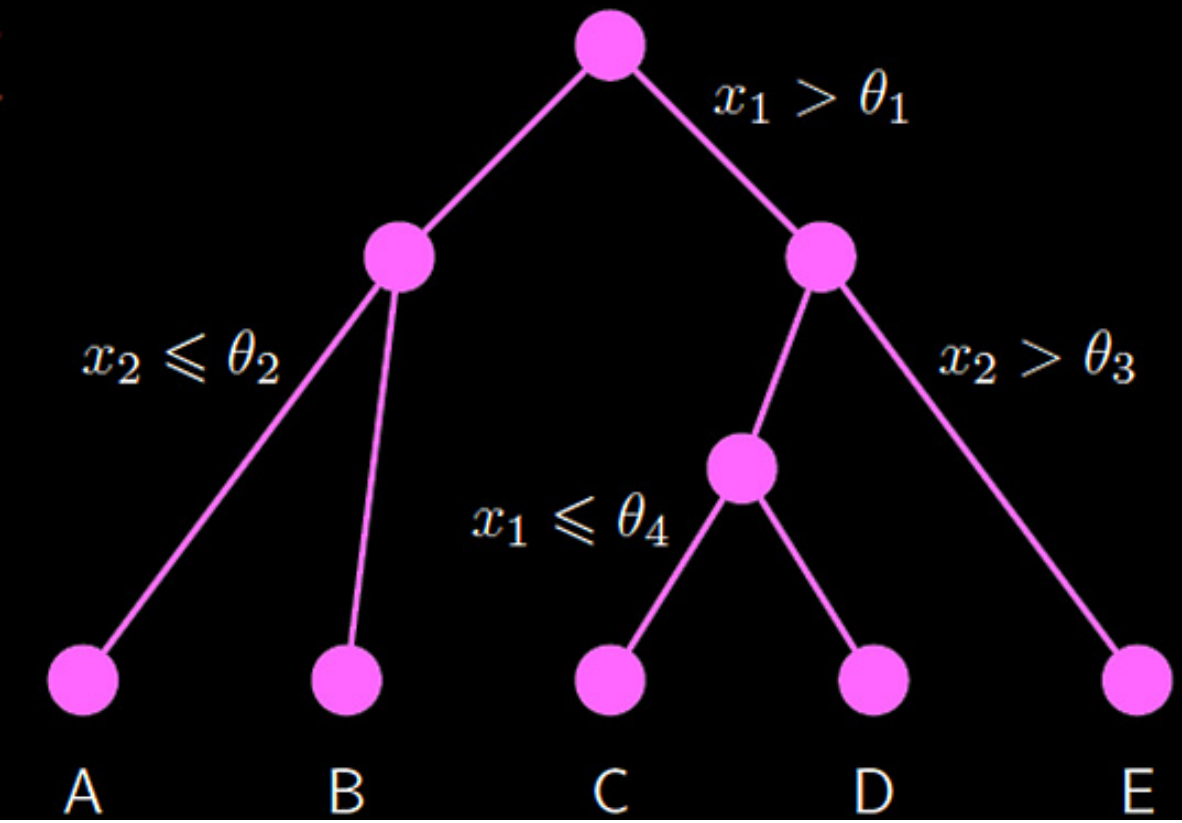
Illustration of a two-dimensional input space that has been partitioned into five regions using axis-aligned boundaries.

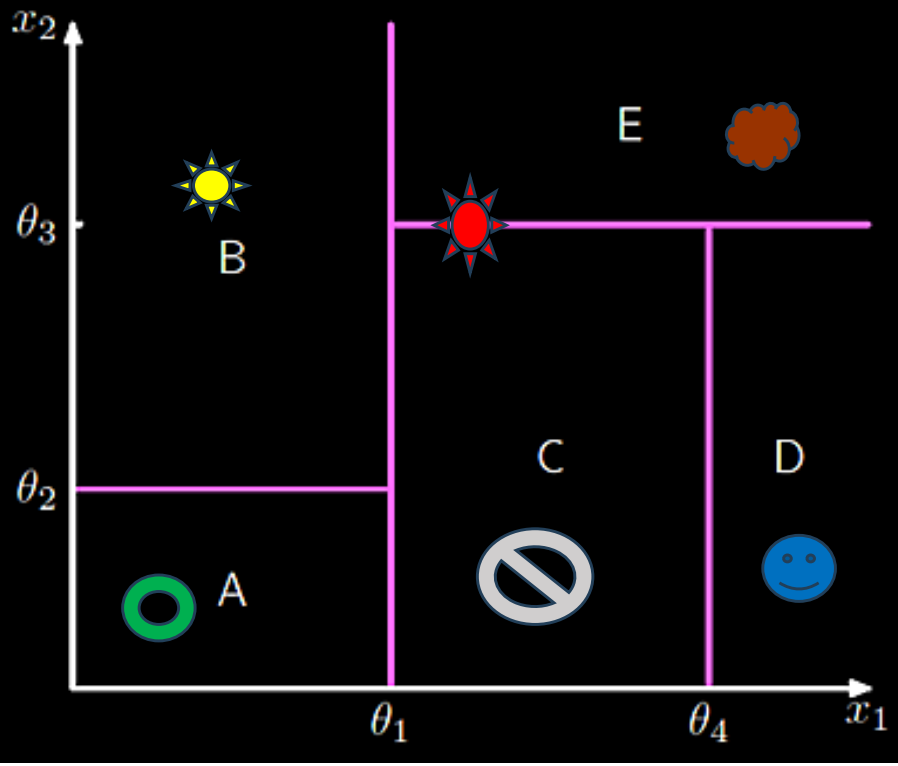


Binary tree corresponding to the partitioning of input space (eg BSP tree)


$x_2 > \theta_3$
$x_2 < \theta_3$
$x_1 > \theta_4$


$x_2 > \theta_3$
$x_1 \leqslant \theta_4$
C
D
E


$x_1 > \theta_1$
$x_2 \leqslant \theta_2$
$x_2 > \theta_3$
$x_1 \leqslant \theta_4$
A
B
C
D
E


x_2
θ_3
θ_2
E
B
C
D
A
θ_1
θ_4
x_1

CLASSIFICATION AND REGRESSION TREES

- In the example given in previous slide, the *first step divides the whole of the input space into two regions* according to whether $x_1 \leq \theta_1$ or $x_1 > \theta_1$ where θ_1 is a parameter of the model.
- This *creates two sub regions, each of which can then be subdivided independently*.
- For instance, the region $x_1 \leq \theta_1$ is further subdivided according to whether $x_2 \leq \theta_2$ or $x_2 > \theta_2$, giving rise to the regions denoted A and B.
- For any new input **x**, we determine which region it falls into by *starting at the top of the tree at the root node and following a path down to a specific leaf node according to the decision criteria at each node*.

CLASSIFICATION AND REGRESSION TREES

- Within each region, there is a separate model (function/ value) to predict the target variable.
- For instance, in *regression* we might simply predict a constant over each region, or in *classification* we might assign each region to a specific class ID.
- **EXAMPLE**: For instance, to predict a patient's disease, we might
  - first ask "is their temperature greater than some threshold?". If the answer is yes, then
  - we might next ask "is their blood pressure less than some threshold?".

Each leaf of the tree is then associated with a specific diagnosis.

Hastie Sec. 9.2 – Also Murphy Sec. 16.2

CLASSIFICATION AND REGRESSION TREES

- Consider first a *regression problem* in which the goal is to predict a single target variable t from a *D*-dimensional vector $\mathbf{x} = (x_1, \ldots, x_D)^T$ of input variables.
- The training data consists of input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ along with the corresponding continuous labels $\{t_1, \ldots, t_N\}$.
- If the partitioning of the input space is given, and we minimize the sum-of-squares error function, then the optimal value of the predictive variable within any given region is just given by the *average of the values of t_n for those data points that fall in that region*.

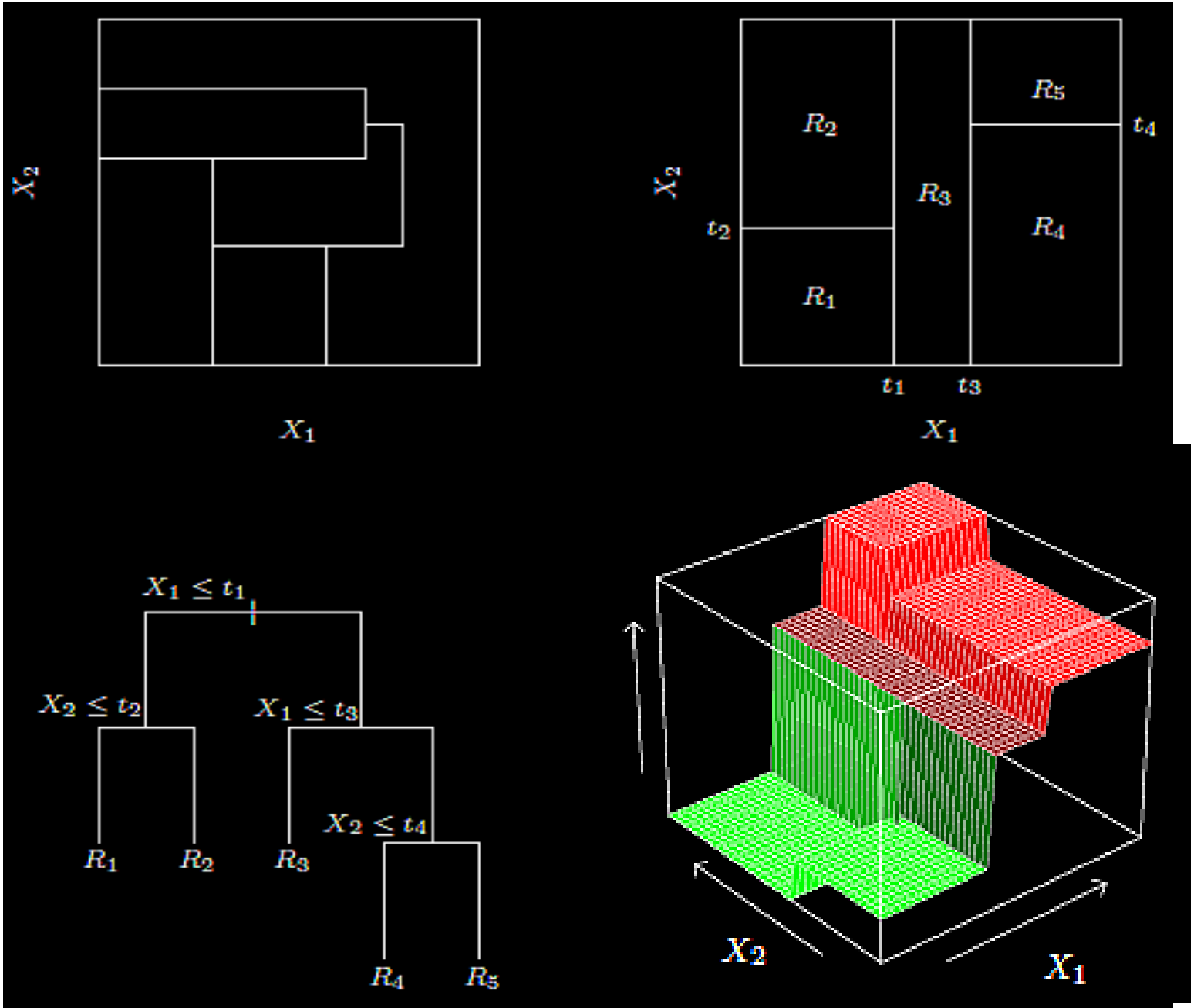


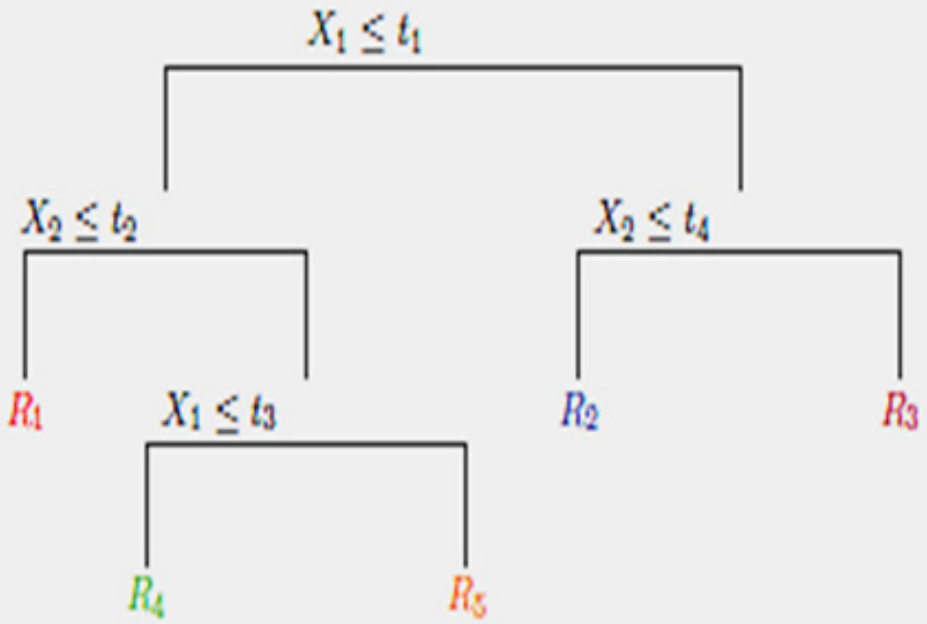
FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.*

Regression Trees – popular method for tree-based regression and classification called **CART**

We choose the variable and split-point to achieve the best fit. Then one or both of these regions are split into two more regions, and this process is continued, until some stopping rule is applied. For example, in the top right panel of Figure 9.2, we first split at $X_1 = t_1$. Then the region $X_1 \leq t_1$ is split at $X_2 = t_2$ and the region $X_1 > t_1$ is split at $X_1 = t_3$. Finally, the region $X_1 > t_3$ is split at $X_2 = t_4$. The result of this process is a partition into the five regions $R_1, R_2, \ldots, R_5$ shown in the figure. The corresponding regression model predicts Y with a constant c_m in region R_m, that is,

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}. \tag{9.9}$$

This same model can be represented by the binary tree in the bottom left panel of Figure 9.2. The full dataset sits at the top of the tree. Observations satisfying the condition at each junction are assigned to the left branch, and the others to the right branch. The terminal nodes or leaves of the tree correspond to the regions $R_1, R_2, \ldots, R_5$. The bottom right panel of Figure 9.2 is a perspective plot of the regression surface from this model.



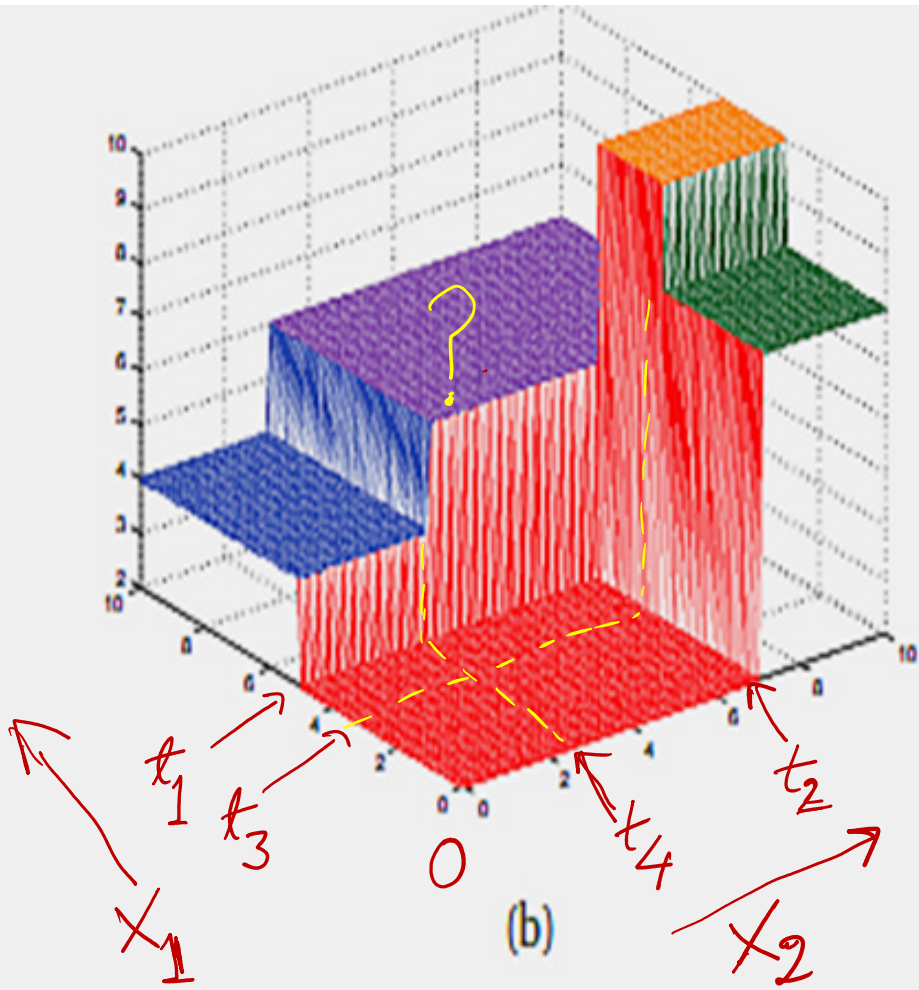
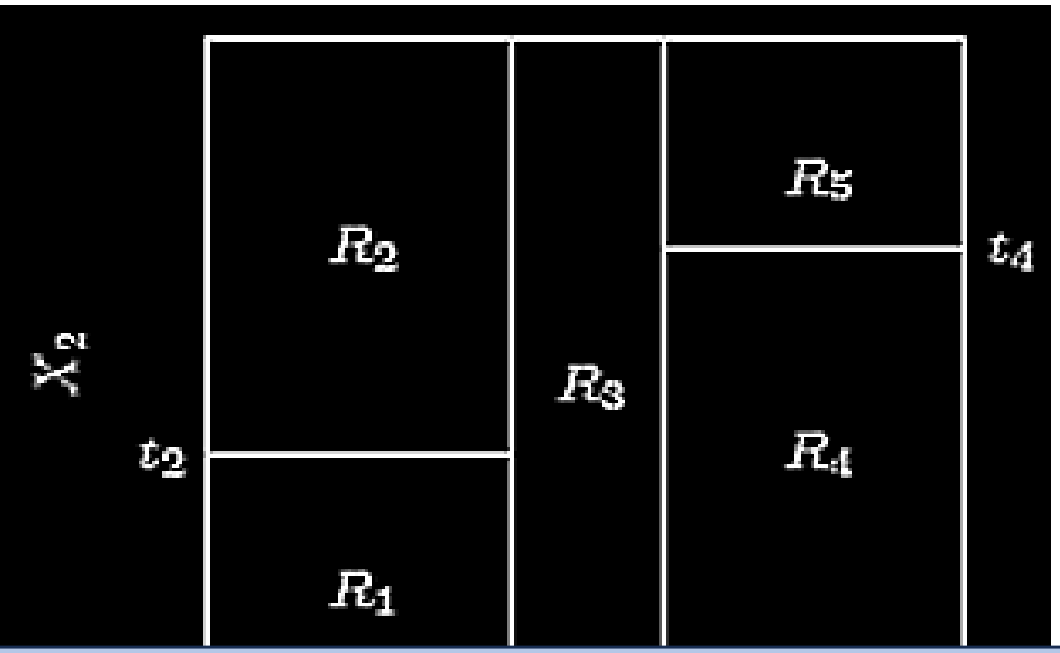
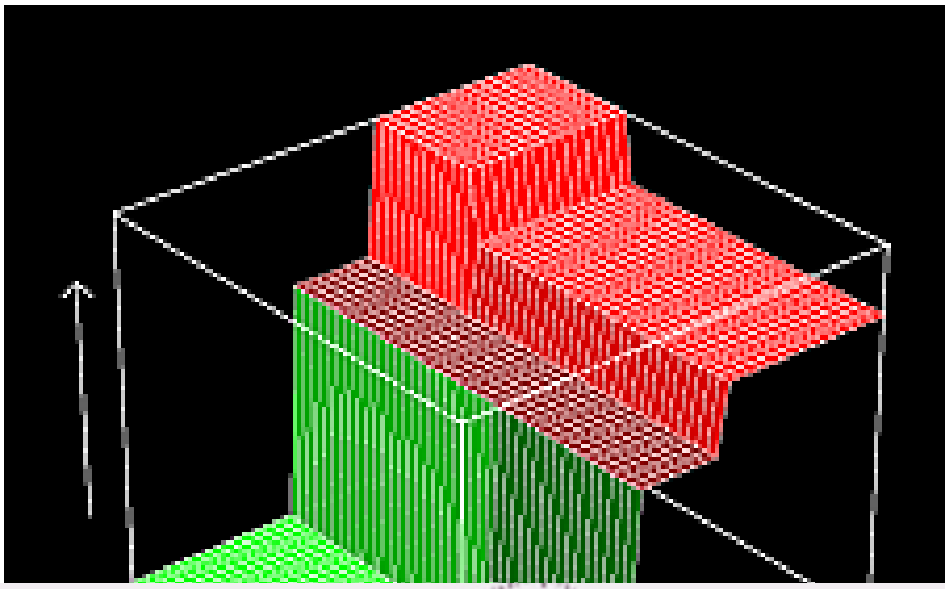
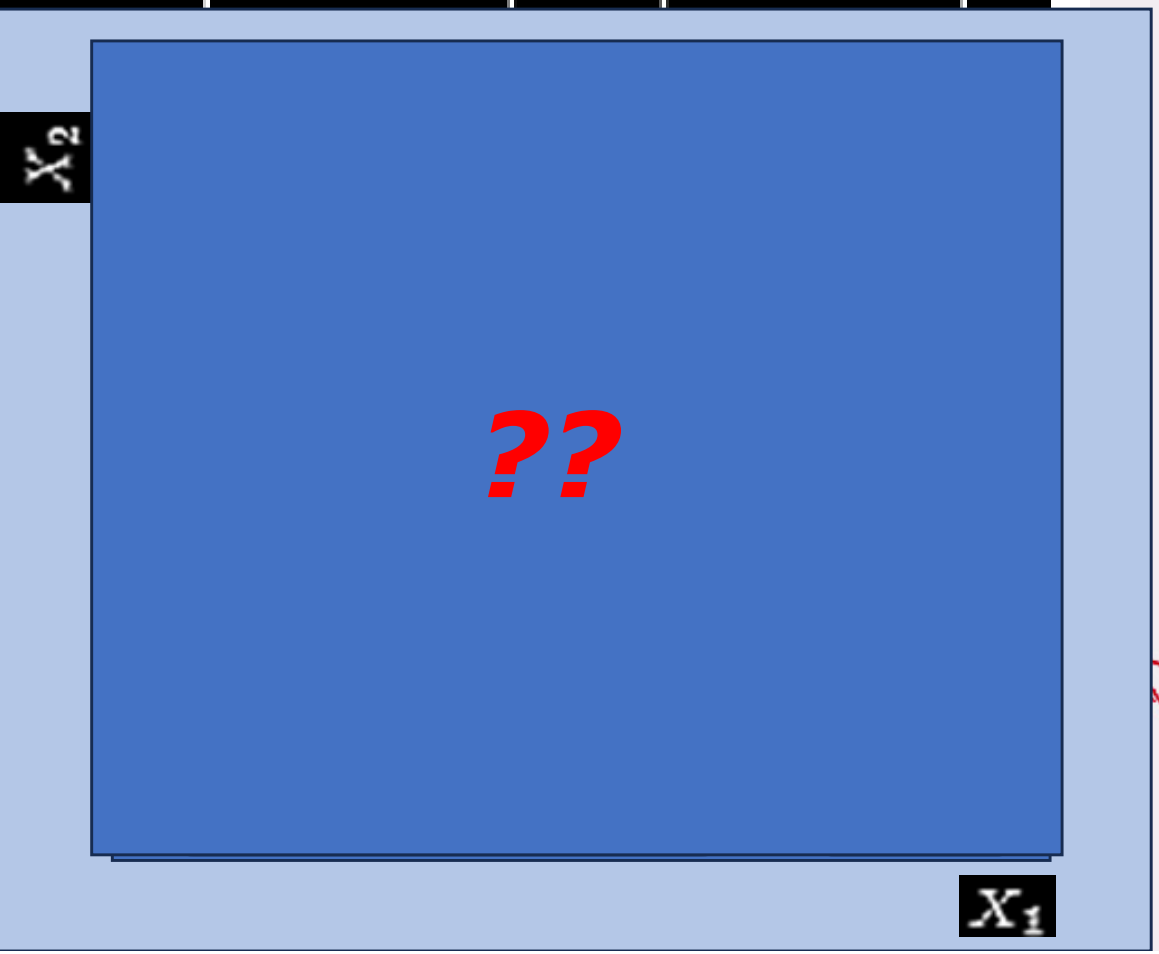
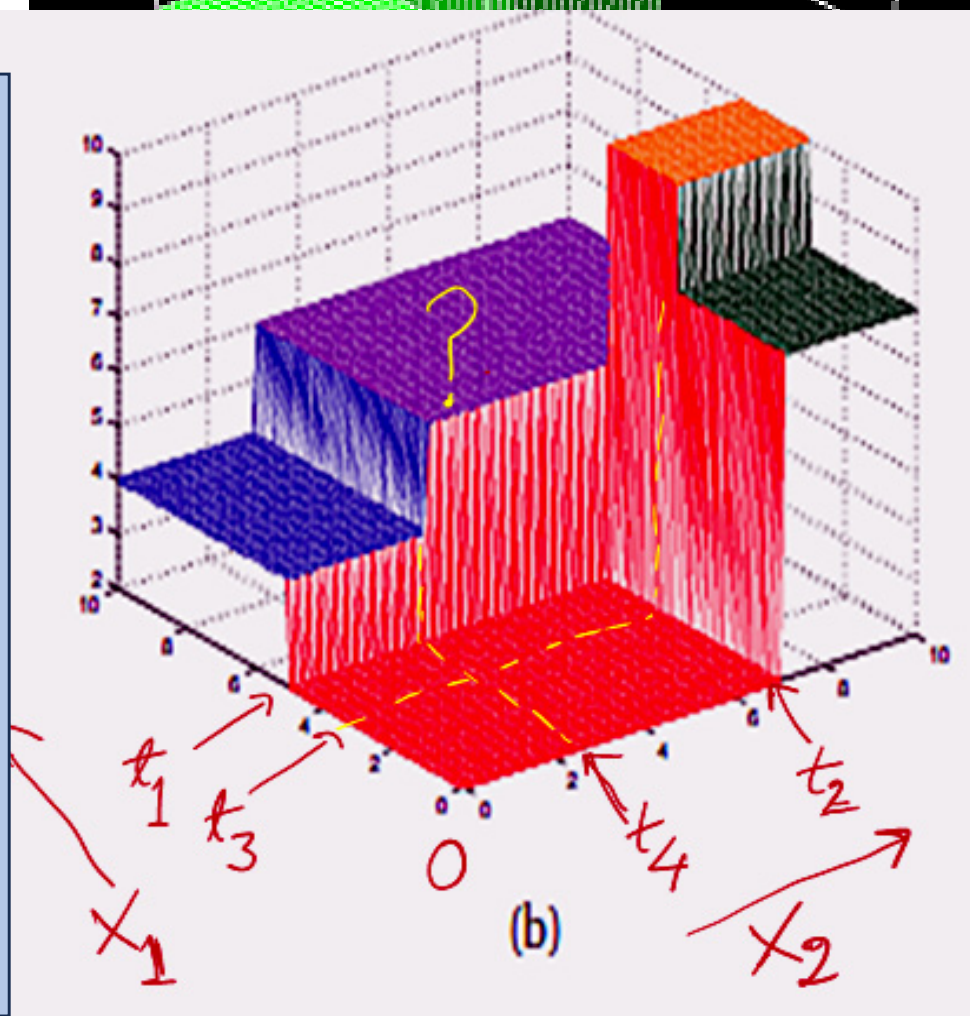


Figure 16.1 A simple regression tree on two inputs. Based on Figure 9.2 of (Hastie et al. 2009)


R_5
R_2
t_4
X_2
R_3
R_4
t_2
R_1




X_2
??
X_1


t_1
t_3
t_2
t_4
O
X_1
X_2
(b)

9.2.2 Regression Trees

We now turn to the question of how to grow a regression tree. Our data consists of p inputs and a response, for each of N observations: that is, (x_i, y_i) for $i = 1, 2, \ldots, N$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have. Suppose first that we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m). \tag{9.10}$$

Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j,s) = \{X | X_j \leq s\} \text{ and } R_2(j,s) = \{X | X_j > s\}. \tag{9.12}$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,\, s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]. \tag{9.13}$$

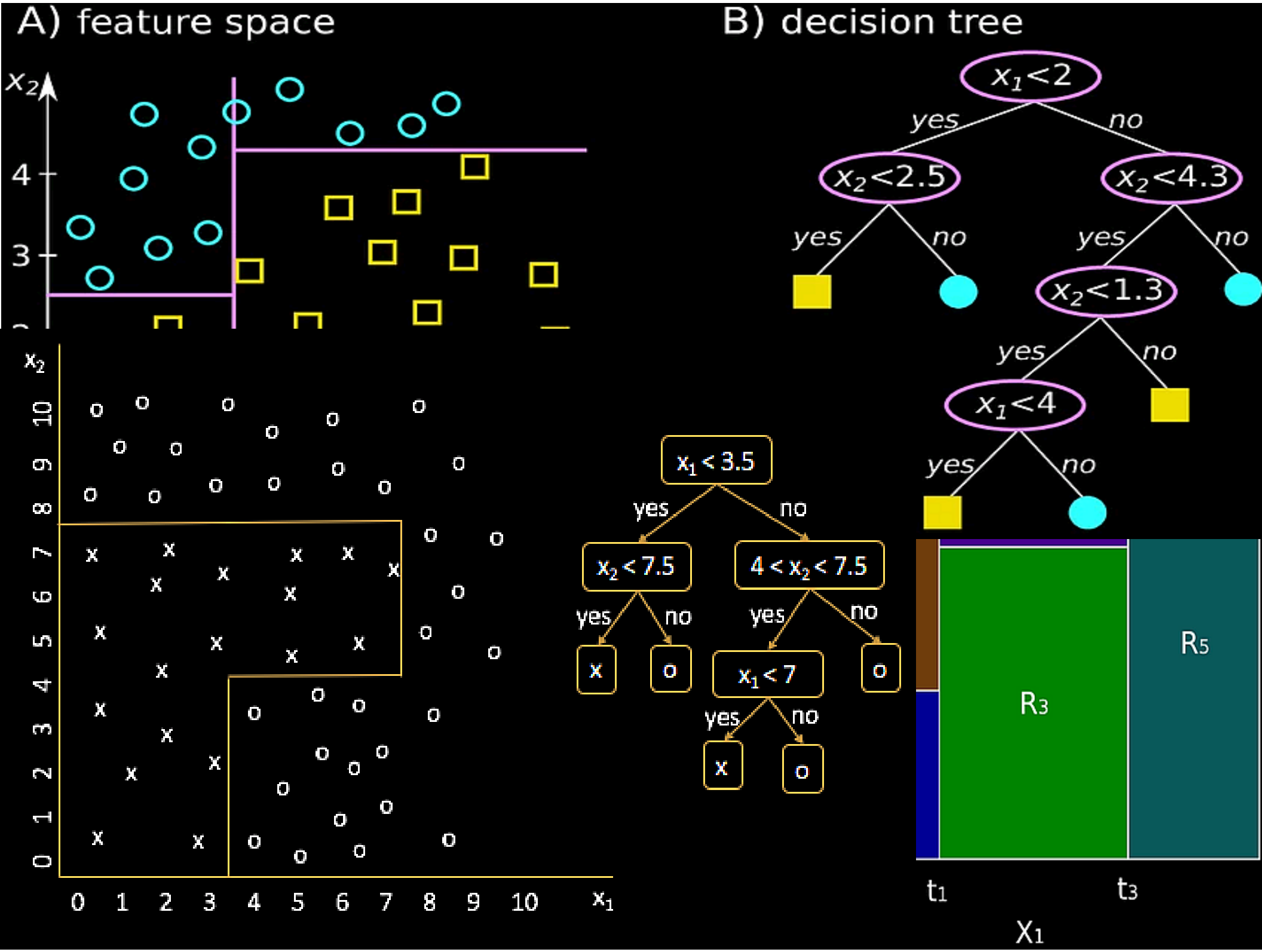
Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. Then this process is repeated on all of the resulting regions.

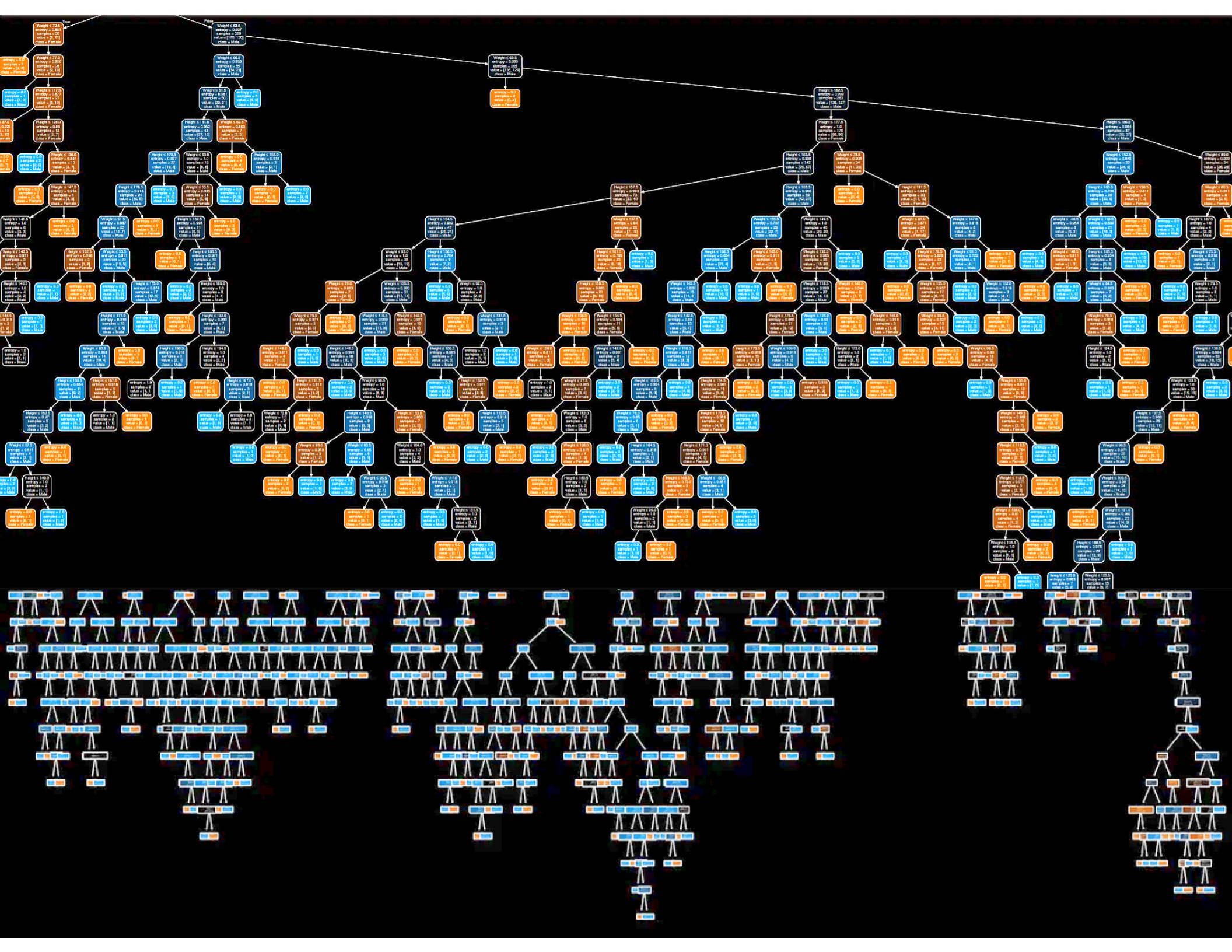
How large should we grow the tree? Clearly a very large tree might overfit the data, while a small tree might not capture the important structure.

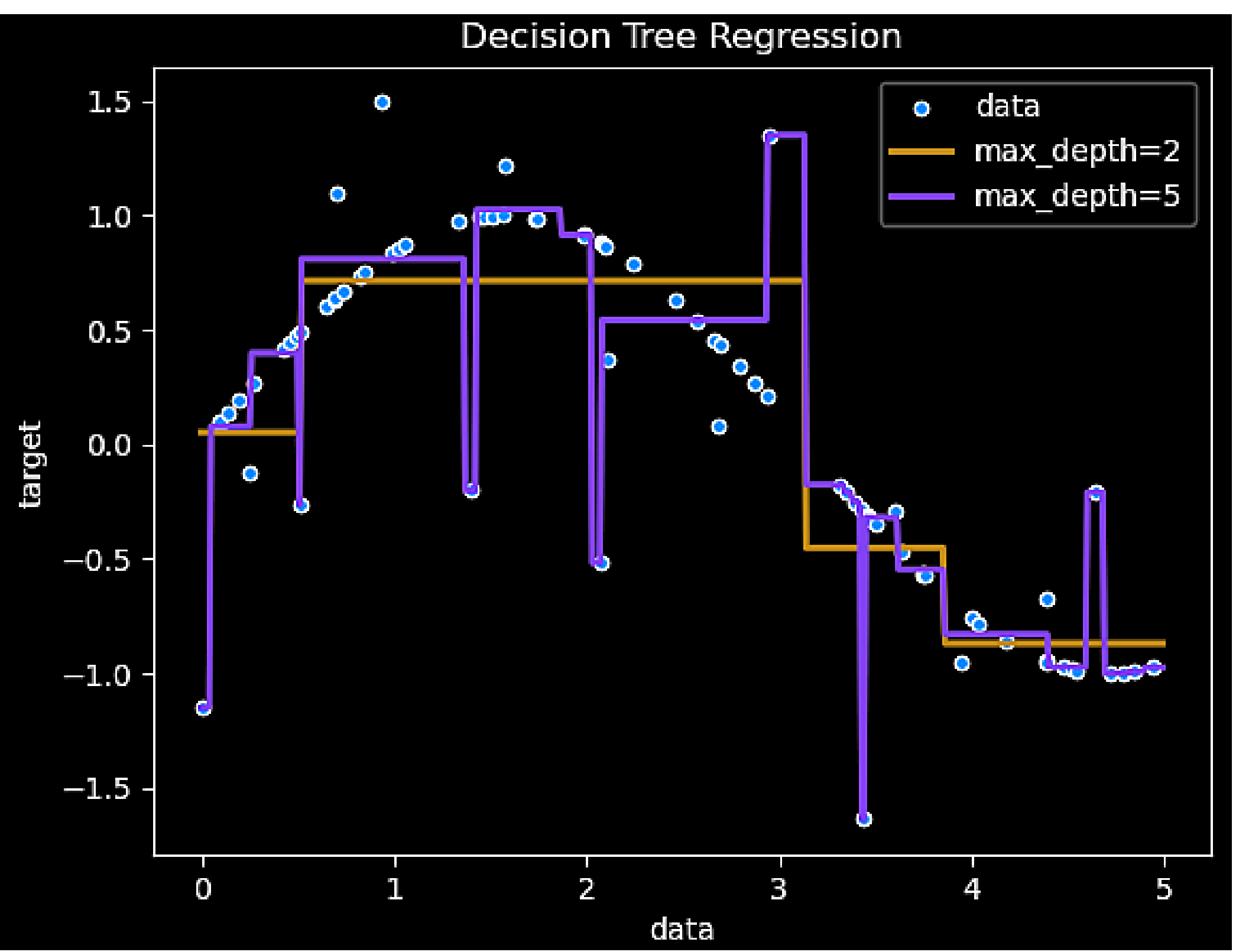
Rudimentary example in next slide →

Tree size is a *tuning parameter governing the model's comp*lexity, and the optimal tree size should be adaptively chosen from the data. One approach would be to ***split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold***. This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it.

The preferred strategy is to grow a large tree T0, stopping the splitting process only when some minimum ***node size*** (say 5) is reached. Then this large tree is pruned using cost-complexity pruning, which will be described later.


A) feature space
B) decision tree
x_2
4
3
$x_1<2$
yes
no
$x_2<2.5$
$x_2<4.3$
yes
no
yes
no
$x_2<1.3$
yes
no
$x_1<4$
yes
no
x_2
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 x_1
$x_1 < 3.5$
yes
no
$x_2 < 7.5$
$4 < x_2 < 7.5$
yes
no
yes
no
x
o
$x_1 < 7$
o
yes
no
x
o
R_3
R_5
t_1
t_3
X_1




Decision Tree Regression
data
max_depth=2
max_depth=5
target
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5

9.2.3 Classification Trees

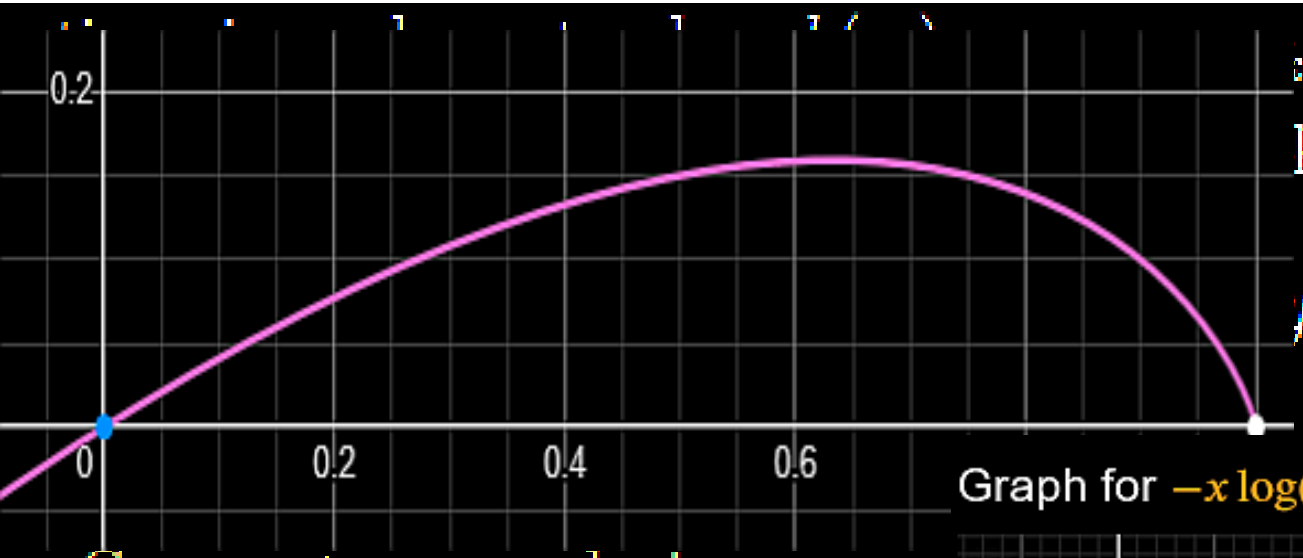
If the target is a classification outcome taking values $1, 2, \ldots, K$, the only classification. In a node m, representing a region R_m with N_m observations, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

the proportion of class k observations in node m. We classify the observations in node m to class $k(m) = \arg\max_k \hat{p}_{mk}$, the majority class in node m. Different measures $Q_m(T)$ of node impurity include the following:

Misclassification error:	$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}.$
Gini index:	$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$
Cross-entropy or deviance:	$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$

(9.17)



$\hat{p}_{mk}$, the majority class in purity include the following:

$i \neq k(m)) = 1 - \hat{p}_{mk(m)}$.

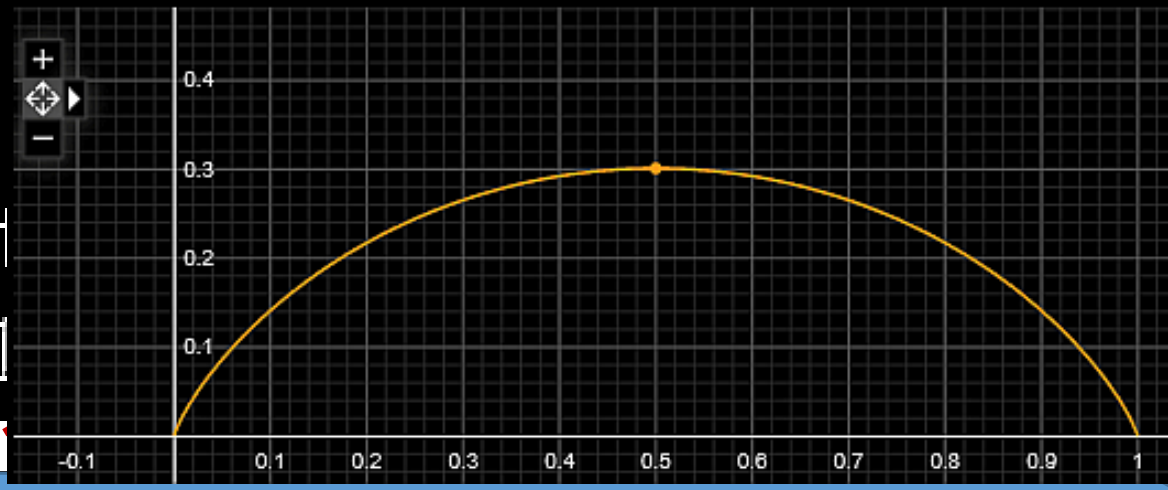
$\sum_{k=1}^{K}$

Cross-entropy or deviance: $-$

For two classes, if p is the propor

sures are $1 - \max(p, 1-p)$, $2p(1$

Graph for $-x\log(x) - (1-x)\log(1-x)$



PLOT THEM

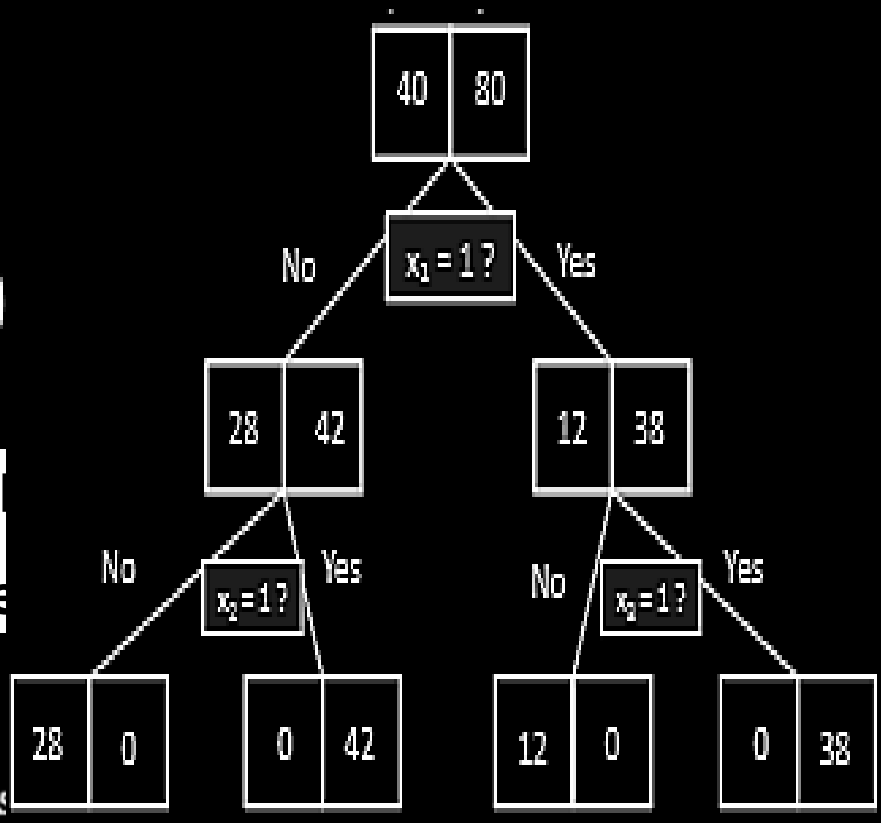
Algorithm Recursive procedure to grow a classification/ regression tree

```
1  function fitTree(node, 𝒟, depth) ;
2  node.prediction = mean(y_i : i ∈ 𝒟) // or class label distribution ;
3  (j*, t*, 𝒟_L, 𝒟_R) = split(𝒟);
4  if not worthSplitting(depth, cost, 𝒟_L, 𝒟_R) then
5      return node
6  else
7      node.test = λx.x_j* < t*
8      node.left = fitTree(node, 𝒟_L, depth+1);
9      node.right = fitTree(node, 𝒟_R, depth+1)
10     return node;
```

Node –

D – Tree

$$\min_{j,\,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s}\right.$$



The split function chooses the best feature, and the best value for that feature, as follow

$$(j^*, t^*) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} \leq t\}) + \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} > t\})$$

The function that checks if a node is worth splitting can use several stopping heuristics, such as the following:

- is the reduction in cost too small? Typically we define the gain of using a feature to be a normalized measure of the reduction in cost:

$$\Delta \triangleq \text{cost}(\mathcal{D}) - \left(\frac{|\mathcal{D}_L|}{|\mathcal{D}|} \text{cost}(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|} \text{cost}(\mathcal{D}_R) \right) \tag{16.6}$$

- has the tree exceeded the maximum desired depth?
- is the distribution of the response in either $\mathcal{D}_L$ or $\mathcal{D}_R$ sufficiently homogeneous (e.g., all labels are the same, so the distribution is pure)?
- is the number of examples in either $\mathcal{D}_L$ or $\mathcal{D}_R$ too small?

Regression cost

In the regression setting, we define the cost as follows:

$$\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \overline{y})^2$$

Classification cost

In the classification setting, there are several ways to measure the quality of a split. First, we fit a multinoulli model to the data in the leaf satisfying the test $X_j < t$ by estimating the class-conditional probabilities as follows:

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i = c) \tag{16.8}$$

where $\mathcal{D}$ is the data in the leaf. Given this, there are several common error measures for evaluating a proposed partition:

- **Misclassification rate.** We define the most probable class label as $\hat{y}_c = \text{argmax}_c \, \hat{\pi}_c$. The corresponding error rate is then

$$\frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i \neq \hat{y}) = 1 - \hat{\pi}_{\hat{y}} \tag{16.9}$$

- **Entropy, or deviance:**

$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c \tag{16.10}$$

Note that minimizing the entropy is equivalent to maximizing the **information gain** (Quinlan 1986) between test $X_j < t$ and the class label Y, defined by

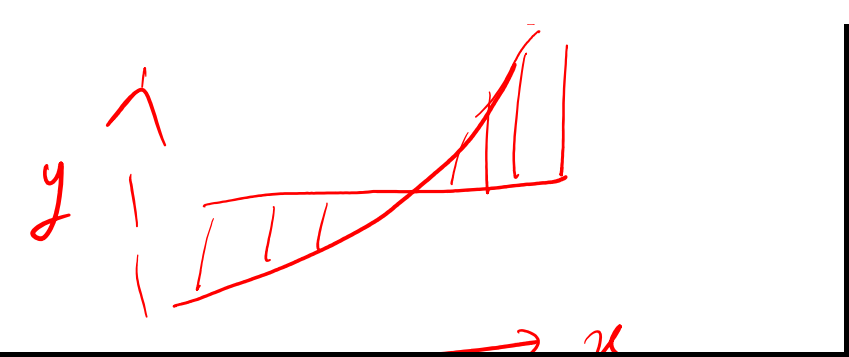
$$\begin{aligned} \text{infoGain}(X_j < t, Y) &\triangleq \mathbb{H}(Y) - \mathbb{H}(Y|X_j < t) && (16.11)\\ &= \left(-\sum_c p(y=c)\log p(y=c)\right) && (16.12)\\ &\quad + \left(\sum_c p(y=c|X_j < t)\log p(c|X_j < t)\right) && (16.13) \end{aligned}$$

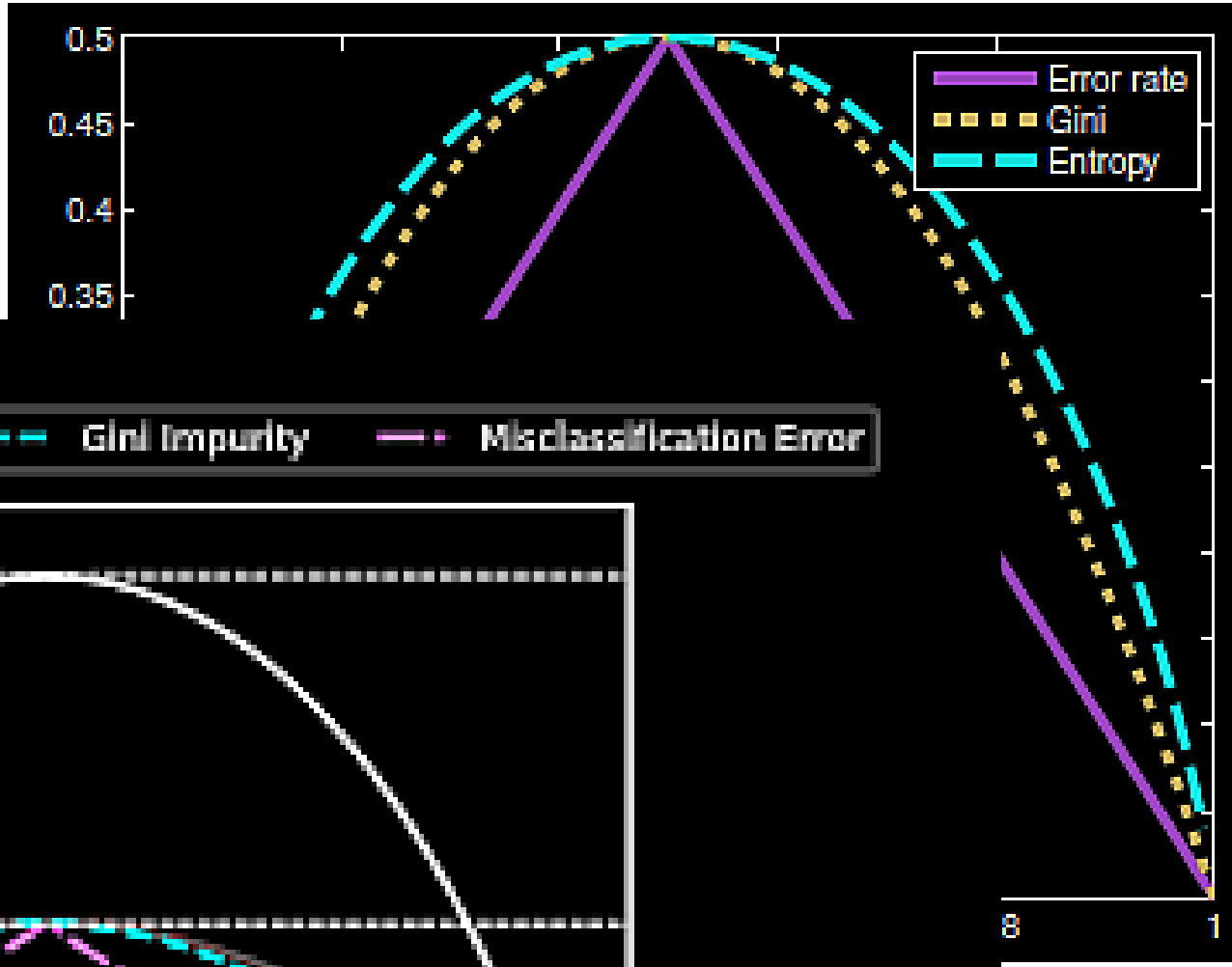
since $\hat{\pi}_c$ is an MLE for the distribution $p(c|X_j < t)$.[1]

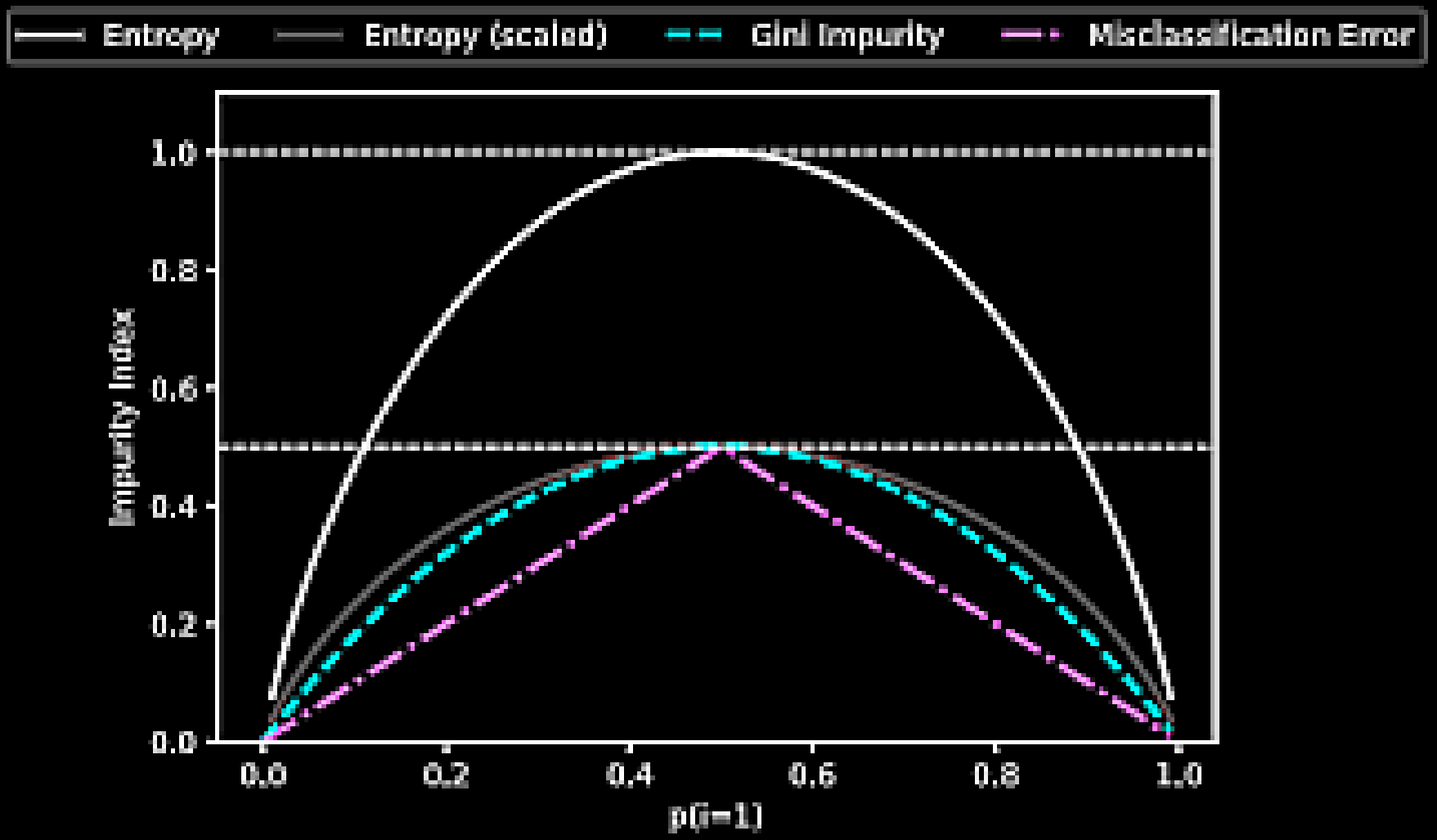
- **Gini index**

$$\sum_{c=1}^{C} \hat{\pi}_c(1-\hat{\pi}_c) = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2 \tag{16.14}$$

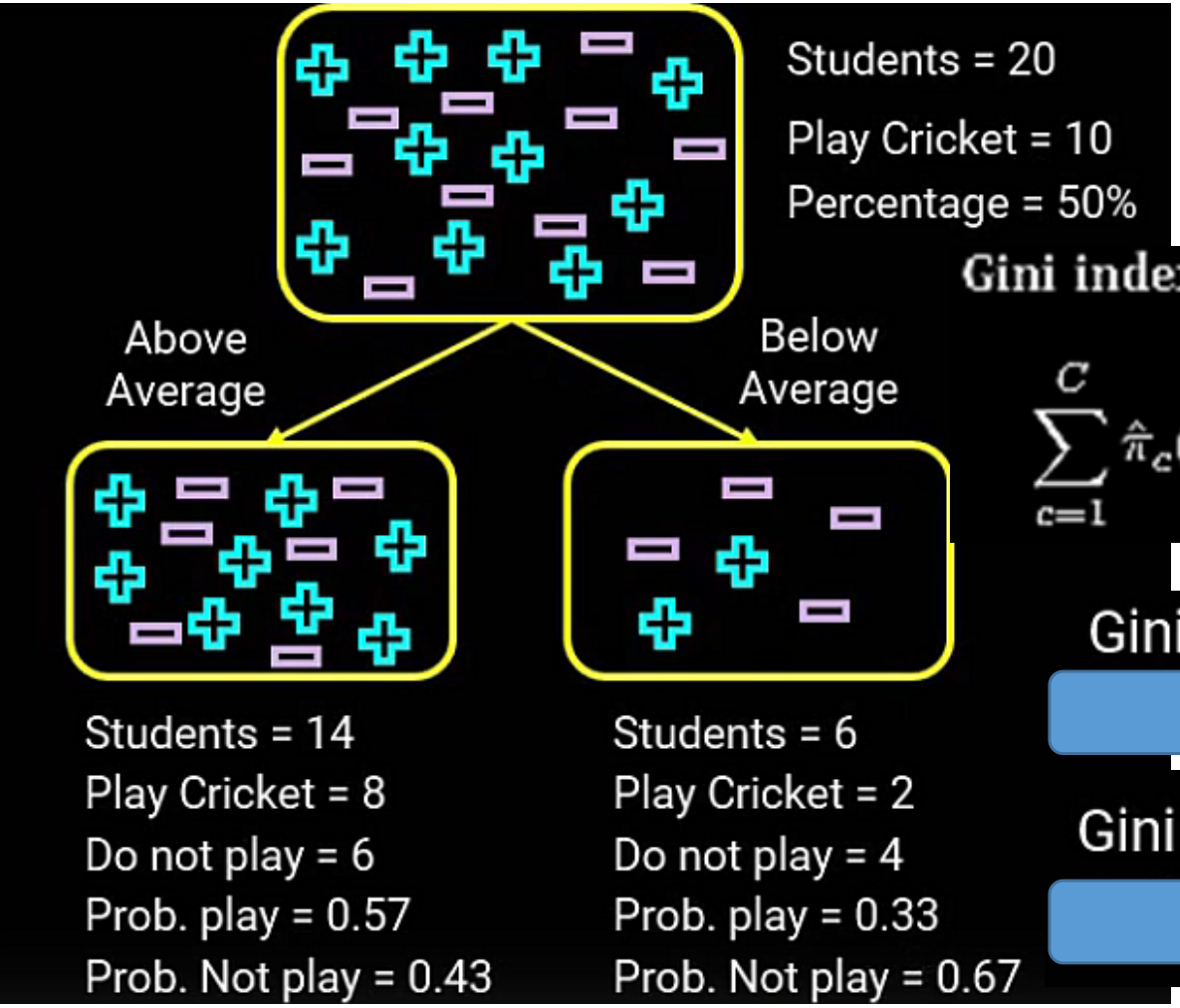
This is the expected error rate. To see this, note that $\hat{\pi}_c$ is the probability a random entry in the leaf belongs to class c, and $(1-\hat{\pi}_c$ is the probability it would be misclassified.







Comparison of different impurity measures.



Gini impurity = 1 – Gini

Gini index

$$\sum_{c=1}^{C} \hat{\pi}_c(1-\hat{\pi}_c) = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2$$

Gini Impurity: sub-node Below Average:

Gini Impurity: sub-node Above Average:

Split on Class

Gini Impurity: sub-node Class IX:
1 - [(0.8)*(0.8) + (0.2)*(0.2)] = 0.32

Gini Impurity: sub-node Class X:
1 - [(0.2)*(0.2) + (0.8)*(0.8)] = 0.32

Weighted Gini Impurity: Performance in Class:
(14/20)*0.49 + (6/20)*0.44 = 0.475

$$\Delta = 0.5 - 0.475; \quad 0.5 - 0.32$$

Weighted Gini Impurity: Class:
(10/20)*0.32 + (10/20)*0.32 = 0.32

Criteria for Splitting nodes - revisited

a node is split into child nodes.

$$Variance = \frac{\sum (X - \mu)^2}{N}$$

Variance is used for calculating the homogeneity of a node. If a node is entirely homogeneous, then the variance is zero.

Here are the steps to split a decision tree using the reduction in variance method:

1. For each split, individually calculate the variance of each child node
2. Calculate the variance of each split as the weighted average variance of child nodes
3. Select the split with the lowest variance
4. Perform steps 1-3 until completely homogeneous nodes are achieved

$$Information\ Gain = 1 - Entropy$$

Entropy is used for calculating the purity of a node. **The lower the value of entropy, the higher the purity of the node.** The entropy of a homogeneous node is zero. Since we subtract entropy from 1, the Information Gain is higher for the purer nodes with a maximum value of 1. Now, let's take a look at the formula for calculating the entropy:

$$Entropy = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Steps to split a decision tree using Information Gain:

1. For each split, individually calculate the entropy of each child node
2. Calculate the entropy of each split as the weighted average entropy of child nodes
3. Select the split with the lowest entropy or highest information gain
4. Until you achieve homogeneous nodes, repeat steps 1-3

$$Gini\ Impurity = 1 - Gini$$

Wait – what is Gini?

Gini is the probability of correctly labeling a randomly chosen element if it is randomly labeled according to the distribution of labels in the node. The formula for Gini is:

$$Gini = \sum_{i=1}^{n} p_i^2$$

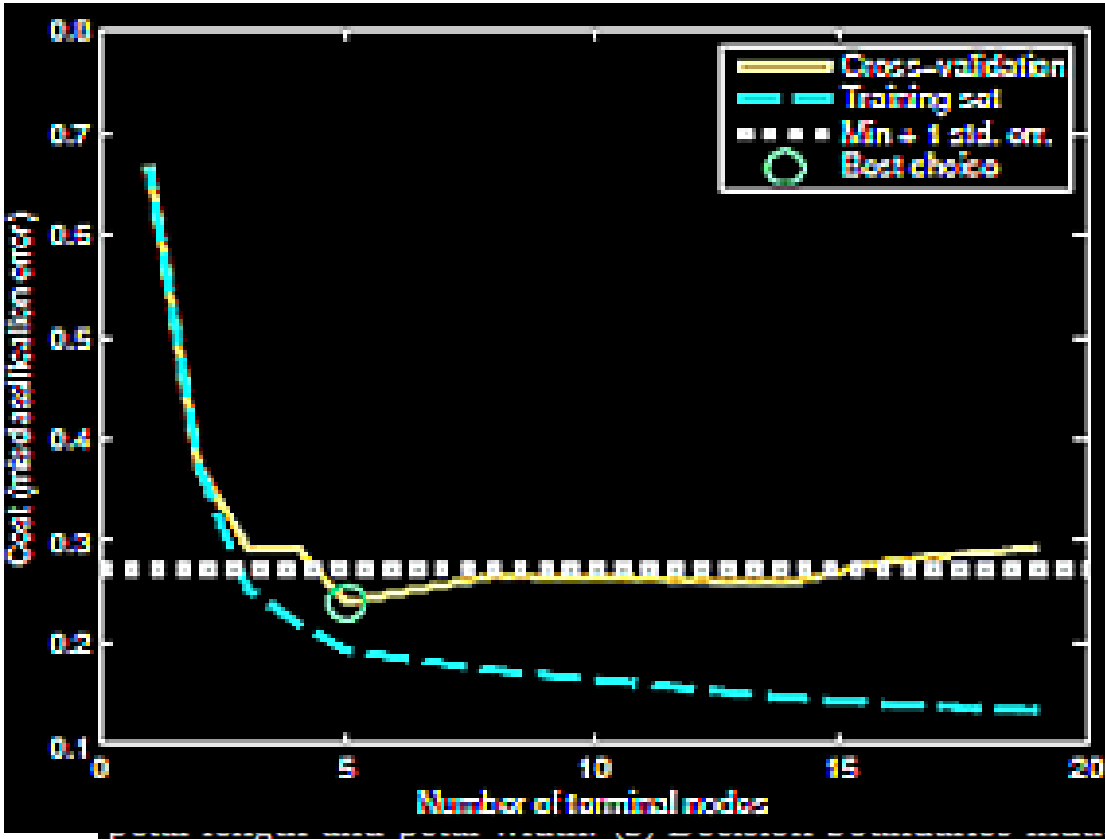
And Gini Impurity is:

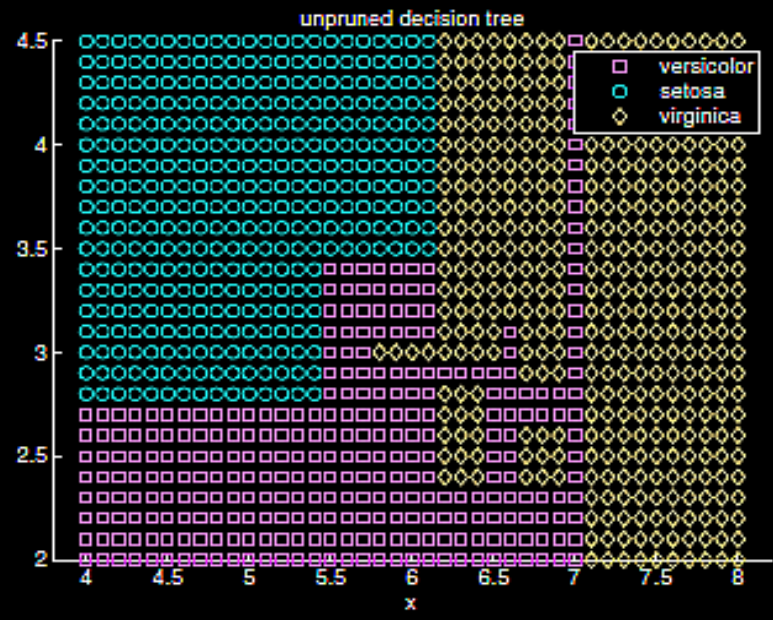
$$Gini\ Impurity = 1 - \sum_{i=1}^{n} p_i^2$$

The lower the Gini Impurity, the higher the homogeneity of the node. **The Gini Impurity of a pure node is zero.**

Steps to split a decision tree using Gini Impurity:

1. Similar to what we did in information gain. For each split, individually calculate the Gini Impurity of each child node
2. Calculate the Gini Impurity of each split as the weighted average Gini Impurity of child nodes
3. Select the split with the lowest value of Gini Impurity
4. Until you achieve homogeneous nodes, repeat steps 1-3



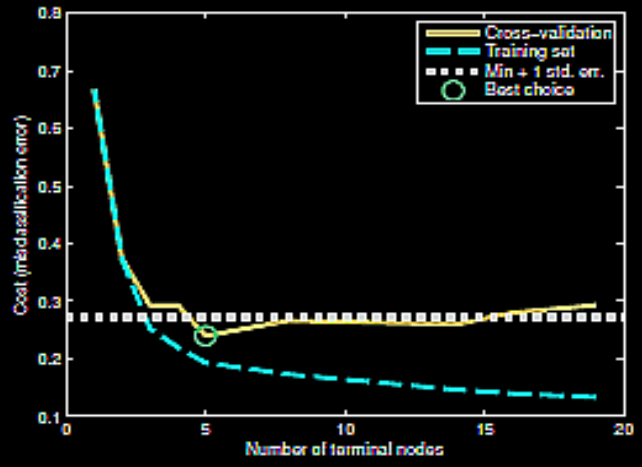


(b)

tures, sepal length and sepal width, and ignore

ced by the decision tree in Figure 16.5(a).

(a)



(b)

Figure 16.5 (a) Unpruned decision tree for Iris data. (b) Plot of misclassification error rate vs depth of tree. Figure generated by `dtreeDemoIris`.

WHEN TO STOP ADDING NODES

- A simple approach would be to stop when the *reduction in residual error falls below some threshold.*
- The most common stopping procedure is to use a minimum count on the number of training instances assigned to each leaf node. If the *count is less than some minimum* then the split is not accepted and the node is taken as a final leaf node.
- However, it is found empirically that often none of the available splits produces a significant reduction in error, and yet after several more splits a substantial error reduction is found.

- For this reason, it is common practice *to grow a large tree*, using a stopping criterion based on the number of data points associated with the leaf nodes, and then *prune back the resulting tree*.
- The pruning is based on a criterion that balances residual error against a measure of model complexity.

WHEN TO STOP ADDING NODES

- If we denote the starting tree for pruning by T_0, then we define $T \subset T_0$ to be a subtree of T_0 if it can be obtained by pruning nodes from T_0 (in other words, by collapsing internal nodes by combining the corresponding regions).
- Suppose the leaf nodes are indexed by $\tau = 1, \ldots, |T|$, with leaf node τ representing a region R_τ of input space having N_τ data points, and $|T|$ denoting the total number of leaf nodes.
- The optimal prediction for region R_τ is then given by

$$y_\tau = \frac{1}{N_\tau} \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} t_n$$

Cont'd in next slide:

WHEN TO STOP ADDING NODES

- and the corresponding contribution to the residual sum-of-squares is then

$$Q_\tau(T) = \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} \{t_n - y_\tau\}^2$$

- The pruning criterion is then given by

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda |T|$$

- The regularization parameter λ determines the trade-off between the *overall residual sum-of-squares error and the complexity of the model* as measured by the number $|T|$ of leaf nodes, and its *value is chosen by cross-validation.*

WHEN TO STOP ADDING NODES

- For classification problems, the process of growing and pruning the tree is similar, except that the sum-of-squares error is replaced by a more appropriate measure of performance.
- If we define $p_{\tau k}$ to be the proportion of data points in region R_τ assigned to class k, where $k = 1, \ldots, K$, then two commonly used choices are the cross-entropy

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k} \ln p_{\tau k}$$

- and, the *Gini index*

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k} (1 - p_{\tau k})$$

- These both vanish for $p_{\tau k} = 0$ and $p_{\tau k} = 1$ and have a maximum at $p_{\tau k} = 0.5$.

Advantages

- The cross entropy and the Gini index are better measures than the misclassification rate for growing the tree because they are more sensitive to the node probabilities.
- Also, unlike misclassification rate, they are differentiable and hence better suited to gradient based optimization methods.
- The human interpretability of a tree model such as CART is often seen as its major strength.

Disadvantages

- In practice it is found that the particular tree structure that is learned is very sensitive to the details of the data set, so that a small change to the training data can result in a very different set of splits.

Decision Tree Pruning

Analytics

And

An Example

We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal (non-terminal) nodes. We index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T. Letting

$$\begin{aligned} N_m &= \#\{x_i \in R_m\}, \\ \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \end{aligned} \tag{9.15}$$

we define the cost complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|. \tag{9.16}$$

The idea is to find, for each α, the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$. The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data. Large values of α result in smaller trees T_α, and conversely for smaller values of α.

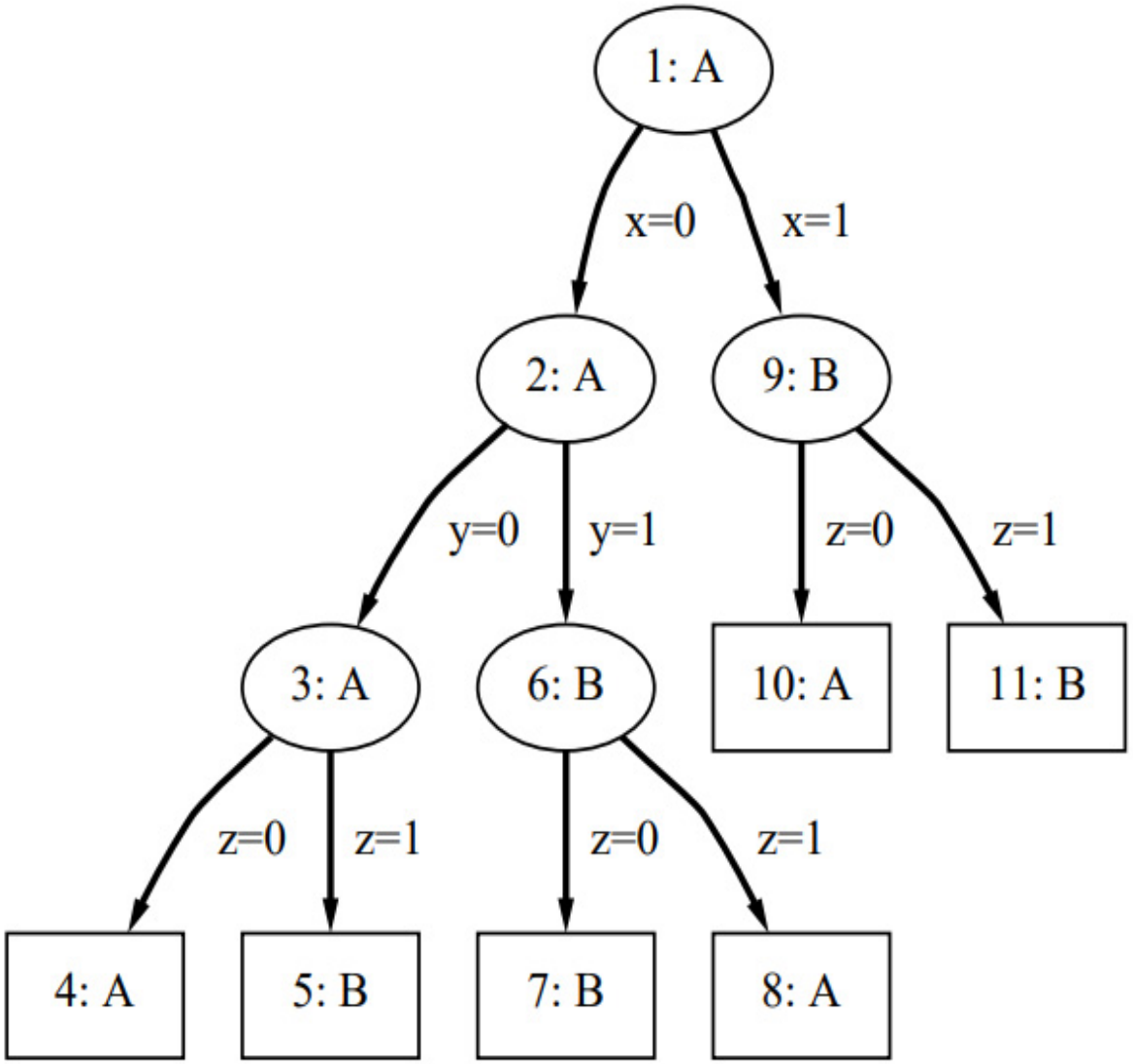
Either **the Gini index or cross-entropy** should be used when growing the tree.

To guide cost-complexity pruning, any of the three measures can be used, but typically it is **the misclassification rate.**

The Gini index can be interpreted in two interesting ways. Rather than classify observations to the majority class in the node, we could classify them to class k with probability $\hat{p}_{mk}$. Then the training error rate of this rule in the node is $\sum_{k \neq k'} \hat{p}_{mk}\hat{p}_{mk'}$—the Gini index. Similarly, if we code each observation as 1 for class k and zero otherwise, the variance over the node of this 0-1 response is $\hat{p}_{mk}(1 - \hat{p}_{mk})$. Summing over classes k again gives the Gini index.

Gini index

$$\sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c) = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2$$

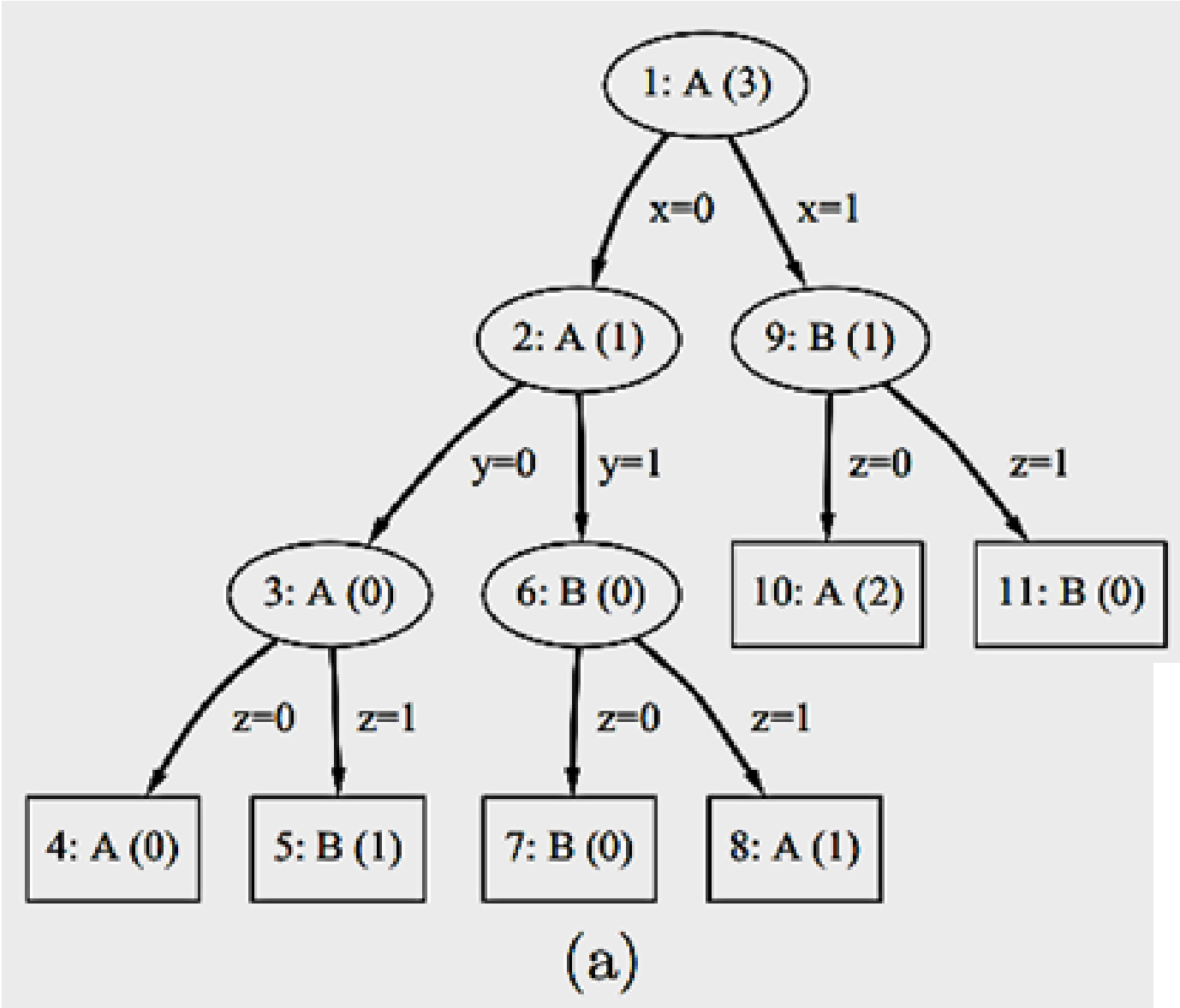


A decision tree with two classes A and B
(with node numbers and class labels)

x	y	z	class
0	0	1	A
0	1	1	B
1	1	0	B
1	0	0	B
1	1	1	A

An example pruning set

- The idea is to hold out some of the available instances—the “pruning set”—when the tree is built, and prune the tree until the classification error on these independent instances starts to increase.
- Because **the instances in the pruning set are not used for building the decision tree**, they provide a less biased estimate of its error rate on future instances than the training data.

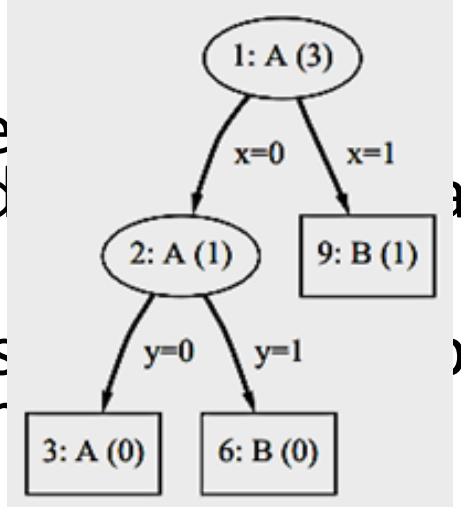


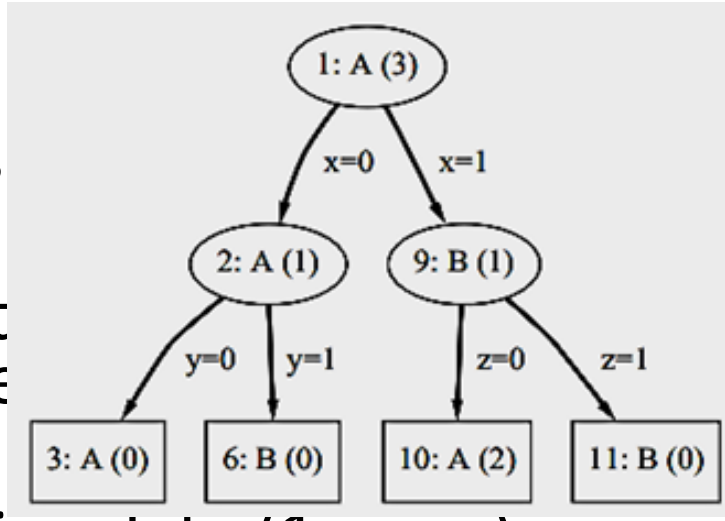
(a)

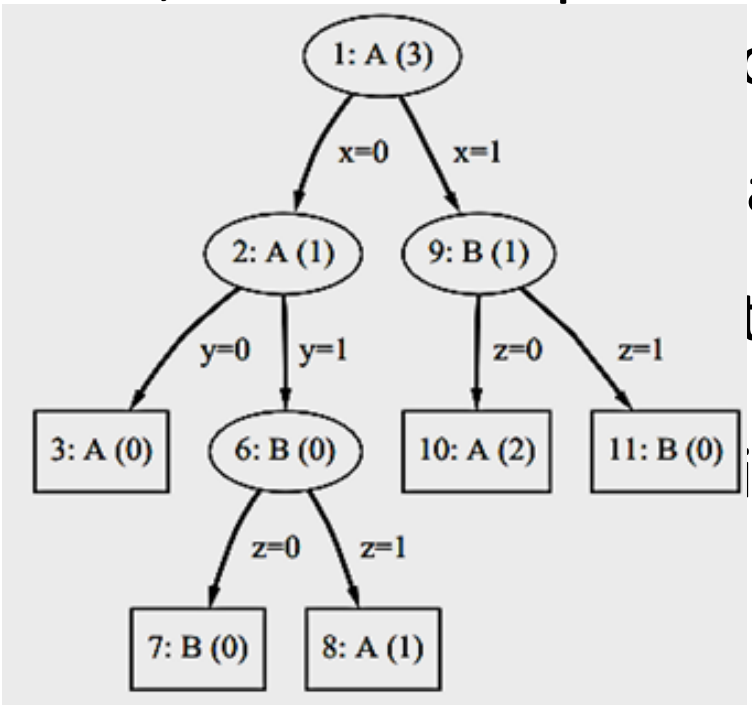
x	y	z	class
0	0	1	A
0	1	1	B
1	1	0	B
1	0	0	B
1	1	1	A

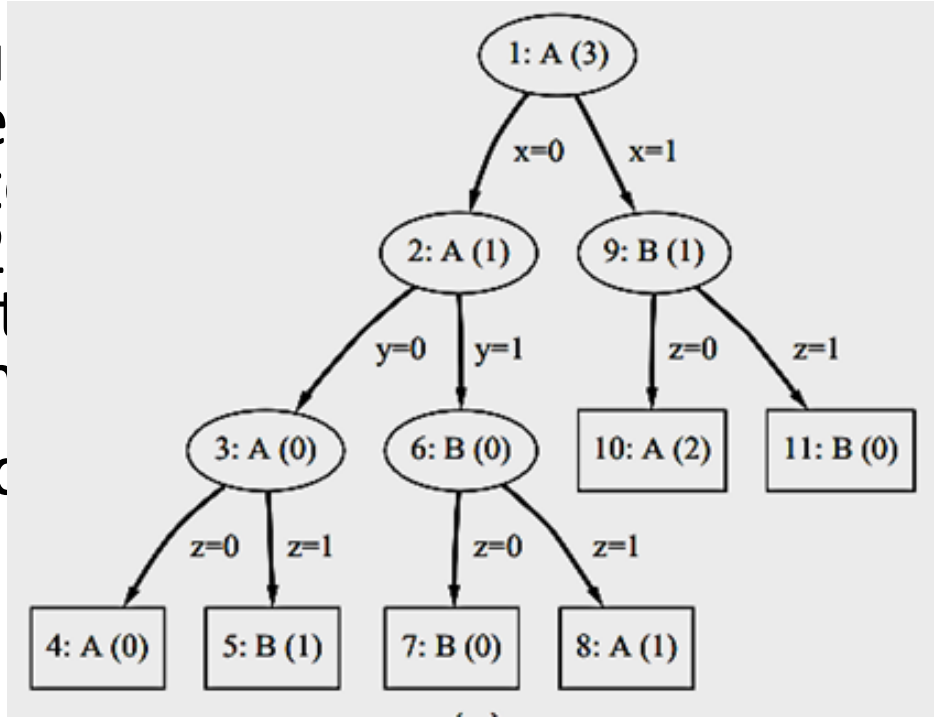
x	y	z	class
0	0	1	A
0	1	1	B
1	1	0	B
1	0	0	B
1	1	1	B

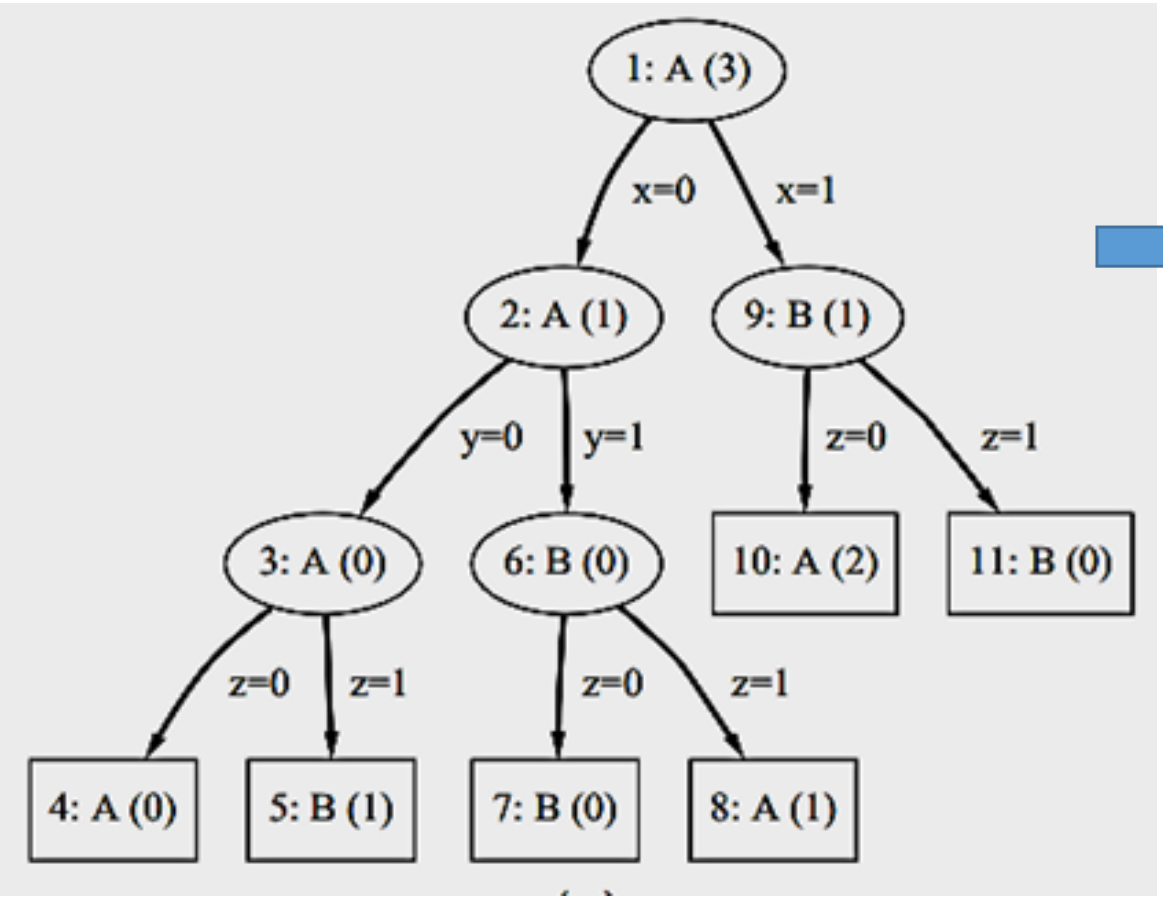
- In each tree, the number of instances in
are misclassified by the individual nodes
parentheses.
- Assuming that the tre
procedure first consid
node 3.
- Because the subtree's
exceeds the error of r
converted to a leaf.
- Next, node 6 is replaced by a leaf for the same reason.
-
-

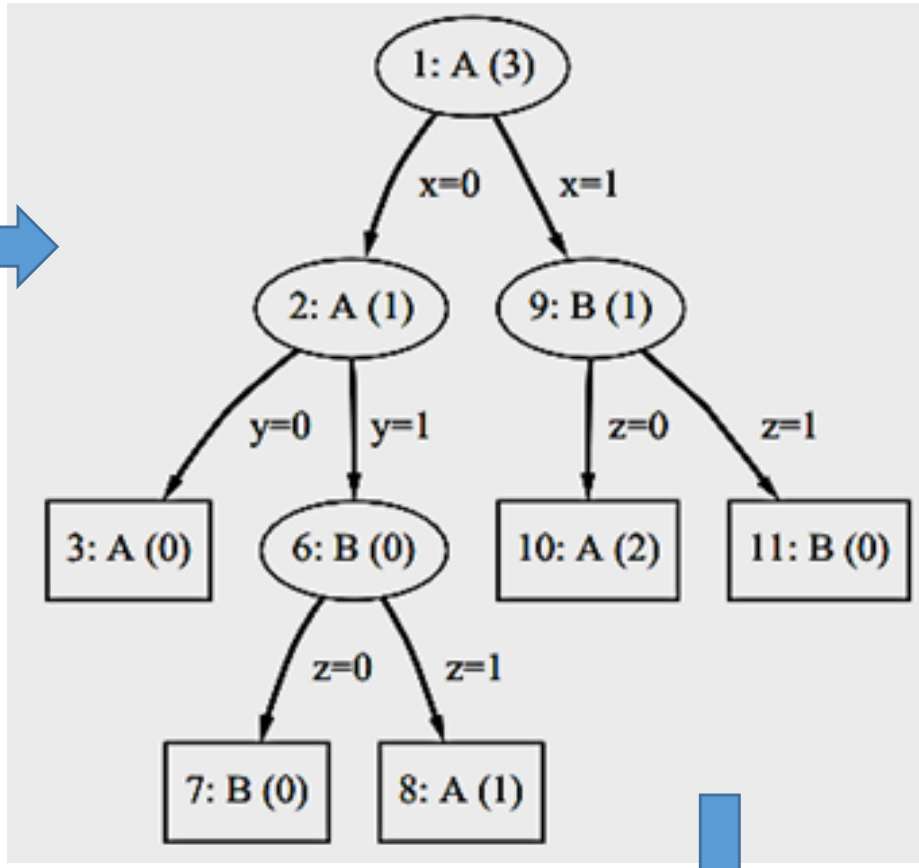


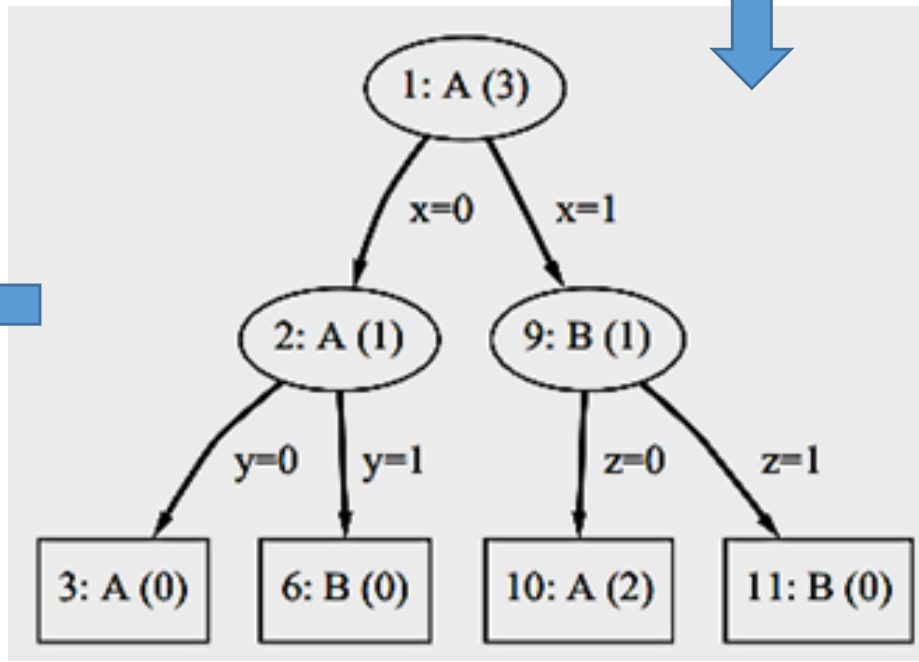


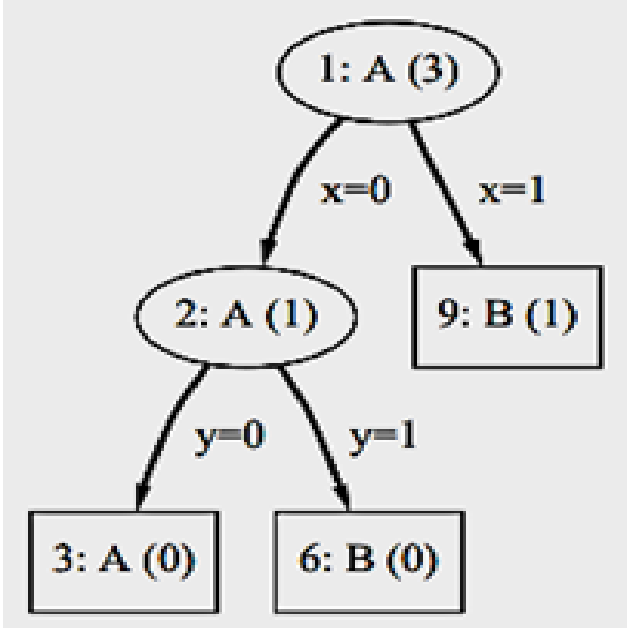


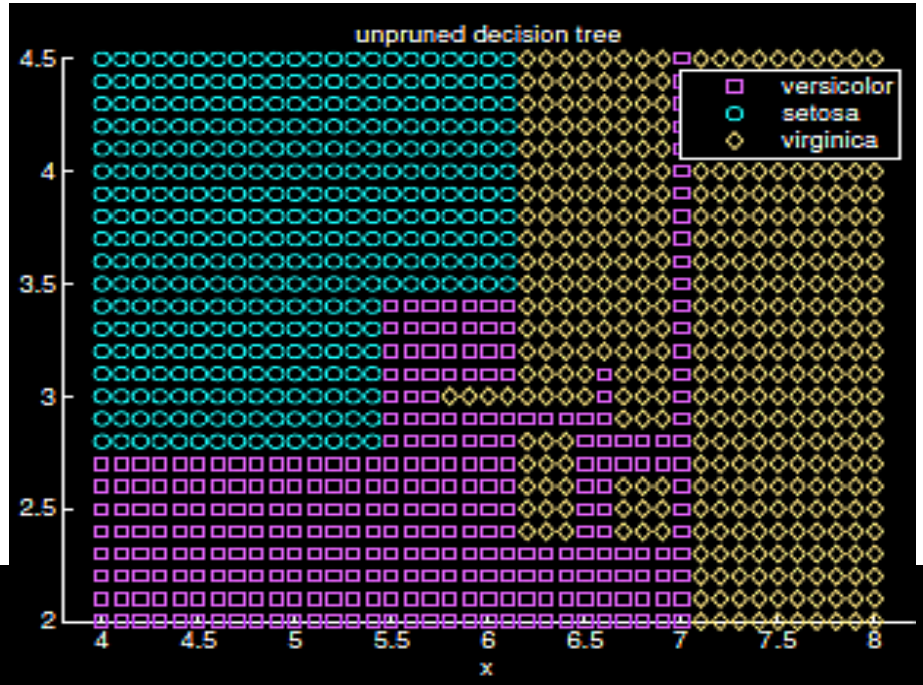


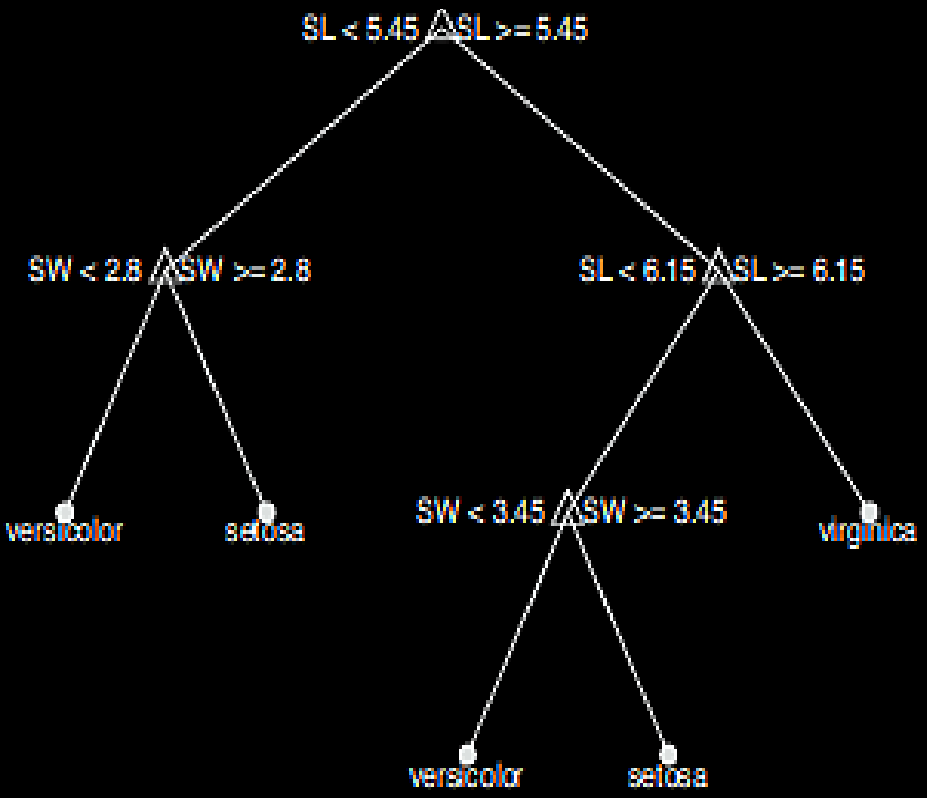

1: A (3)
x=0
x=1
2: A (1)
9: B (1)
y=0
y=1
z=0
z=1
3: A (0)
6: B (0)
10: A (2)
11: B (0)
z=0
z=1
z=0
z=1
4: A (0)
5: B (1)
7: B (0)
8: A (1)

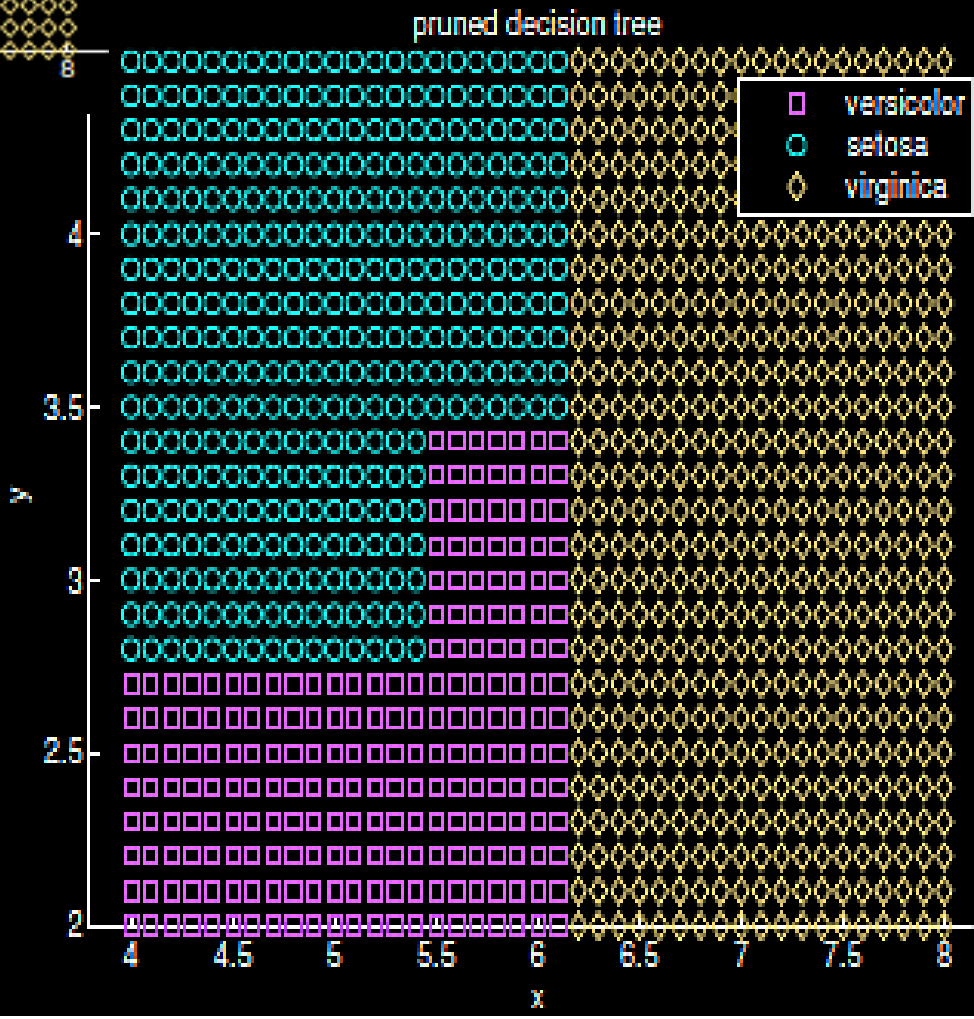

1: A (3)
x=0
x=1
2: A (1)
9: B (1)
y=0
y=1
z=0
z=1
3: A (0)
6: B (0)
10: A (2)
11: B (0)
z=0
z=1
7: B (0)
8: A (1)

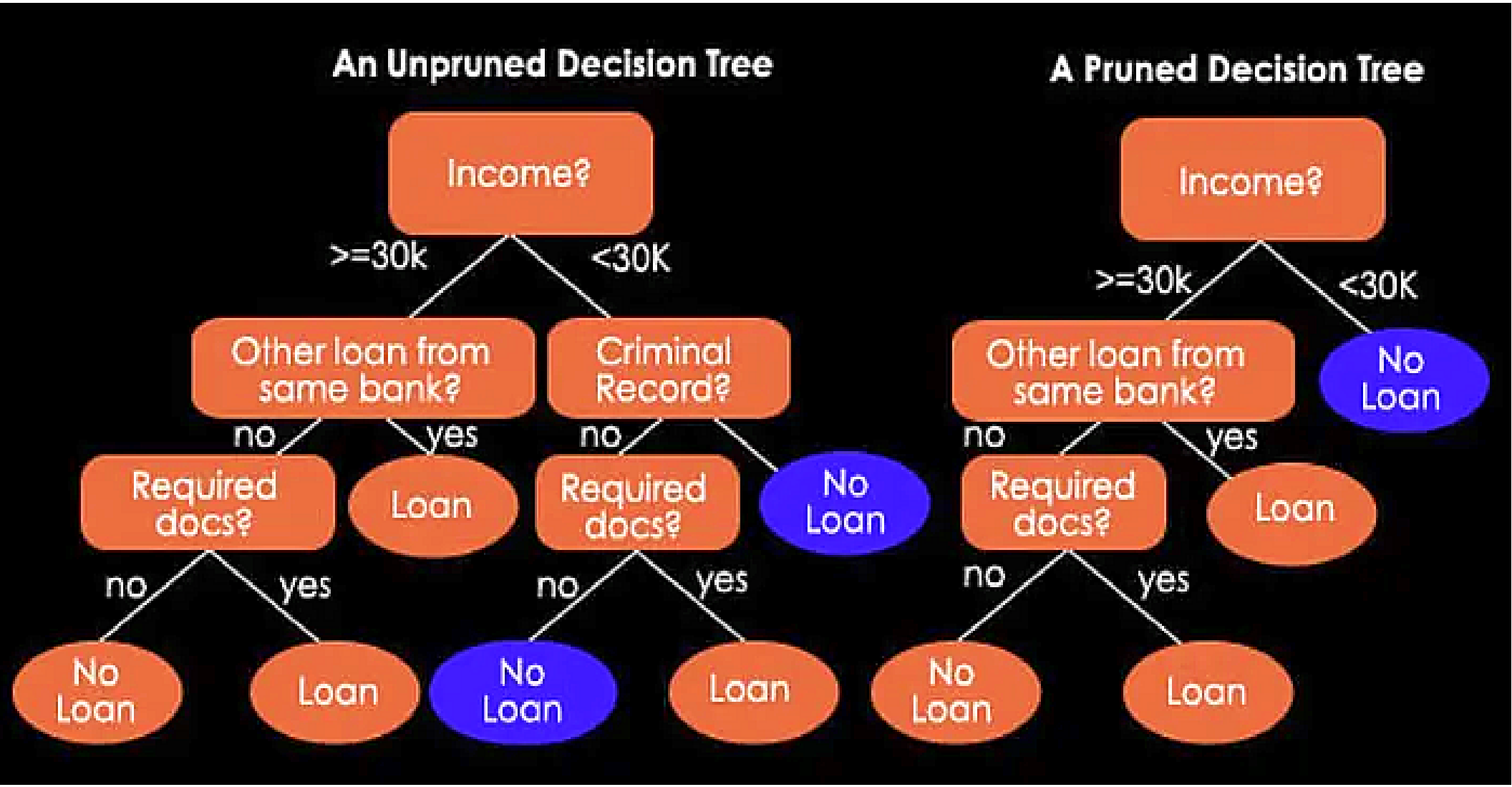

1: A (3)
x=0
x=1
2: A (1)
9: B (1)
y=0
y=1
z=0
z=1
3: A (0)
6: B (0)
10: A (2)
11: B (0)


1: A (3)
x=0
x=1
2: A (1)
9: B (1)
y=0
y=1
3: A (0)
6: B (0)


unpruned decision tree
versicolor
setosa
virginica
4.5
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8
x


SL < 5.45
SL >= 5.45
SW < 2.8
SW >= 2.8
SL < 6.15
SL >= 6.15
versicolor
setosa
SW < 3.45
SW >= 3.45
virginica
versicolor
setosa


pruned decision tree
versicolor
setosa
virginica
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8
x
y


An Unpruned Decision Tree
Income?
>=30k
<30K
Other loan from same bank?
Criminal Record?
no
yes
no
Required docs?
Loan
Required docs?
No Loan
no
yes
no
yes
No Loan
Loan
No Loan
Loan
A Pruned Decision Tree
Income?
>=30k
<30K
Other loan from same bank?
No Loan
no
yes
Required docs?
Loan
no
yes
No Loan
Loan

Evaluating performance of DT Algo:

Accuracy = (T P + T N) / (T P + T N + F P + F N)

Sensitivity (TPR – true positive rate): PR = T P / (T P + F N)

Specificity (TNR – true negative rate): TNR = T N / (T N + F P)

Precision (PPV – positive predictive value): PPV = T P / (T P + F P)

Miss Rate (FNR – false negative rate): FNR = F N / (F N + T P)

False discovery rate (FDR): FDR = F P / (F P + T P)

False omission rate (FOR): FOR = F N / (F N + T N)

Material not covered:

(from Syllabus):
Self-study: Naïve Bayes, Parzen Window, Adaboost

Others: HMM, Bayesian linear regression, Boosting Trees

Outside Syllabus:
Graphical Models, CNN, PLS, ICA, Rough sets;