classification and Regression Trees

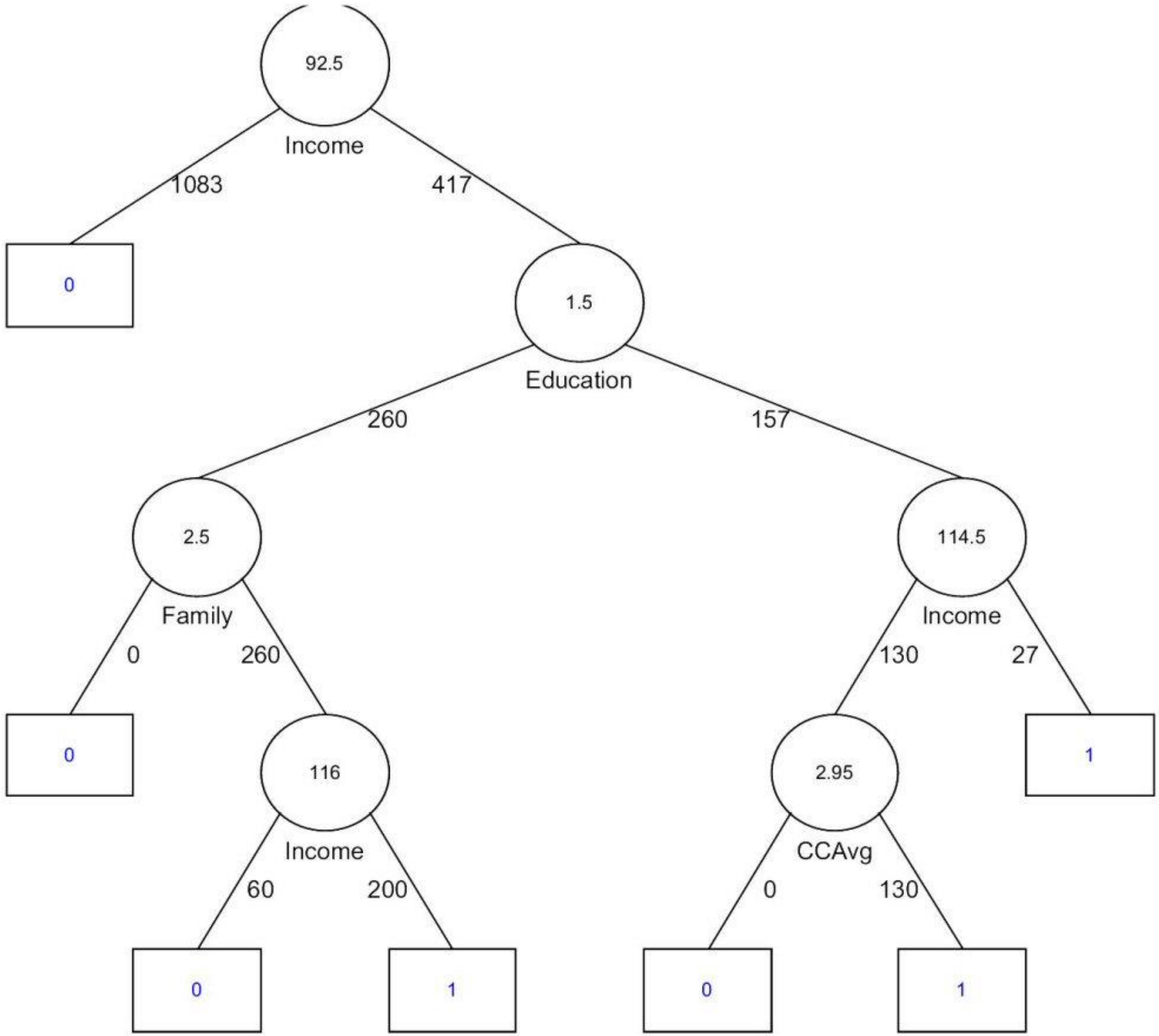
Trees and Rules

Goal: Classify or predict an outcome based on a set of predictors

The output is a set of **rules**

Example:

- Goal: classify a record as "will accept credit card offer" or "will not accept"
- Rule might be "IF (Income > 92.5) AND (Education < 1.5) AND (Family <= 2.5) THEN Class = 0 (nonacceptor)
- Also called CART, Decision Trees, or just Trees
- Rules are represented by tree diagrams


92.5
Income
1083
417
0
1.5
Education
260
157
2.5
Family
114.5
Income
0
260
130
27
0
116
Income
2.95
CCAvg
1
60
200
0
130
0
1
0
1

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 overfitting

Recursive Partitioning

Recursive Partitioning Steps

- Pick one of the predictor variables, x_i
- Pick a value of $x_{i,}$ say s_i, that divides the training data into two (not necessarily equal) portions
- Measure how “pure” or homogeneous each of the resulting portions are

  “Pure” = containing records of mostly one class
- Algorithm tries different values of $x_{i,}$ and s_i to maximize purity in initial split
- After you get a “maximum purity” split, repeat the process for a second split, and so on

Example: Riding Mowers

- Goal: Classify 24 households as owning or not owning riding mowers
- Predictors = Income, Lot Size

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

How to split

- Order records according to one variable, say lot size
- Find midpoints between successive values

  E.g. first midpoint is 14.4 (halfway between 14.0 and 14.8)
- Divide records into those with lotsize > 14.4 and those < 14.4
- After evaluating that split, try the next one, which is 15.4 (halfway between 14.8 and 16.0)

Note: Categorical Variables

- Examine all possible ways in which the categories can be split.

- E.g., categories A, B, C can be split 3 ways

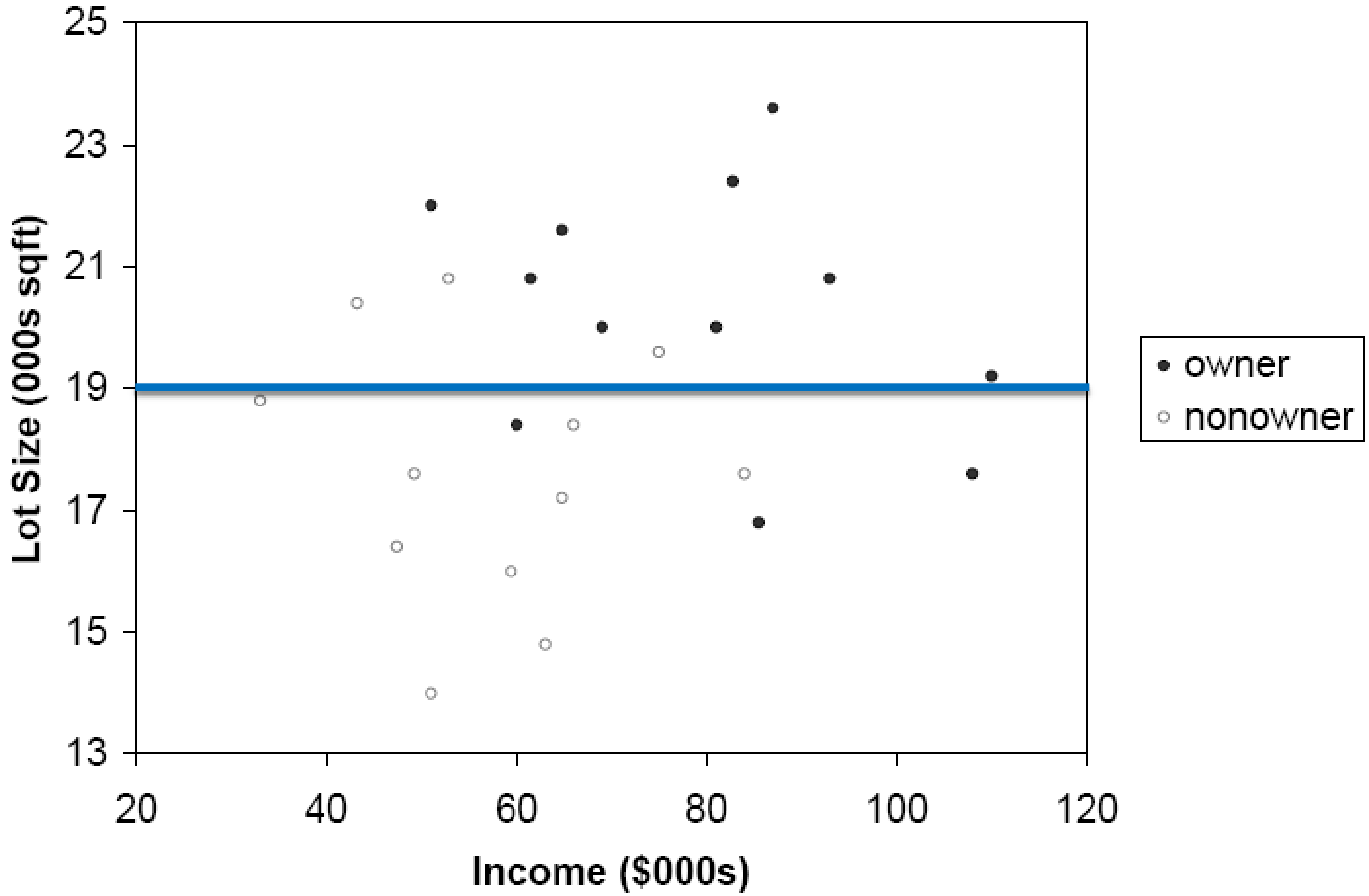
  {A} and {B, C}

  {B} and {A, C}

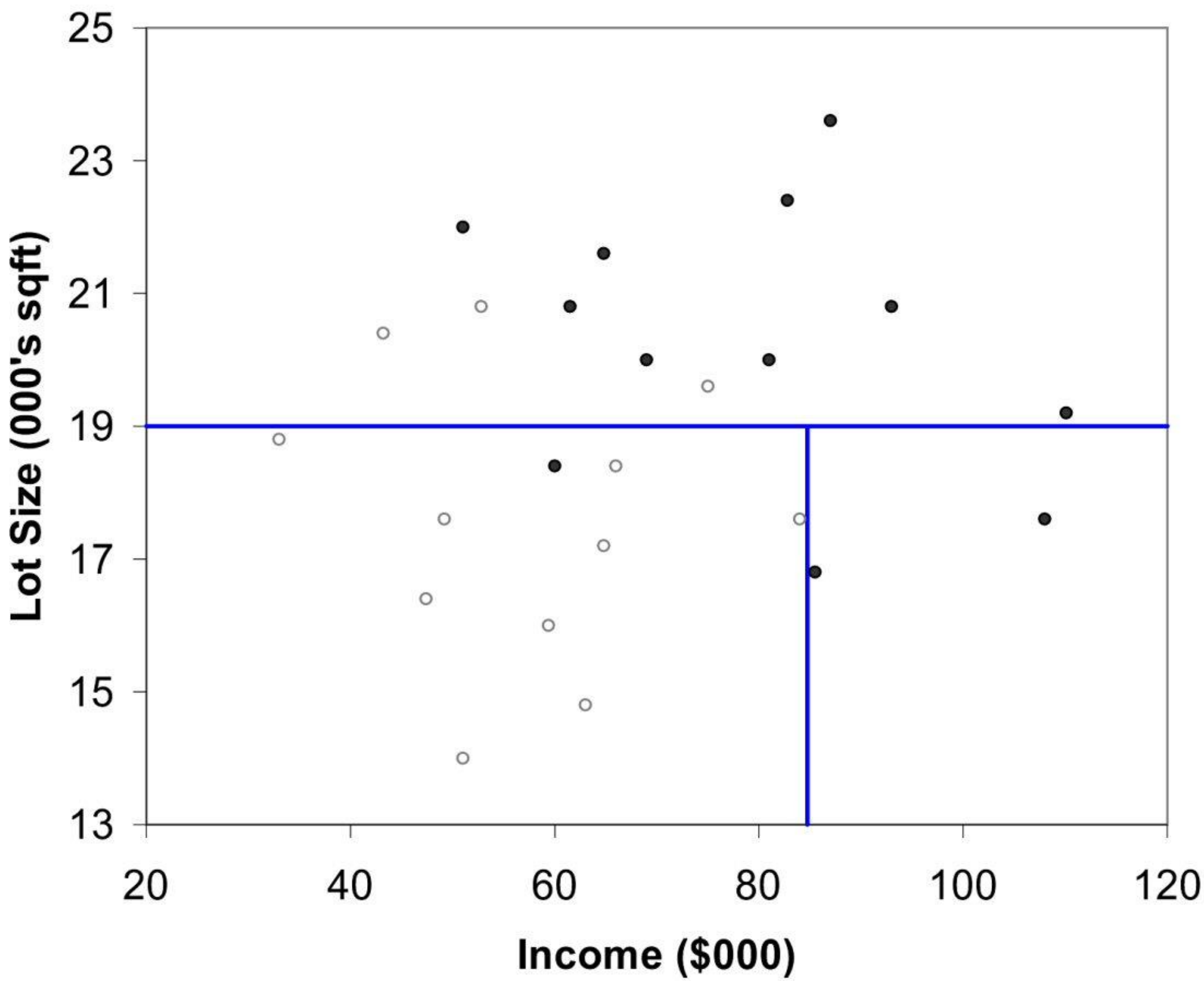
  {C} and {A, B}

- With many categories, # of splits becomes huge
- XLMiner supports only binary categorical variables

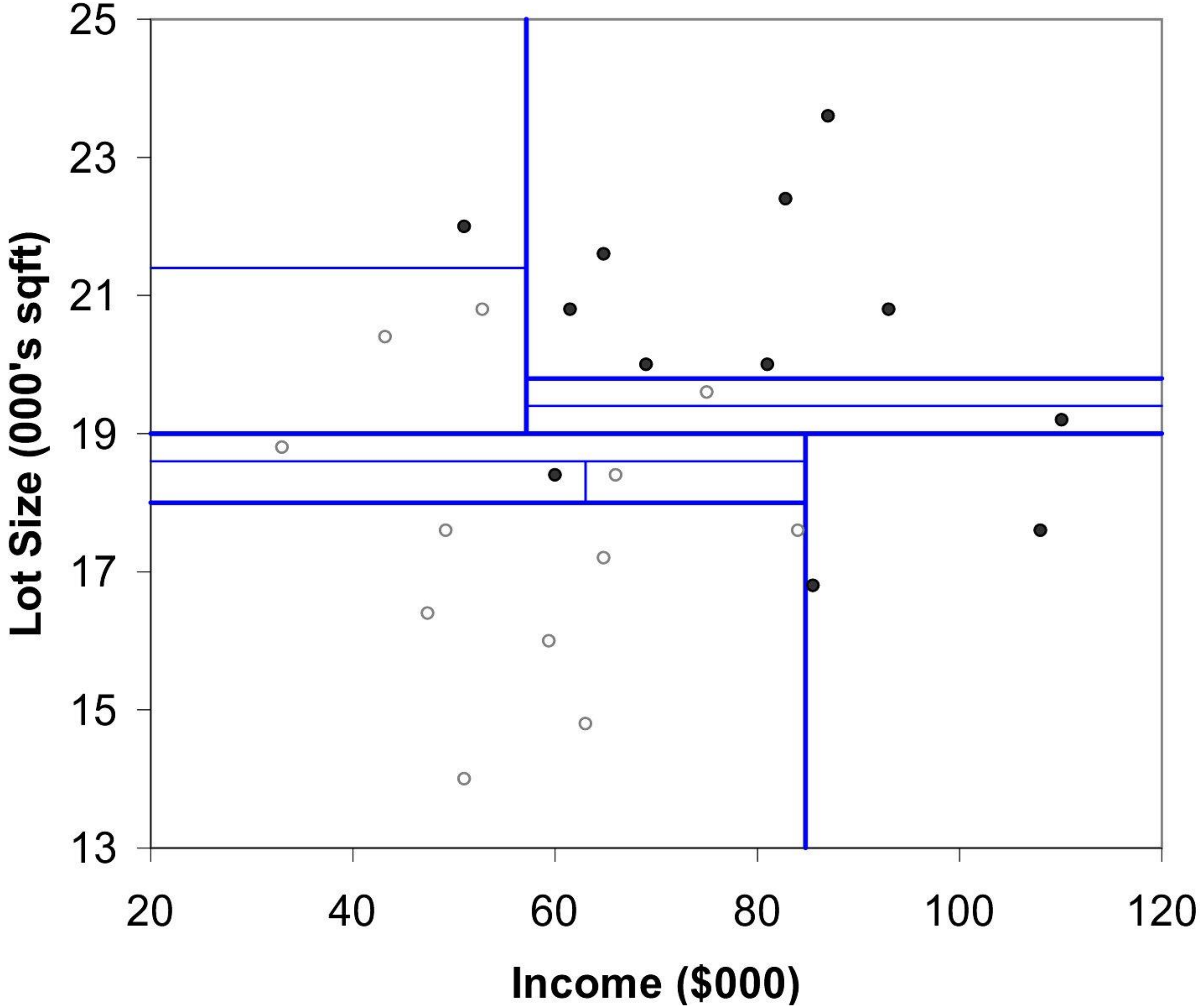
The first split: Lot Size = 19,000



Second Split: Income = $84,000



After All Splits



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) = 0 when all cases belong to same class
- Max value when all classes are equally represented (= 0.50 in binary case)

Note: XLMiner uses a variant called “delta splitting rule”

Entropy

$$entropy\,(A) = -\sum_{k=1}^{m} p_k \, log_2(p_k)$$

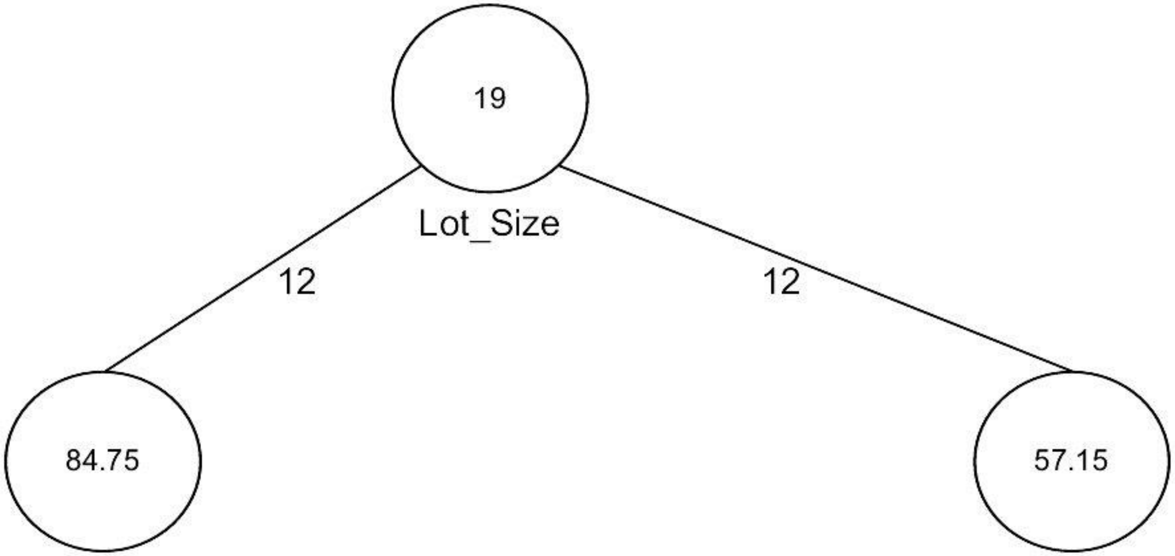
p = proportion of cases (out of *m*) in rectangle *A* that belong to class *k*

- Entropy ranges between 0 (most pure) and $\log_2(m)$ (equal representation of classes)

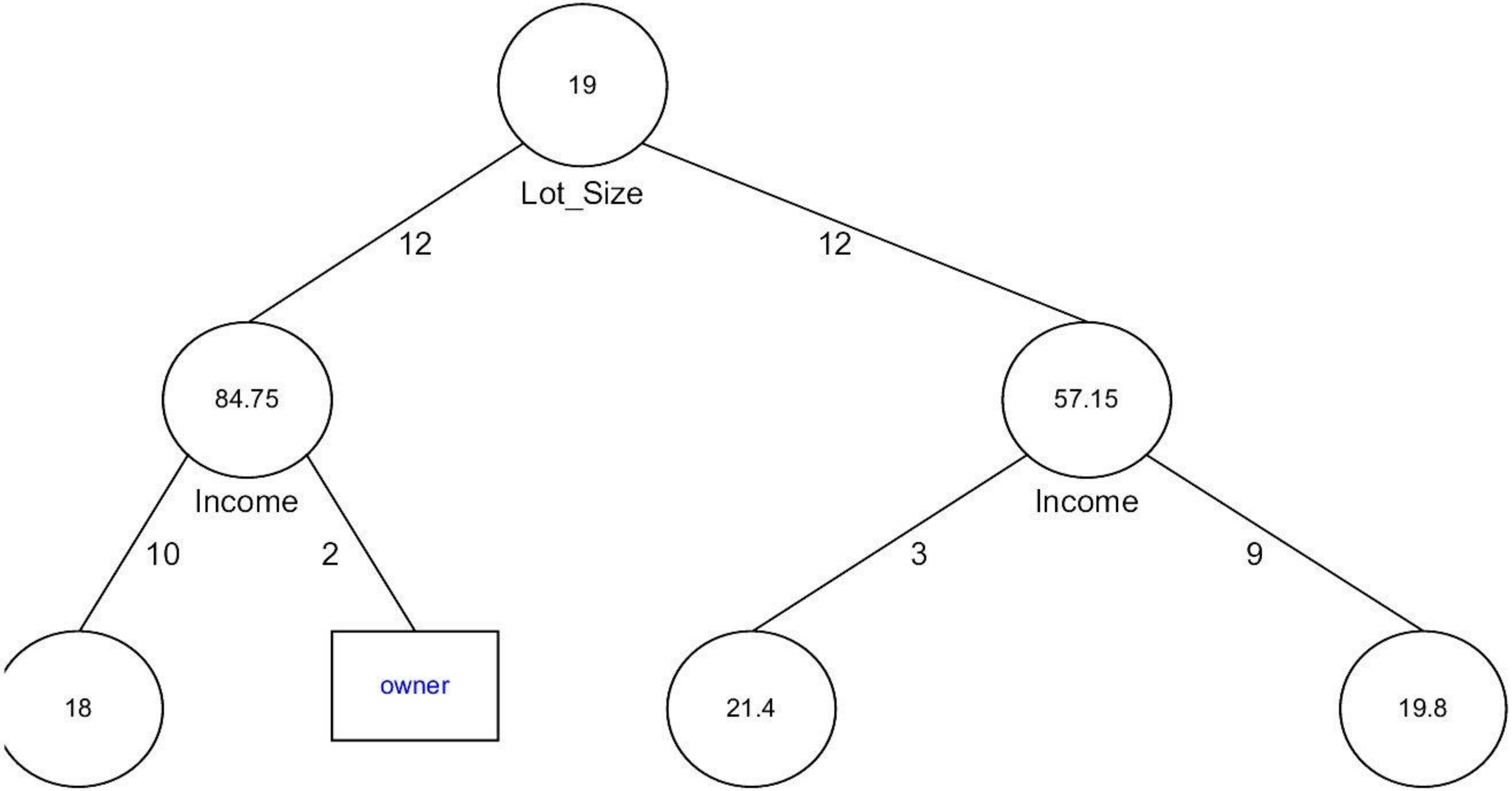
Impurity and Recursive Partitioning

- Obtain overall impurity measure (weighted avg. of individual rectangles)
- At each successive stage, compare this measure across all possible splits in all variables
- Choose the split that reduces impurity the most
- Chosen split points become nodes on the tree

First Split – The Tree



Tree after three splits



Tree Structure

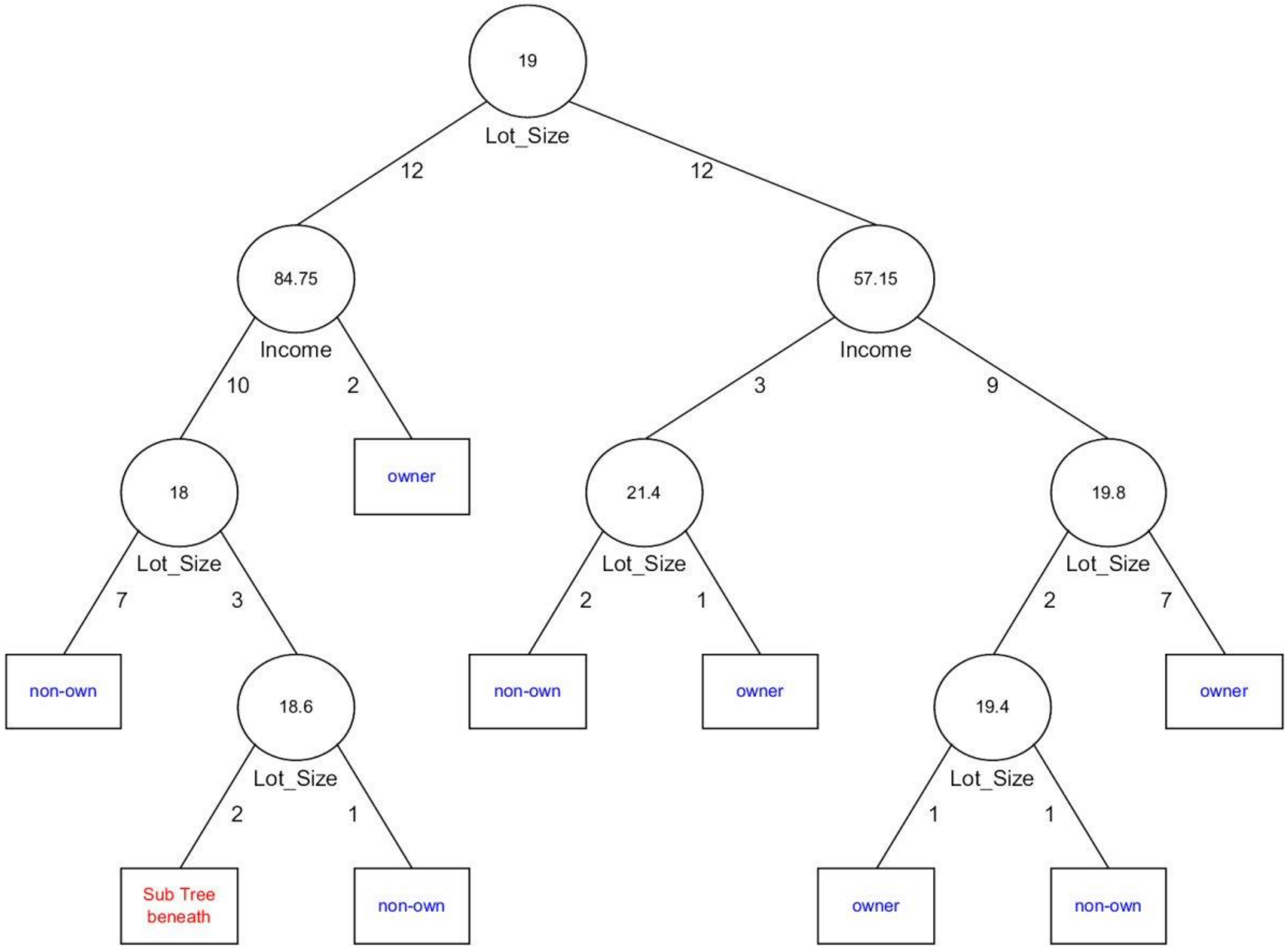
- Split points become nodes on tree (circles with split value in center)
- Rectangles represent “leaves” (terminal points, no further splits, classification value noted)
- Numbers on lines between nodes indicate # cases
- Read down tree to derive rule

  E.g., If lot size < 19, and if income > 84.75, then class = “owner”

Determining Leaf Node Label

- 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% or more “1” records in the leaf to label it a “1” node

Tree after all splits

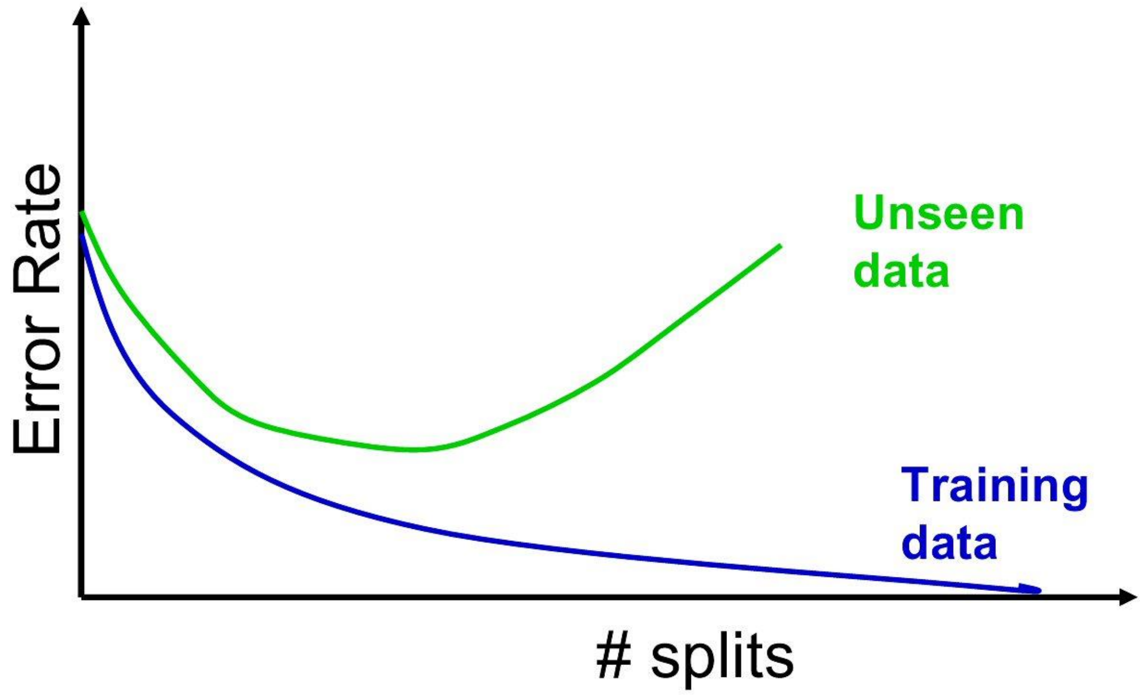


The Overfitting Problem

Stopping Tree Growth

- Natural end of process is 100% purity in each leaf
- This **overfits** the data, which end up fitting noise in the data
- Overfitting leads to low predictive accuracy of new data
- Past a certain point, the error rate for the validation data starts to increase

Full Tree Error Rate



CHAID

CHAID, older than CART, uses chi-square statistical test to limit tree growth

Splitting stops when purity improvement is not statistically significant

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

Cost Complexity

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

$CC(T)$ = cost complexity of a tree

$Err(T)$ = proportion of misclassified records

α = penalty factor attached to tree size (set by user)

- Among trees of given size, choose the one with lowest CC
- Do this for each size of tree

Using Validation Error to Prune

Pruning process yields a set of trees of different sizes and associated error rates

Two trees of interest:

- Minimum error tree
  - Has lowest error rate on validation data
- Best pruned tree
  - Smallest tree within one std. error of min. error
  - This adds a bonus for simplicity/parsimony

Error rates on pruned trees

# Decision Nodes	% Error Training	% Error Validation	
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	
14	[illegible]	[illegible]	
13	1.16	1.6	
12	1.2	1.6	
11	1.2	1.466667	<-- Min. Err. Tree
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	<-- Best Pruned Tree
5	4.44	1.8	
4	5.08	2.333333	
3	5.24	3.466667	

Regression Trees

Regression Trees for Prediction

- Used with continuous outcome variable
- Procedure similar to classification tree
- Many splits attempted, choose the one that minimizes impurity

Differences from CT

- Prediction is computed as the **average** of numerical target variable in the rectangle (in CT it is majority vote)

- Impurity measured by **sum of squared deviations** from leaf mean

- Performance measured by RMSE (root mean squared error)

Advantages of trees

- Easy to use, 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 splits

- Since the process deals with one variable at a time, no way to capture interactions between variables

Summary

- Classification and Regression Trees are an easily understandable and transparent method for predicting or classifying new records

- A tree is a graphical representation of a set of rules

- Trees must be pruned to avoid over-fitting of the training data

- As trees do not make any assumptions about the data structure, they usually require large samples