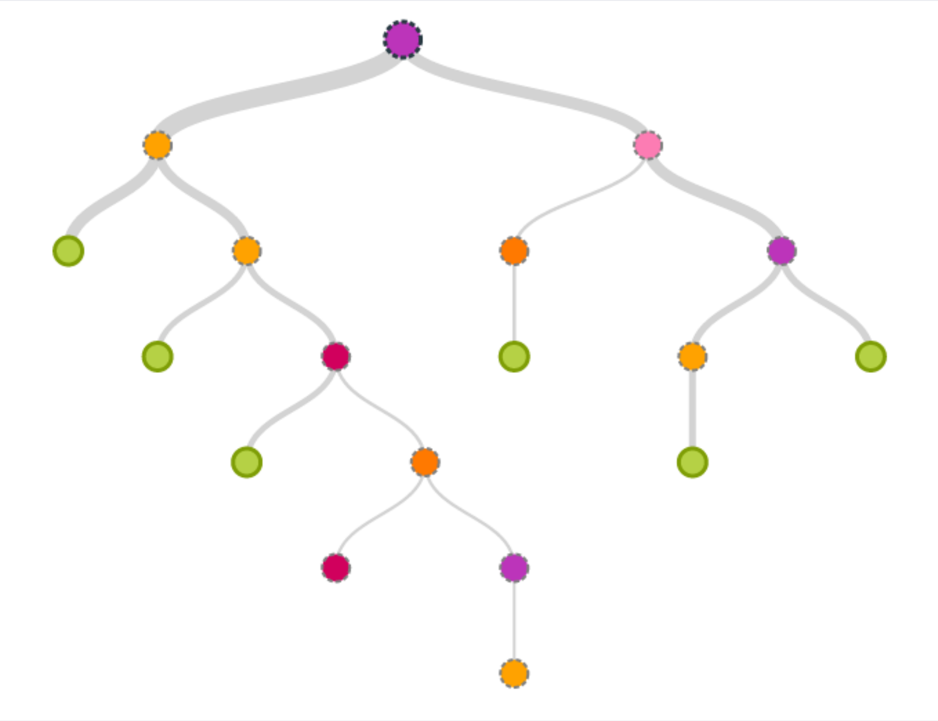


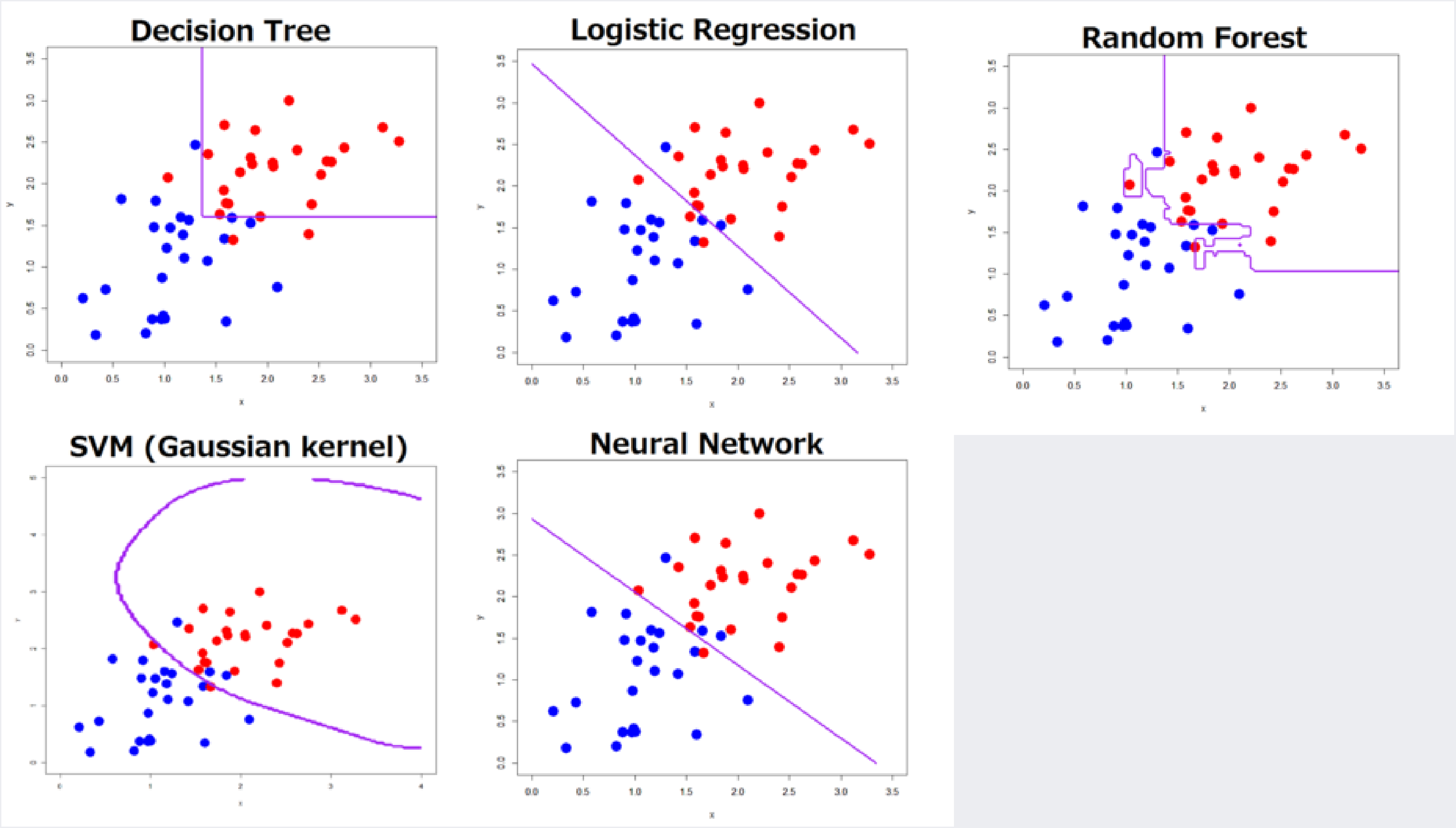
Lecture 10: Decision Tree

Pilsung Kang
School of Industrial Management Engineering
Korea University

AGENDA

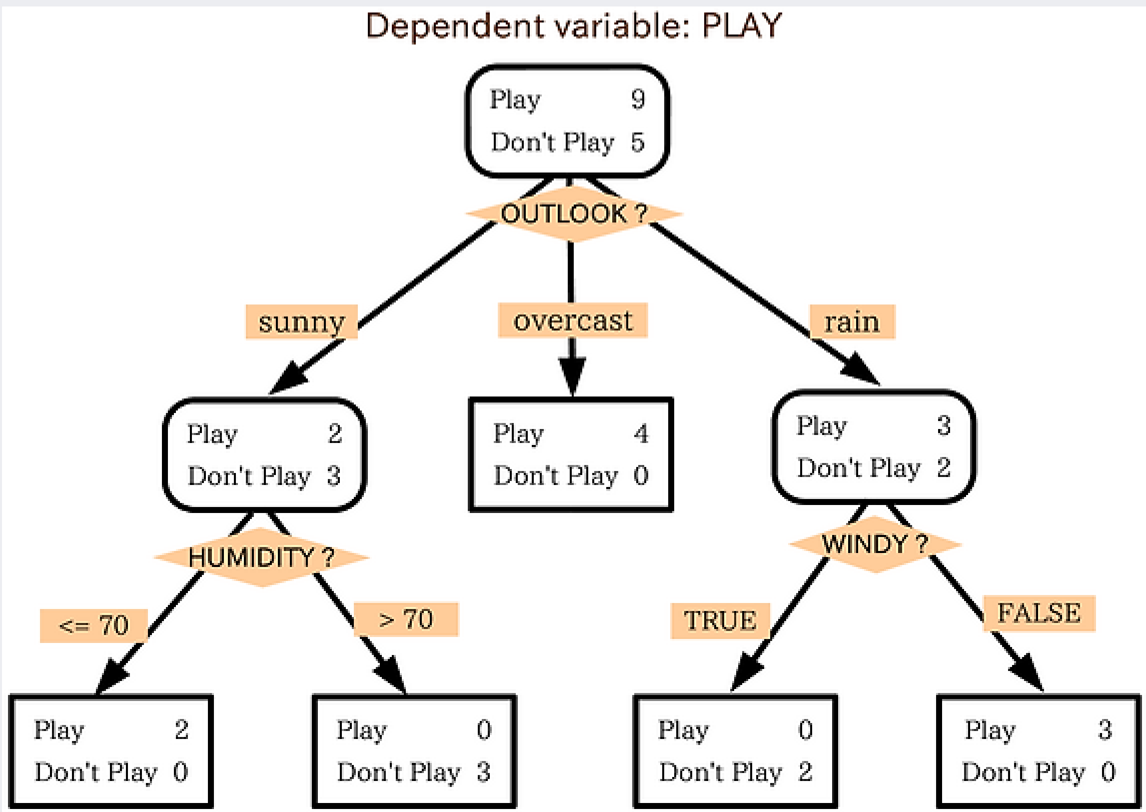
Why Are There So Many Classifiers?

- We cannot guarantee that a single classifier is always better than the others



Classification Tree

- Goal
  - ✓ Classify or predict an outcome based on a set of predictors.
  - ✓ The output is a set of rules.



Rule example

If outlook is sunny

and if humidity > 70

then he does not play

or

If outlook is rainy

and it is not windy

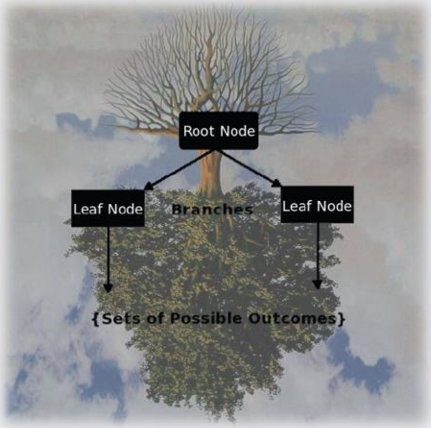
then he does play

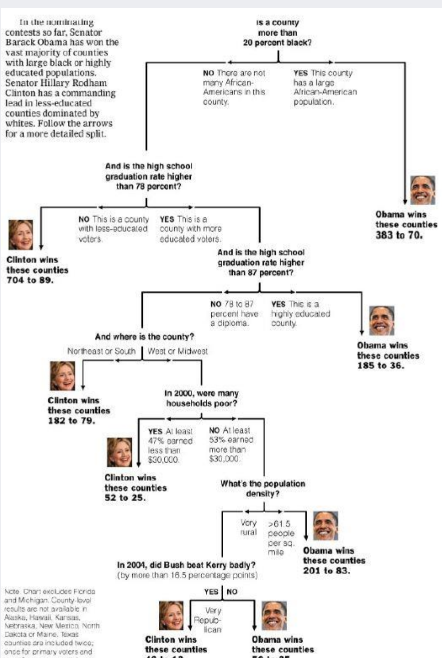
Classification Tree

- Why CART?
  - ✓ Simple to understand and interpret.
  - ✓ Requires little data preparation (normalization, missing value treatments, etc.)
  - ✓ Able to handle both numerical and categorical data.

- Key Ideas
  - ✓ Recursive Partitioning
    - Repeatedly split the records into two parts so as to achieve maximum homogeneity within the new parts.
  - ✓ Pruning the Tree
    - Simplify the tree by pruning peripheral branches to avoid over-fitting.

Classification Tree



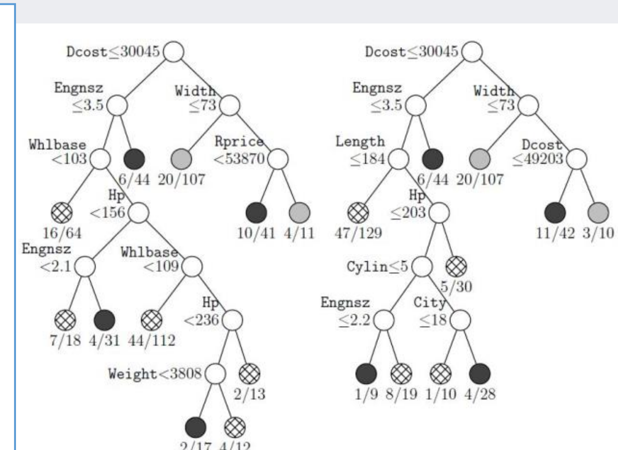


Classification and Regression Tree (CART)

- Generate a set of rules by recursively partitioning the entire datasets to increase the purity of the partitioned area (Breiman, 1984)
- Being able to explain the reason of the prediction result by following the rules to the target leaf node
- Can handle categorical and numerical variables simultaneously

Recursive Partitioning | *Pruning*

- Partition the data in a parent node into two child nodes using a certain value of a certain variable
- Select the split point to maximize the purity of the child nodes
- Gini-index (for categorical variable) and the variance (for numerical variable) are used to measure the impurity of a node

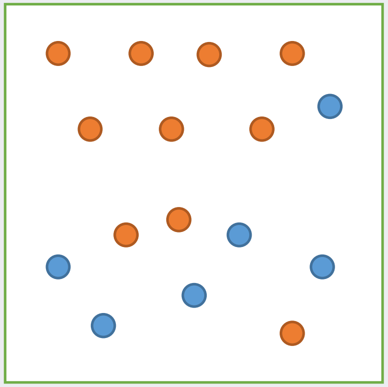


Classification Tree

- Measuring Impurity: Gini Index
  - ✓ Gini Index for rectangle A containing m records

$$I(A) = 1 - \sum_{k=1}^{m} p_k^2$$

    - p = proportion of cases in rectangle A that belong to class k.



$$I(A) = 1 - \sum_{k=1}^{m} p_k^2$$

$$= 1 - \left(\frac{6}{16}\right)^2 - \left(\frac{10}{16}\right)^2$$

$$\approx 0.47$$

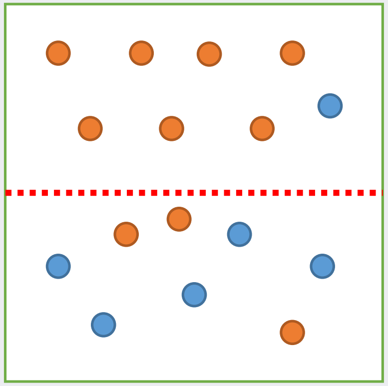
    - I(A) = 0 when all cases belong to the same class.
    - Max value when all classes are equal represented (=0.5 in binary case)

Classification Tree

- Measuring Impurity: Gini Index
  - ✓ When their more than two rectangles

$$I(A) = \sum_{i=1}^{d} \left(R_i \left(1 - \sum_{k=1}^{m} p_{ik}^2 \right) \right)$$

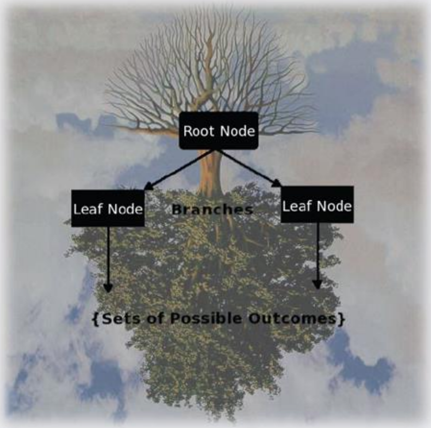
  - R_i = proportion of cases in rectangle Ri among the training data.

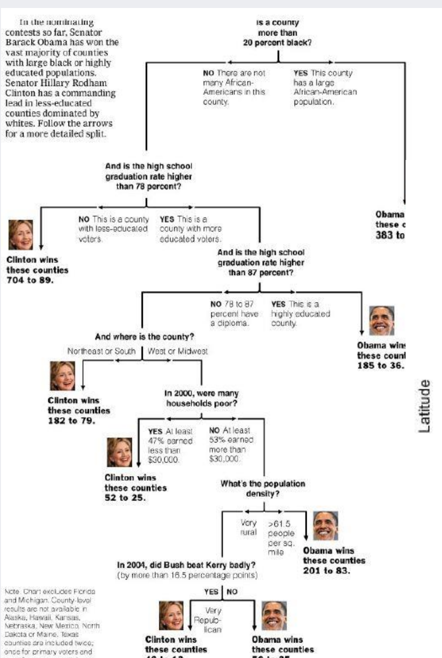


$$I(A)$$
$$= 0.5 \times \left(1 - \left(\frac{7}{8}\right)^2 - \left(\frac{1}{8}\right)^2 \right) + 0.5 \times \left(1 - \left(\frac{3}{8}\right)^2 - \left(\frac{5}{8}\right)^2 \right)$$
$$= 0.34$$

  - "Information gain" after splitting: 0.47-0.34=0.13

Classification Tree



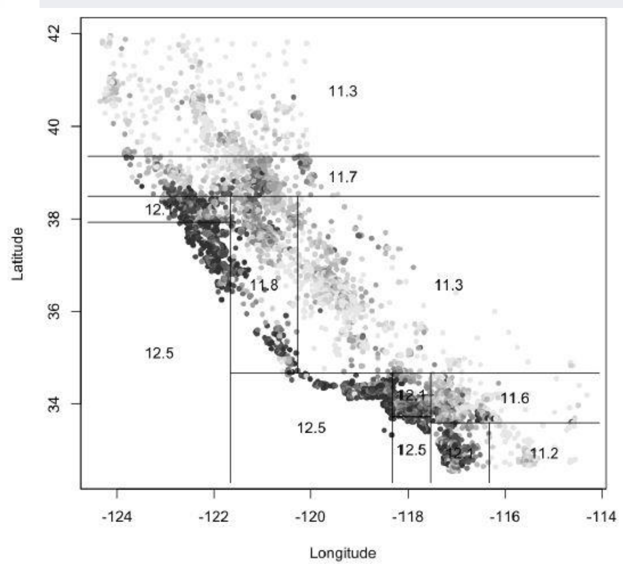


Classification and Regression Tree (CART)

- Generate a set of rules by recursively partitioning the entire datasets to increase the purity of the partitioned area (Breiman, 1984)
- Being able to explain the reason of the prediction result by following the rules to the target leaf node
- Can handle categorical and numerical variables simultaneously

Recursive Partitioning

Pruning



- Aggregate some child node into a parent node to avoid over-fitting
- Pre-pruning: pruning is done during the tree construction
- Post-pruning: Once a full-tree is constructed, nodes are pruned by taking the validation error and tree complexity

Classification Tree

- Example: Riding Mowers
  - ✓ Goal: Classify 24 households as owning or not owning riding mowers
  - ✓ Predictors: Income, Lot size

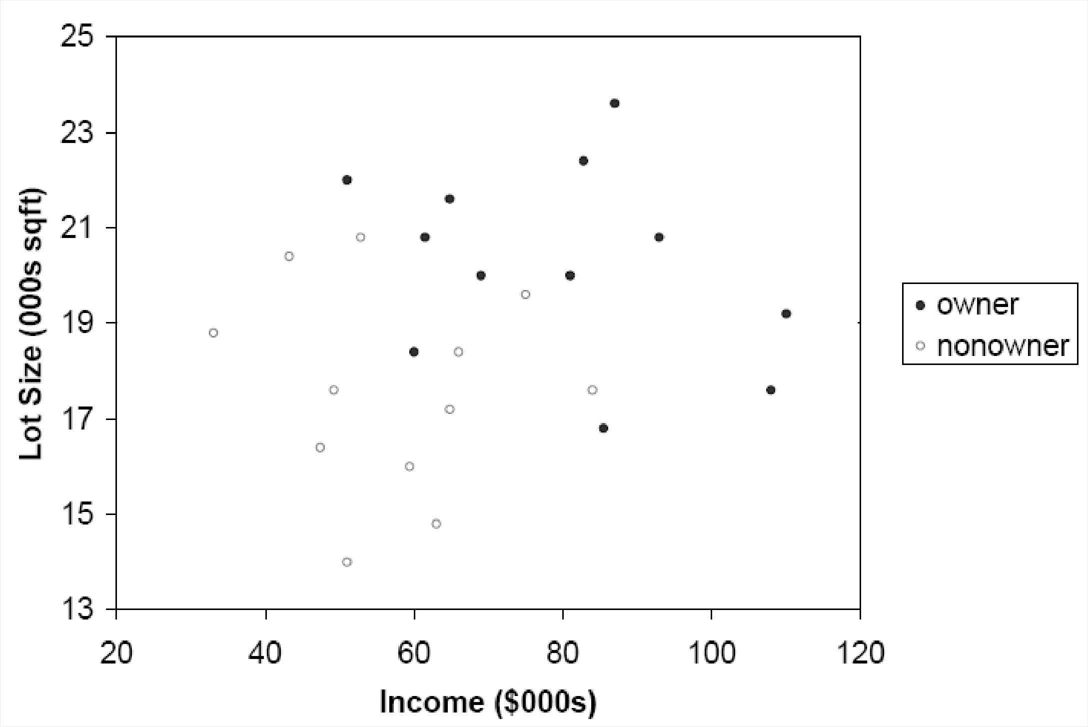
Income	Lot size	Ownership	Income	Lot size	Ownership
60.0	18.4	Owner	75.0	19.6	Non-owner
85.5	16.8	Owner	52.8	20.8	Non-owner
64.8	21.6	Owner	64.8	17.2	Non-owner
61.5	20.8	Owner	43.2	20.4	Non-owner
87.0	23.6	Owner	84.0	17.6	Non-owner
110.1	19.2	Owner	49.2	17.6	Non-owner
108.0	17.6	Owner	59.4	16.0	Non-owner
82.8	22.4	Owner	66.0	18.4	Non-owner
69.0	20.0	Owner	47.4	16.4	Non-owner
93.0	20.8	Owner	33.0	18.8	Non-owner
51.0	22.0	Owner	51.0	14.0	Non-owner
81.0	20.0	Owner	63.0	14.8	Non-owner

Classification Tree

1

Order records according to one variable

- Order the data with regard to lot size



Income	Lot size	Ownership
51.0	14.0	Non-owner
63.0	14.8	Non-owner
59.4	16.0	Non-owner
47.4	16.4	Non-owner
85.5	16.8	Owner
64.8	17.2	Non-owner
108.0	17.6	Owner
84.0	17.6	Non-owner
49.2	17.6	Non-owner
60.0	18.4	Owner
66.0	18.4	Non-owner
33.0	18.8	Non-owner
110.1	19.2	Owner
75.0	19.6	Non-owner
69.0	20.0	Owner
81.0	20.0	Owner
43.2	20.4	Non-owner
61.5	20.8	Owner
93.0	20.8	Owner
52.8	20.8	Non-owner
64.8	21.6	Owner
51.0	22.0	Owner
82.8	22.4	Owner
87.0	23.6	Owner

Classification Tree

2

Find midpoints between successive values

- First midpoint = 14.4 (0.5*(14.0+14.8))
- Divide records into those with Lot size > 14.4 and those < 14.4
- Compute the impurity: Gini index
  - ✓ Before splitting:

$$1-\left(\frac{12}{24}\right)^2-\left(\frac{12}{24}\right)^2=0.5$$

  - ✓ After splitting:

$$\frac{1}{24}\left(1-\left(\frac{1}{1}\right)^2\right)+\frac{23}{24}\left(1-\left(\frac{12}{23}\right)^2-\left(\frac{11}{23}\right)^2\right)\approx 0.48$$

  - ✓ Information gain: 0.50-0.48=0.02

Income	Lot size	Ownership
51.0	14.0	Non-owner
63.0	14.8	Non-owner
59.4	16.0	Non-owner
47.4	16.4	Non-owner
85.5	16.8	Owner
64.8	17.2	Non-owner
108.0	17.6	Owner
84.0	17.6	Non-owner
49.2	17.6	Non-owner
60.0	18.4	Owner
66.0	18.4	Non-owner
33.0	18.8	Non-owner
110.1	19.2	Owner
75.0	19.6	Non-owner
69.0	20.0	Owner
81.0	20.0	Owner
43.2	20.4	Non-owner
61.5	20.8	Owner
93.0	20.8	Owner
52.8	20.8	Non-owner
64.8	21.6	Owner
51.0	22.0	Owner
82.8	22.4	Owner
87.0	23.6	Owner

Classification Tree

2

Find midpoints between successive values

- First midpoint = 14.4 (0.5*(14.0+14.8))
- Divide records into those with Lot size > 14.4 and those < 14.4
- Compute the impurity: Entropy
  - ✓ Before splitting:

$$-\frac{1}{2}\log_2\left(\frac{1}{2}\right)-\frac{1}{2}\log_2\left(\frac{1}{2}\right)=1$$

  - ✓ After splitting:

$$\frac{1}{24}(-\log(1))+\frac{23}{24}\left(-\frac{12}{23}\log_2\left(\frac{12}{23}\right)-\frac{11}{23}\log_2\left(\frac{11}{23}\right)\right)\approx 0.96$$

  - ✓ Information gain: 1-0.96=0.04

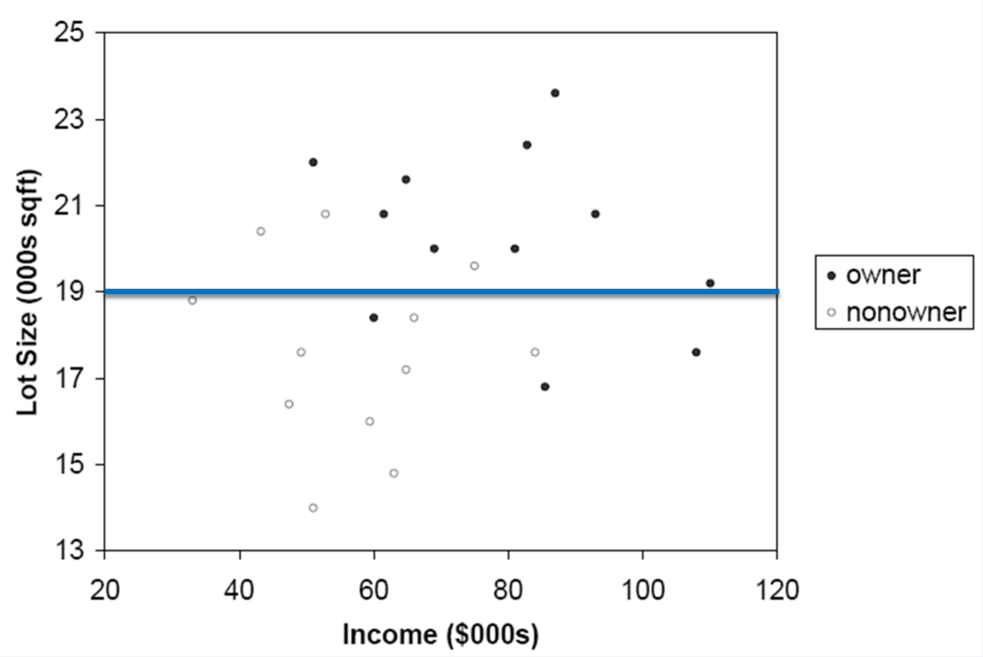
Income	Lot size	Ownership
51.0	14.0	Non-owner
63.0	14.8	Non-owner
59.4	16.0	Non-owner
47.4	16.4	Non-owner
85.5	16.8	Owner
64.8	17.2	Non-owner
108.0	17.6	Owner
84.0	17.6	Non-owner
49.2	17.6	Non-owner
60.0	18.4	Owner
66.0	18.4	Non-owner
33.0	18.8	Non-owner
110.1	19.2	Owner
75.0	19.6	Non-owner
69.0	20.0	Owner
81.0	20.0	Owner
43.2	20.4	Non-owner
61.5	20.8	Owner
93.0	20.8	Owner
52.8	20.8	Non-owner
64.8	21.6	Owner
51.0	22.0	Owner
82.8	22.4	Owner
87.0	23.6	Owner

Classification Tree

3

Find the best split

- Find the best split which maximize the (Gini or information gain)

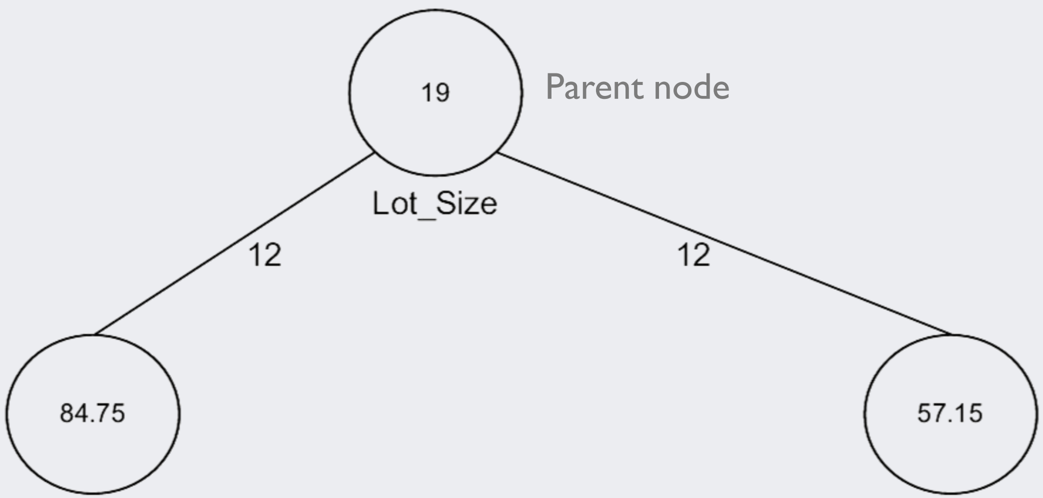


Income	Lot size	Ownership
51.0	14.0	Non-owner
63.0	14.8	Non-owner
59.4	16.0	Non-owner
47.4	16.4	Non-owner
85.5	16.8	Owner
64.8	17.2	Non-owner
108.0	17.6	Owner
84.0	17.6	Non-owner
49.2	17.6	Non-owner
60.0	18.4	Owner
66.0	18.4	Non-owner
33.0	18.8	Non-owner
110.1	19.2	Owner
75.0	19.6	Non-owner
69.0	20.0	Owner
81.0	20.0	Owner
43.2	20.4	Non-owner
61.5	20.8	Owner
93.0	20.8	Owner
52.8	20.8	Non-owner
64.8	21.6	Owner
51.0	22.0	Owner
82.8	22.4	Owner
87.0	23.6	Owner

Classification Tree

3

Tree structure



- Split point become nodes on tree (circles with split value in center)
- Rectangles represent "leaves" (terminal points, no future splits, classification value noted)
- Numbers on lines between nodes indicate # cases.

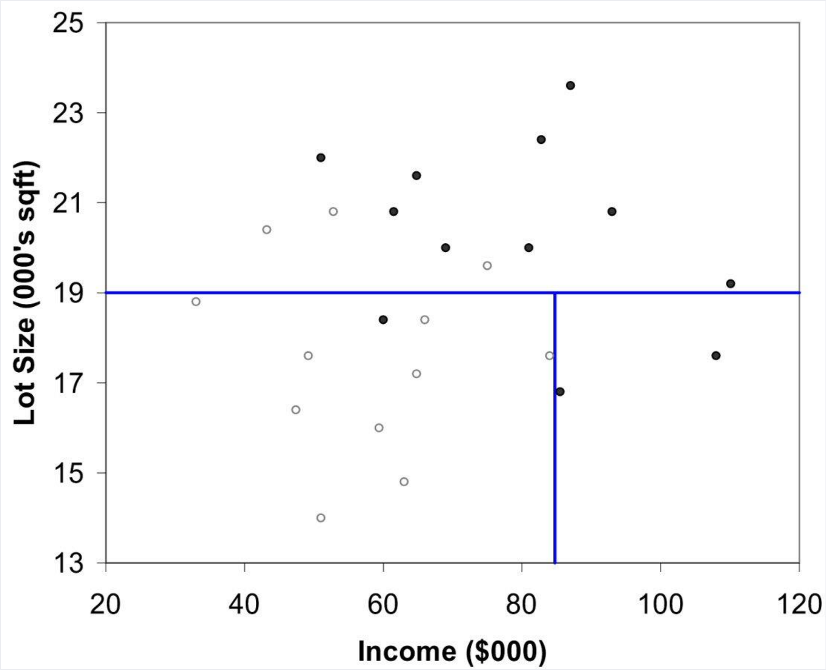
Income	Lot size	Ownership
51.0	14.0	Non-owner
63.0	14.8	Non-owner
59.4	16.0	Non-owner
47.4	16.4	Non-owner
85.5	16.8	Owner
64.8	17.2	Non-owner
108.0	17.6	Owner
84.0	17.6	Non-owner
49.2	17.6	Non-owner
60.0	18.4	Owner
66.0	18.4	Non-owner
33.0	18.8	Non-owner
110.1	19.2	Owner
75.0	19.6	Non-owner
69.0	20.0	Owner
81.0	20.0	Owner
43.2	20.4	Non-owner
61.5	20.8	Owner
93.0	20.8	Owner
52.8	20.8	Non-owner
64.8	21.6	Owner
51.0	22.0	Owner
82.8	22.4	Owner
87.0	23.6	Owner

Classification Tree

4

Repeat the splitting for each node

- Repeat the splitting until there is no gain.
- E.g., second split = income = 84.75

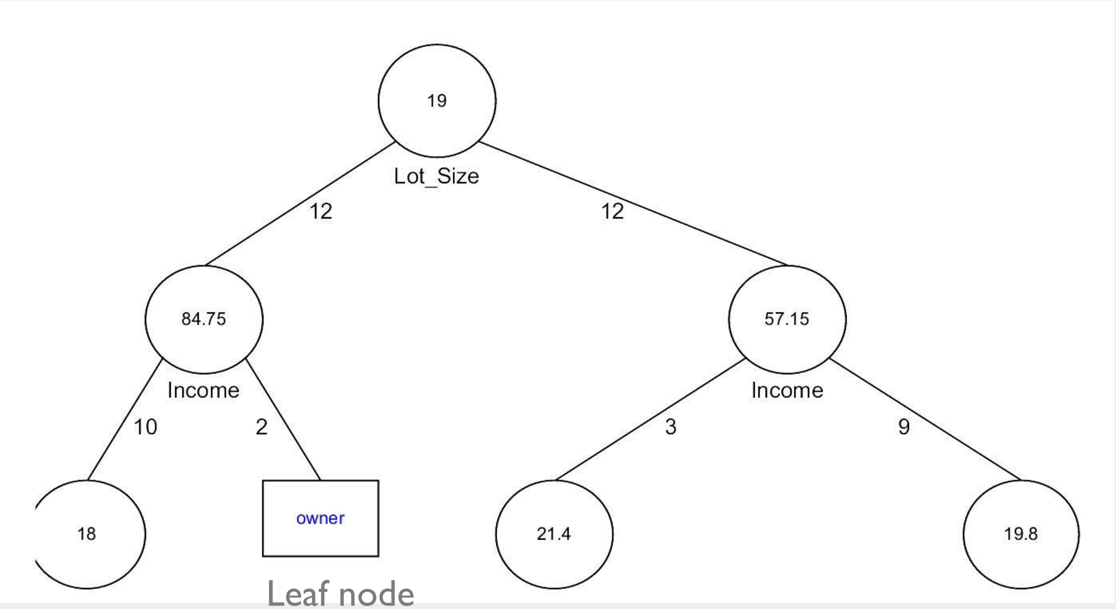


Income	Lot size	Ownership
33.0	18.8	Non-owner
47.4	16.4	Non-owner
49.2	17.6	Non-owner
51.0	14.0	Non-owner
59.4	16.0	Non-owner
60.0	18.4	Owner
63.0	14.8	Non-owner
64.8	17.2	Non-owner
66.0	18.4	Non-owner
84.0	17.6	Non-owner
85.5	16.8	Owner
108.0	17.6	Owner

Classification Tree

Repeat the splitting for each node

- Repeat the splitting until there is no gain.
- E.g., second split = income = 84.75



Leaf node

4

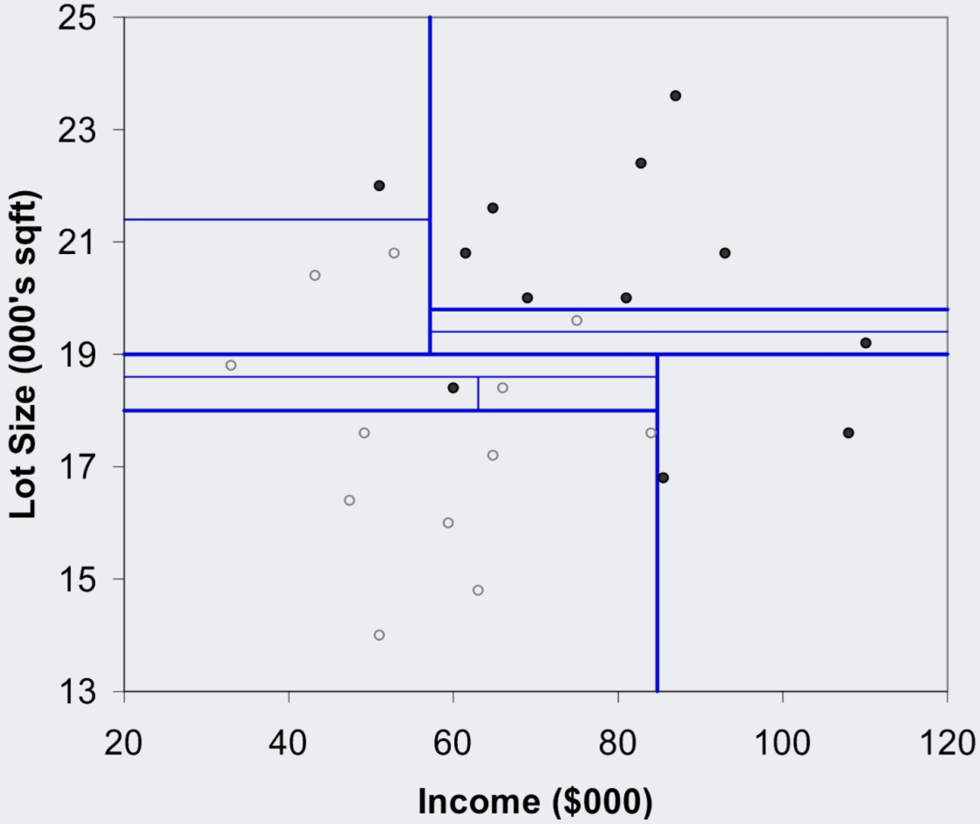
Income	Lot size	Ownership
33.0	18.8	Non-owner
47.4	16.4	Non-owner
49.2	17.6	Non-owner
51.0	14.0	Non-owner
59.4	16.0	Non-owner
60.0	18.4	Owner
63.0	14.8	Non-owner
64.8	17.2	Non-owner
66.0	18.4	Non-owner
84.0	17.6	Non-owner
85.5	16.8	Owner
108.0	17.6	Owner

Classification Tree

4

Repeat the splitting for each node

- Repeat the splitting until there is no gain.
- Final splitting



Classification Tree

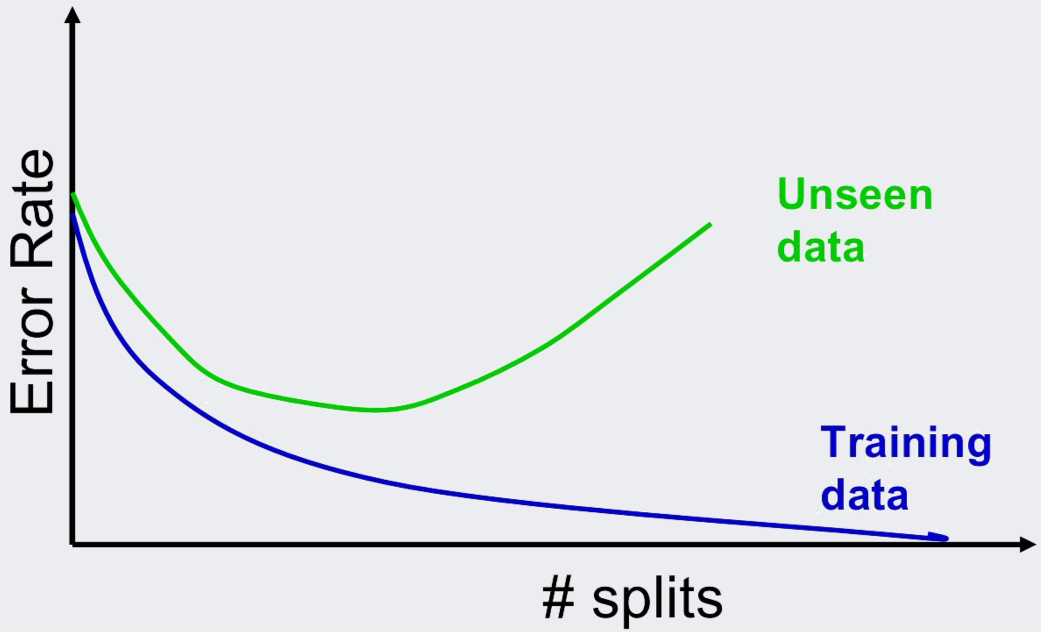
4

Repeat the splitting for each node

- Each leaf node label is determined by “voting” of the records within it, and by the cutoff value.
- Records within each leaf node are from the training data.
- Default cutoff=0.5 means that the leaf node’s label is the majority class.
- Cutoff = 0.75 requires majority of 75% of more “1” records in the leaf to label it a “1” node.

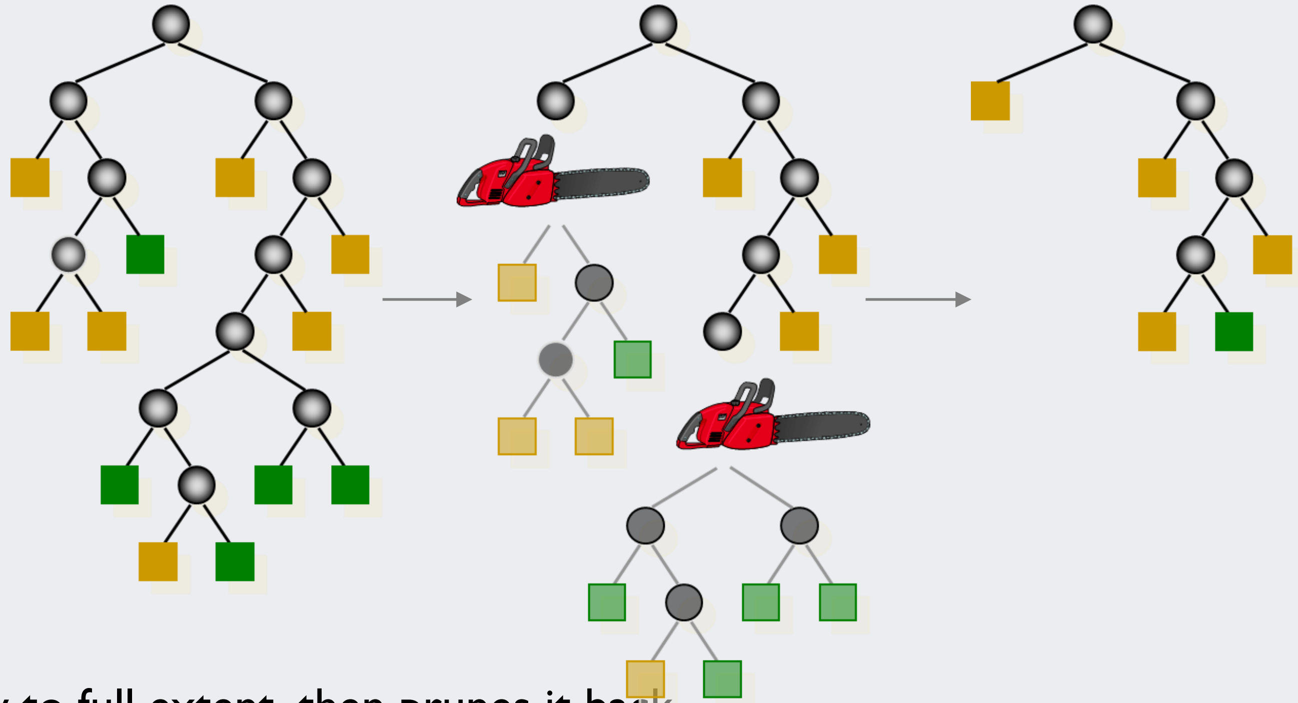
Classification Tree

- Overfitting problem
  - ✓ The end of recursive partitioning process is 100% purity in each leaf
  - ✓ It over-fits the data, ending up fitting noise in the data and leading to low predictive accuracy of new data
  - ✓ Past a certain point, the error rate for the validation data starts to increase



Classification Tree

- Pruning



✓ CART lets tree grow to full extent, then prunes it back.

✓ Idea is to find that point at which the validation error begins to rise.

✓ Generate successively smaller trees by pruning leaves.

✓ At each pruning stage, multiple trees are possible.

✓ Use "cost complexity" to choose the best tree at that stage.

Classification Tree

- Cost complexity

$$CC(T) = Err(T) + \alpha \times L(T)$$

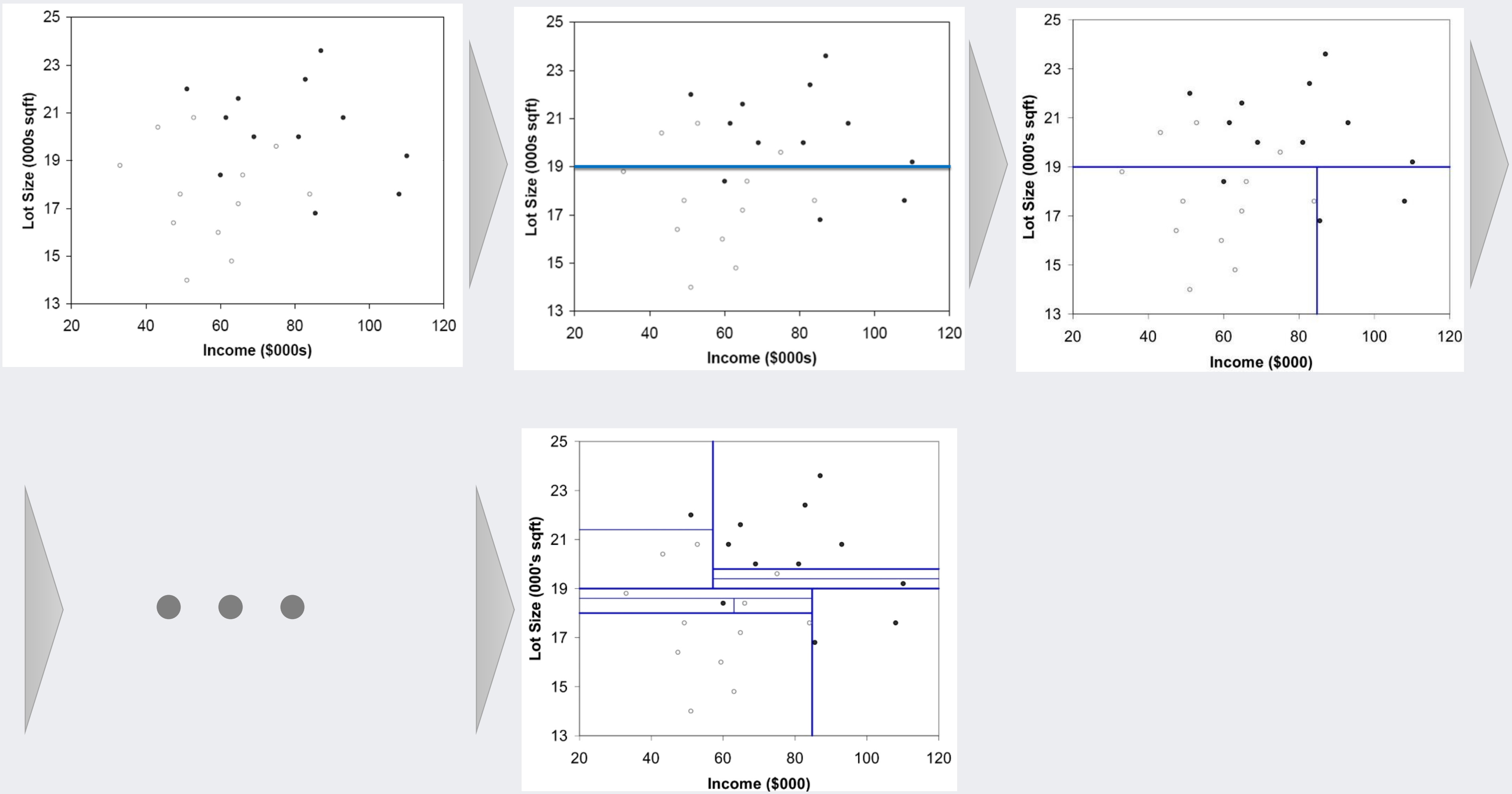
✓ CC(T) = cost complexity of a tree

✓ ERR(T) = proportion of misclassified records in the validation data

✓ Alpha = penalty factor attached to the tree size (set by the user)

Classification Tree

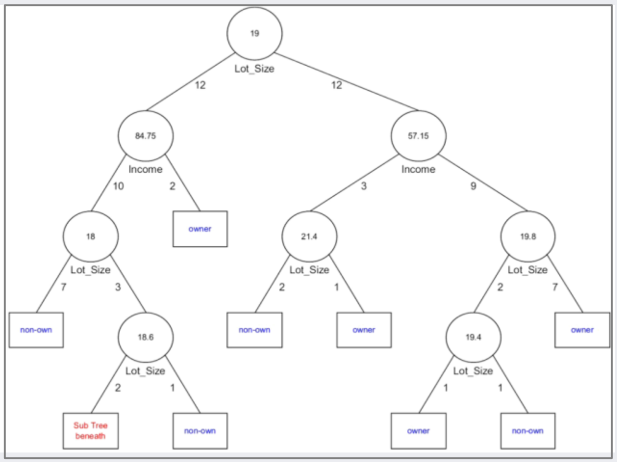
- Full tree constructed by recursive partitioning



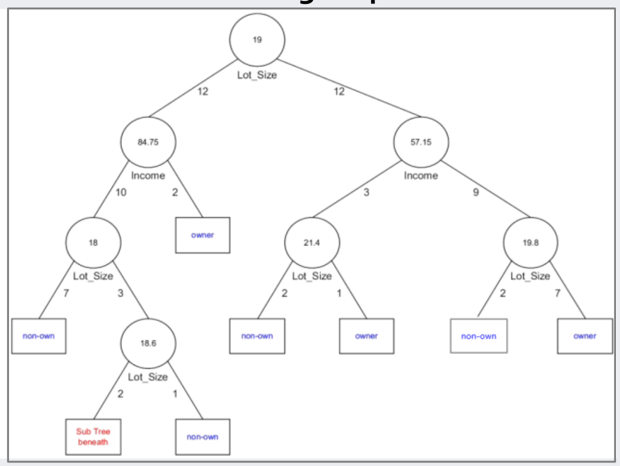
Classification Tree

- Pruning

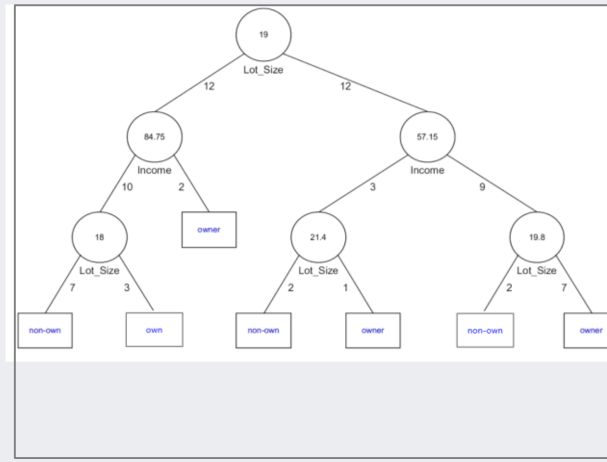
Full Tree

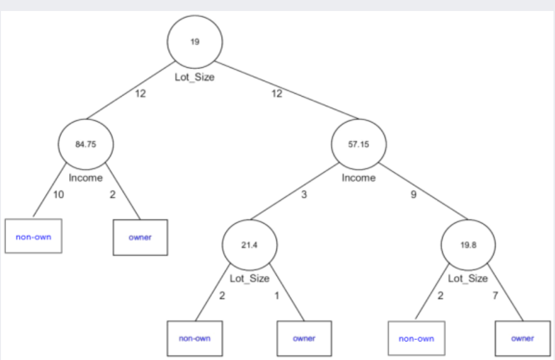


Pruning Step 1

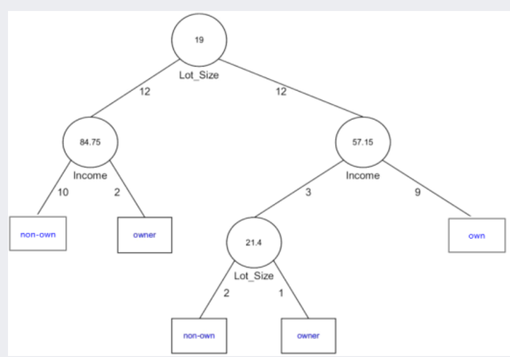


Pruning Step 2

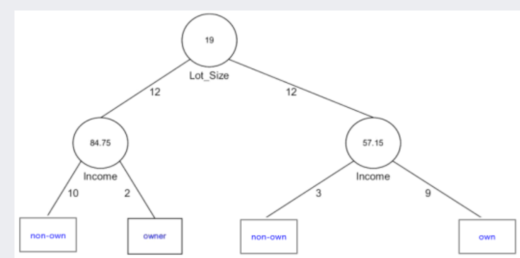




Pruning Step 3



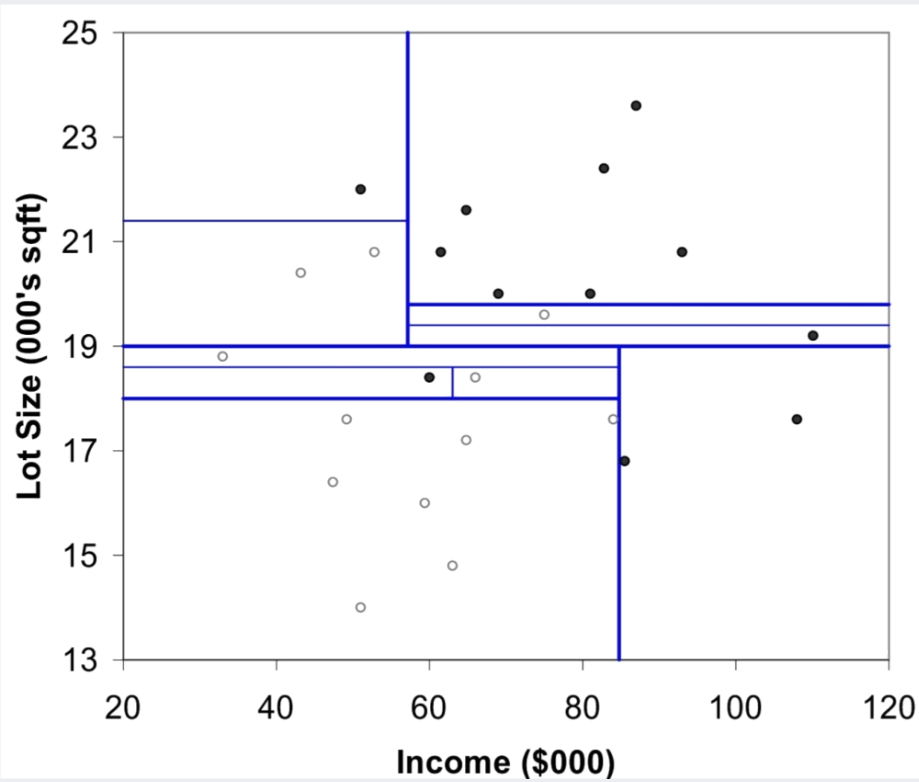
Pruning Step 4

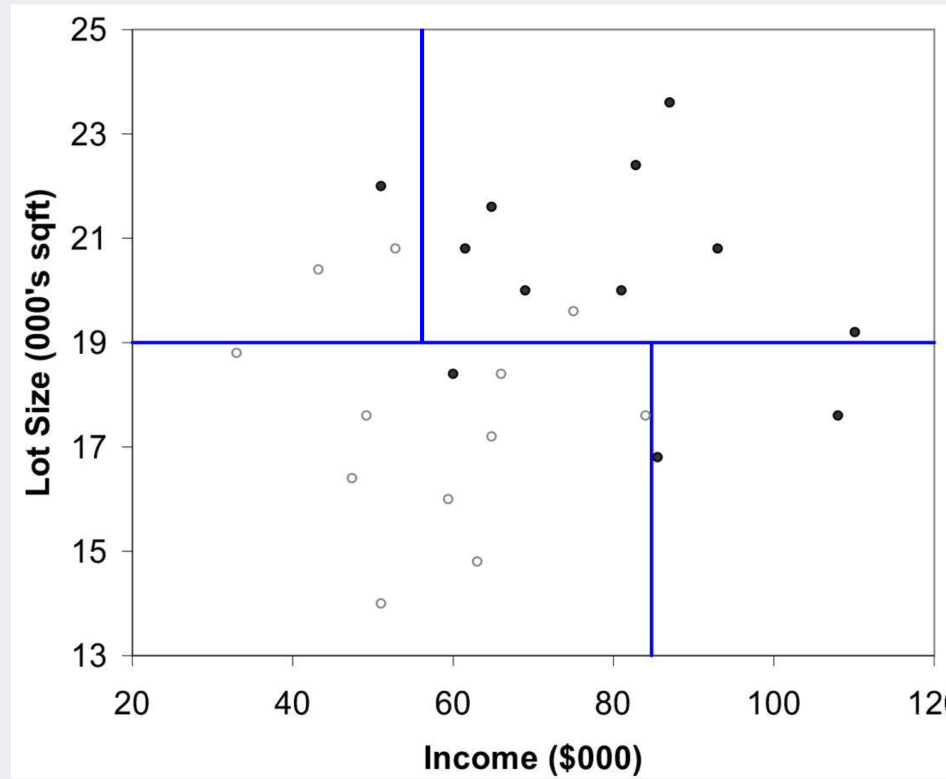


Pruned Tree

Classification Tree

- Full tree vs. Pruned tree





Classification Tree

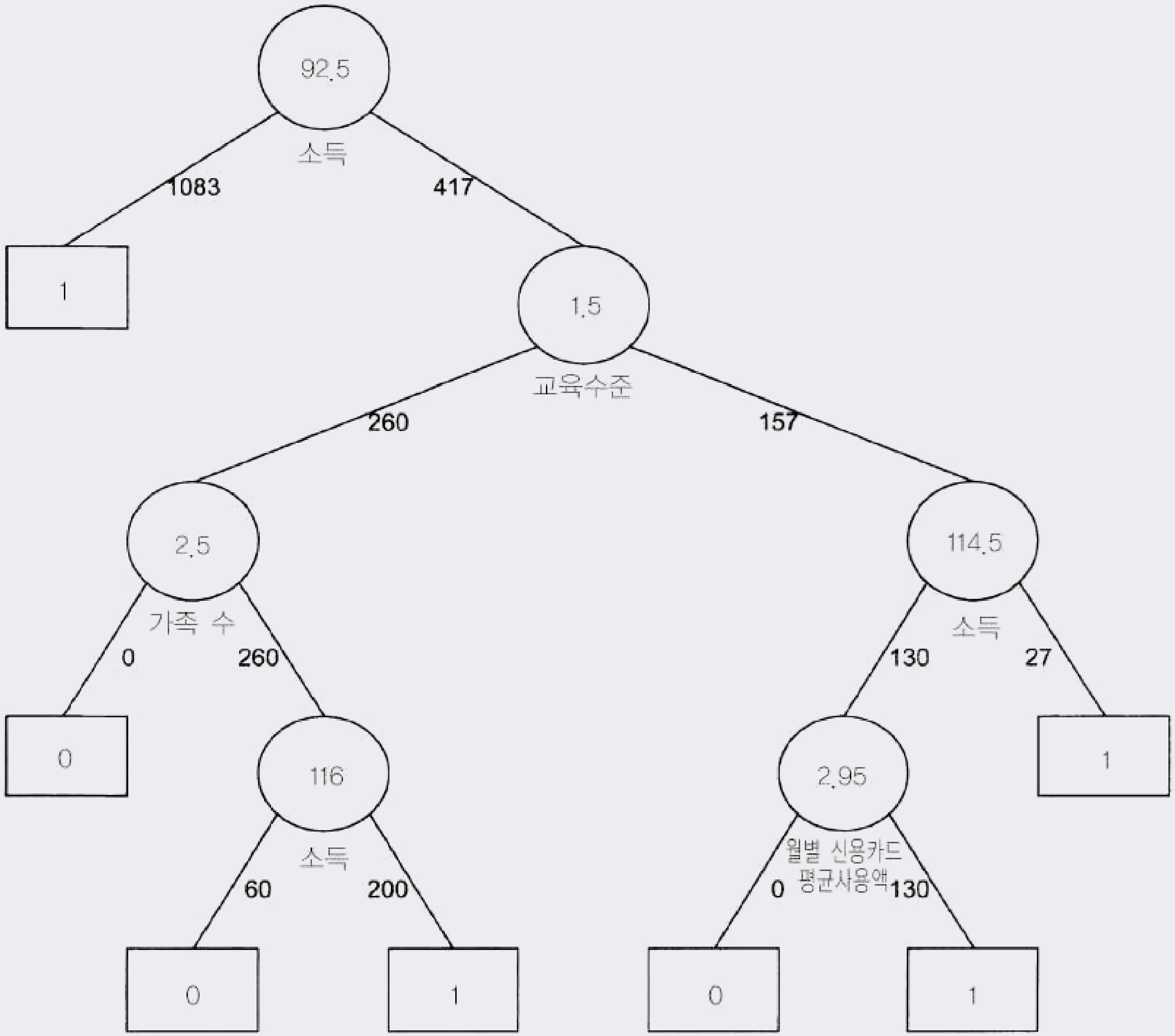
- Example: Universal bank
  - ✓ Goal: to analyze what combination of factors make a customer more likely to accept a personal loan

일련 번호	나이	경력	소득	가족 수	월별 신용카드 평균사용액	교육 수준	담보부 채권	개인 대출	증권 계좌	CD 계좌	온라인 뱅킹	신용 카드
1	25	1	49	4	1.60	UG	0	No	Yes	No	No	No
2	45	19	34	3	1.50	UG	0	No	Yes	No	No	No
3	39	15	11	1	1.00	UG	0	No	No	No	No	No
4	35	9	100	1	2.70	Grad	0	No	No	No	No	No
5	35	8	45	4	1.00	Grad	0	No	No	No	No	Yes
6	37	13	29	4	0.40	Grad	155	No	No	No	Yes	No
7	53	27	72	2	1.50	Grad	0	No	No	No	Yes	No
8	50	24	22	1	0.30	Prof	0	No	No	No	No	Yes
9	35	10	81	3	0.60	Grad	104	No	No	No	Yes	No
10	34	9	180	1	8.90	Prof	0	Yes	No	No	No	No
11	65	39	105	4	2.40	Prof	0	No	No	No	No	No
12	29	5	45	3	0.10	Grad	0	No	No	No	Yes	No
13	48	23	114	2	3.80	Prof	0	No	Yes	No	No	No
14	59	32	40	4	2.50	Grad	0	No	No	No	Yes	No
15	67	41	112	1	2.00	UG	0	No	Yes	No	No	No
16	60	30	22	1	1.50	Prof	0	No	No	No	Yes	Yes
17	38	14	130	4	4.70	Prof	134	Yes	No	No	No	No
18	42	18	81	4	2.40	UG	0	No	No	No	No	No
19	46	21	193	2	8.10	Prof	0	Yes	No	No	No	No
20	55	28	21	1	0.50	Grad	0	No	Yes	No	No	Yes

Classification Tree

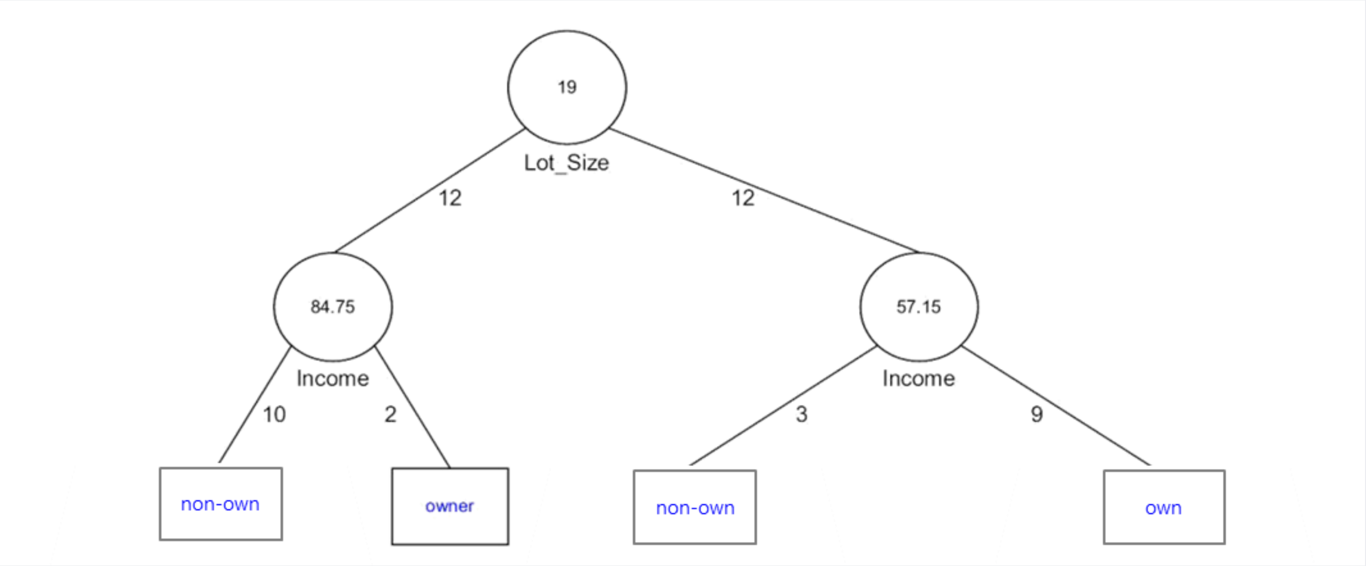
의사결정 마디	학습용 집합의 오차율	평가용 집합의 오차율			
41	0	2.133333			
40	0.04	2.2			
39	0.08	2.2			
38	0.12	2.2			
37	0.16	2.066667			
36	0.2	2.066667			
35	0.2	2.066667			
34	0.24	2.066667			
...	...	...			
13	1.16	1.6			
12	1.2	1.6			
11	1.2	1.466667	최소 오차 나무	표준오차	0.003103929
10	1.6	1.666667			
9	2.2	1.666667			
8	2.2	1.866667			
7	2.24	1.866667			
6	2.24	1.6	<-- 최적의 가지친 나무		
5	4.44	1.8			
4	5.08	2.333333			
3	5.24	3.466667			
2	9.4	9.533333			
1	9.4	9.533333			
0	9.4	9.533333			

Classification Tree



Classification Tree

- Generating the rules from the pruned tree



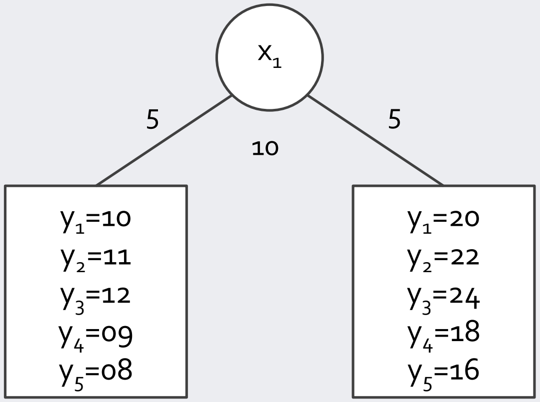
- IF(Lot size < 19) AND IF(Income < 84.75) THEN Owner = No
- IF(Lot size < 19) AND IF(Income > 84.75) THEN Owner = YES
- IF(Lot size > 19) AND IF(Income < 57.15) THEN Owner = NO
- IF(Lot size > 19) AND IF(Income > 57.15) THEN Owner = YES

AGENDA

Regression Tree

Similar process with classification tree except

- Prediction of the node
  - ✓ The average of the outcome variables belonging to the node



- Predicted value of the left leaf node = 10
- Predicted value of the right leaf node = 20

- Impurity
  - ✓ Sum of squared error (SSE: $\sum_{i=1}^{n}(y_i - \hat{y})^2$)
  - ✓ SSE(Parent) = 300, SSE(Left) = 10, SSE(Right) = 40, Gain = 250

Regression Tree

- Predict the selling price of Toyota corolla

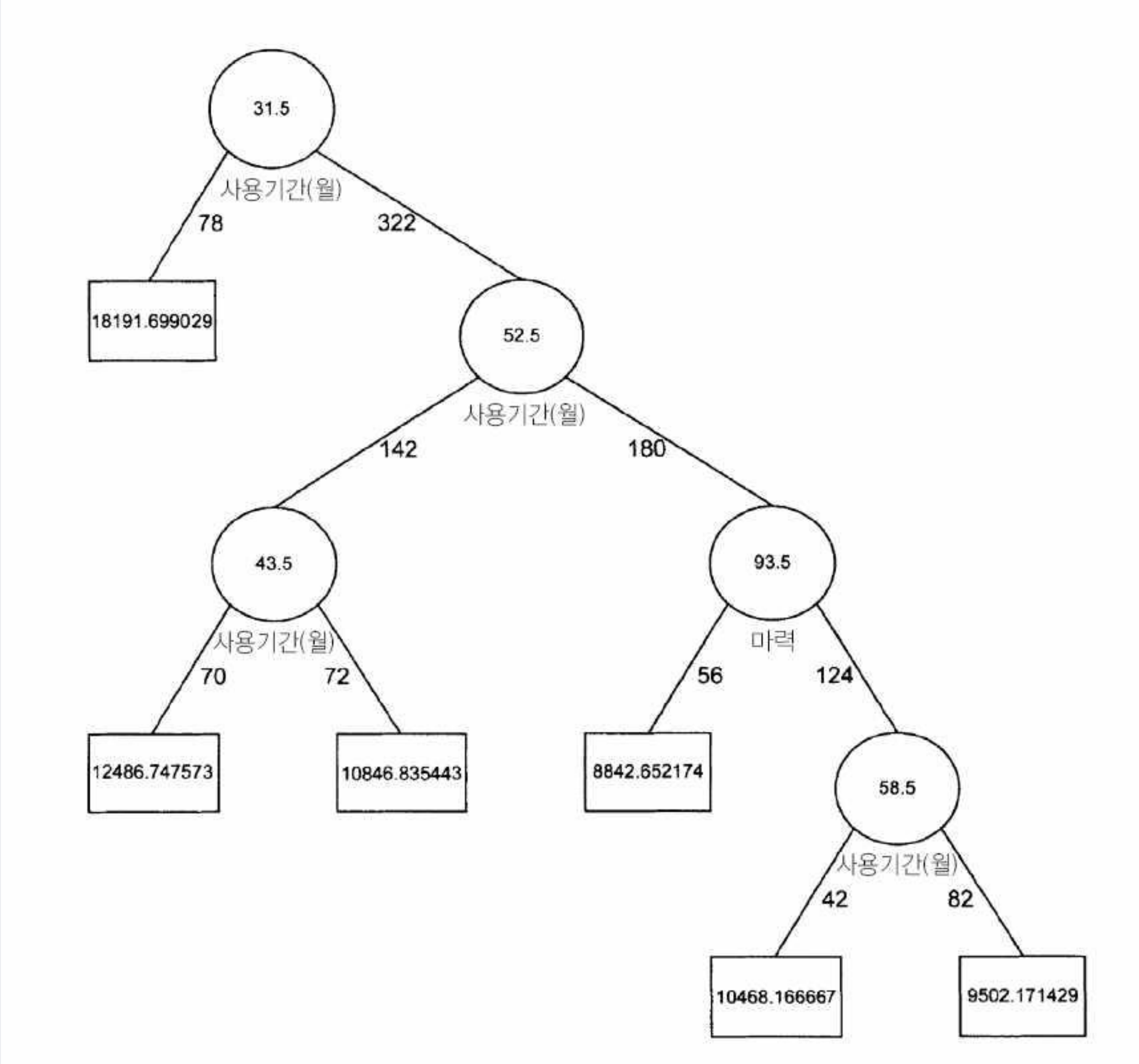

Dependent variable (target)

Independent variables (attributes, features)

Variable	Description
Price	Offer Price in EUROs
Age_08_04	Age in months as in August 2004
KM	Accumulated Kilometers on odometer
Fuel_Type	Fuel Type (Petrol, Diesel, CNG)
HP	Horse Power
Met_Color	Metallic Color? (Yes=1, No=0)
Automatic	Automatic ((Yes=1, No=0)
CC	Cylinder Volume in cubic centimeters
Doors	Number of doors
Quarterly_Tax	Quarterly road tax in EUROs
Weight	Weight in Kilograms

Regression Tree

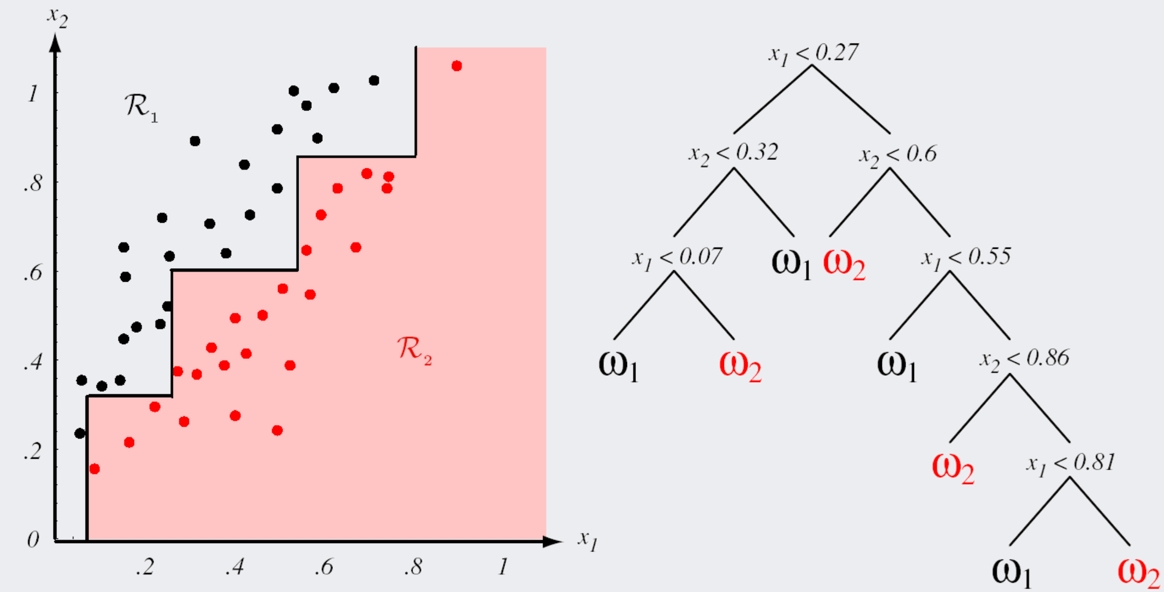
- Pruned Tree

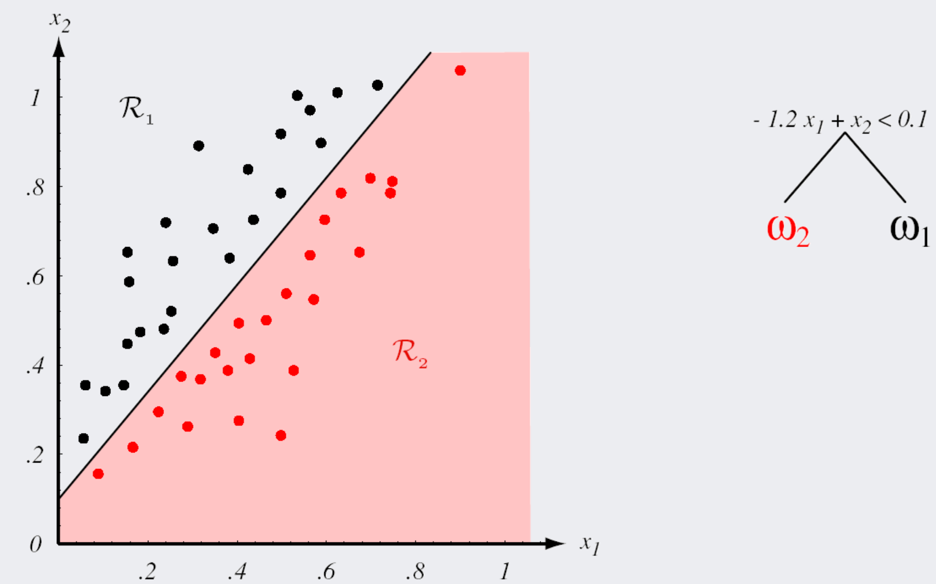


CART: Summary

- Advantages
  - ✓ Classification and regression tree (CART) is easy to use and understand
  - ✓ Produce rules that are easy to interpret & implement
  - ✓ Variable selection & reduction is automatic
  - ✓ Do not require the assumptions of statistical models
  - ✓ Can work without extensive handling of missing data
- Disadvantages
  - ✓ May not perform well where there is structure in the data that is not well captured by horizontal or vertical split
  - ✓ Since the process deals with "one variable at a time", no way to capture interactions between variables

CART: Summary





AGENDA

R Exercise: Data Set

- Personal Loan
  - ✓ Purpose: identify future customer who will use the personal loan service based on his/her demographic information and banking service history

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal L	Securities	CD Accou	Online	CreditCard
2	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
3	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
4	3	39	15	11	94720	1	1	1	0	0	0	0	0	0
5	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
6	5	35	8	45	91330	4	1	2	0	0	0	0	0	1
7	6	37	13	29	92121	4	0.4	2	155	0	0	0	1	0
8	7	53	27	72	91711	2	1.5	2	0	0	0	0	1	0
9	8	50	24	22	93943	1	0.3	3	0	0	0	0	0	1
10	9	35	10	81	90089	3	0.6	2	104	0	0	0	1	0

    - A total of 14 variables (columns)
    - ID, ZIP Code: irrelevant column (remove)
    - Personal loan: target variable

R Exercise: Preprocessing

- Data: Personal loan prediction
  - ✓ Write a performance evaluation function
  - ✓ Load the data
  - ✓ Use the "party" package
  - ✓ Transform the target variable as "factor" type
  - ✓ Divide the dataset into the training (70%) and validation (30%)

R Exercise: Preprocessing

- Install packages & write a performance evaluation function

```
# For CART
install.packages("party")
library(party)

# Performance Evaluation Function -------------------------------------
perf_eval <- function(cm){
     # True positive rate: TPR (Recall)
     TPR <- cm[2,2]/sum(cm[2,])
     # Precision
     PRE <- cm[2,2]/sum(cm[,2])
     # True negative rate: TNR
     TNR <- cm[1,1]/sum(cm[1,])
     # Simple Accuracy
     ACC <- (cm[1,1]+cm[2,2])/sum(cm)
     # Balanced Correction Rate
     BCR <- sqrt(TPR*TNR)
     # F1-Measure
     F1 <- 2*TPR*PRE/(TPR+PRE)

     return(c(TPR, PRE, TNR, ACC, BCR, F1))
}
```

R Exercise: Preprocessing

- Load the dataset and divide it to the training (70%) and test (30%) datasets

```
# Classification and Regression Tree (CART) -------------------------------
# Personal Loan Prediction
ploan <- read.csv("Personal Loan.csv")

ploan.x <- ploan[,-c(1,5,10)]
ploan.y <- as.data.frame(as.factor(ploan[,10]))

trn_idx <- sample(1:dim(ploan.y)[1], round(0.7*dim(ploan.y)[1]))

ploan.trn <- cbind(ploan.x[trn_idx,], ploanYN = ploan.y[trn_idx,])
ploan.val <- cbind(ploan.x[-trn_idx,], ploanYN = ploan.y[-trn_idx,])
ploan.all <- rbind(ploan.trn, ploan.val)
```

R Exercise: Preprocessing

- Hyper-parameter initialization

```r
# construct single tree and evaluation # tree parameter settings
min_criterion = c(0.9, 0.95, 0.99)
min_split = c(10, 30, 50, 100)
max_depth = c(0, 10, 5)

tree_result =
matrix(0,length(min_criterion)*length(min_split)*length(max_depth),10)

colnames(tree_result) <- c("min_criterion", "min_split", "max_depth", "TPR",
"Precision", "TNR", "ACC", "BCR", "F1", "N_leaves")
```

  - ✓ min_criterion: statistical significance to split a node
  - ✓ min_split: minimum number of instances in the current node to be split
  - ✓ max_depth: maximum depth of the tree

R Exercise: Parameter Tuning

- Find the optimal parameters

```
iter_cnt = 1

for (i in 1:length(min_criterion)) {
     for ( j in 1:length(min_split)) {
          for ( k in 1:length(max_depth)) {

               cat("CART Min criterion:", min_criterion[i], ", Min split:",
                    min_split[j], ", Max depth:", max_depth[k], "\n")
```

✓ Run for loop for three difference model parameters

✓ cat( ): print the strings in the console

R Exercise: Parameter Tuning

- Find the optimal parameters

```
tmp_control = ctree_control(mincriterion = min_criterion[i],
                            minsplit = min_split[j], maxdepth = max_depth[k])

tmp_tree <- ctree(ploanYN ~ ., data = ploan.trn, controls = tmp_control)

tmp_tree_val_prediction <- predict(tmp_tree, newdata = ploan.val)

tmp_tree_val_cm <- table(ploan.val$ploanYN, tmp_tree_val_prediction)
```

✓ ctree( ): training a classification tree

- Arg 1: Formula
- Arg 2: Dataset for training
- Arg 3: Parameter values

✓ predict( ): make predictions

- Arg 1: Trained model
- Arg 2: Dataset to predict

R Exercise: Parameter Tuning

- Find the optimal parameters

```
# parameters
tree_result[iter_cnt,1] <- min_criterion[i]
tree_result[iter_cnt,2] <- min_split[j]
tree_result[iter_cnt,3] <- max_depth[k]
tree_result[iter_cnt, 4:9] <- perf_eval(tmp_tree_val_cm)

# Number of leaf nodes
tree_result[iter_cnt,10] = length(nodes(tmp_tree, unique(where(tmp_tree))))
iter_cnt = iter_cnt + 1
```

✓ Store the parameters and corresponding performance values

R Exercise: Parameter Tuning

- Find the optimal parameters

```
# parameters
tree_result[iter_cnt,1] <- min_criterion[i]
tree_result[iter_cnt,2] <- min_split[j]
tree_result[iter_cnt,3] <- max_depth[k]
tree_result[iter_cnt, 4:9] <- perf_eval(tmp_tree_val_cm)

# Number of leaf nodes
tree_result[iter_cnt,10] = length(nodes(tmp_tree, unique(where(tmp_tree))))
iter_cnt = iter_cnt + 1
```

✓ Store the parameters and corresponding performance values

R Exercise: Parameter Tuning

- Find the optimal parameters

```
# Find the best set of parameters
tree_result <- tree_result[order(tree_result[,9], decreasing = T),]
tree_result

best_criterion <- tree_result[1,1]
best_split <- tree_result[1,2]
best_depth <- tree_result[1,3]
```

✓ Sort the performance matrix in terms of AUROC

✓ Find the best parameter values

R Exercise: Parameter Tuning

- Result

	min_criterion	min_split	max_depth	TPR	Precision	TNR	ACC	BCR	F1	N_leaves
[1,]	0.90	10	0	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[2,]	0.90	10	10	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[3,]	0.90	10	5	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[4,]	0.90	30	0	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[5,]	0.90	30	10	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[6,]	0.90	30	5	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[7,]	0.95	10	0	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[8,]	0.95	10	10	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[9,]	0.95	10	5	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[10,]	0.95	30	0	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[11,]	0.95	30	10	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[12,]	0.95	30	5	0.9090909	0.9523810	0.9956140	0.9880000	0.9513694	0.9302326	9
[13,]	0.99	10	0	0.9090909	0.9230769	0.9926901	0.9853333	0.9499713	0.9160305	7
[14,]	0.99	10	10	0.9090909	0.9230769	0.9926901	0.9853333	0.9499713	0.9160305	7
[15,]	0.99	10	5	0.9090909	0.9230769	0.9926901	0.9853333	0.9499713	0.9160305	7
[16,]	0.99	30	0	0.9090909	0.9230769	0.9926901	0.9853333	0.9499713	0.9160305	7
[17,]	0.99	30	10	0.9090909	0.9230769	0.9926901	0.9853333	0.9499713	0.9160305	7
[18,]	0.99	30	5	0.9090909	0.9230769	0.9926901	0.9853333	0.9499713	0.9160305	7
[19,]	0.90	50	0	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[20,]	0.90	50	10	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[21,]	0.90	50	5	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[22,]	0.90	100	0	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[23,]	0.90	100	10	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[24,]	0.90	100	5	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[25,]	0.95	50	0	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[26,]	0.95	50	10	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[27,]	0.95	50	5	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[28,]	0.95	100	0	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[29,]	0.95	100	10	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[30,]	0.95	100	5	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	7
[31,]	0.99	50	0	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	6
[32,]	0.99	50	10	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	6
[33,]	0.99	50	5	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	6
[34,]	0.99	100	0	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	6
[35,]	0.99	100	10	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	6
[36,]	0.99	100	5	0.8333333	0.9482759	0.9956140	0.9813333	0.9108668	0.8870968	6

R Exercise: Training the Tree with the Best Parameters

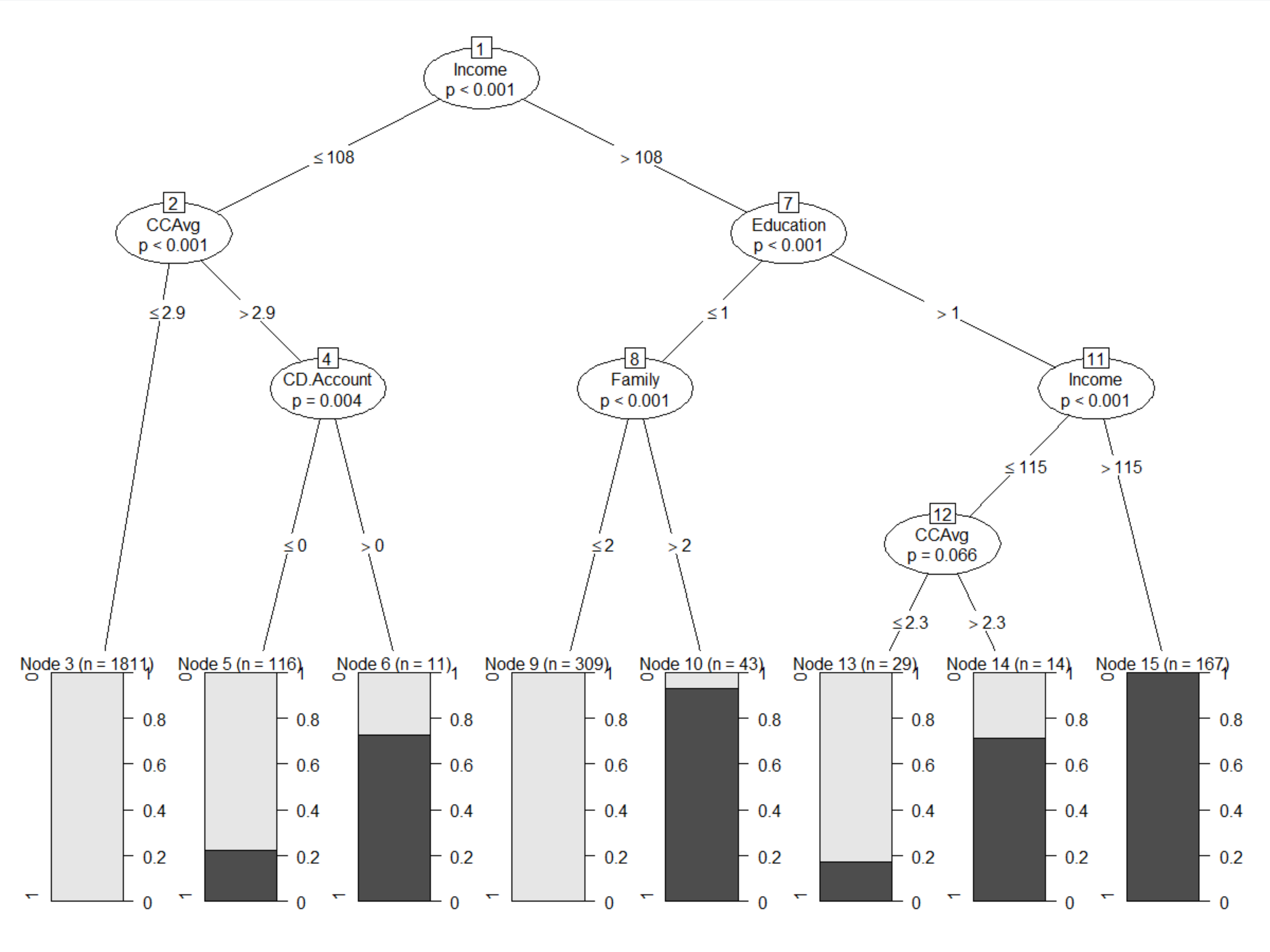
- Training the tree with the best parameters

```
# Construct the best tree
tree_control = ctree_control(mincriterion = best_criterion,
                            minsplit = best_split, maxdepth = best_depth)
tree <- ctree(ploanYN ~ ., data = ploan.all, controls = tree_control)
tree_all_prediction <- predict(tree, newdata = ploan.all)

# Performance of the best tree
tree_all_cm <- table(ploan.all$ploanYN, tree_all_prediction)

# Initialize the performance matrix
best_result <- matrix(0,1,7)
colnames(best_result) <- c("TPR", "Precision", "TNR", "ACC", "BCR", "F1", "N_leaves")

# Evaluate the performance
best_result[1,1:6] = perf_eval(tree_all_cm)

# Number of leaf nodes
best_result[1,7] = length(nodes(tree, unique(where(tree))))
best_result
```

```
> best_result
           TPR Precision       TNR    ACC       BCR        F1 N_leaves
[1,] 0.8789062 0.9574468 0.9955437 0.9836 0.9354088 0.9164969        8
```

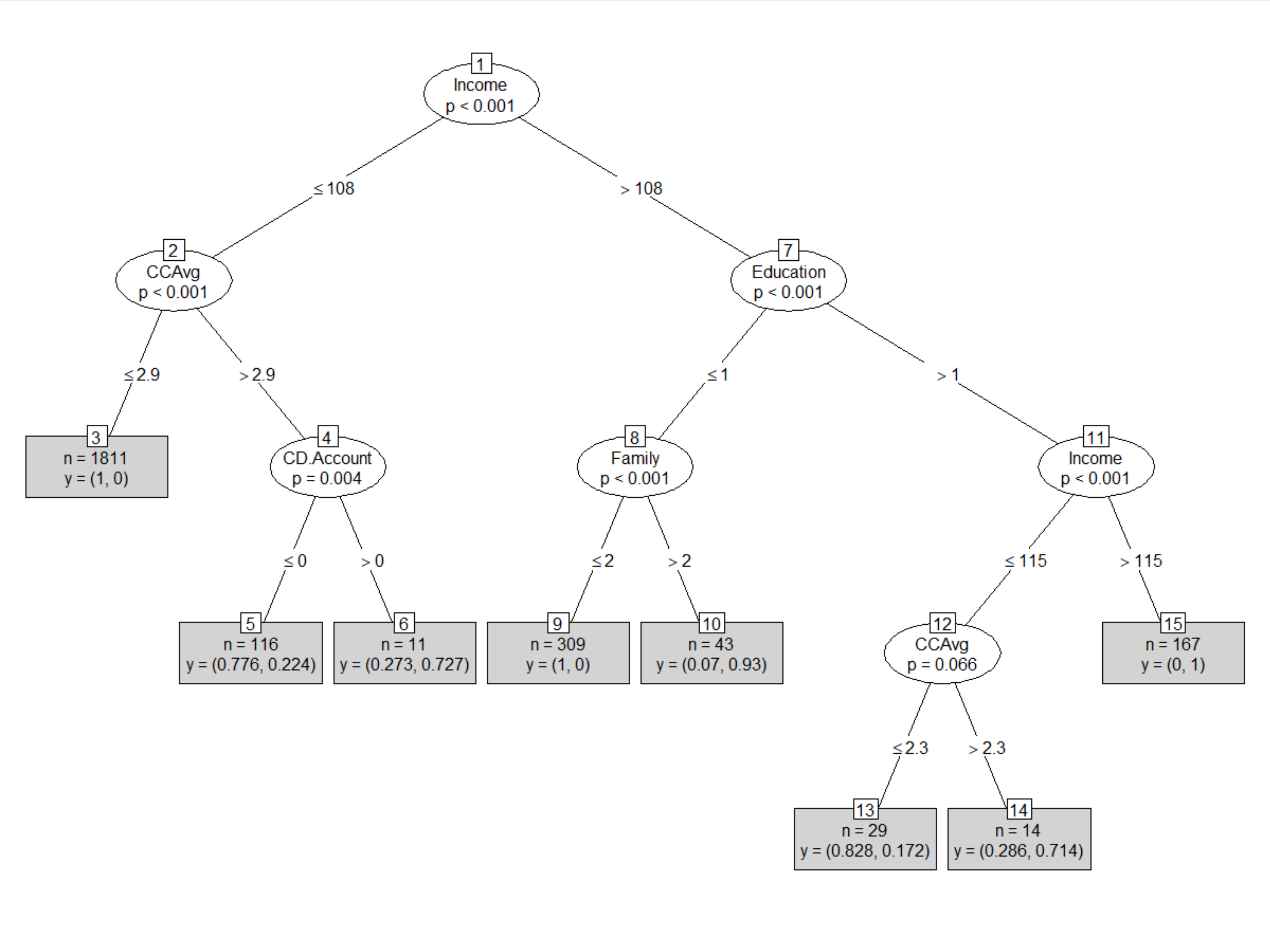
R Exercise: Parameter Tuning

- Tree plot



R Exercise

- Tree plots




ANY
questions?